Machine Learning Cookbook with Python

Create ML and Data Analytics Projects Using Some Amazing Open Datasets

REHAN GUHA

Machine Learning Cookbook with Python

Create ML and Data Analytics Projects Using Some Amazing Open Datasets

Rehan Guha

www.bpbonline.com

FIRST EDITION 2021

Copyright © BPB Publications, India

ISBN: 978-93-89898-00-2

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Distributors:

BPB PUBLICATIONS
20, Ansari Road, Darya Ganj
New Delhi-110002
Ph: 23254990/23254991

DECCAN AGENCIES
4-3-329, Bank Street,
Hyderabad-500195
Ph: 24756967/24756400

MICRO MEDIA
Shop No. 5, Mahendra Chambers,
150 DN Rd. Next to Capital Cinema,
V.T. (C.S.T.) Station, MUMBAI-400 001
Ph: 22078296/22078297

BPB BOOK CENTRE
376 Old Lajpat Rai Market,
Delhi-110006
Ph: 23861747

Published by Manish Jain for BPB Publications, 20 Ansari Road, Darya Ganj,
New Delhi-110002 and Printed by him at Repro India Ltd, Mumbai

www.bpbonline.com

Dedicated to

Ranjan Guha

A Friend, Philosopher, and Guide

The one who gave me unparalleled wisdom about "Dialectical Logic", " To Question Everything", "Everything is Conditional & Relative", and whatnot.

*To top everything, he happens to be my **Father***

Nandita Guha

*My Mom is so much more than just my **Mother***

"Empathy", "Dedication", and "Perseverance" are some of the most beautiful things I learned from her.

The only person in my world who is most concerned about "What did I eat today?" and at the same time, "What shade of Lipstick should I buy?"

LOVE YOU BOTH "EQUALLY"

About the Author

Rehan Guha — A Researcher by the day and an Artist by night.

Our Author is a Scholar-lecturer, an Innovator, and also a Humanitarian-Philanthropist.

He started his life as a Coder, Developer, and now he is into research in the field of Machine Learning and Algorithms. He also has a keen interest in General Science, Technology, Invention, and Innovation.

The author holds a graduate degree from the Institute of Engineering & Management, Kolkata, and a Postgraduate certification on Deep Learning from the Indian Institute of Technology, Kharagpur (IIT-K, AICTE-approved FDP course).

If we talk about Rehan's area of interest, it lies in Optimization Problems, Explainable AI, Deep Learning Architecture, Algorithms, Complexity, Algorithmic Thinking, etc. He has multiple publications in Journals and Open Publications. He has also filed multiple patents for his Innovations and Inventions. One of his early patents was also demonstrated to the Indian Army.

During the span of his career, Rehan has been involved with a variety of Business Verticals, starting from Banking, Medicine, Law, Insurance, Freight and Logistics. He has also provided a patent filing solution for Dispute Banking. He has experience of about three years in IT, where he has worked on a multitude of projects within this small tenure. Rehan has been working on various projects as a Developer, Data Scientist, Machine Learning Engineer, and currently as a Machine Learning Researcher.

His work across multiple disciplines broadly addresses his sheer passion for learning and application of the same. Apart from his professional life and work, he is a philanthropist who believes in knowledge sharing and strives for the concept of "Open/Free Knowledge". He conducts multiple workshops and seminars in his field of research and works for various universities, encouraging students to pursue their passion. Along with this, he has several Open Source Software contributions and is a hobbyist bug bounty hunter, and he has resolved critical bugs for companies like Zomato and others.

The author is an active supporting member of the United Nations World Food Program (WFP) and supports Educational Equality. He volunteers as an Educator and a Personal Development Coach for the needful.

With an avid interest in photography, it will be best to say that “He loves to capture the moment!”. In his leisure time, he is a non-professional Table Tennis and Football player. He is also he is a travel enthusiast and loves to explore People and Culture.

Rehan has a keen interest in one of his earliest hobbies — Art. To be precise, Pencil Sketching & sometimes Ink Sketching. One of his wood engraving sketch artwork under the program “Rivers of the World” can be found in the archives of the British Council Museum, London. This work of his has travelled to parts of Europe and Asia as a part of the British Council Art Exhibition.

Ways to Connect with the Author:

Portfolio & Blog	:: https://rehanguha.github.io/
LinkedIn	:: https://www.linkedin.com/in/rehanguha/
GitHub	:: https://github.com/rehanguha
(*Photography*) Flickr	:: https://flickr.com/rehanguha

Acknowledgement

Writing a book is harder than I thought and more rewarding than I could have ever imagined. None of this would have been possible without my parents. The ones who stood by me during every struggle and all my successes. That is what I call a “True Friendship”.

Having an idea and turning it into a book is as hard as it sounds. The experience is both internally challenging and rewarding. I have to start by thanking a few people who took out a huge chunk of their precious time for my work. First and foremost, Shreya Halder a.k.a. “Bhodor” — From reading early drafts to giving me advice on the understandability, Shreya was there throughout the entire journey. On the other hand, Sachin Kumar a.k.a. “The Trainer” — From reviewing all the mathematics to keeping a check on me so that I do not complicate things and maintain an understandability throughout the book. Along with them, I want to mention some of my closest friends who were an integral part of it from different aspects — Harish K. Raman a.k.a “The Lawyer” and Smrita, a.k.a. “Smita Chowdhury”.

Finally, to all those who have been a part of me getting there: Vekila a.k.a. “X”, Arpita Banerjee, Sauritra, Riya Jaiswal, Simran, Sayantani Biswas, Nikita, Pallabi Changkakoty a.k.a. “Pulu”, Somnath Ghosh and Inderjeet Kaur a.k.a. “Eno” (RIP).

I will take this opportunity to especially thank Bappa Kar, Abhishek Sinha, and Srikumar Subramanian for teaching me valuable lessons throughout different stages of my life. I also wish to express acknowledgement towards Pramati Technologies Pvt. Ltd., and all my gratitude goes to the colleagues I work with.

Finally, I would like to thank BPB Publications for giving me this opportunity to write my first book for them.

Preface

The preface is defined by "an introduction to a book, typically stating its subject, scope, or aims."

I will try to answer what a preface should contain with 5 W's over here, which will mostly cover all the aspects and the areas which I want to touch.

Why am I writing this book?

When I thought of getting into Machine Learning, I faced a few hurdles. One of the major hurdles was how to get started with Machine Learning. I referred to a lot of online courses, videos, and books, but in some of the places, the material is extremely math-centric, and others were only practical with no mathematical explanations. Another major hurdle was the reference material, which was meant for beginners. It hardly contained any mathematics, and the explanations were too simple to dive further into that topic. Similarly, some materials were overly complicated, where someone might get lost easily as there is a lot of information.

What motivated me to write this book?

Being a machine learning enthusiast, when I was trying to jump into and take Machine Learning as my career, I did not find any suitable books or videos that will help someone to onboard into the field of machine learning with ease as I had mentioned earlier. The things which I didn't get in a particular material are the right quantity of maths and coding with practical projects and an explanation of basics. This exact point motivated me to write a well-balanced book that can be utilized and enjoyed by a varied type of readers.

Who are the readers of the book?

This book is not a traditional book you would find in an online or offline marketplace. It is a project book, and has all the basic as well as some advanced concepts of machine learning introduced and explained using sufficient but not overwhelming mathematical concepts.

Keeping all the things in mind, this book can be beneficial for multiple reader groups like machine learning enthusiasts, professionals, and students who want to get the flavor of machine learning in reality. Apart from the shared benefits of the book, one can also use this particular book as a Reference Guide or to know different machine learning practitioners' ways of solving a problem.

What will you find in the book?

The primary goal of this type of book is to provide the right skills and the necessary background to build a machine learning model and Predict what? Along with building a machine learning model, one will also learn the importance of data pre-processing, model evaluation, and model retuning to get better accuracy from the output.

This book contains real-life examples that will give everyone the experience to handle the entire pipeline for a machine learning project. Over the five chapters in the book, you will learn the following:

- **Chapter 1:** It gives an introduction on how to handle data, basics of O.S.E.M.N. Framework, and an explanation of its components.
- **Chapter 2:** This chapter delves into the techniques used for data exploring, and analyzing and crunching the data.
- **Chapter 3:** This chapter onwards, I have introduced Machine Learning concepts like Clustering, Regression, and Classification.
- **Chapter 4:** This chapter focuses on the Classification problem and inferring through a unique type dataset where we are not aware of the features.
- **Chapter 5:** It deals with Model retuning and improving model performance to achieve improved results.

What will be the impact after reading the book?

This book is written using the method of Incremental Writing, i.e., concepts are written down and assembled incrementally. If one reads the book from Inception to the very end, then the person will be capable of performing Data Analytics, obtaining various insights from the Data, Understand the Business problem and make a Predictive Model, and finally tune the Model to get optimal accuracy.

Downloading the code bundle and coloured images:

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/0wmmdmx

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at:

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

BPB is searching for authors like you

If you're interested in becoming an author for BPB, please visit **www.bpbonline.com** and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/Machine-Learning-Cookbook-with-Python**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at **https://github.com/bpbpublications**. Check them out!

PIRACY

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**.

REVIEWS

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Table of Contents

Chapter 1
Boston Crime

Introduction

Everyone has heard that "Data[1] is the new oil," and data is freely available everywhere, starting from newspapers, Twitter, etc. Just the crude oil, i.e., data, has no value by itself, so we will be using different techniques to make sense of the data and gain some information[2] out of it.

Let us start with the basics like O.S.E.M.N. Framework, E.D.A., and some visualization techniques. This chapter will mostly cover some basics of Data Exploring and how to implement some techniques for data cleaning as well.

This chapter is extremely important as it is the introduction to Machine Learning and Data Science. The readers will get the skill and the confidence to play with the data and draw various insights out of it.

Structure

- Types of data
- Let's talk about the Boston dataset
- O.S.E.M.N. framework

[1] "Data - Wikipedia." https://en.wikipedia.org/wiki/Data.

[2] "Information - Wikipedia." https://en.wikipedia.org/wiki/Information.

Objective

In this chapter, we will mostly look into the concept of data analysis and see some techniques to clean the data. At the end of the chapter, the reader should have the ability to process the data and have great insight into the data we are using.

What is Data?

As per the definition of data, it is the "facts and statistics collected together for reference or analysis." But we will define Data as a set of values of subjects concerning qualitative or quantitative variables. Data and information or knowledge are often used interchangeably; however, data becomes information when it is viewed in context or post analysis[3].

Types of Data

- **Structured data** - Structured data is generally stored in tabular form, and it can be stored in a relational database. It can be names, phone numbers, location, or other metrics like distance, loan amount, etc. and generally, we can query the relational table with SQL.
- **Semi-structured data** - Semi-structured data is similar to structured data, but it does not follow the conventional relational table structure. Files like XML, JSON, etc. are examples of semi-structured data.
- **Unstructured data** - As the name suggests, unstructured data follows no formal structure or relational table. E.g., texts, tweets from Twitter, Media (Audio-Video, etc.)

These are some of the building blocks of data types[4] which are used in machine learning.

For this chapter, we will use structured data taken from the Boston Police Department.

Let's talk about the Boston dataset

Boston crime dataset[5] is a collection of crime incident reports that are provided by the **Boston Police Department** (**BPD**). This dataset documents the initial details surrounding an incident to which BPD officers respond. This is a dataset containing

[3] "Data - Wikipedia." https://en.wikipedia.org/wiki/Data.

[4] "Data Types: Structured vs. Unstructured Data | Big Data Framework©." 22 Mar. 2019, https://www.bigdataframework.org/data-types-structured-vs-unstructured-data/.

[5] "Crime Incident Reports (August 2015 - To Date) (Source: New System ...." https://data.boston.gov/dataset/crime-incident-reports-august-2015-to-date-source-new-system.)

records from the new crime incident report system, which includes a reduced set of fields focused on capturing the type of incident as well as when and where it occurred. Records in the new system begin from in June of 2015.

So, the first thing we should do is to know the different features in the dataset.

Data dictionary

Any standard dataset will contain a data dictionary. As per definition, a **data dictionary** is "a set of information describing the contents, format, and structure of a database and the relationship between its elements, used to control access to and manipulation of the database." As mentioned before, we are using structured data, so this can be stored in a relational database. *Table 1.1* shows the data dictionary with all the details.

Field Name, Data Type, Required	Description
[incident_num] [varchar] (20) NOT NULL,	Internal BPD report number
[offense_code] [varchar] (25) NULL,	Numerical code of offense description
[Offense_Code_Group_Description] [varchar] (80) NULL,	Internal categorization of [offense_description]
[Offense_Description] [varchar] (80) NULL,	The primary descriptor of the incident
[district] [varchar] (10) NULL,	What district the crime was reported in
[reporting_area] [varchar](10) NULL,	RA number associated with the location where the crime was reported from.
[shooting][char] (1) NULL,	Indicated, a shooting took place.
[occurred_on] [datetime2](7) NULL,	Earliest date and time the incident could have taken place
[UCR_Part] [varchar](25) NULL,	Universal Crime Reporting Part number (1, 2, 3)
[street] [varchar](50) NULL,	Street name the incident took place

Table 1.1: *Data Dictionary*

For any dataset, we need to know about the data and analyze its each feature and its contribution.

The best way to know more about the data is to get your hands dirty.

Let's start with Python and Jupyter Notebook to explore the dataset.

First, we need to set up the machine and install the required packages to get things going. Please refer to the GitHub link:

https://github.com/bpbpublications/Machine-Learning-Cookbook-with-Python.

The entire code can be compiled and executed online, just with a web browser using Binder. Please use the above GitHub link to find more details about it.

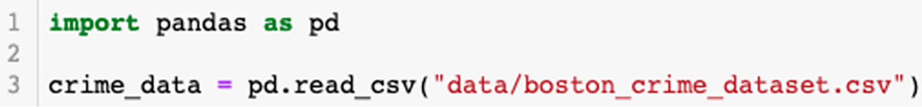
O.S.E.M.N. framework

All Machine Learning Projects and Data Science Projects have a basic framework named **O.S.E.M.N.** (**O**btaining, **S**crubbing, **E**xploring, **M**odeling, **IN**terpreting)[6], and we can see with framework Data Fetching, Data Cleaning, and Data Exploring takes up 60% of the pipeline.

What is Data Obtaining?

In this chapter, we are using data from the Boston Police Department repository, and downloading from it is considered as Data Obtaining/Fetching. There can be cases where we need to scrap[7] the data from websites, media files, log files, etc.All the steps required to gather the data are considered Data Obtaining/Fetching.

After downloading the Data from the given URL, we need to load the dataset using pandas and store it in the DataFrame (*Figure 1.1*) to start cleaning the data for Exploring.



```python
import pandas as pd

crime_data = pd.read_csv("data/boston_crime_dataset.csv")
```

Figure 1.1: *Data loading*

We are using a DataFrame from pandas to store the Boston Crime Dataset. Pandas is a popular library used for Machine Learning and Data Science.

What is Data Scrubbing?

As the name suggests, Data Scrubbing[8] is a process of cleaning the data which will be fit for use in the next stage that is Data Exploration and Analysis. Data Scrubbing, also known as Data Cleaning, takes up the maximum time during the process of

[6] "dataists » A Taxonomy of Data Science." 25 Sep. 2010, http://www.dataists.com/2010/09/a-taxonomy-of-data-science/.

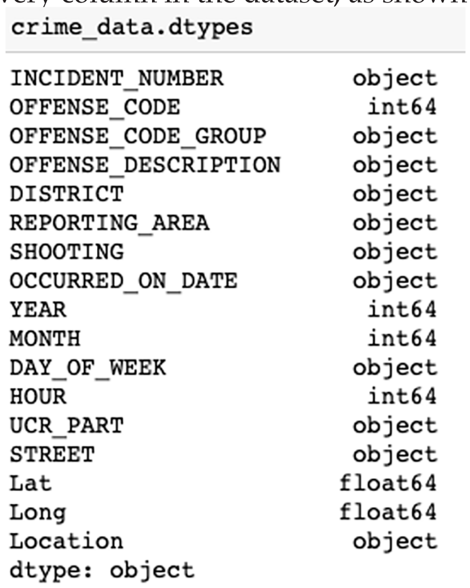
[7] "Data scraping - Wikipedia." https://en.wikipedia.org/wiki/Data_scraping.

[8] "Data cleansing - Wikipedia." https://en.wikipedia.org/wiki/Data_cleansing.

Data Analysis. During this phase, we will mostly focus on handling incorrect data, missing values, and errors related to the data structures.

Finding Data Types

Depending on the data type, different data cleaning techniques can be applied and it's not just cleaning; we need to scrub the data logically to reduce ambiguity. Let's find the data type for every column in the dataset, as shown in *Figure 1.2*.



```
crime_data.dtypes
```

```
INCIDENT_NUMBER        object
OFFENSE_CODE            int64
OFFENSE_CODE_GROUP     object
OFFENSE_DESCRIPTION    object
DISTRICT               object
REPORTING_AREA         object
SHOOTING               object
OCCURRED_ON_DATE       object
YEAR                    int64
MONTH                   int64
DAY_OF_WEEK            object
HOUR                    int64
UCR_PART               object
STREET                 object
Lat                   float64
Long                  float64
Location               object
dtype: object
```

Figure 1.2: *Data types of all the features*

As you can see there are multiple types of data types, and this data is a non-homogeneous dataset.

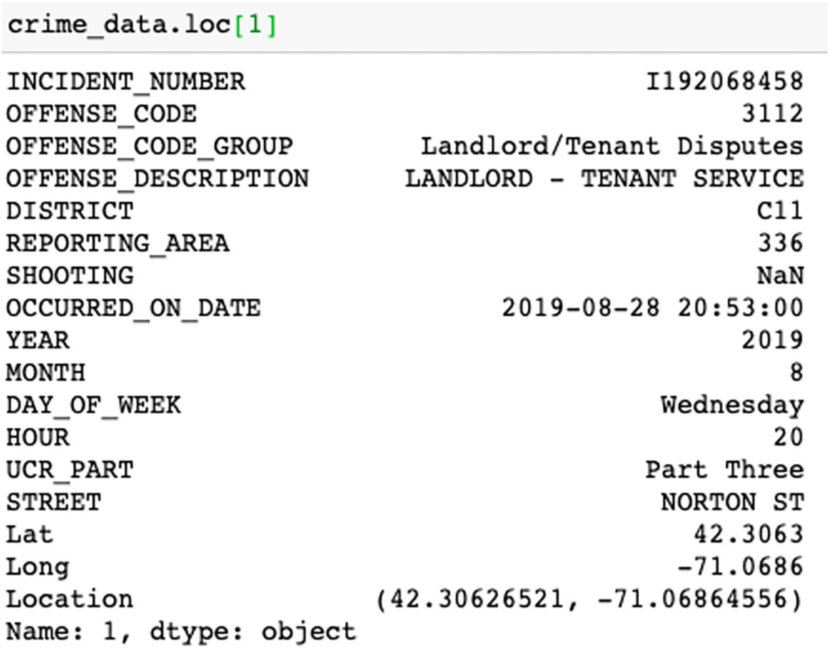
The different data types[9] in the Boston Crime dataset are as follows:

1. int64
2. float64
3. object

Referring to the Data Dictionary of the dataset, we can see how the features are correlated to the real world.

In *Figure 1.3,* we can see all the data and how it looks and why it is of the given data type.

[9] "Essential basic functionality — pandas 0.25.1 documentation." https://pandas.pydata.org/pandas-docs/stable/basics.html#dtypes.



```
crime_data.loc[1]
```

```
INCIDENT_NUMBER                         I192068458
OFFENSE_CODE                                  3112
OFFENSE_CODE_GROUP        Landlord/Tenant Disputes
OFFENSE_DESCRIPTION      LANDLORD - TENANT SERVICE
DISTRICT                                       C11
REPORTING_AREA                                 336
SHOOTING                                       NaN
OCCURRED_ON_DATE               2019-08-28 20:53:00
YEAR                                          2019
MONTH                                            8
DAY_OF_WEEK                              Wednesday
HOUR                                            20
UCR_PART                                Part Three
STREET                                   NORTON ST
Lat                                        42.3063
Long                                      -71.0686
Location               (42.30626521, -71.06864556)
Name: 1, dtype: object
```

Figure 1.3: *Sample data*

Comparing the data with *Figure 1.2*, we can see the types and the structure of the data. Looking at the data, we get information about how to clean the data.

How to Handle Missing Data?

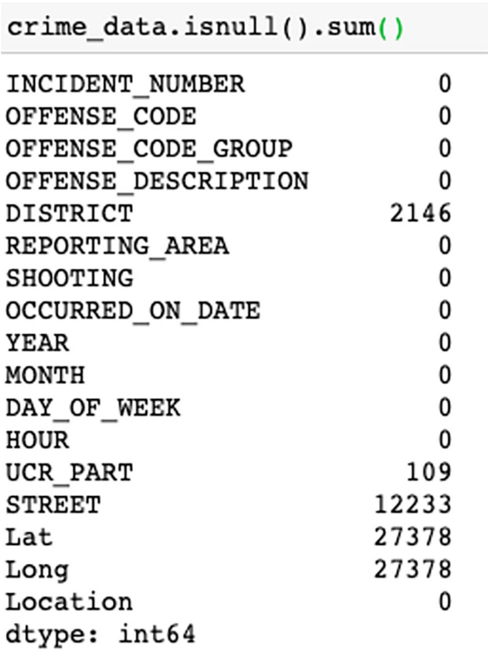
Handling missing values from the data will improve the quality of the data we are using, and in turn, it will yield accurate analysis.

To handle missing data, we generally ask questions as given below:

1. Are there any missing values?
2. Are the missing values significant enough to handle it?

We will answer the above set of questions in due time.

So, let us first see all the missing values of all the features of the Boston Crime Dataset (*Figure 1.4*).

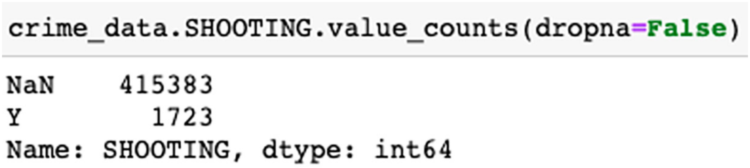


```
crime_data.isnull().sum()
```

```
INCIDENT_NUMBER             0
OFFENSE_CODE                0
OFFENSE_CODE_GROUP          0
OFFENSE_DESCRIPTION         0
DISTRICT                 2146
REPORTING_AREA              0
SHOOTING                    0
OCCURRED_ON_DATE            0
YEAR                        0
MONTH                       0
DAY_OF_WEEK                 0
HOUR                        0
UCR_PART                  109
STREET                  12233
Lat                     27378
Long                    27378
Location                    0
dtype: int64
```

Figure 1.4: *Missing value count for all columns*

Let us consider the variables "`SHOOTING`" and "`STREET`" and find the count of missing values.



```
crime_data.SHOOTING.value_counts(dropna=False)
```

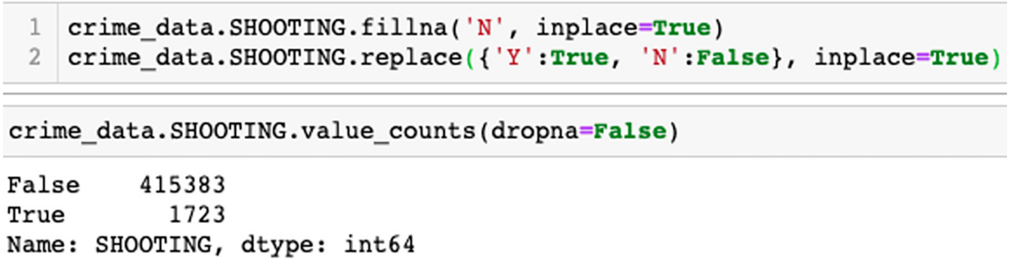
```
NaN    415383
Y        1723
Name: SHOOTING, dtype: int64
```

Figure 1.5: *Frequency table for "SHOOTING"*

As we can see, the feature "`SHOOTING`" is a Boolean field (*Figure 1.5*) from the details of the features and the data, so the values can be either True or False. Either shooting can take place, or it won't take place. As per the missing value report, we can see that around 1723 records have True/Yes rest all the records are missing. But the good thing about the Boolean feature is if something is missing, then it will be either True or False. In the case of "`SHOOTING`," the missing values will be No/False as we already have some value for Yes/True.

As we know how to tackle the "`SHOOTING`" feature we should proceed with it.

You can see in *Figure 1.6* that we have first replaced all the missing values with "N" as we concluded before that in this case of "NaN," it is the same as "N."



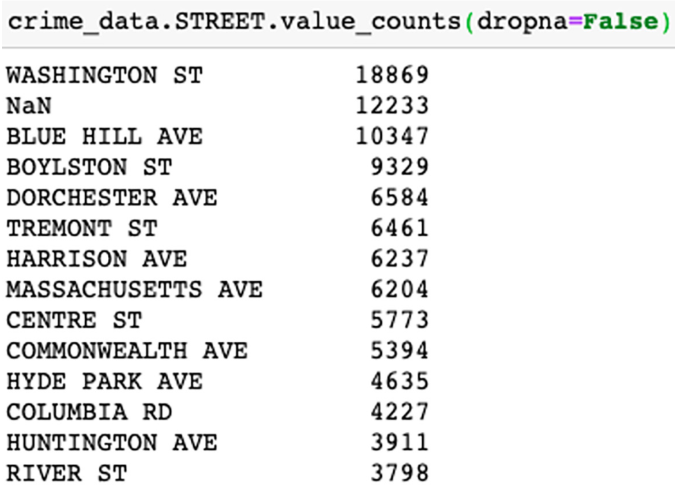
```
crime_data.SHOOTING.fillna('N', inplace=True)
crime_data.SHOOTING.replace({'Y':True, 'N':False}, inplace=True)
```

```
crime_data.SHOOTING.value_counts(dropna=False)
```

```
False    415383
True       1723
Name: SHOOTING, dtype: int64
```

Figure 1.6: *Code for replacing the missing values*

After confirming that there are no missing values anymore, we then replaced the “Y” and “N” with “True” and “False,” respectively, as we know that to represent a Boolean value, we either need 1/0 or True/False.



```
crime_data.STREET.value_counts(dropna=False)
```

```
WASHINGTON ST           18869
NaN                     12233
BLUE HILL AVE           10347
BOYLSTON ST              9329
DORCHESTER AVE           6584
TREMONT ST               6461
HARRISON AVE             6237
MASSACHUSETTS AVE        6204
CENTRE ST                5773
COMMONWEALTH AVE         5394
HYDE PARK AVE            4635
COLUMBIA RD              4227
HUNTINGTON AVE           3911
RIVER ST                 3798
```

Figure 1.7: *Frequency table for “STREET”*

But if we look at the other feature, i.e., “`STREET`” (*Figure 1.7*), we can see that it is not a binary value. So, figuring out the right missing value is next to impossible. In these cases, we need to use a different strategy to tackle this problem. Individually, finding the solution for columns can be fruitful, but here we will make a general approach to solve the missing value.

Similarly, we need to find the missing values for all the columns and handle them in a generic pattern.

As we can see from the above figure (*Figure 1.4*), features in *Figure 1.5* are missing with their respective missing count.

Now we have a list of the features which have missing values and also their respective dtypes. Seeing the data and the contents, we can design a strategy to fill the missing values.

In this chapter, we will discuss a few of the techniques for missing values. While designing the strategy, we need to take care of all the data types like numeric, string/object, etc. and fill them logically.

Let us take a look at another unique column and a unique way to handle the missing value. How should we fill the missing values for the column “Lat” and “Long”? (refer to *Figure 1.2* and *Figure 1.3*).

Suppose we fill the integer missing values with the mean or median of that feature column — that can be one of the strategies. Still, in this case, we see “Lat” and “Long” are the coordinates, and therefore, filling it with the mean value of “Lat” and “Long” column respectively may give a location which is not in Boston, and it won’t be accurate to perform that action. Filling it with the median value won’t give us any wrong location, but it will increase the rate of the Crime Incident by 27,378 for that median location.

If we look carefully at the other features, we have a feature called “Location” which has no missing values and it is a composite field like “(<Lat>, <Long>).” So, we don’t have to handle it separately as we can extract “Lat” and “Long” from the feature “Location” and save it.

Later, we will slowly fill the missing values and get as much information as possible from the data.

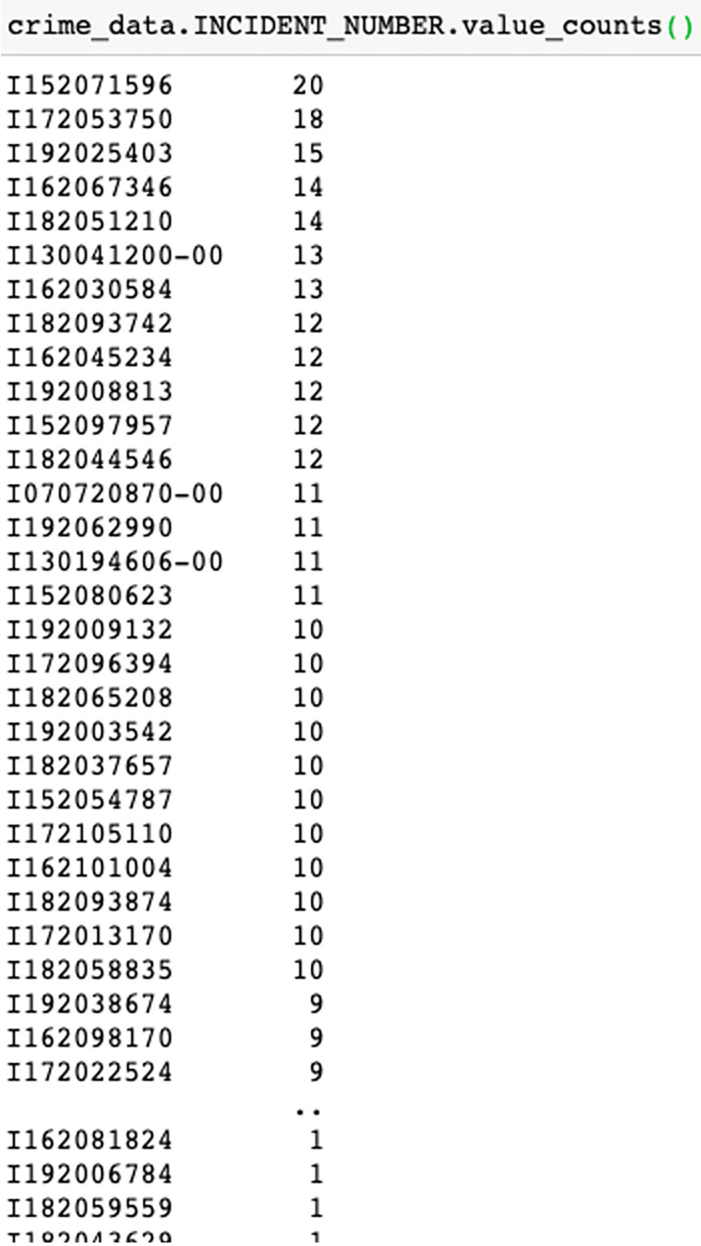
What to Do with Duplicate Values?

As we know, this is a structured dataset from Boston Crime Police Department, and we have seen before that structured data is obtained and can be stored in a relational database. Knowing the above fact, we also know that there will be a Primary key[10] for the table here. It is “`INCIDENT_NUMBER`.” We are sure that “`INCIDENT_NUMBER`” cannot be Null[11] , and the values should be unique as it is the Primary Key.

Referring to *Figure 1.8,* we know that there are many duplicates of the same “`INCIDENT_NUMBER`.” Now we should carefully inspect all the duplicate values and start thinking of a solution.

[10] “Primary key - Wikipedia.” https://en.wikipedia.org/wiki/Primary_key.

[11] “Null (SQL) - Wikipedia.” https://en.wikipedia.org/wiki/Null_(SQL).



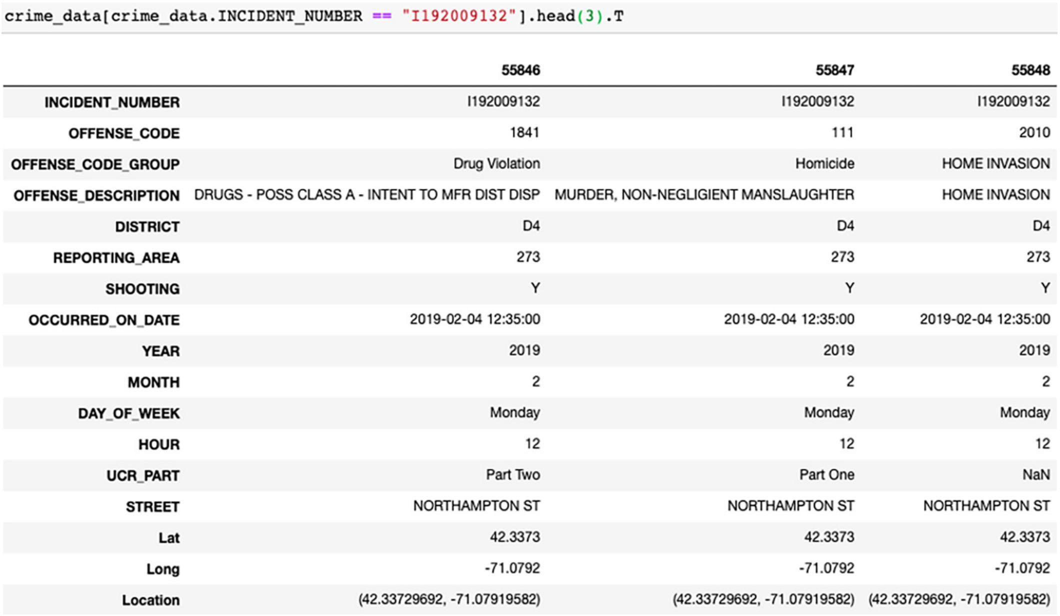
```
crime_data.INCIDENT_NUMBER.value_counts()
```

```
I152071596       20
I172053750       18
I192025403       15
I162067346       14
I182051210       14
I130041200-00    13
I162030584       13
I182093742       12
I162045234       12
I192008813       12
I152097957       12
I182044546       12
I070720870-00    11
I192062990       11
I130194606-00    11
I152080623       11
I192009132       10
I172096394       10
I182065208       10
I192003542       10
I182037657       10
I152054787       10
I172105110       10
I162101004       10
I182093874       10
I172013170       10
I182058835       10
I192038674        9
I162098170        9
I172022524        9
                 ..
I162081824        1
I192006784        1
I182059559        1
I182043629        1
```

Figure 1.8: *Sorted frequency table*

There can be situations where the same primary key is continuously updated with new updates of the same "`INCIDENT_NUMBER`."

For simplicity, we will be using only three records (see *Figure 1.9*) of the incident number "`I192009231`" which has ten duplicates.



```
crime_data[crime_data.INCIDENT_NUMBER == "I192009132"].head(3).T
```

	55846	55847	55848
INCIDENT_NUMBER	I192009132	I192009132	I192009132
OFFENSE_CODE	1841	111	2010
OFFENSE_CODE_GROUP	Drug Violation	Homicide	HOME INVASION
OFFENSE_DESCRIPTION	DRUGS - POSS CLASS A - INTENT TO MFR DIST DISP	MURDER, NON-NEGLIGIENT MANSLAUGHTER	HOME INVASION
DISTRICT	D4	D4	D4
REPORTING_AREA	273	273	273
SHOOTING	Y	Y	Y
OCCURRED_ON_DATE	2019-02-04 12:35:00	2019-02-04 12:35:00	2019-02-04 12:35:00
YEAR	2019	2019	2019
MONTH	2	2	2
DAY_OF_WEEK	Monday	Monday	Monday
HOUR	12	12	12
UCR_PART	Part Two	Part One	NaN
STREET	NORTHAMPTON ST	NORTHAMPTON ST	NORTHAMPTON ST
Lat	42.3373	42.3373	42.3373
Long	-71.0792	-71.0792	-71.0792
Location	(42.33729692, -71.07919582)	(42.33729692, -71.07919582)	(42.33729692, -71.07919582)

***Figure 1.9:** Viewing some of the duplicate records*

Here, there will be two categories of features: one which will be similar throughout the duplicates and two which will change and will be inconsistent throughout the duplicates

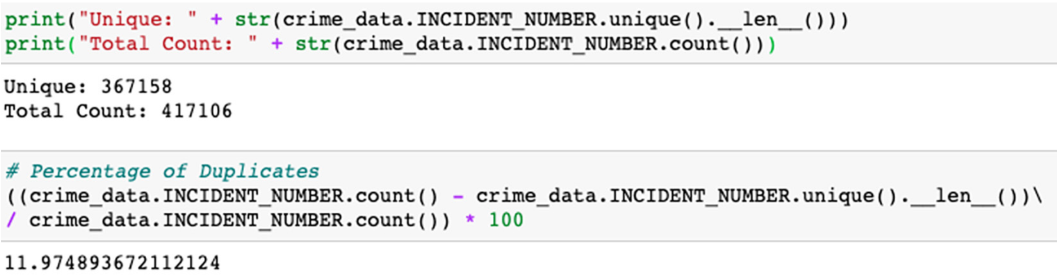
Figure 1.9 clearly shows “`OFFENSE_DESCRIPTION`” is different for all the records but “`Location`,” “`OCCURRED_ON_DATE`,” are the same.

We can conclude that there is a discrepancy in the data, and we need to pick some more random samples and test the hypothesis.

The real problem is which record to delete, and it is extremely difficult to find that information. Rather we can see how many unique values there are for “`INCIDENT_NUMBER`.” This will let us decide whether the change is a significant number or not.

As we can see in *Figure 1.10*, we only have approx. 12% of duplicate data, and considering the data size, we can see it is not statistically significant[12]. So, we can either keep or remove the duplicates because the change won’t impact the analysis significantly.

[12] “Statistical significance - Wikipedia.” https://en.wikipedia.org/wiki/Statistical_significance.



```
print("Unique: " + str(crime_data.INCIDENT_NUMBER.unique().__len__()))
print("Total Count: " + str(crime_data.INCIDENT_NUMBER.count()))
```

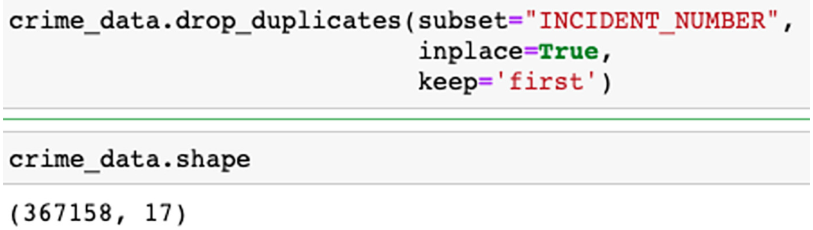
```
Unique: 367158
Total Count: 417106
```

```
# Percentage of Duplicates
((crime_data.INCIDENT_NUMBER.count() - crime_data.INCIDENT_NUMBER.unique().__len__())\
/ crime_data.INCIDENT_NUMBER.count()) * 100
```

```
11.974893672112124
```

Figure 1.10: *Unique count with percentage*

As the percentage of duplicate data is less, and it is distributed among multiple "`INCIDENT_NUMBERS`", we can ignore the face and keep all of the records. But if we are willing to remove the duplicates that can be achieved as well as seen in *Figure 1.11*.



```
crime_data.drop_duplicates(subset="INCIDENT_NUMBER",
                           inplace=True,
                           keep='first')
```

```
crime_data.shape
```

```
(367158, 17)
```

Figure 1.11: *Dropping duplicates*

So, the strategy to drop the duplicates is to treat the first duplicate record as unique and drop the rest of the records. Now we have all the unique entries for the "`INCIDENT_NUMBERS`." Doing basic cleaning, we will be jumping to Data Exploring.

What is Data Exploring?

Data Exploring is a combination of Art and Science. We need to ask the right kind of question and use the right tool to analyze the results.

There are multiple ways to get information about individual features. Here we will discuss some of the important feature analysis techniques and a few common things to infer during the analysis.

Data Analysis is a process of inspecting, cleansing, transforming, and modeling data to find new insights, draw conclusions, and supporting decision-making[13].

To start the **Exploratory Data Analysis** (**EDA**) we need to know what is EDA.

[13] "Data analysis - Wikipedia." https://en.wikipedia.org/wiki/Data_analysis.

Exploratory Data Analysis (EDA)[14] is an approach/philosophy to analyze a given data and derive information about the characteristics of the data using Graphical Visualizations like Box Plots, Scatter Plots, Histograms, etc. EDA is different from Initial Data Analysis (IDA)[15]. IDA primarily focused on Data Quality and Data Cleaning.

EDA can we further divided into the following categories:

1. Univariate Feature Analysis
    a. Distribution
    b. Anomaly or Outlier Detection
2. Multivariate Feature Analysis
    a. Correlation
    b. Dependence

Here we will cover only some of the concepts of EDA techniques, but, in the later chapters, we will dive deep into some of the mathematics behind the techniques which are commonly used.

Univariate Feature Analysis

After cleaning the data, we need to visualize how the data looks and is distributed among the dataset.

As the name suggests, "univariate" implies that we will be dealing with only one feature from the dataset.

What is Statistical Distribution[16]?

Distribution in the most general and simple terms is, how the frequency of every unique observation looks over the sample space. We can have either a discrete distribution or a continuous distribution. Generally, we use a bar chart that is generated from a frequency table. A **discrete distribution** means that the observation has a countable (usually finite) number of values. In contrast, a continuous distribution means that the observation has an infinite (uncountable) number of different values. In the later chapters, we will take a look at the types of distribution.

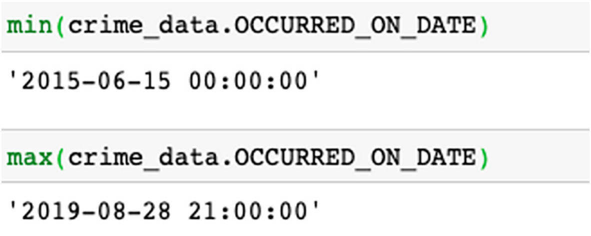
Let us see an example for each of the distribution.

[14] "1. Exploratory Data Analysis." https://www.itl.nist.gov/div898/handbook/eda/eda.htm.

[15] "A systematic approach to initial data analysis is ... - ScienceDirect.com." https://www.sciencedirect.com/science/article/pii/S0022522315017948.

[16] "Statistical Distribution -- from Wolfram MathWorld." http://mathworld.wolfram.com/StatisticalDistribution.html.

For a discrete distribution, we will be using the feature "`DISTRICT`," and we can see the number of crimes that occurred from the entire range of dates in *Figure 1.12*.



```
min(crime_data.OCCURRED_ON_DATE)
```

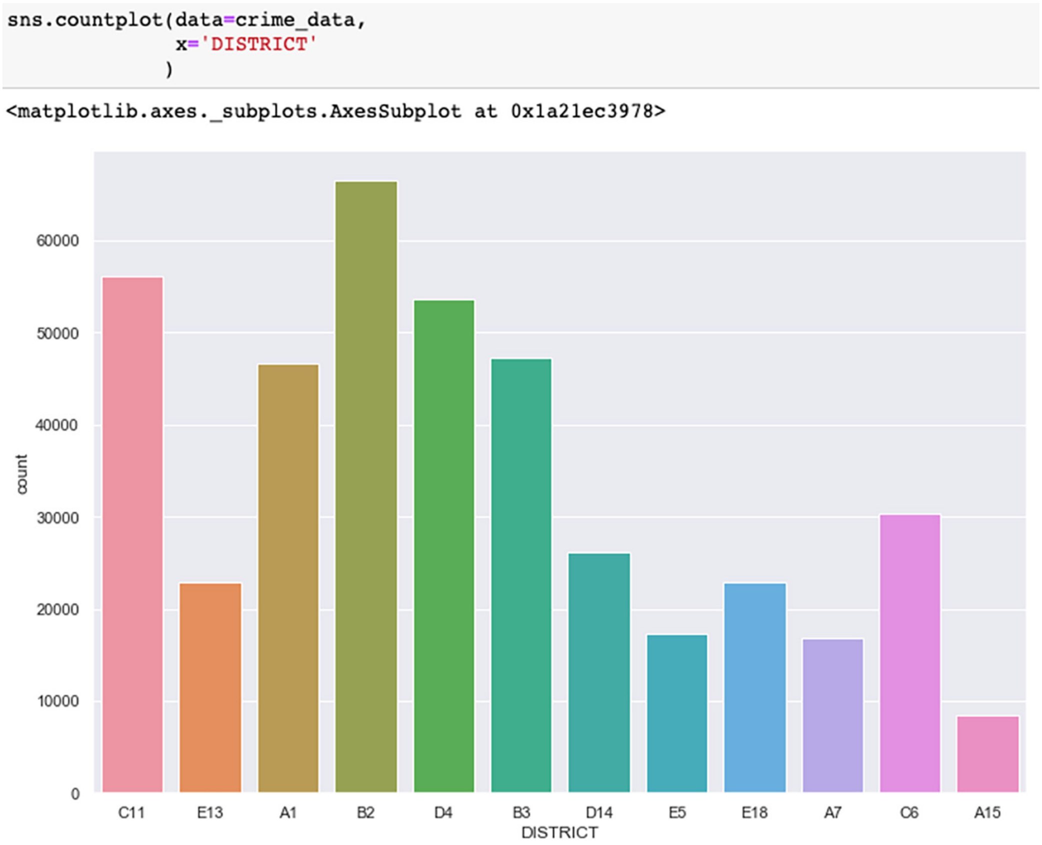
```
'2015-06-15 00:00:00'
```

```
max(crime_data.OCCURRED_ON_DATE)
```

```
'2019-08-28 21:00:00'
```

Figure 1.12: *Minimum and maximum values*

So, from *Figure 1.13*, we can conclude that the district "`B2`" has the highest crime rates from 2015 to 2019, followed by "`C11`".

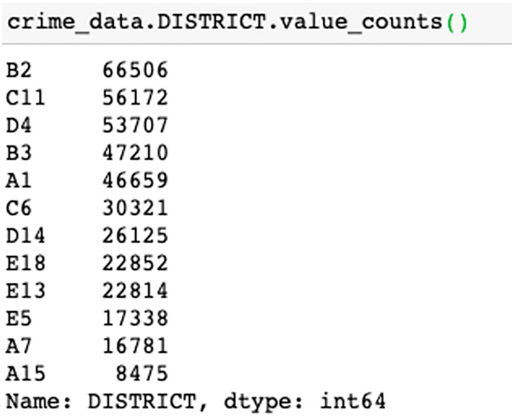


```
sns.countplot(data=crime_data,
              x='DISTRICT'
             )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a21ec3978>
```

Figure 1.13: *Count distribution w.r.t. district*

Below is the frequency table (*Figure 1.14*) for the above chart generated. We can see the crime rates ordered from highest to lowest values.



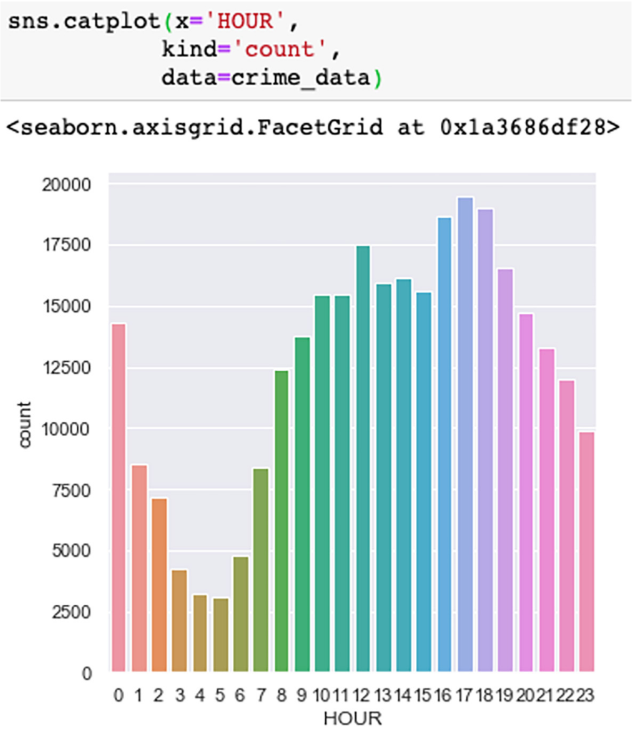
```
crime_data.DISTRICT.value_counts()
```

```
B2      66506
C11     56172
D4      53707
B3      47210
A1      46659
C6      30321
D14     26125
E18     22852
E13     22814
E5      17338
A7      16781
A15      8475
Name: DISTRICT, dtype: int64
```

Figure 1.14: *Frequency table for crime rate w.r.t. district*

So far, we have seen discrete distribution. Now we will take some examples.

Figure 1.15 is a continuous distribution for crime rates during a particular hour of the day.

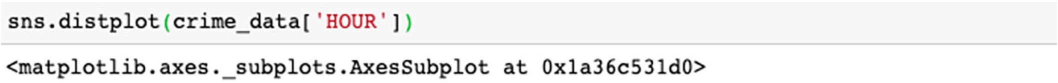


```
sns.catplot(x='HOUR',
            kind='count',
            data=crime_data)
```

```
<seaborn.axisgrid.FacetGrid at 0x1a3686df28>
```

Figure 1.15: *Bar graph for hourly crime rates*

This looks more like a discrete bar graph, but this is a continuous feature and we need to know the distribution of count throughout the dataset. *Figure 1.16* shows a continuous distribution (we will discuss the distribution in later chapters) of the feature “HOUR.”



```
sns.distplot(crime_data['HOUR'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a36c531d0>
```

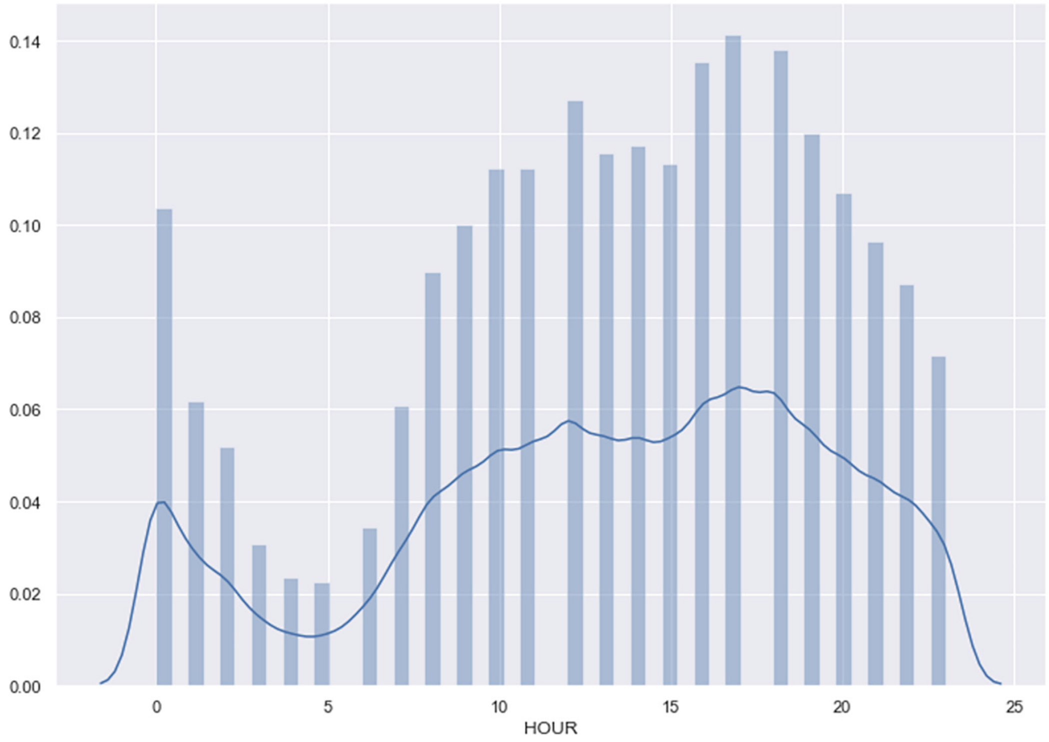


Figure 1.16: *Continuous distribution plot for hourly crime rates*

We can infer that crime rates are highest from 16:00 hour to 20:00 hour, and again there is a rise in crime rates at midnight 00:00 hours. As this is considering the entire dataset, we can create a hypothesis[17] that each year will have a similar distribution.

Multivariate Feature Analysis

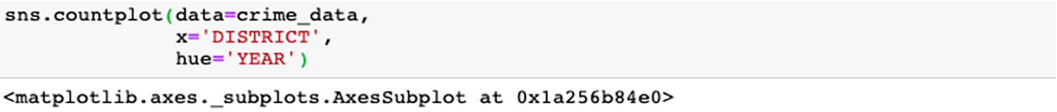
Multivariate feature Analysis[18] refers to analyzing more than one feature at a time.

To analyze more than one feature at the same time, we have multiple techniques like Regression Analysis[19], Cluster Analysis[20], Correlation Analysis, etc.

[17] “Hypothesis - Wikipedia.” https://en.wikipedia.org/wiki/Hypothesis.
[18] “Multivariate analysis - Wikipedia.” https://en.wikipedia.org/wiki/Multivariate_analysis.
[19] “Regression analysis - Wikipedia.” https://en.wikipedia.org/wiki/Regression_analysis.
[20] “Cluster analysis - Wikipedia.” https://en.wikipedia.org/wiki/Cluster_analysis.

Let us consider the above case where we were discussing crime rates throughout different districts within a range of time (refer to *Figure 1.12*). But we need to know how the distributions look annually because that will give us a clearer idea about the crime rates. There can be a case where the highest crime rated district will change from year to year. This way we will find more information than the individual charts.

Below you can find the bar chart (*Figure 1.17*) and we can see the distribution of each district.



```
sns.countplot(data=crime_data,
              x='DISTRICT',
              hue='YEAR')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a256b84e0>
```

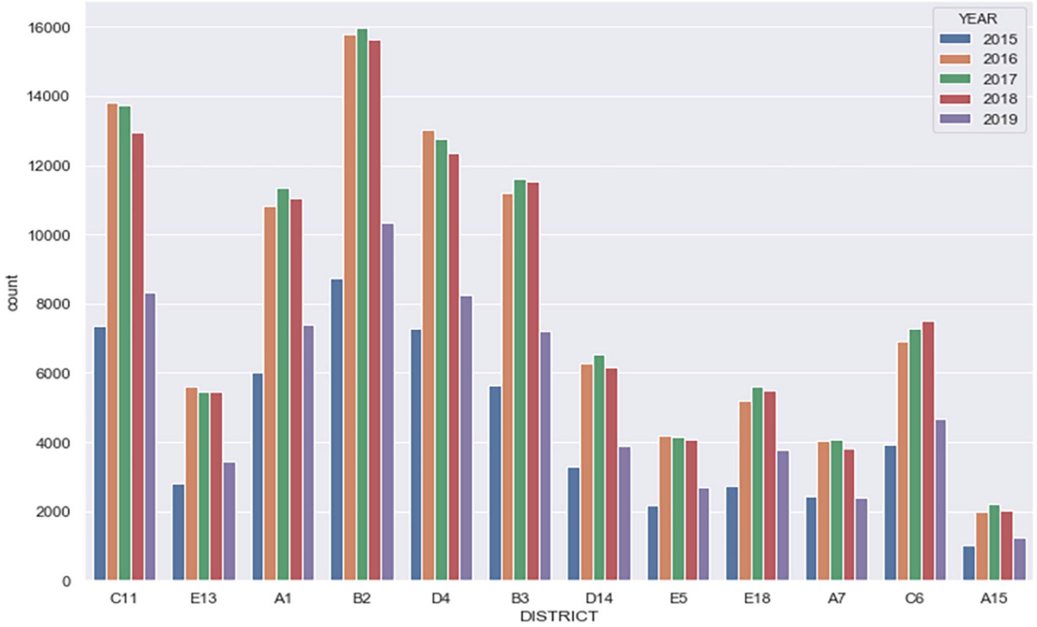


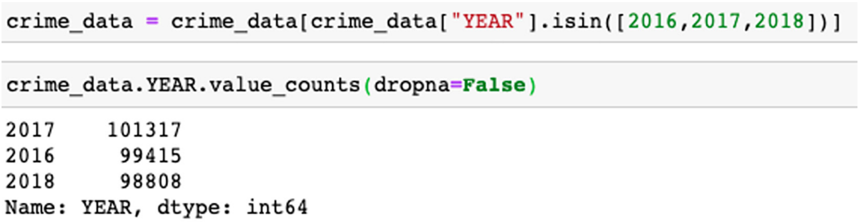
***Figure 1.17:** Count distribution of crime rates for different districts*

From the chart, we can see there is a spike of Crime from the Year 2015 to the Year 2016 for all the Districts. And there is a sudden decrease in Crime rates in 2019. This is absurd if we think logically. So now we need to investigate why it is so.

If we carefully take a look at *Figure 1.12*, we can see the maximum and the minimum time. If we see the minimum date "2015-06-15 00:00:00", we can see that approximately the first six months we do not have any data, and similarly for the maximum date "2019-08-28 21:00:00" we only have data until August.

Seeing the partial data, we can drop all the records for the years 2015 and 2019. After dropping, we will have a complete set of data that can be analyzed annually. But if we want to analyze the data monthly, then we will keep the data for all the months.

Let us drop the records for the years 2015 and 2019 and we will see similar data and analyze the change in *Figure 1.18*.



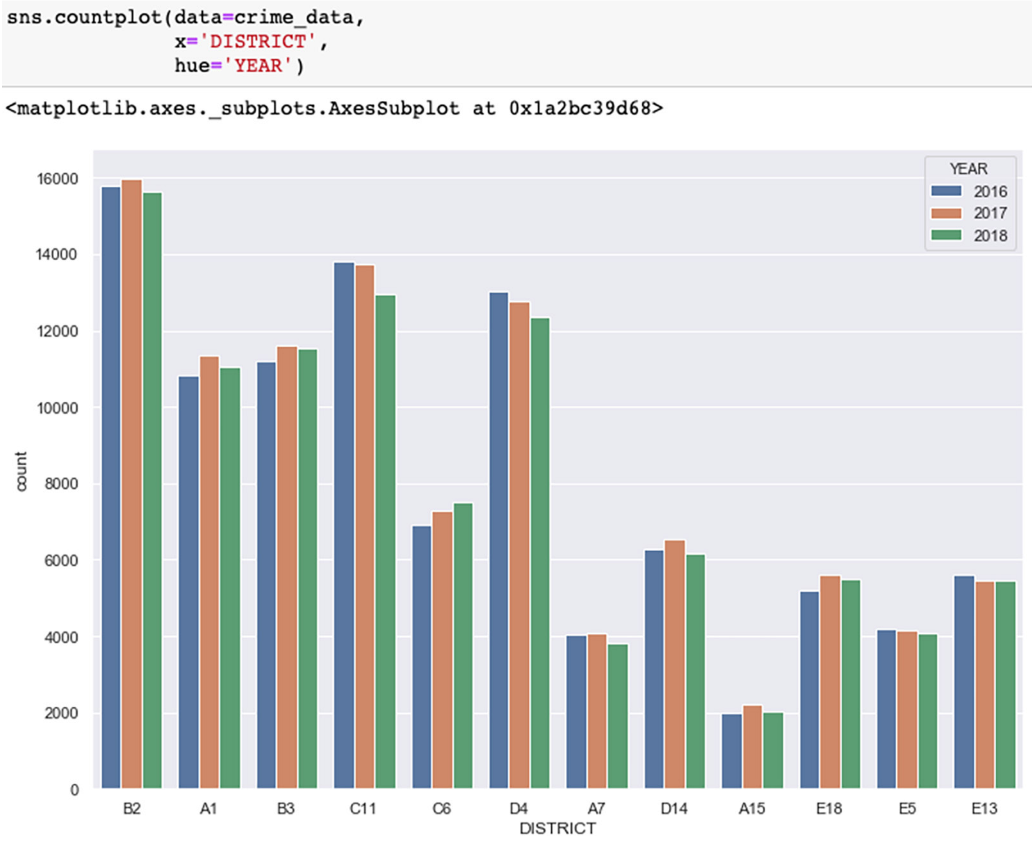
```
crime_data = crime_data[crime_data["YEAR"].isin([2016,2017,2018])]
```

```
crime_data.YEAR.value_counts(dropna=False)
```

```
2017    101317
2016     99415
2018     98808
Name: YEAR, dtype: int64
```

Figure 1.18: *Frequency table for years with complete data*

After the change, we need to determine how it compares with the previous distribution which had all the "YEARS" in it. Seeing *Figure 1.19*, we now have the correct details. With the complete details, we now expect there won't be any change in frequency, as we had seen for years 2015 and 2019.



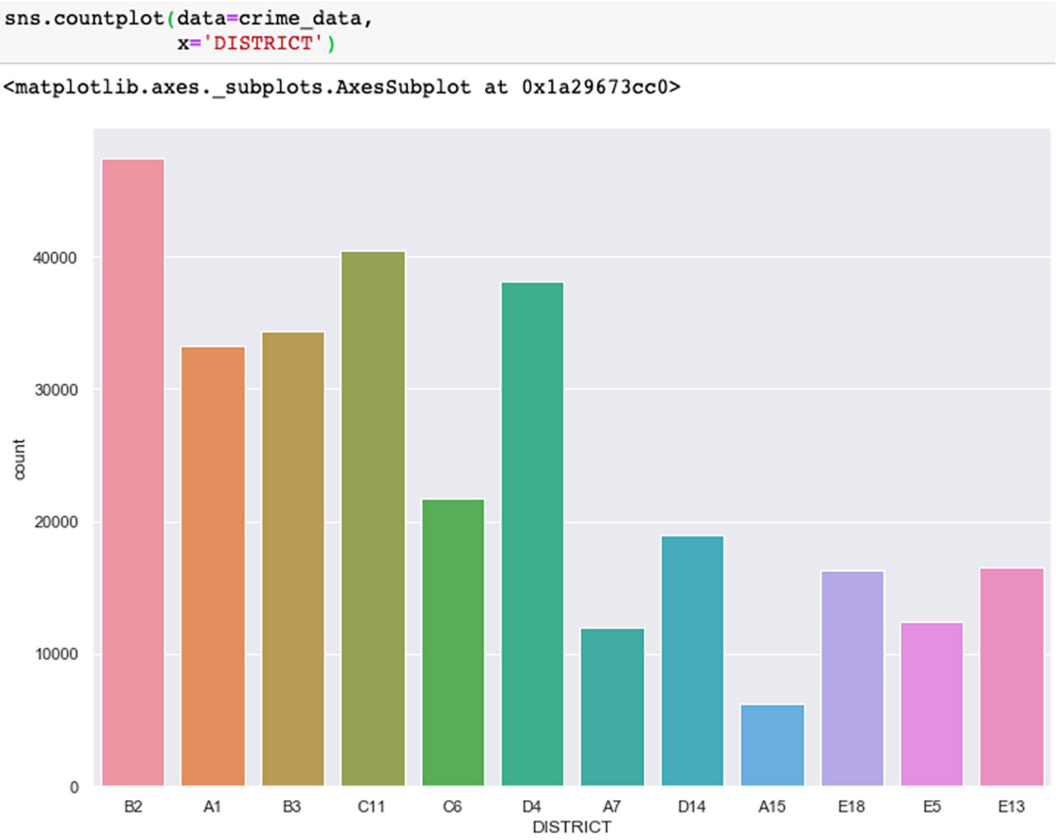
```
sns.countplot(data=crime_data,
              x='DISTRICT',
              hue='YEAR')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2bc39d68>
```

Figure 1.19: *Count distribution of crime rates for different districts (updated data)*

As this was a later observation and a new find, we need to recreate all the visualization regarding the same feature.

As we see in *Figure 1.13*, it is a crime rate distribution w.r.t the districts, and it is correlated with the feature “`YEAR`”. And that is the only reason we need to recreate those charts with the new data (*Figure 1.20*).

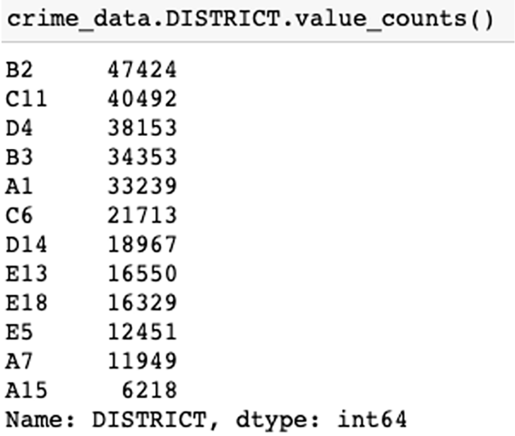


```
sns.countplot(data=crime_data,
              x='DISTRICT')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a29673cc0>
```

Figure 1.20: *Count distribution w.r.t. district (updated data)*

The distribution has no major change with the order of the district crime rate. Each bar chart is generated from a frequency table, and for the above chart, we have the frequency table in *Figure 1.21*.



```
crime_data.DISTRICT.value_counts()
```

```
B2     47424
C11    40492
D4     38153
B3     34353
A1     33239
C6     21713
D14    18967
E13    16550
E18    16329
E5     12451
A7     11949
A15     6218
Name: DISTRICT, dtype: int64
```

Figure 1.21: *Frequency table for crime rate w.r.t. district (updated data)*

Now looking at the frequency table (*Figure 1.21*), we can see some change in the order of Districts "E13" and "E18" if compared with *Figure 1.14*.

We should always keep in mind that Data Exploring is an iterative process (*Figure 1.22*).

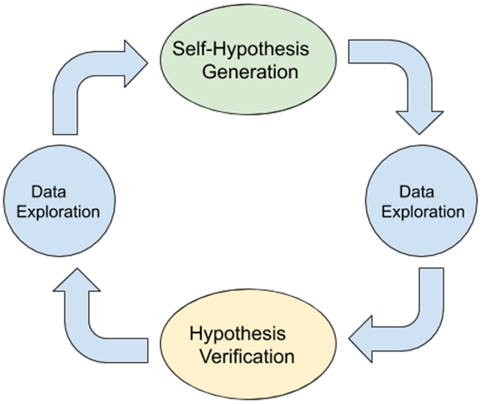


Figure 1.22: *Iterative process of data exploring*

We need to define some of the terms to understand the above chart in detail:

A Hypothesis[21] is a theory that is proposed with some limited knowledge of the data. For example, after seeing the data, we can guess that a particular feature has certain characteristics, or it will behave in a particular way.

When we assume something from the limited knowledge we have about the data, we call that hypothesis generation, and when you do it, it is known as self-hypothesis generation.

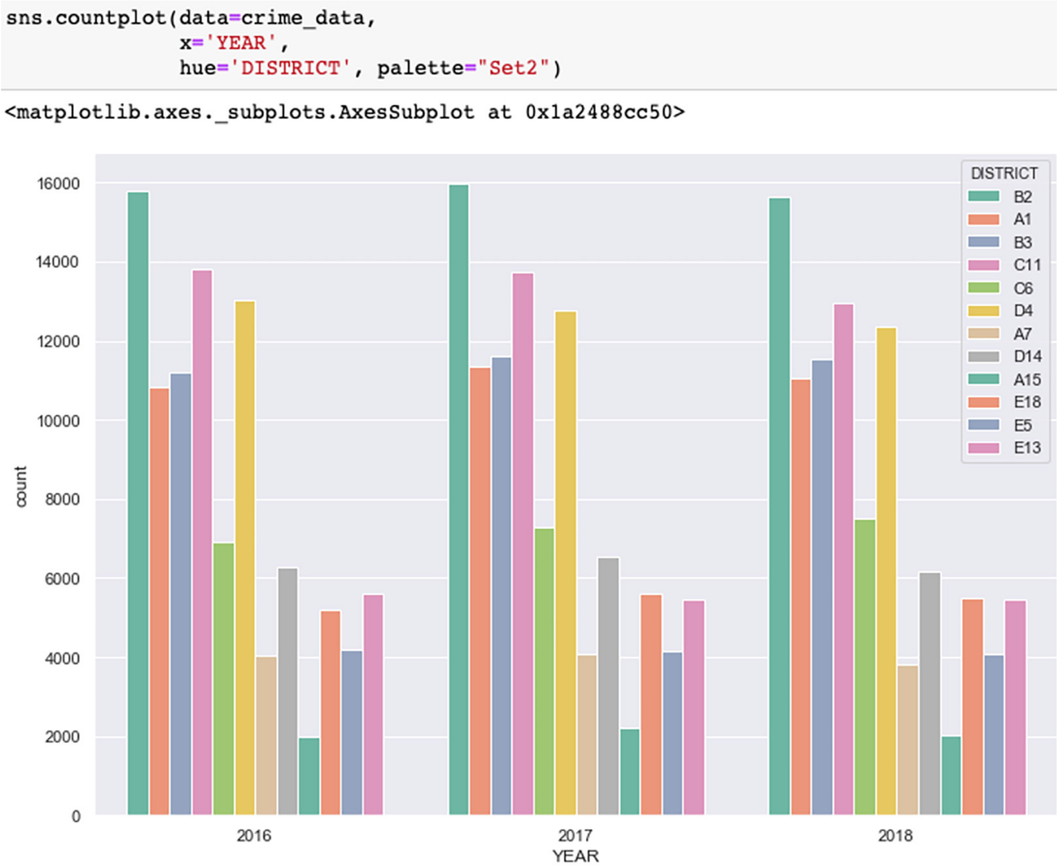
[21] "Hypothesis - Wikipedia." https://en.wikipedia.org/wiki/Hypothesis.

When we prove or validate or have concrete evidence to support our hypothesis, the process is known as hypothesis validation. Both Hypothesis Generation and Hypothesis Validation can be achieved through Data Exploring.

The entire process includes Self-Hypothesis Generation with the knowledge of the data which we are using, and then performing data exploring and verifying the hypothesis. This process will go on until and unless we see any saturation of hypothesis or cannot afford to spend more time on Data Exploring.

We now know the basics of Data Exploring, so we will take a few more examples and perform visualization to know more about the data.

If we carefully visualize the bar chart in *Figure 1.17*, it is very difficult to determine the crime rates annually as it is grouped by "`DISTRICT`" and not grouped by "`YEAR`". So, we need to update the chart for a better understanding of the updated data (*Figure 1.23*).

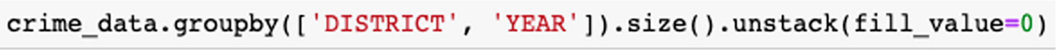


```
sns.countplot(data=crime_data,
              x='YEAR',
              hue='DISTRICT', palette="Set2")
```

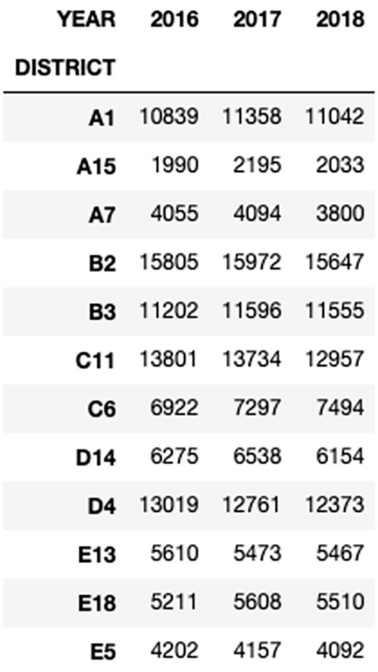
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2488cc50>
```

Figure 1.23: *Count distribution of crime rates for different districts and year*

Seeing the bar chart, we now understand the distribution of the crime count district-wise on an annual basis. As mentioned earlier, these count charts are generated from a frequency table (*Figure 1.24*).



```
crime_data.groupby(['DISTRICT', 'YEAR']).size().unstack(fill_value=0)
```



YEAR	2016	2017	2018
DISTRICT			
A1	10839	11358	11042
A15	1990	2195	2033
A7	4055	4094	3800
B2	15805	15972	15647
B3	11202	11596	11555
C11	13801	13734	12957
C6	6922	7297	7494
D14	6275	6538	6154
D4	13019	12761	12373
E13	5610	5473	5467
E18	5211	5608	5510
E5	4202	4157	4092

Figure 1.24: *Frequency table for multiple variables*

This type of frequency table is extremely helpful as it groups the entire Crime Rate frequency table (*Figure 1.21*) with another feature: "YEAR."

Let us try some other visualization to get some more information.

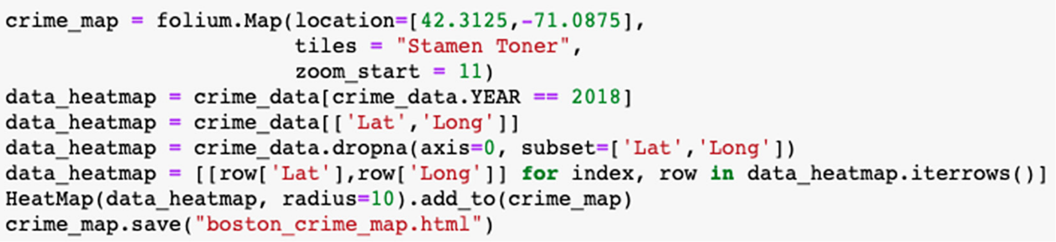
As "Lat" and "Long" stand for Latitude[22] and Longitude,[23] respectively, those values from the dataset can be used to plot a map with the count of crimes for each location. In this case, we will use something known as a heatmap[24] to represent the concentration/density of crimes for the location given in the dataset.

We have written a snippet that saves the map as an HTML file while executing the code given in *Figure 1.25*.

[22] "Latitude - Wikipedia." https://en.wikipedia.org/wiki/Latitude.

[23] "Longitude - Wikipedia." https://en.wikipedia.org/wiki/Longitude.

[24] "Heat map - Wikipedia." https://en.wikipedia.org/wiki/Heat_map.



```
crime_map = folium.Map(location=[42.3125,-71.0875],
                       tiles = "Stamen Toner",
                       zoom_start = 11)
data_heatmap = crime_data[crime_data.YEAR == 2018]
data_heatmap = crime_data[['Lat','Long']]
data_heatmap = crime_data.dropna(axis=0, subset=['Lat','Long'])
data_heatmap = [[row['Lat'],row['Long']] for index, row in data_heatmap.iterrows()]
HeatMap(data_heatmap, radius=10).add_to(crime_map)
crime_map.save("boston_crime_map.html")
```

Figure 1.25: *Plotting a map*

Opening the "`boston_crime_map.html`" will give you an interactive map where it can be zoomed. We can see a gradient of colours as per the density of the crime counts at the given longitude and latitude during the year 2018 (*Figure 1.26*).

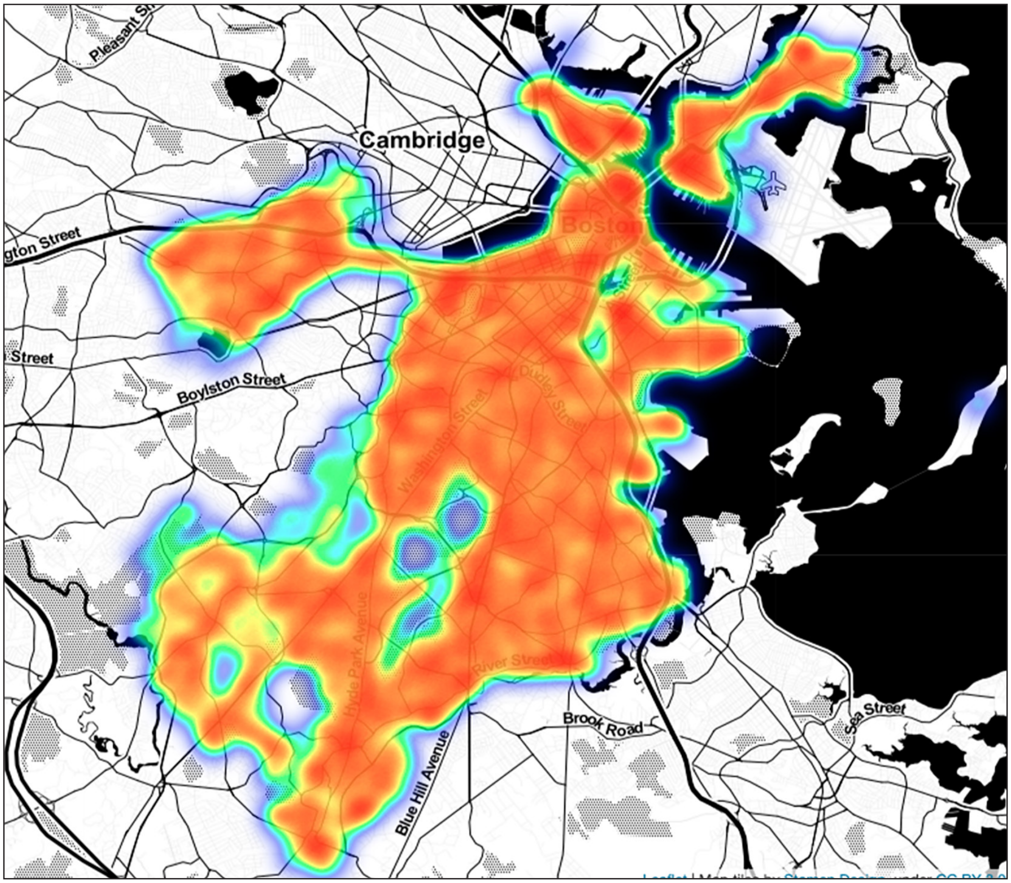


Figure 1.26: *Heat map for Boston crimes (zoomed out)*

"`boston_crime_map.html`" is an interactive map that can be zoomed in to see the details for each street (*Figure 1.27*).

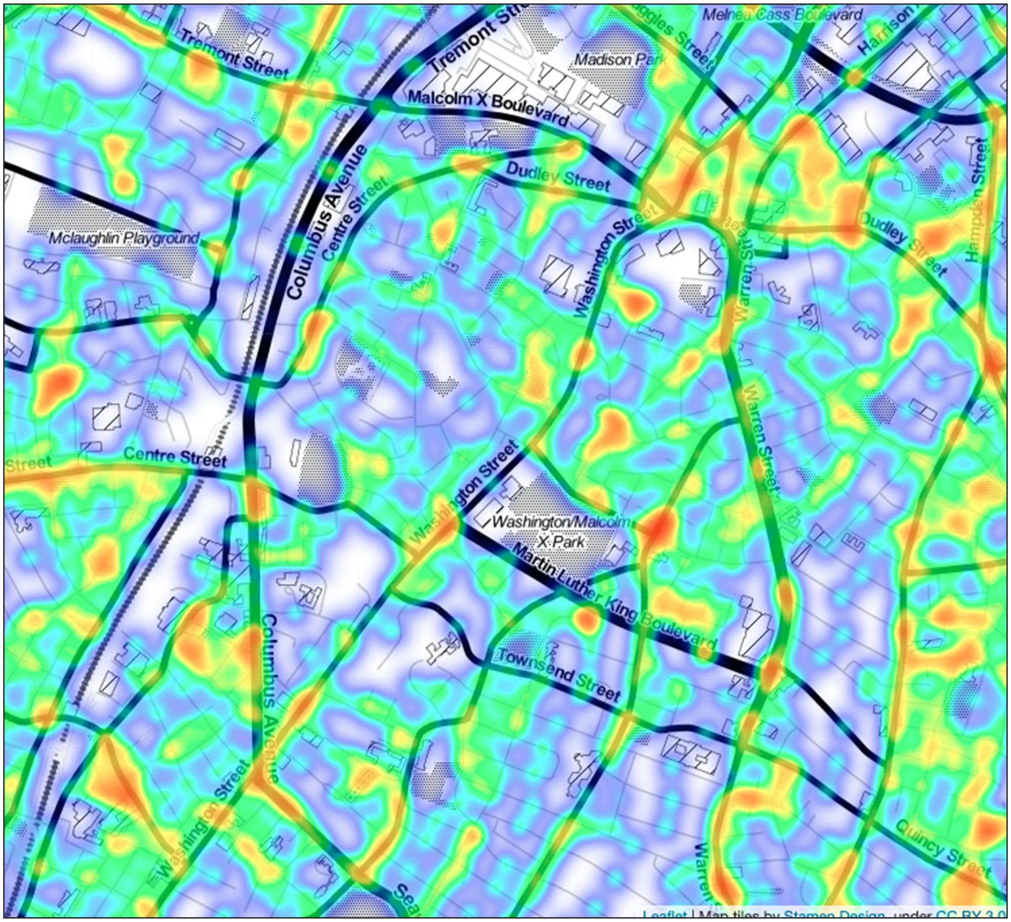


***Figure 1.27:** Heat map for Boston crimes in 2018 (zoomed in)*

The heat map shows the precise region of Boston, where the crime had taken place. With this method, we can analyze places with high crime rates. This plot is only for the year 2018, but as we know how to plot these graphs, we can plot for the other years, and then we can compare how it changes over the years.

Further reading

- Dense data
- Sparse data
- Plotly map
- Bubble map charts

- The philosophy behind Machine Learning
- Hypothesis and inference

Conclusion

What we have seen so far is the stepping stone to the world of Machine Learning and Data Science. We have mostly covered the concepts of O.S.E.M.N. framework and some of the basic techniques to handle the dataset.

If the reader has gone through the book, executed all the codes, then they will have a rough idea about how to handle and visualize data to draw some information out of it. Data Analysis is an important phase of Machine Learning and Data Analytics at the same time. Perusing the footnotes will provide a deeper understanding of the same concepts. The reader should keep in mind that Data Analysis and Analytics are a major part of the ML Pipeline.

In the next chapter, we will discuss different concepts like data types and probability distributions with an in-depth explanation. We will slowly use multiple concepts of mathematics and statistics to explain different techniques and how it works.

Knowing different techniques in depth will give us more insight into the data.

For practice, it will be best to use this dataset and try different types of visualization (for example, type of crime w/o time of day, density, etc.) and make a hypothesis for the same.

CHAPTER 2

World Happiness Report

Introduction

From the last chapter, we have seen a lot of bar charts, both discrete and continuous. You must have observed that all the patterns of the bar graph differ from one another. In this chapter, we will enlighten with questions like why that is so? And what kind of information does a different kind of charts give us? We will also look into mathematics for some of the concepts like Distribution, Skewness, etc.. A Machine Learning practitioner needs to know the Mathematical concepts behind the working of any algorithm as it will enable her/him to tune the model and later explain the working of the model.

Structure

- Let's talk about the dataset
- Pre-requisites
- Data exploration
- Distributions

Objective

This chapter will mainly focus on in-depth analysis of different probabilistic distributions and the main points we can infer from them. Along with the distribution, we will also take a look at the outlier detection method and how to find information from those visualizations.

Let's talk about the dataset

World Happiness Report[1] is an initiative taken by the United Nations, where 156 countries are surveyed on how happy their citizens perceive themselves to be. The size of the dataset is very less compared to other standard datasets. The main aim of this kind of data is to observe how happiness has evolved over the past years considering technology, conflicts, and government policies, social norms. As we have learned before that there can be different types of data, but this falls under the category of structured data.

Pre-requisites

To understand this chapter, we need some knowledge about statistics and probability.

We will take a quick recap of the terms and concepts which will be used in the following chapters. Later with these basic terms and definitions, we will build complex concepts.

It is recommended to go through all the footnotes, which will be linked along with the key recap topics for better understanding.

Time for a quick recapitulation:

1. **Statistics**: **Statistics**[2], in short, is a study of data[3]. It is a field of science that helps to conclude, extract facts and figures after analyzing the data. If you study what statistics is, it will differ from person to person and institution to institution because this field of study can be applied or implemented in multiple fields of study, to name a few psychology, genetics, etc.
2. **Statistical Mean:** Statistical Mean is the average of the entire set of numeric data. Below you can find the formula for different kinds of statistical mean, which is dependent on the data group.

[1] "Home | The World Happiness Report." https://worldhappiness.report/.

[2] "Statistics - Wikipedia." https://en.wikipedia.org/wiki/Statistics.

[3] "Data - Wikipedia." https://en.wikipedia.org/wiki/Data.

Population Mean(μ) is the mean or the average calculated for the entire set of data(N)

$$\mu = \frac{\sum_{i=0}^{N} x_i}{N}$$

Sample Mean($\overline{x}$) is the mean or the average calculated on a set of random variables(n) selected from the entire population(N).

$$\overline{x} = \frac{\sum_{i=0}^{N} x_i}{n}; where\ n \subset N$$

3. **Median: Median** is nothing but the middle value, which is separating the sorted list into two equal halves, upper half and a lower half.

   If the length of the list is odd then the mathematical formula for the median is:

$$Median = \left(\frac{n+1}{2}\right)^{th} term$$

   But, if the list is even then

$$Median = \frac{\left(\frac{n}{2}\right)^{th} term + \left(\frac{n}{2}+1\right)^{th} term}{2}$$

4. **Variance[4] and Standard Deviation[5]: Variance and Standard Deviation** gives us a numerical measure of the spread of a data set around the mean.

   Variance is a measure between the variables that how are they different from one another and how much are they different from one another.

   Population Variance

$$\sigma^2 = \frac{\sum_{i-0}^{N}\left(x_i - \mu\right)^2}{N}$$

   Sample Variance

$$s^2 = \frac{\sum_{i-0}^{N}\left(x_i - \overline{x}\right)^2}{N-1}$$

   It shows how the dataset or the values differ from the mean of the dataset.

   Standard Deviation is the root of the variance of the dataset.

   Population Standard Deviation

$$\sigma^2 = \sqrt{\frac{1}{N}\sum_{i-0}^{N}\left(x_i - \mu\right)^2}$$

[4] "Variance - Wikipedia." https://en.wikipedia.org/wiki/Variance.

[5] "Standard deviation - Wikipedia." https://en.wikipedia.org/wiki/Standard_deviation.

Sample Standard Deviation

$$s = \sqrt{\frac{1}{N-1}\sum_{i-0}^{N}\left(x_i - \overline{x}\right)^2}$$

If someone has a question in mind, that why a sample correction of $N - 1$ is used and not N for the sample variance and sample standard deviation, then they should read through Bessel's Correction[6]. The explanation of this concept is out of scope for this book.

5. **Probability theory[7]: Probability**[8] is a branch of mathematics that can be defined as the chance or likelihood that an event will occur. In this chapter, we will mostly use the concept of Likelihood, Expectations, Random Variable, Distributions, etc.

As we have covered some basics, so we will start with some visualizations. We will follow the same O.S.E.M.N. (Obtaining, Scrubbing, Exploring, Modelling, INterpreting) framework for this chapter. Still, as we have covered Data Obtaining and Data Scrubbing previously, we will skip those steps and directly start with Data Exploration. Please Note: As I have already explained the O.S.E.M.N. framework, we won't strictly follow the process in the following chapter as in real-life, we do follow a rough structure and not the exact one. Different people have different methodologies to achieve the same thing.

Data Exploration

Data Exploration can be of either Univariate Analysis or Multivariate Analysis; here, we won't specify any type, particularly as I am assuming that by now, readers can figure out which distribution is for Univariate Analysis and which one is Multivariate Analysis.

Before moving forward with the analysis, we will explain the different types of data and measurement of data. Previously we have seen different types of data in the form of the structure of the data as structured, semi-structured, and unstructured data but, here we will see different types of data from the perspective of values of the data.

Different types of data

When the values are numeric, it is extremely important to categorize it into different kinds.

[6] "Bessel's correction - Wikipedia." https://en.wikipedia.org/wiki/Bessel%27s_correction.

[7] "Probability theory - Wikipedia." https://en.wikipedia.org/wiki/Probability_theory.

[8] "Probability - Wikipedia." https://en.wikipedia.org/wiki/Probability.

Data represented on Nominal (Categorical) Scale

Data on the Nominal Scale means if we change the data for a record of this type, then it wouldn't alter the nature of the collection. It will be easier if I explain the same concept with an example. Let's consider a house. If we paint the house with different colors, we can confirm that it will still be a house.

Data represented on Ordinal Scale

On the other hand, data on the ordinal scale means the scale is ranked. Those numerical values (Ranks) only make sense when they are ordered that makes them ordinal scales.

The example over here will be a common and simpler one to understand. We have a rating system from 1 to 5, one being the worst/dissatisfied, and five being the best/satisfied. These values have additional information apart from the numerical value; it can also be considered as five different categories.

Measurement of data

Numerical measurements can be categorized into discrete and continuous types, and both of them can be represented in different scales, namely, interval, circular, and ratio.

It is extremely important to know and find out if the variable can be represented in any scale that will help us to find relationships between the variables. If none of the scales is sensible derivative, then we need to use special custom methods for those data columns to build the relation between them.

Discrete variables

Discrete variables are also known as meristic variables, which are generally counted. It only takes discrete values which are represented by natural numbers.

Continuous variables

Continuous variables are floating-point numbers whose precision is limited by the tools we are using. Example - To measure the thickness of a hair, say we have three different instruments regular centimeter ruler, a caliper, and a micrometer. All these instruments have different precision, the lowest precision being regular centimeter ruler, and the highest precision is the micrometer. These instruments generate continuous variables.

Interval scale

The **Interval scale** is a measurement where the difference of both the data points have a meaningful result. For example, if we have a **date** as one of the features, then the difference between them will give the number days. That information is extremely important for analysis and understanding of the data.

Circular scale

If we imagine a normal scale where both ends are joined, that will give us a circular scale. Some of the features will be an hour, day, month, annual dates, etc. Information extracted from these data like differences, ratios are not sensible derivatives, but for these features, we have other methods to analyze them.

Ratio scale

As the name says, the ratio scale is the ratio between two variables that will give a piece of information. Example: For baking a cake, say the ratio of flour, water, and butter is 5:2:1, i.e., if we use 5 KG of flour, then we need 2 KG of water and 1 KG of butter.

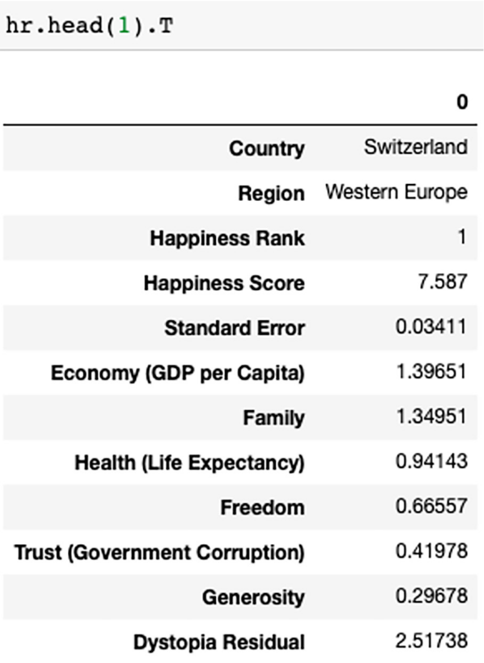
Distributions

The **distribution** of a statistical dataset is the plot that shows us the frequency of occurrence in the dataset.

In the world of a probability distribution, there are thousands of distributions we have. But mostly we will cover 4 to 5 important distribution which we will mostly encounter.

Here we will start with the basic analysis that will help through the chapter.

As this is a new dataset, we should see some sample data and what kind of columns/ features are there for the data.



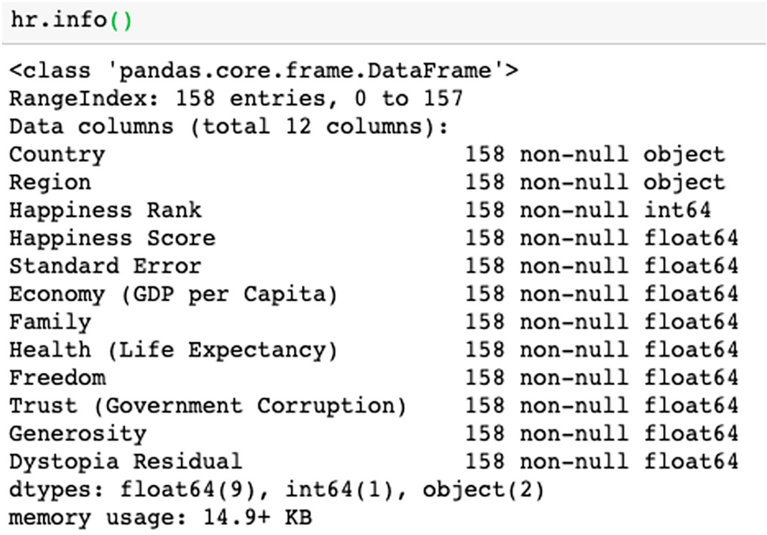
```
hr.head(1).T
```

	0
Country	Switzerland
Region	Western Europe
Happiness Rank	1
Happiness Score	7.587
Standard Error	0.03411
Economy (GDP per Capita)	1.39651
Family	1.34951
Health (Life Expectancy)	0.94143
Freedom	0.66557
Trust (Government Corruption)	0.41978
Generosity	0.29678
Dystopia Residual	2.51738

Figure 2.1: *Sample Data*

As we see above that mostly, we have numerical columns, and one of them is a categorical variable, i.e., “**Region,**” and another is a primary key, i.e., “**Country**.”

Previously we have just seen the dtypes of the columns; here, we will see a few more information on the data.



```
hr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158 entries, 0 to 157
Data columns (total 12 columns):
Country                          158 non-null object
Region                           158 non-null object
Happiness Rank                   158 non-null int64
Happiness Score                  158 non-null float64
Standard Error                   158 non-null float64
Economy (GDP per Capita)         158 non-null float64
Family                           158 non-null float64
Health (Life Expectancy)         158 non-null float64
Freedom                          158 non-null float64
Trust (Government Corruption)    158 non-null float64
Generosity                       158 non-null float64
Dystopia Residual                158 non-null float64
dtypes: float64(9), int64(1), object(2)
memory usage: 14.9+ KB
```

Figure 2.2: *Information for all the Datasets*

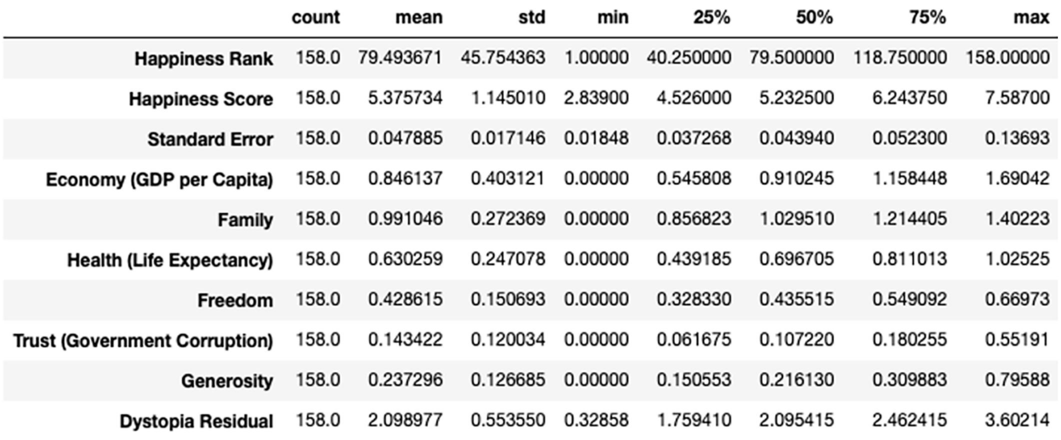
As we can see above that with the dtypes of the columns, we have information like null/non-null, which indicates whether the columns have any null values or not. We have the count of the rows and the total memory used for this dataset.

In the last chapter, we have shown different techniques for scrubbing the data, but, in this chapter, we will focus on different aspects of data analysis, which is not covered before.

This section will be highly related to different techniques of information extraction from probability distributions, so it is highly recommended to have some prior knowledge on probability distributions.

We start by seeing different statistical measures like mean, standard deviation, max, etc.

```
hr.describe().T
```



	count	mean	std	min	25%	50%	75%	max
Happiness Rank	158.0	79.493671	45.754363	1.00000	40.250000	79.500000	118.750000	158.00000
Happiness Score	158.0	5.375734	1.145010	2.83900	4.526000	5.232500	6.243750	7.58700
Standard Error	158.0	0.047885	0.017146	0.01848	0.037268	0.043940	0.052300	0.13693
Economy (GDP per Capita)	158.0	0.846137	0.403121	0.00000	0.545808	0.910245	1.158448	1.69042
Family	158.0	0.991046	0.272369	0.00000	0.856823	1.029510	1.214405	1.40223
Health (Life Expectancy)	158.0	0.630259	0.247078	0.00000	0.439185	0.696705	0.811013	1.02525
Freedom	158.0	0.428615	0.150693	0.00000	0.328330	0.435515	0.549092	0.66973
Trust (Government Corruption)	158.0	0.143422	0.120034	0.00000	0.061675	0.107220	0.180255	0.55191
Generosity	158.0	0.237296	0.126685	0.00000	0.150553	0.216130	0.309883	0.79588
Dystopia Residual	158.0	2.098977	0.553550	0.32858	1.759410	2.095415	2.462415	3.60214

Figure 2.3: *Description of the Table*

From all the statistical information from the description of the table, we can derive a few probability distributions, which will help us to interpret more information about the features.

For continuous variables, we generally get some below curves or some curves which will resemble any one of the below curves.

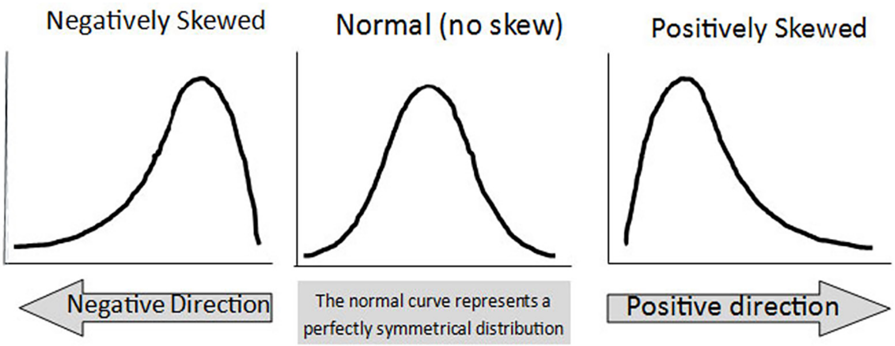


***Figure 2.4:** Distribution patterns*[9]

In the above image, we can see some curves and the world of distributions[10] is not limited to these curves; it's better to know some curves and their types because that will help us to analyze the wider spectrum of data.

Let us start with some basic distribution, as we had seen in previous chapters.

Normal distribution

The normal distribution is a type of probability distribution that is symmetric about the mean of the data, i.e., the density of the data near the mean value is higher than the tails like the below image.

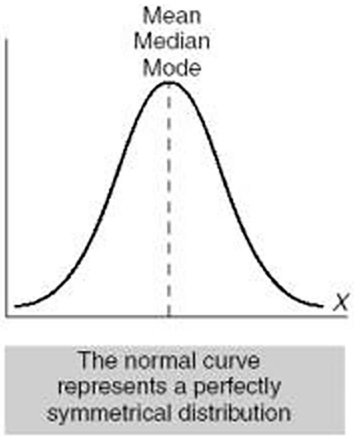

Mean
Median
Mode
X
The normal curve represents a perfectly symmetrical distribution

***Figure 2.5:** Normal Distribution Structure*

[9] "What is Skewness? | aiSource." https://www.managedfuturesinvesting.com/what-is-skewness/.

[10] "List of probability distributions - Wikipedia." https://en.wikipedia.org/wiki/List_of_probability_distributions.

Sometimes Normal Distribution is also known as Gaussian Distribution. When we plot a normal distribution, it will resemble a bell curve.

We should always remember that Normal or Gaussian Distribution is the yardstick to compare other distributions. We will never get the same distribution is a real-time dataset, but what we will get will be similar or close enough to that distribution.

In this case, for getting a close enough distribution, we are using the feature "Dystopia Residual"[11] from the dataset.

To know what exactly Dystopia Residual is, it's better to study about it in the link provided in the footnote.

Dystopia is an imaginary country that has the world's least happy people. It is used as a benchmark to compare other countries' happiness. So, this means all the records will be non-negative numbers.

Residuals are the un-explained components which differ from country to country.

For the "Dystopia Residual," we are expecting a Gaussian Distribution like the figure below (*Figure 2.6*).

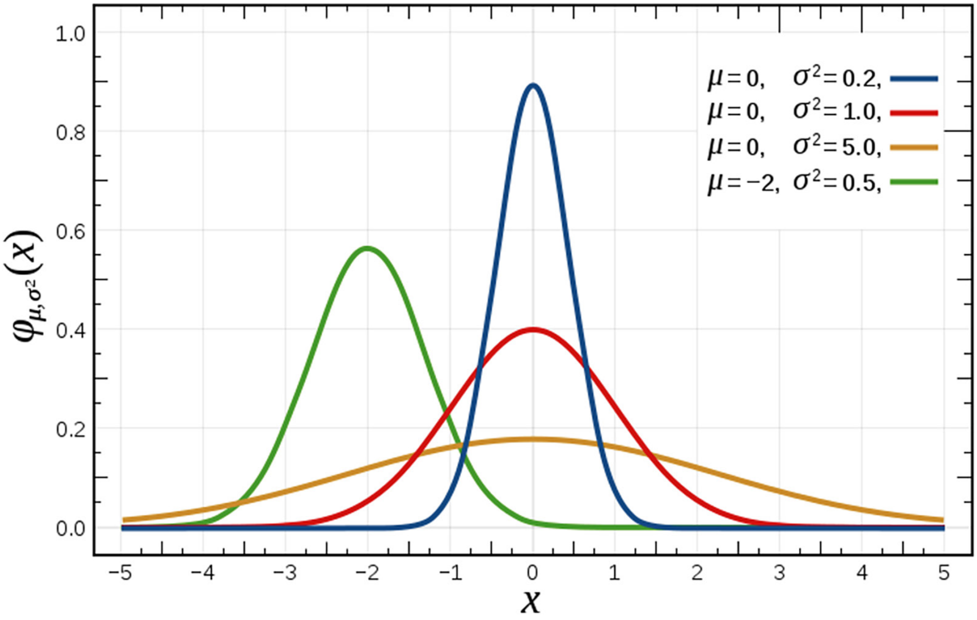


Figure 2.6: *Gaussian Distribution (Source: Wikipedia)*

[11] "FAQ | The World Happiness Report." https://worldhappiness.report/faq/.

The formula for Gaussian Distribution is stated below:

$$f\left(x \mid \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

We can see different behaviors of the Gaussian distribution, with different values of μ and σ^2.

We have seen Normal distribution and its structure, so we should also know about the Empirical Rule.

What is "The Empirical Rule" in Statistics?

The **Empirical Rule**[12] states that for a Gaussian/Normal distribution, around 68.2% will fall between the first standard deviation similarly within the second deviation we will have 95.4% of data. Lastly, for the third deviation there will 99.6% of the data (all the numbers are an approximation).

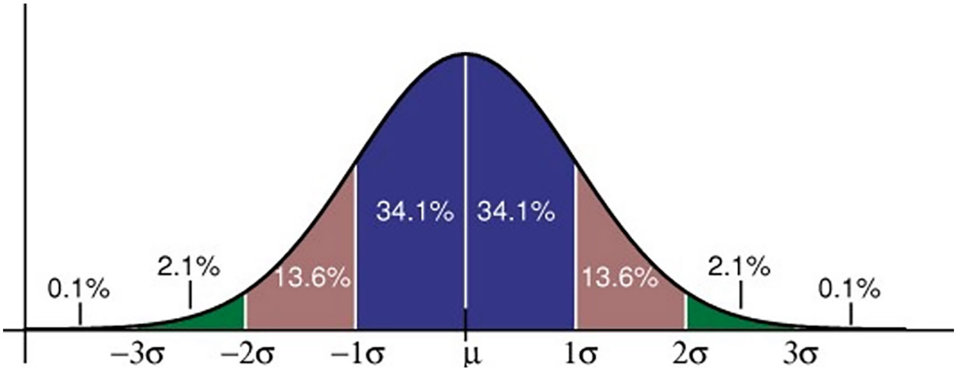


Figure 2.7: *The Empirical Rule*

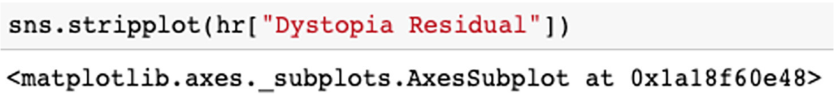
This distribution model is the ideal case for a Gaussian/Normal distribution. But, in reality, percentage values will differ with some error margin.

Distribution plot

To visualize how the feature "Dystopia Residual" will look in terms of distribution, we need to plot a frequency distribution. As we already know to plot, we need a frequency table, so we are skipping that part as here we want to focus on those areas which are not yet covered.

Let us plot the data points and see how it is distributed over the range of values for "Dystopia Residual."

[12] "68–95–99.7 rule - Wikipedia." https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule.



```
sns.stripplot(hr["Dystopia Residual"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a18f60e48>
```

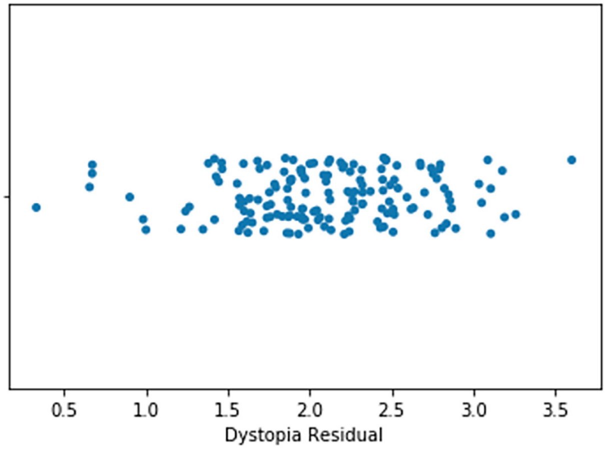
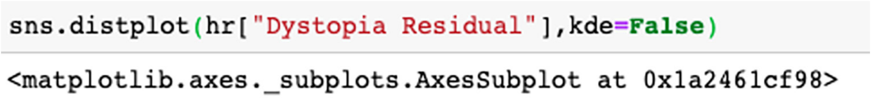


Figure 2.8: Strip plot

In the above diagram, you can see all the data points are overlapped, and we can see the density of the points.

The popular version of plotting something from a binned frequency table is a histogram.

In the below figure, you can find the histogram of the feature "Dystopia Residual," which is a continuous variable.



```
sns.distplot(hr["Dystopia Residual"],kde=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2461cf98>
```

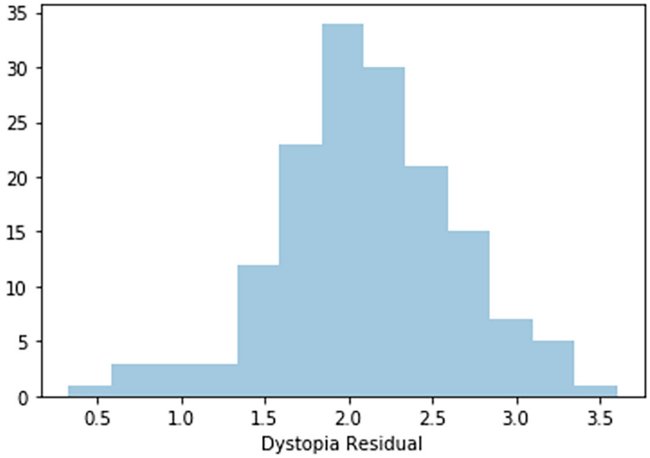


Figure 2.9: Normal Distribution Plot

Seeing the distribution, we know the maximum frequency lies in the range ~1.7 to ~2.5, and some values are distributed towards the tail.

Seeing the above diagram, it is tough to predict the distribution, but the figure looks similar to the Gaussian distribution. If we notice the Gaussian distribution, we will see it's a smooth curve, so we need to smooth the histogram[13] to get the required curve. There are multiple ways to do it, but here we will focus on Kernel Density Estimation to achieve a curve.

All the above charts mostly give us the same interpretation with some unique information as well. Throughout this chapter, we will encounter different charts and visualization that will yield similar or the same information, but it is extremely important to learn different techniques and keep them handy.

Kernel Density Estimation

Kernel Density Estimation[14] is commonly also known as KDE, which is a way to create a smooth curve given a dataset using a density function.

First, we should know what a Kernel is? A kernel is a special type of **Probability Density Function** (**PDF**) with a new property.

Some of the properties of Kernel method[15] that are listed below:

1. Real-valued
2. $K(x) \geq 0$ (Non-negative function)
3. Even function
4. Normalization such that $\int_{-\infty}^{+\infty} K(x)\, dx = 1$;

Now we know what Kernel is so we can define KDE as "Kernel density estimation is a non-parametric method of estimating the probability density function (PDF) of a continuous random variable. It is non-parametric because it does not assume any underlying distribution for the variable."

Below is the mathematical way of how Kernel Density Estimation ($\hat{f}_h$) is calculated.

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i-1}^{n} K_h(x - x_i)$$

$$\Rightarrow \frac{1}{nh}\sum_{i-1}^{n} K\left(\frac{x - x_i}{h}\right), where\ h > 0$$

[13] "Smoothing a histogram." http://www-ist.massey.ac.nz/dstirlin/cast/cast/snormal/normal1.html.

[14] "Kernel density estimation - Wikipedia." https://en.wikipedia.org/wiki/Kernel_density_estimation.

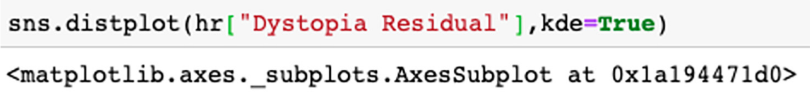
[15] "Kernel (statistics) - Wikipedia." https://en.wikipedia.org/wiki/Kernel_(statistics).

Where K is the Kernel, h is the bandwidth and K_h is the Scaled Kernel.

Scaled Kernel can be written as:

$$K_h = \frac{1}{h} K\left(\frac{x}{h}\right)$$

While using KDE, we generally use Gaussian Kernel. Let's overlay the Gaussian KDE with the histogram and let us plot another visualization to see how it looks. In the next section, we will cover a few kernels, and how does it differ from one another concerning the visualization. Below we will see KDE with a Histogram plot for "Dystopia Residual."



```
sns.distplot(hr["Dystopia Residual"],kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a194471d0>
```

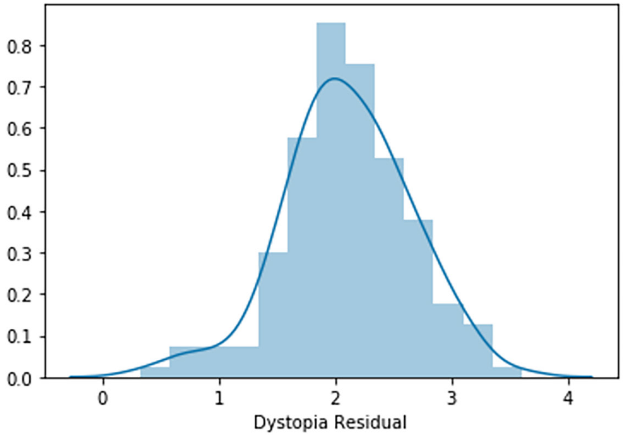


Figure 2.10: *KDE with Histogram for Dystopia Residual*

Now, let us interpret the above figures and find out what kind of information we can extract from it.

KDE shows a higher density when the frequency is high, and when the frequency is low, then the distribution density is very low.

Box plot

Box plot is a graphical method and a way to visualize the distribution using some of the computed values from the data, i.e., "Minimum," "Q1 - First Quartile", "Median," "Q3 - Third Quartile" and "Maximum."

Box Plot gives us information like the outliers, symmetry of the data, how are the data grouped, and the skewness of the data. Let us see the structure of the Box Plot and analyze it further.

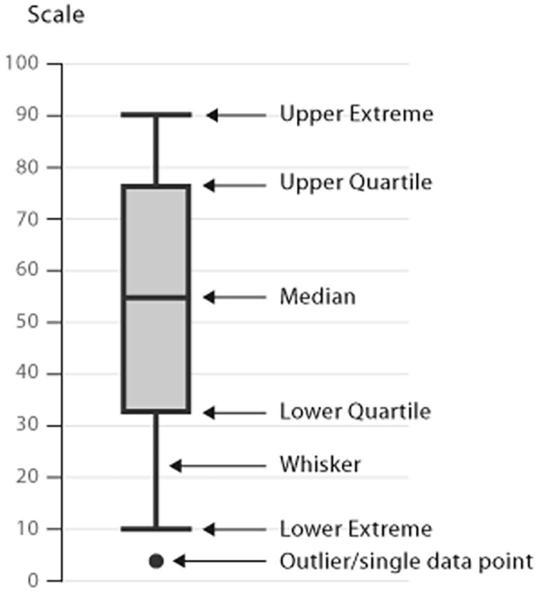


Figure 2.11: *Box Plot Structure*

Let us jump into some mathematics that will give us more clarity about the box plot.

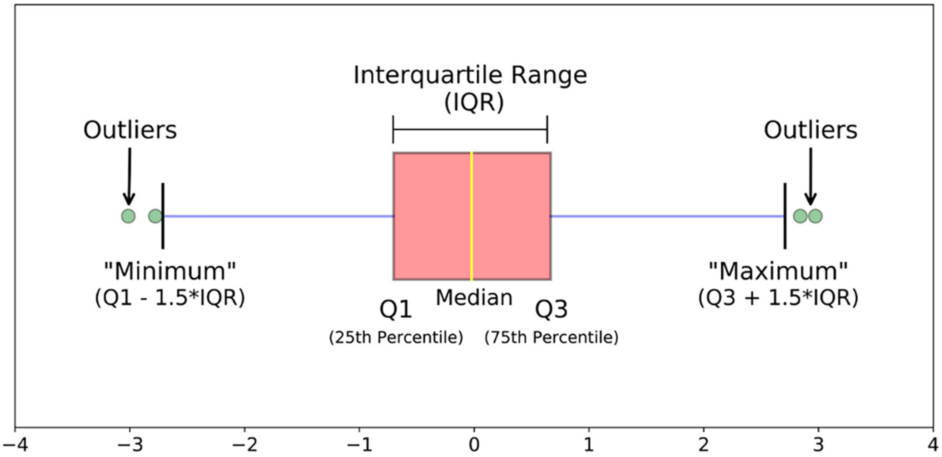


Figure 2.12: *Box Plot Calculations*

It will be way easier if we look at an example.

Let us take a series of numbers: {3, 7, 8, 5, 12, 14, 21, 13, 18}

Step 1: Sort the series

{3, 5, 7, 8, 12, 13, 14, 18, 21}

Step 2: Find Minimum, Median and Maximum

Minimum = 3

Median = 12

Maximum = 21

Step 3: Find Q1 and Q3

To find Q1, we need to find the median of the lower half from the median of the dataset.

Q1 = Median of {3, 5, 7, 8} = 6

Similarly, for Q2, we need to find the median of the upper half from the median of the dataset.

Q2 = Median of {13, 14, 18, 21} = 16

Generally, there are multiple ways to find the Minimum and Maximum.

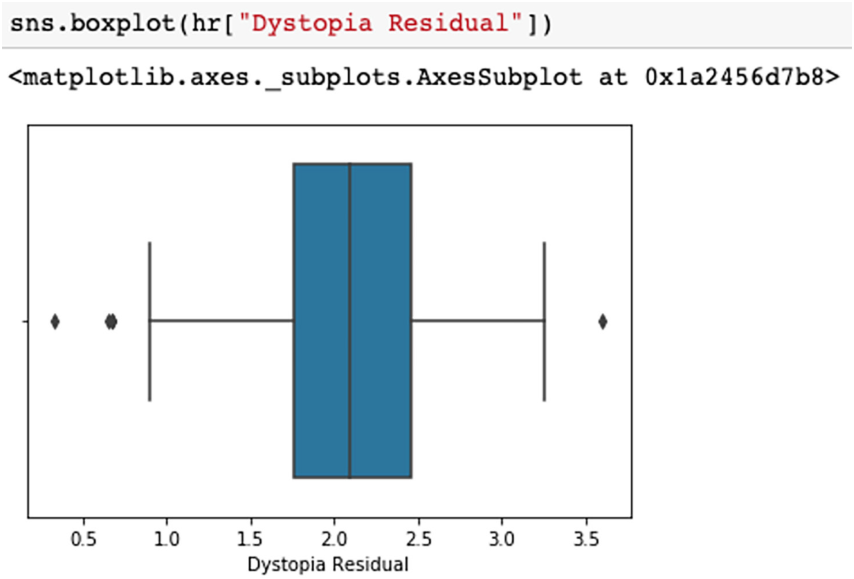
1) Maximum and Minimum is the Highest and Lowest value, respectively, from the given list of data. If we use this method, then we cannot find any outliers Or,

2) Maximum = Q3 + 1.5*IQR and Minimum = Q1 - 1.5*IQR

Where IQR = Q3 - Q1, using this method, we can find the outliers.

With all the above data, we can draw a box-and-whisker plot using the given structure, as shown in *Figure 2.12*.

Below we can see the box plot for the feature, which will be similar to the KDE distribution pattern.



```
sns.boxplot(hr["Dystopia Residual"])
```

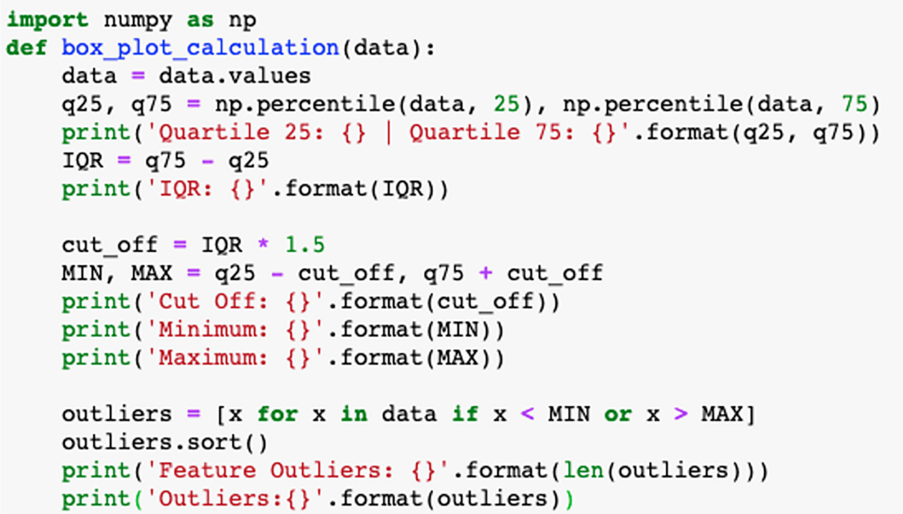
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2456d7b8>
```

Figure 2.13: *Box-plot for a variable*

Seeing the above image, we can say that there are some data points on both left and right sides, which are considered outliers.

Now compare with the normal distribution plot, we can see the maximum density of the values ranges from ~1.8 to ~2.5, and the density decreases as we move away from the mean. At the extreme ends, you can find out the points whose occurrence is rare, i.e., if a random value is chosen then the probability to get the value which will lie in the outliers will be rare and probability to get to the value between the range ~1.8 to ~2.5 is very high.

Now to find the outliers, a function is written that will give us the values of the outliers using the calculation of box-plot. As the box-and-whiskers plot only shows that there are outliers but not the values of the outliers.

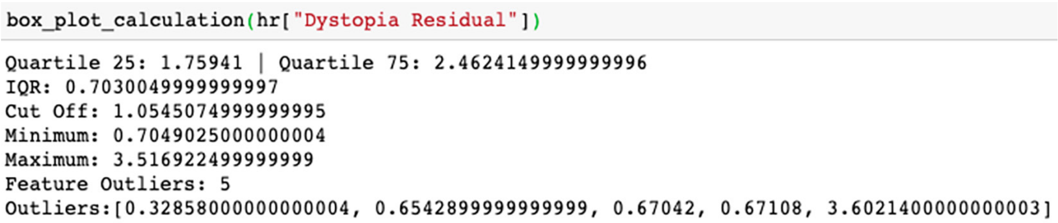


```
import numpy as np
def box_plot_calculation(data):
    data = data.values
    q25, q75 = np.percentile(data, 25), np.percentile(data, 75)
    print('Quartile 25: {} | Quartile 75: {}'.format(q25, q75))
    IQR = q75 - q25
    print('IQR: {}'.format(IQR))

    cut_off = IQR * 1.5
    MIN, MAX = q25 - cut_off, q75 + cut_off
    print('Cut Off: {}'.format(cut_off))
    print('Minimum: {}'.format(MIN))
    print('Maximum: {}'.format(MAX))

    outliers = [x for x in data if x < MIN or x > MAX]
    outliers.sort()
    print('Feature Outliers: {}'.format(len(outliers)))
    print('Outliers:{}'.format(outliers))
```

Figure 2.14: *Box-plot calculation*

The above code will print all the Quartiles, IQR, Minimum, Maximum, and all the outliers.

Let us run this code on the feature "Dystopia Residual" and find out the individual points which are considered outliers.



```
box_plot_calculation(hr["Dystopia Residual"])
```

```
Quartile 25: 1.75941 | Quartile 75: 2.4624149999999996
IQR: 0.7030049999999997
Cut Off: 1.0545074999999995
Minimum: 0.7049025000000004
Maximum: 3.516922499999999
Feature Outliers: 5
Outliers:[0.32858000000000004, 0.6542899999999999, 0.67042, 0.67108, 3.6021400000000003]
```

Figure 2.15: *Box-plot details*

We can see the list of outliers with all the intermediate values to create a Box-plot. In this case, we have five outliers, and four of them are from the left-hand side of the visualization, and one is from the right-hand side.

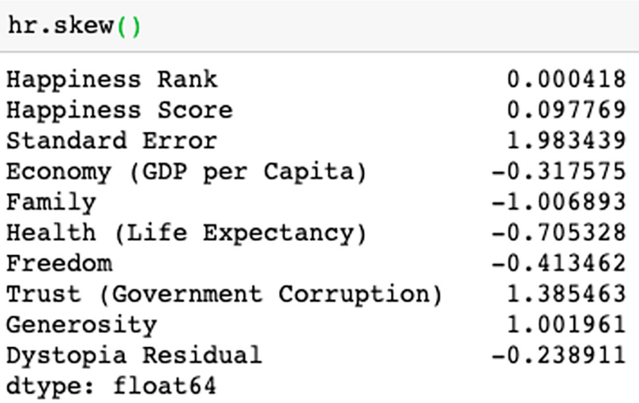
So far, we have seen a histogram, a curve, a box plot, and none of them shows individual data points except the outliers' part of the box-plot. We will use another visualization, i.e., strip plot, where you can see individual data points.

The above strip plot for the feature "Dystopia Residual" is similar to the box plot and the histogram. It tells us a few things which are the same as the other visualizations like the density of the data points, i.e., at what range of values the occurrence of the data is higher, and the places where the occurrence is extremely less.

Skewed distribution

A distribution is said to be a **skewed distribution** when the data points are dense either towards the left or the right side of the curve, thus forming a non-symmetric distribution.

To find the skewness, we can use the third standardized moment that is a measure of skewness.



```
hr.skew()
```

```
Happiness Rank                   0.000418
Happiness Score                  0.097769
Standard Error                   1.983439
Economy (GDP per Capita)        -0.317575
Family                          -1.006893
Health (Life Expectancy)        -0.705328
Freedom                         -0.413462
Trust (Government Corruption)    1.385463
Generosity                       1.001961
Dystopia Residual               -0.238911
dtype: float64
```

Figure 2.16: *Skewness*

The **skewness** is an approximated value, which is a real number. The results are not always accurate, but we can get a rough idea about the skewness. A negative number indicates Left Skewed Distribution, and a positive distribution will represent a Right Skewed Distribution.

We all may have a question about how these values are calculated for skewness. In the next chapters, we will talk about the mathematics behind the skewness calculation.

Left Skewed (Negative Skewness)

Left Skew is sometimes referred to as negative skewness, and to identify a left-skewed distribution; we can observe a long tail at the left side. That indicated less frequency on the left, and the frequency gradually increases towards the right.

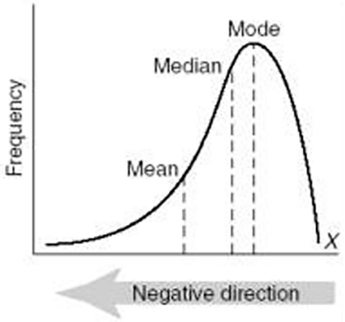


***Figure 2.17:** Structure of a Left Skewed Distribution*

If we look at the structure, then we can see how to Mean, Median, and Mode is in a particular order. This signifies the left-skewed distribution.

As we have seen before, we will follow a similar pattern and see different distribution and the density of the feature and what does it mean.

For a Left Skewed feature, we will go with "Health (Life Expectancy)," but if we explore the dataset, there are a few more left-skewed data like "Family," but I will leave it to you to explore it further.

What is "Health (Life Expectancy)"?

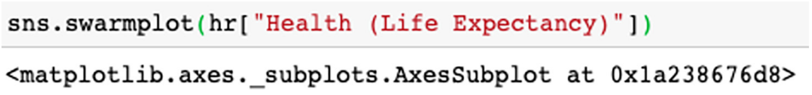
Life Expectancy is a measure of how much a living organism lives. For this dataset, it is the average year people live in the country. Longer someone lives higher, the "Health (Life Expectancy)" will be, and the value will be less if "Health (Life Expectancy)" is lower for a country.

Distribution plot

As we know, the distribution plot is constructed from a binned frequency table. But, from the binned frequency table, we can also generate visualization like strip plot (as shown previously) and swarm plot. For a strip plot, there was an overlap between data points and, if the count of the data points is very high, then it will be tough to visualize the result.

Similarly, if we want that there shouldn't be any overlap between the data points, and we need a similar chart for interpretation, we can go with a swarm plot.

Below we will see how a swarm plot looks and how it is different from the strip plot.



```
sns.swarmplot(hr["Health (Life Expectancy)"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a238676d8>
```

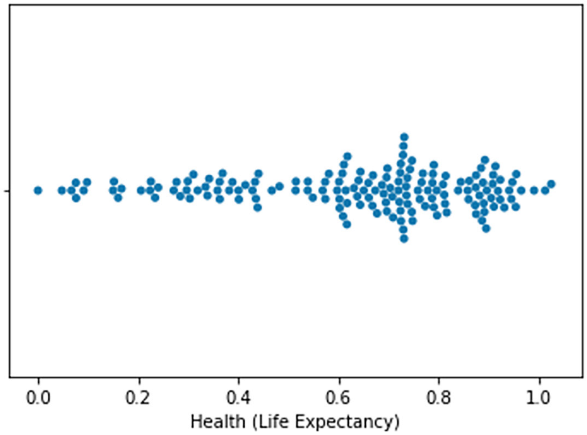
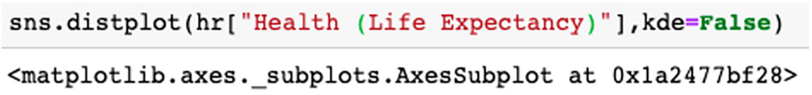


Figure 2.18: *Swarm Plot for Health (Life Expectancy)*

The Swarm plot gives us the same distribution, but the difference is that we can see the individual data points that give us an idea of the density for the data points.

As we are stacking the data and the values are continuous, so one of the best visualizations to see the frequency of the data is through a histogram (as we can see in the below figure).



```
sns.distplot(hr["Health (Life Expectancy)"],kde=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2477bf28>
```

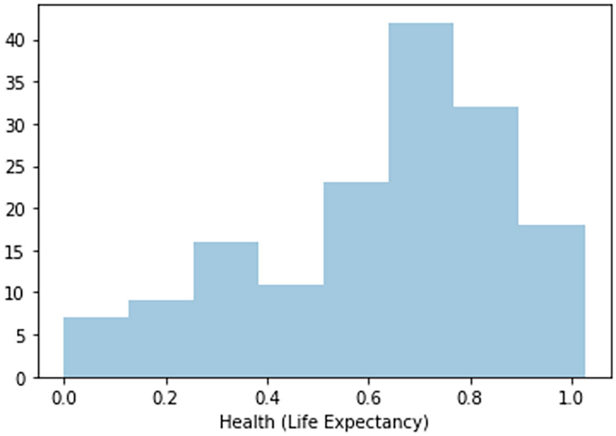


Figure 2.19: *Frequency Distribution Plot*

If we look at this particular histogram, we can see that the pattern is slightly different from the normal distribution, as we have seen earlier. The frequency distribution is more flushed towards the right side of the chart. This is what is unique about the left-skewed distribution.

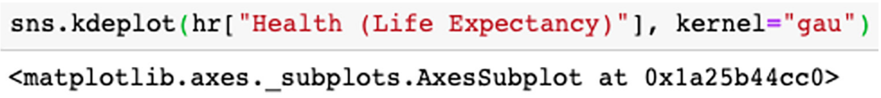
Kernel Density Estimation

We have seen the basics of KDE before, and here we will drive further to see more interesting things about it. In the formula, we have seen something, i.e., Kernel Function

There are multiple kernels which can be used to find out the distribution, for the below example we are using Gaussian Kernel i.e.

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} ; where\ \mu = 0, \sigma^2 = 1$$

For the above Kernel, we can see how it looks for "Health (Life Expectancy)."



```
sns.kdeplot(hr["Health (Life Expectancy)"], kernel="gau")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a25b44cc0>
```

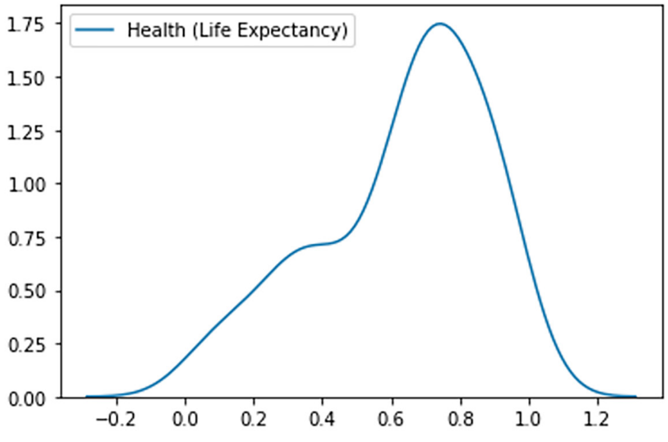
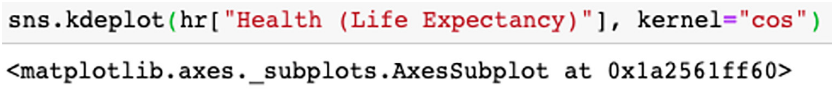


Figure 2.20: *KDE with Gaussian Kernel*

As we can see above, the curve is smooth, and the intricate details for the density are missing. Instead, we can use a Cosine Kernel for the same set of data. Cosine Kernel can be written as:

$$K(x) = \frac{\pi}{4} \cos\left(\frac{\pi}{2} x\right)$$

Similarly, for Cosine Kernel we will see how it looks.



```
sns.kdeplot(hr["Health (Life Expectancy)"], kernel="cos")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2561ff60>
```

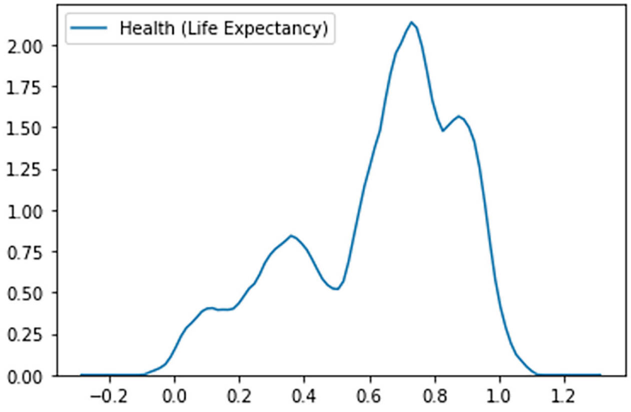
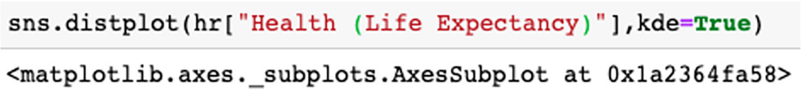


Figure 2.21: *KDE with Cosine Kernel*

We can see above that the Cosine Kernel will give a more accurate density compared to the Gaussian plot. But we will mostly use Gaussian Kernel with the histogram as shown in the below image.



```
sns.distplot(hr["Health (Life Expectancy)"],kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2364fa58>
```

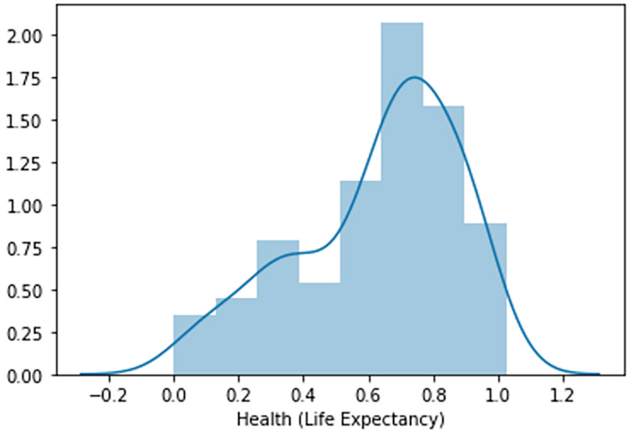
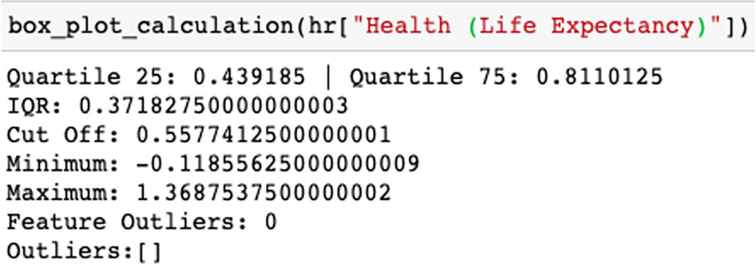


Figure 2.22: *KDE with Histogram for Left Skewed Data*

As we see here, the skewness is not too much, but there is a shift of density towards the right side. We can refer to the skew value from the table (2.16) for all the values and find the skewness value and guess the direction of skewness.

Box plot

We have seen how to compute all the values to generate a box-and-whisker plot in previous distribution. We will perform a similar thing here and see the results.

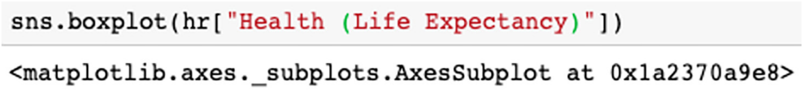


```
box_plot_calculation(hr["Health (Life Expectancy)"])
```

```
Quartile 25: 0.439185 | Quartile 75: 0.8110125
IQR: 0.37182750000000003
Cut Off: 0.5577412500000001
Minimum: -0.11855625000000009
Maximum: 1.3687537500000002
Feature Outliers: 0
Outliers:[]
```

Figure 2.23: *Box-Plot Calculations*

Here we can see that there are no outliers from the output of the function "`box_plot_claculation`" we can verify that result by plotting the box plot. With the above information, we can go ahead and plot a box-and-whiskers plot.



```
sns.boxplot(hr["Health (Life Expectancy)"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2370a9e8>
```

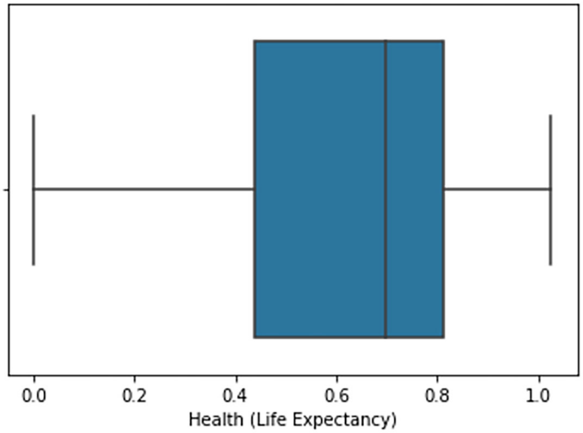


Figure 2.24: *Box-Plot for Health (Life Expectancy)*

We can confirm that there are no outliers, and the shape of the plot is different from the last distribution. The median and mean follows the following structure from *Figure 2.17*.

Right Skewed (Positive Skewness)

Using the skewness table, we figured out that we will use "Trust (Government Corruption)" as the feature from the World Happiness Report.

In the same way, to identify a right-skewed distribution, there is a pattern in which Mode, Median and Mean are arranged (as we can see in the below figure).

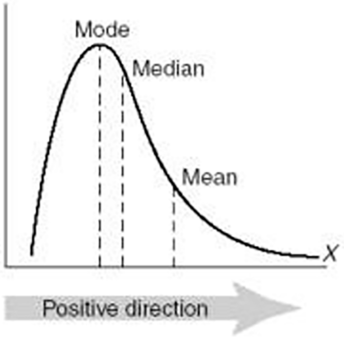


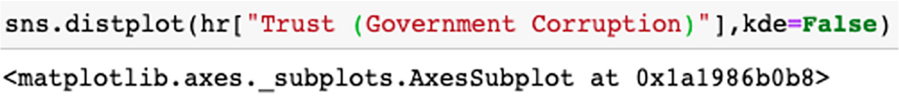
Figure 2.25: *Structure of a Right Skewed Distribution*

Seeing the above structure, we can clearly say that the density of the values is more towards the left side, and there is a long tail at the right end.

Distribution plot

As we have seen the common things so, repeating those visualizations will be monotonous. We will just show the visualization that makes the difference. It is advised to implement all the visualizations to get proper insight.

Let us start with the basic histogram distribution plot.



```
sns.distplot(hr["Trust (Government Corruption)"],kde=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1986b0b8>
```

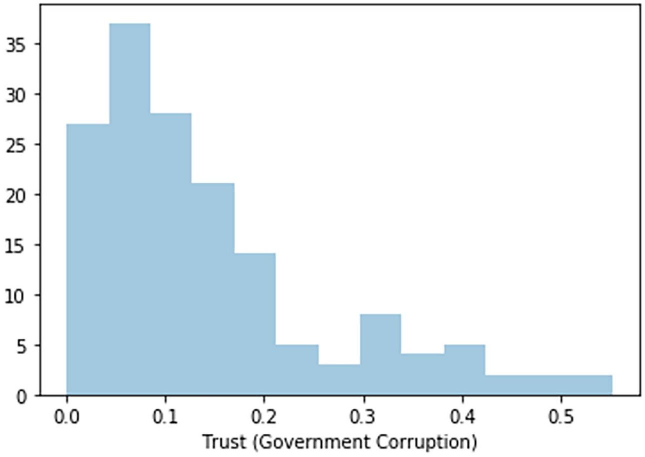


Figure 2.26: *Distribution Plot*

We can see that the density of the data points is concentrated towards the left side of the distribution, and we have a tail on the right side. This follows the structure of right skewness, as shown in *Figure 2.25*.

Kernel Density Estimation

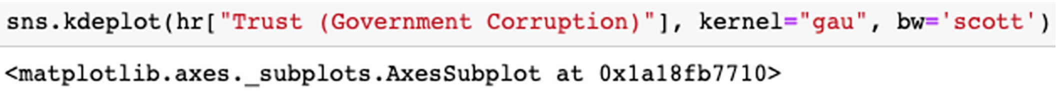
Previously we have seen the importance of bandwidth and different kernels like Gaussian Kernel and Cosine Kernel.

Here we will play with a free parameter[16] for KDE, i.e., bandwidth. Bandwidth determines the smoothness of the curve. The lower value of bandwidth determines a higher number of bins so that the curve be more accurate and will give us more details.

Let us start by seeing the default bandwidth estimator with Gaussian Kernel.

By default, it uses "`scott`"[17] to estimate the bandwidth (*h*), and it is one of the most popular methods and another popular method being "`silverman`".

Below we see the result with the "`scott`" estimator.



```
sns.kdeplot(hr["Trust (Government Corruption)"], kernel="gau", bw='scott')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a18fb7710>
```

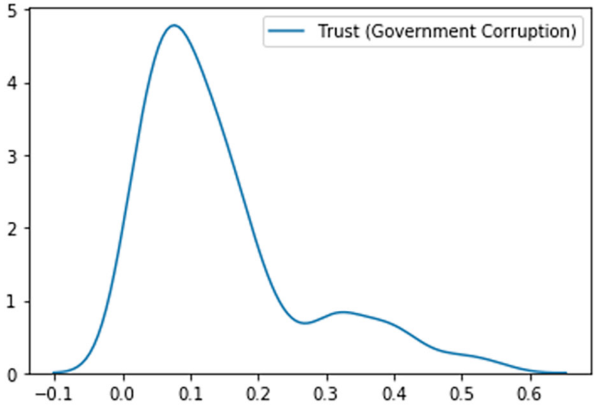
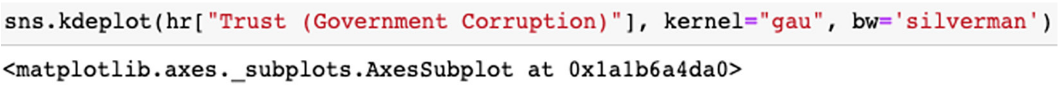


Figure 2.27: *Gaussian Kernel with "scott"*

"`scott`" and "`silverman`" are similar, and results are similar as well. We can also take a look at "`silverman`" bandwidth estimator. Similarly, we can visualize for "`silverman`" below.

[16] "Free parameter - Wikipedia." https://en.wikipedia.org/wiki/Free_parameter.

[17] "On optimal and data-based histograms"- https://doi.org/10.1093/biomet/66.3.605



```
sns.kdeplot(hr["Trust (Government Corruption)"], kernel="gau", bw='silverman')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1b6a4da0>
```

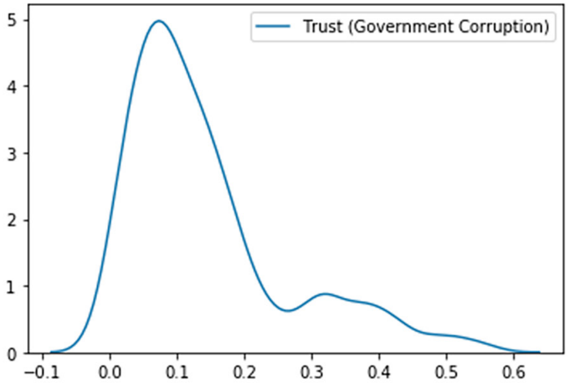
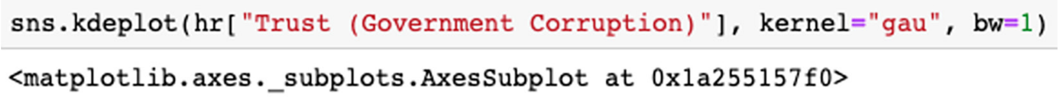


Figure 2.28: *Gaussian Kernel with "silverman"*

"`scott`" and "`silverman`" calculates the optimal value of the bandwidth (we will take a look at how it is calculated later this section) that gives us the best distribution but to get a feel for how bandwidth affects the plot we need to give scalars and see the result.

If the value of the bandwidth is high like *bandwidth = 1 or 2*, then it over-smooths the curve, and we won't get any information from it, this is like we have a binned frequency table with only one bin. Let us confirm the hypothesis but plotting it.



```
sns.kdeplot(hr["Trust (Government Corruption)"], kernel="gau", bw=1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a255157f0>
```

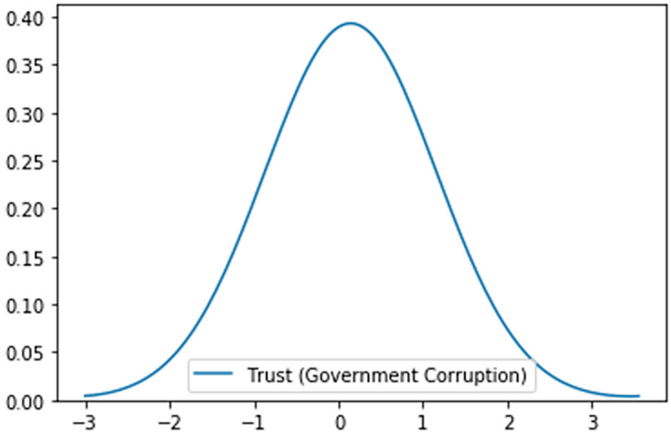
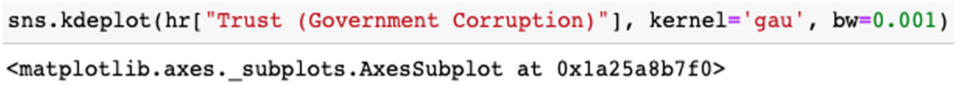


Figure 2.29: *KDE with Bandwidth=1*

It will be a similar problem if *bandwidth* = 0.001 then the problem of under-smoothing will come into the plot like the figure below.



```
sns.kdeplot(hr["Trust (Government Corruption)"], kernel='gau', bw=0.001)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a25a8b7f0>
```

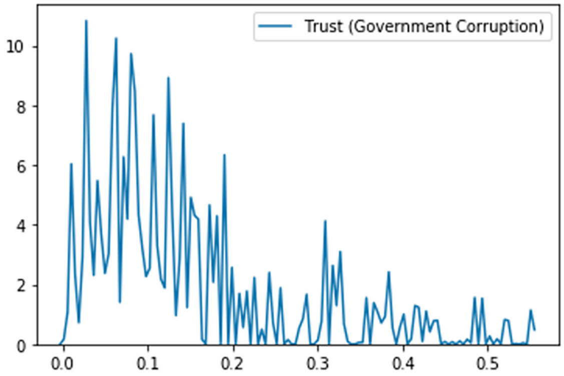


Figure 2.30: *KDE with Bandwidth=0.001*

This problem occurs if the total number of bins is too high, then it will tend to show all the ups and downs. So, we need to strive for the optimal value of bandwidth, which will select the right number of bins to plot the histogram and then smooth the histogram.

Below, we will briefly see the common optimal way to select the parameter Bandwidth using **Silverman's rule of thumb** approach.

We can use L_2 Risk Function(i).

$$\Rightarrow E \| f_h - f \|_2^2 \qquad \ldots(i)$$

Where E is the expected value[18], f is the unknown density and f_h is an estimate based on a sample of h. Here f is from the Kernel Density Estimation.

The above equation (i) is also known as **Mean Integrated Squared Error (MISE)**[19]. As this is a Density problem, we are using integrals.

$$MISE(h) = E\left[\int \left(f_h - f\right)^2 dx\right] \qquad \ldots(ii)$$

MISE is similar to **Mean Squared Error** (**MSE**) with an integral. We will see more about MSE in later chapters.

[18] "Expected value - Wikipedia." https://en.wikipedia.org/wiki/Expected_value.

[19] "Mean integrated squared error - Wikipedia."
https://en.wikipedia.org/wiki/Mean_integrated_squared_error.

If Gaussian Basis Function[20] is used to approximate and the underlying density is a Gaussian, then the optimal choice for h (i.e., the bandwidth that minimizes/reduces the MISE) is given by the general formula:

$$h = \left(\frac{4\hat{\sigma}^5}{3n} \right)^{\frac{1}{5}}$$

Here, h is not a good fit for long tails and skewed distribution as it is optimized for Gaussian Distribution.

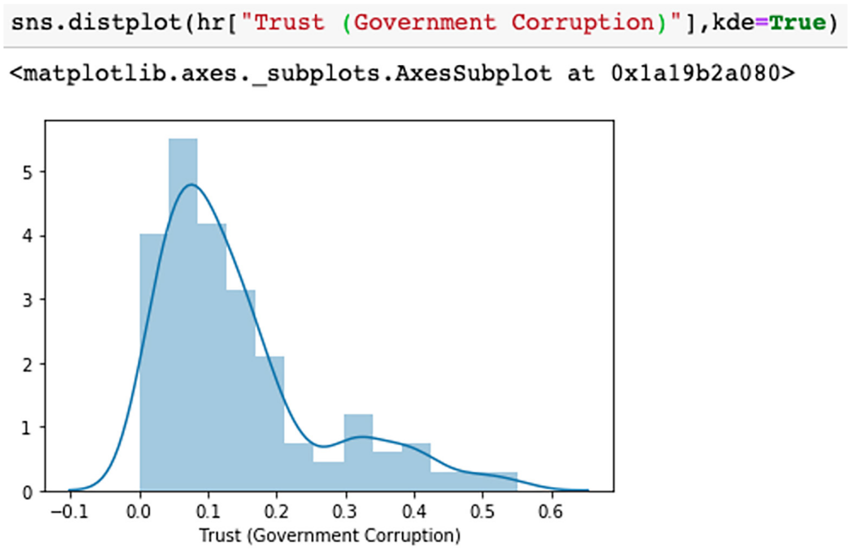
From the general formula, some changes were made to make h more robust. So, the new value of h after considering the drawbacks

$$h = 0.9 * \min\left(\hat{\sigma}, \frac{IQR}{1.34} \right) * n^{-\frac{1}{5}}$$

The above formula is known as Gaussian Approximation or Silverman's rule of thumb.

If someone is interested, they can read about Parzen Density Estimation[21], which was the inception for Probability Density Estimation.

If we overlay the optimal curve from KDE using the Scott estimator, this will look something like the below figure.



```
sns.distplot(hr["Trust (Government Corruption)"],kde=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a19b2a080>
```

Figure 2.31: *KDE with Default bandwidth estimator*

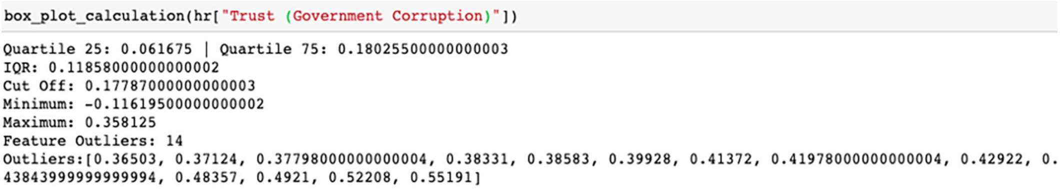
[20] "Smoothing Techniques using basis functions: Gaussian Basis ...." 21 Oct. 2015, https://datascienceplus.com/smoothing-techniques-using-basis-functions-gaussian-basis/.

[21] "On Estimation of a Probability Density Function and Mode." 23 Feb. 2000, http://bayes.wustl.edu/Manual/parzen62.pdf.

Seeing the above, we can say that the same pattern follows, and it is similar to the distribution of the data.

Box plot

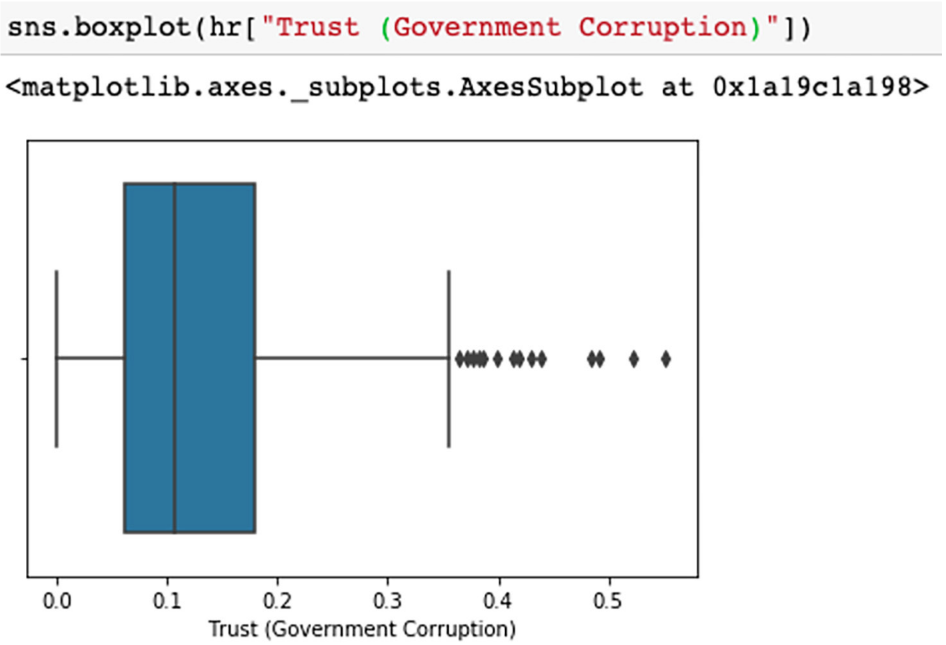
We had written a code snippet before, and to find out the values related to the Box-Plot to create it, let us execute that and see all the values.



```
box_plot_calculation(hr["Trust (Government Corruption)"])

Quartile 25: 0.061675 | Quartile 75: 0.18025500000000003
IQR: 0.11858000000000002
Cut Off: 0.17787000000000003
Minimum: -0.11619500000000002
Maximum: 0.358125
Feature Outliers: 14
Outliers:[0.36503, 0.37124, 0.37798000000000004, 0.38331, 0.38583, 0.39928, 0.41372, 0.41978000000000004, 0.42922, 0.
43843999999999994, 0.48357, 0.4921, 0.52208, 0.55191]
```

Figure 2.32: Box Plot Calculation for Right Skewed Data

With all the derived details, we plotted the box plot, and we can see quite a lot of outliers, and all the outliers are above the “Maximum” so, we are expecting to see all the outliers on the right-hand side of the plot.



```
sns.boxplot(hr["Trust (Government Corruption)"])

<matplotlib.axes._subplots.AxesSubplot at 0x1a19c1a198>
```

Figure 2.33: Box Plot Trust (Govt. Corruption)

As we have anticipated from the output of the code snippet that it will be a right-skewed, and the outliers will be on the right side of the plot.

In this section, we have mostly seen major types of probability distribution and how skewness will affect them. We have also seen what kind of information we get from each kind of graph.

Now we need to see a few more types of distributions, which will be common when we perform data analysis. Unfortunately, we cannot show an example from the dataset we are using as it does not have that kind of distribution. So will give the structure and what does it mean in this section.

Other distributions

Hundreds of probability distribution has different properties and conveys a different message. Here is a list of some of the distributions, and it is advised that to go through all of them and understand the places where its implemented.

1. Uniform Distribution[22]
2. Binomial Distribution[23]
3. Bernoulli Distribution[24]
4. Poisson Distribution[25]
5. Multimodal Distribution[26]
6. Student’s t-Distribution[27] [28]

The more one knows, the better it is. And it is advised to know the pattern of distributions and its properties because it will help to find the same property in the data as the distribution.

Further reading

- Marginal probability
- Random variables
- Expectation, Variance, and Covariance
- Central Limit Theorem
- Probability Distributions

[22] “Uniform distribution (continuous) - Wikipedia.” https://en.wikipedia.org/wiki/Uniform_distribution_(continuous).

[23] “Binomial distribution - Wikipedia.” https://en.wikipedia.org/wiki/Binomial_distribution.

[24] “Bernoulli distribution - Wikipedia.” https://en.wikipedia.org/wiki/Bernoulli_distribution.

[25] “Poisson distribution - Wikipedia.” https://en.wikipedia.org/wiki/Poisson_distribution.

[26] “Multimodal distribution - Wikipedia.” https://en.wikipedia.org/wiki/Multimodal_distribution.

[27] “Student’s t-distribution - Wikipedia.” https://en.wikipedia.org/wiki/Student%27s_t-distribution.

[28] “Student’s t-test - Wikipedia.” https://en.wikipedia.org/wiki/Student%27s_t-test.

- Statistical Inference
- Quantile Statistics

Conclusion

This chapter was mostly about different kinds of distribution, density estimations, and various other plots. We have seen quite a lot of mathematical formulas and equations, which will help us to understand a topic in more detail.

As we know, Data Analysis is 60% to 70% of the entire pipeline. So far, the concepts we have covered will enable you to handle varied types of data. Next chapter onwards, we will see concepts more aligned to Machine Learning. But all the concepts will be used again and again to analyze the data.

This is the last chapter for data analysis, and from the next chapter, we will dive into Machine Learning models. Mostly we will start with popular datasets and classification or regression problems, but slowly, the dataset and the complexity of the problem will increase.

Chapter 3
Iris Species

Introduction

This chapter onwards, we see how Machine Learning algorithms are implemented and evaluated. Here, we won't be making or modeling the best machine learning model with the highest accuracy. But we will try to cover a wide variety of algorithms and techniques that will help us tackle a different kind of datasets and situations.

In this chapter, we will start implementing concepts of machine learning, the mathematics behind it, how to choose the right algorithm, etc. We will mostly cover the basics algorithms for classification, regression, and clustering. This chapter will give us a basic foundation and intuition to handle this classification, regression, and clustering problems. Whenever we encounter a similar type of dataset, the flow and the way we will handle the dataset will remain mostly similar. Still, the algorithm can change as per the requirement and type of dataset.

Structure

- Intro to Machine Learning
- Understanding the dataset
- Exploratory Data Analysis

- How to choose ML algorithms?
- Classification
- Clustering
- Regression

Objective

By the end of this chapter, we will understand what machine learning is, different types of machine learning along with some of its important algorithms. We will mainly focus on different types of algorithms and where it is implemented and, not on the accuracy of the model.

Introduction to Machine Learning

Machine Learning (ML) term was coined by Arthur Samuel (1959), he is an American pioneer in the field of Computer Gaming and Artificial Intelligence (AI) and stated that "it gives computers the ability to learn without being explicitly programmed."

And in 1997, Tom Mitchell gave a "well-posed" mathematical and relational definition that "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

What is Machine Learning[1]?

Machine Learning is the field of study where algorithms and statistical models automatically learn and improve from the data with underlying patterns and inference, without explicit instruction programmed.

It is extremely common to confuse between similar terms like Machine Learning Artificial Intelligence and Deep Learning.

Below you can see a Venn diagram[2] that clearly explains the difference between them.

[1] "Machine learning - Wikipedia." https://en.wikipedia.org/wiki/Machine_learning.

[2] "Venn Diagram Definition - Investopedia." 4 Oct. 2019, https://www.investopedia.com/terms/v/venn-diagram.asp.

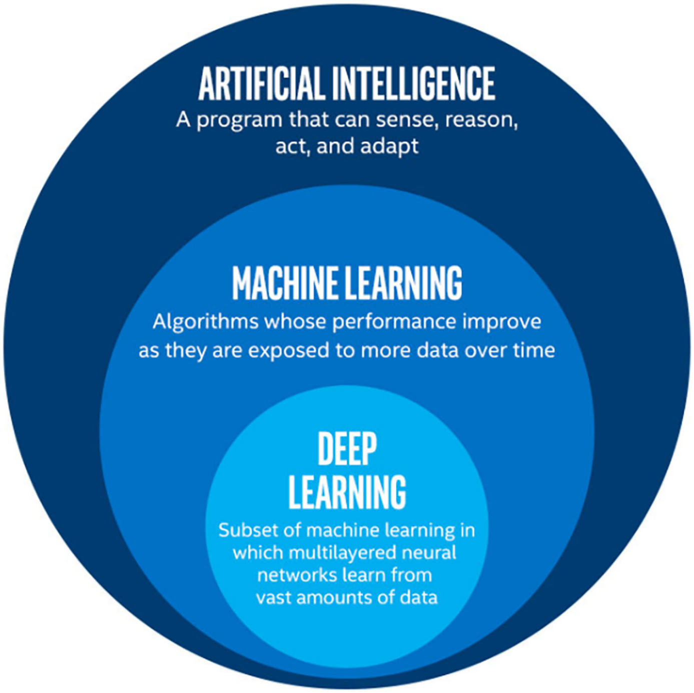

ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

Figure 3.1: *AI Venn diagram (Source: Medium)*

We can see AI is the main field, i.e., it is the superset, then under that, we have Machine Learning (ML), and under ML, we have Deep Learning. This book will cover the basics of ML and different algorithms related to it.

Machine Learning can be classified into multiple types. We will explain some of the types below.

Types of Machine Learning

Machine Learning in a broad field of study and it is divided into multiple types depending upon some parameters like:

1. Type of input data (Labeled or Unlabeled data)
2. The output data type (continuous or categorical)

Supervised Learning[3]

Supervised learning is a type of Machine Learning class that learns from the labeled data. Labeled data[4] is a kind of Data which has both Input and Output parameters. Classification and regression are the major types of Supervised Learning problem.

Unsupervised Learning[5]

Unsupervised learning is a type of ML class that learns from the unlabeled data. Unlabeled data is a kind of data that has no output parameter. Clustering, Anomaly Detection are some Unsupervised Learning problem. There is another class of ML, which is similar to supervised and unsupervised learning, is known as **Semi-Supervised Learning**[6].

Reinforcement Learning[7]

Reinforcement Learning is a type of Machine Learning where the algorithms can learn from the environment and maximize the reward. Some of the well-known algorithms for Reinforcement Learning are Q-Learning[8], SARSA[9], DQN, etc.

Machine Learning pipeline

Machine Learning has its separate pipeline, and it's a general pipeline for most of the projects. This pipeline is not followed word for word, but it is a general structure. Every Machine Learning practitioner follows a modified version of a similar pipeline, as stated below.

[3] "Supervised learning - Wikipedia." https://en.wikipedia.org/wiki/Supervised_learning.

[4] "Labeled data - Wikipedia." https://en.wikipedia.org/wiki/Labeled_data.

[5] "Unsupervised learning - Wikipedia." https://en.wikipedia.org/wiki/Unsupervised_learning.

[6] "Semi-supervised learning - Wikipedia." https://en.wikipedia.org/wiki/Semi-supervised_learning.

[7] "Reinforcement learning - Wikipedia." https://en.wikipedia.org/wiki/Reinforcement_learning.

[8] "Q-learning - Wikipedia." https://en.wikipedia.org/wiki/Q-learning.

[9] "State–action–reward–state–action - Wikipedia." https://en.wikipedia.org/wiki/State%E2%80%93action%E2%80%93reward%E2%80%93state%E2%80%93action.

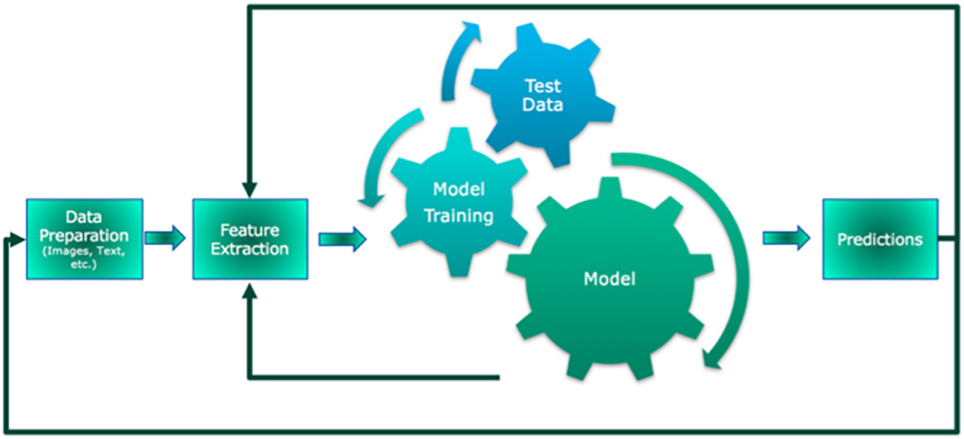


Figure 3.2: Machine Learning Pipeline (Source: datanami)

A brief explanation for all the components from the above is given as follows:

- **Data Preparation**: Data Preparation is the process by which we can make the data ready for training, testing, and validating the ML model.
- **Feature Extraction**: Feature extraction is the process by which we select, modify, or mutate the right features for training the model. There can be few features that have a contribution towards the output (which we will predict using the model), and some features have no direct or indirect contribution towards the output variable. From this step, we will select the features which are not required.
- **Model**: Machine Learning model is a mathematical representation learned/ formed from the pattern of the data, which is given as an input for training the model.
- **Model Training**: Model Training is a process to learn the underlying pattern from the data with the given output feature (for supervised learning).
- **Test Data**: This is a subset of the entire dataset. This data is not given as an input to the model training phase. This data will be used to find the accuracy of the model for the unseen data.
- **Prediction**: Prediction is the process; we generally use the test/unseen data to see how good the model is being performed.
- **Evaluation**: This is the step where we can see the quality of the model using different metrics such as Accuracy, Recall, Precision, etc.

> *Note:* **Machine Learning pipeline** is an iterative model where we have to go back-and-forth to come up with an optimal model for any kind of Machine Learning model.

Understanding the dataset

Iris Species[10] of a flower is one of the most popular datasets for Machine Learning. Iris Flower dataset is from the UCI Machine Learning Repository[11] , among other popular, well-known datasets.

Iris dataset contains data for 3 species namely Iris Setosa (a.k.a. setosa), Iris versicolor (a.k.a. versicolor), Iris virginica (a.k.a. virginica) and each species contain 50 records.

This dataset is primarily for classification, but here we will use this same dataset with some changes to it so that we can use it for different types of Machine Learning problems.

The dataset contains only four main features, i.e., sepal length, sepal width, petal length, and petal width.

In the below image which shows the beautiful iris species and what does the feature exactly mean.

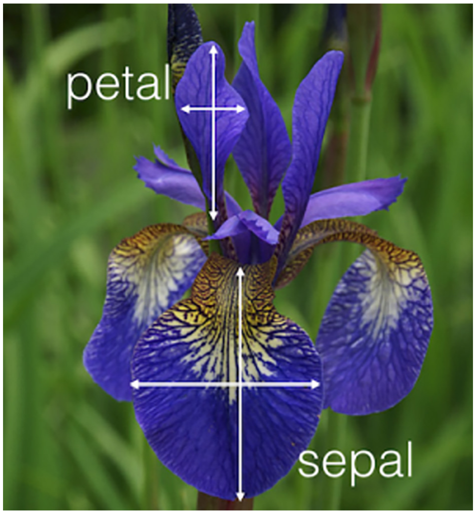


Figure 3.3: *Structure of the flower (Source: Kaggle)*

Seeing the image now, we can visualize all the features in the dataset. This won't be easy for all the available datasets. But I will recommend everyone to get some domain knowledge and learn about the data before starting the Machine Learning work.

[10] "Iris Data Set - UCI Machine Learning Repository." 1 Jul. 1988, https://archive.ics.uci.edu/ml/datasets/Iris.

[11] "Data Sets - UCI Machine Learning Repository." https://archive.ics.uci.edu/ml/datasets.php.

Exploratory Data Analysis

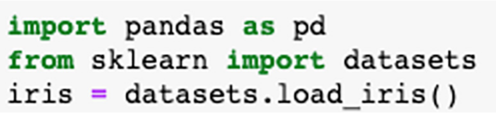
We will start by loading the dataset and see the pattern of the distribution, as seen earlier in the previous chapters.

Load dataset

Iris Species is an extremely popular dataset, and that can be either used from a scikit-learn or seaborn library.

Scikit-learn

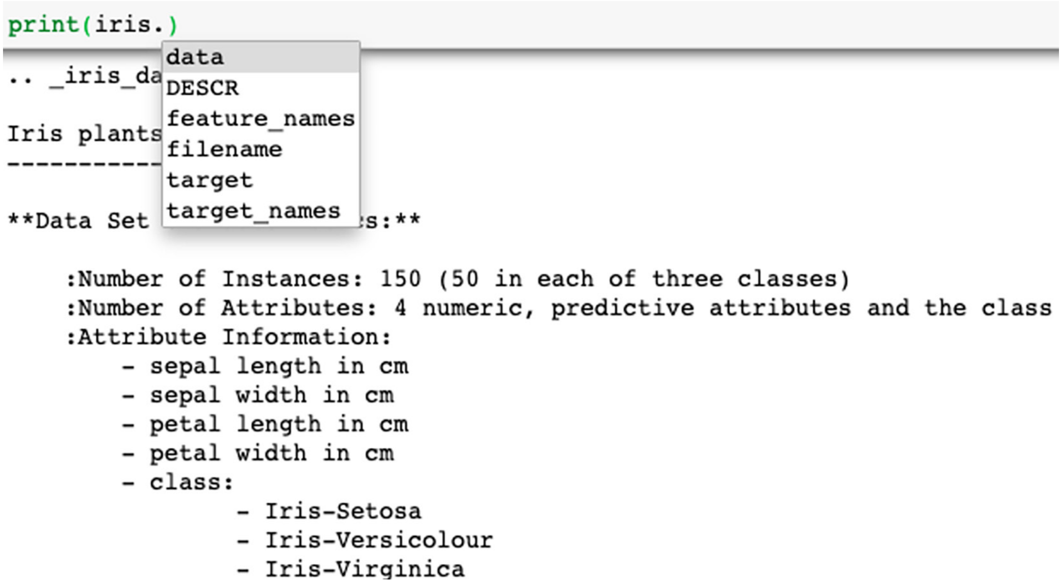
For the first scenario, we will load the dataset from sklearn and see how to make the dataset useful for later phases. This step comes under the data obtaining for O.S.E.M.N. Framework. So, let us start by loading the dataset from the sklearn.



```
import pandas as pd
from sklearn import datasets
iris = datasets.load_iris()
```

Figure 3.4: *Load Dataset from sklearn*

Now, if we try to see the contents of the variable "`iris`", then we will see that it is not in a tabular structure, and it has more information than the data.



```
print(iris.)
```

```
.. _iris_da

Iris plants
-----------

**Data Set                 :s:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
```

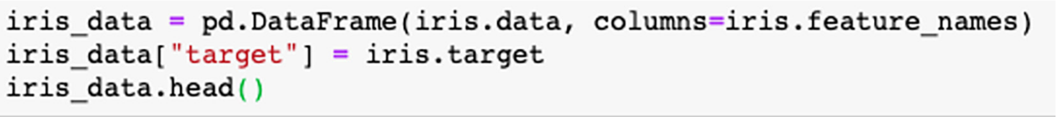
Figure 3.5: *Content of Iris variable*

Above, we can see some parts of the "DESCR," and other information like "data," "`feature_names`," "`target`," etc.

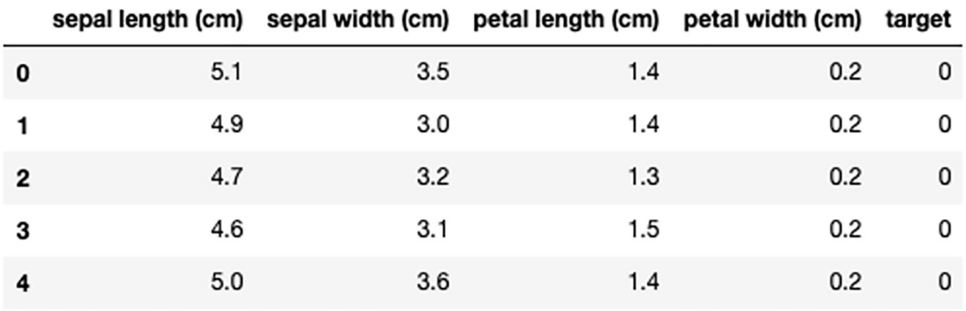
From all the different information, we will only use the "`data`," "`feature_names`," "`target`," "`target_names`."

We will see how we can combine all the data and form a table with input and output features.

Let us see some of the first rows of the data set after loading it from sklearn.



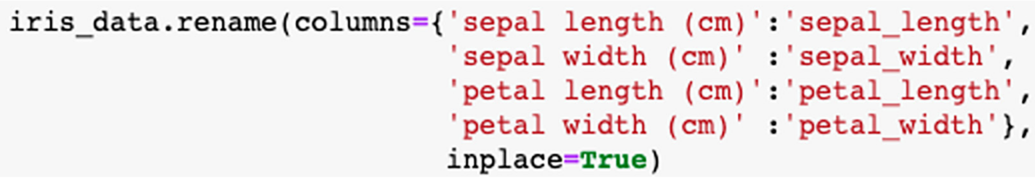
```
iris_data = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_data["target"] = iris.target
iris_data.head()
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

***Figure 3.6:** Basic Data table*

Seeing the above image, we can observe that some irregularities, so some minor changes are required. One, the name of the columns is not as per standards, and it is very important to maintain the naming standard for ease of coding and readability.

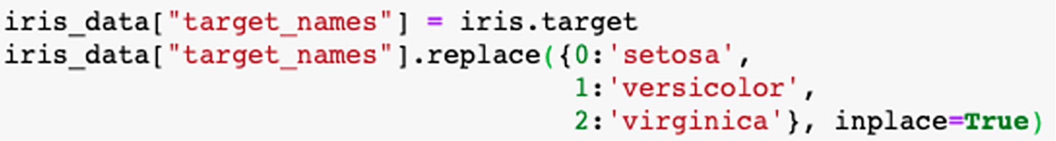


```
iris_data.rename(columns={'sepal length (cm)':'sepal_length',
                          'sepal width (cm)' :'sepal_width',
                          'petal length (cm)':'petal_length',
                          'petal width (cm)' :'petal_width'},
                          inplace=True)
```

***Figure 3.7:** Renaming Columns*

After changing the name of the columns now, it will be simple to use it in the code. We have another problem, i.e., for the target column, we are using integers to represent different categories.

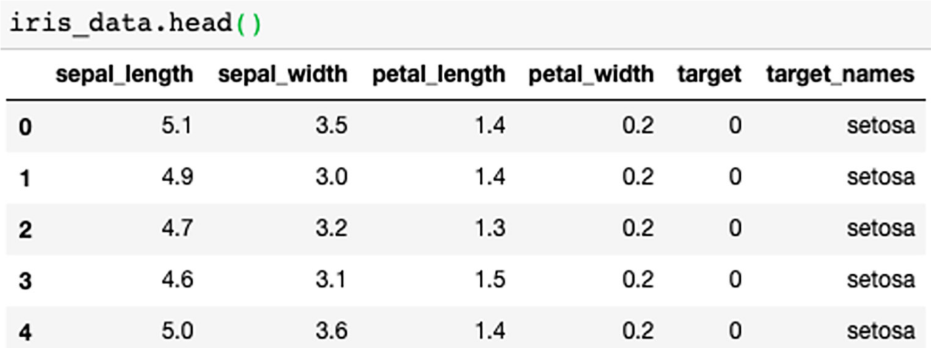
We can add one more column that gives us the name of the species.



```
iris_data["target_names"] = iris.target
iris_data["target_names"].replace({0:'setosa',
                                   1:'versicolor',
                                   2:'virginica'}, inplace=True)
```

Figure 3.8: *Adding Species name*

Having the name of the species will help in the data visualization. It should be noted that most machine learning models don't take a string as input. Some models take in string data as an input, but internally, it encodes the data to a numerical data type for model training. After completing the steps, let us see how the data looks.



```
iris_data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	target	target_names
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Figure 3.9: *Iris Data Table*

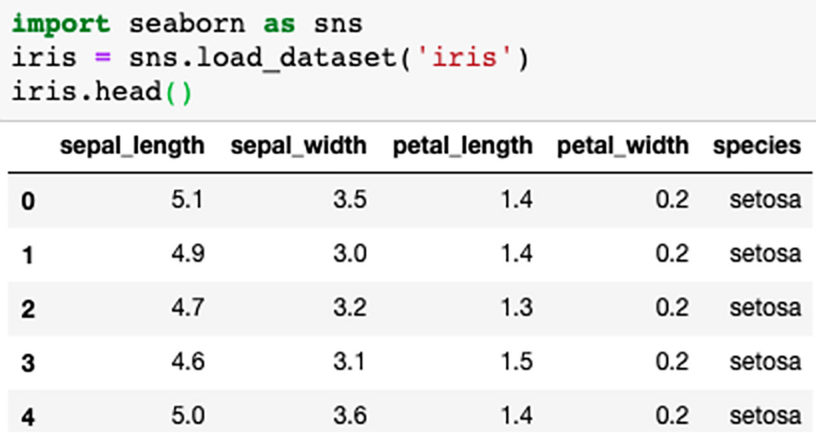
As we have seen, we have mostly covered the basic data preparation for the next step, which is data visualization. But before moving forward, we will see another source where we can obtain the same dataset form.

Seaborn

Seaborn is a data visualization library that we had seen earlier, and it has a limited set of datasets[12] [13]. The code snippet below is for loading and visualizing the data from Seaborn.

[12] "seaborn.load_dataset — seaborn 0.9.0 documentation." https://seaborn.pydata.org/generated/seaborn.load_dataset.html.

[13] "mwaskom/seaborn-data - GitHub." https://github.com/mwaskom/seaborn-data.



```python
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Figure 3.10: *Iris Dataset from Seaborn*

The above data is similar to sklearn data, and with minimal processing, we can make our dataset.

Here in this chapter, we will use the data from the sklearn iris dataset.

Data Analysis

We will see some common techniques for feature analysis, and this will help to select the basic features which will be used for classification and clustering.

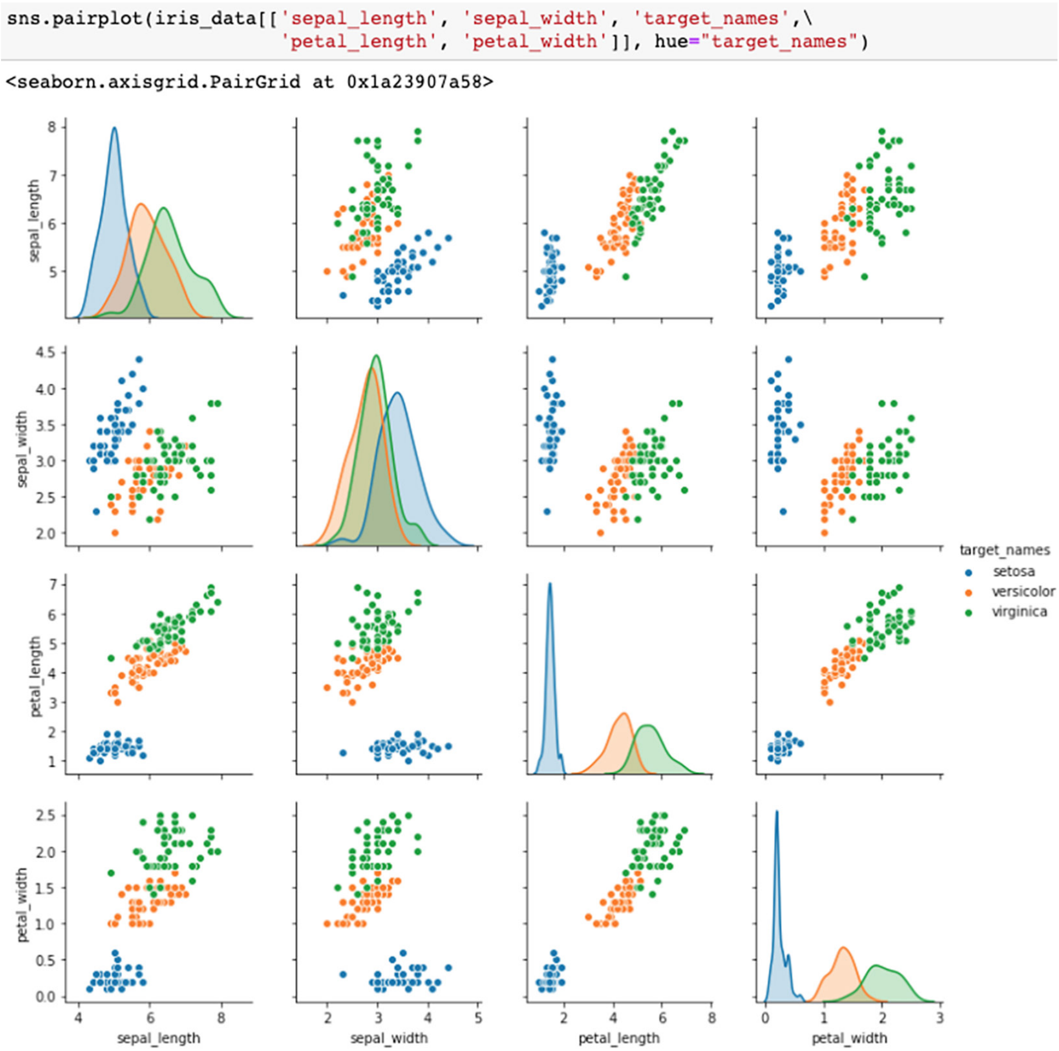
We can use a pair plot to visualize all the features at once and the relationship with each other.

Pair plot

Pair plot enables us to visualize the distribution of the single variable and relation with all the variables.

> ***Note:*** We should keep in mind that the pair plot is an extremely computationally heavy operation. Iris species dataset is a small dataset, and the number of features is too less for computation.

Let us see the pair plot for the dataset and analyze them later.



```
sns.pairplot(iris_data[['sepal_length', 'sepal_width', 'target_names',\
                        'petal_length', 'petal_width']], hue="target_names")
```

```
<seaborn.axisgrid.PairGrid at 0x1a23907a58>
```

Figure 3.11: Pair Plot for entire dataset

Seeing the pair plot, we can make interesting conclusions about the spread of the dataset and its different features.

Observations

Mostly for all the images, "`setosa`" is separated from other species of iris. Even if we look at all the distribution, the overlap between the distribution is quite less.

But when it comes to the feature "`sepal_width`" the overlap of the distribution of 3 different types of the iris is very high. This is not a sign of good feature which will classify different species.

Talking about other features, we can see that "`versicolor`" and "`virginica`" are very close to each other, and with the feature "`sepal_width`" the overlap is high as expected.

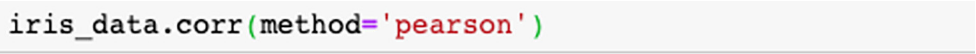
"`sepal_length`" vs "`sepal_width`" is the worst feature descriptor for identifying the species "`versicolor`" and "`virginica`."

To confirm the above hypothesis, we can validate it by finding the correlation coefficient.

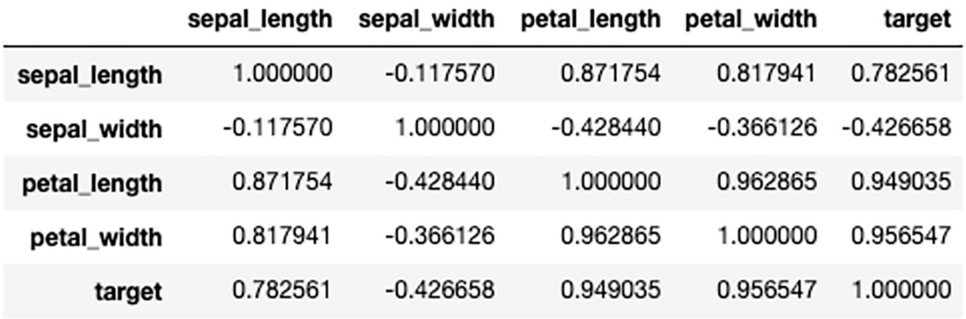
Correlation[14]

Correlation is a technique by which we can find out how strong the features are related. There are multiple methods to find the correlation value of two variables, but the most popular ones are '`pearson`,' '`kendall`', '`spearman`.'

Below we will see the pearson correlation coefficient for iris data.



```
iris_data.corr(method='pearson')
```



	sepal_length	sepal_width	petal_length	petal_width	target
sepal_length	1.000000	-0.117570	0.871754	0.817941	0.782561
sepal_width	-0.117570	1.000000	-0.428440	-0.366126	-0.426658
petal_length	0.871754	-0.428440	1.000000	0.962865	0.949035
petal_width	0.817941	-0.366126	0.962865	1.000000	0.956547
target	0.782561	-0.426658	0.949035	0.956547	1.000000

Figure 3.12: *Pearson correlation coefficient*[15]

The correlation coefficient values range from -1 to 1. Maximum Negative value(-1) implies a stronger negative correlation, and Maximum Positive Value(1) implies a stronger positive correlation. Whereas, values closer to zero(0) indicate a weaker correlation (exact 0 implying no correlation). Roughly, we can say when the correlation is one then two features/variables are linearly dependable[16], i.e. if at least one of the vectors in the set can be defined as a linear combination of the others.

[14] "Correlation - The Survey System." https://www.surveysystem.com/correlation.htm.

[15] "Pearson correlation coefficient - Wikipedia." https://en.wikipedia.org/wiki/Pearson_correlation_coefficient.

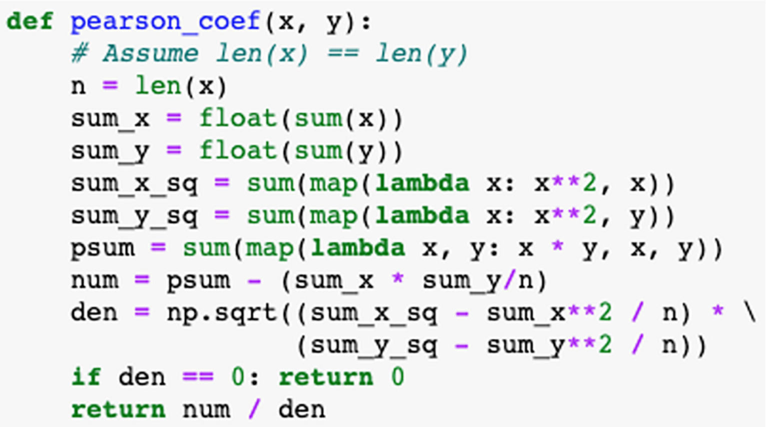
[16] "Linear independence - Wikipedia." https://en.wikipedia.org/wiki/Linear_independence

Let us take a look at how these values are calculated using the Pearson correlation coefficient.

$$r_{x,y} = \frac{\sum\left(x-\overline{x}\right)\left(y-\overline{y}\right)}{\sqrt{\sum\left(x-\overline{x}\right)^2 \cdot \sum\left(y-\overline{y}\right)^2}}$$

Where $\overline{x}$ and $\overline{y}$ are the statistical mean of the features.

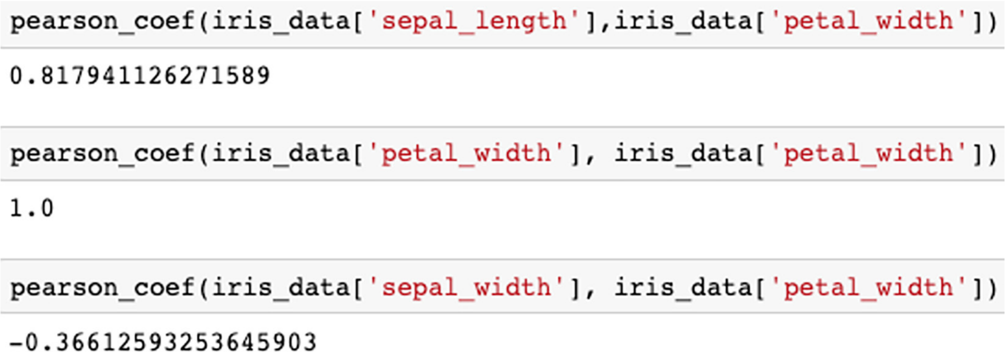
We can implement the same above formula and validate the results.



```python
def pearson_coef(x, y):
    # Assume len(x) == len(y)
    n = len(x)
    sum_x = float(sum(x))
    sum_y = float(sum(y))
    sum_x_sq = sum(map(lambda x: x**2, x))
    sum_y_sq = sum(map(lambda x: x**2, y))
    psum = sum(map(lambda x, y: x * y, x, y))
    num = psum - (sum_x * sum_y/n)
    den = np.sqrt((sum_x_sq - sum_x**2 / n) * \
                  (sum_y_sq - sum_y**2 / n))
    if den == 0: return 0
    return num / den
```

Figure 3.13: *Python Implementation of Pearson's Correlation*

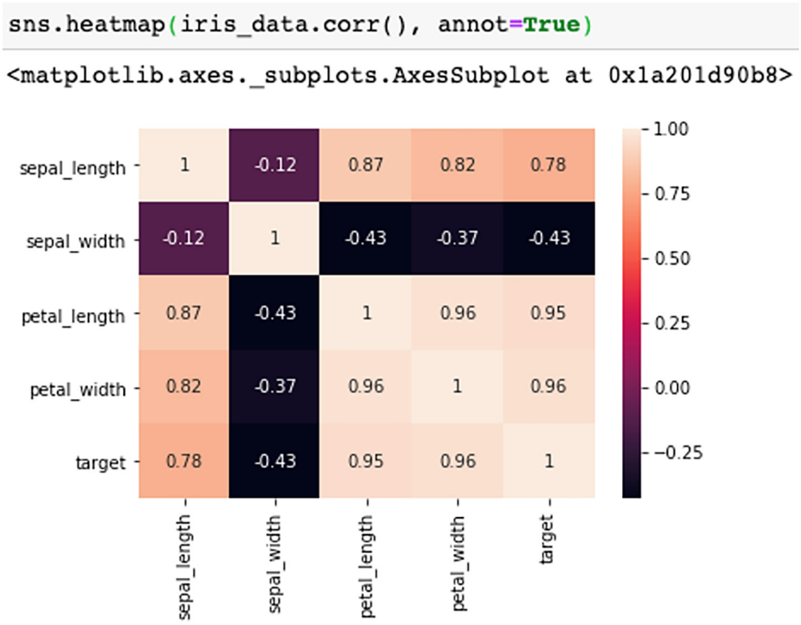
Verifying some of the results with *Figure 3.12: Pearson Correlation*.



```
pearson_coef(iris_data['sepal_length'],iris_data['petal_width'])

0.817941126271589

pearson_coef(iris_data['petal_width'], iris_data['petal_width'])

1.0

pearson_coef(iris_data['sepal_width'], iris_data['petal_width'])

-0.36612593253645903
```

Figure 3.14: *Testing to validate the code*

From the above correlation table, we can plot a heatmap. Heatmap will visually give us the same information.



```
sns.heatmap(iris_data.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a201d90b8>
```

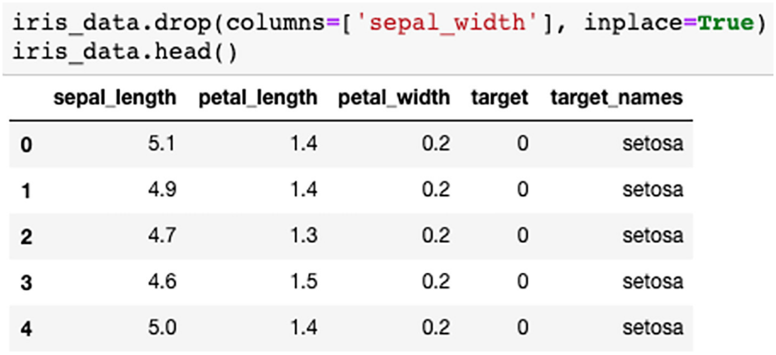
	sepal_length	sepal_width	petal_length	petal_width	target
sepal_length	1	-0.12	0.87	0.82	0.78
sepal_width	-0.12	1	-0.43	-0.37	-0.43
petal_length	0.87	-0.43	1	0.96	0.95
petal_width	0.82	-0.37	0.96	1	0.96
target	0.78	-0.43	0.95	0.96	1

Figure 3.15: Heatmap of Correlation

Seeing the heatmap and the full pair plot, we can conclude that removing "`sepal_width`" can result in a better-correlated dataset with "`target`" as the output column.

In this case, we are removing the near to zero correlated feature because we will fit the data to a linear model. If we have a model that can understand the non-linearity of the data, then it is advised to provide features with less correlation value. When both the included feature is highly correlated, then giving any one of them is fine because the other feature is a linear combination of it, and it doesn't make sense if both of them are provided together.

Now removing the feature "`sepal_width`" for this example, we can visualize the dataset again.



```
iris_data.drop(columns=['sepal_width'], inplace=True)
iris_data.head()
```

	sepal_length	petal_length	petal_width	target	target_names
0	5.1	1.4	0.2	0	setosa
1	4.9	1.4	0.2	0	setosa
2	4.7	1.3	0.2	0	setosa
3	4.6	1.5	0.2	0	setosa
4	5.0	1.4	0.2	0	setosa

Figure 3.16: Dropping negatively Correlated Data Column

We should see the pair plot again to see the changes, and later we will confirm our hypothesis.

```
sns.pairplot(iris_data[['sepal_length', 'target_names',\
                        'petal_length', 'petal_width']], hue="target_names")
```

```
<seaborn.axisgrid.PairGrid at 0x1a240133c8>
```

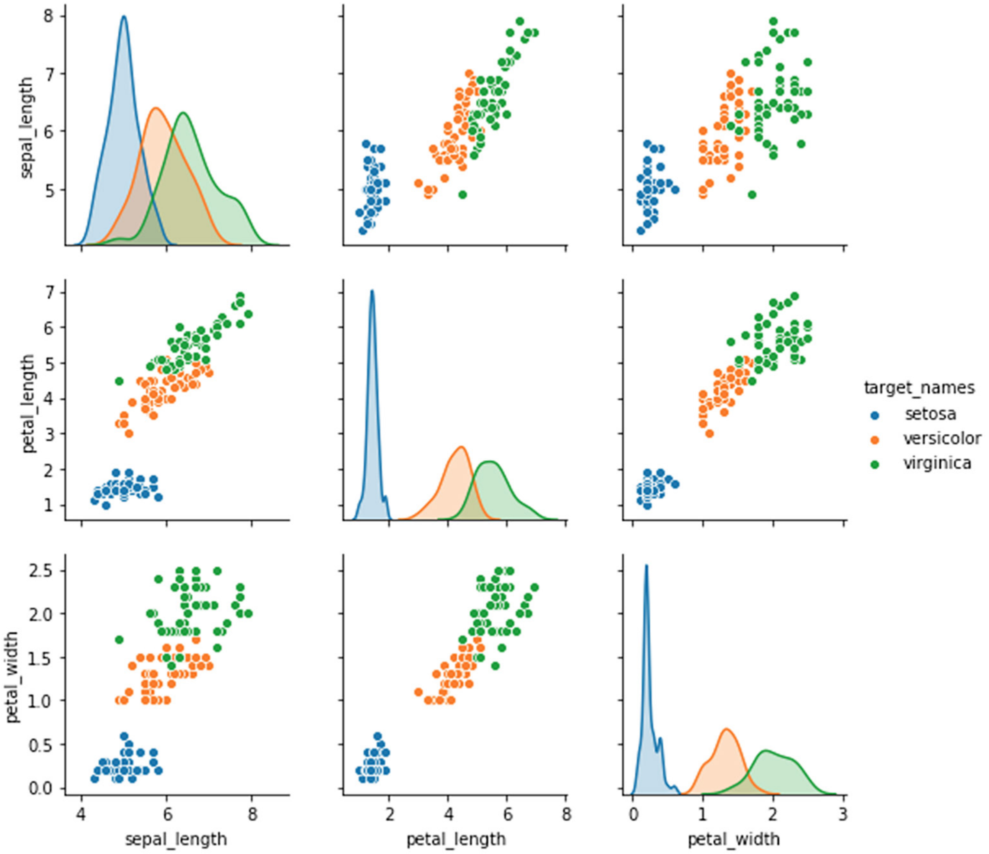


Figure 3.17: Pair Plot with the updated data

Next section onwards, we will see different ML techniques and perform some Data Preparation, which is specific to the ML algorithm.

How to choose ML algorithms?

Choosing an ML model depends on a single factor, i.e., the type of output (categorical, continuous, etc.)

Here we will talk about basic classification, regression, and clustering algorithms, which will be covered in this chapter.

Iris dataset is meant for classification problems but, we will modify the data, tweak a few things here and there in the dataset to make the dataset capable of regression and clustering problem.

Classification

Classification, also known as statistical classification[17] , which identifies and predicts different class/groups using predictive learning. Input for training classification model is a pair of input-output features, and during the phase of testing, it only takes input features to predict. Before moving forward, we need to make the data ready for mostly all classification models.

Now, the entire dataset has to be divided into four parts.

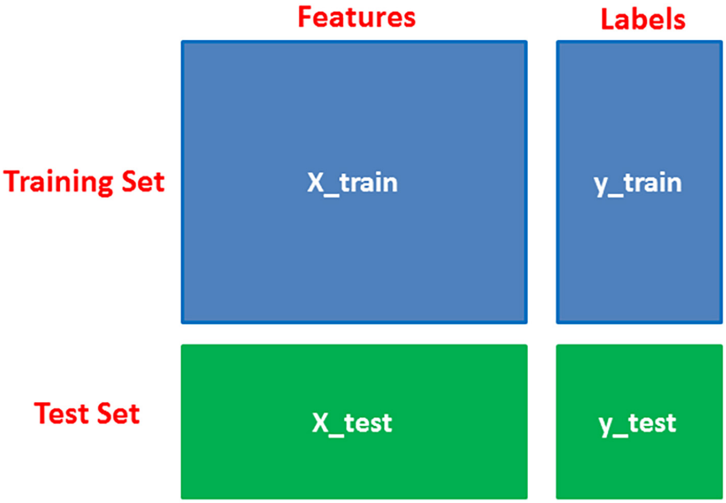
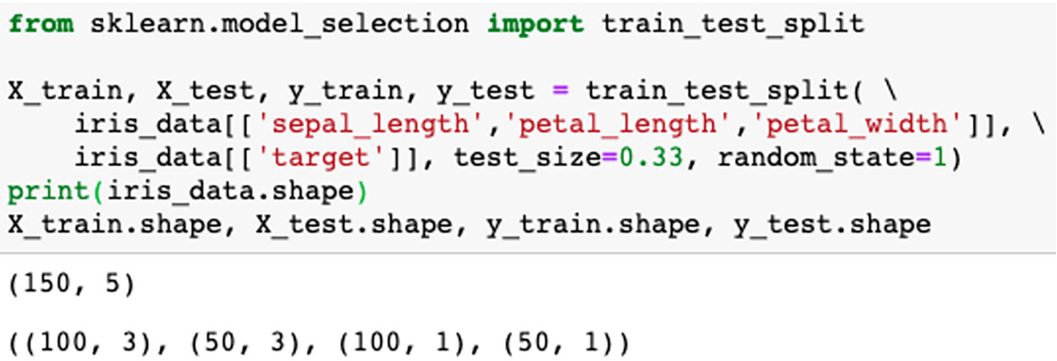


Figure 3.18: Basic Data Splitting for ML (Source: Datacamp)

The training set contains "`X_train`" and "`y_train`" where "`X_train`" will contain all the features and "`y_train`" contain all the respective "`target`" values. As the name says, both of the variables will be used for training.

The test set contains "`X_test`" which will contain all the features to test the model. And "`y_test`" will be used to evaluate the model for its accuracy and other metrics. So, let us split the dataset now.

[17] "Statistical classification - Wikipedia." https://en.wikipedia.org/wiki/Statistical_classification.



```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( \
    iris_data[['sepal_length','petal_length','petal_width']], \
    iris_data[['target']], test_size=0.33, random_state=1)
print(iris_data.shape)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
(150, 5)

((100, 3), (50, 3), (100, 1), (50, 1))
```

Figure 3.19: *Splitting the Dataset*

As we can see, we have split the dataset into 3:1 train to test ratio. For any kind of classification problem, this is the technique used to split the dataset.

We will only cover the K-nn classification algorithm for classifying the Iris dataset.

K - nearest neighbors (K-nn)[18]

K-nn is a non-parametric[19] supervised machine learning algorithm. It is one of the simplest algorithms to implement and understand. This algorithm can be used for classification and regression problems.

How does it work?

K-nn is an iterative model that works on the base formula for distance, and here we will use Euclidean Distance to find the distance between the 2 points in n-dimensional space.

$$d(p,q) = d(q,p) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$

$$\Rightarrow \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

where $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ are two points in Euclidean n-space[20].

Before formally explaining what K-nn is, let's take an example for better understanding:

[18] "k-nearest neighbors algorithm - Wikipedia." https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[19] "Supervised Machine Learning (Parametric)" https://www.slideshare.net/rehanguha/supervised-machine-learning-ml

[20] "Euclidean space - Wikipedia." https://en.wikipedia.org/wiki/Euclidean_space.

Let us assume that we have two classes yellow circle and violet circle class, and we will use the only 2D graph to plot it. That is, we will only use two features to plot it.

1. We will plot all the points on a graph making feature as their coordinates
2. Then we will plot the unknown point giving the features to predict the output class
3. K-nn works on the principle of the count of nearest neighbor (class) within a, particularly given range. We need to choose the right value of K for that.
4. Having the value K with us, we need to find the distance between the unknown point with all the points. Here we will use Euclidean distance for making the distance table.
5. With the distance data, we need to sort the data table into ascending order and consider only top K points
6. The maximum number of classes that belong to within the top K class is considered the new class of the unknown variable

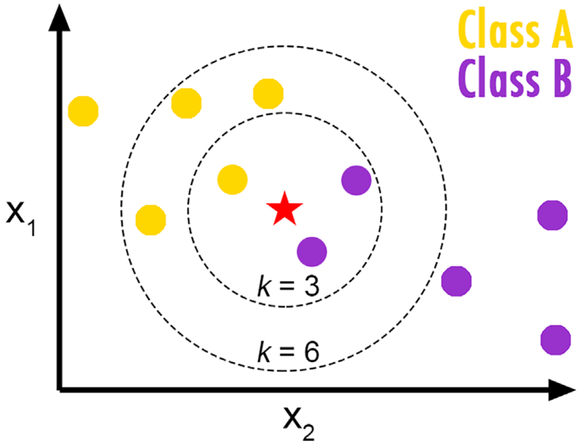


***Figure 3.20:** K-nn Clustering representation (Source: Medium)*

From the above image, we can see when *K* = 3 (the Violet circle, Class B), the class it predicts for the unknown data point is Class B but, when we increase the *K* = 6 (the Yellow circle), then the class of the unknown datapoint is Class A.

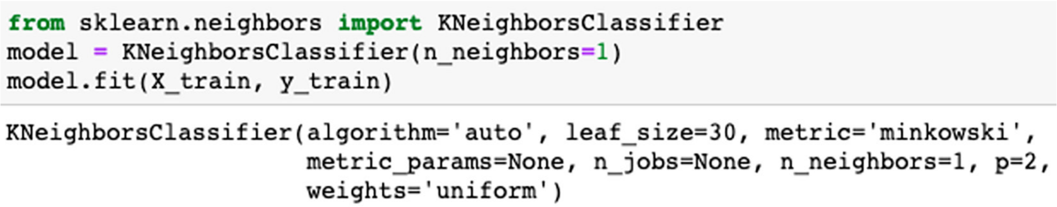
We can clearly understand the importance of selecting the right value of *K* for the model.

A smaller value of *K* will give a noisy and less accurate result, but on the other side, the very high value of *K* will generalize the model. It's always advisable to choose the optimal value of *K*; we will see how we can do it later.

Model training

As this is a classification model, we need to train them with input-output pair.

Let us perform the first iteration of model training with the variables ready.



```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=1)
model.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

Figure 3.21: *Training K-nn*

Yeah, this is it. We have successfully trained the model with $K = 1$.

Now using “`X_test`” and “`y_test`” we need to evaluate the model quality.

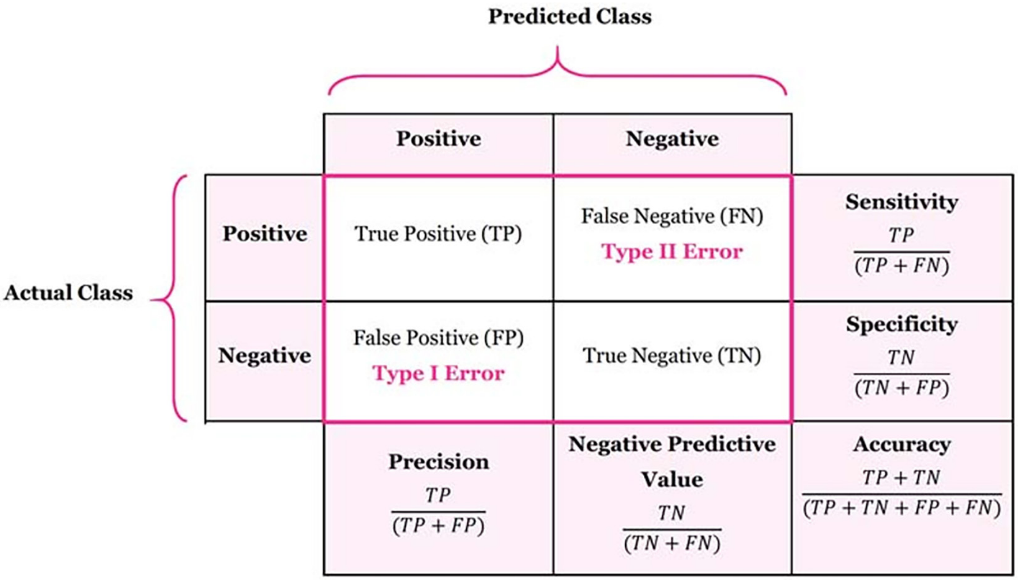
Evaluation

This is the phase where we will see the different metrics; we can derive from Confusion Matrix-like Accuracy, Precision, Sensitivity, etc. Later with the accuracy score, we find the optimal value of K.

What is the Confusion matrix[21]?

A **confusion matrix** is also referred to as an error matrix, which gives us the total number of correct vs. incorrect result in the form of a matrix. We can derive multiple metrics from a confusion matrix.

[21] “Confusion matrix - Wikipedia.” https://en.wikipedia.org/wiki/Confusion_matrix.



		Predicted Class		
		Positive	**Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

Figure 3.22: *Confusion Matrix (Source: Manisha-BlogSpot)*

We will see this confusion matrix a lot in future chapters, but here we will only go ahead with confusion matrix and analyze it.

```
from sklearn.metrics import confusion_matrix

sns.heatmap(confusion_matrix(y_test, model.predict(X_test)), annot=True)
print(y_test['target'].value_counts())
```

```
1    19
0    17
2    14
Name: target, dtype: int64
```

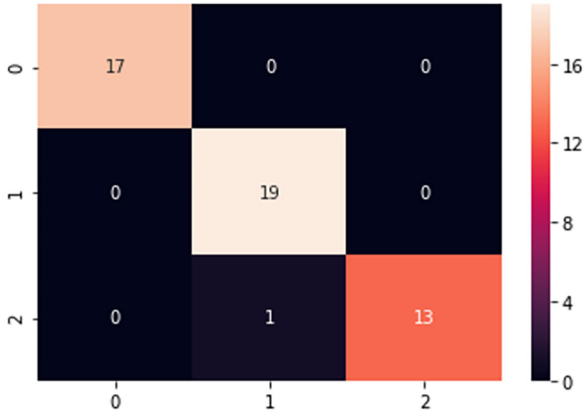


Figure 3.23: *Confusion matrix*

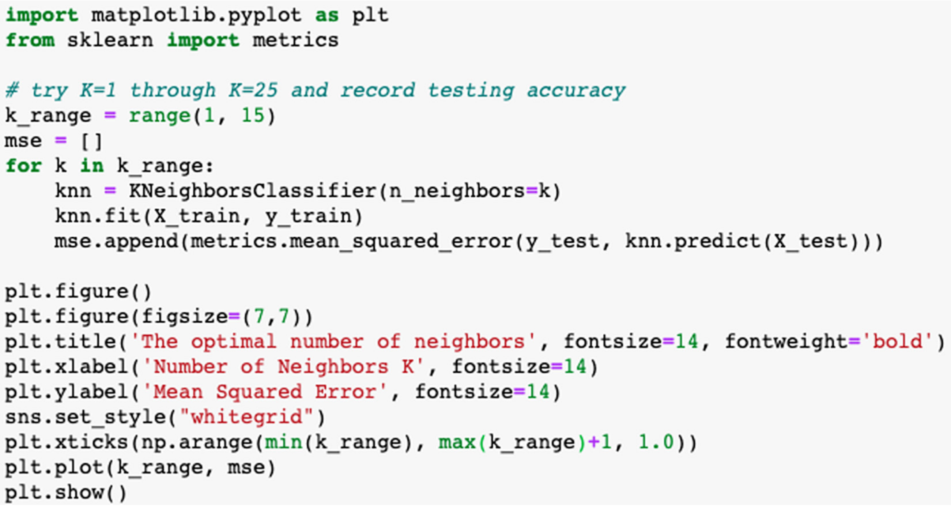
We are predicting the result with "`X_test,`" and comparing it with "`y_test`" and the results are pretty impressive. This result is good because we have spent a lot of time in data pre-processing and preparation.

Note: In any real-time ML project, 70% of your time will go in data pre-processing and rest 30% in ML modeling.

The diagonal represents true positives, i.e., it has correctly predicted all the classes which it was intended to. There is one misclassification for "`class 2`", where it should have been predicted "`class 2`" instead it predicted "`class 1`".

Model tuning

Tuning is a process of improving the model's accuracy by tuning the hyperparameters, in this case; the hyperparameters are no. of clusters and no. of iterations. But as the dataset is small and less complex, so we are keeping no. of iterations as constant. Let us figure out the optimal value of *K*. We can use the below code snippet to find it.



```
import matplotlib.pyplot as plt
from sklearn import metrics

# try K=1 through K=25 and record testing accuracy
k_range = range(1, 15)
mse = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    mse.append(metrics.mean_squared_error(y_test, knn.predict(X_test)))

plt.figure()
plt.figure(figsize=(7,7))
plt.title('The optimal number of neighbors', fontsize=14, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=14)
plt.ylabel('Mean Squared Error', fontsize=14)
sns.set_style("whitegrid")
plt.xticks(np.arange(min(k_range), max(k_range)+1, 1.0))
plt.plot(k_range, mse)
plt.show()
```

Figure 3.24: *Optimal Neighbour Calculation*

It's a simple idea, we will iterate through some values *K*, and it will range between *K*∈[1,15].

The metric we will use here is **Mean Squared Error (MSE)**[22] or **Mean Squared Deviation (MSD)** is the measure for an average of the square of errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$$

[22] "Mean squared error - Wikipedia." https://en.wikipedia.org/wiki/Mean_squared_error.

The above formula gives us the error for each value of *K*.

We can plot all the MSE values with their respective *K* values on a line plot.

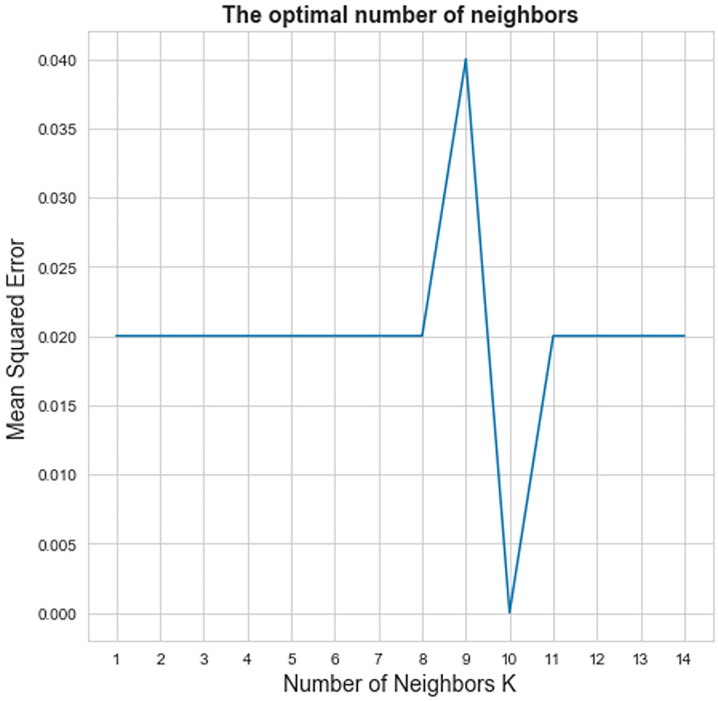


***Figure 3.25:** k-NN vs. Error plot*

Seeing the image, we can see that error is highest when we choose the value of $K = 9$, and it is lowest when $K = 10$. Lower the MSE better the model it is.

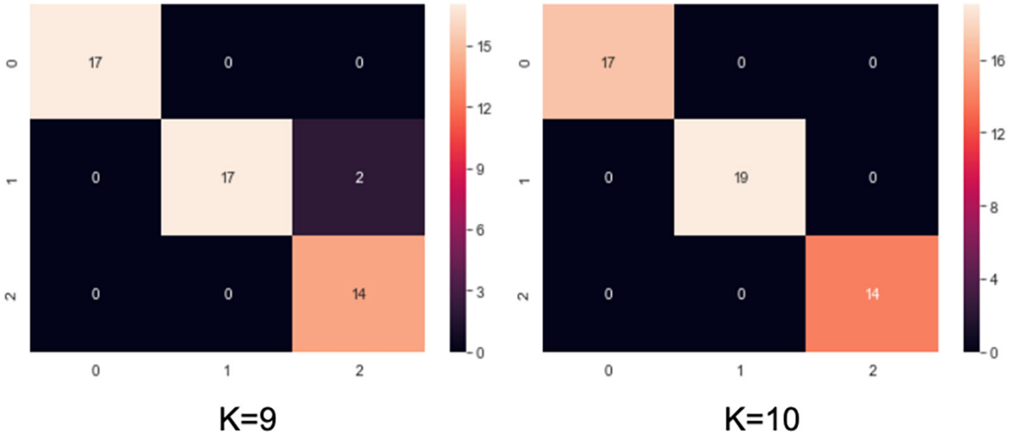


***Figure 3.26:** Comparison of best and the worst model*

The confusion matrix shows that when $K = 10$ the MSE is Zero, which means all the classes are successfully classified, and there is no misclassification.

But for $K = 9$, MSE is the highest, and the model is the worst for that case. And for all other values of K, the misclassification rate is the same.

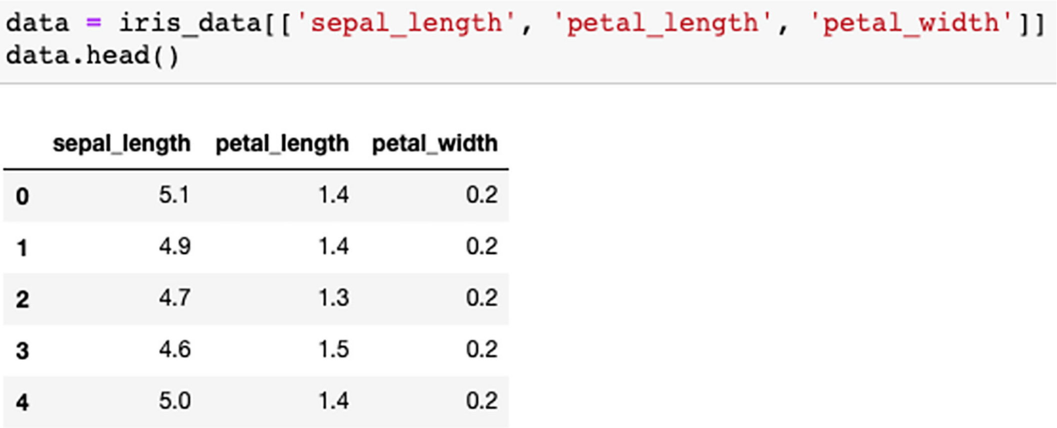
K-nn is an iterative process, and its complexity to execute the algorithm is very high, so to save the computational time, decision boundary[23] is drawn for K-nn[24].

Clustering[25]

Clustering is the method by which the similar data points in n-dimensional space is divided into different groups. Data points in each group will be similar to each other, and it will be different from other groups.

For using the dataset for the clustering problem, we need to remove the column "`target`" and "`target_names`."

We need to remember that clustering is an unsupervised type of machine learning, so the input to the training algorithm will only be the input features.



```
data = iris_data[['sepal_length', 'petal_length', 'petal_width']]
data.head()
```

	sepal_length	petal_length	petal_width
0	5.1	1.4	0.2
1	4.9	1.4	0.2
2	4.7	1.3	0.2
3	4.6	1.5	0.2
4	5.0	1.4	0.2

Figure 3.27: *Data preparation for clustering*

Above, we can see the final dataset that will be used for clustering problems.

[23] "Decision boundary - Wikipedia." https://en.wikipedia.org/wiki/Decision_boundary.

[24] "Drawing Decision Boundaries for Nearest Neighbors - Umich." http://www-personal.umd.umich.edu/~leortiz/teaching/6.034f/Fall06/knn_dt/nn-bounds+soln.pdf.

[25] "Cluster analysis - Wikipedia." https://en.wikipedia.org/wiki/Cluster_analysis.

K-means

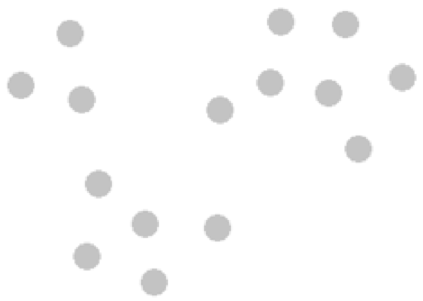
K-means clustering is an unsupervised machine learning algorithm to group unlabeled data into multiple classes.

How does it work?

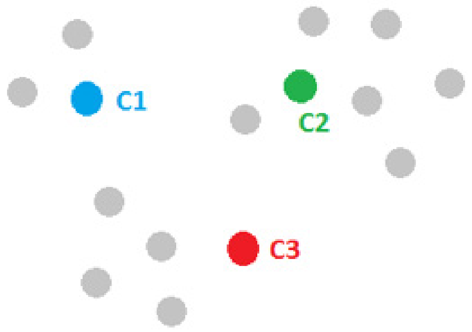
K-means uses the concept of centroids to assign classes/groups to the data points. *K* represents the total number cluster, i.e., *K* number of centroids.

Steps for the K-means algorithm are as follows:

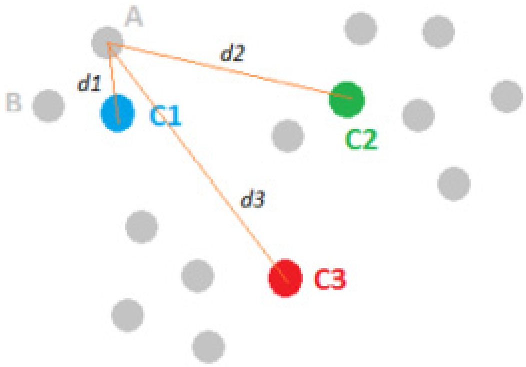
1. Plot all the data points in n-dimensional feature space and decide the value of *K* i.e., the number of clusters.



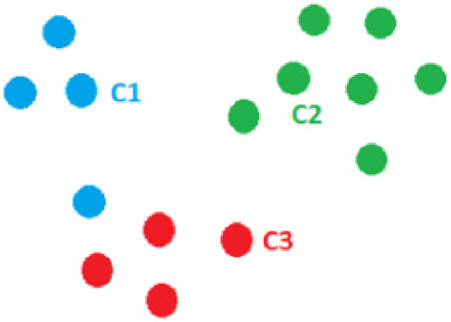
2. Now randomly plot the value of *K* points on the graph that will represent each centroid of the cluster.



3. Calculate all the Euclidean distance of the datapoint from the centroids.



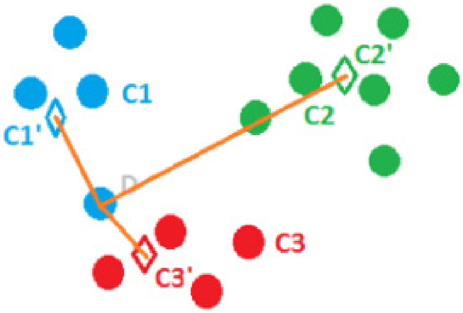
4. Assign the least distant centroid as a class to the data points



We can write this mathematically as:

$$\underset{c_i \cup C}{\arg\min}\, dist(c_i, x)^2$$

5. Calculate the new value of centroid with the data points assigned from the previous step



Assigning new centroids:

$$c_i = \frac{1}{S_i} \sum_{x_i \in S_i} x_i$$

So this process tries to choose centroids that minimize the distance from all the data points to this respective group/cluster.

6. Repeat Step 3 to Step 5 until and unless the position of the centroid is not changing, or we can say until convergence.

We can represent this process using an objective function, and we will aim to minimize that function.

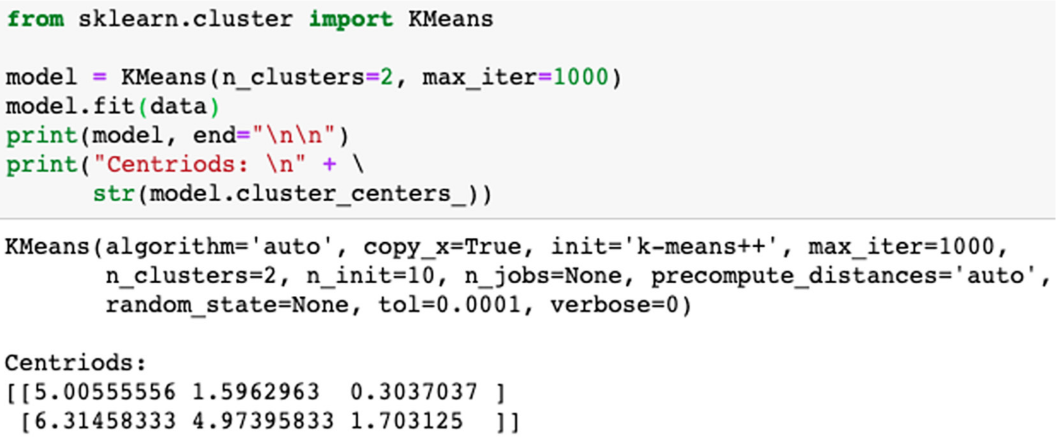
$$\sum_{i=1}^{n}\sum_{j=1}^{k}(\| x_i - c_j \|)^2 = 1$$

Where, $| | x_i - c_j | |$ is the euclidean distance between the datapoint x_i and the centroid c_j which is looped overall n^{th} points in the i^{th} cluster for all k clusters.

The algorithm is simple, but finding the optimal solution to the problem for higher dimensions d and k clusters is an NP-Hard[26] problem[27].

Model training

Let us start with all the features and train K-Means where $K = 2$. It is very important to know that we need to minimize the objective function. We have seen in the above algorithm that the initial values of the centroids are chosen at random. But to optimize the algorithm, we will use K-means++[28] to initialize the values of the centroid. So, now let us train K-means with the dataset and fit it.



```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=2, max_iter=1000)
model.fit(data)
print(model, end="\n\n")
print("Centriods: \n" + \
      str(model.cluster_centers_))
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=1000,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)

Centriods:
[[5.00555556 1.5962963  0.3037037 ]
 [6.31458333 4.97395833 1.703125  ]]
```

Figure 3.28: Modeling K-means clustering with entire data

[26] "NP-hardness - Wikipedia." https://en.wikipedia.org/wiki/NP-hardness.

[27] "NP-Hard Problem -- from Wolfram MathWorld." http://mathworld.wolfram.com/NP-HardProblem.html.

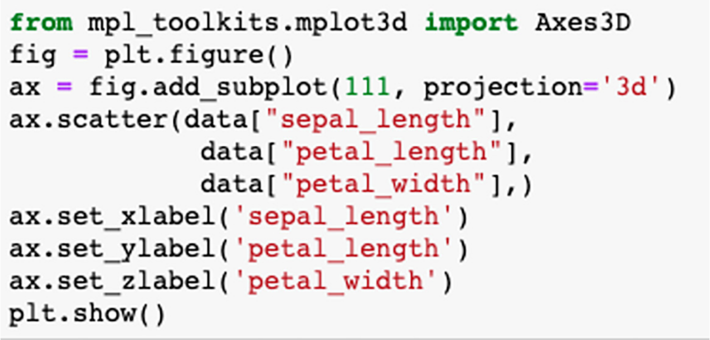
[28] "k-means++ - Wikipedia." https://en.wikipedia.org/wiki/K-means%2B%2B.

As "`X_train`" has three features so all the data points will be plotted in a 3-Dimensional graph, and it is extremely tough to visualize it.

I have coded K-Means from scratch, which will be helpful to refer and to see how K-Means work.

Link: https://gist.github.com/rehanguha/564fcd0ddbb389a1c4c5766ba003ed0c.

Let us go ahead and plot a 3-D graph see how difficult it is to interpret the results.



```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data["sepal_length"],
            data["petal_length"],
            data["petal_width"],)
ax.set_xlabel('sepal_length')
ax.set_ylabel('petal_length')
ax.set_zlabel('petal_width')
plt.show()
```

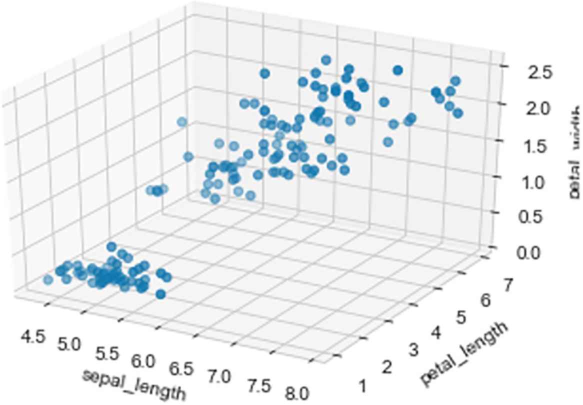
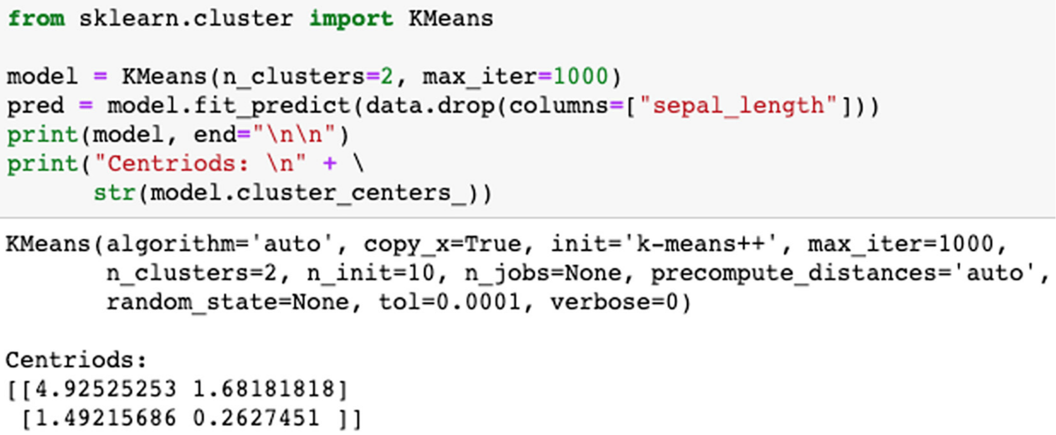


Figure 3.29: 3D plot for three features with data points

We can see the visualizing 3D graph is tough, and understanding the spatial information is complex. Just for visualization, I will be using only two features so that it can be plotted on the 2-D graph. Now we will re-train the entire model dropping one column.

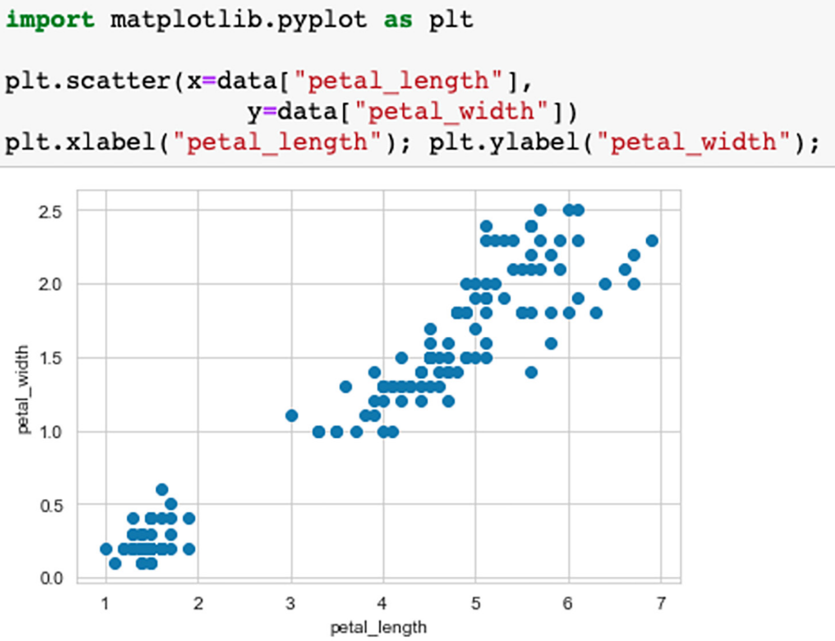


```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=2, max_iter=1000)
pred = model.fit_predict(data.drop(columns=["sepal_length"]))
print(model, end="\n\n")
print("Centriods: \n" + \
      str(model.cluster_centers_))
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=1000,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)

Centriods:
[[4.92525253 1.68181818]
 [1.49215686 0.2627451 ]]
```

Figure 3.30: K-means with two features

We can observe that the position of the centroids has changed, and the stopping condition for the algorithm is a maximum of 1000 iterations. Now, plotting the data for "`petal_length`" and "`petal_width`".

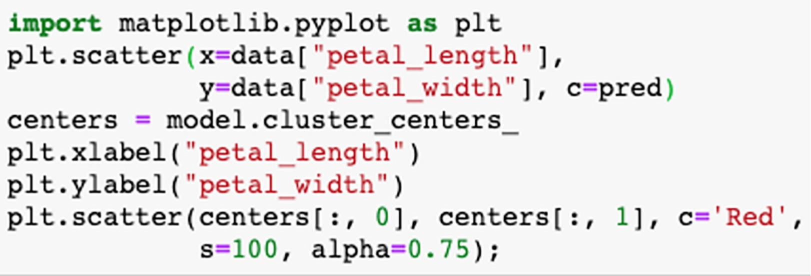


```
import matplotlib.pyplot as plt

plt.scatter(x=data["petal_length"],
            y=data["petal_width"])
plt.xlabel("petal_length"); plt.ylabel("petal_width");
```

Figure 3.31: Visualizing two features data point with scatter plot

If we plot all the data points and manually try to find the cluster when we can see two groups, first group - at the lower left-hand corner, and the second group- at the top right corner.

Now, let's see how and where does K-means place both the centroids.



```
import matplotlib.pyplot as plt
plt.scatter(x=data["petal_length"],
            y=data["petal_width"], c=pred)
centers = model.cluster_centers_
plt.xlabel("petal_length")
plt.ylabel("petal_width")
plt.scatter(centers[:, 0], centers[:, 1], c='Red',
            s=100, alpha=0.75);
```

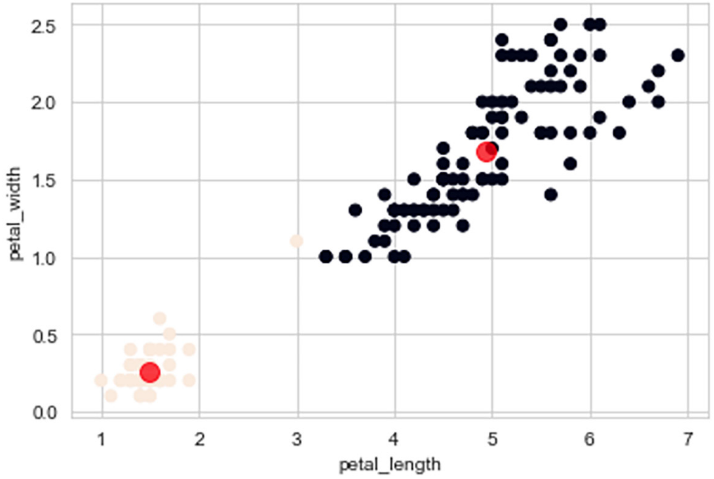


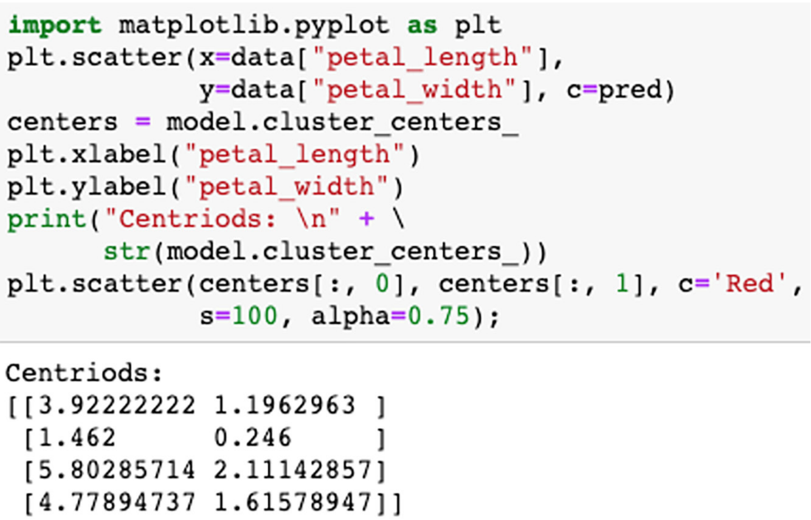
Figure 3.32: *Visualizing the Centroids for two features and K=2*

The red dots are the centroids of the clusters from the trained model with $K = 2$. We had made some hypothesis before plotting and, it matches the hypothesis.

We should remember that mostly the data will be in the higher dimension, and visualizing it will be challenged, so we need to rely on unsupervised learning and their given clusters.

One of the biggest hurdles for any unsupervised machine learning model is to evaluate the results. There are a smaller number of methods to check how good the model is.

Let us see what will happen if we increase the number of clusters and plot on the same data with two features. Now we will train K-means with 3 clusters and see where the centroids are placed.



```
import matplotlib.pyplot as plt
plt.scatter(x=data["petal_length"],
            y=data["petal_width"], c=pred)
centers = model.cluster_centers_
plt.xlabel("petal_length")
plt.ylabel("petal_width")
print("Centriods: \n" + \
      str(model.cluster_centers_))
plt.scatter(centers[:, 0], centers[:, 1], c='Red',
            s=100, alpha=0.75);
```

```
Centriods:
[[3.92222222 1.1962963 ]
 [1.462      0.246     ]
 [5.80285714 2.11142857]
 [4.77894737 1.61578947]]
```

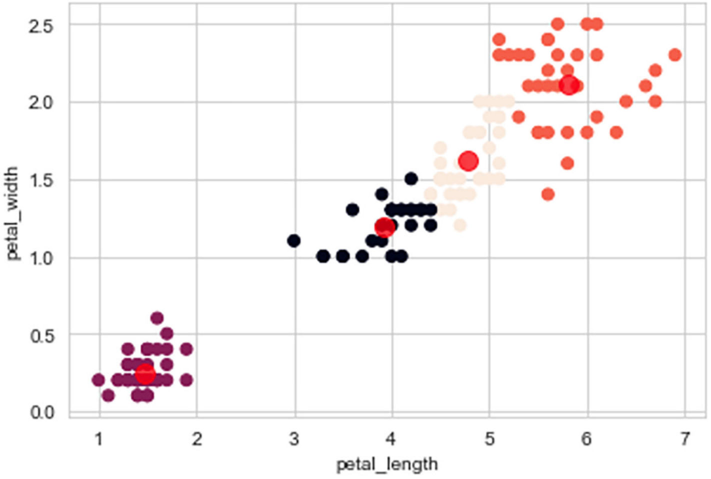


Figure 3.33: Visualizing centroids when K=4

When we have increased the number of centroids to $K = 4$, then we get the clusters like the above image.

Now the problem is how we should find the optimal value of K and analyze the clusters.

Note: Analyzing and finding the optimal value of K is an expensive process for higher dimensions, and that has to be taken care of during the exhaustive search.

Evaluation & model tuning

Here, we will play with the value to *K* and see how it changes the error of the overall cluster.

Inertia or within-cluster sum-of-squares criterion

The within-cluster sum of squares is a metric of the variability of the observations within each cluster. In general, a cluster that has a small sum of squares is more compact than a cluster that has a large sum of squares. Clusters that have higher values show greater variability of the observations within the cluster, i.e., the data points will be sparser.

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\| x_i - c_j \|^2 \right)$$

We had seen the same objective function before, where we tried to minimize it. So, the same mathematical formula is used to calculate the Inertia or within-cluster sum-of-squares criterion.

Inertia can be used to evaluate the clusters, so this method can be used for finding the optimal value of *K*.

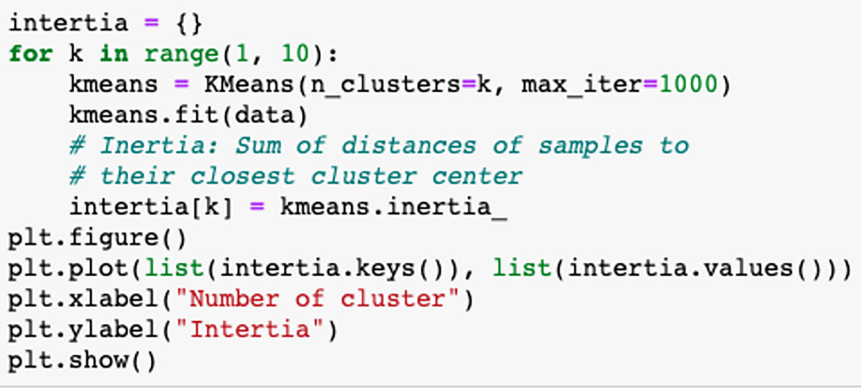
Elbow method[29] using Inertia

The **Elbow method** is a process of selecting the optimal value of *K* for the data with the given clustering algorithm. In this case, it is K-means with 1000 max iteration and K-Means++ initialization.

This is a normal plot which is similar to K-nn model tuning and finding the optimal value of *K*.

We need "error" and "no. of clusters" as an axis to implement Elbow method for a clustering problem. The point with least error and the place where it follows an elbow pattern, that point can be considered as the optimal value of *K*. Below we can see the line plot, and we need to find an elbow from the plot.

[29] "Elbow method (clustering) - Wikipedia." https://en.wikipedia.org/wiki/Elbow_method_(clustering).



```
intertia = {}
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000)
    kmeans.fit(data)
    # Inertia: Sum of distances of samples to
    # their closest cluster center
    intertia[k] = kmeans.inertia_
plt.figure()
plt.plot(list(intertia.keys()), list(intertia.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Intertia")
plt.show()
```

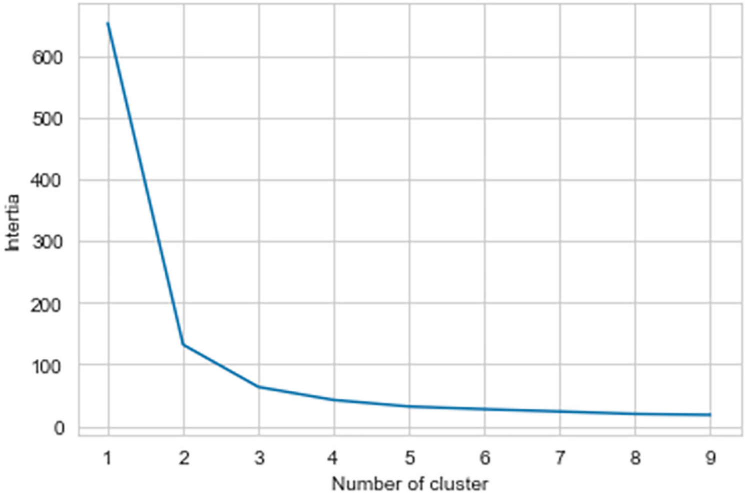


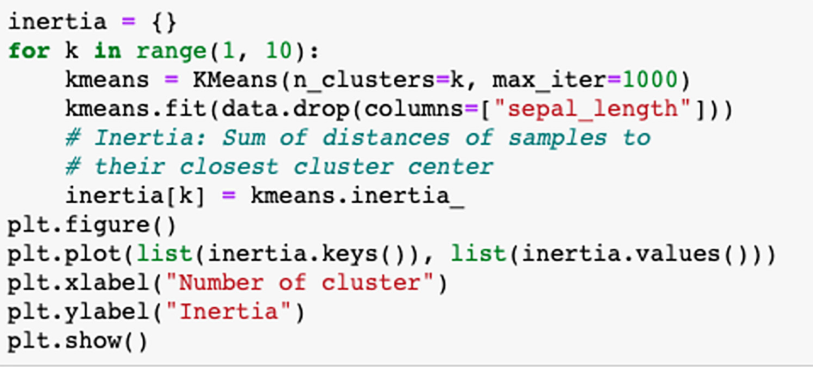
Figure 3.34: *Elbow Chart for 3 Features*

The above image plots the Inertia with the number of clusters for all the three features. Reading elbow graphs is easy. We need to find the spot where the change in Inertia is less.

We can see if the value of $K = 3$, then delta(Δ), is very less. So, for all three features if we cluster the data using K-means, then $K = 3$ is the optimal number of clusters.

Note: As this is a small dataset, it is easy to plot, visualize elbow graph and find the number of clusters, computing multiple clusters is a costly process.

Let us see if the same pattern is followed for the same dataset with three features.



```
inertia = {}
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000)
    kmeans.fit(data.drop(columns=["sepal_length"]))
    # Inertia: Sum of distances of samples to
    # their closest cluster center
    inertia[k] = kmeans.inertia_
plt.figure()
plt.plot(list(inertia.keys()), list(inertia.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Inertia")
plt.show()
```

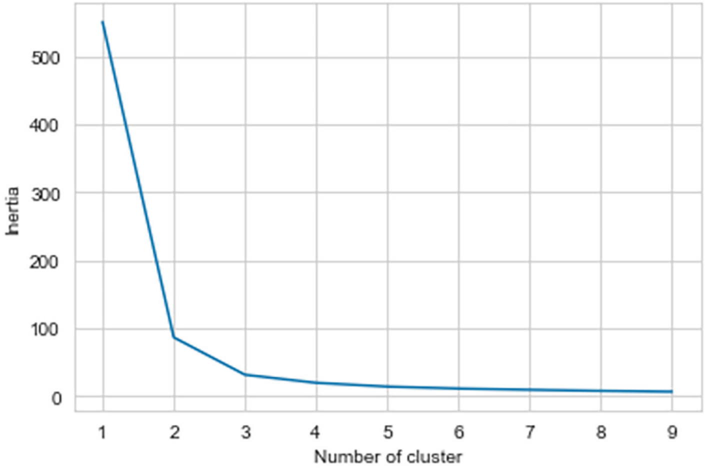


Figure 3.35: *Elbow Chart for 2 Features*

We can confirm our hypothesis and say, that the elbow charts look similar, and it follows the same trend. Just that inertia values of all cluster(s) are slightly different.

Here the optimal value for *K* is three as the 3 has the least value of Inertia. The higher order of *K*'s (number of clusters) is not at all a favorable choice because it does not give us the true nature/pattern of the clusters.

Like, when we saw the clusters on a plot. *Figure 3.30*, we guessed that there could be 2 clusters. But naturally, we didn’t guess that the plot will fit with 10 clusters. So, we need to choose a cluster with the least error, in this case, its within-cluster sum-of-squares criterion.

We will see another metric, i.e., the Silhouette Score.

Silhouette Score[30]

Silhouette gives the consistency of the clusters. It is a measure of its cluster (cohesion) compared to other clusters(separation).

Silhouette Score is represented by,

$$s(i) = \frac{b(i) - a(i)}{max\left\{a(i), b(i)\right\}}, \; if \mid c_i \mid \; > 1$$

Where $a(i)$ and $b(i)$ is,

$$a(i) = \frac{1}{\mid c_i \mid - 1} \sum_{j \in c_i, i \neq j} d(i, j)$$

It is the mean distance between i and other data points in the same cluster. $d(i, j)$ is the distance

And,

$$b(i) = \min_{k \neq i} \frac{1}{\mid c_k \mid} \sum_{j \in c_k} d(i, j)$$

It is the smallest mean distance of i to all points in any other cluster, of which i is not a member.

Therefore,

$$s(i) = 0, \; if \mid c_i \mid \; = 1$$

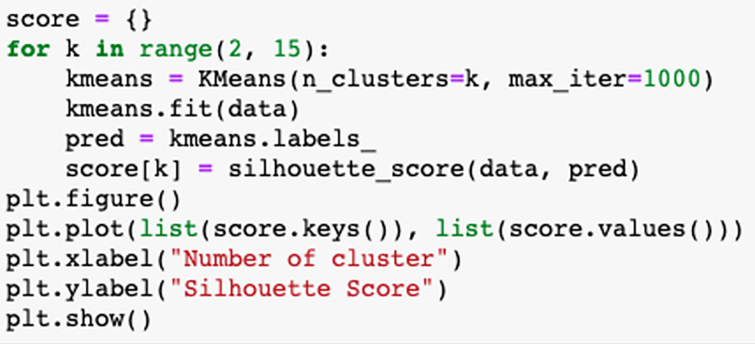
So, we can write the equation as,

$$s(i) = \begin{cases} 1 - a(i) / b(i), & if \;\; a(i) < b(i) \\ 0, & if \;\; a(i) = b(i) \\ b(i) / a(i) - 1, & if \;\; a(i) > b(i) \end{cases}$$

Value of $s(i)$ will range from, $-1 \leq s(i) \leq 1$. And $s(i) = 0$ when the number of clusters is 1. A higher value indicates higher cohesiveness, so the clustering is good. 0 indicates overlapping clusters, whereas negative value means wrong clustering of the object because there exists another more similar cluster.

We will implement the above concept to the clusters and see how good each cluster is.

[30] "Silhouette (clustering) - Wikipedia." https://en.wikipedia.org/wiki/Silhouette_(clustering).



```python
score = {}
for k in range(2, 15):
    kmeans = KMeans(n_clusters=k, max_iter=1000)
    kmeans.fit(data)
    pred = kmeans.labels_
    score[k] = silhouette_score(data, pred)
plt.figure()
plt.plot(list(score.keys()), list(score.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Silhouette Score")
plt.show()
```

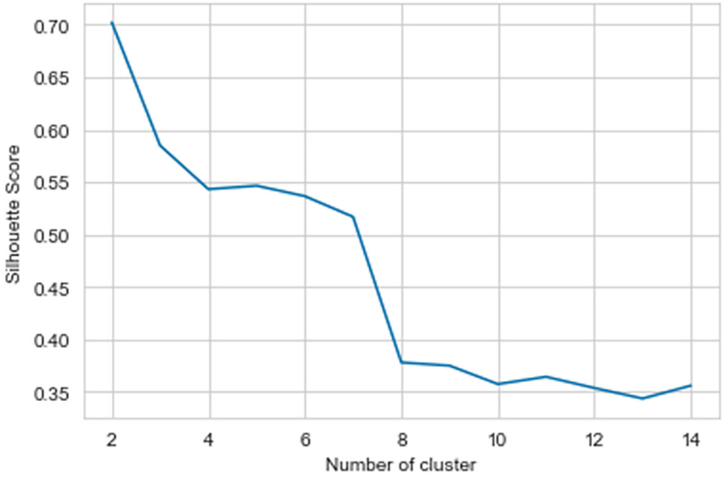


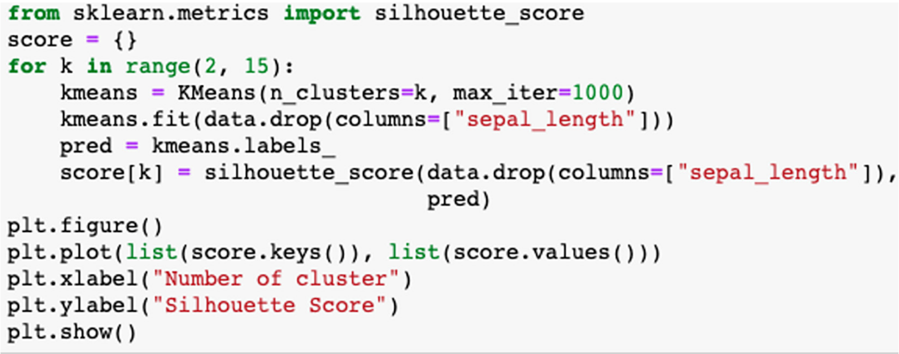
Figure 3.36: *Error vs. Number of clusters for three features*

This visualization is in the simplest form, and understandability is very high amongst other forms of visualizations[31].

Here we are looking for higher values on the Silhouette Score for each value of *K* (Number of clusters).

The above one is with the entire dataset, with all three features. Next, we will see only with two features then conclude the optimal value of *K*.

[31] "Selecting the number of clusters with silhouette ... - Scikit-learn." http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html.



```
from sklearn.metrics import silhouette_score
score = {}
for k in range(2, 15):
    kmeans = KMeans(n_clusters=k, max_iter=1000)
    kmeans.fit(data.drop(columns=["sepal_length"]))
    pred = kmeans.labels_
    score[k] = silhouette_score(data.drop(columns=["sepal_length"]),
                                pred)
plt.figure()
plt.plot(list(score.keys()), list(score.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Silhouette Score")
plt.show()
```

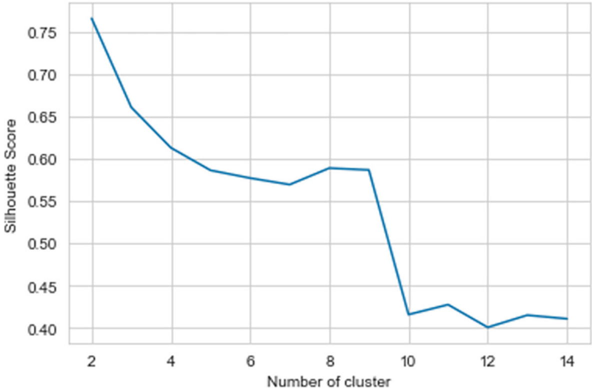


Figure 3.37: *Error vs. Number of clusters for two features*

Seeing the above chart, we can conclude that the optimal value of *K* can either be 3 or 2.

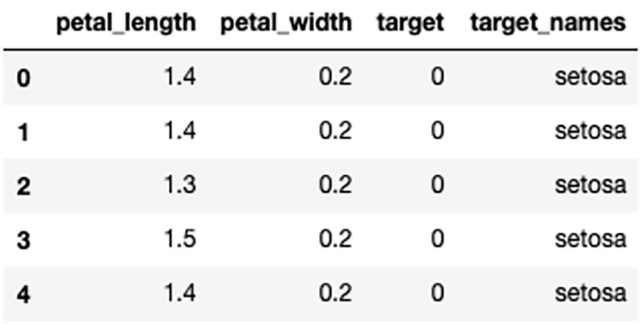
But combining both the results from Silhouette Score and Inertia, we can conclude that the optimal value of $K = 3$.

It is worth noting that the evaluation for the unsupervised model is a challenge. So, it is advised to use multiple evaluation techniques to confirm the results.

Regression

Regression is another type of Machine Learning problem, which helps us to predict continuous values. Some of the known algorithms for regression are linear regression, logistic regression, etc. Here we will use the same dataset with some tweak in it. For any kind of regression, we need input-output pair for training purposes and for predicting, we need only input of the features as said earlier, that Iris dataset is not meant for regression. Still, we need to change and use some parts of the data like the code below.

```
data = iris_data[['petal_length', 'petal_width',
                  'target', 'target_names']]
data.head()
```



	petal_length	petal_width	target	target_names
0	1.4	0.2	0	setosa
1	1.4	0.2	0	setosa
2	1.3	0.2	0	setosa
3	1.5	0.2	0	setosa
4	1.4	0.2	0	setosa

Figure 3.38: *Data Preparation*

Here, we will only work with two features, i.e., "`petal_length`" and "`petal_width`." Here we will predict the "`petal_width`" with the help of "`petal_length`." Let us see how both the features look in a pair plot.

```
sns.pairplot(data[['petal_length', 'petal_width','target_names']],
             hue='target_names')
```

```
<seaborn.axisgrid.PairGrid at 0x1a2b85e4e0>
```

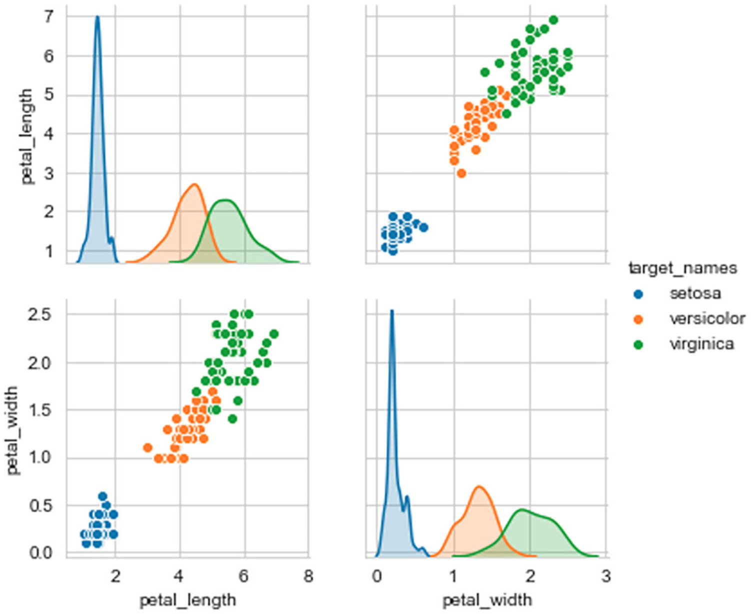
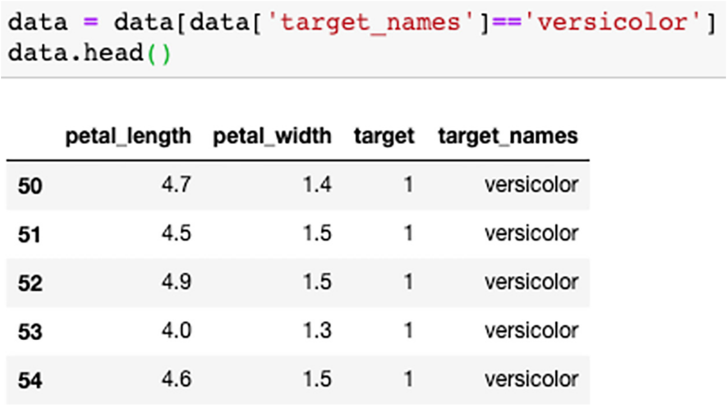


Figure 3.39: *Pair Plot for two selected features*

We will only select “versicolor” to predict its “petal_width” in the following section. Because each species will have their characteristics so, all the features won’t be logical and useful at the same time. Now, we will only keep ‘versicolor’ from the entire dataset.

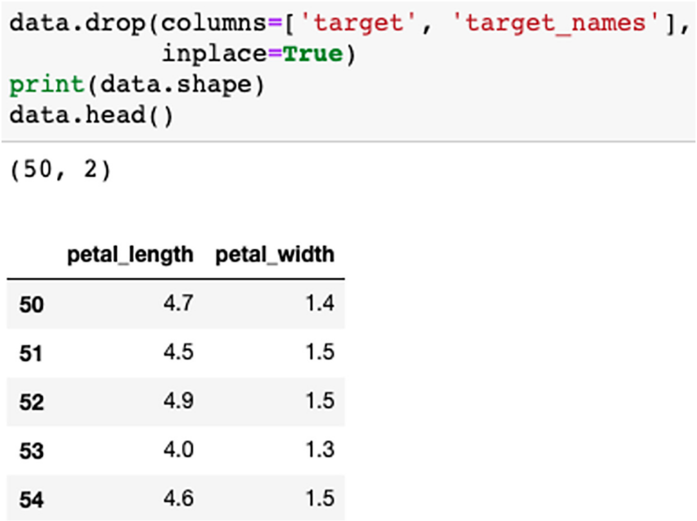


```
data = data[data['target_names']=='versicolor']
data.head()
```

	petal_length	petal_width	target	target_names
50	4.7	1.4	1	versicolor
51	4.5	1.5	1	versicolor
52	4.9	1.5	1	versicolor
53	4.0	1.3	1	versicolor
54	4.6	1.5	1	versicolor

Figure 3.40: *Choosing only one species*

With that, I need to remove/drop the “target_names” and “targets” because there is no use for those columns for a regression problem.



```
data.drop(columns=['target', 'target_names'],
          inplace=True)
print(data.shape)
data.head()
```

```
(50, 2)
```

	petal_length	petal_width
50	4.7	1.4
51	4.5	1.5
52	4.9	1.5
53	4.0	1.3
54	4.6	1.5

Figure 3.41: *Dropping irrelevant columns for regression*

We have completed the data preparation for the Regression problem. Now, we will start by modeling.

Linear regression[32] [33]

Linear regression is a statistical method to model a linear relationship between two scalars. Here, we will use the Ordinary Least Square method to estimate the unknown values of the equation.

How does it work?

Linear regression or simple regression is a numerical formula of a line.

$$y = mx + c$$

Where *c* is the intercept, and *m* is the slope.

We can write the same equation as,

$$\hat{y} = \beta_0 + \beta_1 x$$

Where, β_0 is the intercept and β_1 is slope or co-efficient

The simple formula for the line has two unknowns i.e. β_0, β_1 and two inputs, i.e., *x*, *y*.

To compute the value of β_0 and β_1 we can use the below formula of Ordinary Least Square method[34].

$$\beta_1 = \frac{\sum_i \left(x_i - \overline{x}\right)\left(y_i - \overline{y}\right)}{\sum_i \left(x_i - \overline{x}\right)^2}$$

and,

$$\beta_0 = \overline{y} - \beta_1 \overline{x}$$

$$\Rightarrow \overline{y} - \left(\frac{\sum_i \left(x_i - \overline{x}\right)\left(y_i - \overline{y}\right)}{\sum_i \left(x_i - \overline{x}\right)^2}\right) \cdot \overline{x}$$

Here, $\overline{x}$ and $\overline{y}$ are the statistical mean of the features *x* and *y* respectively.

After iterating through all the values of the training data, we get an equation of a line where we have the values of the two unknown variables i.e. β_0, β_1.

When we use the same line function to predict, we will use the given value of *x* as input and the rest of the unknowns to get the answer.

Note: All the predicted values from the Linear Regression will lie on the line $\hat{y}_{predicted} = \beta_0 + \beta_1 . x_{input}$

[32] "Linear regression - Wikipedia." https://en.wikipedia.org/wiki/Linear_regression.

[33] "Linear Regression." http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm.

[34] "Ordinary least squares - Wikipedia." https://en.wikipedia.org/wiki/Ordinary_least_squares.

This holds true for two features (here x, y), and we can have a similar equation for multiple variables, which has only one output feature that is known as multiple regression[35] [36].

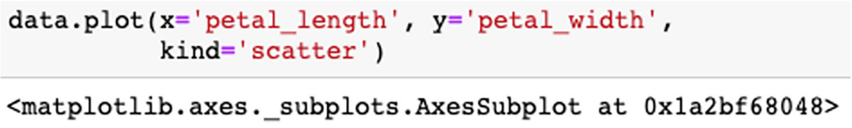
$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \ldots + \beta_n x_n$$
$$\therefore$$
$$\hat{y} = \beta_0 + \sum(\beta_i x_i)$$

To find the value of β_i, $i = 1 \ldots n$ we can use **Ordinary Least Squares (OLS)**[37]. OLS is one of the methods to find unknown values[38].

Here also we need to find the optimal value of β_i, $i = 0 \ldots n$ both simple linear equations and multiple regression. Our target is to minimize the sum of squared errors or mean squared error of the prediction.

Model training

From the data pair plot, we have selected "`petal_length`" and "`petal_width`" for this work. Let us start by visualizing it.



```
data.plot(x='petal_length', y='petal_width',
          kind='scatter')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2bf68048>
```

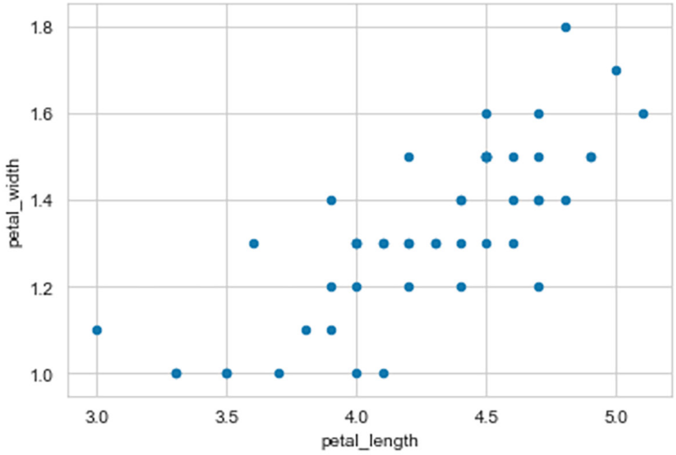


Figure 3.42: *Scatter plot for the data*

[35] "Multiple regression - Handbook of Biological Statistics." 20 Jul. 2015, http://www.biostathandbook.com/multipleregression.html.

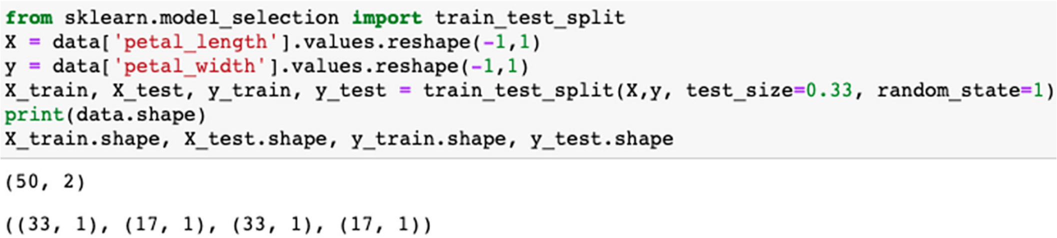
[36] "Chapter 3 Multiple Linear Regression Model The linear ... - IITK." http://home.iitk.ac.in/~shalab/regression/Chapter3-Regression-MultipleLinearRegressionModel.pdf.

[37] "Multiple Regression." http://www.csulb.edu/~msaintg/ppa696/696regmx.htm.

[38] "Multiple Regression - Data Science." http://www.statsoft.com/Textbook/Multiple-Regression.

If we carefully see the data, we see a linear pattern in the data. These kinds of data will perform best for the linear regression problems.

Let us split the dataset into training and testing datasets as we had seen earlier for Classification problems.

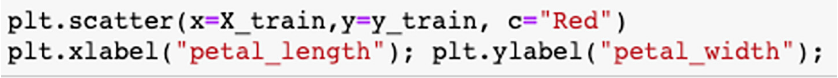


```
from sklearn.model_selection import train_test_split
X = data['petal_length'].values.reshape(-1,1)
y = data['petal_width'].values.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33, random_state=1)
print(data.shape)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
(50, 2)

((33, 1), (17, 1), (33, 1), (17, 1))
```

Figure 3.43: *Splitting data for training a regression model*

After splitting the dataset, we can see use them for training a model. But first, let us visualize the train and the test dataset.



```
plt.scatter(x=X_train,y=y_train, c="Red")
plt.xlabel("petal_length"); plt.ylabel("petal_width");
```

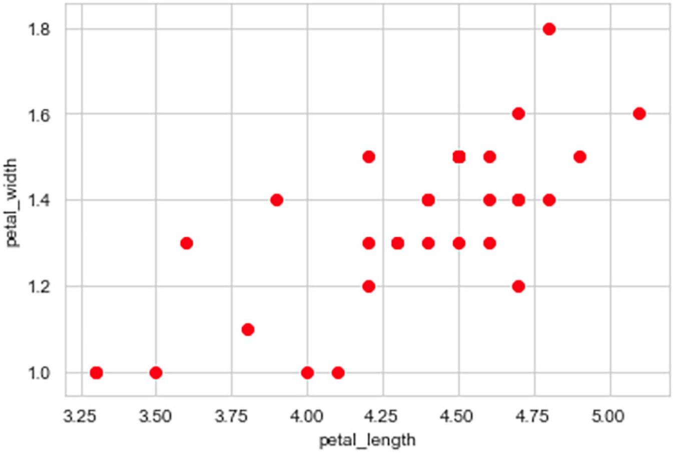
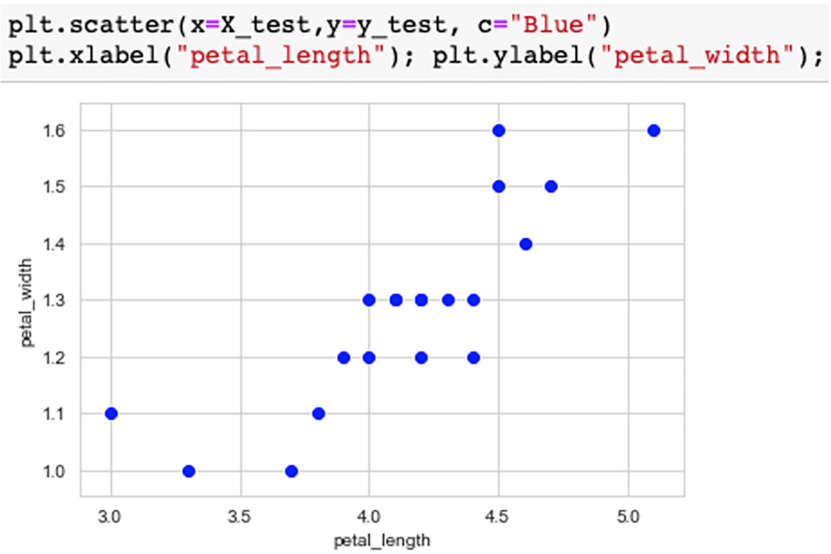


Figure 3.44: *Visualizing training data*

Similarly, we need to plot the test dataset and see if it follows the same pattern or not. There can be a problem with the covariate shift[39]. In short- If the distribution of test data and train data is different, then the model does not perform well for the test data. The below image shows the distribution for the test data.

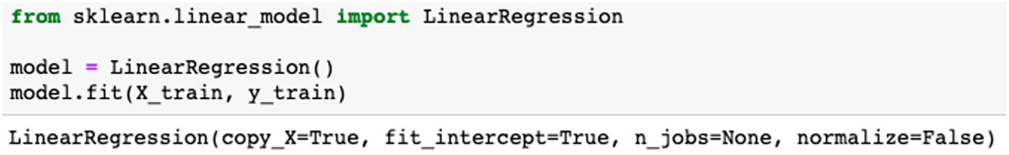
[39] "Domain Adaptation / Sample Selection Bias"
https://rehanguha.github.io//articles/2019-06/domain-adaptation.



```
plt.scatter(x=X_test,y=y_test, c="Blue")
plt.xlabel("petal_length"); plt.ylabel("petal_width");
```

Figure 3.45: Visualizing testing data

Seeing the test data and comparing it with the train data, we see a similar pattern. So, we can make a hypothesis that the model will work fine, given the linear nature of the data, test and train data follows the same linear pattern. So, now let us train the model.

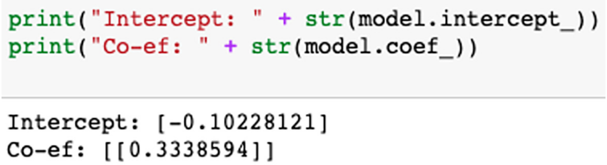


```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Figure 3.46: Modeling linear regression

Training the model is pretty straightforward, as shown above. From the mathematical equation shown previously, we are hoping to find the intercept and the coefficient values. Let us print the values which are computed by the sklearn linear regression model.

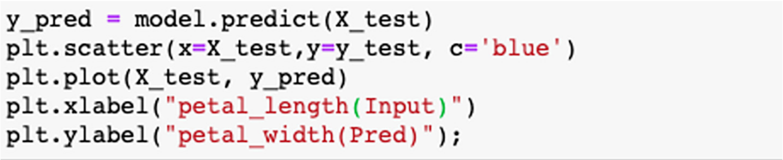


```
print("Intercept: " + str(model.intercept_))
print("Co-ef: " + str(model.coef_))
```

```
Intercept: [-0.10228121]
Co-ef: [[0.3338594]]
```

Figure 3.47: Values for the unknown variables

With the intercept and the coefficient values, we can plot a line. We can overlay the line on top of test data points and see where the predicted points lie.



```
y_pred = model.predict(X_test)
plt.scatter(x=X_test,y=y_test, c='blue')
plt.plot(X_test, y_pred)
plt.xlabel("petal_length(Input)")
plt.ylabel("petal_width(Pred)");
```

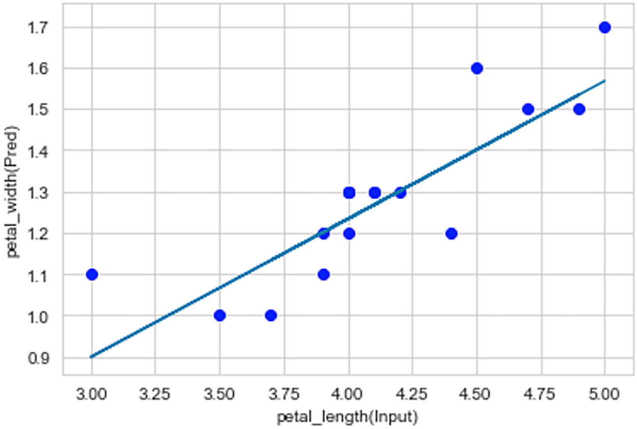
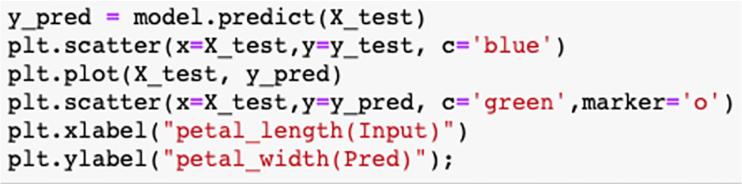


Figure 3.48: *Plotting the predicted regression line*

As discussed earlier that all the predicted points will lie on the line with the given intercept and co-efficient from *Figure 3.45*. We will confirm that hypothesis by plotting the predicted data points.



```
y_pred = model.predict(X_test)
plt.scatter(x=X_test,y=y_test, c='blue')
plt.plot(X_test, y_pred)
plt.scatter(x=X_test,y=y_pred, c='green',marker='o')
plt.xlabel("petal_length(Input)")
plt.ylabel("petal_width(Pred)");
```

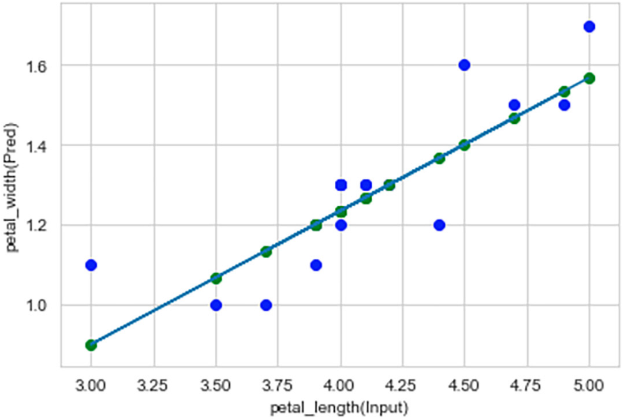
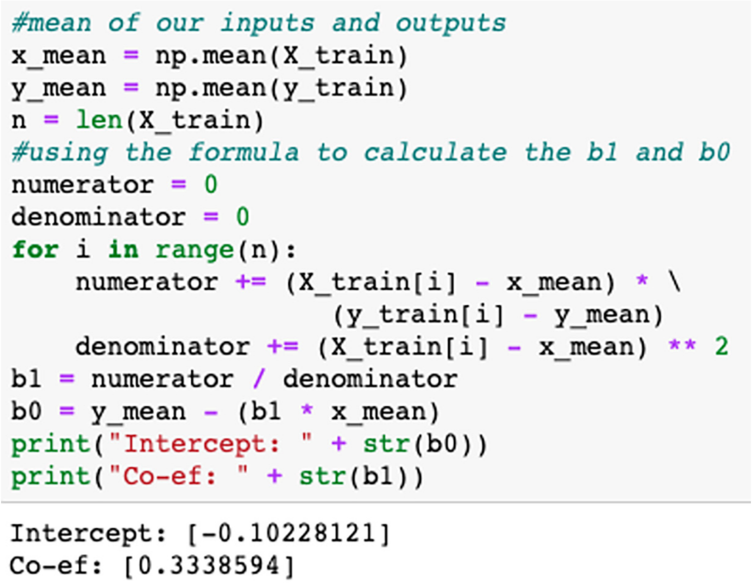


Figure 3.49: *Predicted values lie on the regression line*

It is confirmed that all the data points are on the line, and the difference between the original value and predicted value gives us the error. The logic can be implemented without using the sklearn package, as shown below.



```
#mean of our inputs and outputs
x_mean = np.mean(X_train)
y_mean = np.mean(y_train)
n = len(X_train)
#using the formula to calculate the b1 and b0
numerator = 0
denominator = 0
for i in range(n):
    numerator += (X_train[i] - x_mean) * \
                    (y_train[i] - y_mean)
    denominator += (X_train[i] - x_mean) ** 2
b1 = numerator / denominator
b0 = y_mean - (b1 * x_mean)
print("Intercept: " + str(b0))
print("Co-ef: " + str(b1))
```

```
Intercept: [-0.10228121]
Co-ef: [0.3338594]
```

Figure 3.50: *K-means without any package*

Now we have implemented linear regression, and we can compare the values of intercept and the coefficient, and they are the same.

So, by now, we know how linear regression works, and it is implemented. In the next section, we will see some error evaluation technique(s).

Evaluation

We now need to evaluate the quality of the model. Lower the error better the model. As this is a regression problem, we will know the desired output. With the desired output and the predicted output, we can find the error in the prediction. The below image shows what the error is.

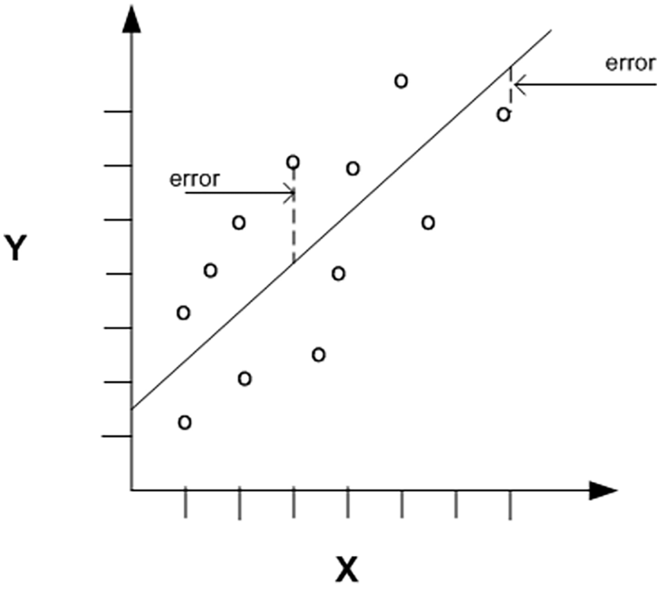


Figure 3.51: *Method to find an error*

As we know, the predicted values will lie on the line, so the error with the desired output is the perpendicular projection on the line. We will see only a handful of error metrics here.

Mean Absolute Error[40]

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Mean Squared Error[41]

It is the average of the squared error of the prediction.

$$MSE = \frac{1}{n}\sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$$

Root Mean Squared Error[42]

Similarly, it is the Root of MSE

$$RMSE = \sqrt{\frac{1}{n}\sum\nolimits_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2}$$

[40] "Mean absolute error - Wikipedia." https://en.wikipedia.org/wiki/Mean_absolute_error.

[41] "Mean squared error - Wikipedia." https://en.wikipedia.org/wiki/Mean_squared_error.

[42] "Root mean square - Wikipedia." https://en.wikipedia.org/wiki/Root_mean_square.

R^2 Score or Coefficient of Determination[43]

R^2 score gives us a statistical measure of how good the predictions approximate the original values. The coefficient of determination value of 1 represents a perfect fit with the data points.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

where,

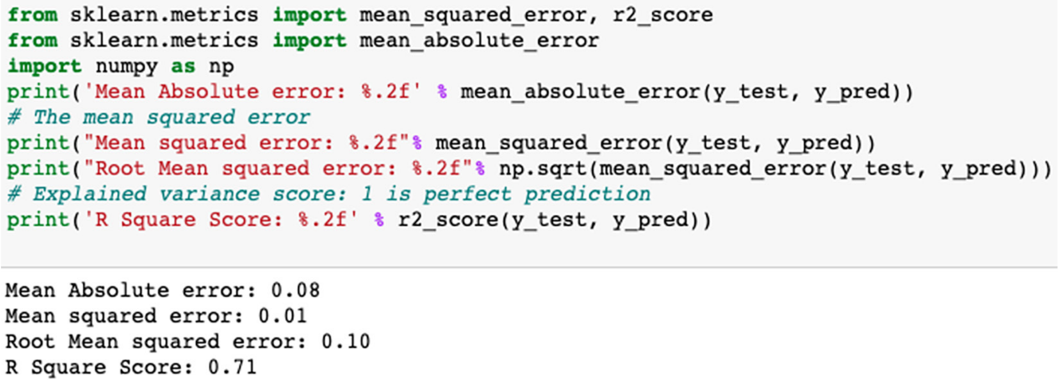
SS_{res} i.e., Residual Sum of Squares[44]

$$SS_{res} = \sum_i \left(y_i - \hat{y}_i\right)^2$$

and, SS_{tot} is the total sum of squares[45] (it is proportional to the variance of the data)

$$SS_{tot} = \sum_i \left(y_i - \bar{y}_i\right)^2$$

Now, let us see all the error metrics, and we are expecting fewer errors as per our hypothesis.



```python
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
import numpy as np
print('Mean Absolute error: %.2f' % mean_absolute_error(y_test, y_pred))
# The mean squared error
print("Mean squared error: %.2f"% mean_squared_error(y_test, y_pred))
print("Root Mean squared error: %.2f"% np.sqrt(mean_squared_error(y_test, y_pred)))
# Explained variance score: 1 is perfect prediction
print('R Square Score: %.2f' % r2_score(y_test, y_pred))
```

```
Mean Absolute error: 0.08
Mean squared error: 0.01
Root Mean squared error: 0.10
R Square Score: 0.71
```

Figure 3.52: *Different error metrics for evaluation*

The error for the model is extremely low, and that's a good sign. But sometimes, if the error is too less, then there can be a problem of over-fitting.

[43] "Coefficient of determination - Wikipedia." https://en.wikipedia.org/wiki/Coefficient_of_determination.

[44] "Residual sum of squares - Wikipedia." https://en.wikipedia.org/wiki/Residual_sum_of_squares.

[45] "Total sum of squares - Wikipedia." https://en.wikipedia.org/wiki/Total_sum_of_squares.

Below we have some images that represent the problem of under-fitting, fit, and over-fitting[46].

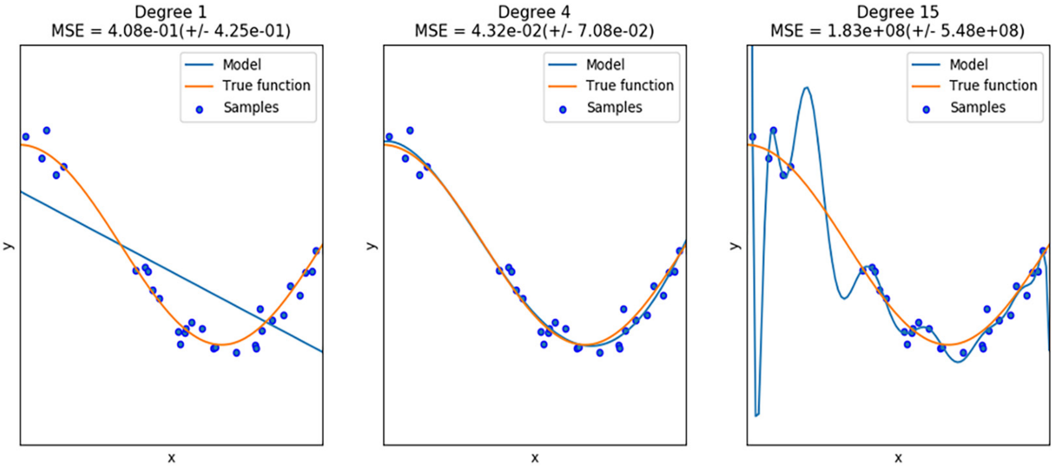


Figure 3.53: *Under-Fitting vs. Fit vs. Over-Fitting*

When the model understands all the points, and it predicts accordingly (Degree 15 polynomial equation from the image) (Generally higher degree of the polynomial equation), then it is said to over-fit the data.

On the other hand, when the model does not understand the underlying pattern on the training dataset, then it is referred to as an under-fitting (Generally lower degree of the polynomial equation).

The optimal solution for this problem is a fit model where the model identifies the underlying data pattern. Still, it does not fit through all the training data points that model will be referred to as a fit model. With this, we will conclude this chapter.

Further reading

In future chapters, we will mostly cover classification related problems, which will be applicable for both regression and clustering problems. But it is advised to go through some more algorithms related to regression and clustering. With the above knowledge, we can explore some more algorithms(the behavior of the algorithm will change but the functionality won't) as listed:

[46] "Overfitting - Wikipedia." https://en.wikipedia.org/wiki/Overfitting.

Regression:

1. Lasso regression
2. Multivariate regression
3. ElasticNet regression
4. Ridge regression
5. Stepwise regression

Clustering:

1. Expectations – Maximization clustering using Gaussian Mixture Models (GMM)
2. Agglomerative Hierarchical Clustering
3. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
4. Mean-Shift clustering

Others:

1. Auto Encoder
2. Machine Translation (Natural Language Processing -NLP)
3. Transcription
4. Image Denoising, etc.

Conclusion

This chapter mostly covered the basics of machine learning, how to handle different scenarios and play with the data, data preparation, last but not least, Machine Learning Modeling. We need to know one thing very clearly that Data Preparation, Data Pre-processing takes up 60%-70% of the entire ML Project.

In this chapter, we have introduced the basics of Machine Learning and some rudimentary algorithms for different types of ML problems. If someone has implemented the code, then they can easily start analyzing simple structured data and draw some insights from it and then use ML concepts to create a model as per the type of problem.

Next chapter, we will see different ML algorithms, different datasets, and put ourselves into different situation w.r.t. Data. Iris dataset was a basic dataset or a steppingstone to the world of Machine Learning problems. Later we will see real-world data, which will solve practical problems.

Chapter 4
Credit Card Fraud Detection

Introduction

Credit Card Fraud[1] **Detection** is one of the hottest topics of Machine Learning. For simplicity, we can compare this same concept with any financial transaction[2]. This chapter will cover various techniques to handle skewed data, and some relevant machine learning models to address this problem.

This chapter will give the readers the skill to handle the features just with their statistical analysis, when domain knowledge (that information of the feature is extremely important when it is available) information is not available. Readers will also find ways to handle a skewed dataset and how to draw information out of it. Finally building the right model for the same.

This skill is to understand different types of data and making an ML model is one of the crucial skills for any ML practitioner. It is important to know that data crunching concepts are **"The Skill"** to have for this role.

[1] "Credit card fraud - Wikipedia." https://en.wikipedia.org/wiki/Credit_card_fraud.

[2] "Financial transaction - Wikipedia." https://en.wikipedia.org/wiki/Financial_transaction.

Structure

- Let's understand the data
- Prerequisites
- Data analysis
- Modeling

Objective

This chapter won't follow a proper structure as a Machine Learning pipeline. Instead, this section will be how I being a Machine Learning Researcher, handle a project given a problem. The structure will be a representation of how I work and handle and not exactly what I do. In real life scenarios we go back and forth multiple times for a single analysis or step, and representing that will make the chapter a bit haphazard. I will try my level best to represent my working style, and every ML Practitioner has her/his own style to tackle a problem. But if we broadly categorize the style, then all the Machine Learning pipeline blocks will be covered. Here the dataset used won't be usual so, handling it won't be straight forward. With all the above, we will also learn about handling an imbalanced dataset.

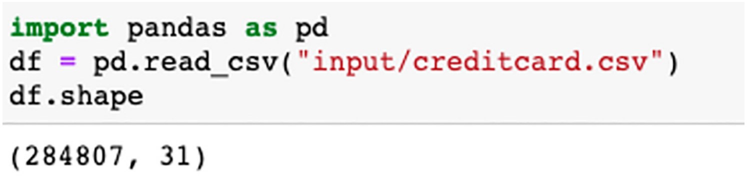
Let's understand the data

This is a credit card transaction data for two days of September 2013 for European cardholders. The statistics say there were a total of 492 frauds out of 2,84,807 transactions. This is a highly imbalanced dataset that we need to handle.

Source of the dataset:

The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

Let us start by loading the dataset to a Pandas DataFrame.



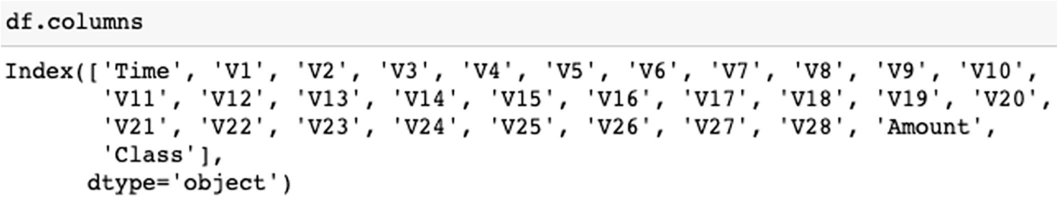
```
import pandas as pd
df = pd.read_csv("input/creditcard.csv")
df.shape
```

```
(284807, 31)
```

Figure 4.1: *Load dataset*

As we can see, the total number of columns is `31,` and the total number of records is `2,84,807`.

Below we have a code snippet to see all the columns.



```
df.columns
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

***Figure 4.2:** Dataset Columns*

Seeing all the columns, we can observe there is a column like “Time,” “Class,” “Amount,” “V1,”..., “V28”.

Column descriptions

- **Time**: Number of seconds elapsed between this transaction and the first transaction in the dataset
- **Amount**: Transaction amount
- **Class**: 1 for fraudulent transactions, 0 non-fraudulent
- **V1..., V28**: These are the features of the dataset. It may be the result of a PCA Dimensionality reduction to protect user identities and sensitive features. But this is all that we have to work with.

“Class” is the feature that we need to predict from this dataset using all the 28 features, “Time” and “Amount.”

The features V1, ..., V28 are dimensionality reduction[3] of some original features. The problem with this feature is that we are not aware of what feature it is, so logically tackling the features; in this case, is not possible. Say, if we have a feature called “Name” and we wanted to predict whether a transaction is fraudulent or not. Here, we can clearly say that the feature “Name” is not correlated with the type of prediction. But, in the case of the dataset we are using, we have no clue about the feature.

In this case, we will only be dependent on statistical metrics and relations to choose the best features for predicting the desired class.

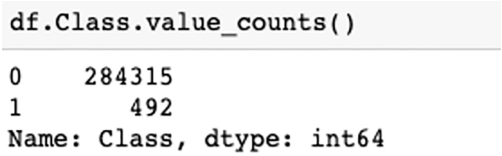
Now we will see another problem, i.e., imbalanced data, which we need to tackle for getting a good model for the problem space.

[3] “Dimensionality reduction - Wikipedia.” https://en.wikipedia.org/wiki/Dimensionality_reduction.

What is an imbalanced dataset?

An **imbalanced dataset** is when one output class has extremely high entries compared to the other output class. In this case, the positive class, i.e., fraudulent transaction class, has a few entries compared to the non-fraudulent/normal transaction class. So, the positive class(i.e., frauds) only accounts for 0.172% of the total transactions.

We will verify these numbers when the dataset is loaded.



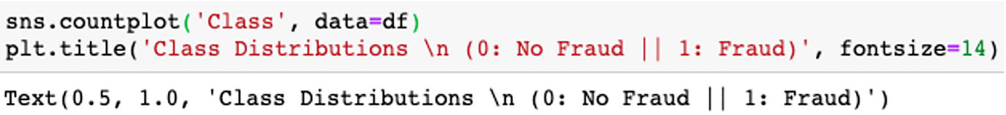
```
df.Class.value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

Figure 4.3: *Class Frequency*

It is evident that this is a highly imbalanced dataset, and among 2,84,807 total records, only 492 records are class 1, i.e., those records are a fraudulent transaction. So, we can confirm that the positive class only accounts for 0.172% of the entire dataset.

Now, let us visually see how imbalance the dataset is?



```
sns.countplot('Class', data=df)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
```

```
Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')
```

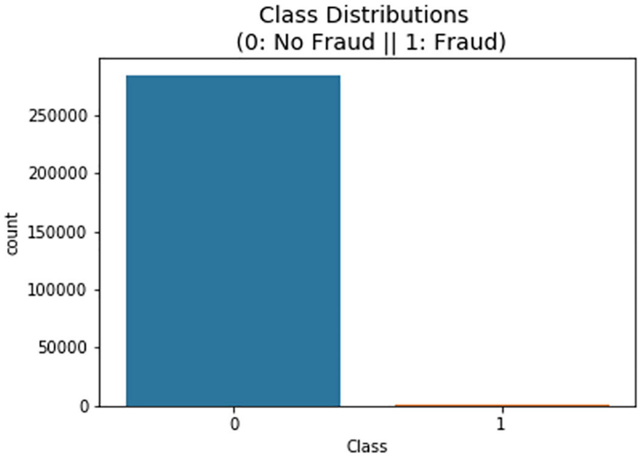
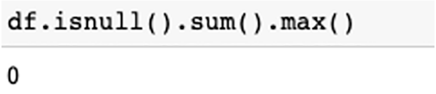


Figure 4.4: *Class distribution*

We can now see the difference visually and, the difference is so high that we can coin this dataset as a highly imbalanced dataset. Next, we can check for NULL values in the dataset.



```
df.isnull().sum().max()
```

```
0
```

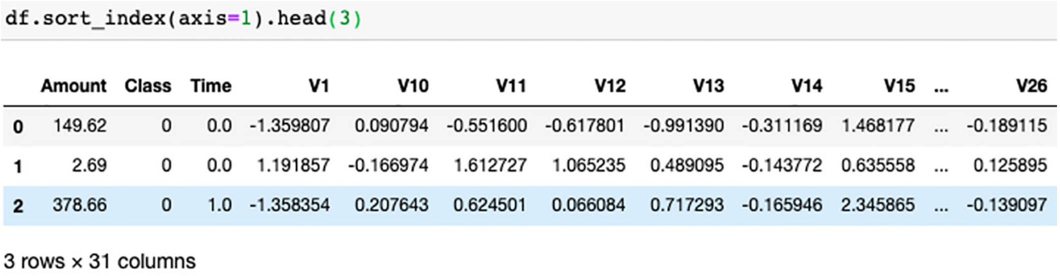
Figure 4.5: *NULL count*

Hopefully, we do not have any NULL values to handle in the dataset. So, going ahead, we will only take care of the imbalanced nature of the dataset and making sense of the features.

Knowing the features

Till now, we have seen many things about the dataset and its nature. In this section, we will analyze the features, know their types, and some similar information.

Let us see some of the values for all the features and know, how the value looks.



```
df.sort_index(axis=1).head(3)
```

	Amount	Class	Time	V1	V10	V11	V12	V13	V14	V15	...	V26
0	149.62	0	0.0	-1.359807	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177	...	-0.189115
1	2.69	0	0.0	1.191857	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558	...	0.125895
2	378.66	0	1.0	-1.358354	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865	...	-0.139097

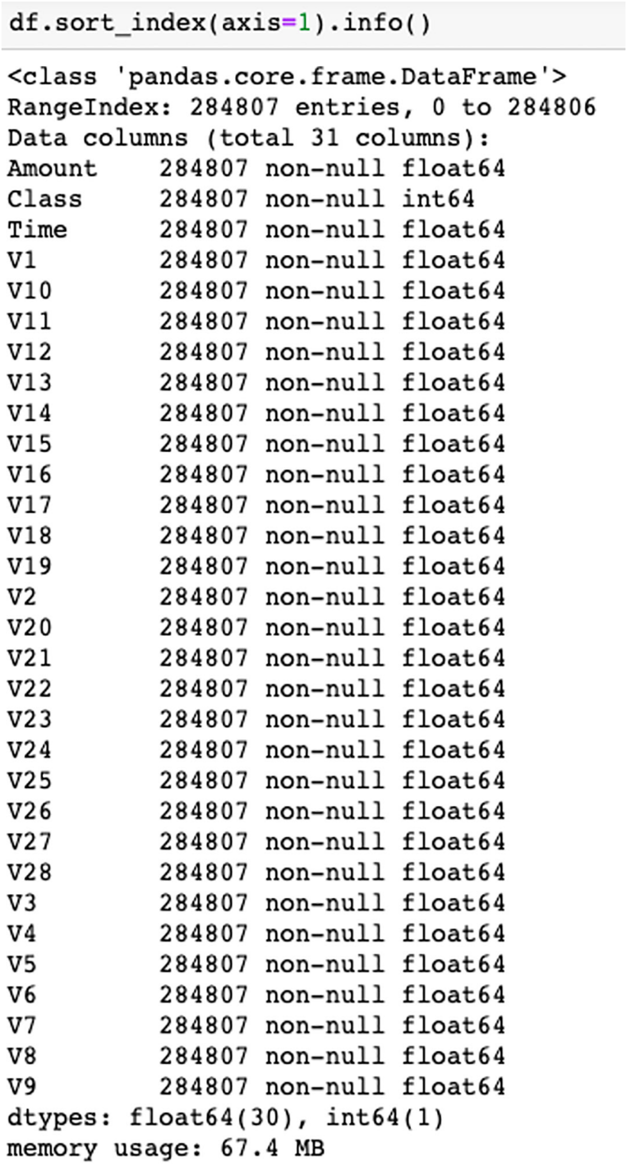
3 rows × 31 columns

Figure 4.6: *Dataset rows*

We can observe a few things from the above code output.

- Class is either 0 or 1
- An amount is a floating-point number (and it cannot be negative), any financial transaction cannot be negative
- Time as we know it will start from 0 and always be a positive integer
- V1...V28 can be negative or positive from the above *Figure 4.6*

Let us see all the information about the dataframe now.



```
df.sort_index(axis=1).info()
```

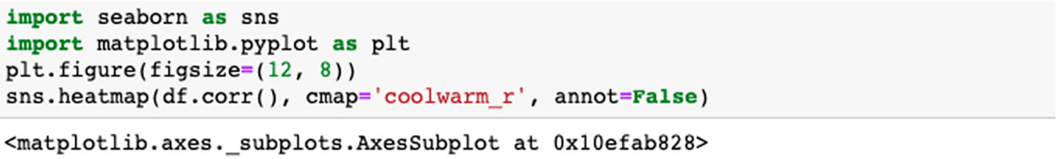
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Amount    284807 non-null float64
Class     284807 non-null int64
Time      284807 non-null float64
V1        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V2        284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Figure 4.7: Dataset Information

We can see it's a relatively large dataset compared to the previous dataset used. All of them are non-null, so we don't have to take care of null values for this dataset. Training a model with all the features generally don't make sense. We need to choose the features in such a way that those features will have some contribution to the decision of the output class.

Correlation of the dataset will give us a rough idea of what we are dealing with.

We will try to see a correlation plot with a heat map to figure out the highly correlated and non-correlated features.



```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), cmap='coolwarm_r', annot=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10efab828>
```

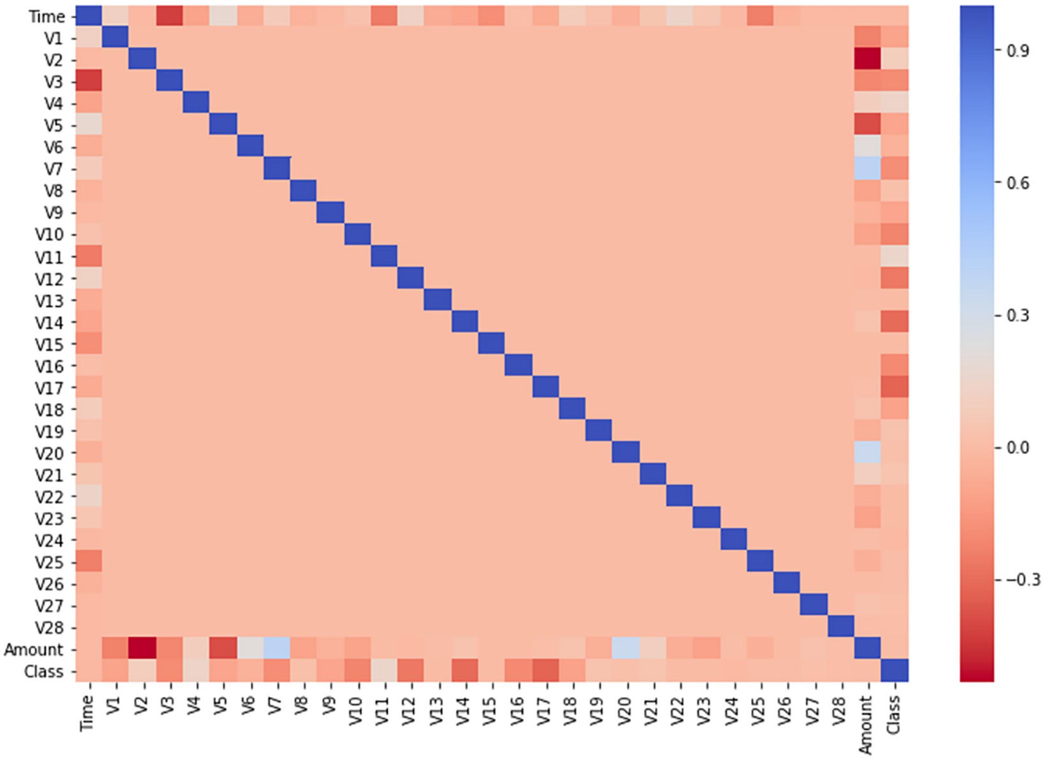


Figure 4.8: Correlation of heat map

We now need to find some features among 28 dimensionally reduced features that we can use to train the model. We went with the heat map and not the table because seeing the numeric table will be confusing for everyone, and the number of features is so high tracking them is a real problem. But anything visually gives us higher information compared to reading through a large table.

Let us take some time to analyze the heat map.

Here red is negatively correlated, and blue is positively correlated, and peach represents not correlated.

For simplicity, we will only compare the correlation against "`Class`," because in the end, we will try to predict the "`Class`" of the record.

If we analyze the features, then we can eliminate some of the features seeing the correlation between the features. In this case, we take features that are not highly correlated and zero as well. For this dataset, we will use both linear and nonlinear models.

I will again reiterate that we select a correlation based on the model and what we try to achieve. Taking multiple highly correlated features won't make any sense. We should always consider the entire correlation matrix and see how each correlation of the feature is linked. For example, we know feature A is highly correlated with the output class. And, B is highly correlated with A. Therefore, A and B are linearly dependent on the output class. So, taking any one of them will do. The more someone plays on this space, the more efficient one will become in terms of feature selection.

Features like V1, V3, V5, V7, V9, V10, V12, V14, V16, V17, V18 have some relation with "`Class`" compared to the other features.

But now there are some interesting things which we need to take care of. All the 28 features are dimensionally reduced features through **Principal Component Analysis** (**PCA**). Before PCA is applied, the dataset is first normalized. So, we need to apply normalization to rest of the columns as well. We will see all the concepts in the later section.

Prerequisites

We will introduce some of the concepts which we will use in this chapter. These topics, which will be discussed, are widely used in the Machine Learning domain. Normalization, Cross-Validation, PCA all play an important role in handling the data, and it has a significant effect on the result/output from the model.

Normalization[4]/Feature scaling[5]

Normalization is used in a variety of ways in statistics. It is also known as **feature scaling**. The meaning we are going after, is the normalization of the range of the data to a standard scale.

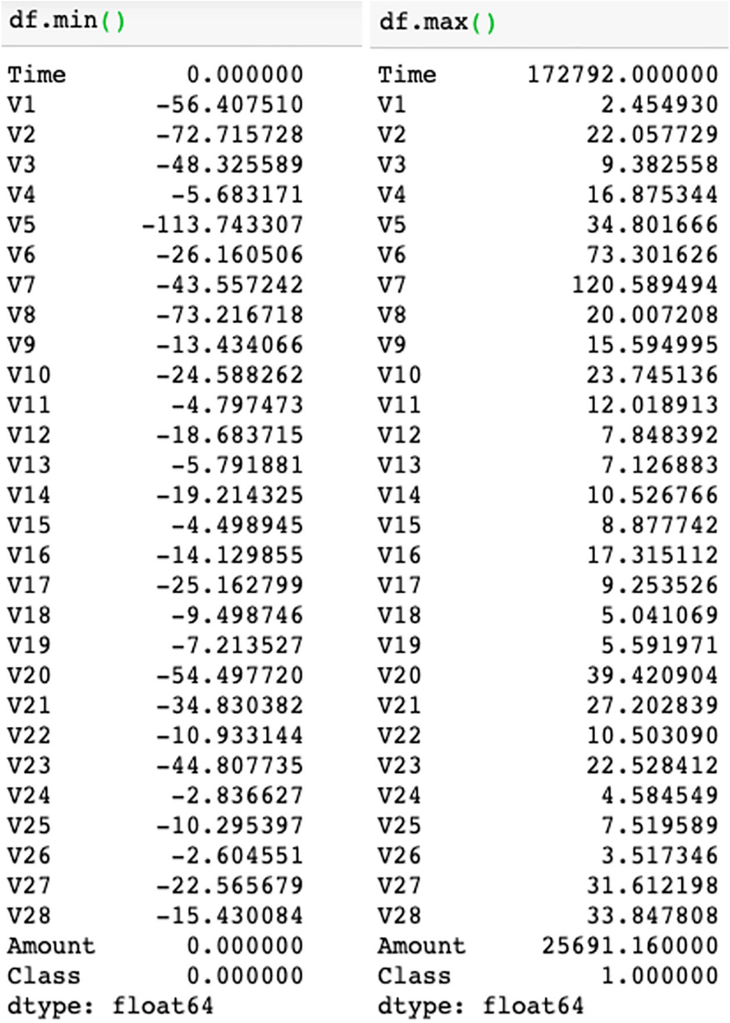
- **Case 1**: Say we have four lengths; all of them are in centimeters, and only one of them is inches. So, in those cases, we need to follow one standard and generalize all the length units.

[4] "Normalization - Wikipedia." https://en.wikipedia.org/wiki/Normalization.

[5] "Feature scaling - Wikipedia." https://en.wikipedia.org/wiki/Feature_scaling.

- **Case 2**: We have two features like Age and Salary, and we can easily say that both of the features will have a different range. Hence, we will use normalization/scaling techniques to bring them on the same scale.

As seen in the previous section, that all the values had a different range. Let's consolidate those values and see their ranges for all the features.



```
df.min()
```

```
Time        0.000000
V1        -56.407510
V2        -72.715728
V3        -48.325589
V4         -5.683171
V5       -113.743307
V6        -26.160506
V7        -43.557242
V8        -73.216718
V9        -13.434066
V10       -24.588262
V11        -4.797473
V12       -18.683715
V13        -5.791881
V14       -19.214325
V15        -4.498945
V16       -14.129855
V17       -25.162799
V18        -9.498746
V19        -7.213527
V20       -54.497720
V21       -34.830382
V22       -10.933144
V23       -44.807735
V24        -2.836627
V25       -10.295397
V26        -2.604551
V27       -22.565679
V28       -15.430084
Amount      0.000000
Class       0.000000
dtype: float64
```

```
df.max()
```

```
Time      172792.000000
V1             2.454930
V2            22.057729
V3             9.382558
V4            16.875344
V5            34.801666
V6            73.301626
V7           120.589494
V8            20.007208
V9            15.594995
V10           23.745136
V11           12.018913
V12            7.848392
V13            7.126883
V14           10.526766
V15            8.877742
V16           17.315112
V17            9.253526
V18            5.041069
V19            5.591971
V20           39.420904
V21           27.202839
V22           10.503090
V23           22.528412
V24            4.584549
V25            7.519589
V26            3.517346
V27           31.612198
V28           33.847808
Amount     25691.160000
Class          1.000000
dtype: float64
```

Figure 4.9: *Range for all columns*

From the above image, we can see the max and min of the feature side by side. We need to apply some scaling concepts to standardize/normalize the dataset.

Seeing the feature ranges, we can see "`Time`" and "`Amount`" has not been scaled.

To gain more understanding about the dataset, we need to understand different types of scaling techniques that will help us to deal with the dataset.

Min-max feature scaling (Rescaling)

This is one of the most common, important, and simple scaling algorithms. Here we scale the entire column with any given range into a usable given range [*a*, *b*].

$$x' = a + \frac{(x_i - min(x))(b - a)}{max(x) - min(x)}$$

The above equation will iterate through all the elements and scale with the columns/ list's min, max values.

Generally, we scale the list/series into the range [0, 1].

Therefore, $a = 0$ and $b = 1$, replacing them will give us the below equation.

$$x' = \frac{x_i - min(x)}{max(x) - min(x)}$$

When the min-max feature is generally used, the above equation is referred to with range [0, 1].

But there can be some cases where we need to scale down to a given range.

> ***Note:*** Scaling down using mix-max feature scaling, won't change the distribution of the feature. It only changes the range and keeping the distribution the same for the feature.

Mean normalization

We have another variety of scaling, i.e., **Mean normalization,** and the mathematical formula for it is:

$$x' = \frac{x_i - \overline{x}}{max(x) - min(x)}$$

There can be situations where this can be used but, generally, it is used less compared to the other scaling techniques. We have another form of normalization, which can be achieved by tweaking the above equation.

Standardization (Z-score normalization)

This is one of the most popular normalization methods. In the process of **standardization**, each feature has zero-mean and unit variance or standard deviation.

We can achieve that using the below equation:

$$x' = \frac{x_i - \overline{x}}{\sigma}$$

Note: Scaling down using standardization will change the distribution of the feature and try to make mean close to zero, and the standard deviation equals to one.

Principal component analysis[6]

Principal component analysis (**PCA**) is a statistical method to explain variance and covariance structure of a set of variables through linear combination. It uses the concept of orthogonal transformation[7] to convert the set of variables (mostly correlated variables) into a set of linearly uncorrelated values[8], i.e. also known as Principal component.

We will see briefly in the form of an algorithm of how it works:

- **Step 1**: Get the data in the form of *m x n* matrix
- **Step 2**: Standardize the matrix feature-wise
- **Step 3**: Calculate the Covariance Matrix[9]
- **Step 4**: Calculate Eigenvectors[10] and Eigenvalues[11] of the Covariance Matrix, i.e., Eigen decomposition[12] [13] of the Covariance Matrix
- **Step 5**: Choose the Principal Components and form a feature vector
- **Step 6**: Deriving the new dataset and form the clusters

PCA is one of the popular methods to perform dimension reduction on a large dataset, and it helps in multiple ways like visualization, handling a smaller number of columns/features for modeling. I will recommend going through the mathematics of PCA for a better understanding of how it works, and it is one of the important and rudimentary concepts of Machine Learning.

[6] "Principal component analysis - Wikipedia." https://en.wikipedia.org/wiki/Principal_component_analysis.

[7] "Orthogonal transformation - Wikipedia." https://en.wikipedia.org/wiki/Orthogonal_transformation.

[8] "Correlation and dependence - Wikipedia." https://en.wikipedia.org/wiki/Correlation_and_dependence.

[9] "Covariance matrix - Wikipedia." https://en.wikipedia.org/wiki/Covariance_matrix.

[10] "Eigenvector -- from Wolfram MathWorld." http://mathworld.wolfram.com/Eigenvector.html.

[11] "Eigenvalue -- from Wolfram MathWorld." http://mathworld.wolfram.com/Eigenvalue.html.

[12] "Eigen Decomposition -- from Wolfram MathWorld." http://mathworld.wolfram.com/EigenDecomposition.html.

[13] "Eigendecomposition of a matrix - Wikipedia." https://en.wikipedia.org/wiki/Eigendecomposition_of_a_matrix.

Cross-validation

Cross-validation is a technique by which we can assess how it will perform in practice, i.e., in real-time data. Cross-validation is a model validation technique to ensure its performance, but it only works best when the training dataset distribution is similar to real-time data distribution.

There are different types of cross-validation techniques:

- Exhaustive cross-validation
    - Leave-p-out cross-validation[14]
    - Leave-one-out cross-validation
- Non-exhaustive cross-validation
    - k-fold cross-validation
    - Holdout method
    - Repeated random sub-sampling validation[15]

We will take a look at the k-fold cross-validation technique as that is one of the most popular validation techniques.

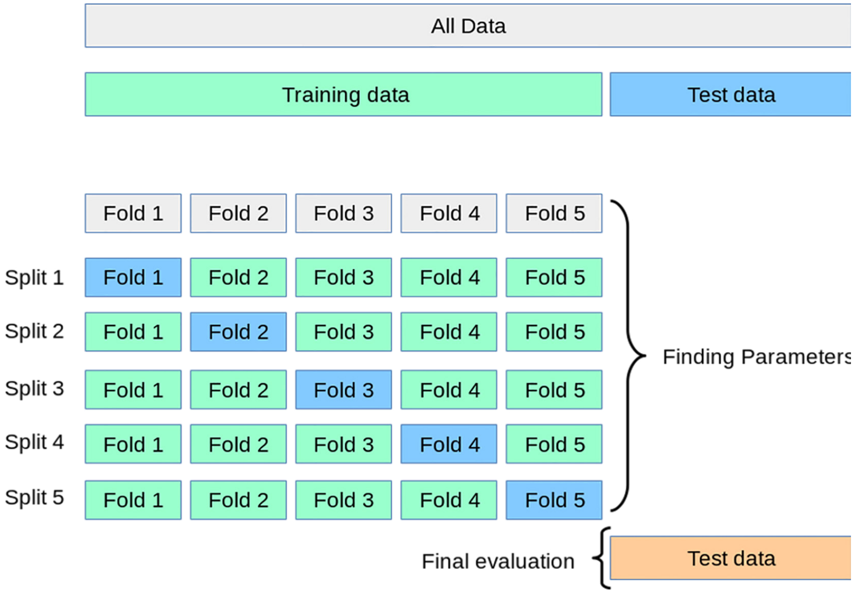


***Figure 4.10:** k-fold cross-validation (Source: scikit-learn)*

[14] "sklearn.model_selection.LeavePOut — scikit-learn 0.22 ...." http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeavePOut.html.

[15] "Lecture 13: Validation." http://research.cs.tamu.edu/prism/lectures/iss/iss_l13.pdf.

The above diagram shows the image for k-fold cross validation where $k = 5$.

We can see from the above image that we divide the training data into k folds or k sections. Then each section is selected once as a validation dataset, and rest sections are used for training. Due to the folding concepts, the models are less prone to overfitting.

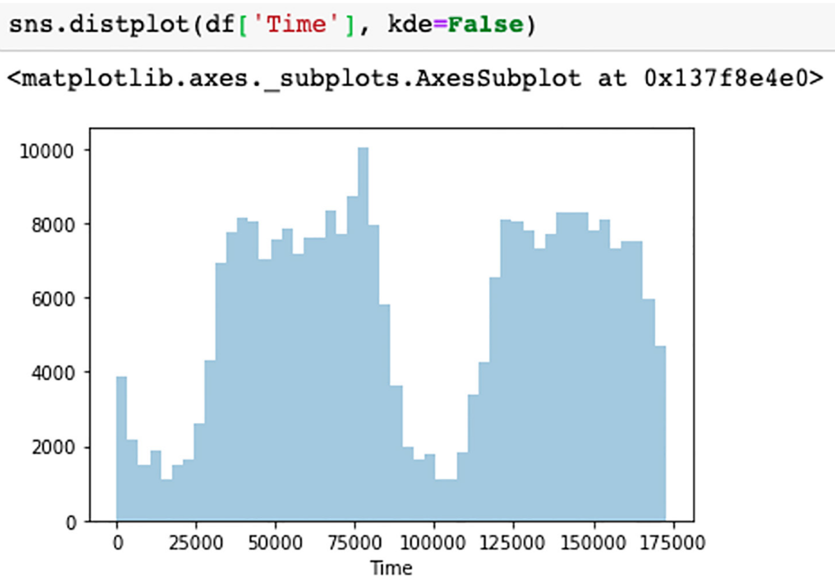
The goal of cross-validation techniques help the model to solve problems like overfitting or selection bias and gives an insight into how the model will generalize to an independent/unknown dataset.

Data analysis

We need to work on the data and analyze all the features and later choose the best features for predicting the class.

Let us start by seeing some distribution of the features and analyze them.

Previously, we have seen "`Time`" was not scaled, and the range for that feature was [0,172792]. We can check the distribution of the feature and see how it looks.



```
sns.distplot(df['Time'], kde=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x137f8e4e0>
```

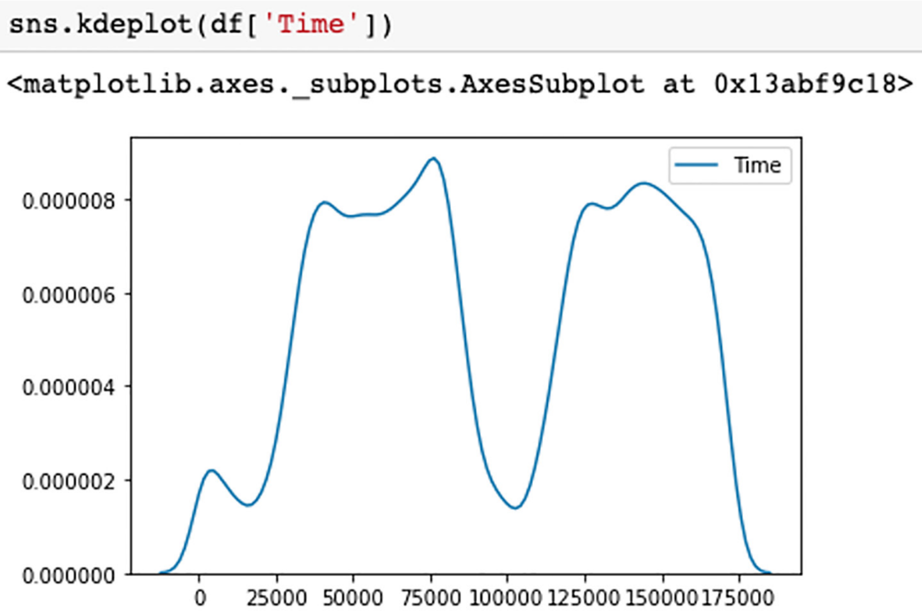
Figure 4.11: *Bi-modal "Time" distribution plot*

Seeing the above distribution, we can see two spikes in transaction one at 50000 sec and another at the 13500-sec range. That means we can see there are two similar modes[16] in the distribution.

[16] "Mode (statistics) - Wikipedia." https://en.wikipedia.org/wiki/Mode_(statistics).

This is a Bimodal Distribution[17], and it is a continuous probability distribution with two different modes.

We can confirm the structure with KDE.



```
sns.kdeplot(df['Time'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13abf9c18>
```

Figure 4.12: *KDE for "Time"*

KDE clearly shows that there are two similar modes to this continuous distribution.

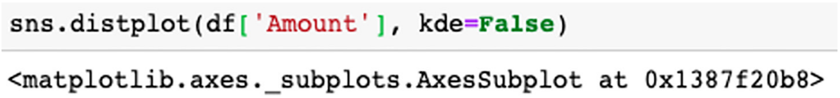
> ***Note:*** There can be more than two modes in a continuous probability distribution, and those are known as Multimodal Distribution[18].

We have seen the distribution for "`Time`," and now we need to see the distribution for the feature "`Amount`."

We are expecting a skewed distribution as we know it cannot be uniform because of the extremely high-value credit card transactions are rare compared to the average amount transaction.

[17] "Bimodal Distribution - STATISTICA Help." https://documentation.statsoft.com/STATISTICAHelp.aspx?path=Glossary/GlossaryTwo/B/BimodalDistribution

[18] "Multimodal distribution - Wikipedia." https://en.wikipedia.org/wiki/Multimodal_distribution.



```
sns.distplot(df['Amount'], kde=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1387f20b8>
```

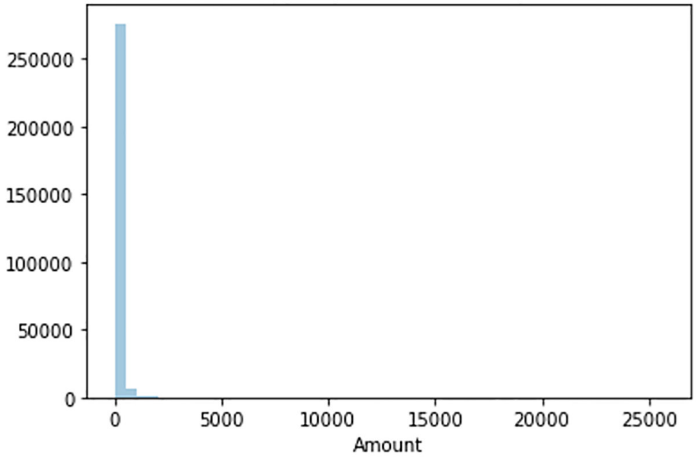
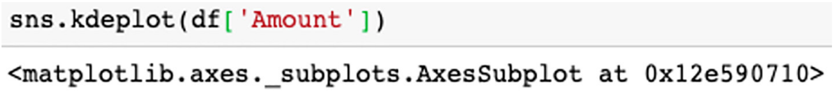


Figure 4.13: *"Amount" Frequency Distribution*

Seeing the distribution, we can confirm the hypothesis about the skewness. We see the mode for the distribution lies at the far left of the distribution, may be the mode will range from [1, 1000] for this distribution. But, this kind of distribution is not odd because it is an expected distribution for any transaction dataset.

We can see the KDE plot for the same as well.



```
sns.kdeplot(df['Amount'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12e590710>
```

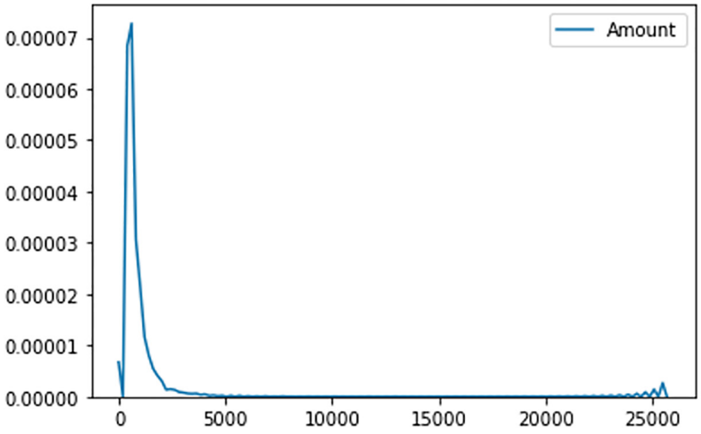
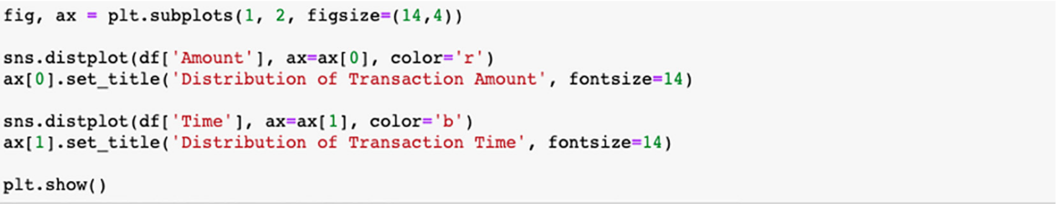


Figure 4.14: *KDE for "Amount"*

KDE plot also shows some density at the right end, and those are outliers. And for this business statement, the outliers are extremely important because there can be fraudulent transactions with any transaction amount.

We should try to see if there is any visual link/similarity among the distributions.



```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.distplot(df['Amount'], ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)

sns.distplot(df['Time'], ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)

plt.show()
```

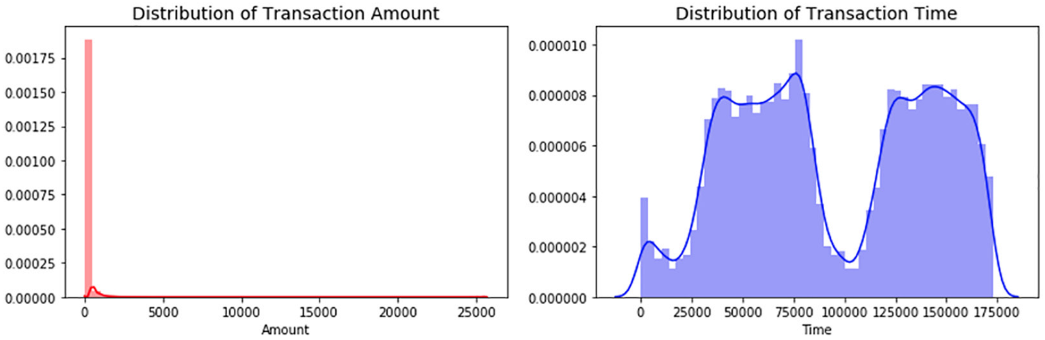
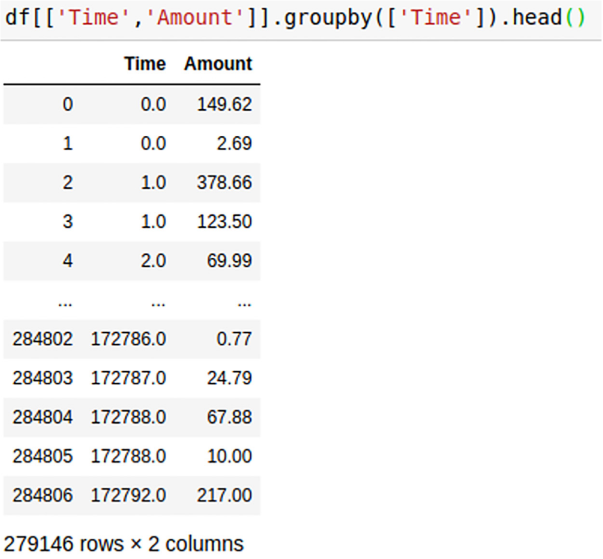


Figure 4.15: Comparison for an "Amount" and "Time" KDEs

Unfortunately, from the distribution, we cannot make any sense. So, we need to drop this idea to compare "`Time`" and "`Amount`" distribution separately.

It will make sense if we plot a chart between "`Time`" vs. "`Amount`." We should know that for every "time" what will be the "amounts" like it is shown above.



```
df[['Time','Amount']].groupby(['Time']).head()
```

	Time	Amount
0	0.0	149.62
1	0.0	2.69
2	1.0	378.66
3	1.0	123.50
4	2.0	69.99
...	...	...
284802	172786.0	0.77
284803	172787.0	24.79
284804	172788.0	67.88
284805	172788.0	10.00
284806	172792.0	217.00

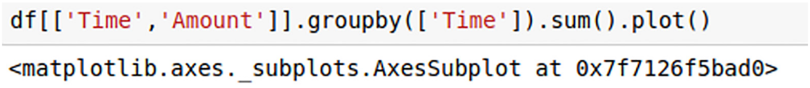
279146 rows × 2 columns

Figure 4.16: Summed "Amount" grouped by "Time."

For any plot, especially the bar plot, we cannot plot multiple values for the same time interval, so it's better to plot the sum of "`Amount`" for each of the time intervals.

We are expecting some time intervals, which will have a spike, in the sum of their total transaction amount.

Let us go ahead and plot such kind of bar plot.



```
df[['Time','Amount']].groupby(['Time']).sum().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7126f5bad0>
```

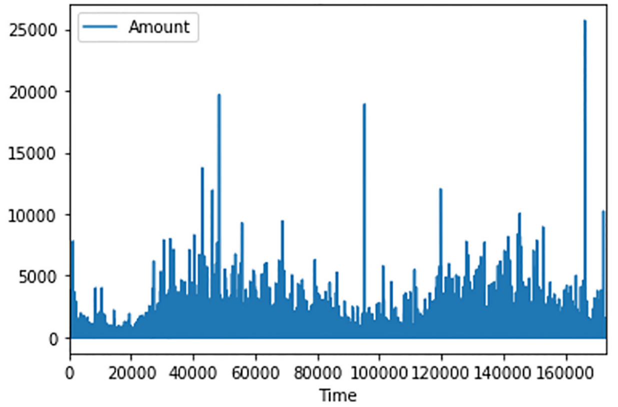
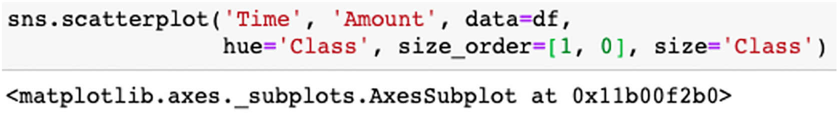


Figure 4.17: *Summed "Amount" Bar Plot*

This shows what we exactly, thought, so; this gives us a rough idea about the sum of the amount distributed.

Here, we can only see the frequency of the amounts for the given time. But, need a plot that will help us to see the fraudulent transactions and the range for amounts.



```
sns.scatterplot('Time', 'Amount', data=df,
                hue='Class', size_order=[1, 0], size='Class')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11b00f2b0>
```

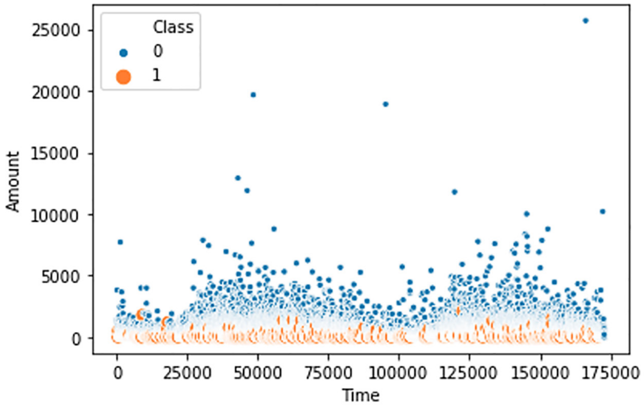
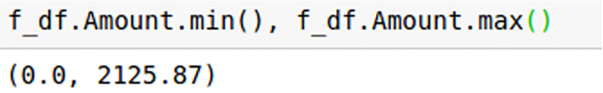


Figure 4.18: *Class-wise "Amount" vs. "Time" Scatter Plot*

The above figure shows the range for the fraudulent transaction to be around [0, 2500]. We can test this out and the range of the fraudulent transaction amount.

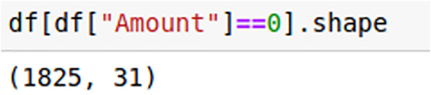


```
f_df.Amount.min(), f_df.Amount.max()
```

```
(0.0, 2125.87)
```

Figure 4.19: *Range for Fraudulent Class*

We can confirm the hypothesis seeing the range above. One more thing we can check and think about is, what are the total number of records whose amount value is Zero. Does it make sense to keep zero-valued transactions in the dataset? How can there be a fraud when the transaction value is zero?

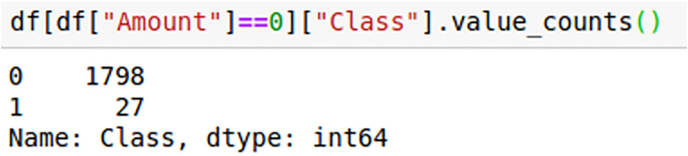


```
df[df["Amount"]==0].shape
```

```
(1825, 31)
```

Figure 4.20: *Count of records where "Amount" is zero*

We have a small chunk of records where "`Amount`" equals to zero. We need to see the breakdown down of "`Class`" as well and decide whether to keep it or ignore it.



```
df[df["Amount"]==0]["Class"].value_counts()
```

```
0    1798
1      27
Name: Class, dtype: int64
```

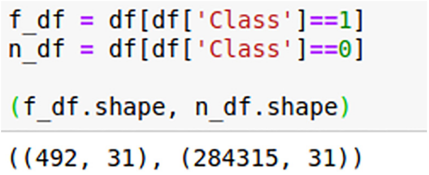
Figure 4.21: *Class-wise Zero "Amount" Transaction*

Seeing the breakdown, we can clearly say that it is not significant enough to impact the decision or prediction. But the question we raised about the zero amount transactions does not have a proper logical solution as we cannot back it up with any proof. The dataset description does not say much about this type of transaction, so the chances of removing this kind of transaction are pretty high.

We will see more distributions and later decide to keep it or not.

Let's think more about the dataset and see what can be done with it.

So, now we should try to plot distributions for all the class only for "`Time`" and "`Amount`." Before that, we need to split the dataset into two different data frames based on "`Class`."



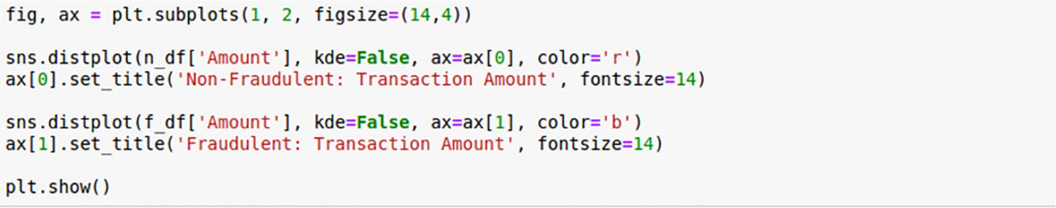
```
f_df = df[df['Class']==1]
n_df = df[df['Class']==0]

(f_df.shape, n_df.shape)
```

```
((492, 31), (284315, 31))
```

Figure 4.22: Shape of Fraudulent and Non-fraudulent Class

For some time, we will ignore those transaction zero records and consider the entire dataset. Seeing the above code, we now have two data frames, one containing all the fraudulent transactions and another containing all the non-fraudulent transactions.

With those, we can now individually plot the distributions. We can start with the "`Amount`" distribution.



```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.distplot(n_df['Amount'], kde=False, ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Amount', fontsize=14)

sns.distplot(f_df['Amount'], kde=False, ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Amount', fontsize=14)

plt.show()
```

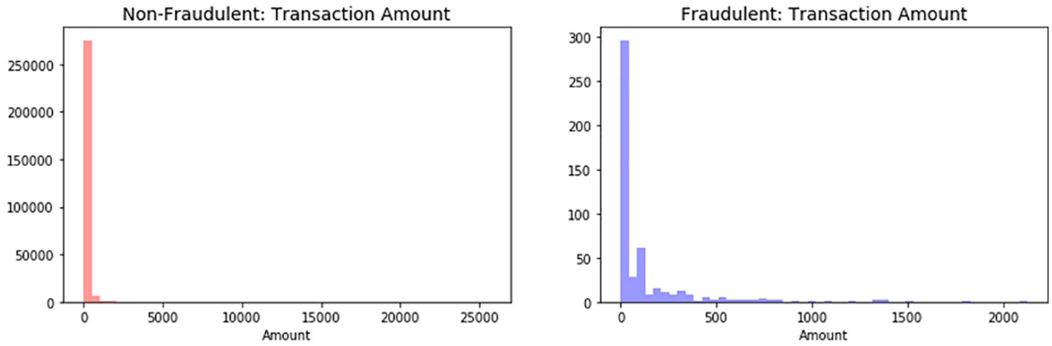
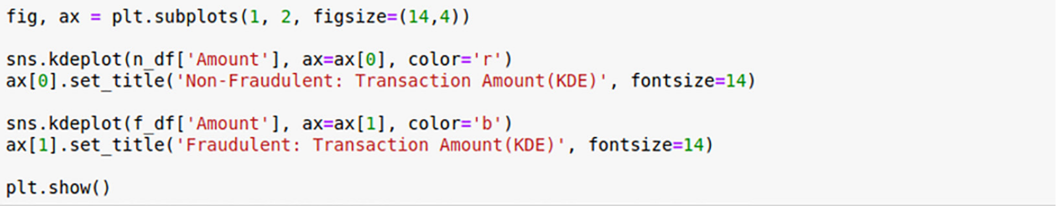


***Figure 4.23:** "Amount" Frequency Distribution for Fraudulent and Non-fraudulent Class*

Seeing the above image, we can see the distribution is same or nearly identical if we compare them. KDE will be same as well, which we can confirm by plotting it.



```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.kdeplot(n_df['Amount'], ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Amount(KDE)', fontsize=14)

sns.kdeplot(f_df['Amount'], ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Amount(KDE)', fontsize=14)

plt.show()
```

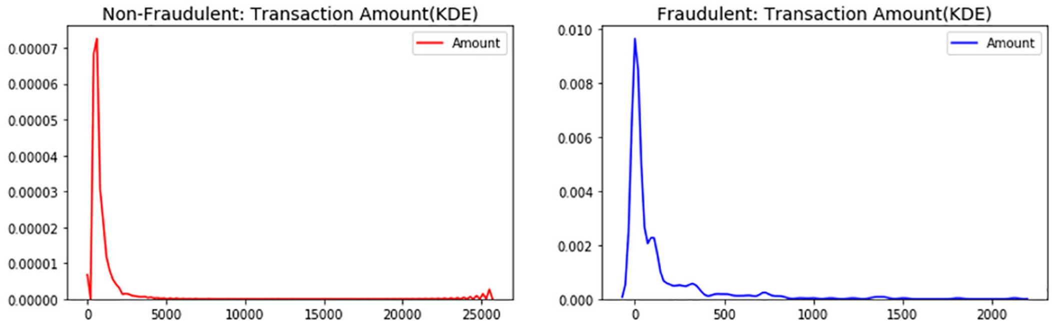
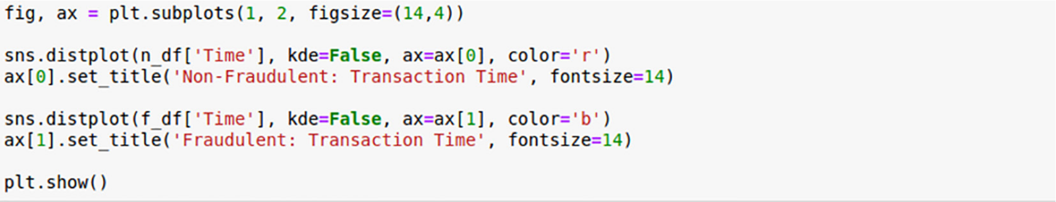


Figure 4.24: *"Amount" KDE for Fraudulent and Non-fraudulent Class*

The KDE is similar, and the distribution looks similar. The fraudulent transaction KDE is a bit spread compared to the non-fraudulent transaction but, it won't be of much use because the size of the dataset is small.

We need to perform something similar for "`Time`" as well and see the distribution.



```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.distplot(n_df['Time'], kde=False, ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Time', fontsize=14)

sns.distplot(f_df['Time'], kde=False, ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Time', fontsize=14)

plt.show()
```

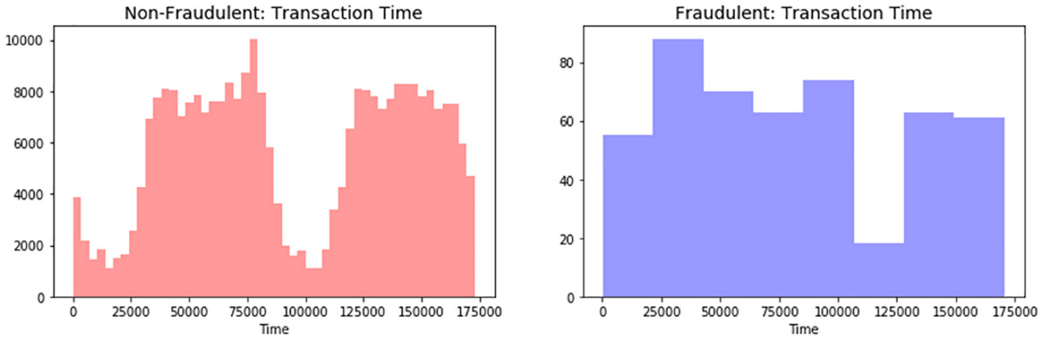
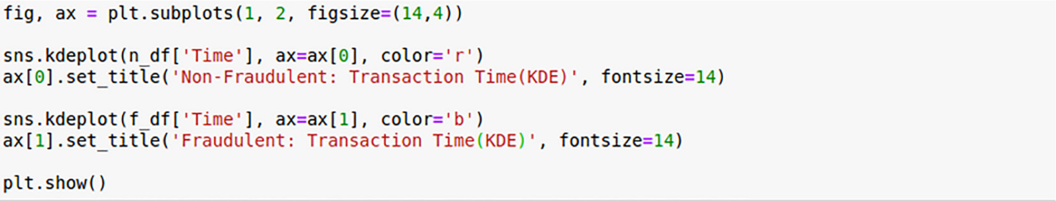


Figure 4.25: *"Time" Frequency Distribution for Fraudulent and Non-fraudulent Class*

Interestingly, the "`Time`" distribution for Fraudulent transaction is different. We can confirm and analyze after plotting the KDE.



```
fig, ax = plt.subplots(1, 2, figsize=(14,4))

sns.kdeplot(n_df['Time'], ax=ax[0], color='r')
ax[0].set_title('Non-Fraudulent: Transaction Time(KDE)', fontsize=14)

sns.kdeplot(f_df['Time'], ax=ax[1], color='b')
ax[1].set_title('Fraudulent: Transaction Time(KDE)', fontsize=14)

plt.show()
```

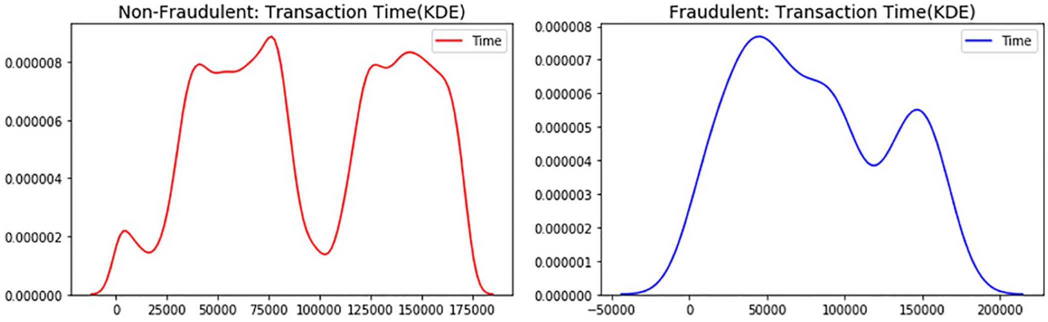


Figure 4.26: *"Time" KDE for Fraudulent and Non-fraudulent Class*

Seeing the above distribution, we can draw two conclusions:

1. Non-fraudulent transaction for both "`Time`" and "`Amount`" looks the same when compared with the entire population. That is because the size of the Fraudulent dataset is so less the impact is not significant enough to change the distribution.
2. Fraudulent transaction for "`Time`" looks different, and that has some significance, i.e., the density of the distribution is higher at the beginning of time, and there is only one statistical model to the distribution. We need to keep this in mind, when we scale the data and make sure this kind of pattern is not lost.

Lastly, for this section, we will see individual "`Class`" plots for "`Time`" v/s "`Total Amount`" as we have seen the *Figure 4.17*.

Let us go ahead and see the distribution for the non-fraudulent dataset.

```
n_df[['Time','Amount']].groupby(['Time']).sum().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7131404890>
```

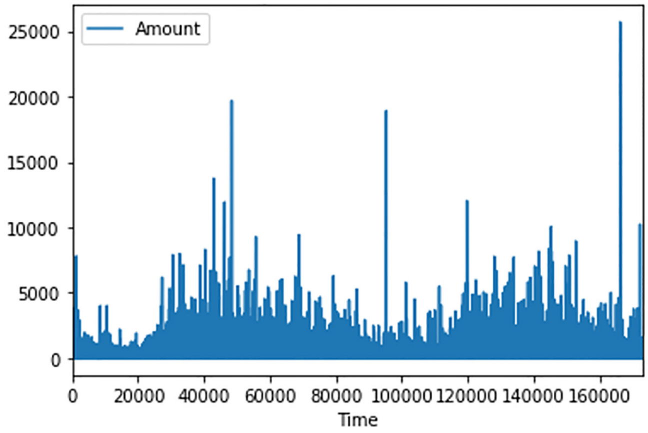


Figure 4.27: *Non-Fraudulent Summed "Amount" vs. "Time"*

As with the previous plots for this class, this is highly similar to the plot for the entire population. But we are expecting something different from the Fraudulent dataset as the distribution for the "`Time`" feature was a bit different.

```
f_df[['Time','Amount']].groupby(['Time']).sum().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7131643250>
```

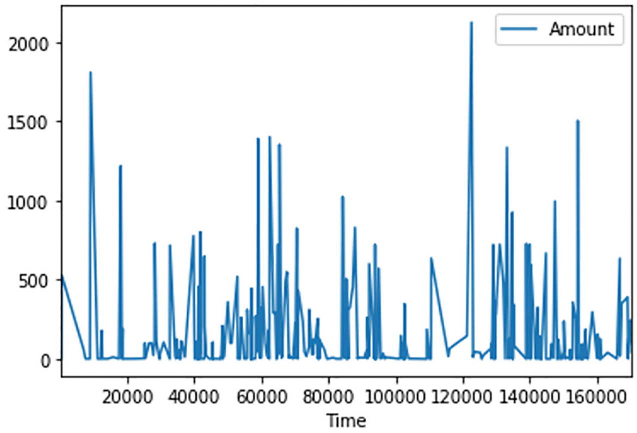
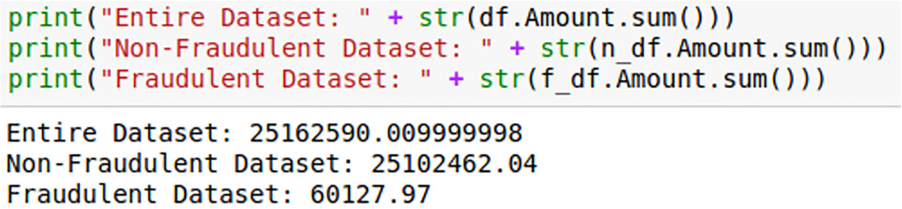


Figure 4.28: *Fraudulent Summed "Amount" vs. "Time"*

As we thought, the plot will be different from the hypothesis, and we find the hypothesis to be true. We can see that the sum of the amount is not that high compared to Non-Fraudulent one, and that is obvious but, we can say the transaction amount is not of high variance. We can confirm the total amount of different classes.

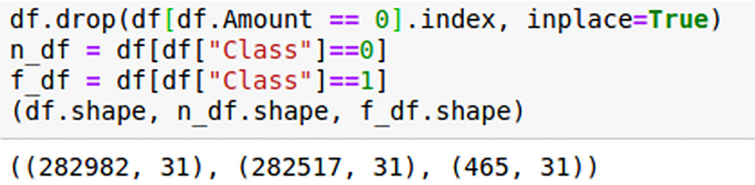


```
print("Entire Dataset: " + str(df.Amount.sum()))
print("Non-Fraudulent Dataset: " + str(n_df.Amount.sum()))
print("Fraudulent Dataset: " + str(f_df.Amount.sum()))
```

```
Entire Dataset: 25162590.009999998
Non-Fraudulent Dataset: 25102462.04
Fraudulent Dataset: 60127.97
```

Figure 4.29: *Total summed amount for different class*

We can see the total amount for each group and removing the zero-amount transaction would not affect the "`Amount`" distribution, and it won't change the "`Time`" distribution as the size of the exception is very low. So, it will be safe to remove those records.



```
df.drop(df[df.Amount == 0].index, inplace=True)
n_df = df[df["Class"]==0]
f_df = df[df["Class"]==1]
(df.shape, n_df.shape, f_df.shape)
```

```
((282982, 31), (282517, 31), (465, 31))
```

Figure 4.30: *Removing Zero "Amount" Transaction*

With the above code snippet, we have removed those transactions. Moving forward, we will use this dataset for all analyses and modeling.

Mostly, we have covered different analyses for different features. This step is extremely tedious, and I will be repeating the same thing over and over again because it is one of the most crucial parts of the pipeline. On top of that, this is an iterative process, i.e., we need to go through the same visualization again and again when we perform a slight change in the dataset. We should be well aware of the change, cause, and the impact for any tweak which is performed on the dataset.

This is one part of the data analysis, and we are yet to touch the features V1..., V28. Before that, to make the comparison, we need to perform some other operations as well so that the analysis is logical, and we could make some sense out of it.

We are yet to cover concepts like scaling/standardization, handling imbalance, data analysis.

Scaling[19]

We had talked about scaling and normalization before under the head “Prerequisites.” Those concepts will be used on “`Time`,” and “`Amount`” as those were the only columns that were not scaled. V1..., V28 features are scaled during the process of PCA itself. Now we need to make all the features same for the comparison.

Standard scaler[20]

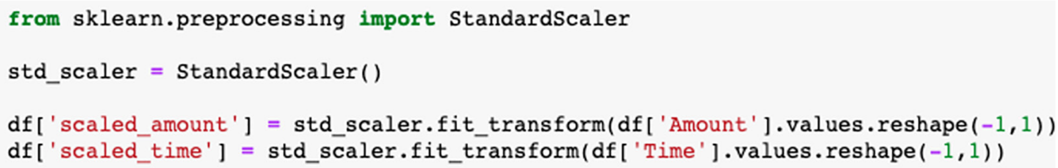
Standard scalar from the sklearn implements standardization/Z-score normalization.

$$x' = \frac{x_i - \overline{x}}{\sigma}$$

Where, $\overline{x}$ is the mean, and σ is the standard deviation.

We know from the data description that all the features from V1..., V28 are scaled as those are the output result of a Dimension Reduction Algorithm, i.e., PCA.

So, we will only implement standard scaling for “`Time`” and “`Amount`.”



```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()

df['scaled_amount'] = std_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = std_scaler.fit_transform(df['Time'].values.reshape(-1,1))
```

Figure 4.31: *Standard scaling the features*

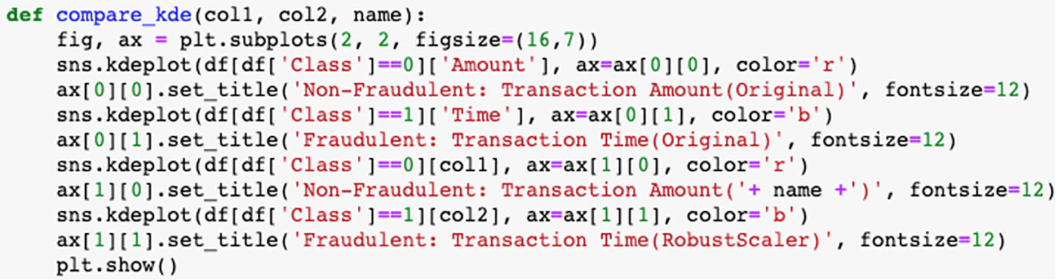
Here, we are using different features to store the scaled values because we need to compare them with the original values and choose to keep the best feature.

We need to visualize the change in the distribution compared with the original distribution so that we can decide whether to keep or discard it.

[19] “Feature scaling - Wikipedia.” https://en.wikipedia.org/wiki/Feature_scaling.

[20] “sklearn.preprocessing.StandardScaler — scikit-learn 0.22 ....” http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

The below function will help to plot the comparison with the original dataset.



```
def compare_kde(col1, col2, name):
    fig, ax = plt.subplots(2, 2, figsize=(16,7))
    sns.kdeplot(df[df['Class']==0]['Amount'], ax=ax[0][0], color='r')
    ax[0][0].set_title('Non-Fraudulent: Transaction Amount(Original)', fontsize=12)
    sns.kdeplot(df[df['Class']==1]['Time'], ax=ax[0][1], color='b')
    ax[0][1].set_title('Fraudulent: Transaction Time(Original)', fontsize=12)
    sns.kdeplot(df[df['Class']==0][col1], ax=ax[1][0], color='r')
    ax[1][0].set_title('Non-Fraudulent: Transaction Amount('+ name +')', fontsize=12)
    sns.kdeplot(df[df['Class']==1][col2], ax=ax[1][1], color='b')
    ax[1][1].set_title('Fraudulent: Transaction Time(RobustScaler)', fontsize=12)
    plt.show()
```

Figure 4.32: *Function to plot KDE*

From the above code snippet, we are expecting the same distribution. The original distribution will be on the top row, and the scaled distribution will be on the bottom row.

We can see the plot below and analyze it further.

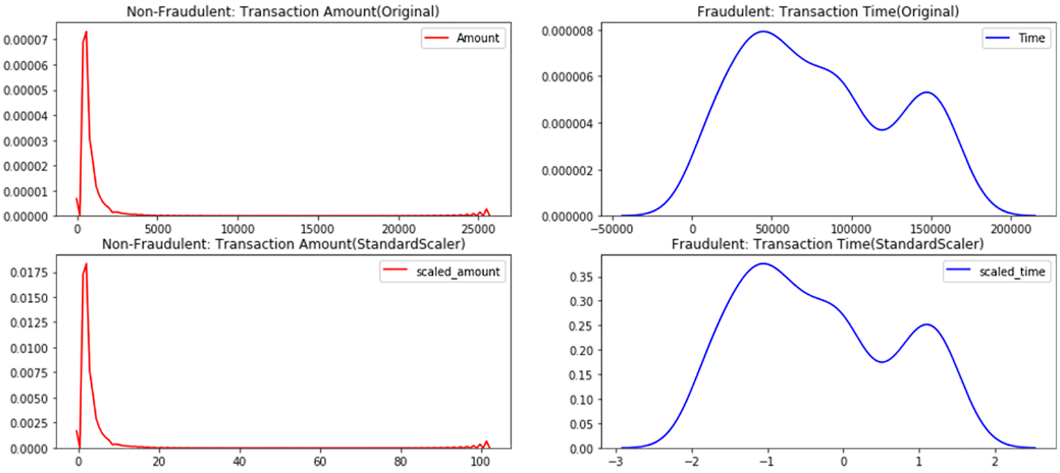
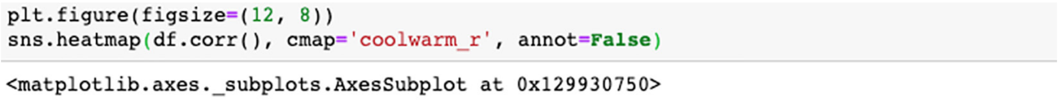


Figure 4.33: *Comparing Standard Scaled Features with Original Features*

We can see from the above-scaled distribution that the values of the axis have been changed and scaled-down a lot, keeping the distribution same. If we now see the difference between the "`Time`" and "`Amount`" scale, it has significantly reduced from the original one.

Let us see the same in the correlation matrix and see if there are any changes or not.



```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), cmap='coolwarm_r', annot=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x129930750>
```

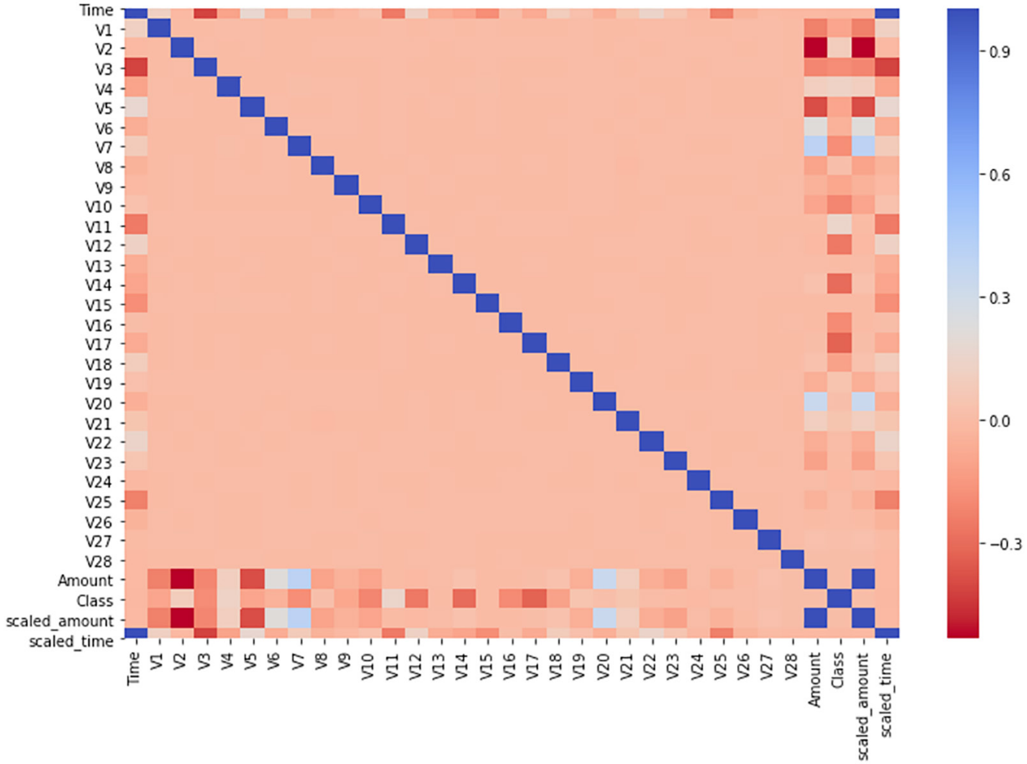


Figure 4.34: Correlation heat map after scaling

Interestingly, after scaling, it did not make any visual change, and that is only expected because, "`Amount`" and "`scaled_amount`" features correlation equals to one.

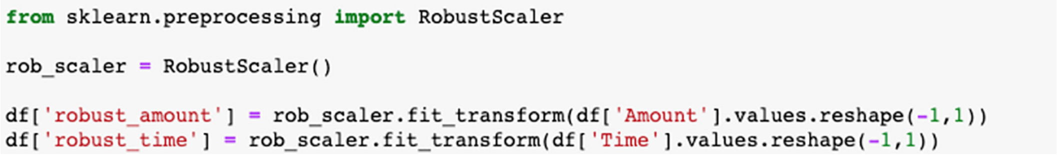
We will see a few more scaling algorithms and see whether we can use it or not.

From the next algorithm onwards, we won't be plotting the correlation heat map. At the end of the scaling section, we will see a single correlation heat map and analyze it.

Robust scaling[21]

Robust scaling is similar to standard scaling, but it removes the median and scales the data points according to the IQR.

[21] "sklearn.preprocessing.RobustScaler — scikit-learn 0.22 ...." http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html.



```
from sklearn.preprocessing import RobustScaler

rob_scaler = RobustScaler()

df['robust_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['robust_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))
```

Figure 4.35: *Robust scaling*

Robust scaling helps to handle the outliers better than standard scaling. We will also check the distribution and compare it with Standard Scalar.

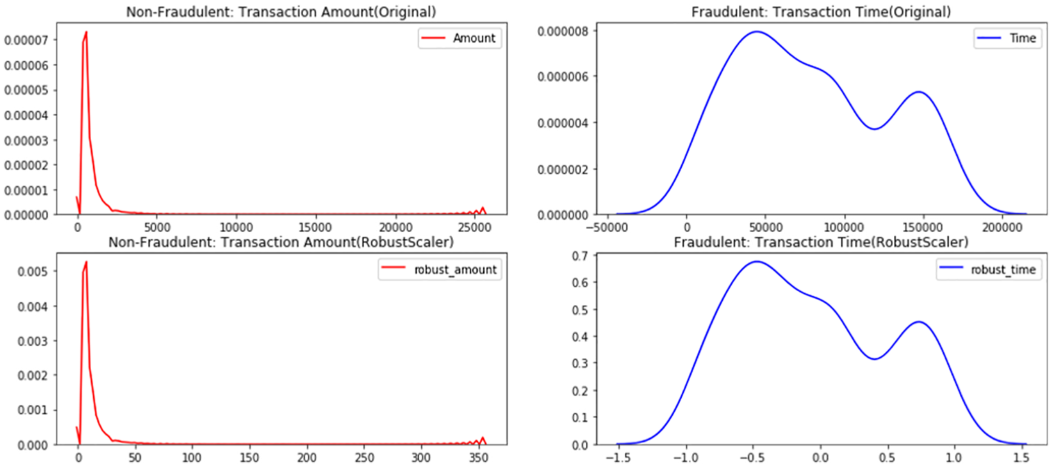


Figure 4.36: *Comparing Robust Scaling*

It didn't change that distribution, but the range and the scale are changed. Compared with both the Original and Standard Scalar, it is quite similar.

Power transformer[22]

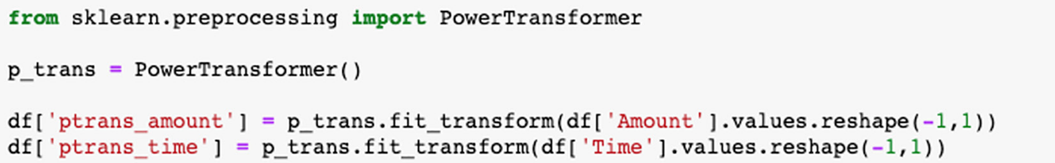
Power transformer belongs to a family of parametric[23], monotonic transformations[24] that target to map/project data points from any distribution to as close to a Normal/ Gaussian distribution. This process helps to stabilize the variance of the data and minimize skewness. This method has the highest effects on skewed data.

As a power transformer is based on monotonic transformation, so it preserves the rank of values among the feature.

[22] "sklearn.preprocessing.PowerTransformer — scikit-learn 0.22 ...." http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html.

[23] "Parametric equation - Wikipedia." https://en.wikipedia.org/wiki/Parametric_equation.

[24] "Monotonic function - Wikipedia." https://en.wikipedia.org/wiki/Monotonic_function.



```
from sklearn.preprocessing import PowerTransformer

p_trans = PowerTransformer()

df['ptrans_amount'] = p_trans.fit_transform(df['Amount'].values.reshape(-1,1))
df['ptrans_time'] = p_trans.fit_transform(df['Time'].values.reshape(-1,1))
```

Figure 4.37: *Code for power transformer*

Similarly, like before, we will use a different feature to store the scaled values. It will later help us to compare all the scaled features and come to some conclusion.

Let us see the KDE for the scaled feature and can assume that this plot will be interesting as "`Amount`" was highly skewed data. But, for "`Time`," the change won't be significant.

Power transformer uses the algorithm Yeo-Johnson transform to scale the values with the below law:

$$y_i^{(\lambda)} = \begin{cases} \left(\left(y_i + 1\right)^{\lambda} - 1\right) / \lambda & if\, \lambda \neq 0, y \geq 0 \\ \log\left(y_i + 1\right) & if\, \lambda = 0, y \geq 0 \\ -\left[\left(-y_i + 1\right)^{(2-\lambda)} - 1\right] / (2 - \lambda) & if\, \lambda \neq 2, y < 0 \\ -\log\left(-y_i + 1\right) & if\, \lambda = 2, y < 0 \end{cases}$$

Where the Yeo–Johnson transformation allows zero and negative values of y. λ can be any real number, and $\lambda = 1$ produces the identity transformation.

Now let us apply the above concept with the dataset and see how it looks.

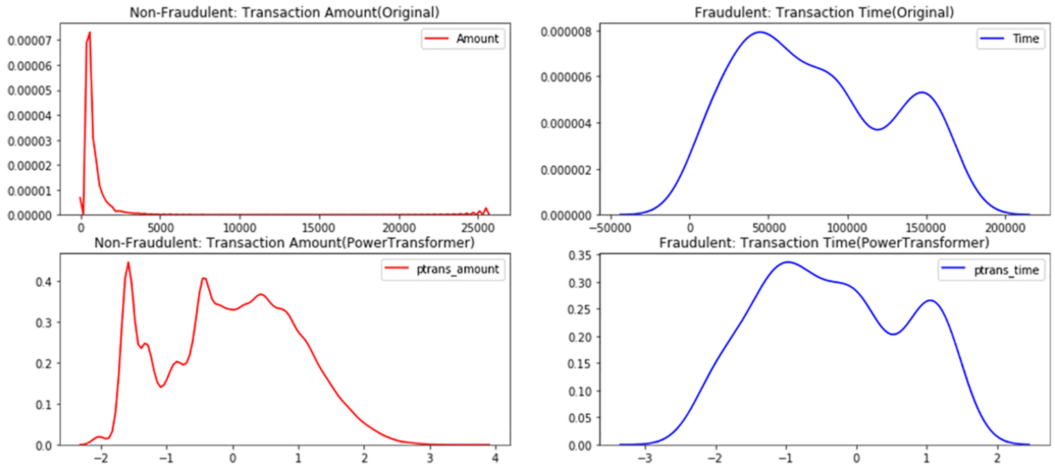


Figure 4.38: *Comparing power transformer*

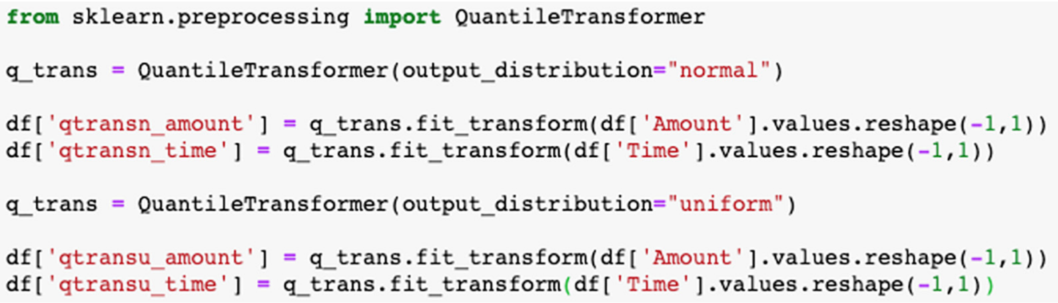
"Amount" feature had quite a lot of change, and it is not similar to a normal distribution but, when compared to "Time" distribution, the change is a lot.

If we again scale the scaled feature, then this will transform into a normal distribution. But it won't make sense to scale twice.

Quantile transformer[25]

Quantile transformer is a non-parametric method to transform the features such that it follows a uniform or a normal distribution. Therefore, for a given feature, this transformation tends to spread out the most frequent values. It also reduces the impact of (marginal) outliers: this is, therefore, a robust pre-processing scheme.

Even quantile transformer is based on monotonic transformation, so it preserves the rank of values among the feature.



```
from sklearn.preprocessing import QuantileTransformer

q_trans = QuantileTransformer(output_distribution="normal")

df['qtransn_amount'] = q_trans.fit_transform(df['Amount'].values.reshape(-1,1))
df['qtransn_time'] = q_trans.fit_transform(df['Time'].values.reshape(-1,1))

q_trans = QuantileTransformer(output_distribution="uniform")

df['qtransu_amount'] = q_trans.fit_transform(df['Amount'].values.reshape(-1,1))
df['qtransu_time'] = q_trans.fit_transform(df['Time'].values.reshape(-1,1))
```

Figure 4.39: *Quantile transformer*

Here, we will be performing for both uniform and normal distribution for both the features "Time" and "Amount."

We will start with transforming to a normal distribution and see how it works out.

[25] "sklearn.preprocessing.QuantileTransformer — scikit-learn ...." http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html.

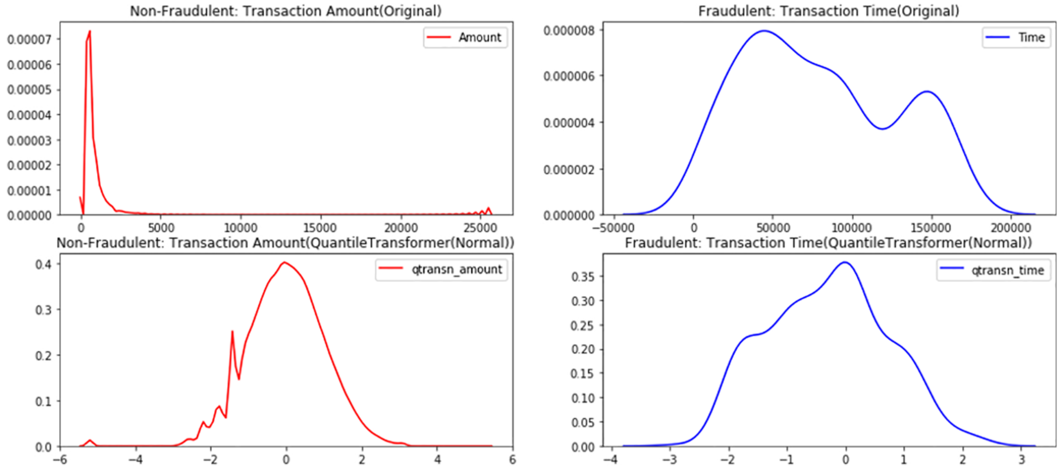


Figure 4.40: *Comparing Quantile Transformer -Normal*

The above distributions have a massive change from the skewed distribution, and now the "`Amount`" looks similar to a Gaussian distribution, it is the same case for "`Time`" as well.

Similarly, we will see for uniform distribution that the scaled value will resemble a uniform distribution.

Let us plot and confirm the above hypothesis.

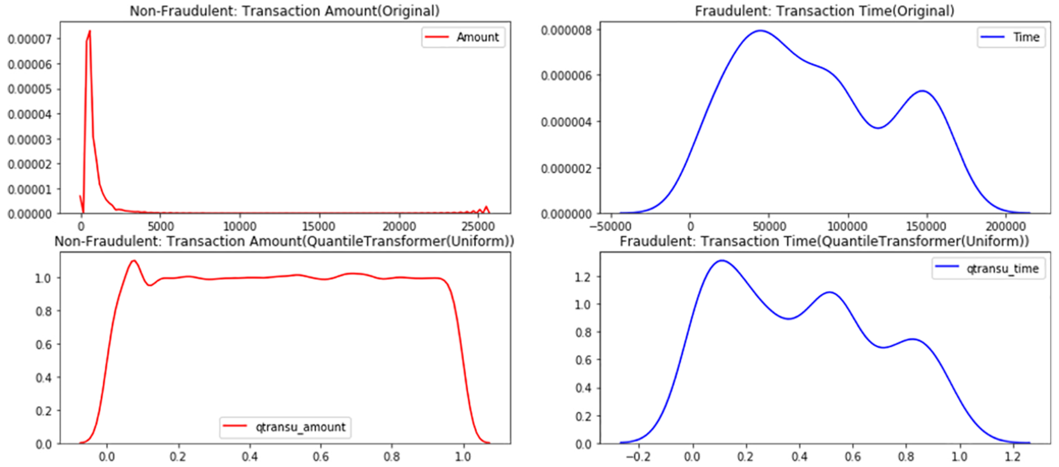


Figure 4.41: *Comparing Quantile Transformer -Uniform*

We can see that "`Amount`" is now a uniform distribution, and "`Time`" is close to a uniform distribution.

Seeing the distributions alone won't make any sense until and unless we see the effect. We can plot the correlation heat map for all the scaled features which we have done so far and see which type of scaling serves the best.

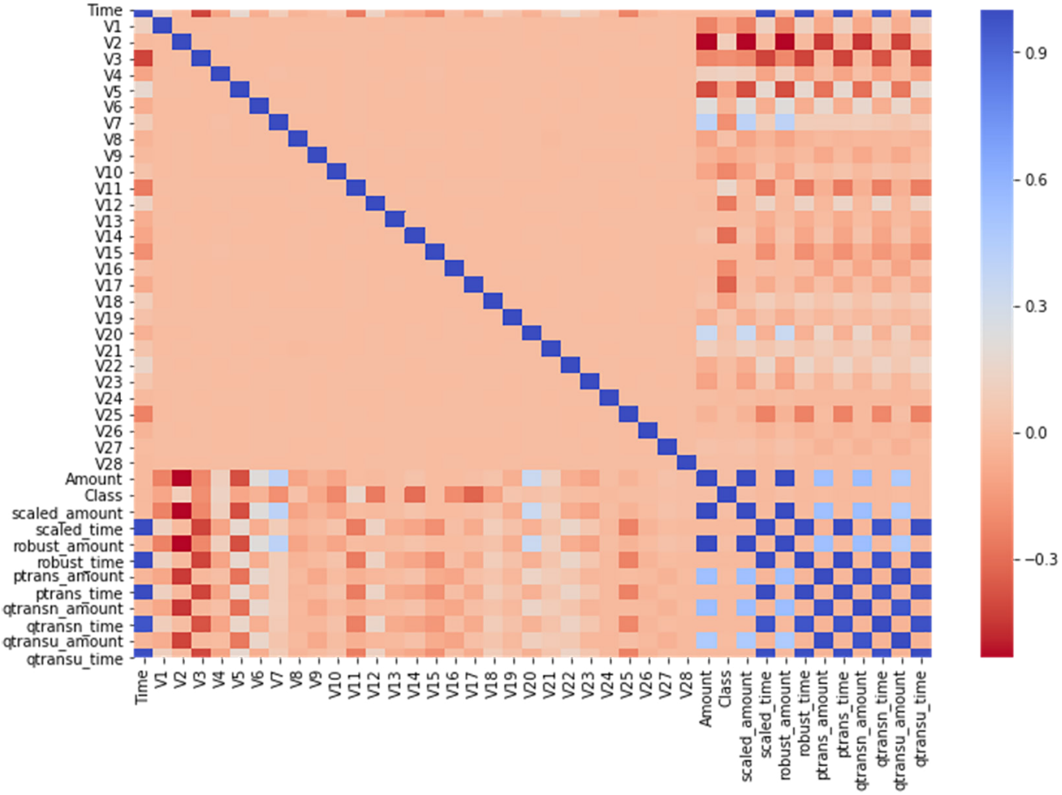


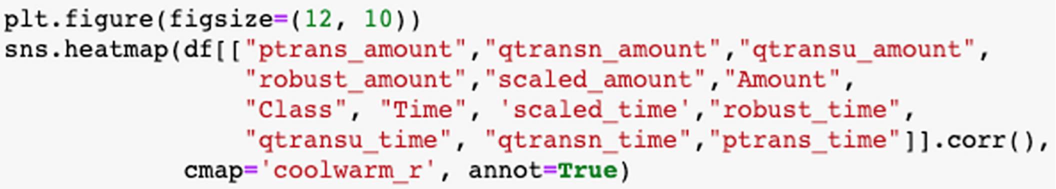
Figure 4.42: *Correlation heat map after applying all scaling algorithms*

Now, we need to analyze this correlation matrix and figure out the available scaled feature.

It is 100% possible that we choose one scaling algorithm/technique for one feature and another scaling algorithm/technique for a different feature.

We have one observation that none of the dimensionally reduced features changed its correlation with respect to "`Time`" and "`Amount`." There can be one reason why this is happening, i.e., due to the imbalance. Next section, we will see how to tackle the imbalance nature of the dataset and again visualize the correlation. Then we can see there will be a significant change in the correlation, and it will be easier to choose the best features that will have a contribution to the decision we make.

Analyzing this correlation heat map is very tough as we are not taking V1..., V28 into consideration as of now. So, it is better to plot all the scaled amounts and time separately to analyze them.



```
plt.figure(figsize=(12, 10))
sns.heatmap(df[["ptrans_amount","qtransn_amount","qtransu_amount",
                "robust_amount","scaled_amount","Amount",
                "Class", "Time", 'scaled_time',"robust_time",
                "qtransu_time", "qtransn_time","ptrans_time"]].corr(),
           cmap='coolwarm_r', annot=True)
```

Figure 4.43: Code for plotting only scaled features

The above code snippet will generate the below image. We will use the simplified and reduced version of the correlation matrix like the below image.

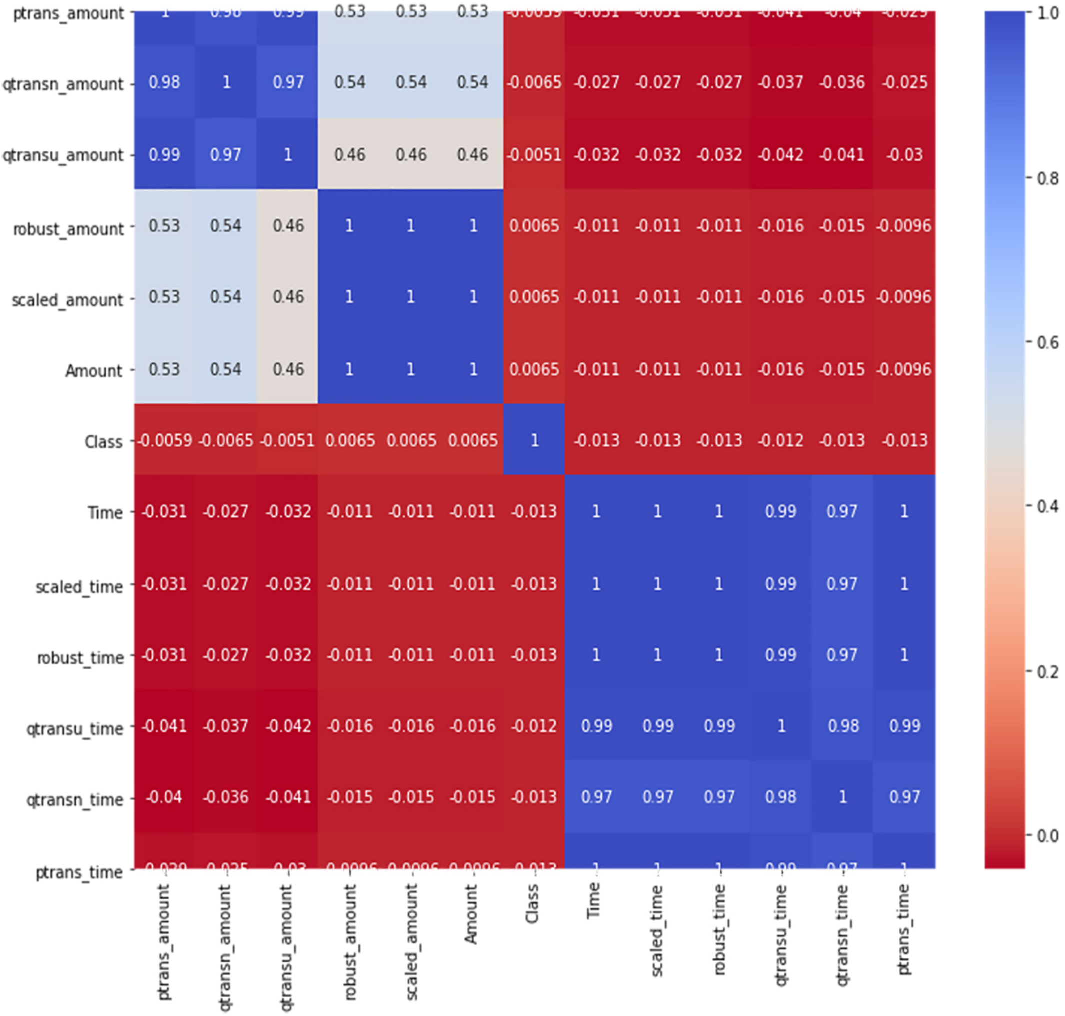
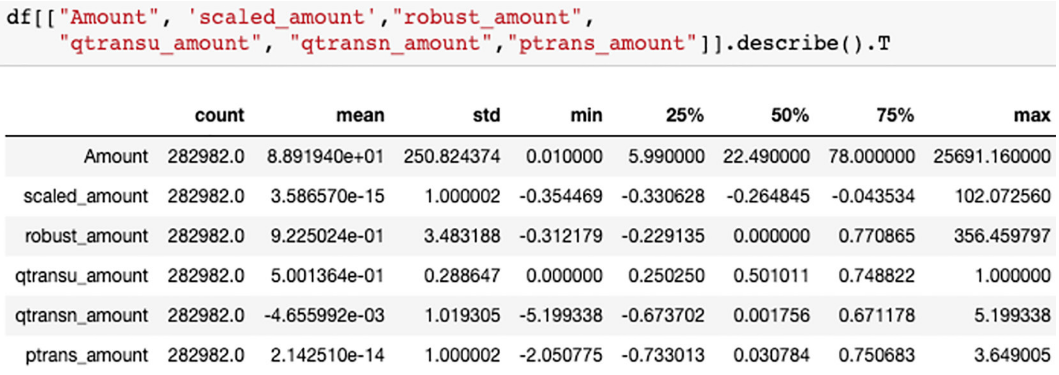


Figure 4.44: Correlation for all scaled features

We need to compare "Time," "Class," and "Amount" and see an interesting result that has nearly zero change in correlation concerning "Class." But some changes can be seen for statistical mean, standard deviation, and other features, as shown in the below diagram.

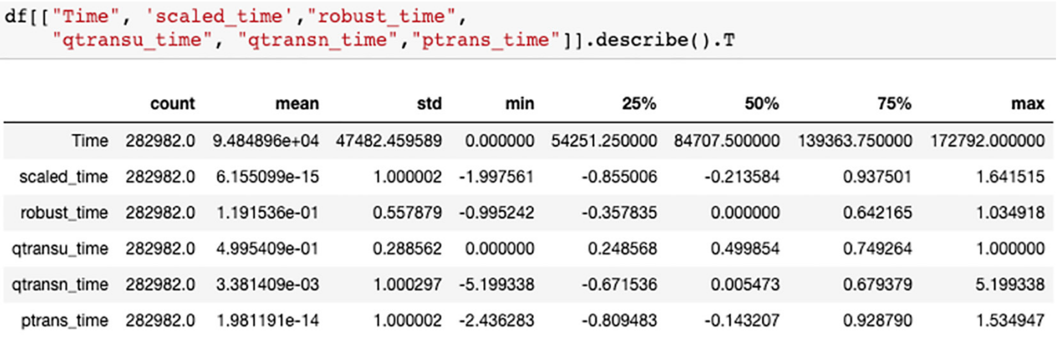


```
df[["Amount", 'scaled_amount',"robust_amount",
    "qtransu_amount", "qtransn_amount","ptrans_amount"]].describe().T
```

	count	mean	std	min	25%	50%	75%	max
Amount	282982.0	8.891940e+01	250.824374	0.010000	5.990000	22.490000	78.000000	25691.160000
scaled_amount	282982.0	3.586570e-15	1.000002	-0.354469	-0.330628	-0.264845	-0.043534	102.072560
robust_amount	282982.0	9.225024e-01	3.483188	-0.312179	-0.229135	0.000000	0.770865	356.459797
qtransu_amount	282982.0	5.001364e-01	0.288647	0.000000	0.250250	0.501011	0.748822	1.000000
qtransn_amount	282982.0	-4.655992e-03	1.019305	-5.199338	-0.673702	0.001756	0.671178	5.199338
ptrans_amount	282982.0	2.142510e-14	1.000002	-2.050775	-0.733013	0.030784	0.750683	3.649005

Figure 4.45: *Description of scaled "Amount."*

The above diagram shows all the scaled features for "Amount." We can see some changes like "std," "mean," IQR, and min-max. Next, we will see similar changes to the feature "Time."



```
df[["Time", 'scaled_time',"robust_time",
    "qtransu_time", "qtransn_time","ptrans_time"]].describe().T
```

	count	mean	std	min	25%	50%	75%	max
Time	282982.0	9.484896e+04	47482.459589	0.000000	54251.250000	84707.500000	139363.750000	172792.000000
scaled_time	282982.0	6.155099e-15	1.000002	-1.997561	-0.855006	-0.213584	0.937501	1.641515
robust_time	282982.0	1.191536e-01	0.557879	-0.995242	-0.357835	0.000000	0.642165	1.034918
qtransu_time	282982.0	4.995409e-01	0.288562	0.000000	0.248568	0.499854	0.749264	1.000000
qtransn_time	282982.0	3.381409e-03	1.000297	-5.199338	-0.671536	0.005473	0.679379	5.199338
ptrans_time	282982.0	1.981191e-14	1.000002	-2.436283	-0.809483	-0.143207	0.928790	1.534947

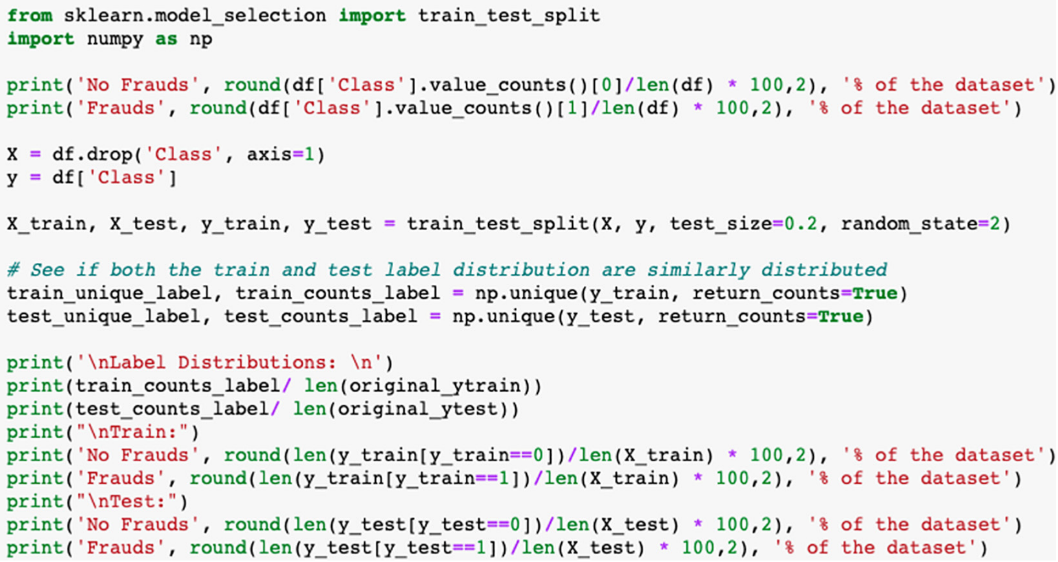
Figure 4.46: *Description of scaled "Time."*

"Time" also reflects the same characteristics as "Amount," but the change is drastic for some algorithm for the future "Amount." Being an ML practitioner, we need to do the same thing again and again, and every work depends on the experience. Like we have seen, there is not much information we found out from this scaling section but, when we will go ahead with the next section, i.e., handling the imbalanced data, we will see a huge improvement with the results from scaling too.

Splitting dataset

Before moving forward with handling the imbalance, we need to split the data for many reasons. One of the major reasons is for testing the model. We do not want to test the model with oversampled or undersampled data because that may lead to the wrong result.

Previously, we have used `train_test_split` function to split the dataset into a test and train. So, let us see split the dataset using the below code snippet.

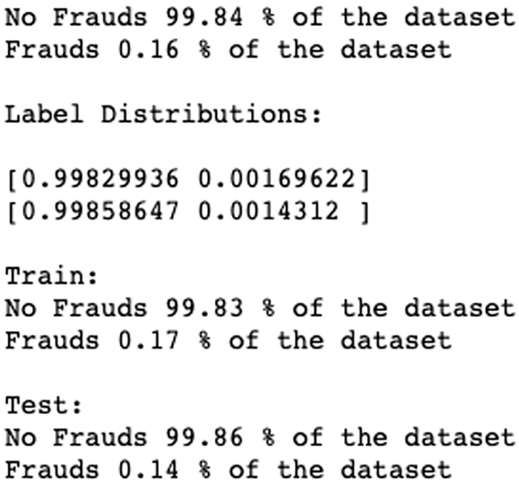


```
from sklearn.model_selection import train_test_split
import numpy as np

print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')

X = df.drop('Class', axis=1)
y = df['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(y_train, return_counts=True)
test_unique_label, test_counts_label = np.unique(y_test, return_counts=True)

print('\nLabel Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
print("\nTrain:")
print('No Frauds', round(len(y_train[y_train==0])/len(X_train) * 100,2), '% of the dataset')
print('Frauds', round(len(y_train[y_train==1])/len(X_train) * 100,2), '% of the dataset')
print("\nTest:")
print('No Frauds', round(len(y_test[y_test==0])/len(X_test) * 100,2), '% of the dataset')
print('Frauds', round(len(y_test[y_test==1])/len(X_test) * 100,2), '% of the dataset')
```

Figure 4.47: *Splitting dataset (Traditional)*

We are trying to split the dataset then printing the percentage of fraudulent and non-fraudulent transactions for both train and test datasets.

Let us see the output and analyze it.



```
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset

Label Distributions:

[0.99829936 0.00169622]
[0.99858647 0.0014312 ]

Train:
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset

Test:
No Frauds 99.86 % of the dataset
Frauds 0.14 % of the dataset
```

Figure 4.48: *Split Data Statistics (Traditional)*

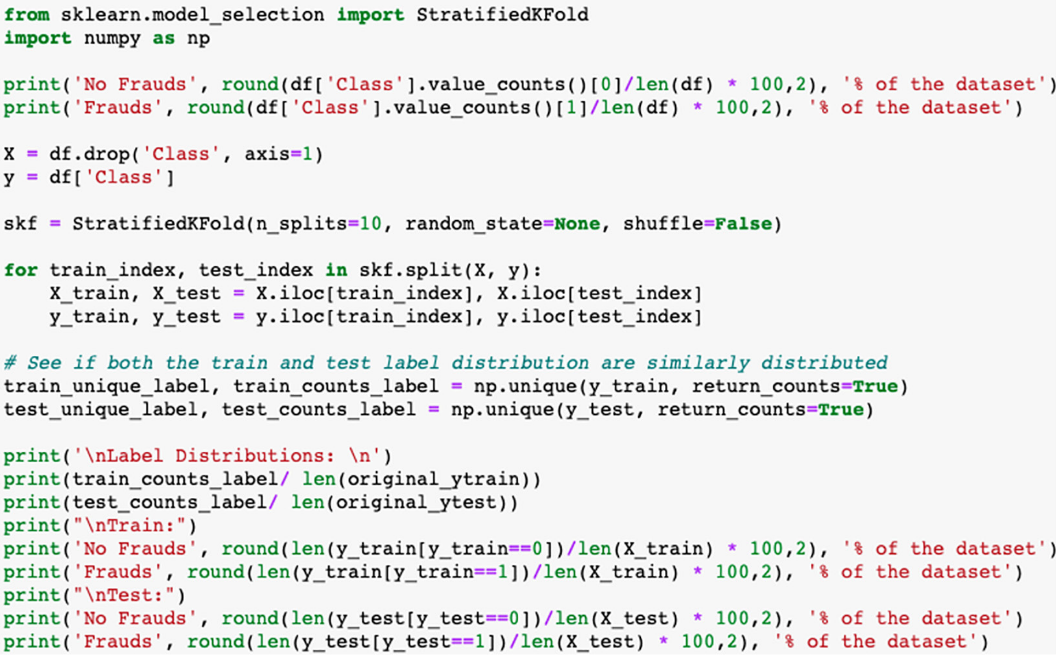
We can see a few things that are the percentage of fraud training data is 0.17%, and the percentage of fraud test data is 0.14% but, the original dataset has only 0.16% of data.

We need to make the fraud percentage the same or similar to the original fraud percentage. As we are dealing with an extremely small percentage of fraud data, our target should be the same.

This will help us to make the dataset distribution similar to the original one. We have to always keep in mind that the total number of fraud class is too less.

There is a solution to this problem that is stratified K-fold[26], it is a similar concept like a cross-validation K-fold technique. It gives us the same distribution as the original distribution.

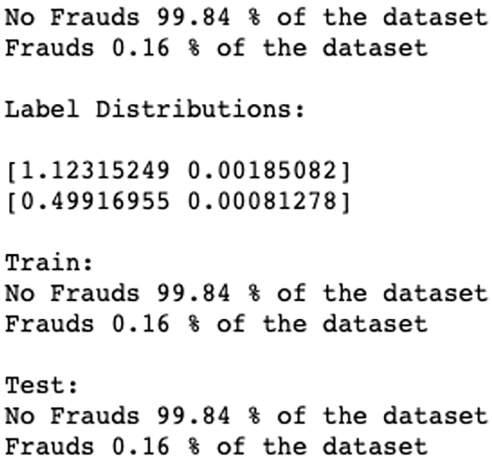
[26] "sklearn.model_selection.StratifiedKFold — scikit-learn 0.22 ...." http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html.



```
from sklearn.model_selection import StratifiedKFold
import numpy as np

print('No Frauds', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('Frauds', round(df['Class'].value_counts()[1]/len(df) * 100,2), '% of the dataset')

X = df.drop('Class', axis=1)
y = df['Class']

skf = StratifiedKFold(n_splits=10, random_state=None, shuffle=False)

for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(y_train, return_counts=True)
test_unique_label, test_counts_label = np.unique(y_test, return_counts=True)

print('\nLabel Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
print("\nTrain:")
print('No Frauds', round(len(y_train[y_train==0])/len(X_train) * 100,2), '% of the dataset')
print('Frauds', round(len(y_train[y_train==1])/len(X_train) * 100,2), '% of the dataset')
print("\nTest:")
print('No Frauds', round(len(y_test[y_test==0])/len(X_test) * 100,2), '% of the dataset')
print('Frauds', round(len(y_test[y_test==1])/len(X_test) * 100,2), '% of the dataset')
```

Figure 4.49: *Splitting Dataset (StratifiedKFold)*

From the above code snippet, we are expecting the training and testing dataset with the class distribution like the original one.



```
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset

Label Distributions:

[1.12315249 0.00185082]
[0.49916955 0.00081278]

Train:
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset

Test:
No Frauds 99.84 % of the dataset
Frauds 0.16 % of the dataset
```

Figure 4.50: *Split Data Statistics (StratifiedKFold)*

We got the expected output, i.e., the percentage of the class distribution is the same for the train, test, and the original dataset. Now, we can go ahead with handling imbalance characteristic of the dataset.

Handling imbalance

Tackling imbalance dataset can be done in a few ways

1. By oversampling the smaller dataset
2. By under-sampling the larger dataset
3. Mix of oversampling and under-sampling

In oversampling, we will generally create synthetic data points for the class, which has low counts. So, for oversampling, we have to use the test data from the original data, because after performing oversampling, if we split the data that won't be correct, and we will get a wrong result.

But for under-sampling, we can use the original dataframe to make a sub-sample out of it. But, in this case, we will only use the train test dataset.

What is a sub-sample?

Sub-sample just means a part of the original dataset, but in this scenario, the subsample will be 50-50. We will take the entire Fraudulent class and under-sample the non-fraudulent class so that the ratio is 50-50.

Advantages of sub-subsample are:

1. **Overfitting** is a common problem when there is an imbalance because it assumes that, in most cases, there are non-fraudulent transactions. But for a subsample, this problem will be less prone to overfitting.
2. Correlation with the class will improve drastically as our subsample will have no imbalance problem. Although we don't know what the "V" features stand for (but, my assumption is "Vector"), it will be useful to understand how each of these features influences the result with the class (Fraud or No Fraud) by having an imbalance dataframe we are not able to see the true correlations between the class and features.

We have seen some of the importance of sub-sampling and how it can affect some of the other aspects of the data like analysis and model output. Now, we will take a look at a different type of resampling techniques.

What is Resampling[27]?

Resampling is a technique to handle imbalance data by creating a sub-sampled dataset from the original data. Resampling can be done in two ways one removing samples from the majority class, i.e., also known as under sampling and adding synthetic samples to the minority class, which is also known as over-sampling.

[27] "Resampling (statistics) - Wikipedia." https://en.wikipedia.org/wiki/Resampling_(statistics).

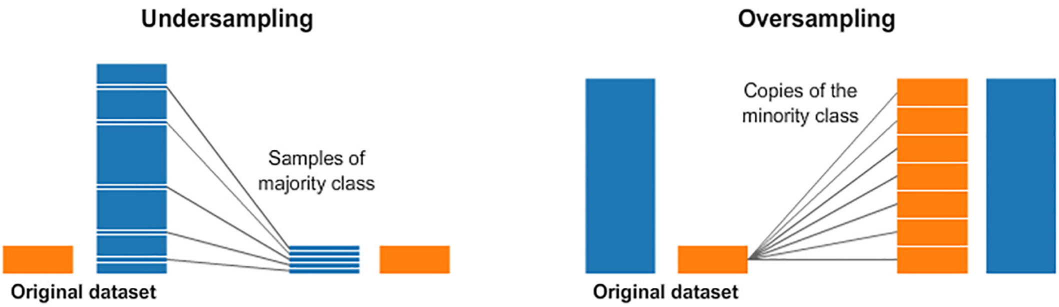
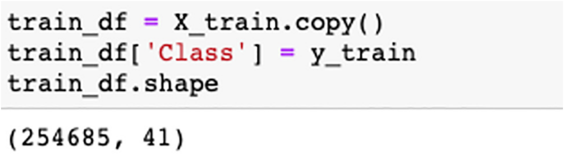


Figure 4.51: *Oversampling & Under sampling (Kaggle)*

We can see from the above illustration that how over and under sampling works. Going forward, we will see some of the classic and modern techniques to oversample and under-sample the dataset.

Before that, we need to make sure that we are using only the train dataset and not using the test dataset at all. The test dataset will only be used to validate the model and see how good it is.

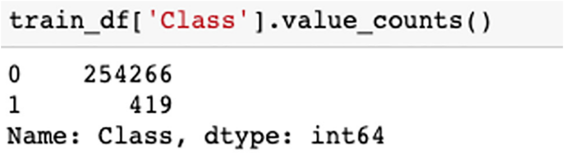


```
train_df = X_train.copy()
train_df['Class'] = y_train
train_df.shape
```

```
(254685, 41)
```

Figure 4.52: *Train data frame*

We can see that, we will only use 2,54,685 records. Within that training dataframe, there will be a mixture of fraud and non-fraud datasets in the ratio of the Original dataset as we have used the Stratified K-fold technique. Let's see the counts of different classes.



```
train_df['Class'].value_counts()
```

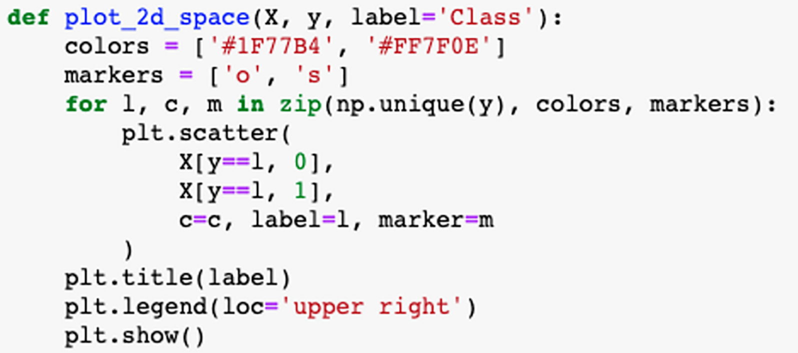
```
0    254266
1       419
Name: Class, dtype: int64
```

Figure 4.53: *Train Dataframe Class Distribution*

We can see we only have 419 records for Class 1, i.e., Fraud class. We will perform resampling on this dataset and analyze the dataset.

Before going ahead, we need to find a way to visualize the 41 column features. We can use a dimensionality reduction algorithm like Principal Component Analysis to reduce to 2 principal components and plot in a 2D plot.

Below a function will help us to plot a 2D graph.

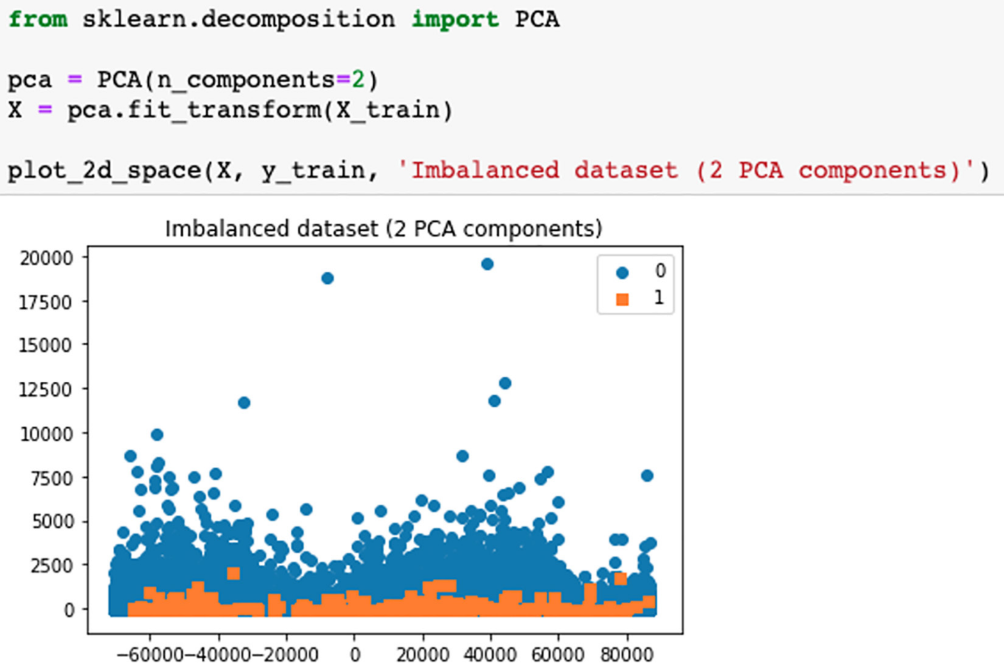


```
def plot_2d_space(X, y, label='Class'):
    colors = ['#1F77B4', '#FF7F0E']
    markers = ['o', 's']
    for l, c, m in zip(np.unique(y), colors, markers):
        plt.scatter(
            X[y==l, 0],
            X[y==l, 1],
            c=c, label=l, marker=m
        )
    plt.title(label)
    plt.legend(loc='upper right')
    plt.show()
```

Figure 4.54: *Function to plot 2 component PCA*

As we will plot a 2D graph, again and again, we are using a function for that. Then a simple function call is sufficient to plot it.

So, let us plot for the original train dataframe and see how it looks.



```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(X_train)

plot_2d_space(X, y_train, 'Imbalanced dataset (2 PCA components)')
```

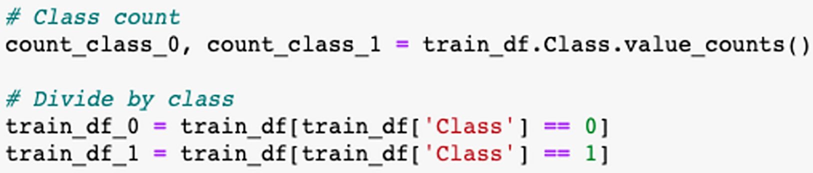
Figure 4.55: *PCA plot for Original dataframe*

We can see a certain pattern of different classes of the transaction from the above image. Going forward, we will see two varieties of under-sampling and two varieties of over-sampling.

Random under-sampling

Random under-sampling will remove the majority class data points such that it is equivalent/equal/proportional to the minority class. This process creates a balanced dataset, thus preventing common problems like overfitting.

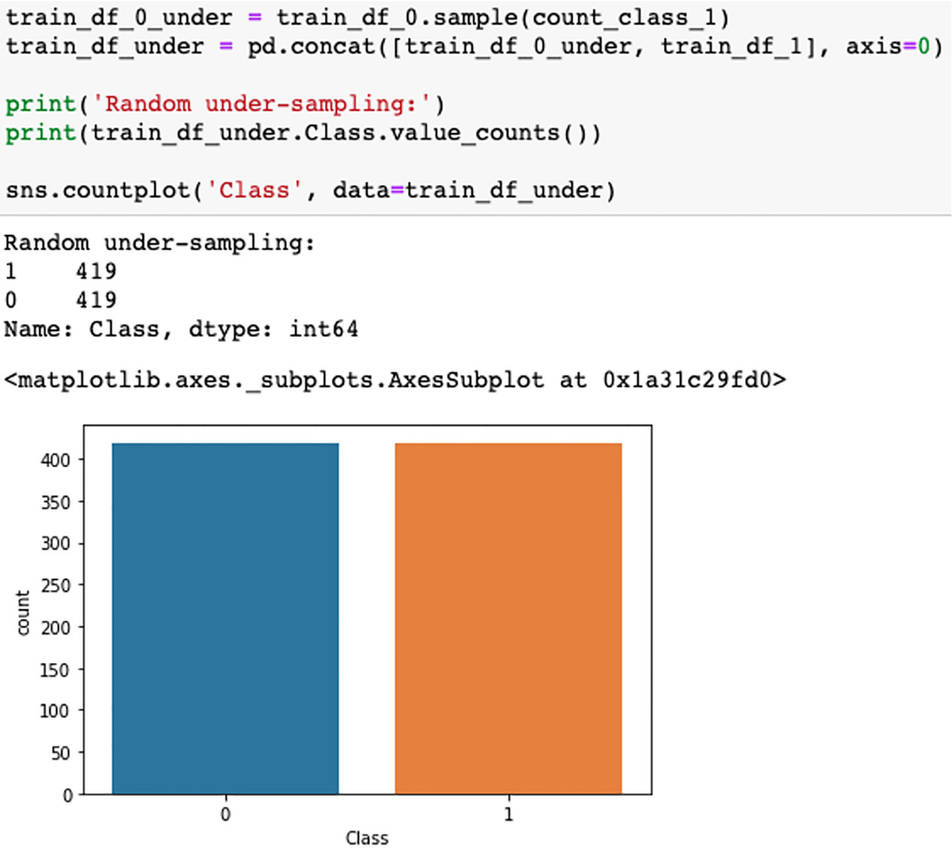
In the below code snippet, we will divide the train dataframe into two classes.



```
# Class count
count_class_0, count_class_1 = train_df.Class.value_counts()

# Divide by class
train_df_0 = train_df[train_df['Class'] == 0]
train_df_1 = train_df[train_df['Class'] == 1]
```

Figure 4.56: Diving the training dataset

As we have divided the training dataset as per different classes, now we can perform random under-sampling or reduce the record counts for the majority class. Under-sampling can be achieved in multiple ways, but here we will only see random sampling, the next section, we will focus on another method.



```
train_df_0_under = train_df_0.sample(count_class_1)
train_df_under = pd.concat([train_df_0_under, train_df_1], axis=0)

print('Random under-sampling:')
print(train_df_under.Class.value_counts())

sns.countplot('Class', data=train_df_under)
```

```
Random under-sampling:
1    419
0    419
Name: Class, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x1a31c29fd0>
```

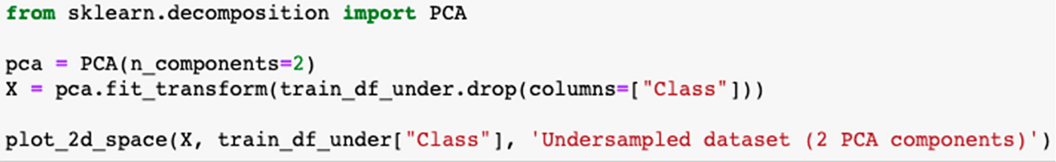
Figure 4.57: Random undersampling

From the above illustration, we can see Class 0, i.e., the non-fraudulent class, has the same records as a fraudulent class. Now, this dataset is equidistributed, and it is likely to give a better correlation and model results, as we have hypothesized before.

But, reducing/ resampling the number of records drastically can arise multiple problems due to information loss as we are bringing 419 non-fraud transactions from 284,315 non-fraud transactions.

After completing random undersampling, we need to visualize the dataset with 41 features.

With the original dataset, we have dimensionally reduced using PCA into two principal components; similarly, we have to perform the same operation on this dataset such that we will be able to compare them.



```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(train_df_under.drop(columns=["Class"]))

plot_2d_space(X, train_df_under["Class"], 'Undersampled dataset (2 PCA components)')
```

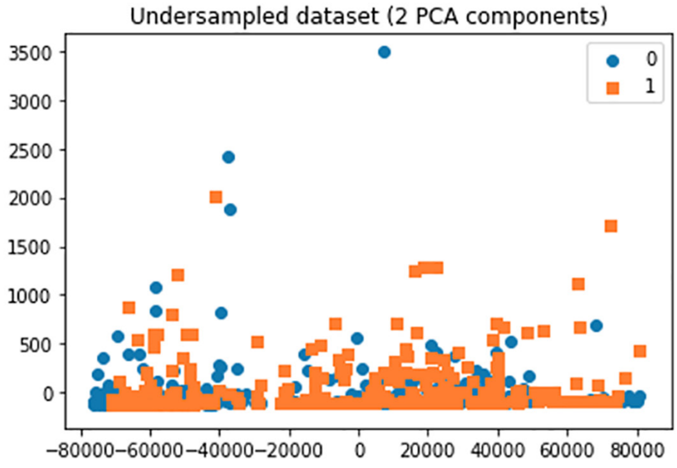


Figure 4.58: PCA plot for random undersampling

From the above diagram, we can see the homogeneity in the data points of different classes.

With this new homogeneous dataset, we are expecting a good correlation between all the features and primarily with the feature "class." Let us plot the correlation matrix heat map and analyze it.

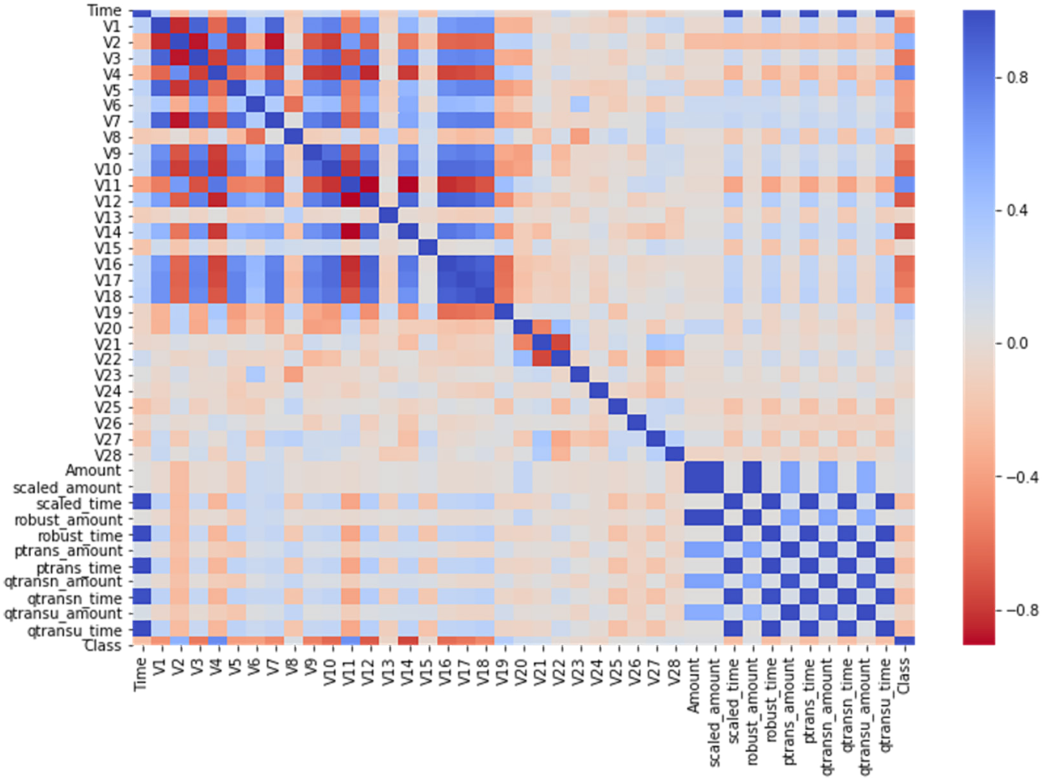


Figure 4.59: *Correlation for Random Under Sampled data*

We can see a drastic change with the correlation matrix for mostly all the features. Some features are now positively and negatively correlated, but due to the imbalanced nature of the original dataset, the correlation matrix nearly shows zero correlation for the features V1..., V28.

If we individually see the features V1..., V28 , then there are some positively and negatively correlated features as listed below:

- Positive correlations- V2, V4, V11, and V19 are positively correlated.
- Negatively correlations- V17, V14, V12, V16, and V10 are negatively correlated.

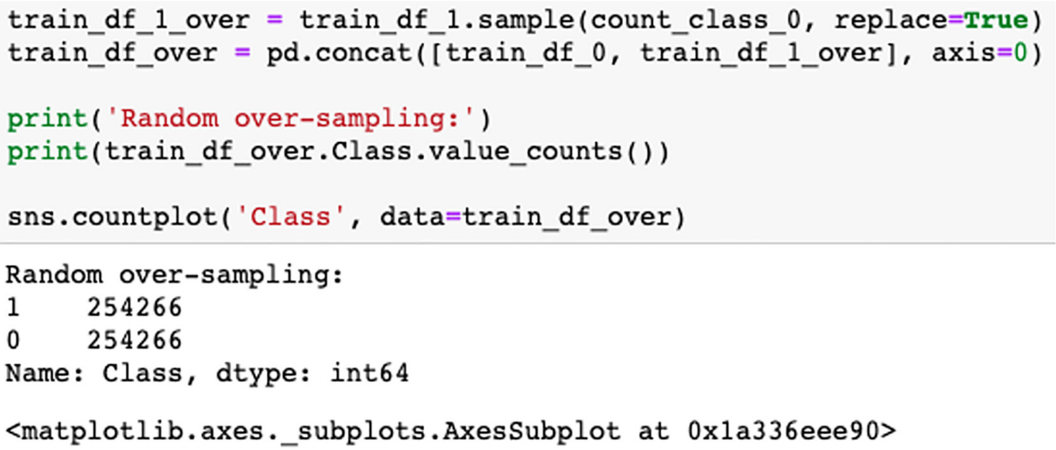
Later we will analyze the above nine features individually.

Now, we have a list of features that will affect the output feature, i.e., "Class." There is one more interesting observation that after random under-sampling, all the scaled features and their respective original features have the same or similar correlation.

Random over-sampling

Similar to random under-sampling, we have random over-sampling where we will oversample the minor class, i.e., in this case, it will be the fraudulent transaction dataset.

Let us see how the new dataset will look after it is oversampled.



```
train_df_1_over = train_df_1.sample(count_class_0, replace=True)
train_df_over = pd.concat([train_df_0, train_df_1_over], axis=0)

print('Random over-sampling:')
print(train_df_over.Class.value_counts())

sns.countplot('Class', data=train_df_over)
```

```
Random over-sampling:
1    254266
0    254266
Name: Class, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x1a336eee90>
```

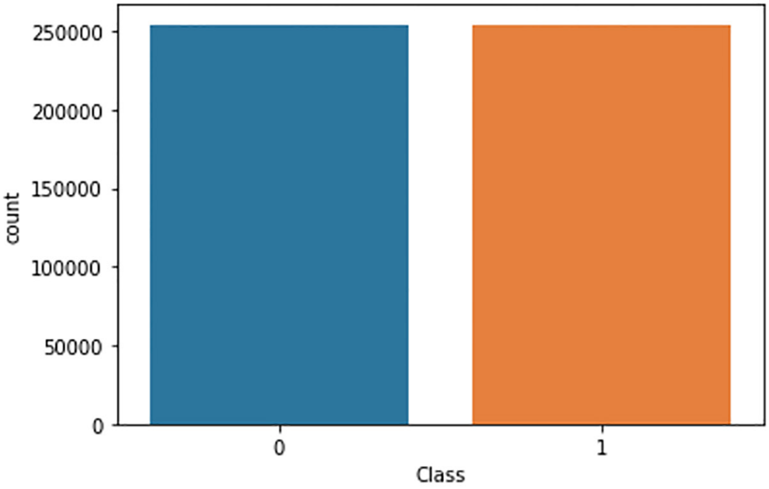
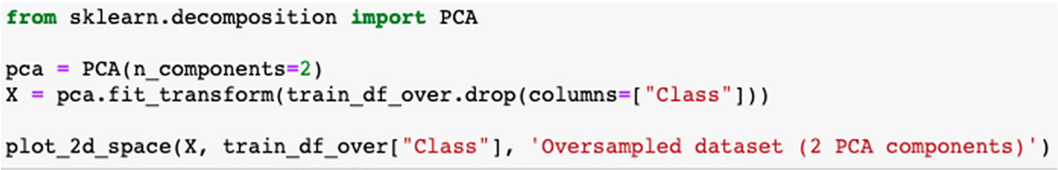


Figure 4.60: *Random over-sampling*

We can see both the classes have 2,54,266 records, and now the problem of imbalance data is not there. This doesn't mean the dataset is good to use; we need to analyze further to come to this conclusion.



```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(train_df_over.drop(columns=["Class"]))

plot_2d_space(X, train_df_over["Class"], 'Oversampled dataset (2 PCA components)')
```

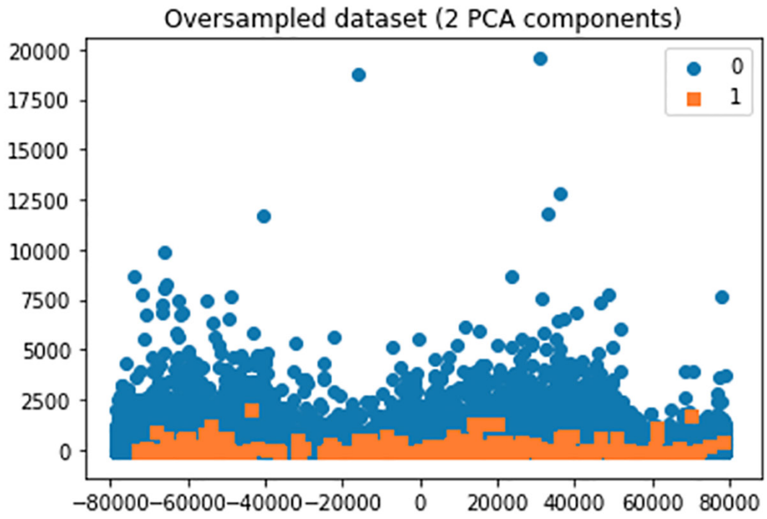


Figure 4.61: *PCA plot for random over-sampling*

The Oversampled data set shows the same plot as the original dataset plot. So that it means the correlation heat map will be similar to the original one. But unfortunately, no, it will be extremely different because this oversampled dataset is equidistributed so it will have a better correlation and will be similar to the randomly under-sampled data set.

Let us plot the correlation heat map and match it with the original correlation heat map and the under-sampled correlation heat map.

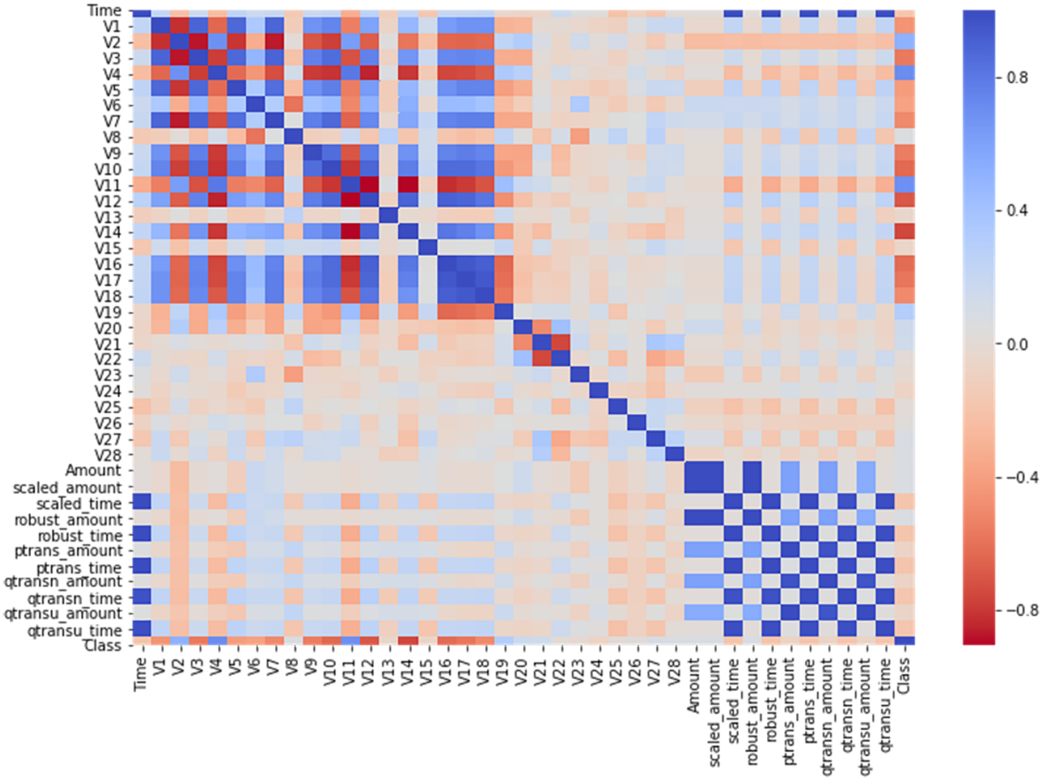


Figure 4.62: *Correlation heat map for random oversampled data*

Seeing the above correlation, we can say it has no resemblance with the original heat map, but in turn, it is similar to a random under-sampled dataframe. Let us see some other algorithms and compare them along the way.

Tomek's Links under sampling

This is another type of under-sampling where we generally remove Tomek's Links[28] from the dataset.

The detailed explanation is out of scope for this book.

But we can explain/visualize the concept, in brief, using some visualization.

Note: All the steps below are not detailed out. It is just a brief workflow

Step 1: The data points are plotted in n-dimensions(n-features) in this case for simplicity, we are using 2-axis and differentiate them with respect to class (here, there are 2 classes only).

[28] "Two Modifications of condensed nearest-neighbor (CNN)" https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4309452.

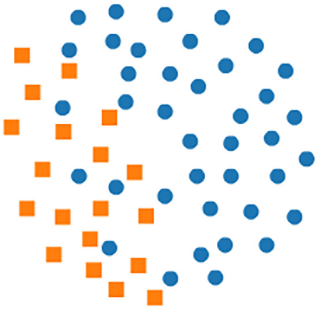


Figure 4.63: Plot data

Step 2: We find the nearest neighbor where k=1 where the minor and the major classes are coupled.

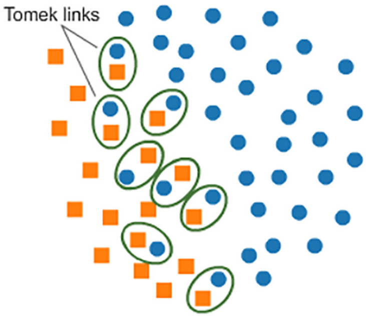


Figure 4.64: Find Tomek Links

Step 3: After we find the coupled data, we will remove the major class for under-sampling.

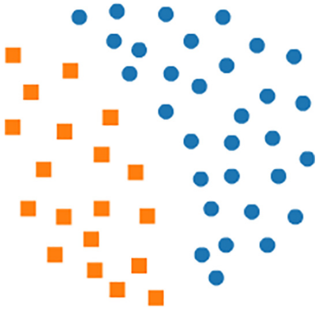
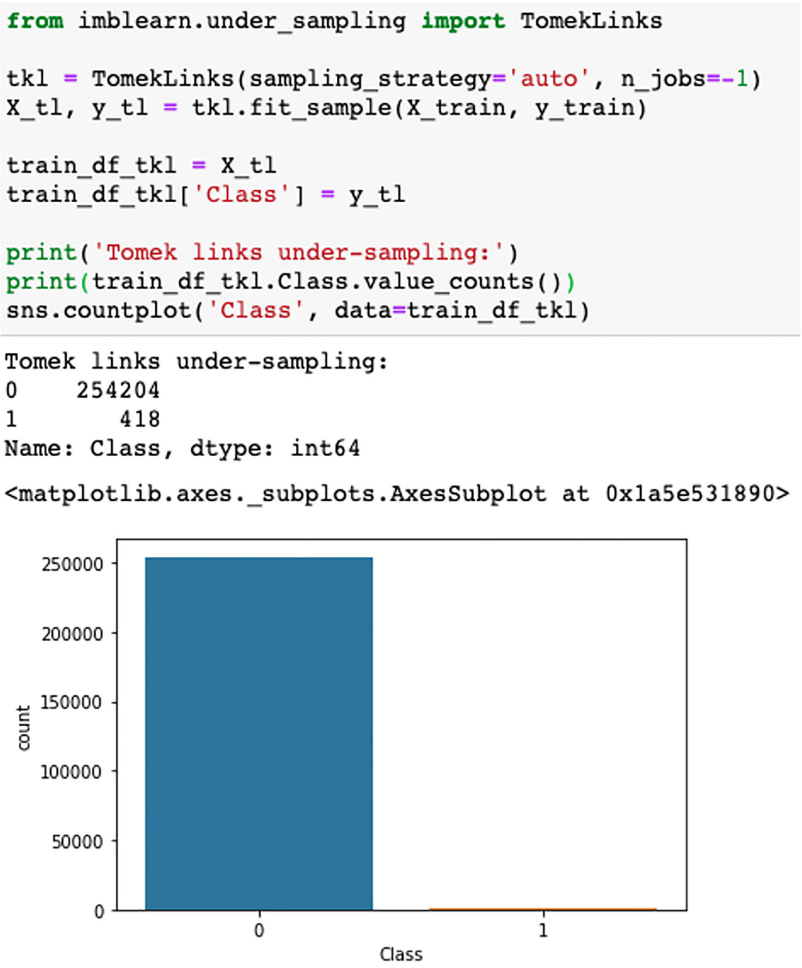


Figure 4.65: Remove the Major Class

The above illustrations give us a simple working of Tomek's Link. This helps us to define a good decision boundary and remove some of the data points from the majority class. So, this can be termed as under-sampling.



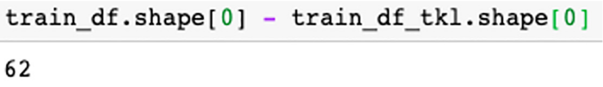
```
from imblearn.under_sampling import TomekLinks

tkl = TomekLinks(sampling_strategy='auto', n_jobs=-1)
X_tl, y_tl = tkl.fit_sample(X_train, y_train)

train_df_tkl = X_tl
train_df_tkl['Class'] = y_tl

print('Tomek links under-sampling:')
print(train_df_tkl.Class.value_counts())
sns.countplot('Class', data=train_df_tkl)
```

```
Tomek links under-sampling:
0    254204
1       418
Name: Class, dtype: int64
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a5e531890>
```

Figure 4.66: Tomek Links

Seeing the image and the counts, we can see there is not much change. We can clearly say there were extremely a smaller number of Tomek's link.

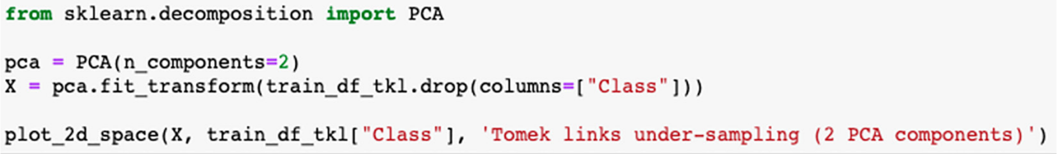


```
train_df.shape[0] - train_df_tkl.shape[0]
```

```
62
```

Figure 4.67: Total data points removed

We only have 62 records like that which can be removed from the dataset. We can clearly state that there won't be any change from the original data as the change is not significant enough. Let's plot the principal components and see how it looks.



```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(train_df_tkl.drop(columns=["Class"]))

plot_2d_space(X, train_df_tkl["Class"], 'Tomek links under-sampling (2 PCA components)')
```

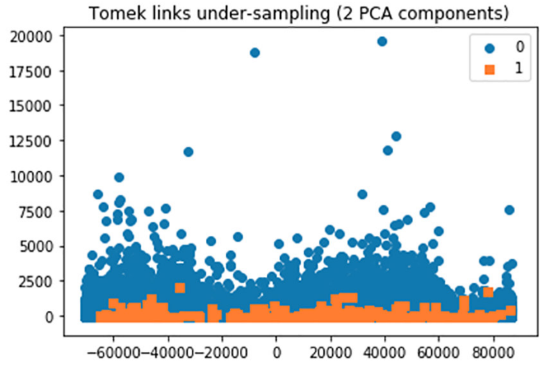


Figure 4.68: *PCA plot for Tomek's Link*

We can see it is the same distribution as the original dataframe. So, we can say that correlation will be the same, and the distribution did not change at all.

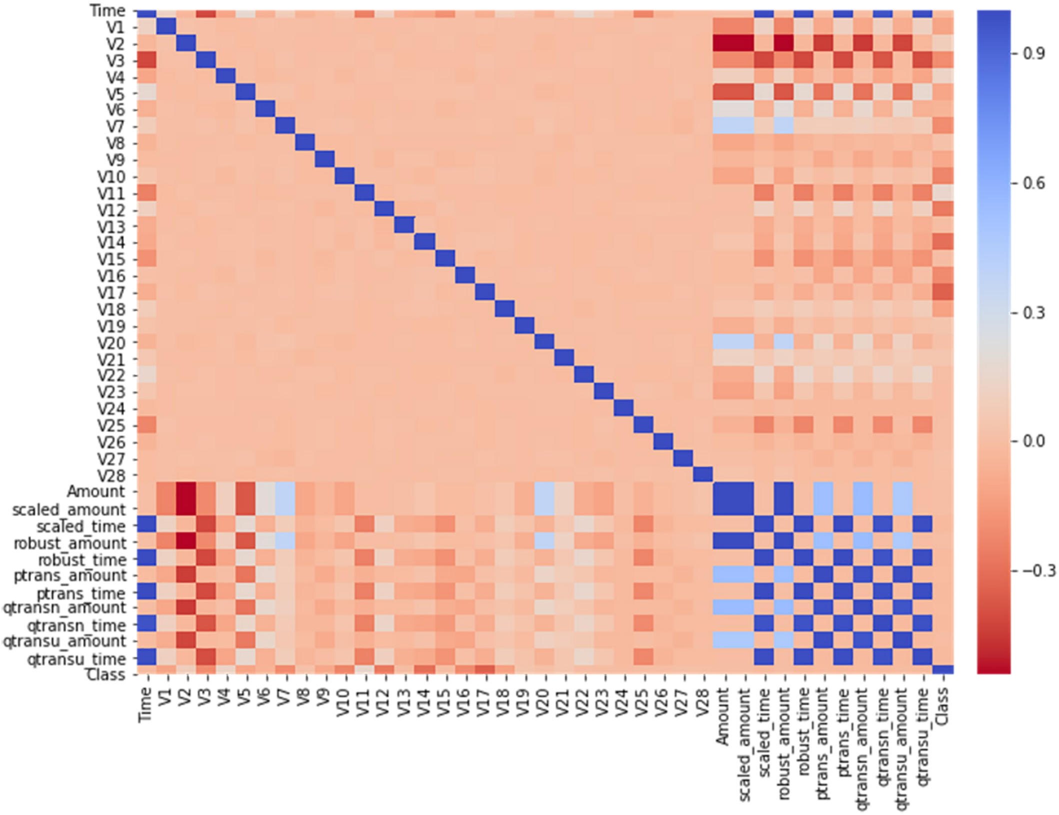


Figure 4.69: *Correlation Heatmap for Tomek Links*

We can confirm that the heat map did not change at all, so this technique cannot be used for under-sampling.

In real-time, this work is highly tedious and time taking, to go through many hypotheses, and later it turns out to be wrong. This gives us experience and more insight into the dataset.

As this is a small number of Tomek's link, we can remove them so that other algorithms can work better if their algorithms have some effect due to those links.

Synthetic minority over-sampling technique

We will see one more oversampling technique **Synthetic Minority Over-sampling Technique**[29] **(SMOTE)**. This technique oversamples the minor class using the pattern of the dataset.

To understand SMOTE, we will see how it works, and to know how this algorithm over samples the minority class.

Step 1: Let us plot the data points n-dimensional space(n-features) and classify them as per the "`Class`."

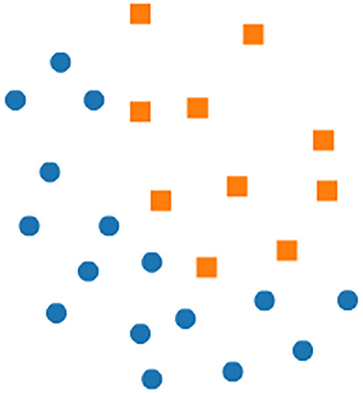


Figure 4.70: *Plot the data points*

Step 2: Create a pattern/boundary of the minority dataset with the data points. We can create a space using the boundary of the minority data and add synthetic values within the boundary where the nearest neighbor is also of the same class.

[29] "SMOTE: Synthetic Minority Over-sampling Technique | Journal ...." https://jair.org/index.php/jair/article/view/10302.

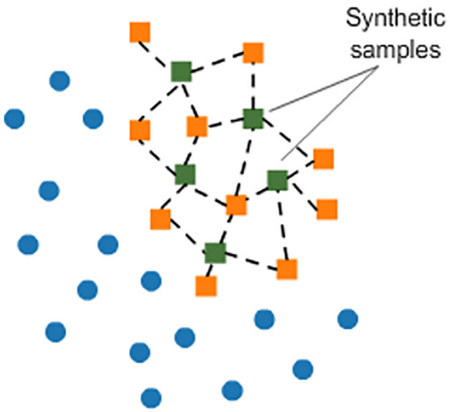


Figure 4.71: Find the relation between the minor datapoints

Step 3: Plot those points and which meets step 2 conditions and assign them the minor class.

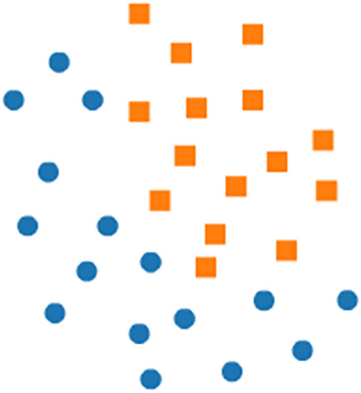
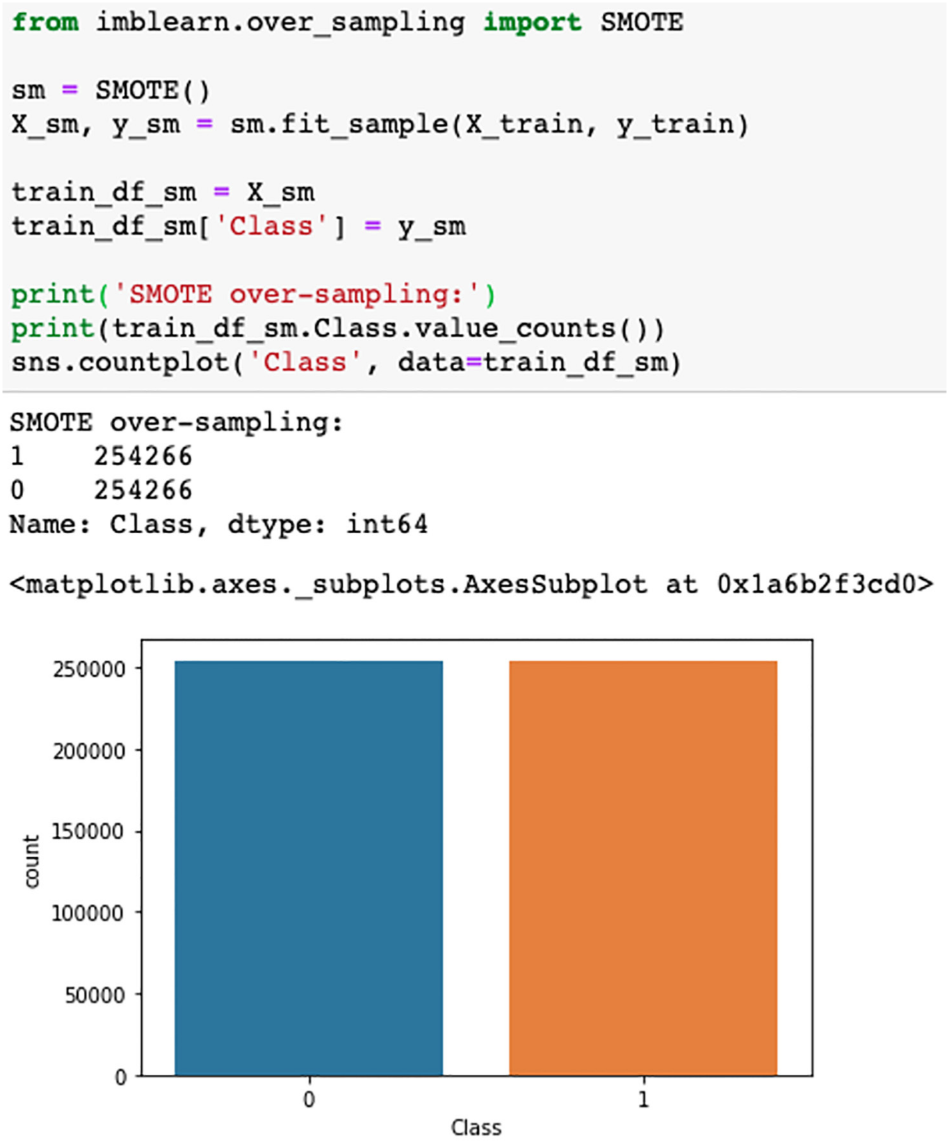


Figure 4.72: Oversample the minor class

We have seen a basic working of SMOTE, so; now we can start implementing the above concept to the training dataset. So, after implementing SMOTE, which is an oversampling algorithm, the minority class will have the same count of data as the majority class. Let us execute the below code to see how it will look like.



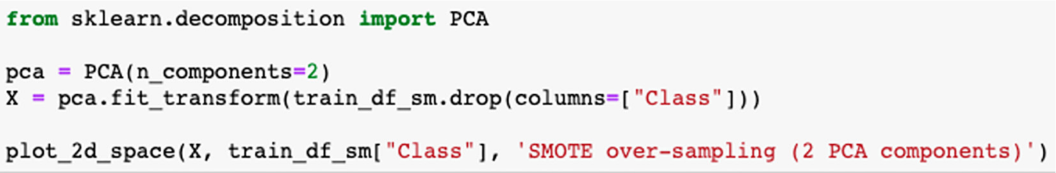
```
from imblearn.over_sampling import SMOTE

sm = SMOTE()
X_sm, y_sm = sm.fit_sample(X_train, y_train)

train_df_sm = X_sm
train_df_sm['Class'] = y_sm

print('SMOTE over-sampling:')
print(train_df_sm.Class.value_counts())
sns.countplot('Class', data=train_df_sm)
```

```
SMOTE over-sampling:
1     254266
0     254266
Name: Class, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x1a6b2f3cd0>
```

Figure 4.73: SMOTE

We can see that it equidistributed, and we have the same number of records for both the classes. But we can think that increasing the minor class with this amount can affect the quality of the dataset. The answer is mostly, it won't affect as the synthetic data points follow the minor class pattern.

We can take a look at the plot of the data and compare it with the random oversampling dataset.



```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X = pca.fit_transform(train_df_sm.drop(columns=["Class"]))

plot_2d_space(X, train_df_sm["Class"], 'SMOTE over-sampling (2 PCA components)')
```

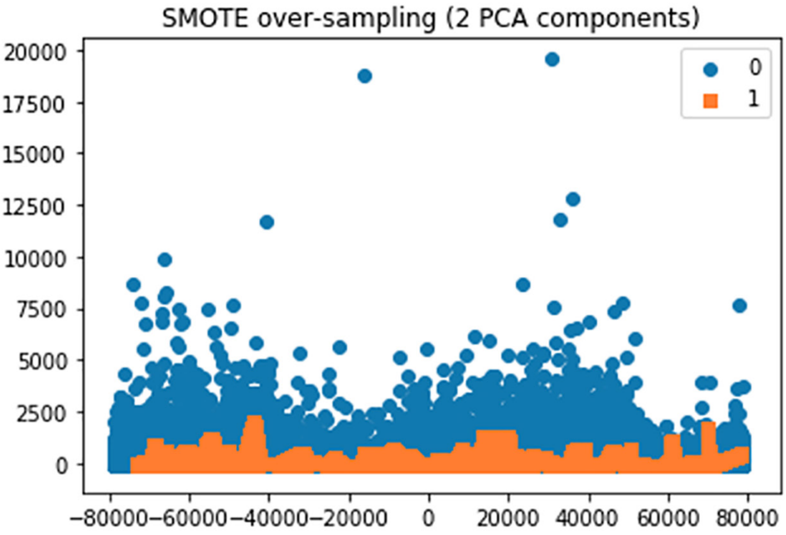


Figure 4.74: PCA plot for SMOTE

We can see there is a difference in the distribution. But it is not significant enough from my perspective. We can test these hypotheses in the latter part of the chapter.

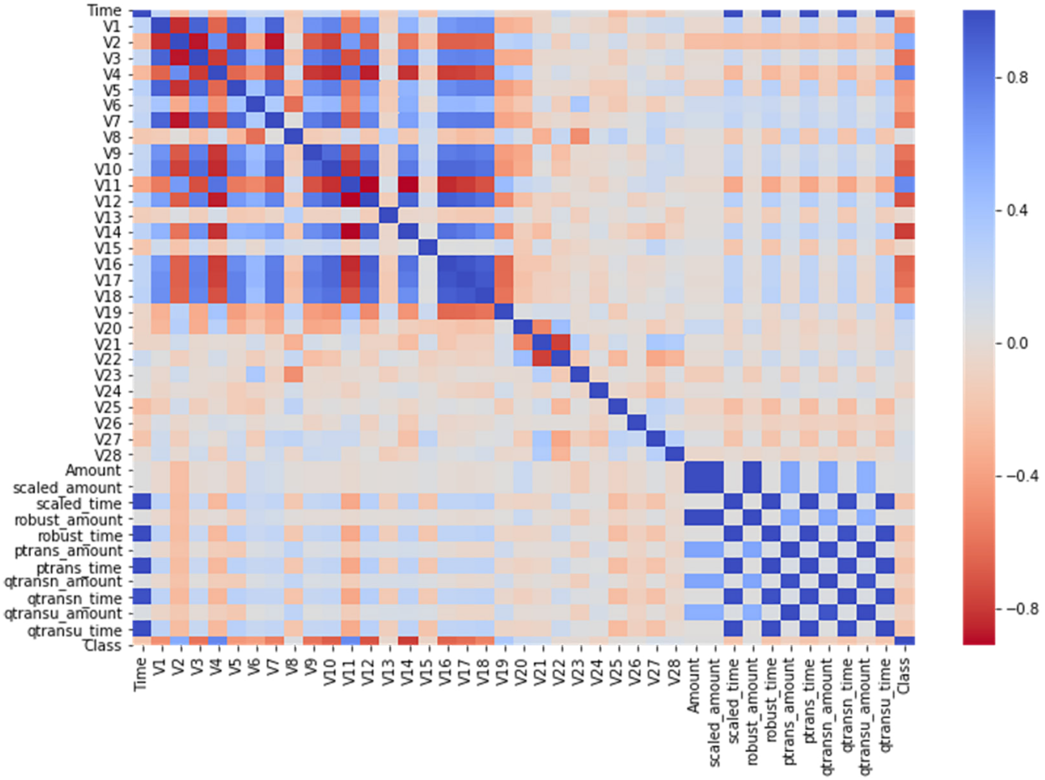


Figure 4.75: *Correlation Heatmap for SMOTE dataset*

Interestingly, we can see a similar heat map for both Random Oversampling and SMOTE.

So, in the next section, we will see some analysis of the Positive and Negative correlations that we have inferred from this section.

Data re-analysis

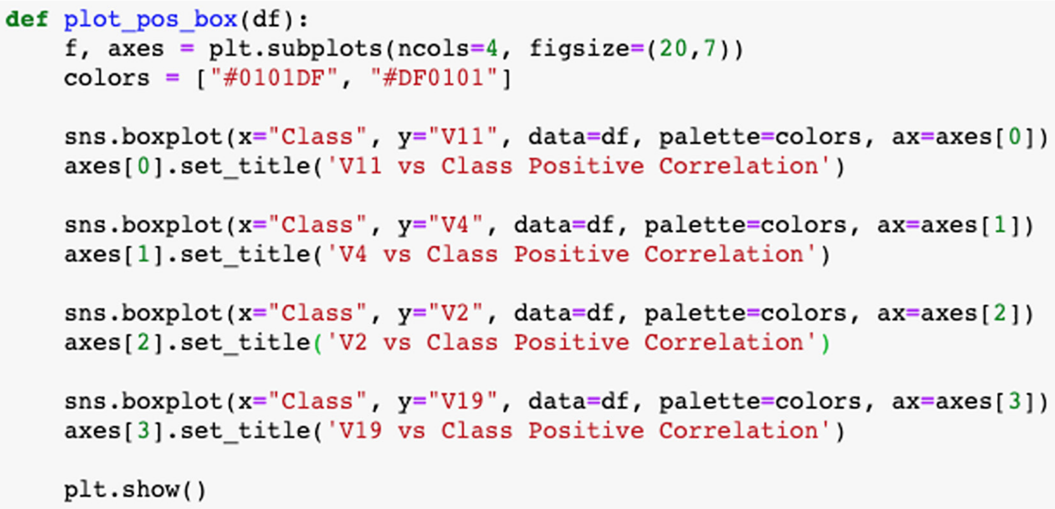
In this section, we will focus on analyzing the positive and negative correlation features, especially V1..., V28.

Here we will take a look at the positive and negative correlated features with different sampling techniques and finally choose one or two of the techniques for modeling only.

Positive feature (V1..., V28)

From the last section, we have seen the features V2, V4, V11, and V19 are positively correlated, so going forward, we will see these features only.

Before that, we will write a function that will help us to plot all the boxplots to compare them.



```
def plot_pos_box(df):
    f, axes = plt.subplots(ncols=4, figsize=(20,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="V11", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('V11 vs Class Positive Correlation')

    sns.boxplot(x="Class", y="V4", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('V4 vs Class Positive Correlation')

    sns.boxplot(x="Class", y="V2", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('V2 vs Class Positive Correlation')

    sns.boxplot(x="Class", y="V19", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('V19 vs Class Positive Correlation')

    plt.show()
```

Figure 4.76: Function for plot Box-and-whiskers plot for Positive Correlated Data

We will use the above code for all the datasets we have created so far like SMOTE, Tomek's Link, etc.

Original training data

Below we will see for the original dataframe and all the positively correlated features.

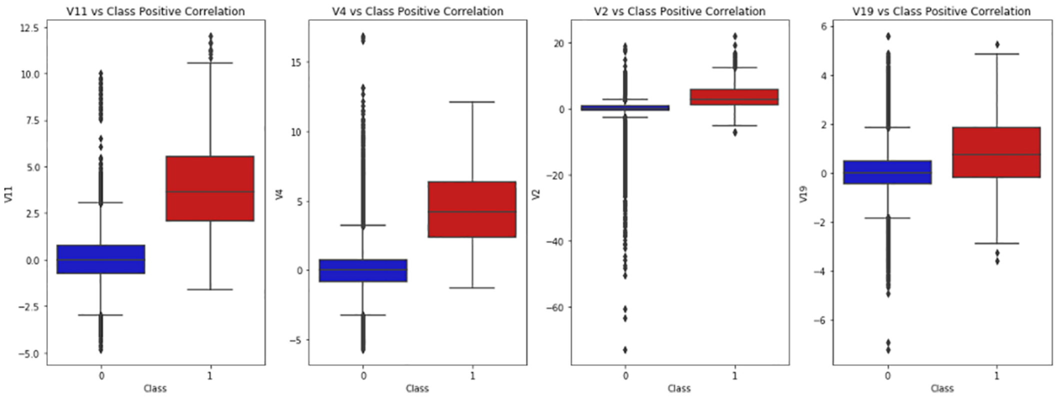


Figure 4.77: Original dataframe box plot

We can see that the original training dataframe has an extremely large number of outliers for class 0, and that is very bad for modeling, so we went for resampling the dataset.

All the positive features have an extremely high number of outliers.

Random under-sampling

Now let us see the same visualization for Random under sample and compare them.

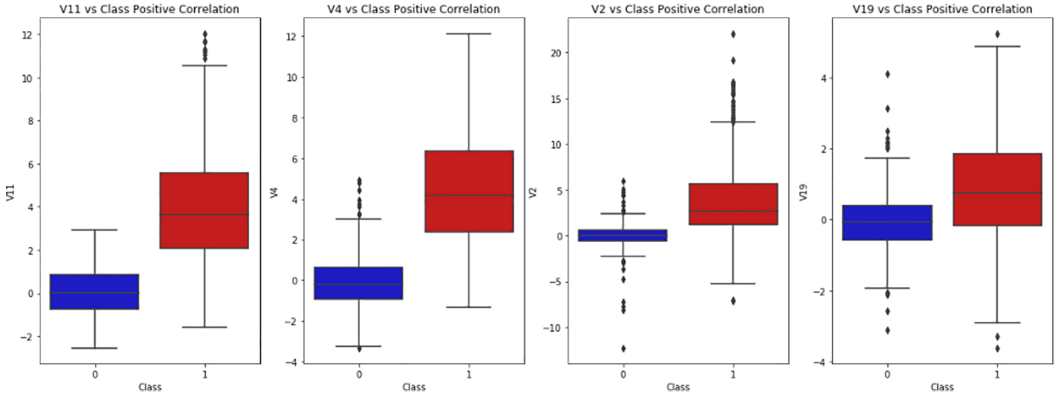


__Figure 4.78:__ Random under-sampling box plot

This plot looks way better as there are a smaller number of outliers, and this can be used for training machine learning models.

Random over-sampling

Now for random oversample, we are expecting the same outliers as the original training frame as there is no change in Class 0 data for random over-sampling.

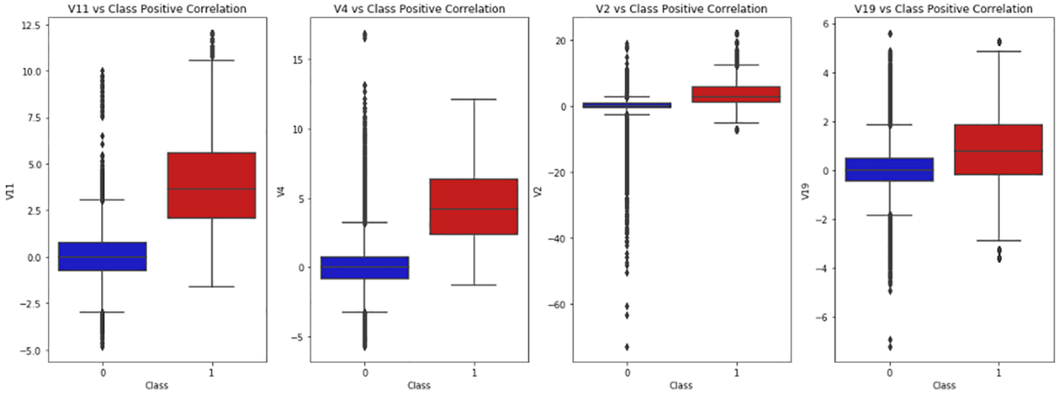


__Figure 4.79:__ Random over-sampling box plot

As this shows so many outliers, we cannot use this dataset for modeling. So, we will be ignoring this dataset.

Tomek's Link under-sampling

Tomek's Link under-sample was not good in the first place as there were extremely small number of records that were removed due to Tomek's link.

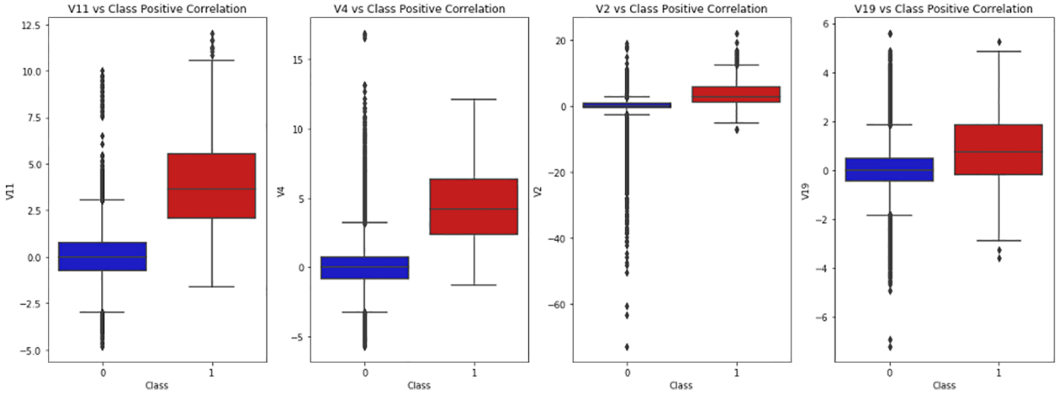


Figure 4.80: *Tomek's Link under-sampling box plot*

This plot is exactly similar to the original training dataset, as there were only 62 records that were removed from the majority class.

SMOTE

SMOTE is also an oversampling technique so that it will look similar to Random Oversampling.

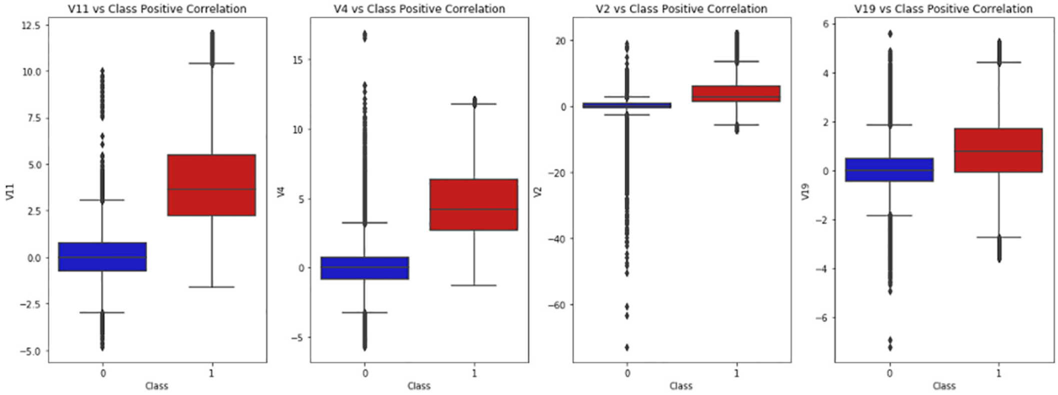


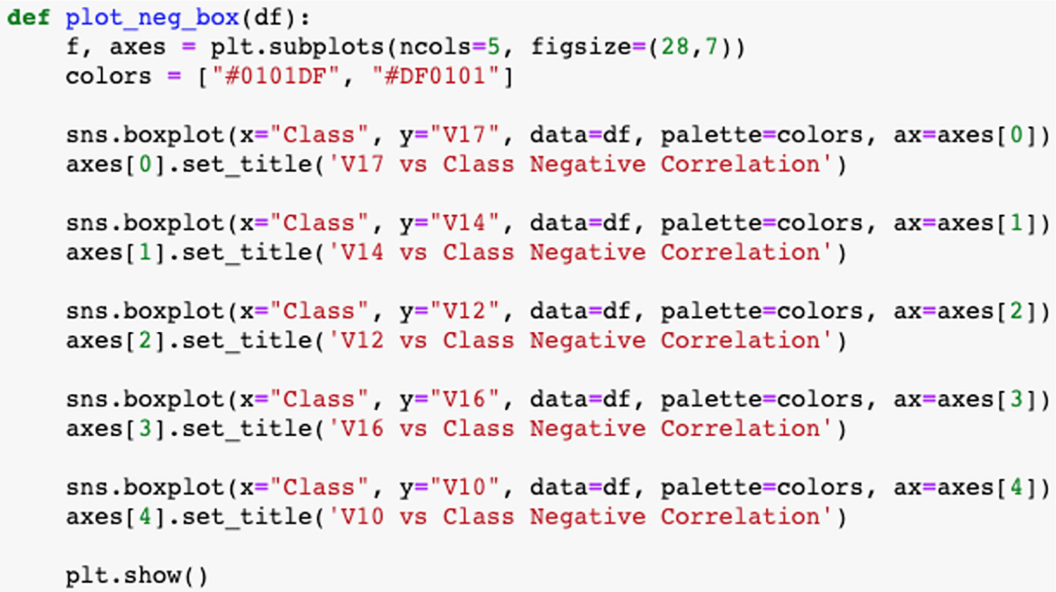
Figure 4.81: *SMOTE box plot*

Seeing the plot, we can confirm the hypothesis, but SMOTE is an extremely popular technique that is used in case of handling an imbalanced dataset.

We can train the models with one oversampled and one undersampled data and see how it will look.

Negative feature (V1.. ., V28)

Similarly, we will write a function for negatively correlated values to plot box-and-whiskers and find the optimal resampling technique.



```
def plot_neg_box(df):
    f, axes = plt.subplots(ncols=5, figsize=(28,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="V17", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('V17 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V14", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('V14 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V12", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('V12 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V16", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('V16 vs Class Negative Correlation')

    sns.boxplot(x="Class", y="V10", data=df, palette=colors, ax=axes[4])
    axes[4].set_title('V10 vs Class Negative Correlation')

    plt.show()
```

Figure 4.82: *Function for plot Box-and-whiskers plot for Negative Correlated Data*

Similar to the above code, you can visualize all the box and whiskers plot. But I won't be showing any of them as it will be repetitive. But it is recommended to plot all of them and analyze them.

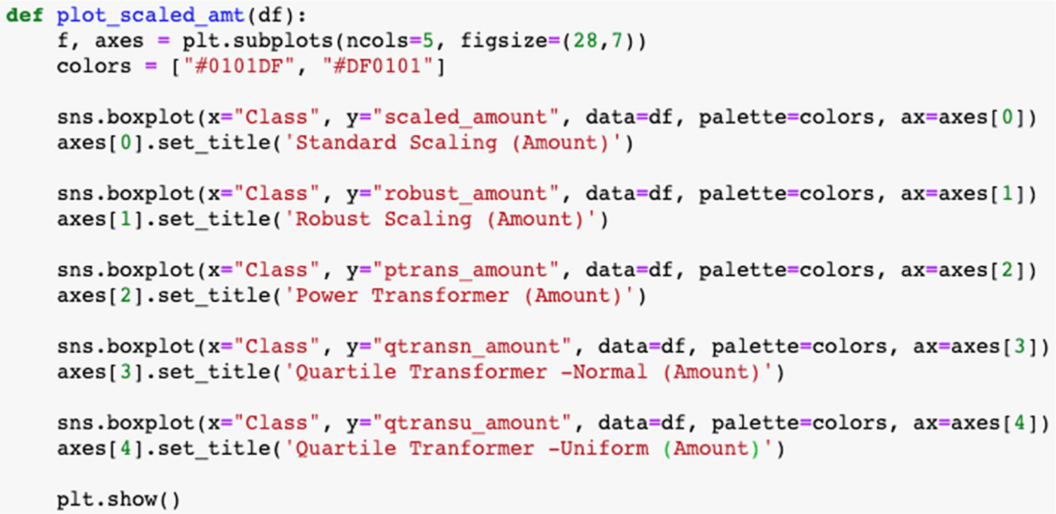
We have selected some of the features from V1..., V28 as we have decided from the correlation matrix, and we will be using SMOTE and Random Under-sampling henceforth.

Scaled features

Now we have to select which scaling technique to use for the modeling purpose. We have selected only two resampling techniques from the previous section. With that dataframe, we will be looking at the scaled data and select the optimal algorithm for the scaled feature ("`Time`" and "`Amount`").

Like before, we have written two functions, so we don't have to write the same code again and again.

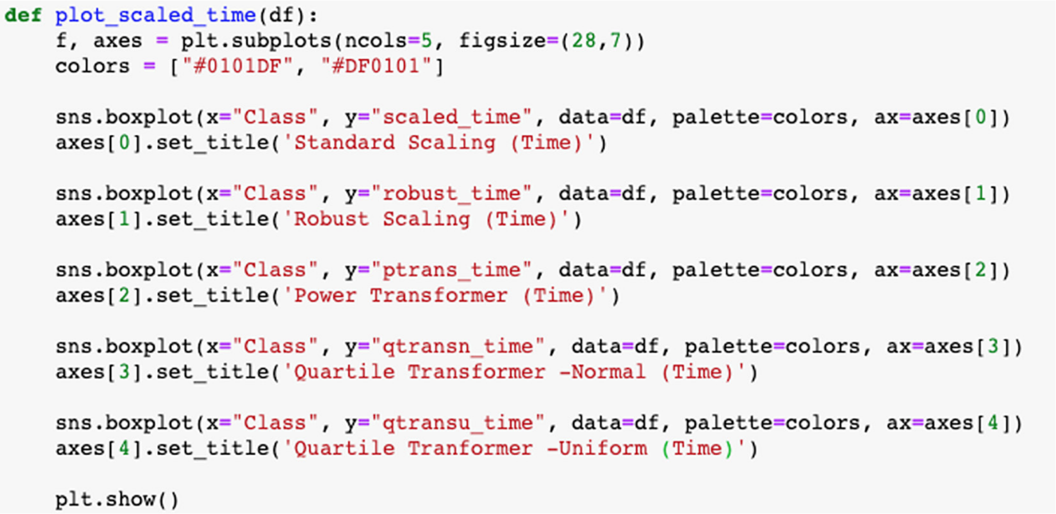
The first function is for the feature "`Amount`."



```
def plot_scaled_amt(df):
    f, axes = plt.subplots(ncols=5, figsize=(28,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="scaled_amount", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('Standard Scaling (Amount)')

    sns.boxplot(x="Class", y="robust_amount", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('Robust Scaling (Amount)')

    sns.boxplot(x="Class", y="ptrans_amount", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('Power Transformer (Amount)')

    sns.boxplot(x="Class", y="qtransn_amount", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('Quartile Transformer -Normal (Amount)')

    sns.boxplot(x="Class", y="qtransu_amount", data=df, palette=colors, ax=axes[4])
    axes[4].set_title('Quartile Tranformer -Uniform (Amount)')

    plt.show()
```

Figure 4.83: *Function for plot Box-and-whiskers plot for Scaled Amount*

And the second function is for the feature "`Time`."



```
def plot_scaled_time(df):
    f, axes = plt.subplots(ncols=5, figsize=(28,7))
    colors = ["#0101DF", "#DF0101"]

    sns.boxplot(x="Class", y="scaled_time", data=df, palette=colors, ax=axes[0])
    axes[0].set_title('Standard Scaling (Time)')

    sns.boxplot(x="Class", y="robust_time", data=df, palette=colors, ax=axes[1])
    axes[1].set_title('Robust Scaling (Time)')

    sns.boxplot(x="Class", y="ptrans_time", data=df, palette=colors, ax=axes[2])
    axes[2].set_title('Power Transformer (Time)')

    sns.boxplot(x="Class", y="qtransn_time", data=df, palette=colors, ax=axes[3])
    axes[3].set_title('Quartile Transformer -Normal (Time)')

    sns.boxplot(x="Class", y="qtransu_time", data=df, palette=colors, ax=axes[4])
    axes[4].set_title('Quartile Tranformer -Uniform (Time)')

    plt.show()
```

Figure 4.84: *Function for plot Box-and-whiskers plot for Scaled Time*

Now, let us analyze "`Time`" and "`Amount`" individually and come to a consensus about which scaling algorithm/s will we choose to go ahead with modeling.

Time

Let us first make a yardstick to compare the results with. We will plot first for the original dataframe.

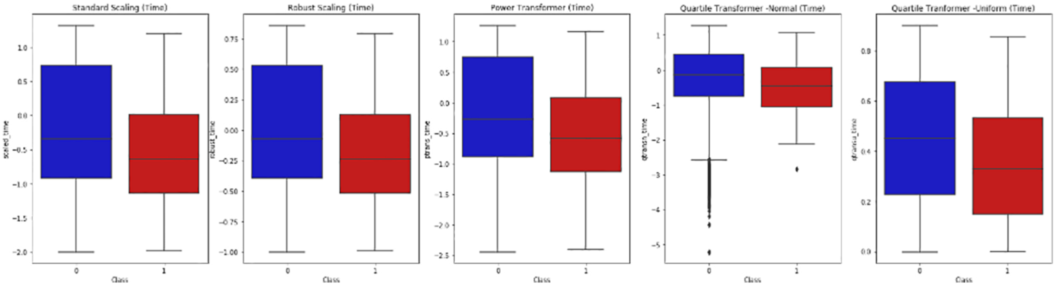


Figure 4.85: *Original Training Data for Box plot*

We can see for "Quantile Transformer -Normal" there is skewness and outliers in the plot. We now have to compare this entire plot with Random Undersampling and SMOTE. We will go ahead with the under-sampling dataset and see how it will look.

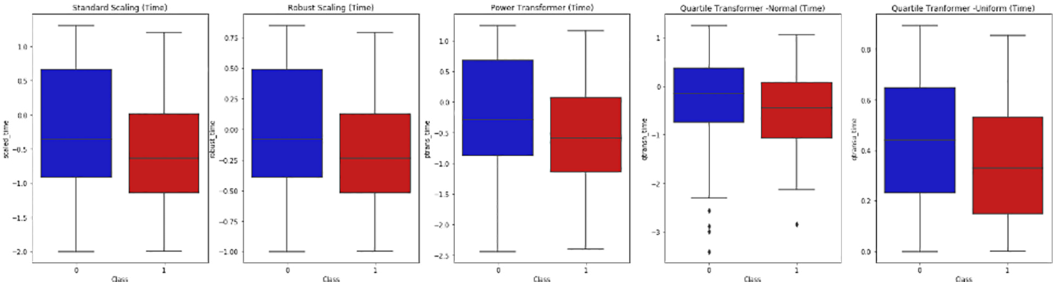


Figure 4.86: *Random under-sampling for Box plot*

For this dataset, we can see the same thing that for "Quantile Transformer -Normal," there are outliers in the plot. Apart from that, all looks good for use. Previously, we have seen that "Quantile Transformer -Uniform" actually changed the distribution so that can be ignored as well.

Let us see for SMOTE as well, and conclude with some optimal results.

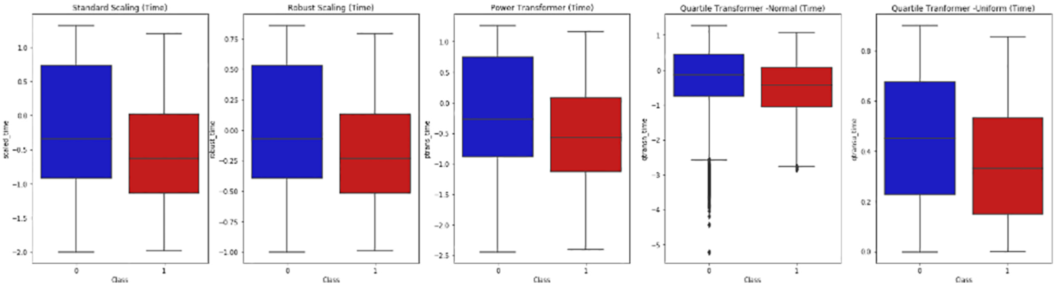


Figure 4.87: *SMOTE for Box plot*

For the SMOTE dataframe, we see the same result, so, for the feature "Time," we can choose any algorithm apart from "Quantile Transformer."

We have to be in sync with the feature "Amount" as well. It is not mandatory to select the same scaling algorithm for all the features, but whatever we do, we need to have some justification for that action.

Amount

We are looking for a different behavior/pattern for the amount. Let us first see the original training data distribution.

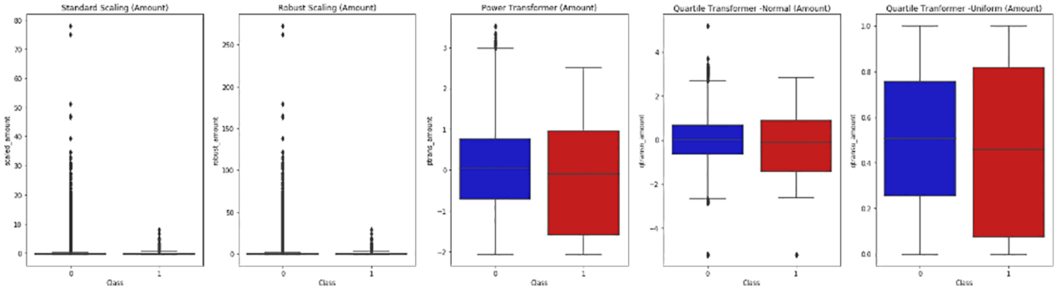


Figure 4.88: Original Training Data for Box plot

Here, for the amount, we cannot ignore or eliminate the outliers as they will give us a lot of information. We will try to stick with the original distribution with some minor changes.

Let us see for the random under-sampled dataset as we know it reduced the major class, so the density of the outliers will be reduced.

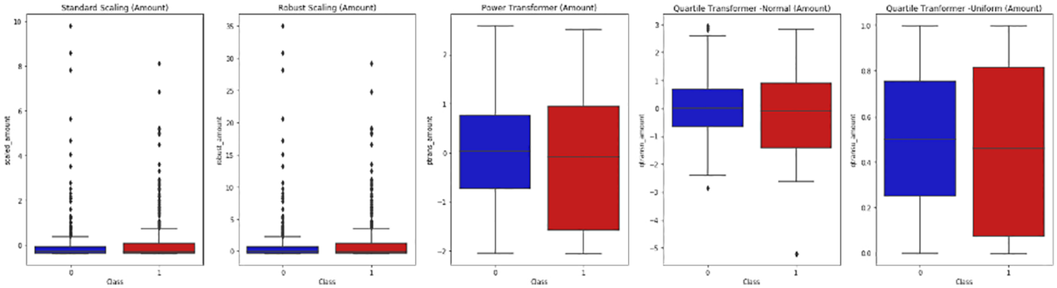


Figure 4.89: Random under sampling for Box plot

We can see outliers have reduced, and Power Transformer looks good in terms of distribution. But we will also look at Robust Scaler as it more or less gives us the original distribution. "Quantile Transformer - Normal" looks good, but it has already performed badly for the future "Time," so we will ignore it.

We will also take a look at the SMOTE dataset and see its plot.

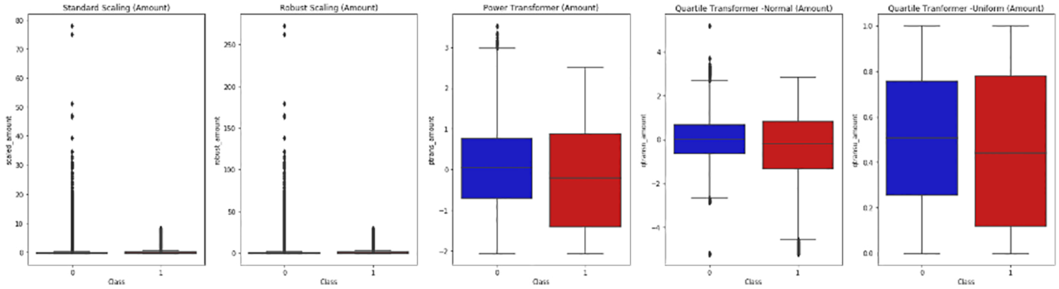


Figure 4.90: *SMOTE for Box plot*

SMOTE plot shows a similar thing like the original training dataset; hence we will stick with our last decision.

So, we will go ahead with both Robust Scaling and Power Transformer for both "Time" and "Amount." Finally, we will have four dataframe to run all the models.

SMOTE Dataset 1 - Robust Scaled Time, Robust Scaled Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

SMOTE Dataset 2 - Power Transformer Time, Power Transformer Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

Random Under-sample Dataset 1 - Robust Scaled Time, Robust Scaled Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

Random Under-sample Dataset 2 - Power Transformer Time, Power Transformer Amount, V2, V4, V11, and V19 (Positive), V17, V14, V12, V16 and V10 (Negative)

We have to do at least four iterations from the database and find the best data for the model. Then separately, we have to tune the model as well.

Modeling

We always need to know that data preparation is 70% of the entire pipeline, so we have to give importance to the data. If the data is well prepared, then modeling will be ease, and we can achieve optimal accuracy for the model. Then we need to tune the model iteratively to improve it further.

Machine Learning algorithms

Here we will see multiple algorithms, but here we will explain some basic algorithms to make the foundation.

Logistic regression[30]

Logistic regression also known as a logit regression, is a statistical model and is somewhat similar to the concepts of linear regression. The basic version of Logistic Regression utilizes logistic function to model a binary dependent variable[31].

Logistic regression is generally used for classification problems in the field of Machine Learning.

Why use logistic regression (comparison to linear regression)?

Linear regression is used for the regression problem now with the same if we need to make some classification; we can use the sigmoidal function[32] , which ranges from [0, 1].

What is the logit function[33]?

In statistics, `logit` function creates a map of probability values from [0, 1] to [-∞, +∞].

$$logit(p) = log\left(\frac{p}{1-p}\right) = log(p) - log(1-p)$$

$$\Rightarrow -log\left(\frac{1}{p} - 1\right)$$

We can see the plot for the logit function below:

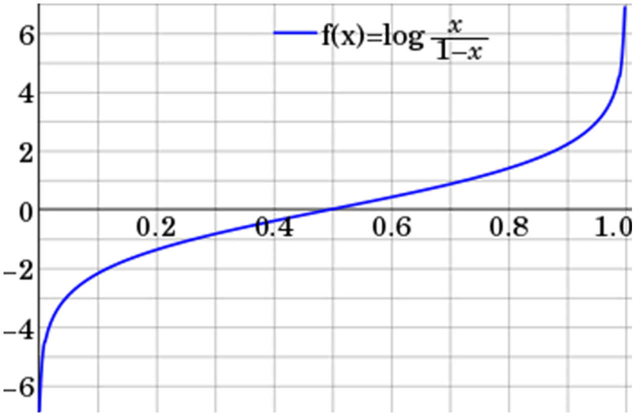


Figure 4.91: *Logit (Wikipedia)*

[30] “Logistic regression - Wikipedia.” https://en.wikipedia.org/wiki/Logistic_regression.

[31] “Binary Dependent Variables.” http://personal.rhul.ac.uk/uhte/006/ec2203/Binary%20Dependent%20Variables.pdf.

[32] “Sigmoid Function -- from Wolfram MathWorld.” http://mathworld.wolfram.com/SigmoidFunction.html.

[33] “Logit - Wikipedia.” https://en.wikipedia.org/wiki/Logit.

We can confirm the range from the above image.

Now, the sigmoidal logistic function[34] which is nothing but the inverse of the logit function

$$logit^{-1}(\alpha) = logistic(\alpha) = \frac{1}{1+\exp(-\alpha)} = \frac{\exp(\alpha)}{\exp(\alpha)+1}$$

The above equation is nothing but a sigmoid function[35]. Below is how a Sigmoid look like.

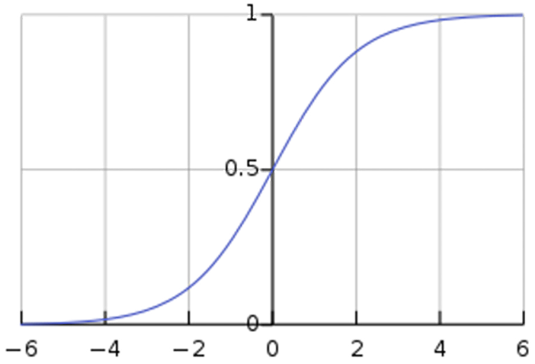


***Figure 4.92:** Sigmoid (Wikipedia)*

Sigmoid only ranges from [0, 1].

> ***Note:*** Logistic function and logit function are two different things, and it should not be confused at all.

Derivation:

Let us start with the Linear Regression equation.

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

Here, the linear regression will give us a continuous value, which will range from [– ∞, + ∞]. But what we need is a probabilistic value as an output.

We can use odd ratio(OR)[36] as an output:

$$odds(p) = \frac{p}{1-p}$$

[34] “Logistic function - Wikipedia.” https://en.wikipedia.org/wiki/Logistic_function.
[35] “Sigmoid function - Wikipedia.” https://en.wikipedia.org/wiki/Sigmoid_function.
[36] “Odds ratio - Wikipedia.” https://en.wikipedia.org/wiki/Odds_ratio.

And it ranges from [0, +∞] still, now we cannot equate this with the linear equation as the range is not the same but if we take the Log(Natural[37]) of Odds, i.e., Logit.

$$logit(p) = log\left(\frac{p}{1-p}\right)$$

Logit ranges from [-∞, +∞]

$$log\left(\frac{y}{1-y}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

Let us represent $\beta_0 + \beta_1 x_1 + ... + \beta_n x_n$ as z

$$\therefore log\left(\frac{y}{1-y}\right) = z$$

$$\Rightarrow \frac{y}{1-y} = \exp(z)$$

$$\Rightarrow \frac{1-y}{y} = \frac{1}{\exp(z)}$$

$$\Rightarrow \frac{1}{y} = \frac{1}{\exp(z)} + 1$$

$$\Rightarrow y = \frac{\exp(z)}{\exp(z)+1}$$

If we carefully take a look into the last equation, it is the same as the sigmoid function.

By now, we know the sigmoid function gives us a continuous value between [0, 1].

This can be treated as probability values in terms of prediction value.

To make the continuous value fall into two classes, then we will use something known as the threshold. A threshold is nothing but a value between the range [0, 1], which will help the predicted probability value to be assigned so some binary class (either Class 0/Negative or Class 1/Positive).

Logistic regression for multiple classes can be achieved either using the one-vs-rest[38] scheme[39] in which for each class a binary classification problem (whether the data

[37] "Natural logarithm - Wikipedia." https://en.wikipedia.org/wiki/Natural_logarithm.

[38] "Multiclass classification - Wikipedia." https://en.wikipedia.org/wiki/Multiclass_classification.

[39] "sklearn.multiclass.OneVsRestClassifier — scikit-learn 0.22.1 ...."
http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html.

belongs to that class or not) or changing the loss function from sigmoid to cross-entropy[40] loss.

Decision tree

A decision tree is a popular classification algorithm, but it can be used as a regression algorithm as well.

Structure of decision tree

The decision tree resembles a binary tree[41] which we generally study under Data Structures. This tree is upside down of an original tree as "roots" starts from the top, and as we go down, we have "leaves."

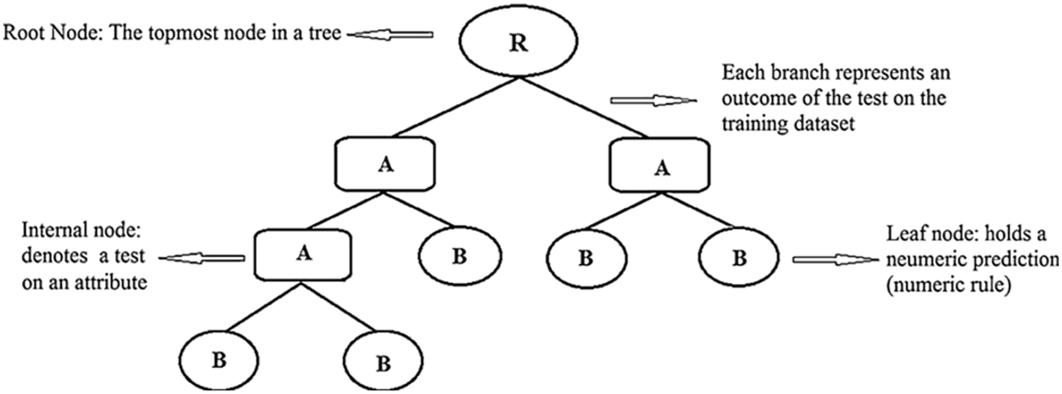

Root Node: The topmost node in a tree
R
Each branch represents an outcome of the test on the training dataset
A
A
Internal node: denotes a test on an attribute
A
B
B
B
Leaf node: holds a neumeric prediction (numeric rule)
B
B

Figure 4.93: *Decision tree structure*

This is an important structure to remember, and the internal node is a decision node, and it splits the data.

Algorithm

- **Step 1**: Select the best decision node using **Attribute Selection Methods (ASM)**
- **Step 2**: Now using the selected Decision Node break the dataset into smaller subsets
- **Step 3**: Repeat Step 1 and Step 2 for all the child nodes until and unless the below conditions are met.
    - o There are no more remaining features to deal with
    - o There are no more records
    - o All records belong to the same feature

[40] "Cross entropy - Wikipedia." https://en.wikipedia.org/wiki/Cross_entropy.

[41] "Binary tree - Wikipedia." https://en.wikipedia.org/wiki/Binary_tree.

Mathematics behind ASM

Iterative Dichotomiser 3 (ID3)[42]

In ID3, information gain is calculated for each remaining attribute. The attribute with the largest information gain is used as a decision node that splits the set on this iteration.

- Entropy[43]

  Entropy is a measure of the uncertainty in the dataset S.

  $$H(S) = \sum_{x \in X} -p(x)\log_2 p(x)$$

  Where S is the current dataset, X is the features in set S, $p(x)$ is the proportion of the number of elements in class x to the count of elements in set S.

- Information Gain[44]

  Information Grain is a measure for entropy before and after set S is split on feature A.

  $$IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t) = H(S) - H(S \mid A)$$

There are other techniques as well, like Gini, C4.5, CART, etc.

Support Vector Machine (SVM)[45]

Support Vector Machine (**SVM**) is a model generally used for classification and regression problems. It can solve linear and nonlinear problems that help the algorithm to tackle real-time projects.

The concept of Linear SVM is simple. The algorithm creates a line or a hyperplane[46] which separates the data into classes.

How does it work?

Let us see the working of Linear SVM visually, as it will be simpler to understand. Other types of SVM will be recommended for further studies.

[42] "ID3 algorithm - Wikipedia." https://en.wikipedia.org/wiki/ID3_algorithm.

[43] "Entropy (information theory) - Wikipedia." https://en.wikipedia.org/wiki/Entropy_(information_theory).

[44] "Information gain in decision trees - Wikipedia." https://en.wikipedia.org/wiki/Information_gain_in_decision_trees.

[45] "Support Vector Machines 1 Support Vector Machines Revisited." 12 Apr. 2014, https://www.stat.berkeley.edu/~arturof/Teaching/EE127/Notes/support_vector_machines.pdf.

[46] "Hyperplane - Wikipedia." https://en.wikipedia.org/wiki/Hyperplane.

First, we will plot all the data points with class, and then we draw multiple hyperplanes[47], here, it will try to separate the two classes.

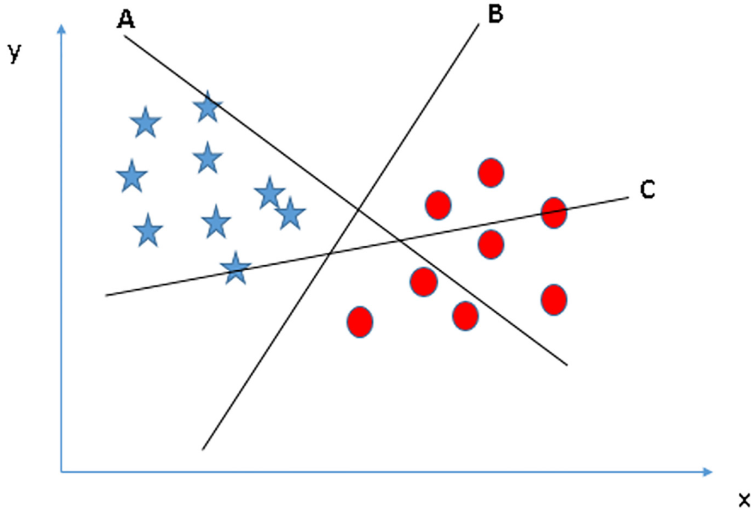


Figure 4.94: *Draw Hyperplane (Analytics Vidhya)*

We have three hyperplanes[48] , namely A, B, C. Now, we need to find the best line that separates both the classes.

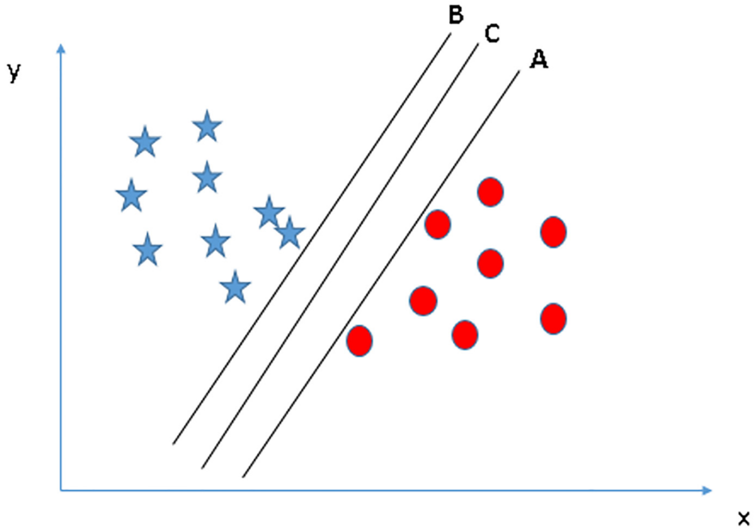


Figure 4.95: *Find the hyperplane separating them (Analytics Vidhya)*

[47] "Hyperplane -- from Wolfram MathWorld." http://mathworld.wolfram.com/Hyperplane.html.

[48] "8.2 Planes and Hyperplanes - Faculty." http://faculty.bard.edu/belk/math213/PlanesAndHyperplanes.pdf.

Say, we have three hyperplanes that separate both the classes like the above image. Now, we need to choose one optimal hyperplane.

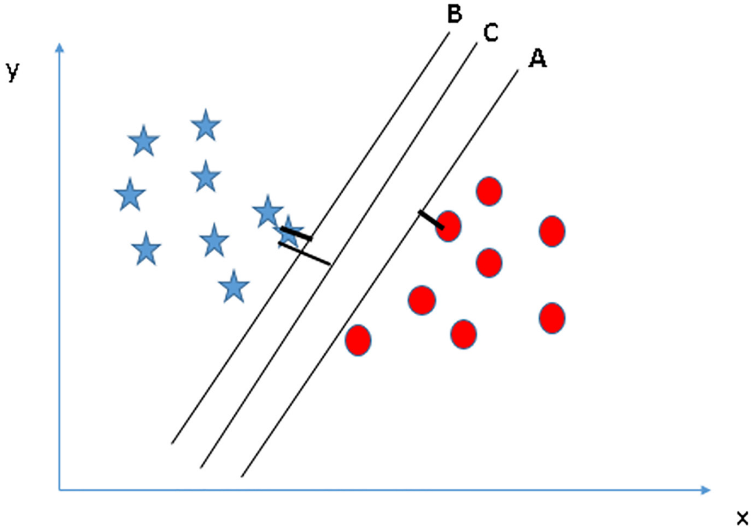


Figure 4.96: *Find the best hyperplane separating them (Analytics Vidhya)*

To select optimal hyperplane from the three planes, we need to see the margin like the above image. Margin is the distance from the hyperplane to the nearest data point with both the classes. Our target is to maximize both the distance from two classes and come to an optimal point. Here we can see the particular hyperplane is C.

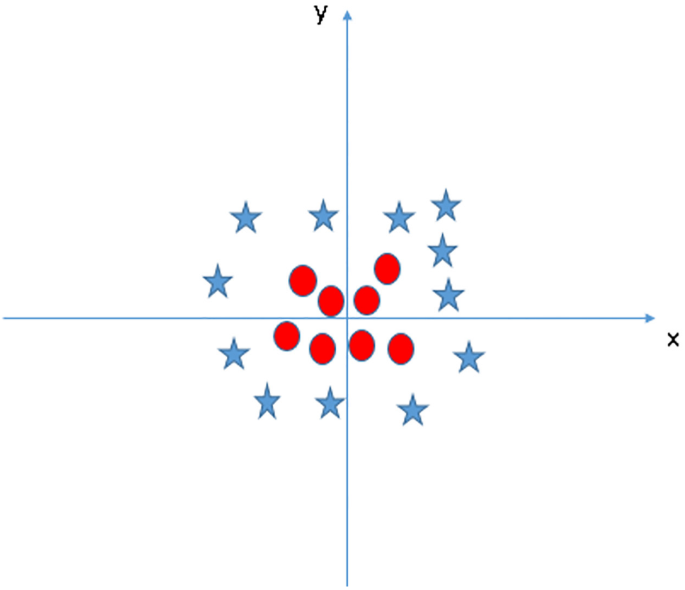


Figure 4.97: *Non-linearly separable (Analytics Vidhya)*

In the above example, we saw that we could easily divide both the classes using a Line, Plane, Hyperplane, but that won't be the case all the time.

Let us see one more example where we cannot use a linear SVM to solve the problem.

From the above image, we can see that the data point and class won't be separated by a hyperplane. We have multiple techniques to handle nonlinear decision boundaries using kernels[49].

For the above plot, we can transform the data and plot like the below figure.

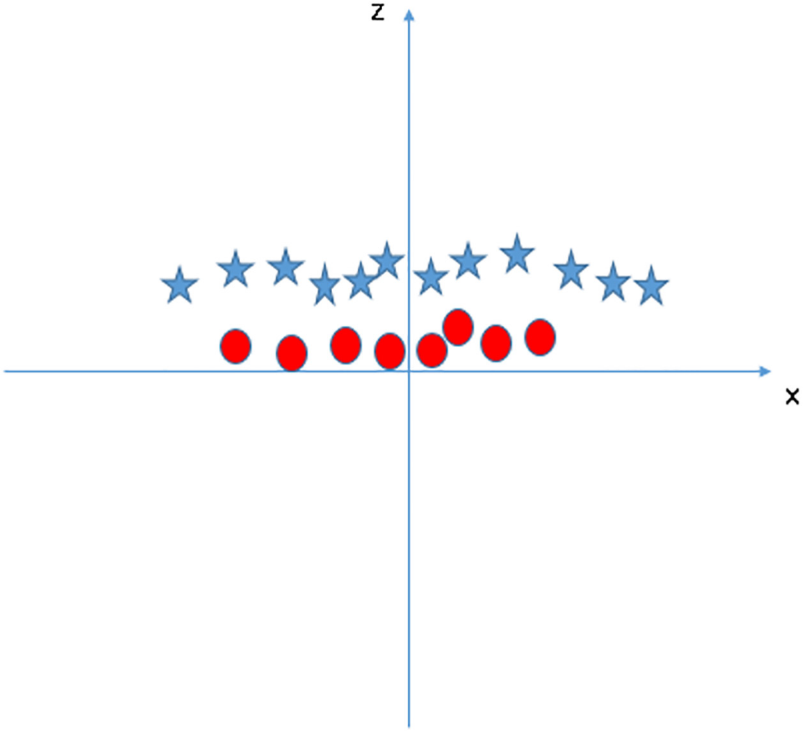


Figure 4.98: *Transform Data (Analytics Vidhya)*

Here for this particular example, we are projecting[50] [51] the same data points to a higher dimension and trying to separate it by a hyperplane. But when we transform back to the original dimension, it will look like the below image.

[49] "Kernel method - Wikipedia." https://en.wikipedia.org/wiki/Kernel_method.

[50] "Projection (linear algebra) - Wikipedia." https://en.wikipedia.org/wiki/Projection_(linear_algebra).

[51] "Projection (mathematics) - Wikipedia." https://en.wikipedia.org/wiki/Projection_(mathematics).

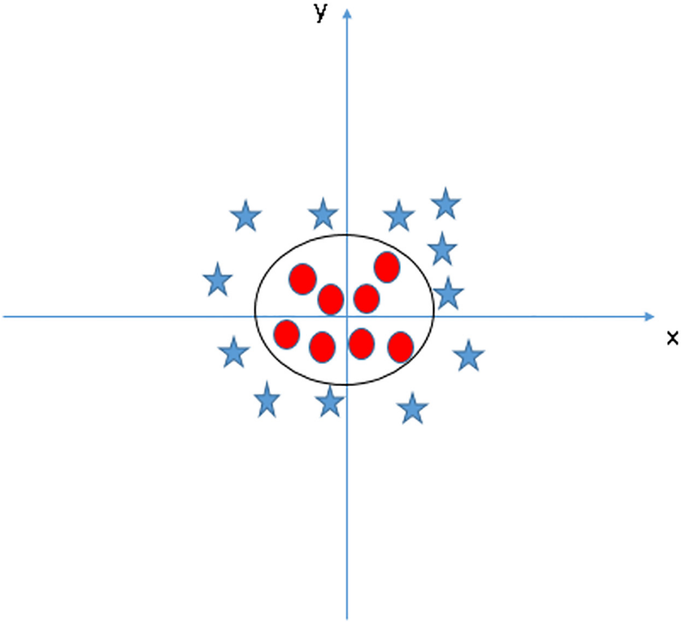


***Figure 4.99:** Separate using nonlinear kernel (Analytics Vidhya)*

After bringing back to the original dimension, we can see an elliptical decision boundary, or we can say this as nonlinear hypersurface. In SVM, we generally achieve this by Radial Basis Kernel[52].Next, we will take a brief look at the mathematics behind linear SVM.

Basic mathematics behind the Linear Hard-Margin Classifier(Linear SVM) for Strictly Separable Case:

Let us start by defining the plane.

$$d(x, w, b) = w^T x + b = \sum_{i=1}^{n} w_i x_i + b$$

So, now we need to do a classification. If:

1. d(x, w, b) > 0, classify x as class 1 (i.e. its associated y = +1)
2. d(x, w, b) < 0, classify x as class 2 (i.e. its associated y = −1)
3. d(x, w, b) = 0, then x is equi-probable for both classes so, as per the business problem we classify this data point

Where, d(x, w, b) is the distance of point x from the hyperplane. Greater the distance stronger the classification of x.

[52] "Radial basis function kernel - Wikipedia." https://en.wikipedia.org/wiki/Radial_basis_function_kernel. Accessed 8 Jan. 2020.

After finding the optimal of the hyperplane w^*, b^* our decision boundary[53] has a form like below:

$$\hat{f}(x) = sgn(d(x, w^*, b^*)) = \text{sgn}(w^{*T}\ x + b); \text{sgn}(x) = \begin{cases} 1 & if\ \ x > 0 \\ -1 & if\ \ x < 0 \\ 0 & x = 0 \end{cases}$$

These are the basics of SVM that can be covered as per the scope of the book, but there are other classifiers as well for further reading.

Next section, we will use the different combinations of the dataset and see the performance of the same algorithm.

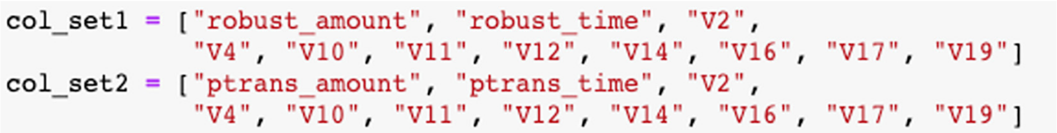
Model training

In this section, we will train different models with different parameters of it on all datasets and find the optimal model to use for this problem statement.

Before we start with it, we need to make the dataset ready for training.

Data preparation

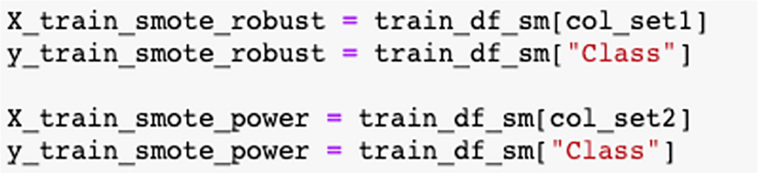
We already have decided with the list of columns we will use to for training as shown below:



```
col_set1 = ["robust_amount", "robust_time", "V2",
            "V4", "V10", "V11", "V12", "V14", "V16", "V17", "V19"]
col_set2 = ["ptrans_amount", "ptrans_time", "V2",
            "V4", "V10", "V11", "V12", "V14", "V16", "V17", "V19"]
```

Figure 4.100: *Multiple set of columns*

We will have two sets of columns, and we will also use different resampled dataframe. In this case, we will use SMOTE and Random under-sampled dataset. So, there will be a total of 4 datasets that we need to train.

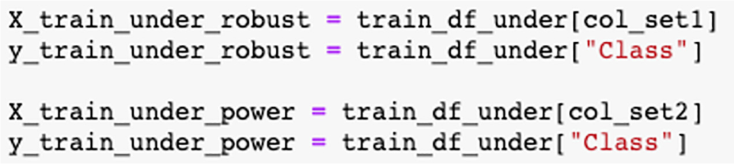


```
X_train_smote_robust = train_df_sm[col_set1]
y_train_smote_robust = train_df_sm["Class"]

X_train_smote_power = train_df_sm[col_set2]
y_train_smote_power = train_df_sm["Class"]
```

Figure 4.101: *SMOTE data preparation*

[53] "Decision boundary - Wikipedia." https://en.wikipedia.org/wiki/Decision_boundary.

The above set is for SMOTE with both Robust Sampling and Power Transformed data. We will similarly have two dataframe for Random Under Sample.



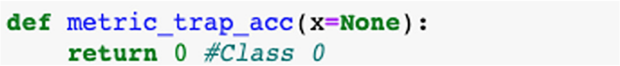
```
X_train_under_robust = train_df_under[col_set1]
y_train_under_robust = train_df_under["Class"]

X_train_under_power = train_df_under[col_set2]
y_train_under_power = train_df_under["Class"]
```

Figure 4.102: *Random under-sample data preparation*

So, we now have four training datasets, and we will use multiple algorithms to find the best algorithm out of it.

Metric trap

This is one of the most important points which we need to ponder upon, i.e., the evaluation metrics. Here "accuracy" cannot be used as a metric because this is an imbalanced dataset, so we make a function that will always return class zero, then the accuracy of the model(function) is more than 95%.



```
def metric_trap_acc(x=None):
    return 0 #Class 0
```

Figure 4.103: *Pseudo model (should not be used)*

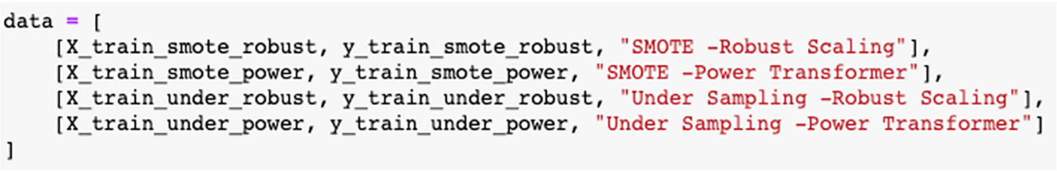
The above function/pseudo-model has great accuracy as it will always return class zero. The probability of getting class zero for the imbalanced data set is more than 95%.

But, for quick analysis, we will take a look at accuracy to make some ideas and guess about the model's performance.

Training & evaluation

In this section, we will train all the models with four different kinds of the dataset and evaluate some of the models, rest models can be evaluated by you as a practice and get more insight about the data and the process.

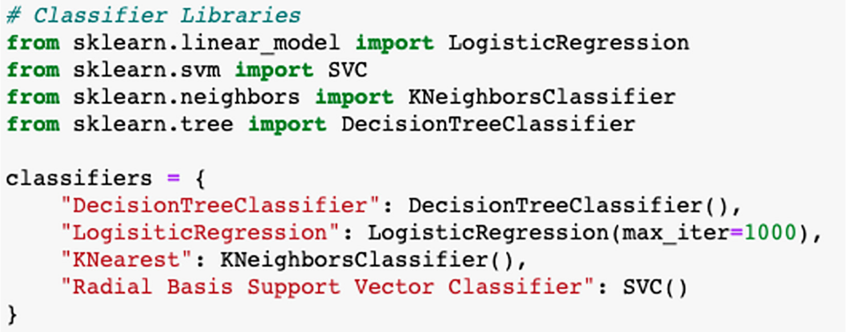
Let us prepare the dataset, and we will use them in a loop to train.



```
data = [
    [X_train_smote_robust, y_train_smote_robust, "SMOTE -Robust Scaling"],
    [X_train_smote_power, y_train_smote_power, "SMOTE -Power Transformer"],
    [X_train_under_robust, y_train_under_robust, "Under Sampling -Robust Scaling"],
    [X_train_under_power, y_train_under_power, "Under Sampling -Power Transformer"]
]
```

Figure 4.104: *Preparation of training data*

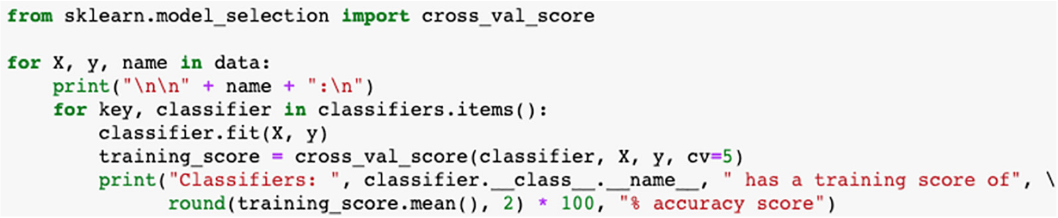
Above, we can see that we will be using one variable for all the training data. We are planning to write a small code that will help us to train all the models for all the data at once using cross-validation.



```
# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

classifiers = {
    "DecisionTreeClassifier": DecisionTreeClassifier(),
    "LogisiticRegression": LogisticRegression(max_iter=1000),
    "KNearest": KNeighborsClassifier(),
    "Radial Basis Support Vector Classifier": SVC()
}
```

Figure 4.105: *Initializing models*

We have a separate set of testing data, but we need to test with the training data as well, using the k-fold cross-validation technique. This will decrease the chance of selection bias. With this process, we will select the optimal dataset and some of the models for the evaluation.

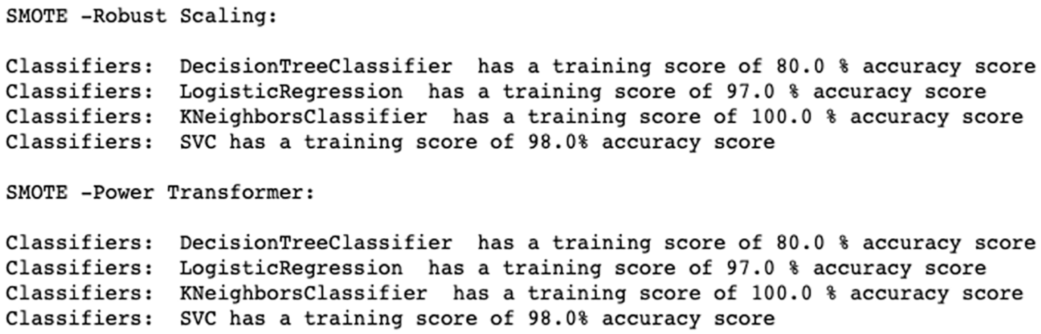


```
from sklearn.model_selection import cross_val_score

for X, y, name in data:
    print("\n\n" + name + ":\n")
    for key, classifier in classifiers.items():
        classifier.fit(X, y)
        training_score = cross_val_score(classifier, X, y, cv=5)
        print("Classifiers: ", classifier.__class__.__name__, " has a training score of", \
              round(training_score.mean(), 2) * 100, "% accuracy score")
```

Figure 4.106: *Fitting and cross-validating*

From the above code, we will get the statistics about training data accuracy for all the combinations. With every step, we should always aim to reduce the number of computations and have limited models to evaluate.

Let's start with the SMOTE dataset and see how it looks.

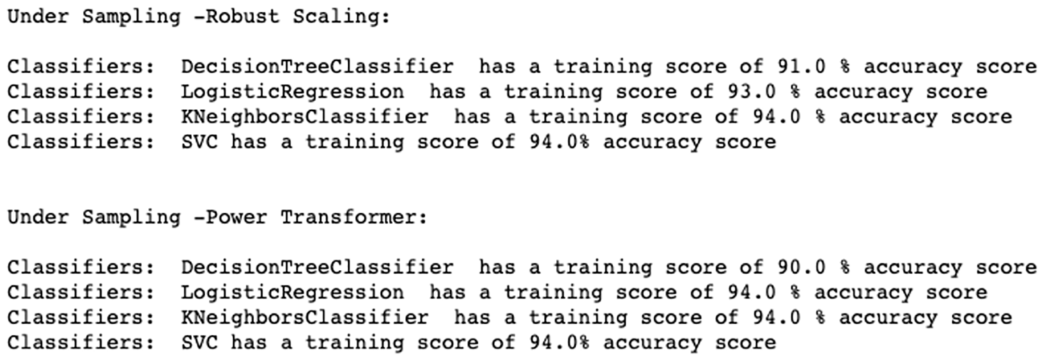


```
SMOTE -Robust Scaling:

Classifiers:  DecisionTreeClassifier  has a training score of 80.0 % accuracy score
Classifiers:  LogisticRegression  has a training score of 97.0 % accuracy score
Classifiers:  KNeighborsClassifier  has a training score of 100.0 % accuracy score
Classifiers:  SVC has a training score of 98.0% accuracy score

SMOTE -Power Transformer:

Classifiers:  DecisionTreeClassifier  has a training score of 80.0 % accuracy score
Classifiers:  LogisticRegression  has a training score of 97.0 % accuracy score
Classifiers:  KNeighborsClassifier  has a training score of 100.0 % accuracy score
Classifiers:  SVC has a training score of 98.0% accuracy score
```

Figure 4.107: *Cross-validation training score for SMOTE*

We have a similar result for both Robust Scaling and Power Transformer. For a new algorithm, we are using k-NN as well, but we won't be using it. We can see that k-NN's accuracy is 100%, and it is highly probable and possible that it is suffering from overfitting.

From the SMOTE section, we can choose any algorithm, and we will decide according to the random under Sample results.



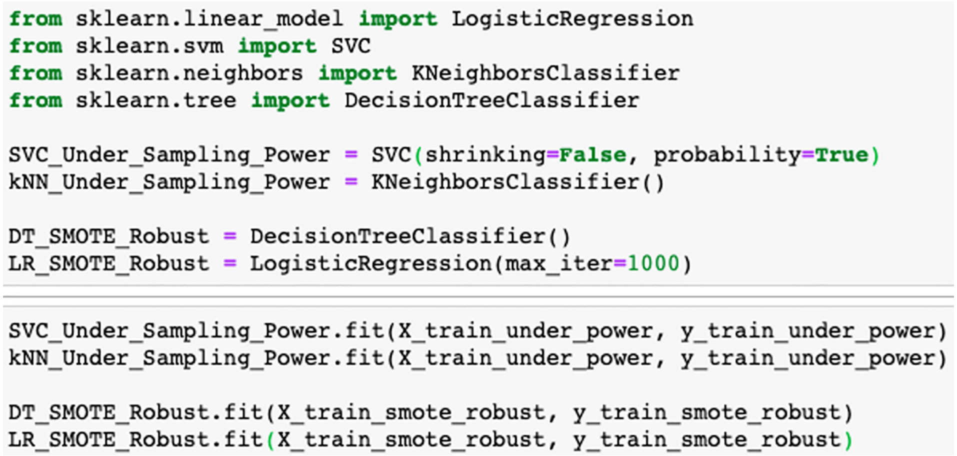
```
Under Sampling -Robust Scaling:

Classifiers:  DecisionTreeClassifier  has a training score of 91.0 % accuracy score
Classifiers:  LogisticRegression  has a training score of 93.0 % accuracy score
Classifiers:  KNeighborsClassifier  has a training score of 94.0 % accuracy score
Classifiers:  SVC has a training score of 94.0% accuracy score


Under Sampling -Power Transformer:

Classifiers:  DecisionTreeClassifier  has a training score of 90.0 % accuracy score
Classifiers:  LogisticRegression  has a training score of 94.0 % accuracy score
Classifiers:  KNeighborsClassifier  has a training score of 94.0 % accuracy score
Classifiers:  SVC has a training score of 94.0% accuracy score
```

Figure 4.108: *Cross-validation training score for under-sampling*

Random under-sampling has a different result and is not as great as the oversampling results. We can use Decision Tree, Logistic Regression for SMOTE, and Support Vector Machine, k-NN for Random Under Sampling. From now on, we will be using the test data to evaluate the model.

> ***Note:*** The order of the features in the dataset during training and testing should be the same else it will give us a wrong result.

Let us write a code snippet to do the same, as we mentioned above.

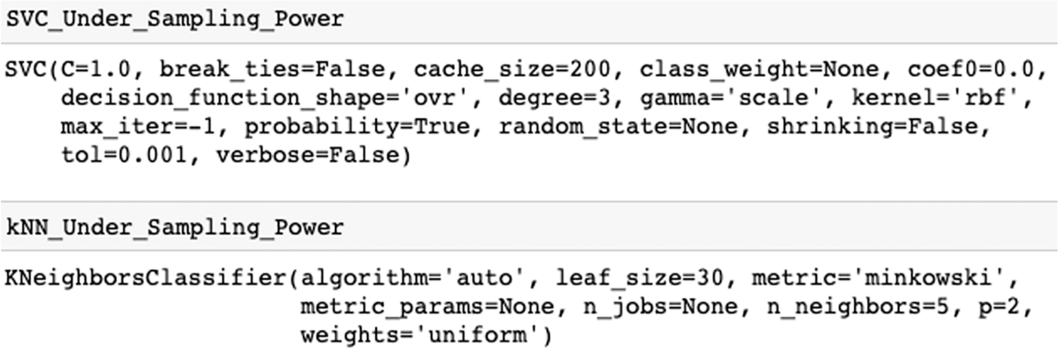


```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

SVC_Under_Sampling_Power = SVC(shrinking=False, probability=True)
kNN_Under_Sampling_Power = KNeighborsClassifier()

DT_SMOTE_Robust = DecisionTreeClassifier()
LR_SMOTE_Robust = LogisticRegression(max_iter=1000)
```

```python
SVC_Under_Sampling_Power.fit(X_train_under_power, y_train_under_power)
kNN_Under_Sampling_Power.fit(X_train_under_power, y_train_under_power)

DT_SMOTE_Robust.fit(X_train_smote_robust, y_train_smote_robust)
LR_SMOTE_Robust.fit(X_train_smote_robust, y_train_smote_robust)
```

Figure 4.109: *Training models*

From now on, we will use these names (*Figure 4.109*) instead of the model name and the scaling name every time. We will also take a look at the model configuration for the above algorithms.

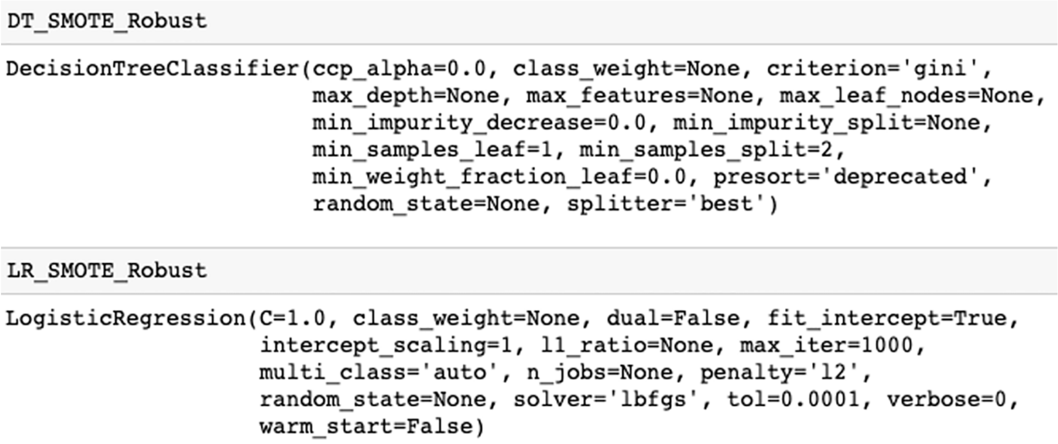


```
SVC_Under_Sampling_Power

SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=True, random_state=None, shrinking=False,
    tol=0.001, verbose=False)
```

```
kNN_Under_Sampling_Power

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

Figure 4.110: *Random under-sample model configuration*

We can see different parameters for Support Vector Classifier and k-NN models. It is advised to take a look at the parameters from the sklearn documentation.



```
DT_SMOTE_Robust

DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
LR_SMOTE_Robust

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Figure 4.111: *SMOTE Model configuration*

The above configurations are for oversampled dataset trained algorithms. This will help us to fine-tune the model if results are not as expected.

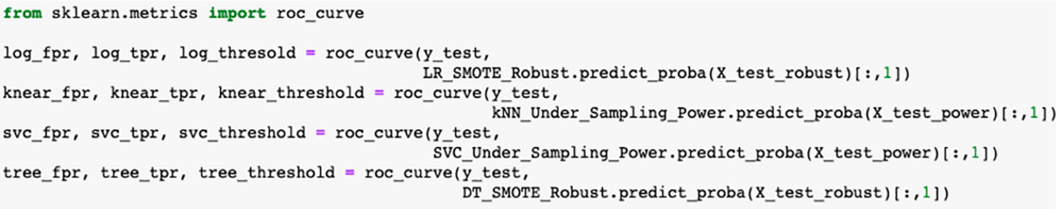
We had discussed before the metric trap, and just using accuracy will give us wrong information. There are multiple metrics we can see. But we will use Receiver Operating Characteristics[54] (ROC) to analyze all the models.

[54] "Receiver operating characteristic - Wikipedia." https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

ROC[55] is a way to analyze the performance of a binary classifier system as its discrimination threshold is valid.

We will first take a look at the ROC Curve then it will be simpler to explain what ROC Score is.

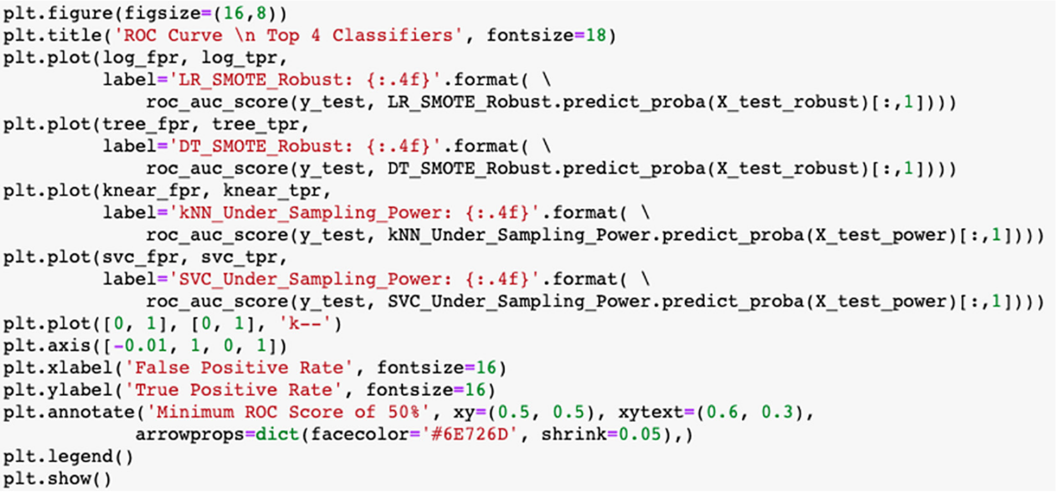
ROC curve is a plot between True Positive Rate and False Positive Rate at various threshold values.



```
from sklearn.metrics import roc_curve

log_fpr, log_tpr, log_thresold = roc_curve(y_test,
                                           LR_SMOTE_Robust.predict_proba(X_test_robust)[:,1])
knear_fpr, knear_tpr, knear_threshold = roc_curve(y_test,
                                                  kNN_Under_Sampling_Power.predict_proba(X_test_power)[:,1])
svc_fpr, svc_tpr, svc_threshold = roc_curve(y_test,
                                            SVC_Under_Sampling_Power.predict_proba(X_test_power)[:,1])
tree_fpr, tree_tpr, tree_threshold = roc_curve(y_test,
                                               DT_SMOTE_Robust.predict_proba(X_test_robust)[:,1])
```

Figure 4.112: *Generating values for plotting ROC curve*

The above code snippet will give true positive rates, false-positive rates, and the threshold for all the models. Now we have a task to plot the data and visualize it.



```
plt.figure(figsize=(16,8))
plt.title('ROC Curve \n Top 4 Classifiers', fontsize=18)
plt.plot(log_fpr, log_tpr,
         label='LR_SMOTE_Robust: {:.4f}'.format( \
             roc_auc_score(y_test, LR_SMOTE_Robust.predict_proba(X_test_robust)[:,1])))
plt.plot(tree_fpr, tree_tpr,
         label='DT_SMOTE_Robust: {:.4f}'.format( \
             roc_auc_score(y_test, DT_SMOTE_Robust.predict_proba(X_test_robust)[:,1])))
plt.plot(knear_fpr, knear_tpr,
         label='kNN_Under_Sampling_Power: {:.4f}'.format( \
             roc_auc_score(y_test, kNN_Under_Sampling_Power.predict_proba(X_test_power)[:,1])))
plt.plot(svc_fpr, svc_tpr,
         label='SVC_Under_Sampling_Power: {:.4f}'.format( \
             roc_auc_score(y_test, SVC_Under_Sampling_Power.predict_proba(X_test_power)[:,1])))
plt.plot([0, 1], [0, 1], 'k--')
plt.axis([-0.01, 1, 0, 1])
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.annotate('Minimum ROC Score of 50%', xy=(0.5, 0.5), xytext=(0.6, 0.3),
            arrowprops=dict(facecolor='#6E726D', shrink=0.05),)
plt.legend()
plt.show()
```

Figure 4.113: *Plotting ROC curve*

In this case, we will only plot the ROC curve, and that is good for analyzing the model, but it is recommended to plot some visualization with a threshold as one axis.

[55] "Plotting and Interpreting an ROC Curve." http://gim.unmc.edu/dxtests/roc2.htm.

Below we will see the ROC curve generated by the above code.

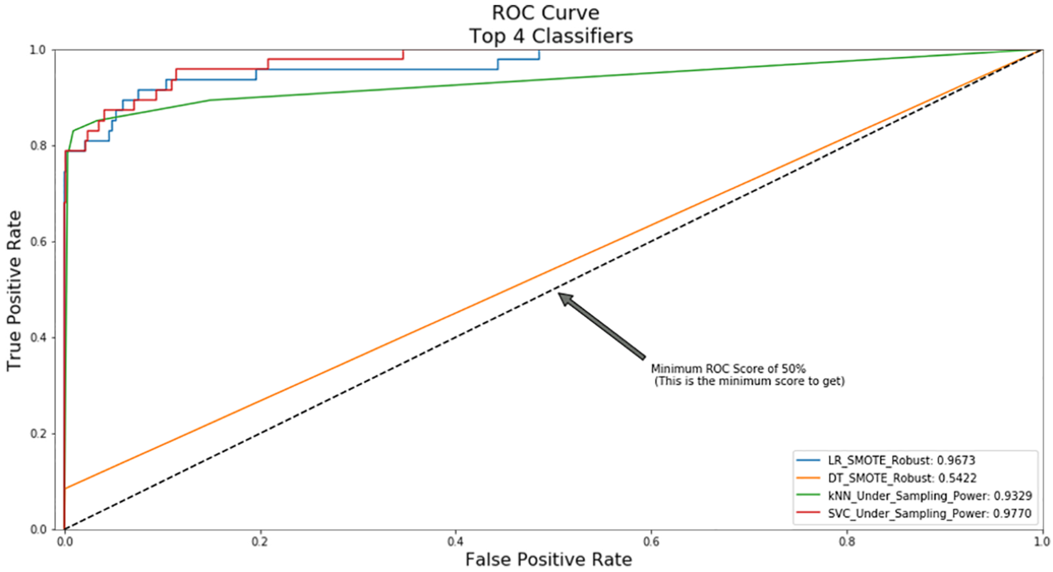


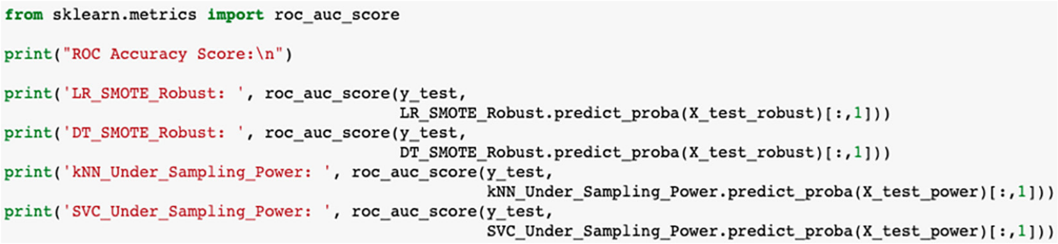
Figure 4.114: *ROC curve*

We should know the higher the value of ROC, the better the model is. We have drawn a reference line with 0.5 as a ROC score. It means that lines/models close to the reference line signifies a random guess like a coin toss where the probability is 0.5 for each class.

While explaining the ROC curve, it was mentioned that for each threshold, what is the value for true positive rate and false-positive rate, but from where are we getting a single value/score ROC?

ROC score is nothing but the area under the curve, which is generated from the ROC curve. ROC score is also referred to as AUC or AUROC.

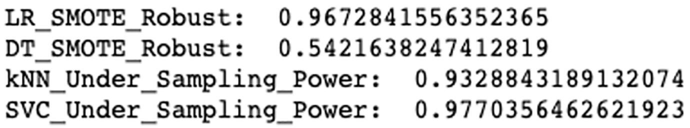
ROC score summarizes the curve information in a single value. We will see what the score for all the models is.



```
from sklearn.metrics import roc_auc_score

print("ROC Accuracy Score:\n")

print('LR_SMOTE_Robust: ', roc_auc_score(y_test,
                                         LR_SMOTE_Robust.predict_proba(X_test_robust)[:,1]))
print('DT_SMOTE_Robust: ', roc_auc_score(y_test,
                                         DT_SMOTE_Robust.predict_proba(X_test_robust)[:,1]))
print('kNN_Under_Sampling_Power: ', roc_auc_score(y_test,
                                                  kNN_Under_Sampling_Power.predict_proba(X_test_power)[:,1]))
print('SVC_Under_Sampling_Power: ', roc_auc_score(y_test,
                                                  SVC_Under_Sampling_Power.predict_proba(X_test_power)[:,1]))
```

Figure 4.115: *Generating AUROC Score*

The above code will give us the scores for all the values, and this will help us to select the best or optimal model, among others.



```
LR_SMOTE_Robust:   0.9672841556352365
DT_SMOTE_Robust:   0.5421638247412819
kNN_Under_Sampling_Power:   0.9328843189132074
SVC_Under_Sampling_Power:   0.9770356462621923
```

Figure 4.116: *ROC Score*

We can see from the score and curve that Decision Tree performed worst hence we can drop that model. The best model as per score is SVC, but we also see a close score to SVC that is of the model Logistic Regression model. Now, both of the models are similar, and deciding the right model will be more of a business decision, which we will discuss in brief later.

We have one more plot to see, i.e., Precision-Recall Curve[56] for all the models as well.

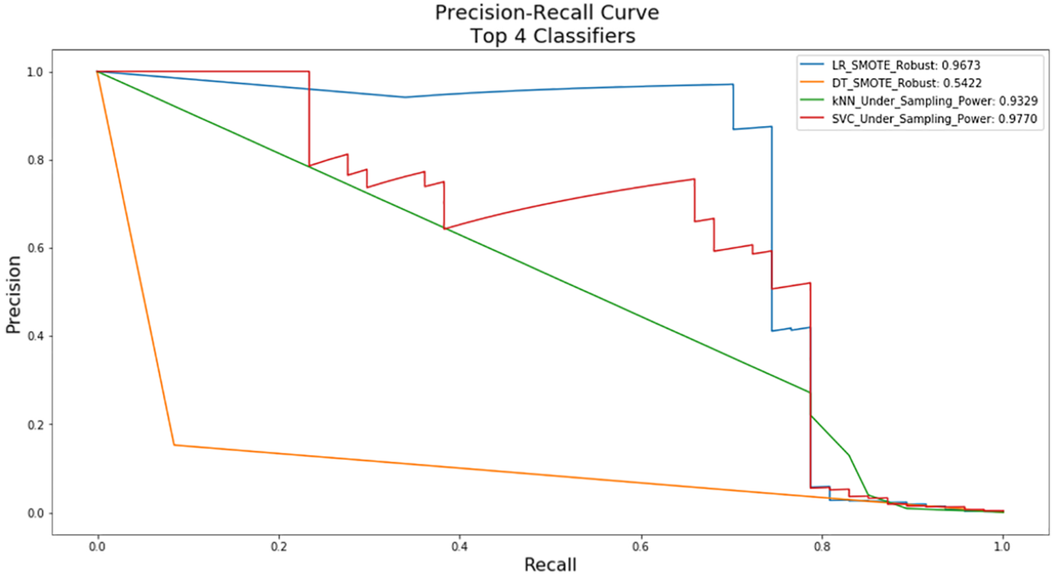


Figure 4.117: *Precision-Recall curve*

Precision-Recall curve is calculated in a similar way like the ROC curve. In this case, Precision and Recall values are calculated for different thresholds, and the plot is made.

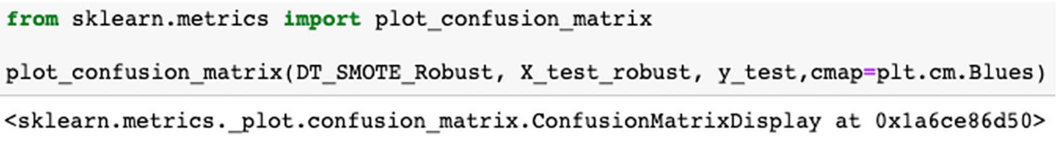
[56] "Precision-Recall — scikit-learn 0.22.1 documentation." http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

Now, the question is: When should we use **ROC vs. Precision-Recall Curves?**[57] [58] [59]

- ROC curves are generally used when there is a roughly equal number of records for each class.
- Precision-Recall curves are used when there is a moderate to high-class imbalance.

The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance, but here we are making the distribution equal using Random Under Sampling and SMOTE. So, ROC will be the right choice of plot.

We will mostly complete the evaluation here, but before that, let us see the confusion matrix for all the models, then it will validate the above model selection. Here we will focus on Class 1, i.e., the Fraudulent transaction class.



```
from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(DT_SMOTE_Robust, X_test_robust, y_test,cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a6ce86d50>
```

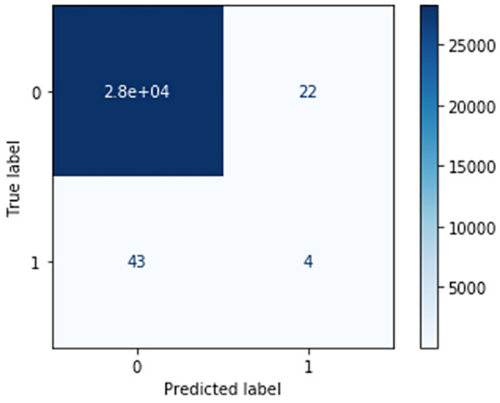


Figure 4.118: *Decision Tree + SMOTE + Robust Scaling*

This is the confusion matrix for the decision tree, and it is the worst confusion matrix among all. During the training, cross-validation, we saw a good performance for the decision tree, and that was a metric trap. Reading the matrix, we can mostly say all values of Class 1 are misclassified, and the model is extremely biased towards Class

[57] "The relationship between Precision-Recall and ROC curves ...." https://dl.acm.org/doi/10.1145/1143844.1143874.

[58] "The Precision-Recall Plot Is More Informative than the ... - NCBI." 4 Mar. 2015, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4349800/.

[59] "ROC Graphs: Notes and Practical ... - School of Systems Biology." 16 Mar. 2004, http://binf.gmu.edu/mmasso/ROC101.pdf.

0. Seeing the performance, we will surely drop this model and not consider it for further re-tuning.

Mostly this is self-explanatory, and we have analyzed the confusion matrix before.

```
plot_confusion_matrix(LR_SMOTE_Robust, X_test_robust, y_test,cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a7085fd90>
```

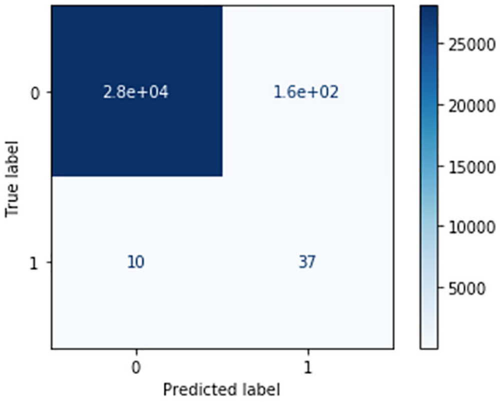


__Figure 4.119:__ Logistic Regression + SMOTE + Robust Scaling

We always need to choose whether we need a low misclassification for Fraudulent Class or low misclassification for Non-Fraudulent class. This is purely a business call which one to select.

k-NN was a similar performing model like Logistic Regression and SVC. We can see the confusion matrix below.

```
plot_confusion_matrix(kNN_Under_Sampling_Power, X_test_power, y_test,cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a6bed7710>
```

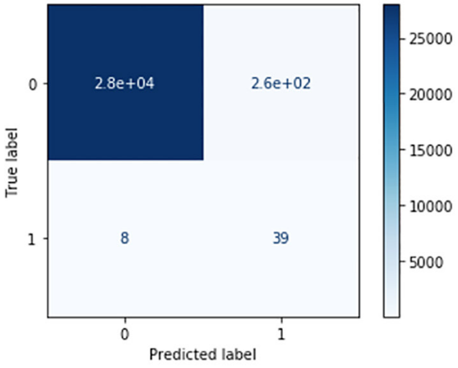
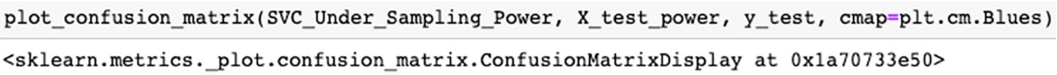


__Figure 4.120:__ k-NN + Random Under Sampling + Power Transformer

For k-NN, the misclassification for Class 0 is a bit high compared to Logistic and SVC. So, for this reason, we will be rejecting this model too.

We will see the last model's confusion matrix, which we select as one of the best models.



```
plot_confusion_matrix(SVC_Under_Sampling_Power, X_test_power, y_test, cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a70733e50>
```

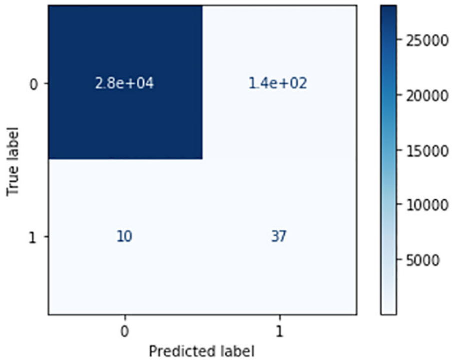
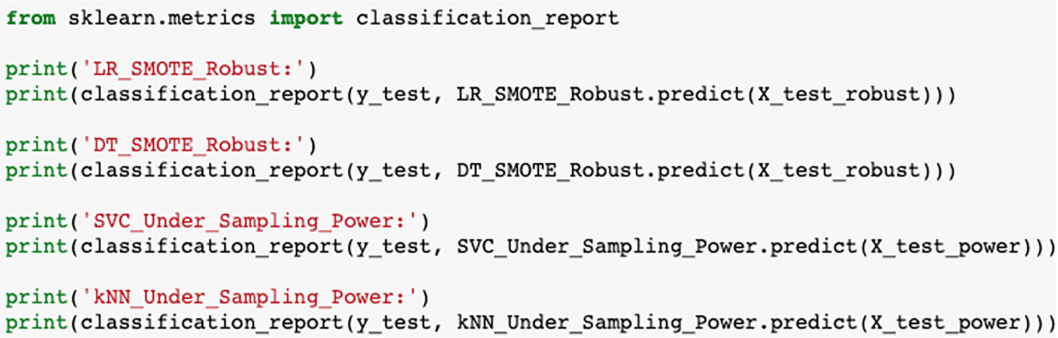


Figure 4.121: *SVC + Random Under Sampling + Power Transformer*

SVC is similar to Logistic Regression, and we see the difference is minimal, and both can be used interchangeably.

Now, we need to choose between SVC and Logistic Regression. We can think in the direction of computation time, business logic, convenience, and explain-ability to select a model. SVC takes a huge time to compute, and it is not advised for extremely large datasets. These are the factors which affect the selection.

There is another way we can get all the important metrics for analysis. We can use the Classification Report[60].



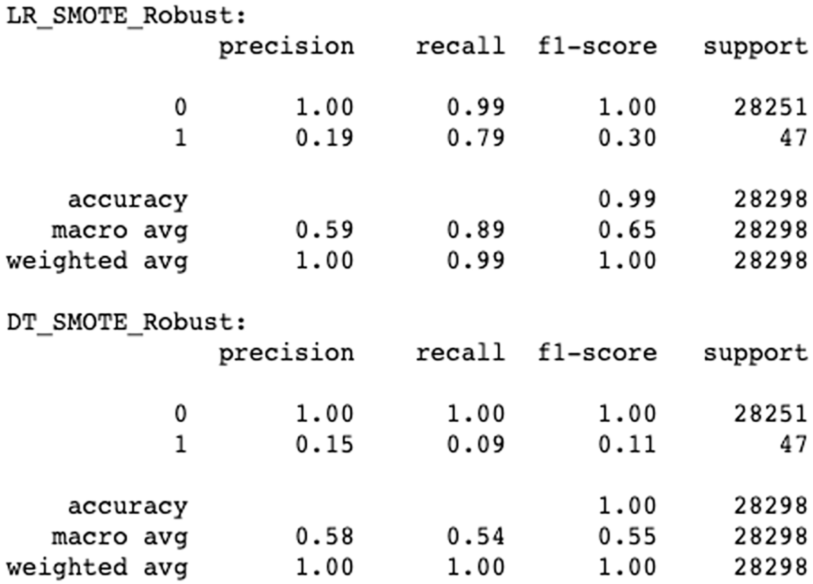
```
from sklearn.metrics import classification_report

print('LR_SMOTE_Robust:')
print(classification_report(y_test, LR_SMOTE_Robust.predict(X_test_robust)))

print('DT_SMOTE_Robust:')
print(classification_report(y_test, DT_SMOTE_Robust.predict(X_test_robust)))

print('SVC_Under_Sampling_Power:')
print(classification_report(y_test, SVC_Under_Sampling_Power.predict(X_test_power)))

print('kNN_Under_Sampling_Power:')
print(classification_report(y_test, kNN_Under_Sampling_Power.predict(X_test_power)))
```

Figure 4.122: *Generating a classification report*

[60] "sklearn.metrics.classification_report — scikit-learn 0.22.1 ...." http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

The above code snippet gives us the classification report for all the models. We will divide the output into two sections.

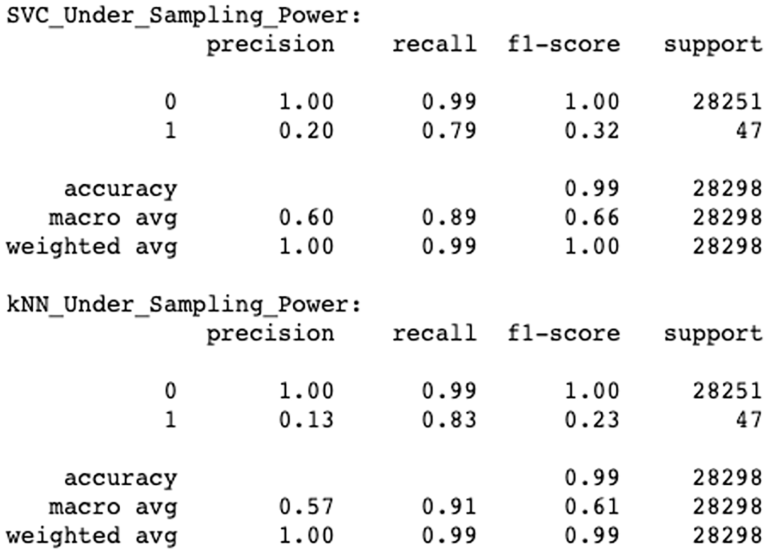
SMOTE dataset classification report.



```
LR_SMOTE_Robust:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     28251
           1       0.19      0.79      0.30        47

    accuracy                           0.99     28298
   macro avg       0.59      0.89      0.65     28298
weighted avg       1.00      0.99      1.00     28298

DT_SMOTE_Robust:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28251
           1       0.15      0.09      0.11        47

    accuracy                           1.00     28298
   macro avg       0.58      0.54      0.55     28298
weighted avg       1.00      1.00      1.00     28298
```

Figure 4.123: *SMOTE classification report*

Now for the Under Sampled Dataset Classification Report, we have the corresponding image below. It is important to note that the code gives us a single output and not in two sections.



```
SVC_Under_Sampling_Power:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     28251
           1       0.20      0.79      0.32        47

    accuracy                           0.99     28298
   macro avg       0.60      0.89      0.66     28298
weighted avg       1.00      0.99      1.00     28298

kNN_Under_Sampling_Power:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     28251
           1       0.13      0.83      0.23        47

    accuracy                           0.99     28298
   macro avg       0.57      0.91      0.61     28298
weighted avg       1.00      0.99      0.99     28298
```

Figure 4.124: *Random under-sampling classification report*

These reports give us the gist for the important metrics, and we use it very often to compare the models fast, if we have many models to deal with for the same data.

With this, we will end this chapter, and it is advised to make different models and test it out. The more you practice; it will enhance your confidence to handle the data modeling.

Further reading

- Sparse Matrix
- Log Transformation
- Over Sampling
  - ADASYN
  - Borderline SMOTE
  - SMOTE for Nominal and Continuous
- Under Sampling
  - Under-sampling with ensemble learning
  - Cluster
  - Instance hardness threshold
  - One-sided selection method
  - Neighborhood cleaning rule
- Decision Tree
  - Gini Index
  - CART
  - C4.5
- SVM
  - The Linear Soft-Margin Classifier
  - Different Kernel Tricks
- Classification Report

Conclusion

In this chapter, we have seen a rough representation of how I work when I get a dataset, and this style will vary from other practitioners. But, surely, everyone will end up doing the same visualizations, but the order will be different.

The chapter was very extensive and detailed, as this is one of the most important fields of research and application where the dataset is highly imbalanced. Financial Services, Litigation, etc. are some areas where handling an imbalanced dataset is common.

The next chapter will cover evaluation techniques and model re-tuning where data which we will be dealing with is highly critical. Business decisions will also play a huge role in the chapter. The next chapter will be short and concise and focus on aspects that we have dealt the least.

CHAPTER 5
Heart Disease UCI

Introduction

In the field of Medicine, Machine Learning is creeping in very fast, and it is still in research. Machine Learning in Medicine is a critical area to work, as there will be a lot of lives involved from the decision which model makes. Here we will discuss these kinds of critical decisions that we need to make by evaluating and re-training the model.

This chapter will be like a business case study and we will mainly focus on model re-tuning and how to align models with the business. This skill is extremely important for analyzing the business problem, ask the right questions, make a machine learning model out of it, and finally tune it to improve the accuracy of the prediction and finally explain the model's output.

This chapter will tackle a multitude of topics, and many of the topics are huge and a chapter of its entirety. So, it is advised to give some time for each topic before jumping to the next.

Structure

- Prerequisites
- Let's understand the data
- Machine Learning modeling
- Explainable AI

Objective

We will change the pattern of this chapter a bit and handle it differently. We will have only a few data analysis steps that are required to move forward with modeling. We will focus on model evaluation, model re-tuning, and aligning all the efforts with the business case. Here we will also have a slight introduction to explain the ability of a particular decision.

Prerequisites

We will introduce some of the concepts which we will use in this chapter. The below concepts will mainly cover technical and theoretical concepts.

Why is ML in medicine so critical?

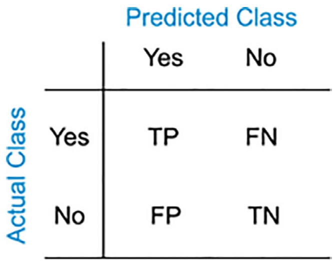
ML in medicine can have different business cases and use cases. ML can be applied for inventory management systems for pre-ordering medicines as per the demand or directly predicting cancer like colon cancer, breast cancer, etc. Roughly we can divide this field into segments. One, ML is directly involved in diagnosing the patients and Two, ML has an indirect involvement with the patients.

Segment one is the field where the decision made by the ML is critical as it directly affects human life. Let's take an example.

We need to predict whether a patient has cancer or not. Now, as per the confusion matrix, we know that there can be four types of output.

1. True Positive (TP)
2. True Negative (TN)
3. False Positive (FP)
4. False Negative (FN)

Let us take a look at the confusion matrix once again, so that will make this simpler and easier to read.



		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

Figure 5.1: *Binary Confusion Matrix (Source: WikiMedia)*

The **actual class** is used for reference, as we have seen in the test data when we split. After splitting the data into the training and testing dataset, we use the training dataset to train the model and use the test dataset to evaluate the efficiency of the model by comparing the model/predicted output with the original/actual output.

TP and TN are easy things to explain. When the "actual class" output is the same as the "predicted class" output, then it is either True Positive (TP) or True Negative (TN) depending on the actual values. If we explain with the example, then TP is when the model predicts the positive cancer patients as a "positive class," i.e., "True Class" using the features. Similarly, the model predicts negative class for the negative cancer patients, which is known as TN.

FN, i.e., False Negative, is when the model incorrectly predicts a cancer negative patient as a positive class, i.e., cancer positive patient.

On the other hand, FP is when the model predicts a cancer positive patient as a negative class, i.e., cancer negative person. And this is DEADLY!!!

Now, I guess we know why it is so critical while working with human lives. In this case, FP can still be accepted as they will be diagnosed again.

We should always know that in real time a 100% accurate model is not possible, so there will be some FN and FP.

Now, as per the business problem and requirements, we have to have a trade-off: either higher FN and lower FP; or, similar count of FP and FN; or, Lower FN and higher FP.

In this example, the trade-off will be Zero FN because there should not be any case where a cancer positive person is put to a negative class.

What is Explainable AI?

Explainable AI is again a hot topic for research. In simple words, whenever a model predicts an output we need to justify why did the model give us a particular class?

Which features are affecting that decision (Similar to the correlation matrix, but there are more to it)?

Generally, ML models are considered as a black box[1] so, it just takes in the features and gives us a class, continuous number, probability, cluster number, etc. Now the main question which is asked is what resulted in that decision.

This reasoning is useful in many fields where we want to answer questions like why and how. Medicine, Financial Services, Insurance are some of the sectors where explainable ML is useful.

Now, we will see some terms that we need to know.

Regularization[2]

Regularization is a practice to avoid over-fitting the model where it penalizes high-valued regression coefficients. Let us explain this concept in more depth.

Over-fitting is a problem when the model learns each and every pattern of the data point with the noise as well. It has a high prediction power for the training dataset, but it underperforms for tests and real-time data. That means the model was incapable of learning the pattern of the data. Instead, it sticks too much to the data. To solve this problem, we use regularization.

Let us take an example of linear regression for the explanation purpose then we will generalize it.

Here we will discuss Lasso and Ridge regression, but there are other types of regularizations as well.

LASSO (Least Absolute Shrinkage and Selection Operator)[3]

Lasso uses L1[4] norm[5] as a penalty with the regular OLS[6]

$$\beta^* = arg\ min_{\beta} \sum_{i=1}^{k} \left(y_i - \beta^T x_i \right)^2 + \lambda \sum_{i=1}^{k} | \beta_j |^1$$

If we see carefully, the regularizer part is

$$\lambda \sum_{i=1}^{k} | \beta_j |^1$$

[1] "Black box - Wikipedia." https://en.wikipedia.org/wiki/Black_box.

[2] "Regularization (mathematics) - Wikipedia." https://en.wikipedia.org/wiki/Regularization_(mathematics).

[3] "Lasso (statistics) - Wikipedia." https://en.wikipedia.org/wiki/Lasso_(statistics).

[4] "L1-Norm - Wolfram MathWorld." http://mathworld.wolfram.com/L1-Norm.html.

[5] "Norm (mathematics) - Wikipedia." https://en.wikipedia.org/wiki/Norm_(mathematics).

[6] "Ordinary least squares - Wikipedia." https://en.wikipedia.org/wiki/Ordinary_least_squares.

Ridge regression[7]

Ridge regression used the L2[8] norm with the OLS.

$$\beta^* = arg\ min_{\beta} \sum_{i=1}^{k} \left(y_i - \beta^T x_i\right)^2 + \lambda \sum_{i=1}^{k} |\beta_j|^2$$

So, the regularization factor is

$$\lambda \sum_{i=1}^{k} |\beta_j|^2$$

Let us take a quick look at some of the illustrations for the same.

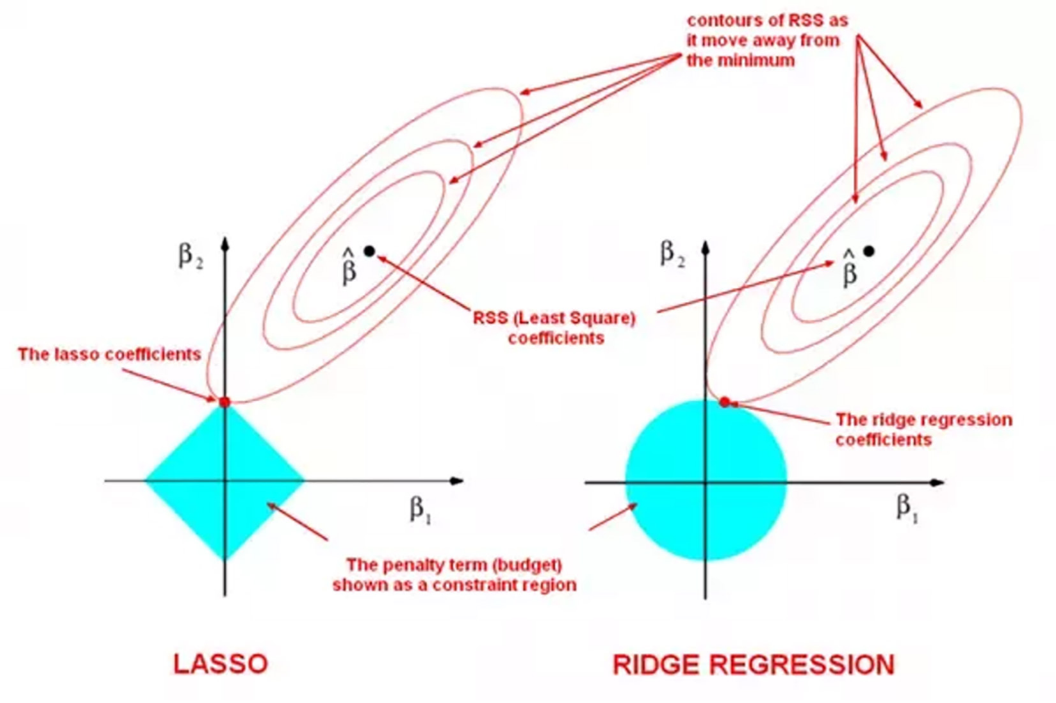

contours of RSS as it move away from the minimum
β_2
$\hat{\beta}$
RSS (Least Square) coefficients
β_2
$\hat{\beta}$
The lasso coefficients
The ridge regression coefficients
β_1
β_1
The penalty term (budget) shown as a constraint region
LASSO
RIDGE REGRESSION

Figure 5.2: *L1 & L2 Regularization (Source: Quora)*

The above image shows the difference between the penalty term between Lasso and Ridge regression.

Let us generalize the regularizer first and see its components:

$$R(\beta) = \sum_{i=1}^{k} |\beta_j|^a$$

[7] "Ridge Regression." https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf

[8] "L2-Norm - Wolfram MathWorld." http://mathworld.wolfram.com/L2-Norm.html.

When $\alpha = 1$ then it is L1 regularization $\alpha = 2$, and when then it is L2 regularization.

Let us now see a general form of the equation:

$$\textit{Cost Function} = \textit{Loss} + \lambda \, \mathrm{R}(\beta)$$

λ is the tuning parameter that decides how much we want to penalize the flexibility of our model? When $\lambda = 0$, it is just the Loss function, and when λ is higher, it will add too much weight, and it will lead to under-fitting.

Among L1 and L2 regularizers, L2 is differentiable for every value of β, and that's why L2 regularizer is the popular regularization technique.

Hypothesis testing[9]

Previously, we have talked a lot about hypothesis and some flow diagram of how to test the hypothesis. Here we will take a look at a more formal approach to Hypothesis testing using statistical methods.

In ML we use some hypothesis quite often like,

1. A test that assumes the test data as a normal distribution
2. Similarly, a test that assumes the test distribution similar to the train distribution
3. A test that assumes that two samples were used from the same or similar underlying population/sample distribution.

So, the assumption of statistical tests is known as the **Null Hypothesis** or **Hypothesis Zero** or simply H0. **H0** is the default assumption or the assumption that nothing has changed.

There is another type of hypothesis that is **Alternative Hypothesis** or **First Hypothesis** or **Hypothesis One** or in short **H1**. We can guess that it is the opposite of the Null Hypothesis, and yes, it is. The violation of the test's assumption is known as H1, or we can say some other hypothesis that means the evidence suggests that H0 can be rejected.

So, we can say in short that:

- H0 is an assumption that the test holds and is failed to be rejected at some level of significance.
- H1 is an assumption that the test does not hold and is rejected at some level of significance.

[9] "Statistical hypothesis testing - Wikipedia."
https://en.wikipedia.org/wiki/Statistical_hypothesis_testing.

Let us take an example of Pearson Correlation and explain:

Null Hypothesis (H0): There is no relationship between the two variables

Alternative Hypothesis (H1): There is a relationship between the two variables

We can represent:

$$H_0 : \rho_{xy} = 0$$

$$H_1 : \rho_{xy} \neq 0$$

Where 0 (zero) means there is no relationship, i.e., there is no correlation.

There are some properties if we design a hypothesis.

The hypothesis is mutually exclusive and exhaustive.

The hypothesis is exclusive because there shouldn't be any case where there is an overlap between the results. A hypothesis is exhaustive because it must cover all possible output.

We can see all the cases are covered with the above example of Pearson Correlation.

This is also known as a two-tailed test because there are two possibilities, either the test will be positive, or it will be negative.

Now, let's take a look when the statistical testing returns a value to be more specific a p-value[10]. The p-value is used to quantify and interpret the Null Hypothesis and Alternative Hypothesis using a threshold value, which is chosen according to the business case beforehand, is called the significance level[11].

Here the significance level is represented with alpha (α).

So, we can rewrite H0 and H1 for the p-value and statistical significance level.

- If p-value > α: Fail to reject the null hypothesis (i.e., not significant result).
- If p-value ≤ α: Reject the null hypothesis (i.e., significant result).

If we see carefully, this hypothesis is mutually exclusive and exhaustive. We will leave this topic here, and it is recommended to read on this topic a bit more from the mentioned footnotes.

[10] "p-value - Wikipedia." https://en.wikipedia.org/wiki/P-value.

[11] "Statistical significance - Wikipedia." https://en.wikipedia.org/wiki/Statistical_significance.

Ensemble learning[12]

The **ensemble method** uses multiple Machine Learning algorithms to achieve better predictive performance. This method helps to obtain better performance than any of the Machine Learning algorithms alone. In most cases, a tree-based algorithm is used as a decision tree.

Let's see some of the techniques which are basic yet popular and used in the field of Machine Learning.

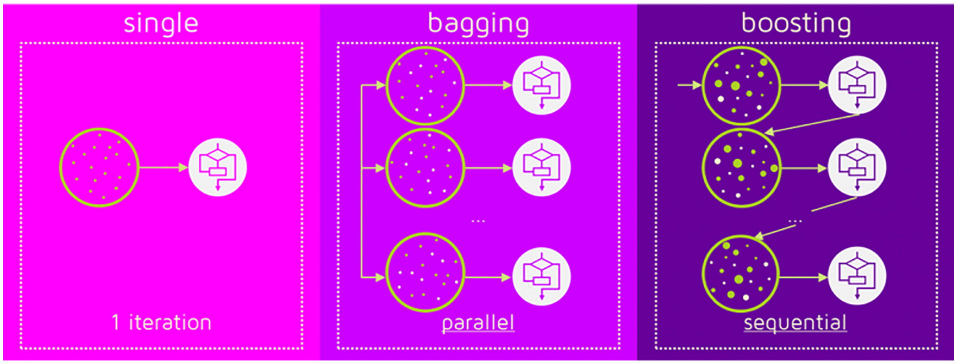


Figure 5.3: Ensemble Learning Techniques (Source: Quantdare)

There are some of the basic techniques we will see in the next section, where we will explain the parallel and the sequential concept.

Bagging

Bagging is also known as **Bootstrap Aggregating**, involves having each model in the ensemble vote with an equal weight where each model takes in random data from the dataset. This method helps to reduce the variance.

If we simply say it's an average between all the trained models, which is trained with random records.

$$D(x) = \sum ad_i(x)$$

Where $D(x)$ is a strong classifier, α is the constant weight for all weak classifier, $d_i\,(x)$ is a weak classifier, that is computed parallelly.

Therefore, we can say, aggregating equally the results of the weak classifiers built independently on random samples to create a strong learner.

[12] "Ensemble learning - Wikipedia." https://en.wikipedia.org/wiki/Ensemble_learning.

Boosting

Boosting multiple ML models are trained sequentially where the training of the model at a given step will have a dependency from the previously trained models using the entire dataset. It mainly focuses on reducing the bias and trying to fit all kinds of observation, especially the difficult ones, by altering the weights for each level and that weight is used to train the next model, i.e., the next model learns from the previous mistakes. When an input is misclassified by a hypothesis, its weight is increased so that the next hypothesis is more likely to classify it correctly, and this process goes on serially.

If we simply say it's a weighted average (sequentially) for all trained models. One thing to remember is Bagging can be achieved in parallel, but Boosting cannot be computed in parallel. So, in terms of time computation, it takes a long time to execute a boosting algorithm compared to a Bagging algorithm.

$$D(x) = \sum a_j d_j(x)$$

Where, $D(x)$ is a strong classifier, a_j is the different weight for each weak classifier (which depends on the previous classifier), $d_j(x)$ is a weak classifier that is computed serially/sequentially.

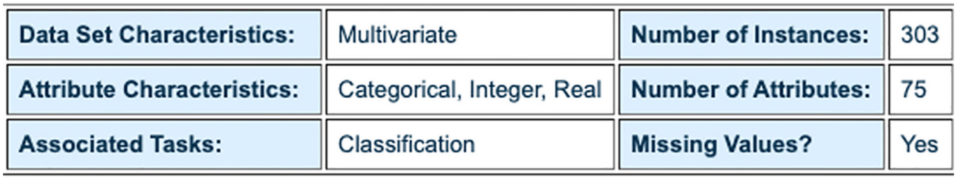
Therefore, we can say, combining differently the results of weak learners built sequentially on the whole dataset to create a strong learner.

There are many other types of ensemble learning techniques, but the above ones are the most popular and basic ones to start with.

Let's understand the data

The data we will be using in the chapter is a medical dataset for heart diseases from the UCI Repository[13].

Some information about the dataset is provided on the website mentioned below.



Data Set Characteristics:	Multivariate	**Number of Instances:**	303
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	75
Associated Tasks:	Classification	**Missing Values?**	Yes

Figure 5.4: *Data Information from UCI (Source: UCI)*

[13] "Heart Disease Data Set - UCI Machine Learning Repository." 1 Jul. 1988, https://archive.ics.uci.edu/ml/datasets/Heart+Disease.

The above information is highly useful as this data is provided by the authors of the dataset. We will get a rough idea of how to handle the data and what work needs to be done on this dataset.

We can see that this dataset is meant for classification, and it has some missing values as well.

Let us see the list of columns for this dataset:

1. Age (age in years)
2. Sex (Gender)
    a. 1 = male
    b. 0 = female
3. CP[14] (chest pain type)
    a. 0 = typical angina
    b. 1 = atypical angina
    c. 2 = non-anginal pain
    d. 3 = asymptomatic
4. Trestbps (resting blood pressure (in mm Hg on admission to the hospital))
5. Chol (serum cholesterol[15] in mg/dl)
6. Fbs (fasting blood sugar > 120 mg/dl)
    a. 1 = true
    b. 0 = false
7. Restecg (resting electrocardiographic[16] results)
    a. 0 = normal
    b. 1 = Having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    c. 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria
8. Thalach (maximum heart rate achieved)

[14] "Angina (Chest Pain) | American Heart Association." https://www.heart.org/en/health-topics/heart-attack/angina-chest-pain.

[15] "Understanding Your Cholesterol Report - WebMD." 5 Sep. 2018, https://www.webmd.com/cholesterol-management/understanding-your-cholesterol-report.

[16] "Resting electrocardiography - Preoperative Tests (Update ...." https://www.ncbi.nlm.nih.gov/books/NBK367910/.

9. Exang (exercise-induced angina)[17]
    a. 1 = yes
    b. 0 = no
10. Oldpeak (ST depression induced by exercise relative to rest)
11. Slope (the slope of the peak exercise ST segment)
    a. 0 = upsloping
    b. 1 = flat
    c. 2 = downsloping
12. Thal (A blood disorder called thalassemia)[18]
    a. 3 = normal
    b. 6 = fixed defect
    c. 7 = reversible defect
13. Ca (number of major vessels (0-3) colored by fluoroscopy[19])
14. Target
    a. 1
    b. 0

We will use the above list as a reference throughout the chapter as, we won't be renaming the columns.

Data Analysis

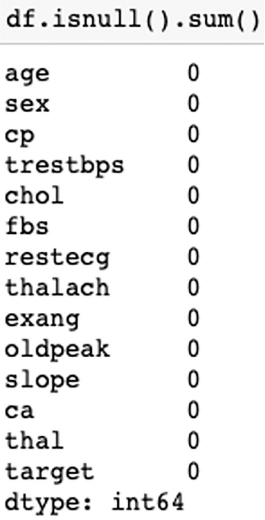
In this chapter, I won't focus on code explanation as that will be the same as the other chapters, and it will be redundant too.

Let us start by validating the missing value and see how many missing values are per column.

[17] "Exercise-induced angina in the cold. - NCBI." https://www.ncbi.nlm.nih.gov/pubmed/4068968.

[18] "Thalassemia - StatPearls - NCBI Bookshelf." 15 Oct. 2019, https://www.ncbi.nlm.nih.gov/books/NBK545151/.

[19] "A noninvasive method for coronary artery diseases diagnosis ...." https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4468223/.

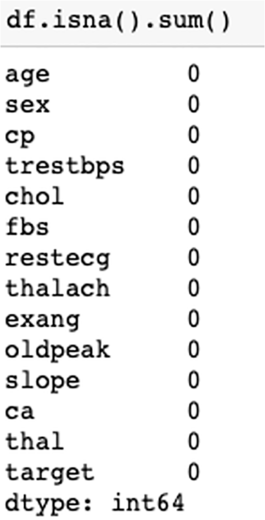


```
df.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Figure 5.5: *Null counts*

Interestingly there are no NULL values in any column. But before concluding, let's take a look at NA as well.



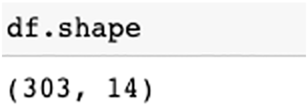
```
df.isna().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Figure 5.6: *NA counts*

We can see that there are no NA values as well in the dataset. Then we have to assume that all the missing values are filled with 0 or some relevant values.

Now, let us confirm the shape of the data.



```
df.shape
```

```
(303, 14)
```

Figure 5.7: *Shape of the data*

The shape of the data is the same as the information is from the UCI Information table. That's a sigh of relief.

We will directly jump into some visualization we have already seen before. The code for the respective sections won't be present as these are overlapping concepts.

Correlation

Let us perform some analysis like correlation, distribution, and statistics about some of the selected data points. Let's start with Pearson's correlation.

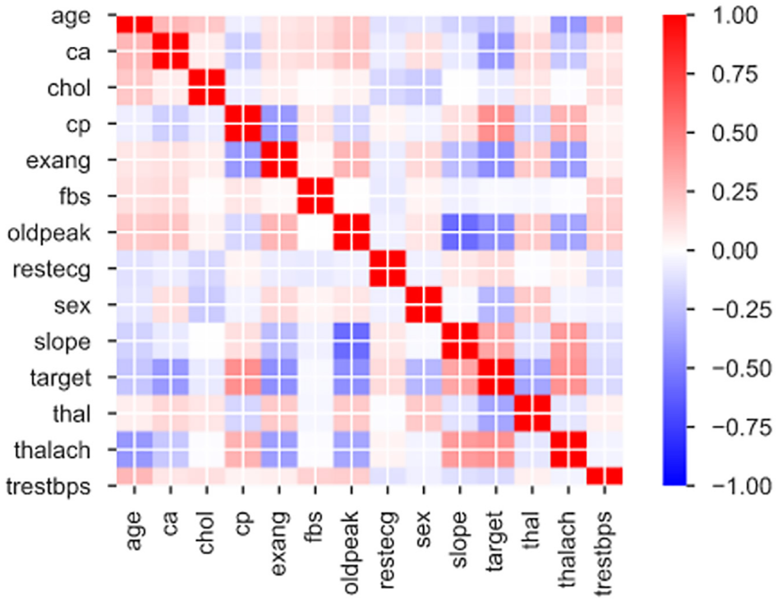


Figure 5.8: *Pearson's r*

Previously, we have seen how Pearson's correlation works, so I am not going to explain this in this chapter. But there is another method related to Pearson's coefficient; rather, it uses Pearson's Coefficient, i.e., Spearman's rank correlation coefficient[20] [21].

[20] "Spearman's rank correlation coefficient - Wikipedia." https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient.

[21] "Spearman's Rank Correlation Coefficient." https://geographyfieldwork.com/SpearmansRank.htm.

Spearman's rank correlation coefficient is a non-parametric[22] measure of rank correlation[23] that is of any ordinal association. Those who are interested in the working and the mathematics behind it are advised to read the links provided at footnotes.

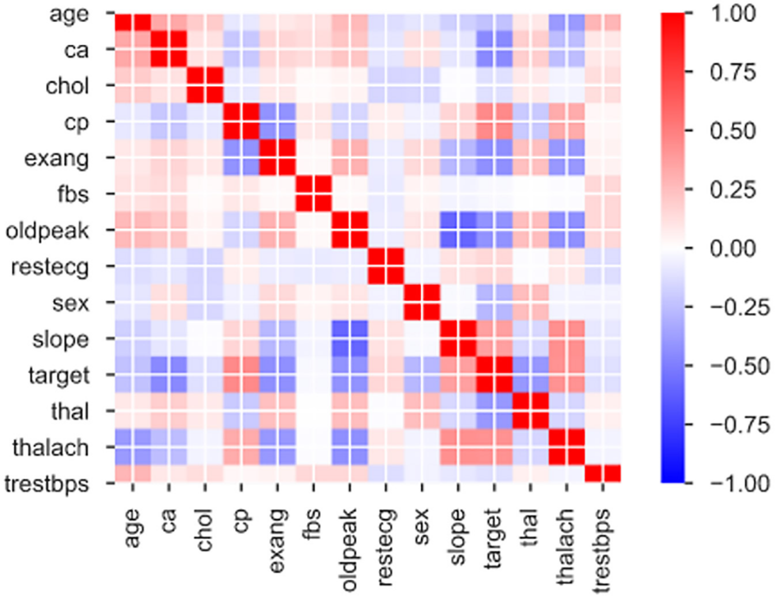


Figure 5.9: *Spearman's*

Both Pearson and Spearman are type correlation. From here, we will only choose the features which are positively correlated and negatively correlated for this data analysis. But finally, we will use the entire dataset and all the features for the training purpose.

From the above correlations, we can see, positively correlated features like "`ca`," "`oldpeak`," and negatively correlated features like "`thalach`", "`cp`."

There is one more observation and point to note down, i.e., when two features are positively correlated, then it doesn't make sense to include both of the features in the training phase as it will give similar information. If we see logically, "`oldpeak`" and "`slope`" should we linearly related[24] , and the correlation shows the same result.

Here, we won't be discussing any topic regarding medicine as I am not the right person to explain those concepts. We will be mostly using common sense, logic, and statistical explanations for EDA, Modelling, etc…

[22] "Nonparametric statistics - Wikipedia." https://en.wikipedia.org/wiki/Nonparametric_statistics.

[23] "Rank correlation - Wikipedia." https://en.wikipedia.org/wiki/Rank_correlation.

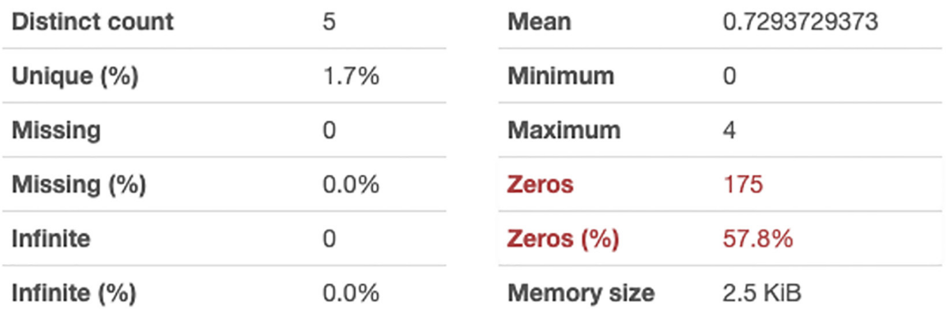
[24] "Linear Relationship Definition - Investopedia." 13 Nov. 2019, https://www.investopedia.com/terms/l/linearrelationship.asp.

This is a very common situation where the statistical results will differ from the results provided by domain experts. It's purely a business call what they want to do.

For some of the features, I have given a footnote, but if someone wants to know more about the features, searching online is the best option.

"ca"

We will perform some feature analysis on "ca," i.e., the number of major vessels colored by fluoroscopy. First, let us take a look at some metrics like missing values, uniqueness, etc.

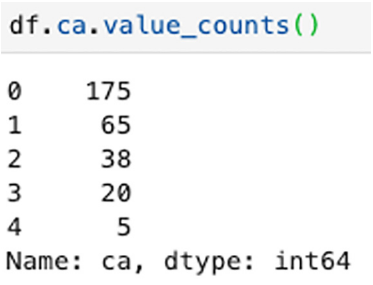


Distinct count	5	**Mean**	0.7293729373
Unique (%)	1.7%	**Minimum**	0
Missing	0	**Maximum**	4
Missing (%)	0.0%	**Zeros**	175
Infinite	0	**Zeros (%)**	57.8%
Infinite (%)	0.0%	**Memory size**	2.5 KiB

Figure 5.10: *Feature Details for "ca."*

We can see that more than 50% of the data is zero. Now we need to find whether zero is for missing values or it has some meaning.

The picture will be clearer when we see the frequency distribution or histogram, which will verify the distribution.



```
df.ca.value_counts()

0    175
1     65
2     38
3     20
4      5
Name: ca, dtype: int64
```

Figure 5.11: *"ca" frequency table*

4 has the least count in the entire dataset, and zero has the highest count. With this frequency table, we can build a histogram and visually see the proportions.

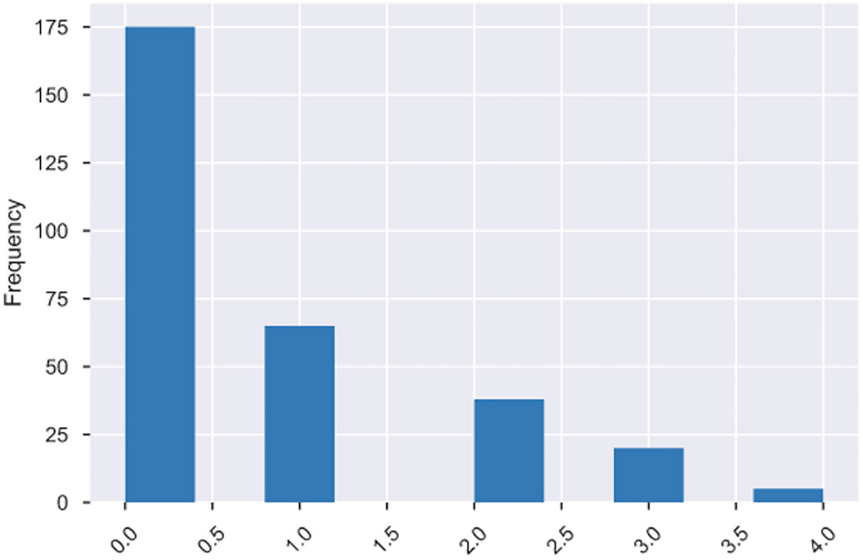
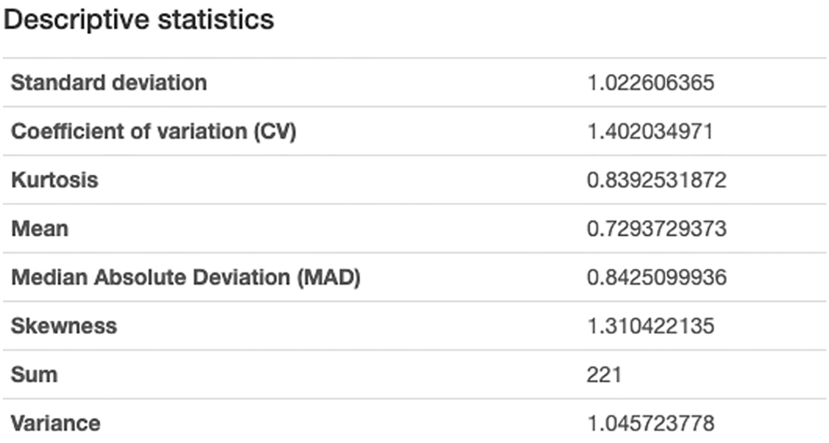


***Figure 5.12:** "ca" feature distribution (bin=10)*

From the distribution, we can see that zero is too high, which may lead to biases. We can deal with that later, and mostly it won't affect us as this is a feature and not the target variable.

For each variable, there will be some variable statistics. Here we will divide the statistics into Descriptive Statistics and Quantile Statistics.

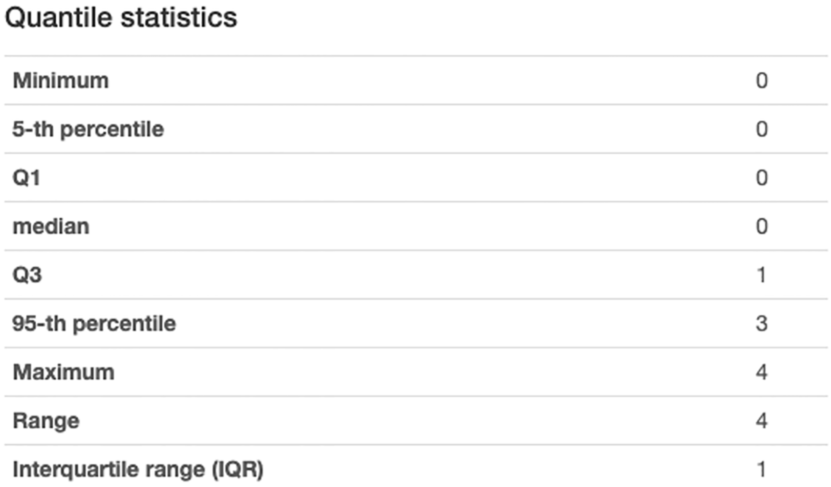


Descriptive statistics

Standard deviation	1.022606365
Coefficient of variation (CV)	1.402034971
Kurtosis	0.8392531872
Mean	0.7293729373
Median Absolute Deviation (MAD)	0.8425099936
Skewness	1.310422135
Sum	221
Variance	1.045723778

***Figure 5.13:** Descriptive statistics for "ca."*

These are the general statistical measures we need to know for all the variables depending upon whether it's a continuous or categorical value. Seeing the values, we can say it is positively skewed. This was evident when we saw the count of zero is more than 50%. Here, we won't be plotting Box plots, but in turn, we will see the metrics regarding those.



Quantile statistics

Minimum	0
5-th percentile	0
Q1	0
median	0
Q3	1
95-th percentile	3
Maximum	4
Range	4
Interquartile range (IQR)	1

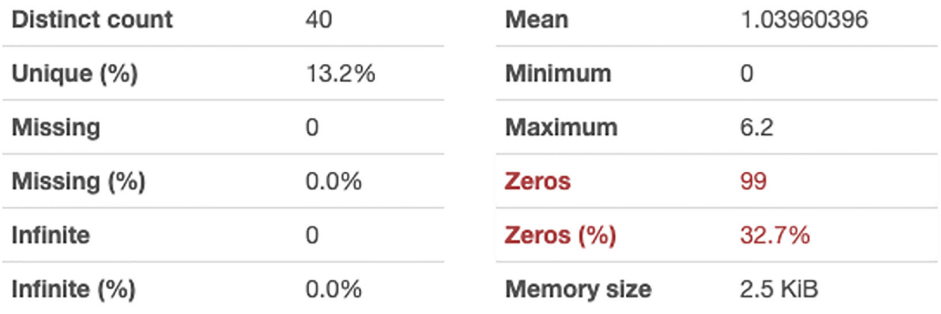
Figure 5.14: *Quantile statistics for "ca."*

We need to keep in mind that it is a categorical variable and not a continuous one.

For categorical variables, analyzing Quantile Statistics doesn't make a whole lot of a sense so we will ignore that.

"oldpeak"

We will perform a similar analysis on "`oldpeak`" as we have done before for "ca." First, let us see some feature details that will help to know about the feature distribution.



Distinct count	40	Mean	1.03960396
Unique (%)	13.2%	Minimum	0
Missing	0	Maximum	6.2
Missing (%)	0.0%	Zeros	99
Infinite	0	Zeros (%)	32.7%
Infinite (%)	0.0%	Memory size	2.5 KiB

Figure 5.15: *Feature Details for "oldpeak"*

Clearly, it is a continuous variable, and we can see there is no missing value, and count of zero is also considerable. It is good to know some of the statistics of a variable that will help to make some decisions later if required.

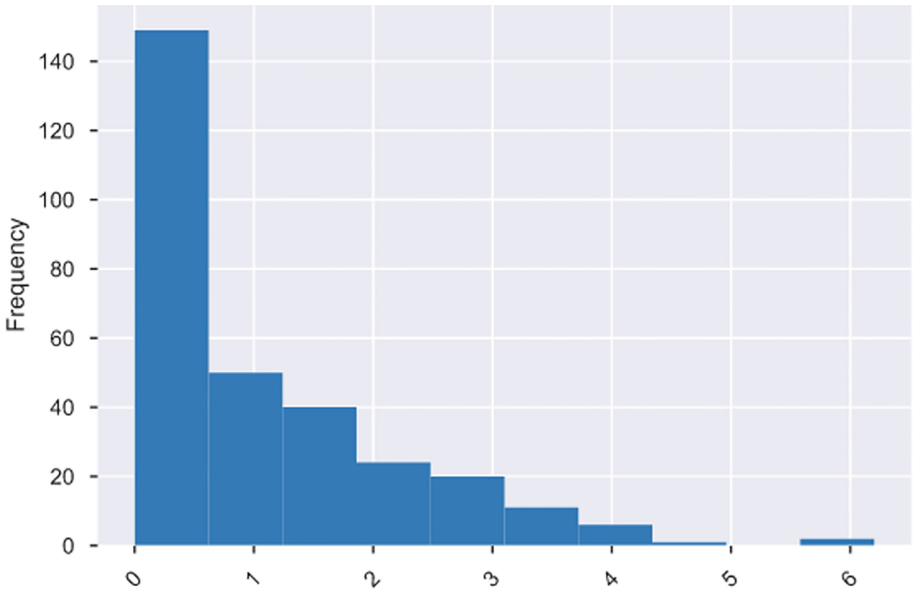
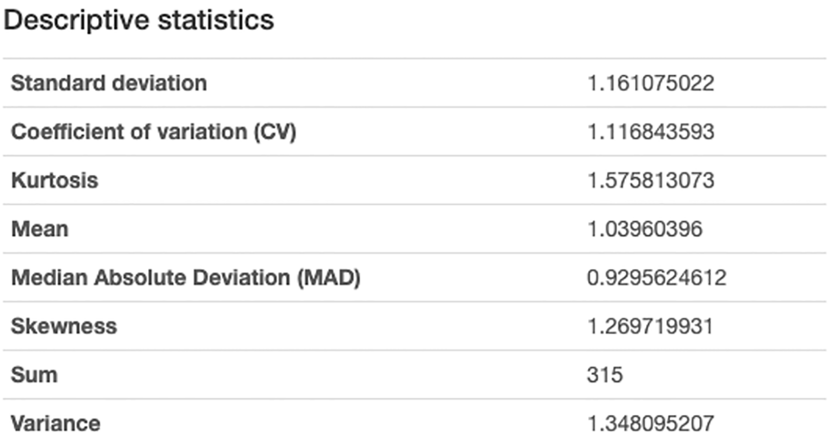


Figure 5.16: *Histogram for "oldpeak"*

This is also slightly positively skewed data, and this won't affect much in terms of prediction but, the insight of feature will help to explain the ability of the model. We will also take a look at the descriptive statistics.



Descriptive statistics

Standard deviation	1.161075022
Coefficient of variation (CV)	1.116843593
Kurtosis	1.575813073
Mean	1.03960396
Median Absolute Deviation (MAD)	0.9295624612
Skewness	1.269719931
Sum	315
Variance	1.348095207

Figure 5.17: *Descriptive Statistics for "oldpeak"*

From the above image, we can see a somewhat new term, i.e., coefficient of variation[25] (CV):

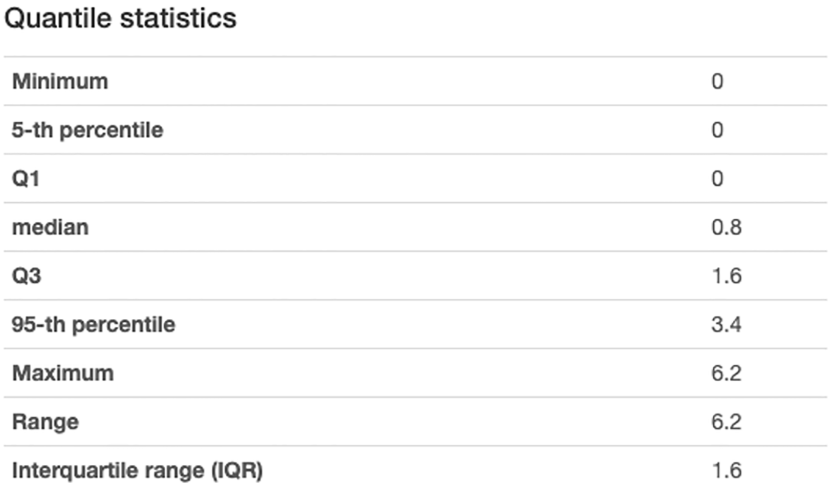
$$CV = \frac{\sigma}{\mu}$$

[25] "Coefficient of variation - Wikipedia." https://en.wikipedia.org/wiki/Coefficient_of_variation.

Where σ is the standard deviation, and μ is the statistical mean of the data.

Coefficient of Variation measures the dispersion of data about the mean of the entire population or sample.

Like before, we won't be plotting any box-and-whiskers plot instead, we will see the Quantile statistics.



Quantile statistics

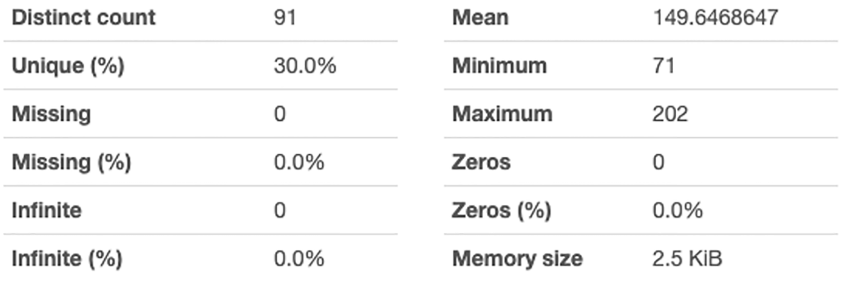
Minimum	0
5-th percentile	0
Q1	0
median	0.8
Q3	1.6
95-th percentile	3.4
Maximum	6.2
Range	6.2
Interquartile range (IQR)	1.6

Figure 5.18: *Quantile statistics for "oldpeak"*

There are some outliers in the data, but the count is not that high. We can ignore the outliers as of now as the size of the dataset is small, so removing any data will decrease the size of the dataset. Later seeing the results, we can decide whether to remove the data or not.

"thalach"

"`thalach`" is the maximum heart rate achieved. Maybe it is a result of an exercise or some form of medically supervised workout. From the description of the feature, we can guess it will be continuous data, but there will be less to no fractional numbers. First, let us see some of the feature details.



Distinct count	91	Mean	149.6468647
Unique (%)	30.0%	Minimum	71
Missing	0	Maximum	202
Missing (%)	0.0%	Zeros	0
Infinite	0	Zeros (%)	0.0%
Infinite (%)	0.0%	Memory size	2.5 KiB

Figure 5.19: *Feature details for "thalach"*

We can see the distinct count as 91, and that is quite high to be a categorical variable so we can consider this as a continuous value.

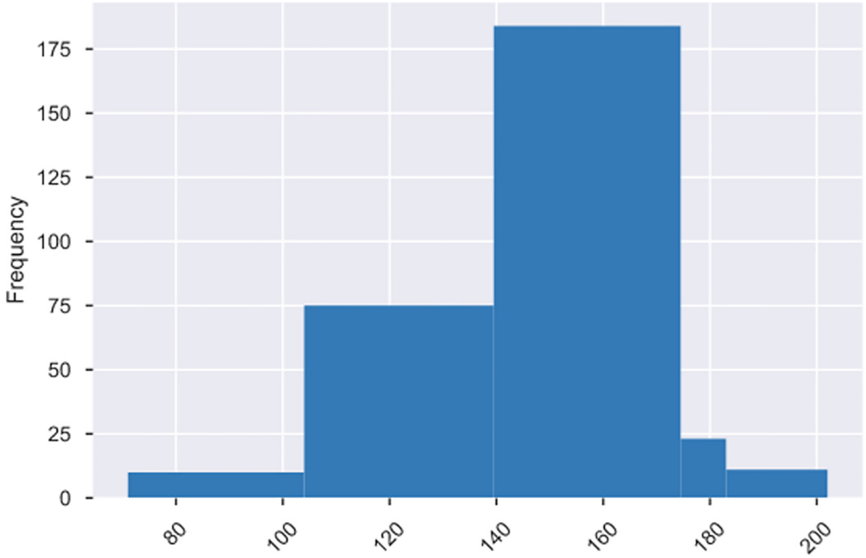
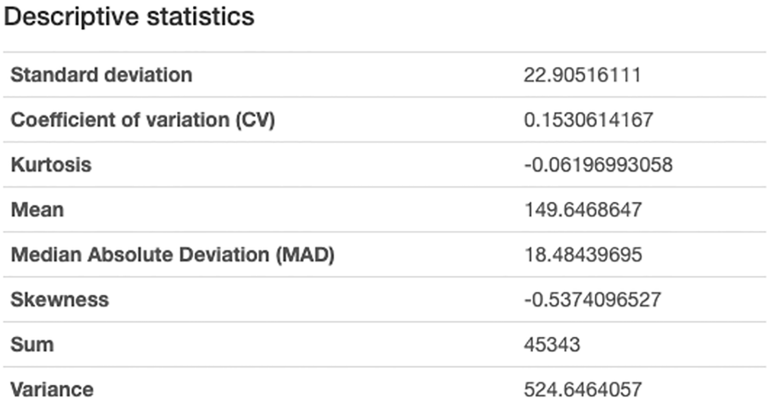


Figure 5.20: *"thalach" histogram (bins=[71. 104. 139.5 174.5 183. 202.]) using "Bayesian Blocks"[26] binning strategy*

We can see that it is negatively skewed data, and it uses Bayesian Blocks to find the bin size. This technique is extremely useful because sometimes incorrect binning can lead to a wrong analysis of data, and finding the right bin gives us the true distribution among the variables.

We are expecting the descriptive statistics to say the same information similar to the above diagram.



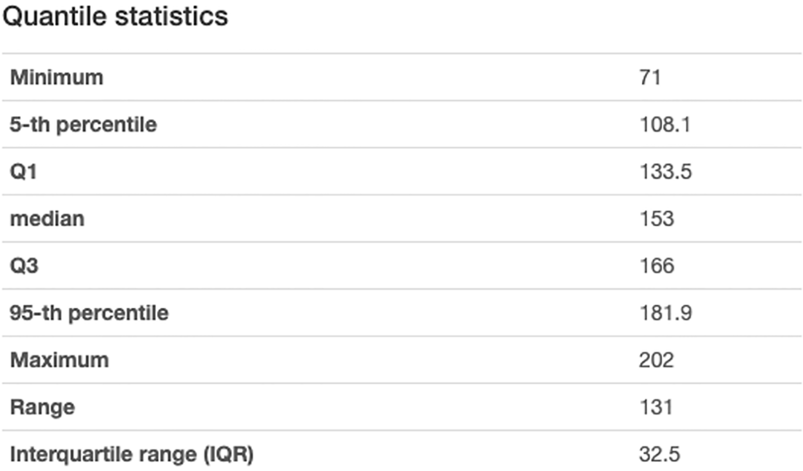
Descriptive statistics

Standard deviation	22.90516111
Coefficient of variation (CV)	0.1530614167
Kurtosis	-0.06196993058
Mean	149.6468647
Median Absolute Deviation (MAD)	18.48439695
Skewness	-0.5374096527
Sum	45343
Variance	524.6464057

Figure 5.21: *Descriptive Statistics for "thalach"*

[26] "Studies in Astronomical Time Series Analysis. VI. Bayesian ...." https://ui.adsabs.harvard.edu/abs/2013ApJ...764..167S/abstract.

Here we can see the skewness is negative, as we had seen in the histogram representing the same. Other details are mostly required for insight and, later, for explainability.



Quantile statistics

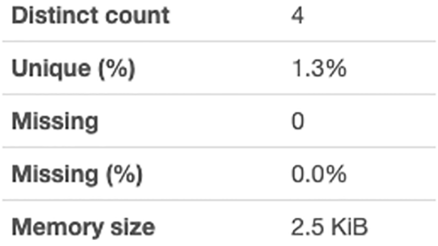
Minimum	71
5-th percentile	108.1
Q1	133.5
median	153
Q3	166
95-th percentile	181.9
Maximum	202
Range	131
Interquartile range (IQR)	32.5

Figure 5.22: *Quantile Statistics for "thalach"*

In this feature, there are some outliers too, but again, we will ignore those outliers. It will be recommended to calculate the intermediate values of the Box plot and compare it with the Quantile statistics.

"cp"

CP stands for Chest Pain, and it has four categories which are listed in the data dictionary above. This feature is categorical. So here we will only see counts and percentages of distribution.



Distinct count	4
Unique (%)	1.3%
Missing	0
Missing (%)	0.0%
Memory size	2.5 KiB

Figure 5.23: *Feature details for "cp."*

From the above statistics, we can see it only takes up 2.5 KiB of storage in memory. This is extremely useful statistics as when we try to perform any operation, then we can guess the space complexity[27] [28] of the operation from the data side.

[27] "EECS 311: Space Complexity." https://courses.cs.northwestern.edu/311/html/space-complexity.html.

[28] "Space complexity - Wikipedia." https://en.wikipedia.org/wiki/Space_complexity.

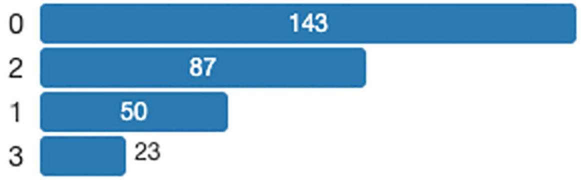
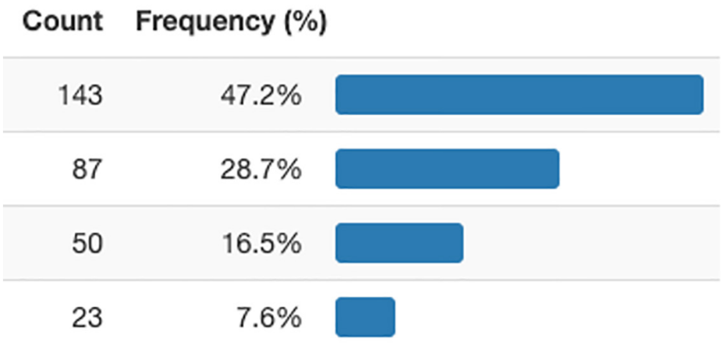


Figure 5.24: *Categorical Feature distribution for "cp"*

This gives us the count of the unique categorical variables. We can now see the percentage of distribution.



Count	Frequency (%)	
143	47.2%	
87	28.7%	
50	16.5%	
23	7.6%	

Figure 5.25: *Percentages for distribution*

The above image gives the frequency and percentage for the different categories.

With this, we will conclude the data analysis for this chapter. Here we have used Pandas-Profiling[29] [30] to generate all the data analysis. This is never my go-to tool, but this is good for a quick overview and analysis of the data.

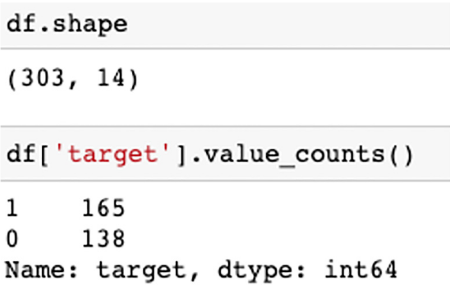
Next section, we will take a look at splitting the dataset into training and testing as we had done before.

Splitting dataset

Before splitting the dataset, let us see the count of individual target values and see whether it is an imbalanced dataset or not.

[29] "Pandas Profiling - GitHub Pages." 7 Jan. 2020, https://pandas-profiling.github.io/pandas-profiling/docs/.

[30] "pandas-profiling/pandas-profiling: Create HTML ... - GitHub." https://github.com/pandas-profiling/pandas-profiling.



```
df.shape
```

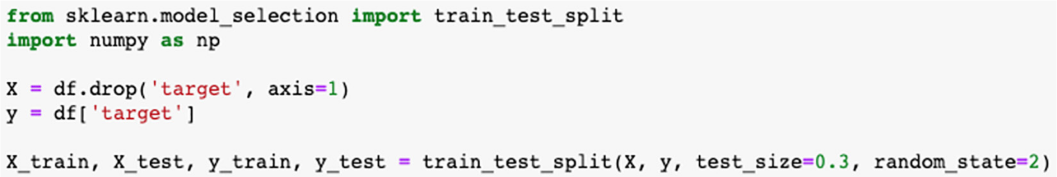
```
(303, 14)
```

```
df['target'].value_counts()
```

```
1    165
0    138
Name: target, dtype: int64
```

Figure 5.26: *Target Frequency Count*

Seeing the count and the distribution, we can say it is more or less equidistributed. Observing this kind of distribution, we can only use "`train_test_split`" and not the Stratified K-fold method.

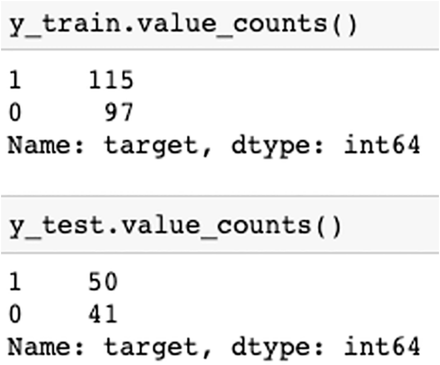


```
from sklearn.model_selection import train_test_split
import numpy as np

X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
```

Figure 5.27: *Train test split*

Like before, we will be using train data only for training and validation purposes, and the model should not see the testing dataset.

Testing the dataset is solely used for testing the model, evaluating it, and improving it using the training data.



```
y_train.value_counts()
```

```
1    115
0     97
Name: target, dtype: int64
```

```
y_test.value_counts()
```

```
1    50
0    41
Name: target, dtype: int64
```

Figure 5.28: *Frequency count for train and test dataset*

We can see the size of the dataset is quite small, and we have to manage the training phase with that only.

Next section onwards, we will start with the model training and evaluation, and also get familiarized with some new machine learning models.

Machine Learning modeling

In this chapter, we will introduce some new algorithms like Ensemble Learning-based algorithms and Probabilistic models. During the training phase, we will use this algorithm to showcase its efficiency.

Machine Learning algorithms

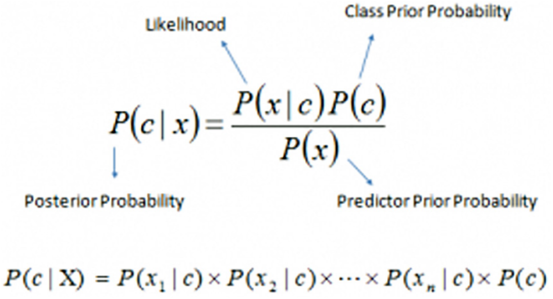
In this section, we will see multiple algorithms, but, here, we will explain some basic algorithms to make the foundation. To know more about these algorithms, it's recommended to go through the footnotes and gain a piece of extensive knowledge.

Naive Bayes Classifier[31] [32]

Naive Bayes is a classification technique in machine learning which uses Bayes' Theorem with the assumption of independence among features. We can simply say that a particular feature in a class is unrelated to the presence/existence of any other feature.

Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability of the output class, and that is why it is known as 'Naive.'

Let us look at the Bayes' Theorem that will make it easier to understand.



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 5.29: *Bayes' theorem*

From now on, we will use $P(y \mid x)$ as this will resemble something similar to features and target format.

[31] "Naive Bayes classifier - Wikipedia." https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

[32] "1.9. Naive Bayes — scikit-learn 0.22.1 documentation." http://scikit-learn.org/stable/modules/naive_bayes.html.

So, let's start with the Bayes' Theorem where given target is y, and the dependent feature vectors are x_1 to x_n.

$$P\left(x \mid x_1,\ldots,x_n\right) = \frac{P(y)P\left(x_1,\ldots,x_n \mid y\right)}{P\left(x_1,\ldots,x_n\right)}$$

Now, using the naive conditional independence[33] assumption that

$$P\left(x_i \mid y, x_1,\ldots,x_{i+1}, x_{i+1},\ldots,x_n\right) = P(x_i \mid y)$$

Simplifying the above equation for all i,

$$P\left(y \mid x_1,\ldots,x_n\right) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1,\ldots,x_n)}$$

Since $P(x_1, ..., x_n)$ is constant for the input, so we can use the classification rule[34]:

$$P\left(y \mid x_1,\ldots,x_n\right) \propto P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

$$\therefore \hat{y} = \arg\max_{y} P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

Now, we can use a MAP(Maximum A Posteriori) estimation for estimating $P(y)$ and $P(x_i \mid y)$.

There are different types of Naive Bayes' classifiers which assume different distribution of $P(x_i \mid y)$. But the working principle remains the same.

We will see another Machine Learning algorithm which uses the Ensemble Learning method.

Random forest[35]

Random forests or **random decision forests** are an ensemble learning method for classification, regression, and other tasks. Here we will mostly use it for classification.

Random forests constructs a multitude number of decision trees at training and, outputs the class. As there are multiple trees, the decision can be calculated in multiple ways that are by the mode of the classes or mean prediction of the individual trees.

[33] "Conditional independence - Wikipedia." https://en.wikipedia.org/wiki/Conditional_independence.

[34] "Classification rule - Wikipedia." https://en.wikipedia.org/wiki/Classification_rule.

[35] "Random forest - Wikipedia." https://en.wikipedia.org/wiki/Random_forest.

The statistical mode is used for categorical target variables, and mean prediction is generally used for continuous target variables.

Let's take an example of a random forest for regression[36].

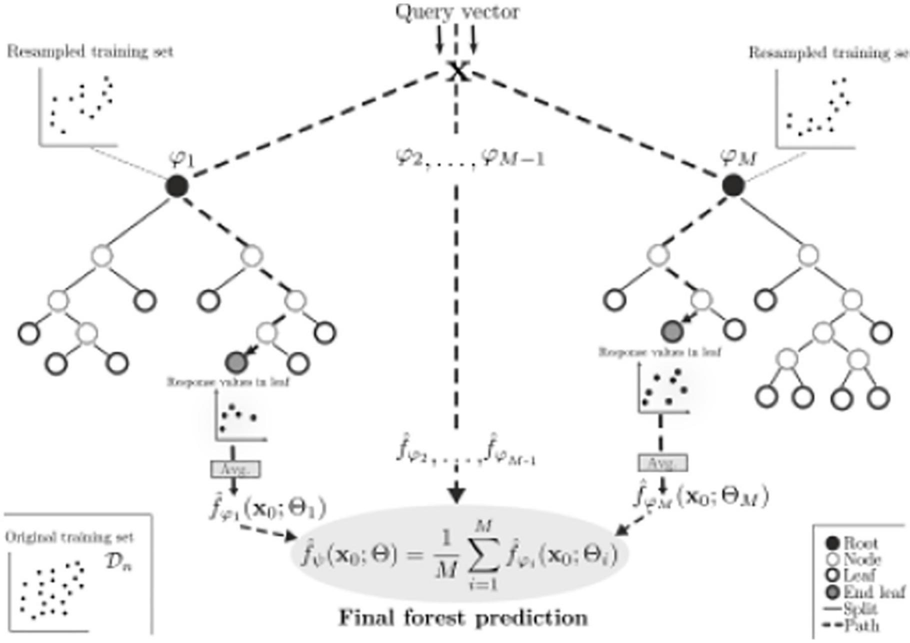

Query vector
Resampled training set
Resampled training set
$\mathbf{X}$
φ_1
$\varphi_2, \ldots, \varphi_{M-1}$
φ_M
Response values in leaf
Response values in leaf
Avg.
Avg.
$\hat{f}_{\varphi_2}, \ldots, \hat{f}_{\varphi_{M-1}}$
$\hat{f}_{\varphi_1}(\mathbf{x}_0; \Theta_1)$
$\hat{f}_{\varphi_M}(\mathbf{x}_0; \Theta_M)$
Original training set
$\mathcal{D}_n$
$\hat{f}_{\psi}(\mathbf{x}_0; \Theta) = \frac{1}{M} \sum_{i=1}^{M} \hat{f}_{\varphi_i}(\mathbf{x}_0; \Theta_i)$
Final forest prediction
● Root
○ Node
○ Leaf
● End leaf
— Split
-- Path

Figure 5.30: *Random Forest with Mean Prediction (Source: wur.nl)*

From the above schematic, we can see there is number of decision trees φ_m. Each decision tree is not the same else, it will yield the same result, and that will not make any sense so, each tree uses a resampled training set.

So, now we will have *m* outputs from φ_m trees i.e. $\hat{f}_{\varphi m}$. We will now take the average of all $\hat{f}_{\varphi m}$ And that will be the output from the random forest. This uses a simple average, but if there is a requirement, a weighted average can be used as well.

Similarly, if we want to use a random forest for classification[37], we can use voting to achieve it.

[36] "Random Forest Regression | Turi Machine Learning Platform ...." https://turi.com/learn/userguide/supervised-learning/random_forest_regression.html.

[37] "Random forests - classification description." https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.

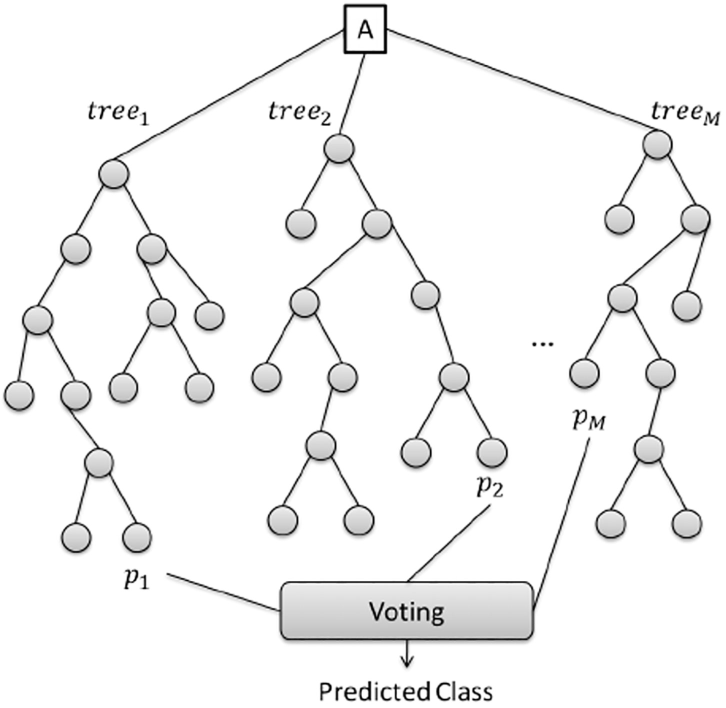


Figure 5.31: *Random Forest with Mode Prediction (Source: ResearchGate)*

In the last image, we have mostly seen how a regression will be computed by taking the average of all the model outputs. From the above image, we can see that we will be conducting voting based on the classification results from all the decision trees. Here we are having *M* count of decision trees, and naturally, we will have *M* predictions. From all the *M* decisions, we need to find the statistical mode. Whichever class has the highest count will be the output from the voting based random forest classifier.

Model training

In this section, we will train different models with different parameters setting, on all datasets and find the optimal model to use for this problem statement.

Generally, before we start with model training, we need to make the dataset ready for training, but in this case, we will be using the given data itself in it's vanilla form. Previously, I have covered the basics of data preparation, so that it will be adequate for this purpose too. But seeing the data, I don't feel additional processing is required, and I will use all the features from the data for the training purpose.

Training & basic evaluation

We will be using multiple algorithms with cross-validation to get optimal results. In this section, we will also handle some hyperparameter optimization techniques using a search algorithm, also known as **GridSearchCV**.

We will start with model training, and in the following section, we will talk more about some new techniques.

Naive Bayes

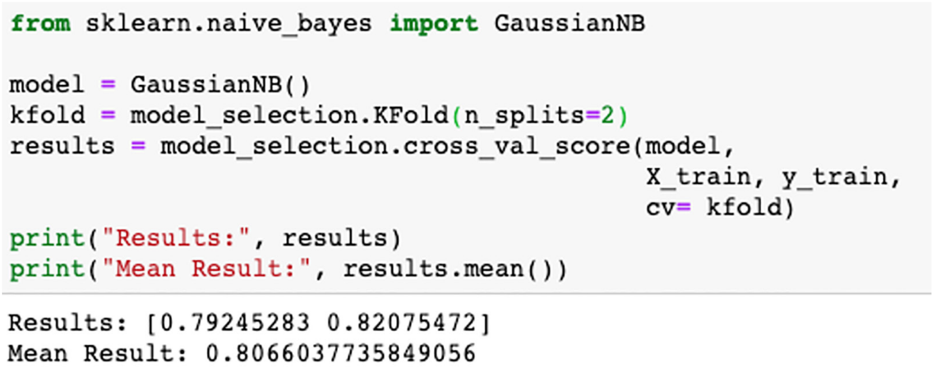
We now know what Naive Bayes algorithm is by now and we have discussed that different Naive Bayes algorithm considers the different distribution of $P(x_i \mid y)$.

In this section the likelihood $P(x_i \mid y)$ of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2} \right)$$

Seeing the above equation, we can say it resembles a Gaussian distribution, which we had seen in previous chapters.

Using this likelihood distribution, let us fit the training data and see how it scores.



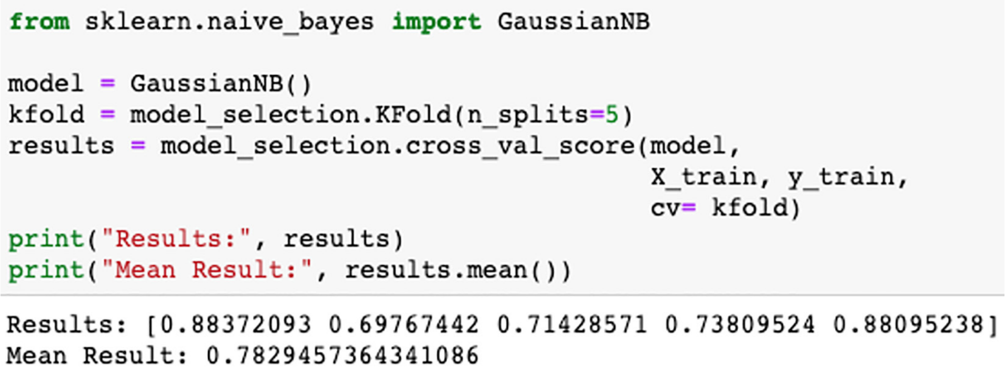
```python
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
kfold = model_selection.KFold(n_splits=2)
results = model_selection.cross_val_score(model,
                                          X_train, y_train,
                                          cv= kfold)
print("Results:", results)
print("Mean Result:", results.mean())
```

```
Results: [0.79245283 0.82075472]
Mean Result: 0.8066037735849056
```

Figure 5.32: Gaussian Naive Bayes with 2 folds

Sklearn offers a Gaussian Naive Bayes to fit the dataset. We are using cross-validation with 2-fold, and we can see individual cross-validation results for both of them. When we use multiple folds, we tend to find the mean and give it as a single result.

Similarly, we will now see if we increase the number of folds will there be any significant change in the result.



```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
kfold = model_selection.KFold(n_splits=5)
results = model_selection.cross_val_score(model,
                                          X_train, y_train,
                                          cv= kfold)
print("Results:", results)
print("Mean Result:", results.mean())
```

```
Results: [0.88372093 0.69767442 0.71428571 0.73809524 0.88095238]
Mean Result: 0.7829457364341086
```

Figure 5.33: *Gaussian Naive Bayes with five folds*

We can now see the results if we have five folds interestingly, it has decreased the average score. This is because the size of the dataset is so small when we perform k-fold cross-validation, it reduces the size of the dataset in each fold that leads to a low cross-validation score.

This is a medical dataset, and we have to reduce total False Negatives from the dataset, i.e., we have to reduce the cases to make it nil where a positive heart disease case is mispredicted as a negative heart patient. This kind of misprediction can lead to the worst cases, which we need to take care of. Let us see the confusion matrix, which is made using the test data.

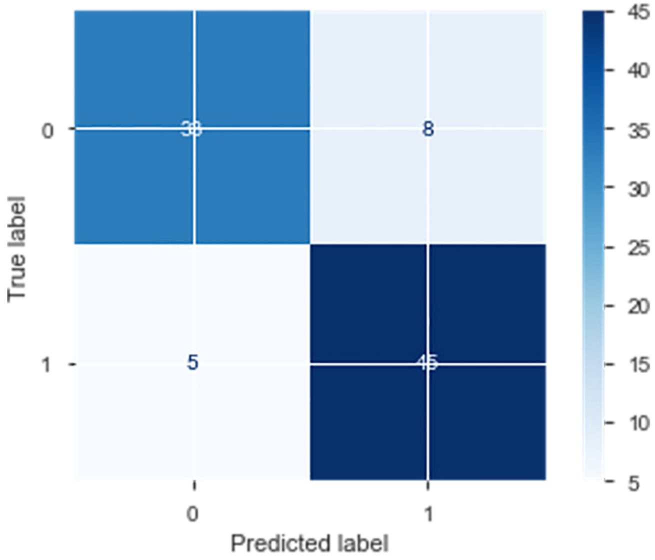


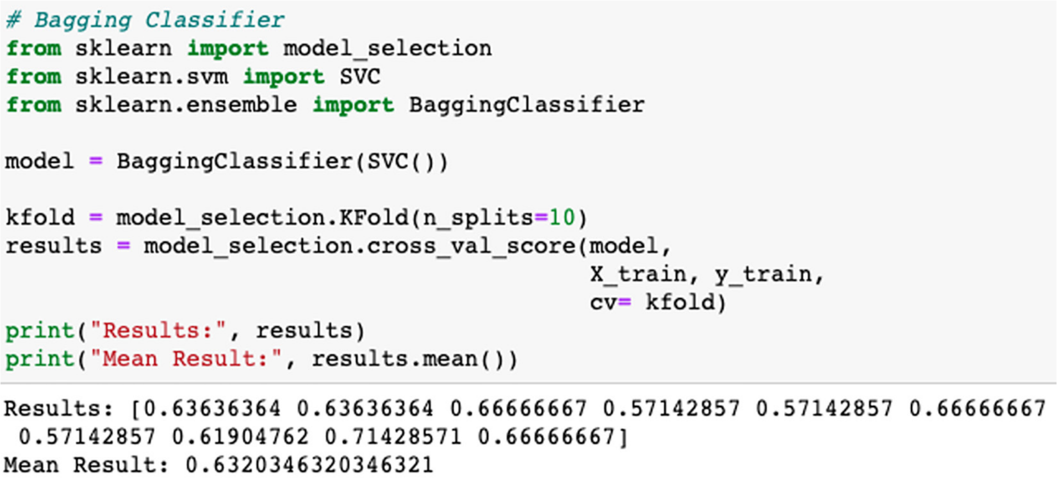
Figure 5.34: *Confusion Matrix for Naive Bayes*

There's a total of 5 such cases where there is a misprediction in False Negative. As we had said earlier that we need to either compromise on False Positive or compromise on False Negative when re-tuning the model.

Ensemble method (Bagging)

Previously we have explained different types of Ensemble methods. Here we will take a look at some ensemble bagging methods.

Generally, ensembles are represented with tree-like structures, but we should know that it can be used with other models too. In the below example, we will use a support vector machine and Bagging technique to fit the heart disease dataset.



```
# Bagging Classifier
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier

model = BaggingClassifier(SVC())

kfold = model_selection.KFold(n_splits=10)
results = model_selection.cross_val_score(model,
                                          X_train, y_train,
                                          cv= kfold)
print("Results:", results)
print("Mean Result:", results.mean())
```

```
Results: [0.63636364 0.63636364 0.66666667 0.57142857 0.57142857 0.66666667
 0.57142857 0.61904762 0.71428571 0.66666667]
Mean Result: 0.6320346320346321
```

Figure 5.35: *Bagging Classifier*

From the above code snippet, we can see that we are splitting the dataset into ten folds for cross-validation and fitting it into the Support Vector Machine Classifier and finally packing it into a Bagging Classifier. Bagging Classifier can either perform a mean or a mode depending on the output from the model. Later we will see another technique where it will only perform mode, i.e., voting from all the models.

Now, let us see the confusion matrix for the Bagging classifier on the testing dataset.

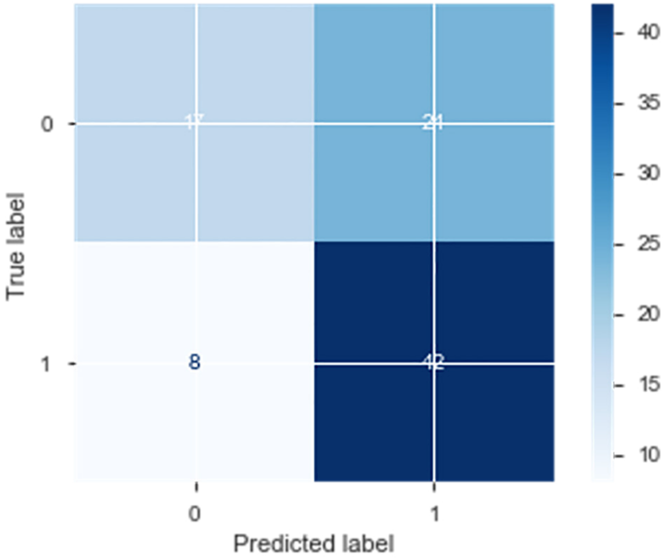
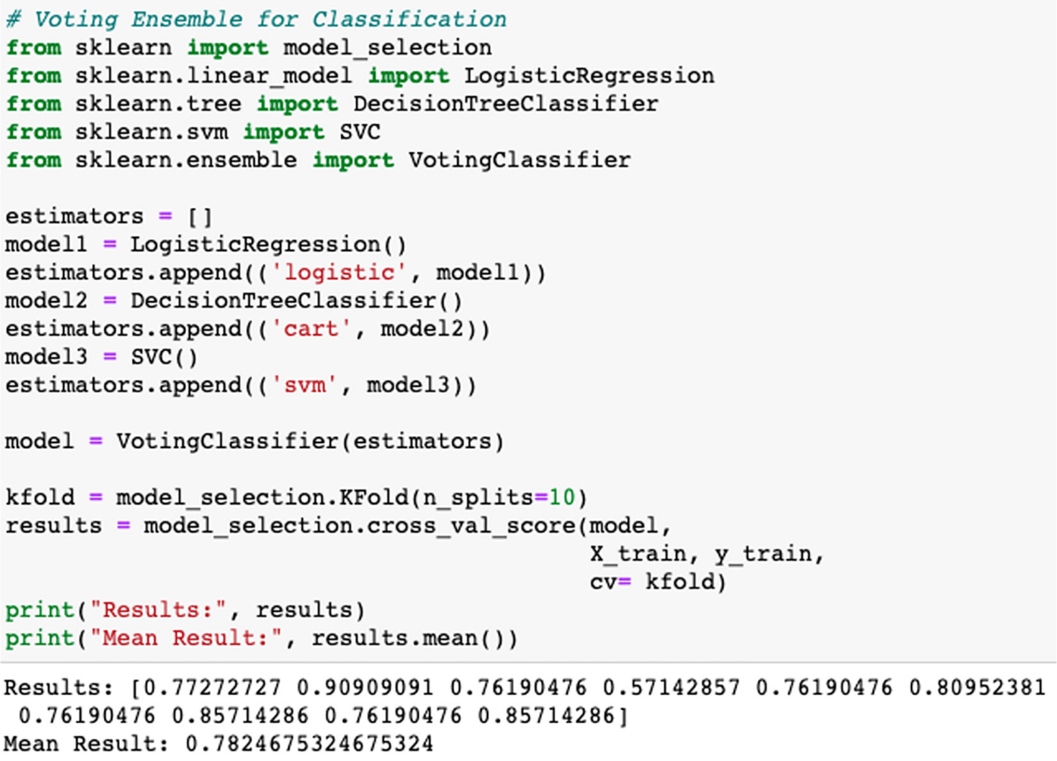


Figure 5.36: *Confusion Matrix for Bagging Classifier*

We can see the False Negatives have substantially increased, so we either need to tune the model or change the type of classifier for the optimal result.

Let us take a look at another type of ensemble bagging classifier, i.e., voting bases classifier. Here we will use multiple models to fit the dataset and evaluate them. Using multiple models increases the coverage of the problem, and the results tend to be better, and it performs extremely well in testing data.

We are planning to use SVM, Logistic Regression, Decision Tree, and make a package of that single model.



```
# Voting Ensemble for Classification
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))

model = VotingClassifier(estimators)

kfold = model_selection.KFold(n_splits=10)
results = model_selection.cross_val_score(model,
                                          X_train, y_train,
                                          cv= kfold)
print("Results:", results)
print("Mean Result:", results.mean())
```

```
Results: [0.77272727 0.90909091 0.76190476 0.57142857 0.76190476 0.80952381
 0.76190476 0.85714286 0.76190476 0.85714286]
Mean Result: 0.7824675324675324
```

Figure 5.37: *Voting Classifier of Mixed ML models*

We can say that each model will have three algorithms, and each one will give an output. From all the output, we need to mind the statistical mode of the model, and that will be the final output.

Let us take a look at the confusion matrix, which will be performing better compared to others as per the logic behind creating a mixture model.

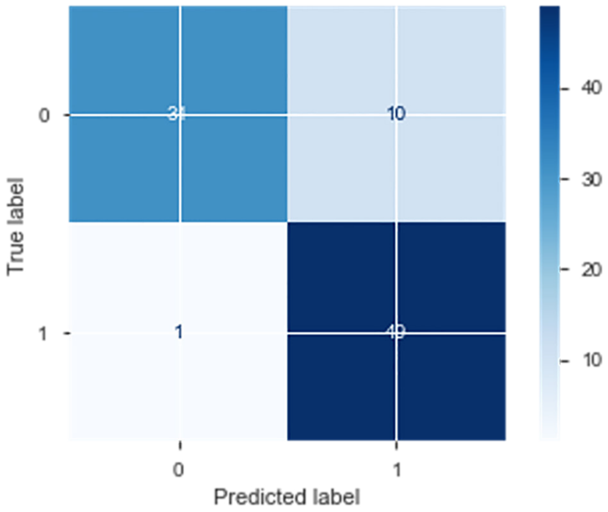


Figure 5.38: Confusion Matrix for Voting Classifier

We can see that this model performed extremely well as there is only one case of False Negative, and that is a significantly low number. We had discussed that if False Negative is low, then it generally is a trade-off where False Positives will be high, in this case, we can see the same. Here, the ratio between FN and FP is 1:10.

But this model is good enough to be pushed for production.

Ensemble method (Boosting)

In the last section, we saw the Ensemble method for Bagging, and here we will see another method for that is Boosting. We have covered the theory for boosting in the Prerequisites section.

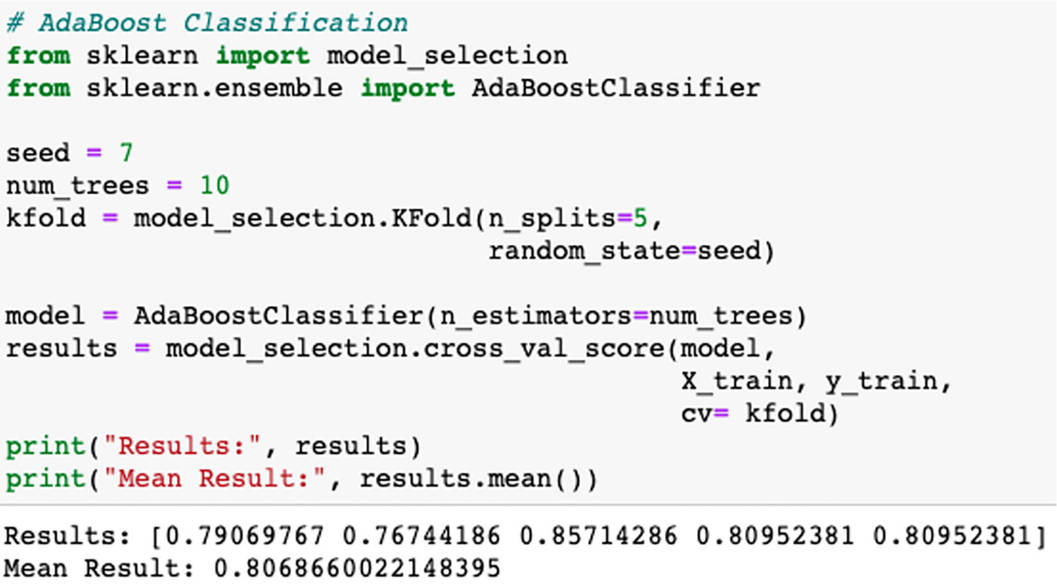
For ensemble boosting methods, we will be using adaptive boosting techniques[38] , which are also known as **Adaboost**.

Adaboost can be used in conjunction with many other types of machine learning algorithms to improve the overall performance of the test dataset. The output of the other machine learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier.

AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. This process generally helps to reduce the misclassification rate.

[38] "AdaBoost - Wikipedia." https://en.wikipedia.org/wiki/AdaBoost.

Let us train the model with the training dataset and later see the confusion matrix for the performance.



```
# AdaBoost Classification
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier

seed = 7
num_trees = 10
kfold = model_selection.KFold(n_splits=5,
                              random_state=seed)

model = AdaBoostClassifier(n_estimators=num_trees)
results = model_selection.cross_val_score(model,
                                          X_train, y_train,
                                          cv= kfold)
print("Results:", results)
print("Mean Result:", results.mean())
```

```
Results: [0.79069767 0.76744186 0.85714286 0.80952381 0.80952381]
Mean Result: 0.8068660022148395
```

Figure 5.39: AdaBoost

As this is a tree-based model, we can explicitly tune the number of trees that will be one of the hyperparameters for Adaboost. We will know more about hyperparameter and hyperparameter tuning in the upcoming sections.

Now let us see what the confusion matrix looks like.

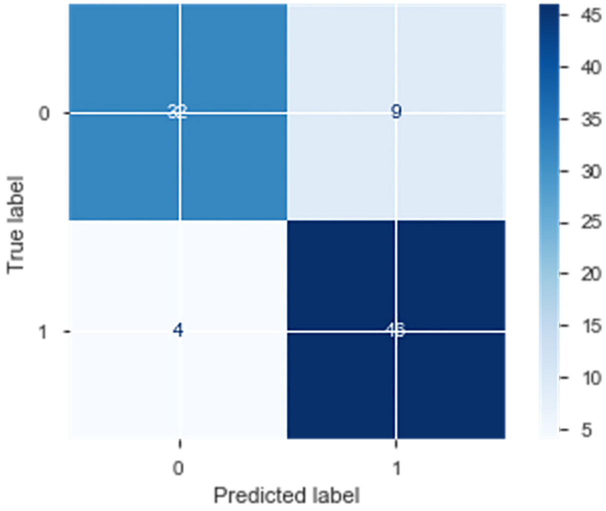


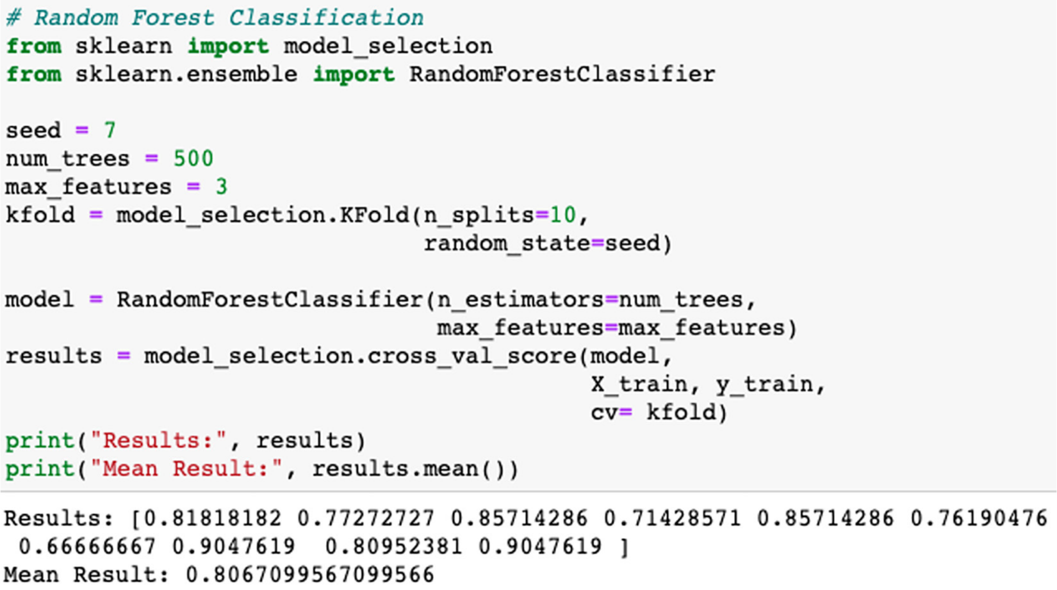
Figure 5.40: Confusion Matrix for AdaBoost

Adaboost performs better than some of the approaches, but it is not the best. A mixed model bagging classifier performed best on testing data. But this model is good as well, but not best among the bunch.

Random forest

As we know, the random forest is an ensemble method of learning, and it is formed with multiple random decision trees. Like other algorithms, the random forest has a lot of hyper-parameters. We will be using "`n_estimators`" and "`max_features`" for this example, but it can be any tunable parameters.

Let us first set some default values to the hyperparameters and fit the model to the training data.



```
# Random Forest Classification
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier

seed = 7
num_trees = 500
max_features = 3
kfold = model_selection.KFold(n_splits=10,
                              random_state=seed)

model = RandomForestClassifier(n_estimators=num_trees,
                               max_features=max_features)
results = model_selection.cross_val_score(model,
                                          X_train, y_train,
                                          cv= kfold)
print("Results:", results)
print("Mean Result:", results.mean())
```

```
Results: [0.81818182 0.77272727 0.85714286 0.71428571 0.85714286 0.76190476
 0.66666667 0.9047619  0.80952381 0.9047619 ]
Mean Result: 0.8067099567099566
```

Figure 5.41: *Random forest*

With ten folds, 500 trees, three max features, we are getting a good score of 0.80194.

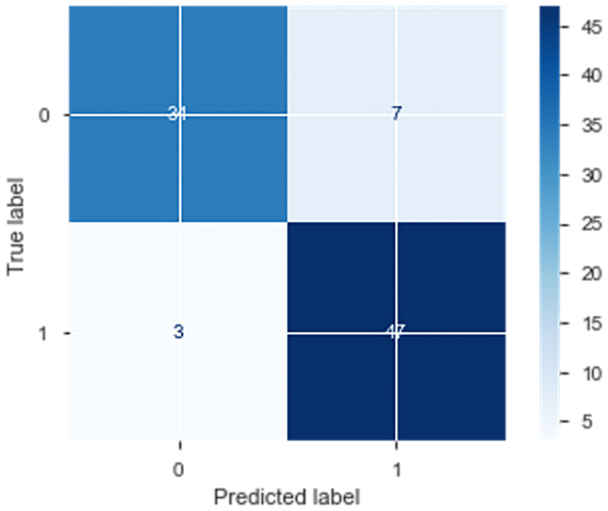


Figure 5.42: *Confusion Matrix for random forest*

As this is a tree-based algorithm, it will be computed using either Information Gain or Gini. If that is the case, then all tree-based algorithms will have something known as **Feature Importance** (where all the features will be ranked as per some important score).

We will take a rough idea of how the calculation for feature importance works for the random forest.

First, let us calculate the node importance, i.e., *ni* using Gini Importance

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - W_{right(j)} C_{right(j)}$$

Where,

ni_j is the importance of node *j*

w_j is weighted no. of samples reaching node *j*

C_j is impurity value of node *j*

Left and right indicated the child node from the split of node *j*

Now, we need to find the Feature Importance for each feature in the decision tree.

$$FI_i = \frac{\sum_{j:node\ j\ splits\ on\ feature\ i} ni_j}{\sum_{k \in all\ nodes} ni_k}$$

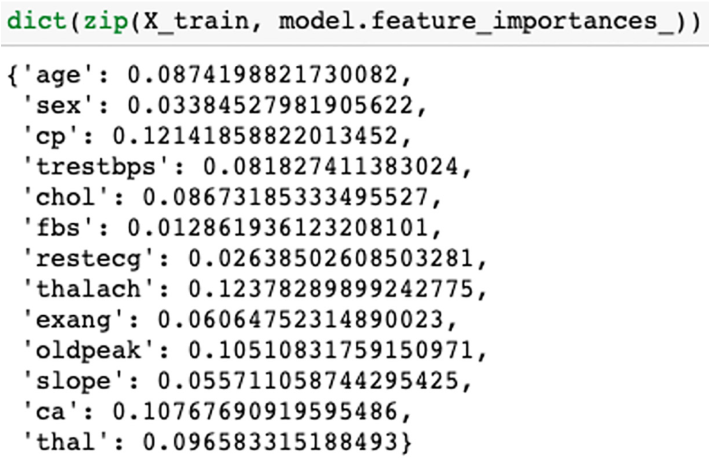
These Feature Importance can be normalized from 0 to 1 by dividing it by the sum of all Feature Importance values.

$$Normalise\ FI_i = \frac{FI_i}{\sum_{j\in all\ features} FI_j}$$

As we have now normalized the feature, we can now find the feature importance for the random forest. It is now just, the overall average of the decision trees.

$$RFFI_i = \frac{\sum_{j\in all\ trees} FI_{ij}}{T}$$

With the above concept, let us see the feature importance for the model from the random forest.



```
dict(zip(X_train, model.feature_importances_))
```

```
{'age': 0.0874198821730082,
 'sex': 0.03384527981905622,
 'cp': 0.12141858822013452,
 'trestbps': 0.081827411383024,
 'chol': 0.08673185333495527,
 'fbs': 0.012861936123208101,
 'restecg': 0.02638502608503281,
 'thalach': 0.12378289899242775,
 'exang': 0.06064752314890023,
 'oldpeak': 0.10510831759150971,
 'slope': 0.055711058744295425,
 'ca': 0.10767690919595486,
 'thal': 0.096583315188493}
```

Figure 5.43: *Feature Importance*

From the above feature importance dictionary, we can see that features like '`ca`,' '`cp`,' '`age`,' '`oldpeak`' have higher scores.

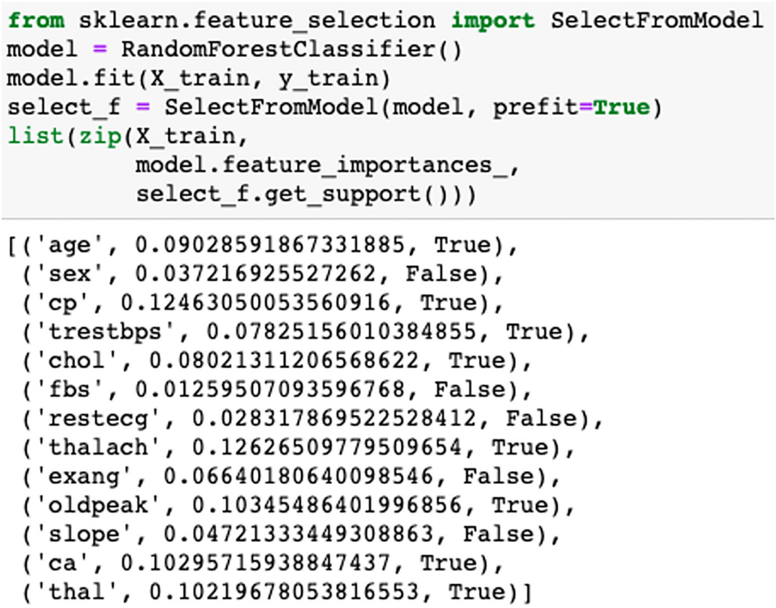
Previously we have used correlation to select some of the features from a large set for training the model. But now we have seen another method to find the importance. We can use Feature Importance from Random Forest for feature selection, like after sorting the list; we can select the top 5 or 7 features from the list to reduce from the size of the entire feature set.

Feature selection & Grid searching

This section will cover something related to hyperparameter tuning and some of the feature selection as well. We will also learn how we can pipeline a model with pre-processing.

Feature selection

We have seen a correlation-based feature selection, but here we will be using tree-based feature importance for feature selection.



```
from sklearn.feature_selection import SelectFromModel
model = RandomForestClassifier()
model.fit(X_train, y_train)
select_f = SelectFromModel(model, prefit=True)
list(zip(X_train,
         model.feature_importances_,
         select_f.get_support()))
```

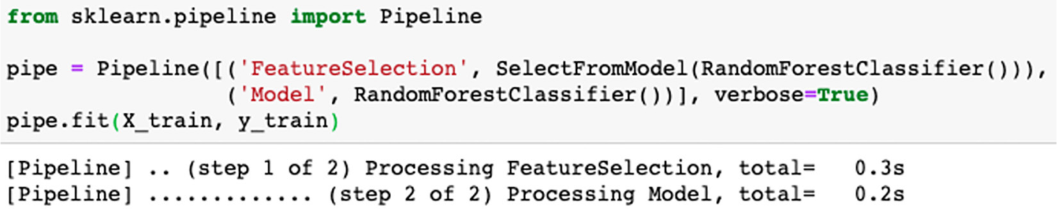
```
[('age', 0.09028591867331885, True),
 ('sex', 0.037216925527262, False),
 ('cp', 0.12463050053560916, True),
 ('trestbps', 0.07825156010384855, True),
 ('chol', 0.08021311206568622, True),
 ('fbs', 0.01259507093596768, False),
 ('restecg', 0.028317869522528412, False),
 ('thalach', 0.12626509779509654, True),
 ('exang', 0.06640180640098546, False),
 ('oldpeak', 0.10345486401996856, True),
 ('slope', 0.04721333449308863, False),
 ('ca', 0.10295715938847437, True),
 ('thal', 0.10219678053816553, True)]
```

Figure 5.44: *Feature Importance with Feature Selection*

From the above image, we can see that we are using the random forest for feature selection, and we can also see which features are selected among the entire feature set. With each feature, there is a Boolean flag, i.e., if it is true, then it is selected for training, and if it is False, then the feature will be ignored. This is achieved using a threshold value or counts. Say, if we keep the threshold as 0.1, then any value greater or equal to that value will be considered for training else it will be ignored and dropped. Or we can select the features like top features from the feature set.

Pipeline

A **pipeline** is a great way to package pre-processing, feature selection, model, etc. in a single package or pipeline. When we make a pipeline, all the steps become serial. Let us first see an example; then we will explain bits and parts of it.



```
from sklearn.pipeline import Pipeline

pipe = Pipeline([('FeatureSelection', SelectFromModel(RandomForestClassifier())),
                 ('Model', RandomForestClassifier())], verbose=True)
pipe.fit(X_train, y_train)
```

```
[Pipeline] .. (step 1 of 2) Processing FeatureSelection, total=   0.3s
[Pipeline] ............. (step 2 of 2) Processing Model, total=   0.2s
```

Figure 5.45: *Making Pipeline*

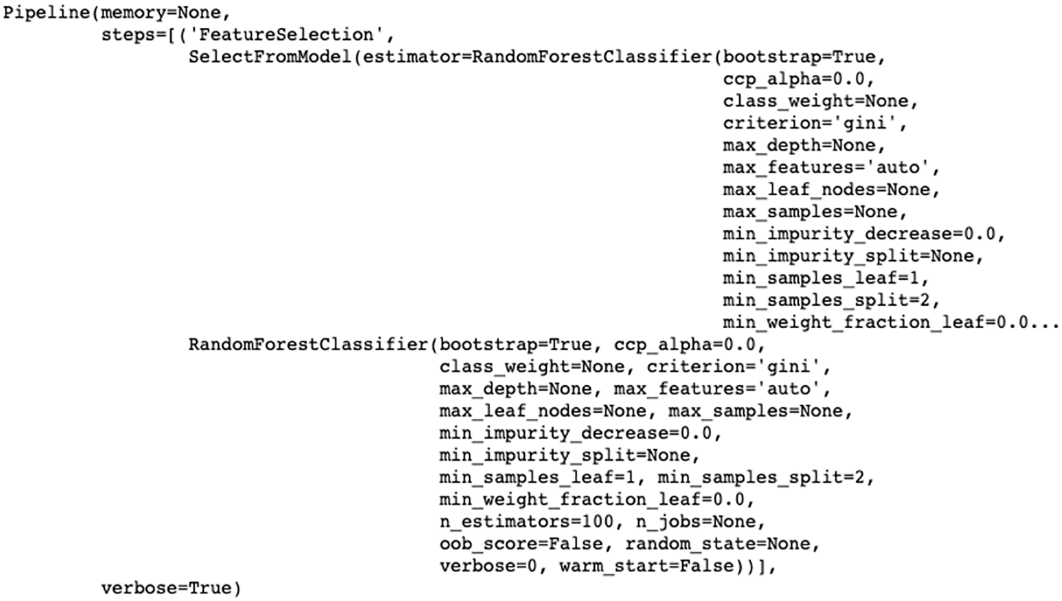
From the above, we know how we can pipeline a feature selection and a model at the same time. This can be used with many functions, and this is serial.

The pipeline consists of 'FeatureSelection' and 'Model,' and when we fit the data to the pipeline, it will first execute 'FeatureSelection,' then the output will be fitted to the model. Using 'FeatureSelection,' we will reduce the size of the trainable features first; then we will use the small set of trainable features and fit it to Random Forest.

If there are steps to the pipeline, then the data will be transferred one after another.

Pipelining a model with everything is a great way to use it in production.

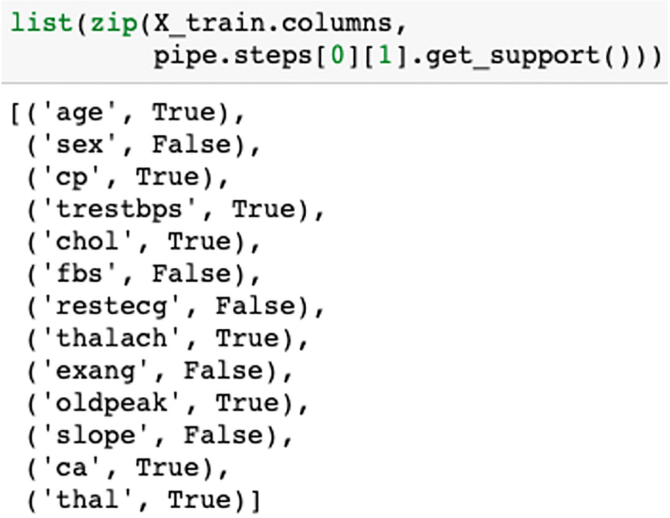
Let us see some of the contents of the variable pipeline.



```
Pipeline(memory=None,
         steps=[('FeatureSelection',
                 SelectFromModel(estimator=RandomForestClassifier(bootstrap=True,
                                                                  ccp_alpha=0.0,
                                                                  class_weight=None,
                                                                  criterion='gini',
                                                                  max_depth=None,
                                                                  max_features='auto',
                                                                  max_leaf_nodes=None,
                                                                  max_samples=None,
                                                                  min_impurity_decrease=0.0,
                                                                  min_impurity_split=None,
                                                                  min_samples_leaf=1,
                                                                  min_samples_split=2,
                                                                  min_weight_fraction_leaf=0.0...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='gini',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=100, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=True)
```

Figure 5.46: *Pipeline contents*

In the variable, we can see that we have all the models and pre-processing in 'steps' in form of 'list.' Let us access the first step of the pipeline and see which of the variables got selected for training.

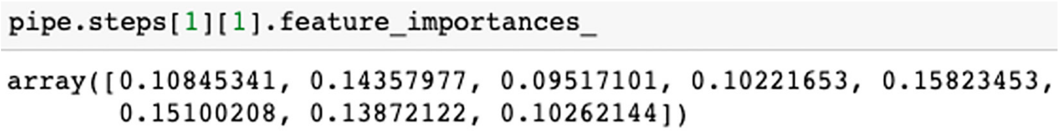


```
list(zip(X_train.columns,
         pipe.steps[0][1].get_support()))
```

```
[('age', True),
 ('sex', False),
 ('cp', True),
 ('trestbps', True),
 ('chol', True),
 ('fbs', False),
 ('restecg', False),
 ('thalach', True),
 ('exang', False),
 ('oldpeak', True),
 ('slope', False),
 ('ca', True),
 ('thal', True)]
```

***Figure 5.47:** Pipeline Feature Selection*

We can see when the flag is "`True`," then that will be considered as a trainable feature else it will be ignored. Similarly, we will take a look at the feature importance for all the True flagged features.



```
pipe.steps[1][1].feature_importances_
```

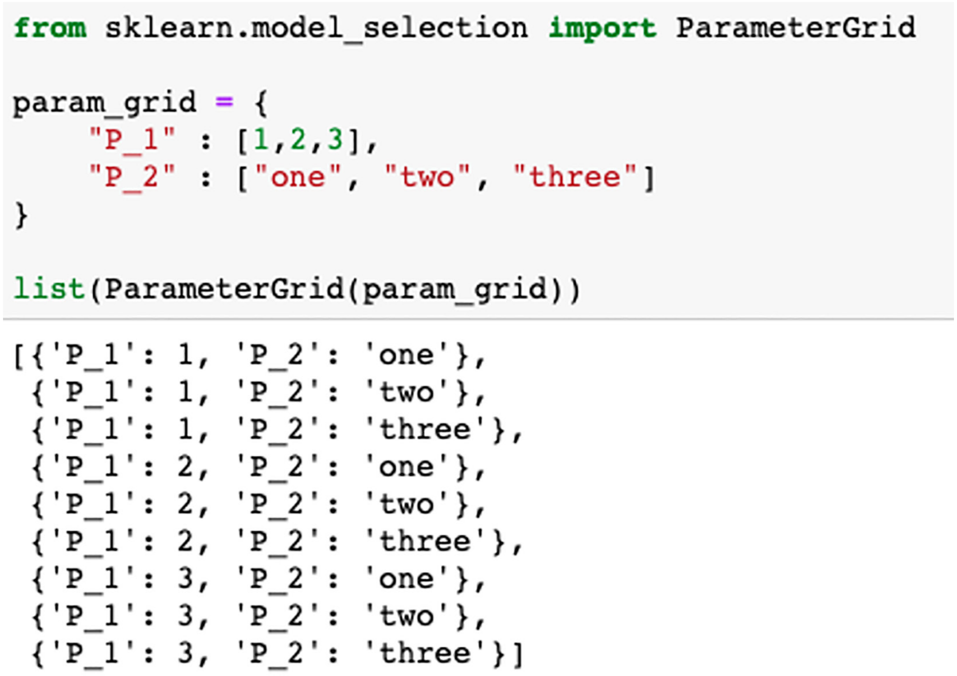
```
array([0.10845341, 0.14357977, 0.09517101, 0.10221653, 0.15823453,
       0.15100208, 0.13872122, 0.10262144])
```

***Figure 5.48:** Pipeline Model Feature Importance*

We can see there were 8 "true flagged features" from *Figure 5.47*, and the above image shows their feature importance in order. This is a basic example of using a pipeline. We can make it more complex, depending on the situation. But here we will only see the basic one. Please refer to the footnote for more reference.

Grid searching

Grid searching is a way to select the best hyperparameter combination from the given set of options. Let us first see an example code snippet to understand the concept further.



```
from sklearn.model_selection import ParameterGrid

param_grid = {
    "P_1" : [1,2,3],
    "P_2" : ["one", "two", "three"]
}

list(ParameterGrid(param_grid))
```

```
[{'P_1': 1, 'P_2': 'one'},
 {'P_1': 1, 'P_2': 'two'},
 {'P_1': 1, 'P_2': 'three'},
 {'P_1': 2, 'P_2': 'one'},
 {'P_1': 2, 'P_2': 'two'},
 {'P_1': 2, 'P_2': 'three'},
 {'P_1': 3, 'P_2': 'one'},
 {'P_1': 3, 'P_2': 'two'},
 {'P_1': 3, 'P_2': 'three'}]
```

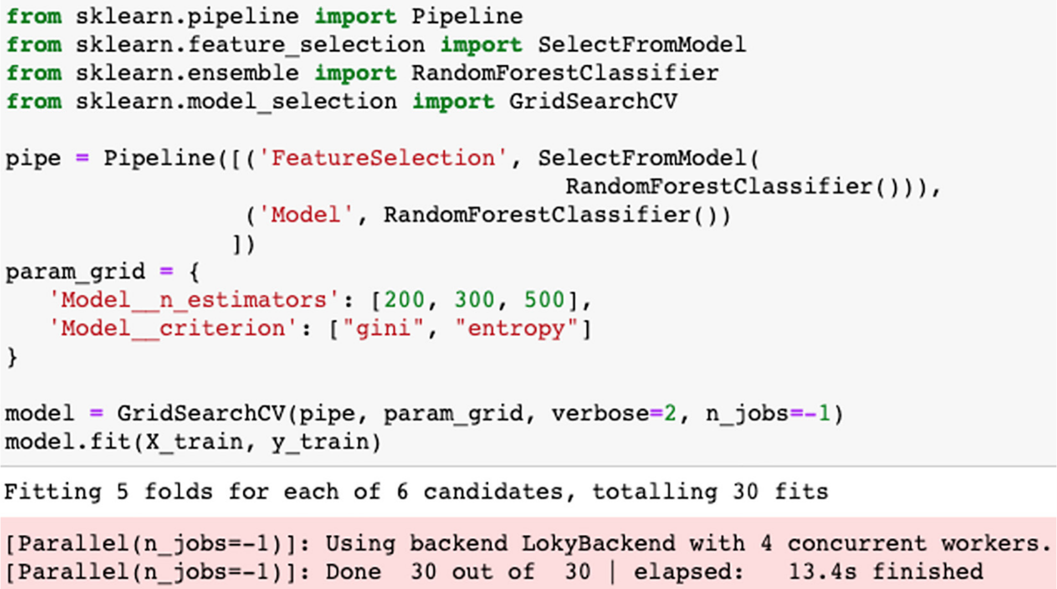
***Figure 5.49:** Parameter Grid*

From the above code, we can assume there are two hyperparameters: ‘`P_1`’ and ‘`P_2`’. These parameters can have a list of values, or we can search with a range of values. The search space for all the hyperparameters is defined by us.

From the output of the code, we can see that it gives us all the permutation and combination of all the hyperparameters. It searches the entire space, and thus, it increases space and time complexity.

Grid searching is a stochastic method for finding the best/optimal value of the hyperparameter for the model, which yields the best score. And, in terms of complexity, it is very high.

As we have already seen, pipelines and feature selection, so we will write a code snippet combining all the techniques.



```
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

pipe = Pipeline([('FeatureSelection', SelectFromModel(
                                        RandomForestClassifier())),
                 ('Model', RandomForestClassifier())
                ])
param_grid = {
   'Model__n_estimators': [200, 300, 500],
   'Model__criterion': ["gini", "entropy"]
}

model = GridSearchCV(pipe, param_grid, verbose=2, n_jobs=-1)
model.fit(X_train, y_train)
```

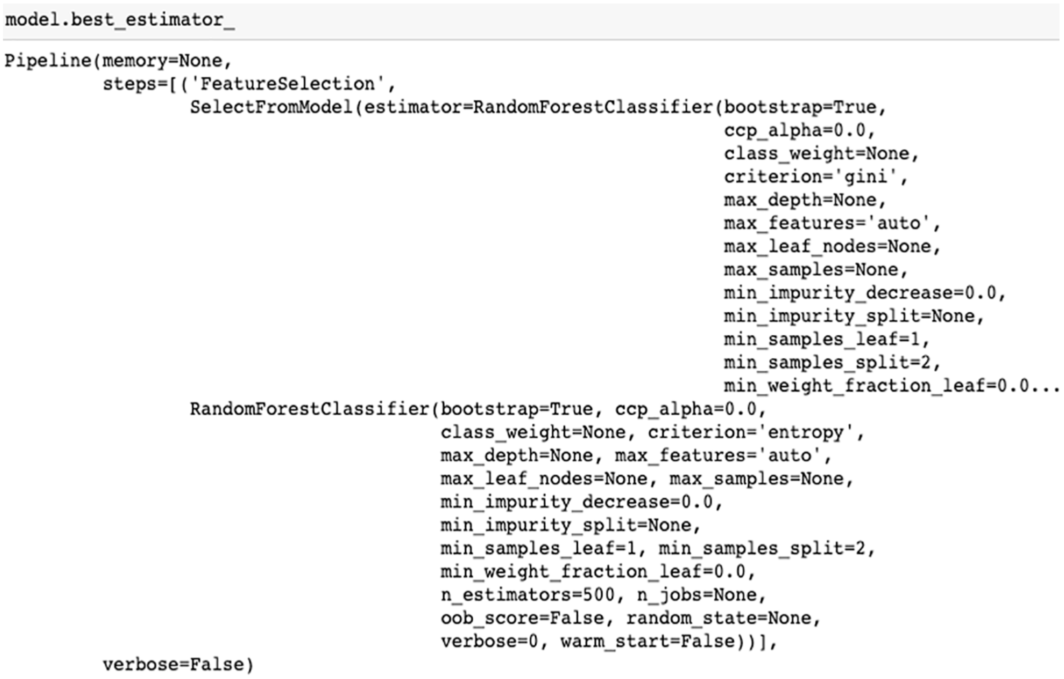
```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:   13.4s finished
```

Figure 5.50: *Pipeline + GridSearchCV with random forest*

The above code snippet makes a pipeline, then we initialize a hyperparameter search space for ‘`Model`’ alone using ‘`<Step Name>__<Tunable Hyperparameter Name>`’ (in this example we are only tuning two hyper parameters, but this can be performed with all tuneable hyperparameters for any functions). For the model Random Forest, the hyperparameters we will be using are ‘`n_estimators`’ and ‘`criterion`.’ ‘`n_estimators`’ are the count of trees in the Random Forest model and ‘criterion’ is the method by which a node in a decision tree is decided either using entropy or using Gini index/coefficient. We are planning to find the best combination of parameters which yield the best result.

After executing the model using the GridSearchCV, let us see the best estimator and its configuration.



```
model.best_estimator_
```

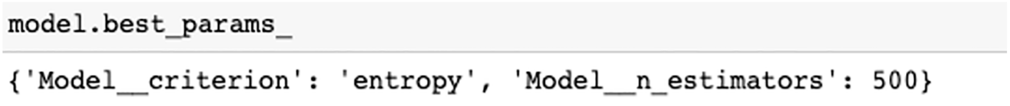
```
Pipeline(memory=None,
         steps=[('FeatureSelection',
                 SelectFromModel(estimator=RandomForestClassifier(bootstrap=True,
                                                                  ccp_alpha=0.0,
                                                                  class_weight=None,
                                                                  criterion='gini',
                                                                  max_depth=None,
                                                                  max_features='auto',
                                                                  max_leaf_nodes=None,
                                                                  max_samples=None,
                                                                  min_impurity_decrease=0.0,
                                                                  min_impurity_split=None,
                                                                  min_samples_leaf=1,
                                                                  min_samples_split=2,
                                                                  min_weight_fraction_leaf=0.0...
                 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                        class_weight=None, criterion='entropy',
                                        max_depth=None, max_features='auto',
                                        max_leaf_nodes=None, max_samples=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_samples_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=500, n_jobs=None,
                                        oob_score=False, random_state=None,
                                        verbose=0, warm_start=False))],
         verbose=False)
```

Figure 5.51: *Best estimator from GridSearch*

As we are using a pipeline, so we will get the entire pipeline as an estimator, and not only the machine learning model. After the execution of the grid search, we can see the above pipeline with the given steps, which yielded the best result for the trained data.

We should always keep in mind that "GridSearchCV" is more of a concept that can be implemented using ParameterGrid. More we use, we will understand building a custom grid search where we will search both hyperparameter space and model space. Like, we will have three models, and each model 6 different permutation and combination of the hyperparameter, then the total search space becomes 18. So, it will make 18 models and train their performance.

Later we will test the model on training data and see the confusion matrix. As we performed a GridSearchCV, we will have the best parameters too.

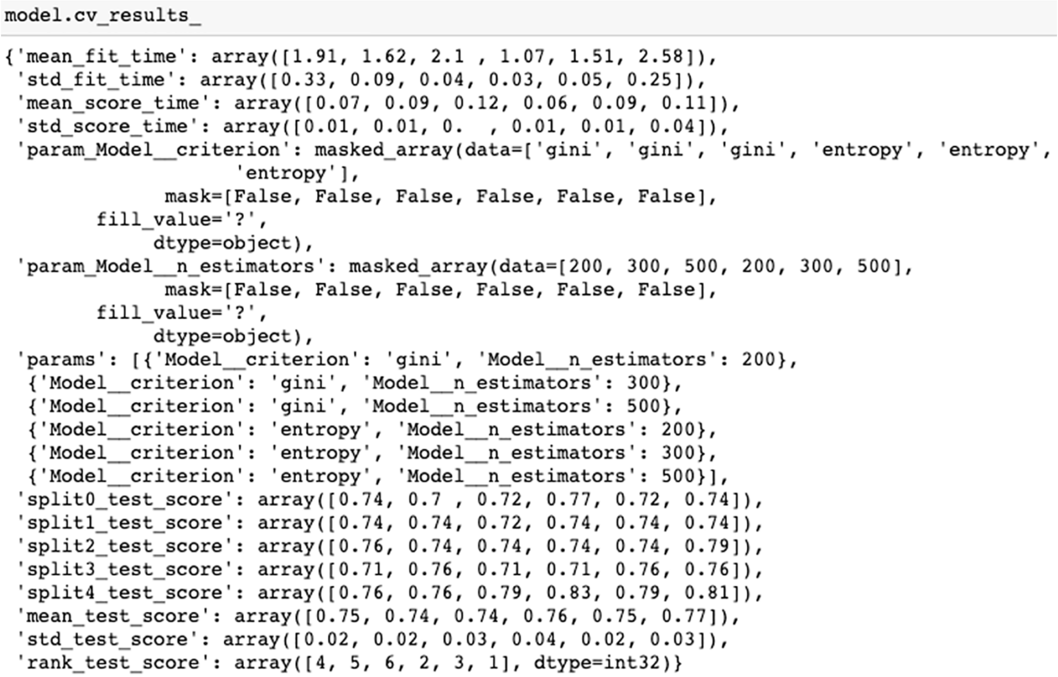


```
model.best_params_
```

```
{'Model__criterion': 'entropy', 'Model__n_estimators': 500}
```

Figure 5.52: *Best Parameters from GridSearch*

From the above figure, we can see the best parameters for the model pipeline using GridSearchCV. Interestingly ‘entropy’ turned out to be the best feature we can see all the results for all the combinations for validation or selecting or tuning the grid search parameters. Let us quickly take a look at all the results for all the permutation and combination.



```
model.cv_results_
```

```
{'mean_fit_time': array([1.91, 1.62, 2.1 , 1.07, 1.51, 2.58]),
 'std_fit_time': array([0.33, 0.09, 0.04, 0.03, 0.05, 0.25]),
 'mean_score_time': array([0.07, 0.09, 0.12, 0.06, 0.09, 0.11]),
 'std_score_time': array([0.01, 0.01, 0.  , 0.01, 0.01, 0.04]),
 'param_Model__criterion': masked_array(data=['gini', 'gini', 'gini', 'entropy', 'entropy',
                    'entropy'],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_Model__n_estimators': masked_array(data=[200, 300, 500, 200, 300, 500],
              mask=[False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'Model__criterion': 'gini', 'Model__n_estimators': 200},
  {'Model__criterion': 'gini', 'Model__n_estimators': 300},
  {'Model__criterion': 'gini', 'Model__n_estimators': 500},
  {'Model__criterion': 'entropy', 'Model__n_estimators': 200},
  {'Model__criterion': 'entropy', 'Model__n_estimators': 300},
  {'Model__criterion': 'entropy', 'Model__n_estimators': 500}],
 'split0_test_score': array([0.74, 0.7 , 0.72, 0.77, 0.72, 0.74]),
 'split1_test_score': array([0.74, 0.74, 0.72, 0.74, 0.74, 0.74]),
 'split2_test_score': array([0.76, 0.74, 0.74, 0.74, 0.74, 0.79]),
 'split3_test_score': array([0.71, 0.76, 0.71, 0.71, 0.76, 0.76]),
 'split4_test_score': array([0.76, 0.76, 0.79, 0.83, 0.79, 0.81]),
 'mean_test_score': array([0.75, 0.74, 0.74, 0.76, 0.75, 0.77]),
 'std_test_score': array([0.02, 0.02, 0.03, 0.04, 0.02, 0.03]),
 'rank_test_score': array([4, 5, 6, 2, 3, 1], dtype=int32)}
```

Figure 5.53: *Cross-Validation Results for GridSearch*

The above figure gives a detailed feature for all the models starting from ‘mean_fit_time,’ ‘mean_test_score’ to ‘rank_test_score.’ If we carefully take a look at ‘rank_test_score,’ that gives us the ranking for all the permutation and combinations for the hyperparameters based on ‘mean_test_score.’ For more understanding, it is recommended to go through all the metrics and individually understand the significance of it.

Now, we know the best parameter, and it is already fitted with the training data. So now, we can go ahead and evaluate it with the test data.

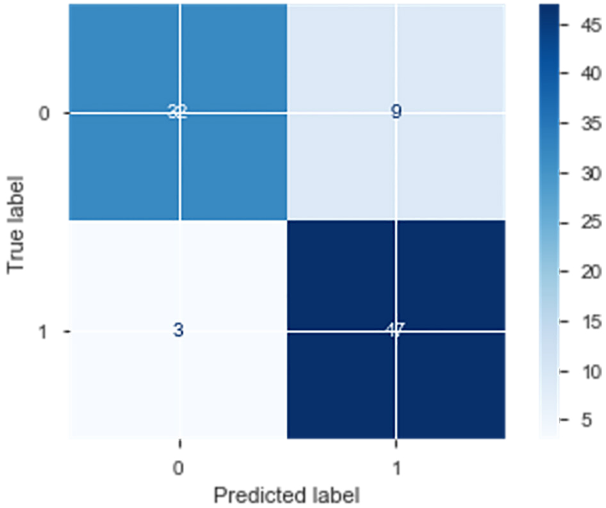


Figure 5.54: *Confusion Matrix for the Best Estimator*

Whatever we did so far was all because we want a good result on unknown, unseen data. This model performs fairly well, and it is used for production as well.

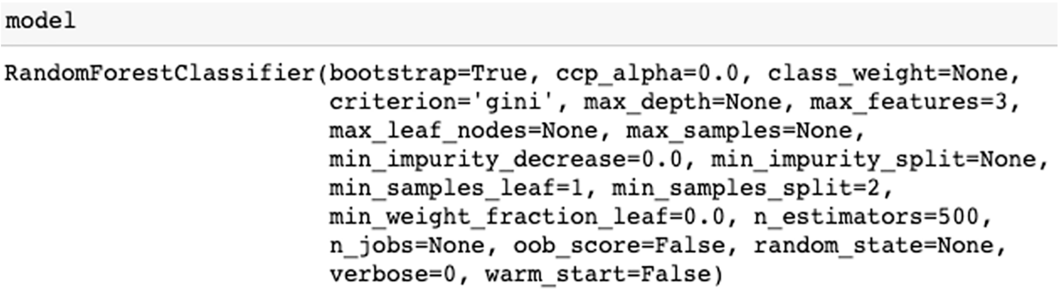
One thing we should keep in mind is that when we encounter situations where a pre-build package won't be of any help, then we should have the practice to build it from scratch to make a solution.

The next section will be a kick-starter for the concept of Explainable AI, and all the explanations will be partial, and it won't be covered as it will be out of scope for this book. But I feel this is an important aspect of Machine Learning along with the ethics of Machine Learning, which everyone should be aware of and use it in practice.

Explainable AI

Few sections back, we have discussed Explainable AI and why it is so important for us to know the working of the black-box model. Here, in this section, we will take a look at some of the techniques with a rudimentary explanation. We will only see some techniques on how a model's output can be explained for this dataset, which we are dealing with in this chapter. If someone is willing to explore this area of Machine Learning, then I will advise them to go through all the footnotes in this section. Hopefully, that will give a base introduction to understand and explain a machine learning model.

Let us first take a look at which model we will explain and understand their outputs.



```
model
```

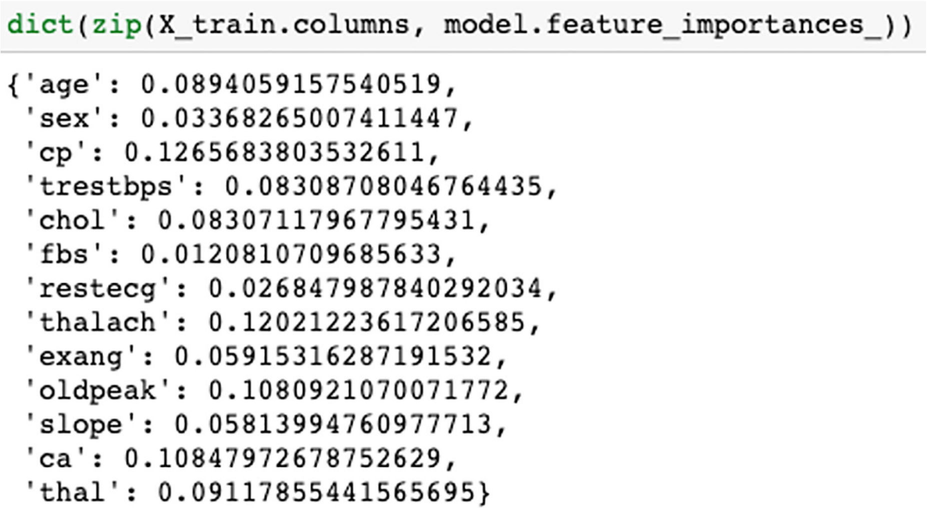
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features=3,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Figure 5.55: *Model which will be used for Explanation*

We will use the basic model of Random Forest, which is trained with the training data with no feature selection.

Feature Importance

We have seen how important "Feature Importance" is when it comes to feature selection. But when we know the order/position in terms of ranking or score of the feature, then we can clearly say which feature is more important than others and which of them are similar.



```
dict(zip(X_train.columns, model.feature_importances_))
```

```
{'age': 0.0894059157540519,
 'sex': 0.03368265007411447,
 'cp': 0.1265683803532611,
 'trestbps': 0.08308708046764435,
 'chol': 0.08307117967795431,
 'fbs': 0.0120810709685633,
 'restecg': 0.026847987840292034,
 'thalach': 0.12021223617206585,
 'exang': 0.05915316287191532,
 'oldpeak': 0.1080921070071772,
 'slope': 0.05813994760977713,
 'ca': 0.10847972678752629,
 'thal': 0.09117855441565695}
```

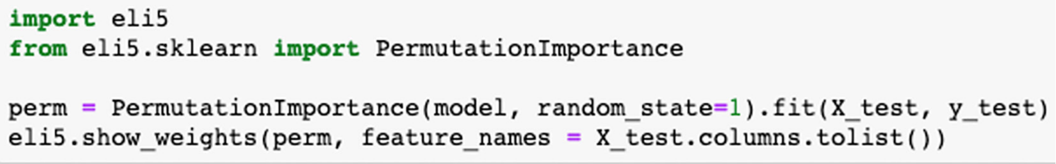
Figure 5.56: *Model Feature Importance*

The above code snippet shows the feature importance for all the features in the model. But this time, we will use the above feature importance for explainable AI and not for feature selection.

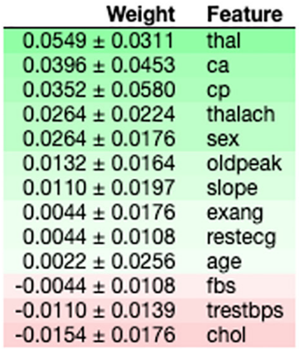
For a given model output, we can clearly say that "cp" and "ca" have higher weightage over other features.

We will use another method using the package "ELI5,"[39] which helps to debug machine learning models and explain their predictions.

We won't go in-depth, but we will show one method, i.e., PermutationImportance. PermutationImportance considers the model as a black box and measures the change in the score when the feature is not present; this is also known as Mean Decrease Accuracy (MDA)[40].



```
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(model, random_state=1).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```



Weight	Feature
0.0549 ± 0.0311	thal
0.0396 ± 0.0453	ca
0.0352 ± 0.0580	cp
0.0264 ± 0.0224	thalach
0.0264 ± 0.0176	sex
0.0132 ± 0.0164	oldpeak
0.0110 ± 0.0197	slope
0.0044 ± 0.0176	exang
0.0044 ± 0.0108	restecg
0.0022 ± 0.0256	age
-0.0044 ± 0.0108	fbs
-0.0110 ± 0.0139	trestbps
-0.0154 ± 0.0176	chol

Figure 5.57: *Permutation Importance*

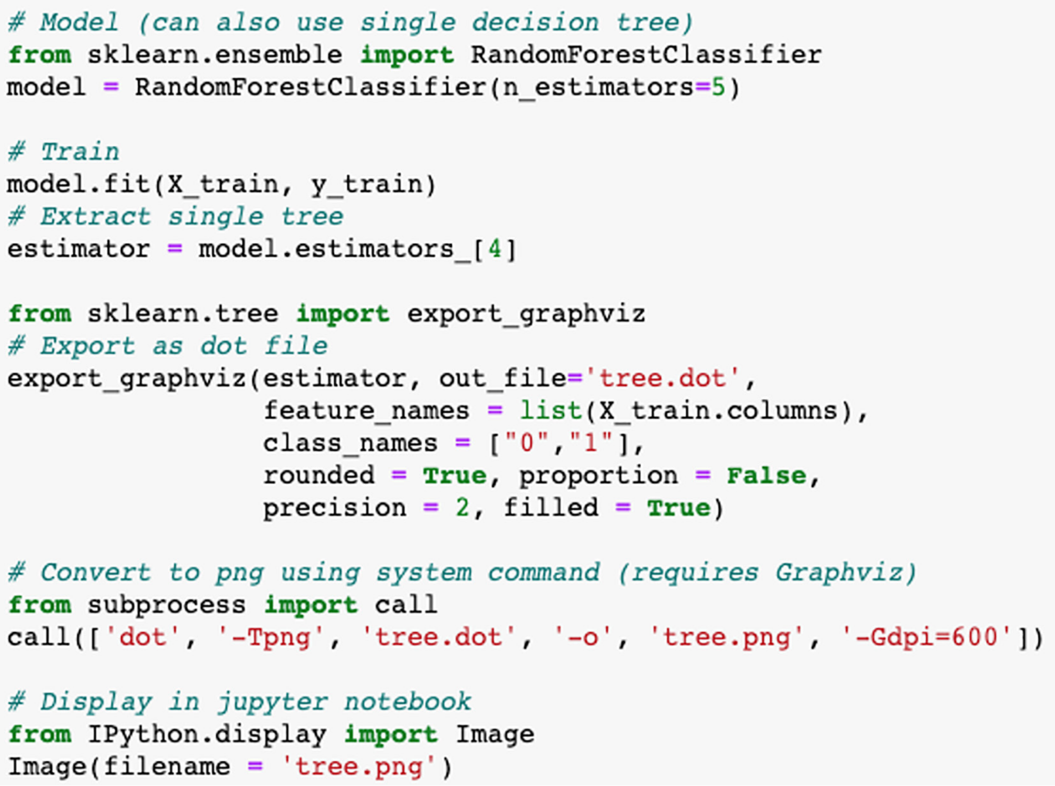
From the above image, we can see it is quite similar to feature importance. "ca" and "cp" are on the top on the feature importance list. This table has no direct link with feature importance, but with both the table and feature importance, we can make some sense about the features.

[39] "Welcome to ELI5's documentation! — ELI5 0.9.0 documentation." https://eli5.readthedocs.io/en/latest/.

[40] "Variable selection using Mean Decrease Accuracy and Mean ...." https://ieeexplore.ieee.org/document/7883053.

Model visualization

Model visualization is another technique to interpret the model's prediction. This technique can be performed for selected machine learning algorithms like tree-based algorithms, linear regression, Naive Bayes, etc.. Using this method on non-linear SVM, k-NN predictions will be tough to analyze. It works best for tree-based methods like Random Forest Classifier, which we are using for model explainability in this section.



```
# Model (can also use single decision tree)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=5)

# Train
model.fit(X_train, y_train)
# Extract single tree
estimator = model.estimators_[4]

from sklearn.tree import export_graphviz
# Export as dot file
export_graphviz(estimator, out_file='tree.dot',
                feature_names = list(X_train.columns),
                class_names = ["0","1"],
                rounded = True, proportion = False,
                precision = 2, filled = True)

# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree.png')
```

Figure 5.58: Visualizing one tree from the forest

We are using a random forest with five classifiers, i.e., there will be five decision trees. We can explain the output of the random forest by visualizing the decision trees. The above code snippet only saves the 5th estimator, i.e., the 5th decision tree. We will only analyze a part of the last tree from the model and see how the decision works.

Let us start by visualizing the entire tree.

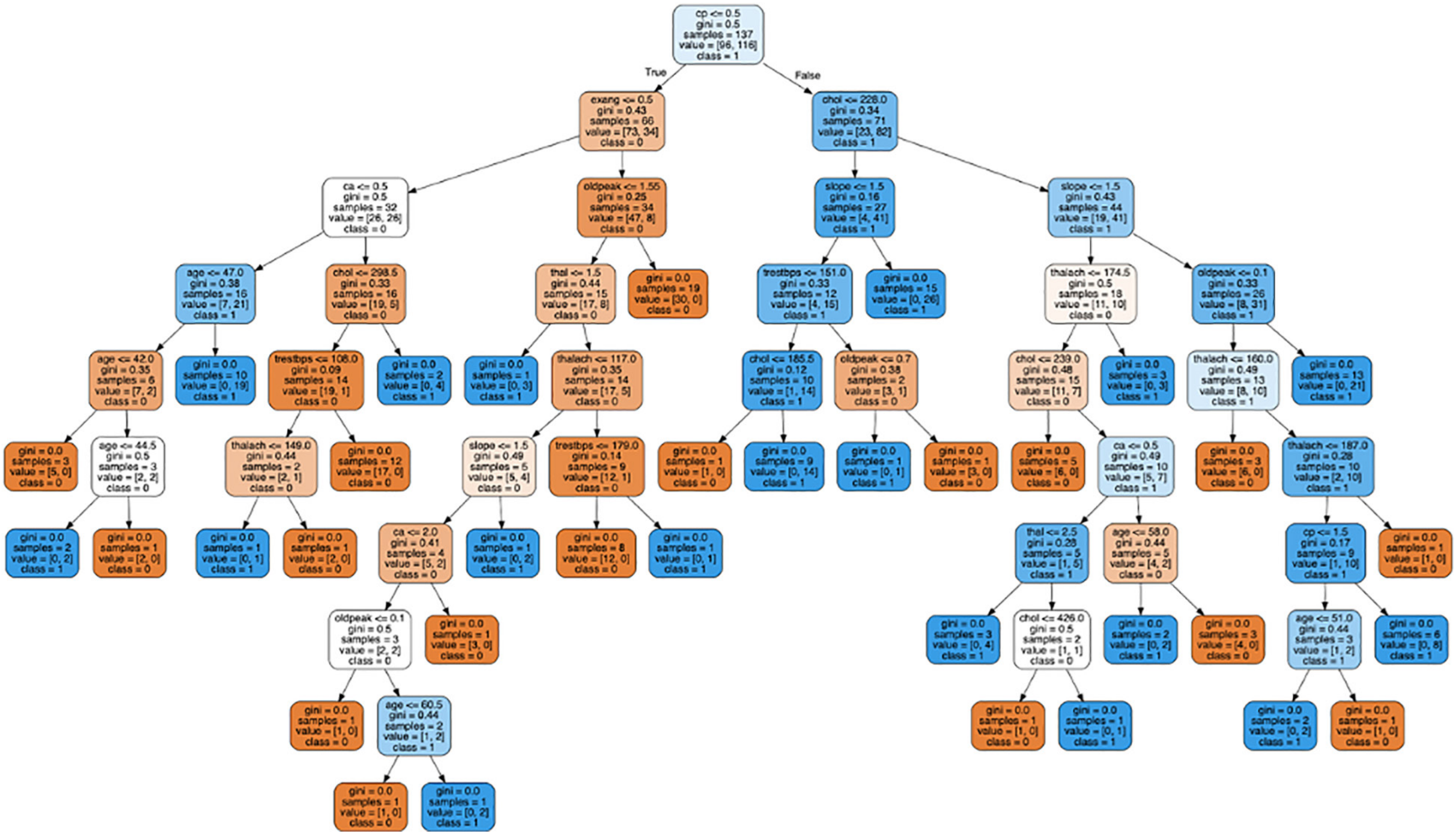

cp <= 0.5 gini = 0.5 samples = 137 value = [96, 116] class = 1
True
False
exang <= 0.5 gini = 0.43 samples = 66 value = [73, 34] class = 0
chol <= 228.0 gini = 0.34 samples = 71 value = [23, 82] class = 1
ca <= 0.5 gini = 0.5 samples = 32 value = [26, 26] class = 0
oldpeak <= 1.55 gini = 0.25 samples = 34 value = [47, 8] class = 0
slope <= 1.5 gini = 0.16 samples = 27 value = [4, 41] class = 1
slope <= 1.5 gini = 0.43 samples = 44 value = [19, 41] class = 1
age <= 47.0 gini = 0.38 samples = 16 value = [7, 21] class = 1
chol <= 298.5 gini = 0.33 samples = 16 value = [19, 5] class = 0
thal <= 1.5 gini = 0.44 samples = 15 value = [17, 8] class = 0
gini = 0.0 samples = 19 value = [30, 0] class = 0
trestbps <= 151.0 gini = 0.33 samples = 12 value = [4, 15] class = 1
gini = 0.0 samples = 15 value = [0, 26] class = 1
thalach <= 174.5 gini = 0.5 samples = 18 value = [11, 10] class = 0
oldpeak <= 0.1 gini = 0.33 samples = 26 value = [8, 31] class = 1
age <= 42.0 gini = 0.35 samples = 6 value = [7, 2] class = 0
gini = 0.0 samples = 10 value = [0, 19] class = 1
trestbps <= 108.0 gini = 0.09 samples = 14 value = [19, 1] class = 0
gini = 0.0 samples = 2 value = [0, 4] class = 1
gini = 0.0 samples = 1 value = [0, 3] class = 1
thalach <= 117.0 gini = 0.35 samples = 14 value = [17, 5] class = 0
chol <= 185.5 gini = 0.12 samples = 10 value = [1, 14] class = 1
oldpeak <= 0.7 gini = 0.38 samples = 2 value = [3, 1] class = 0
chol <= 239.0 gini = 0.48 samples = 15 value = [11, 7] class = 0
gini = 0.0 samples = 3 value = [0, 3] class = 1
thalach <= 160.0 gini = 0.49 samples = 13 value = [8, 10] class = 1
gini = 0.0 samples = 13 value = [0, 21] class = 1
gini = 0.0 samples = 3 value = [5, 0] class = 0
age <= 44.5 gini = 0.5 samples = 3 value = [2, 2] class = 0
thalach <= 149.0 gini = 0.44 samples = 2 value = [2, 1] class = 0
gini = 0.0 samples = 12 value = [17, 0] class = 0
slope <= 1.5 gini = 0.49 samples = 5 value = [5, 4] class = 0
trestbps <= 179.0 gini = 0.14 samples = 9 value = [12, 1] class = 0
gini = 0.0 samples = 1 value = [1, 0] class = 0
gini = 0.0 samples = 9 value = [0, 14] class = 1
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 1 value = [3, 0] class = 0
gini = 0.0 samples = 5 value = [6, 0] class = 0
ca <= 0.5 gini = 0.49 samples = 10 value = [5, 7] class = 1
gini = 0.0 samples = 3 value = [6, 0] class = 0
thalach <= 187.0 gini = 0.28 samples = 10 value = [2, 10] class = 1
gini = 0.0 samples = 2 value = [0, 2] class = 1
gini = 0.0 samples = 1 value = [2, 0] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 1 value = [2, 0] class = 0
ca <= 2.0 gini = 0.41 samples = 4 value = [5, 2] class = 0
gini = 0.0 samples = 1 value = [0, 2] class = 1
gini = 0.0 samples = 8 value = [12, 0] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
thal <= 2.5 gini = 0.28 samples = 5 value = [1, 5] class = 1
age <= 58.0 gini = 0.44 samples = 5 value = [4, 2] class = 0
cp <= 1.5 gini = 0.17 samples = 9 value = [1, 10] class = 1
gini = 0.0 samples = 1 value = [1, 0] class = 0
oldpeak <= 0.1 gini = 0.5 samples = 3 value = [2, 2] class = 0
gini = 0.0 samples = 1 value = [3, 0] class = 0
gini = 0.0 samples = 3 value = [0, 4] class = 1
chol <= 426.0 gini = 0.5 samples = 2 value = [1, 1] class = 0
gini = 0.0 samples = 2 value = [0, 2] class = 1
gini = 0.0 samples = 3 value = [4, 0] class = 0
age <= 51.0 gini = 0.44 samples = 3 value = [1, 2] class = 1
gini = 0.0 samples = 6 value = [0, 8] class = 1
gini = 0.0 samples = 1 value = [1, 0] class = 0
age <= 60.5 gini = 0.44 samples = 2 value = [1, 2] class = 1
gini = 0.0 samples = 1 value = [1, 0] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 2 value = [0, 2] class = 1
gini = 0.0 samples = 1 value = [1, 0] class = 0
gini = 0.0 samples = 1 value = [1, 0] class = 0
gini = 0.0 samples = 1 value = [0, 2] class = 1

Figure 5.59: *One of the decision trees*

Seeing the above image, we have seen a binary tree as the decision for each node is either yes or no. It is tough to see all the nodes and its decision, so we will go ahead and focus on a small area of the graph and explain that itself. Those who are executing the code (which is recommended) can zoom in, to visualize the entire tree for understanding.

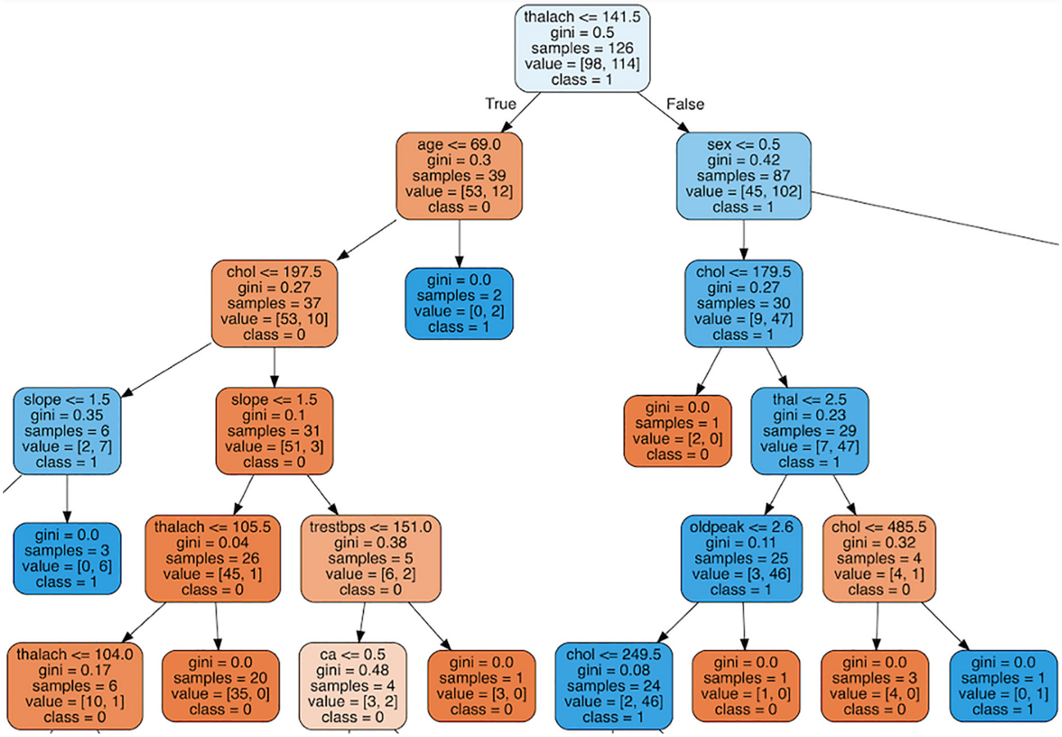


Figure 5.60: *Zoomed-In version of the above decision tree*

Here is the zoomed version of the same binary tree. The blue nodes represent decision favors towards class 1 and, similarly, the orange ones represent class 0. If we carefully take a look at the nodes, then we can see that with some decision, there are some metadata as well. Let's use the first node and explain it.

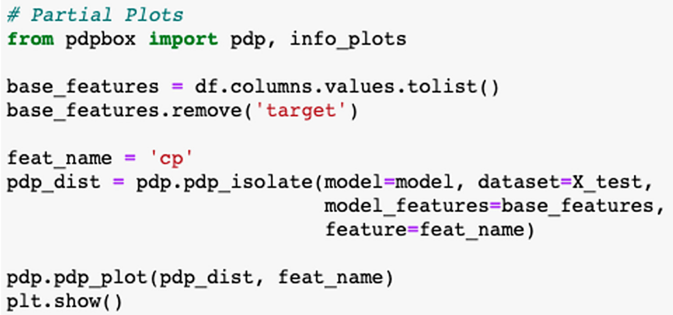
This tree is starting with 126 samples, and the first decision is if "`thalach <= 141.5`". With this decision, the target values are referred, and it is found that there are 39 samples where the target value is true, i.e., heart disease is positive, and the rest of the samples, i.e., 87 samples have no heart disease. For every node, the corresponding Gini value is provided as well.

Now, if someone asks, why the model has predicted a certain output, then we can run through all the trees and can give a definite answer. This is one of the positive sides of using a tree-based algorithm.

Partial Dependence Plot[41]

The **Partial Dependence Plot** (**PDP**) shows the marginal effect for one or two features on a predicted outcome of the model. PDP shows whether a relationship between the target and the given feature is linear, monotonous, or something more complex.

We will just see the plots for one of the top important features and another from bottom of the Feature Importance list.



```
# Partial Plots
from pdpbox import pdp, info_plots

base_features = df.columns.values.tolist()
base_features.remove('target')

feat_name = 'cp'
pdp_dist = pdp.pdp_isolate(model=model, dataset=X_test,
                           model_features=base_features,
                           feature=feat_name)

pdp.pdp_plot(pdp_dist, feat_name)
plt.show()
```

Figure 5.61: *Code for plotting*

The above code analyses the feature '`cp`' and plots a partial dependence plot using PDPbox[42]. Let us take a look at that PDP for the feature.

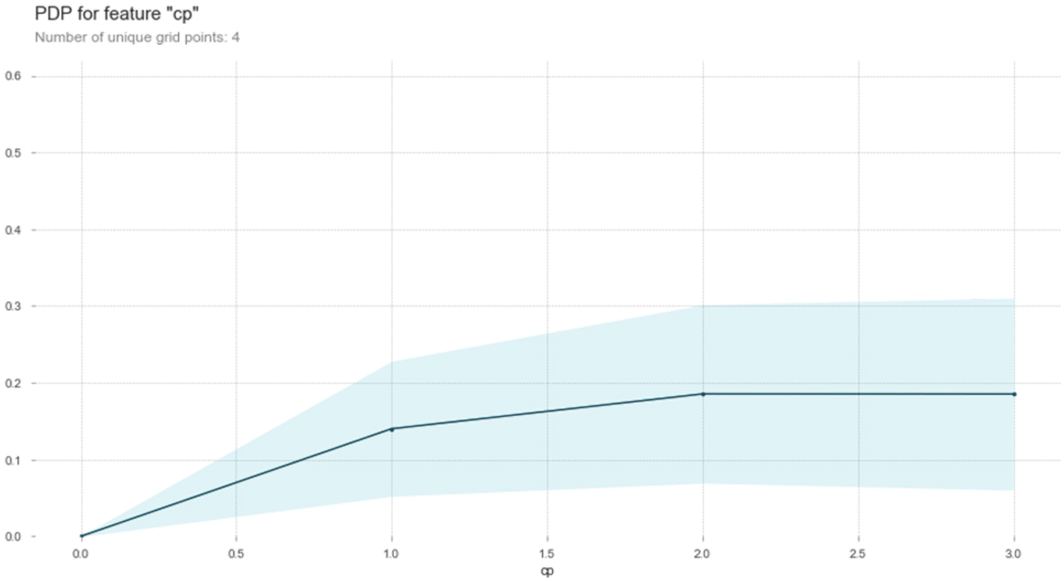


Figure 5.62: *One of the top features*

[41] "5.1 Partial Dependence Plot (PDP) | Interpretable Machine ...." https://christophm.github.io/interpretable-ml-book/pdp.html.

[42] "PDPbox — PDPbox 0.2.0+13.g73c6966.dirty documentation." https://pdpbox.readthedocs.io/en/latest/.

We can see that the feature "cp" is not entirely linear, but it follows a linear pattern over all the data points. Lower values of "cp" have a less positive relationship compared to the higher values. We should also note that "cp" is a categorical variable.

Similarly, let us take a look at another feature that is continuous, i.e., 'chol,' and it has a low feature importance.

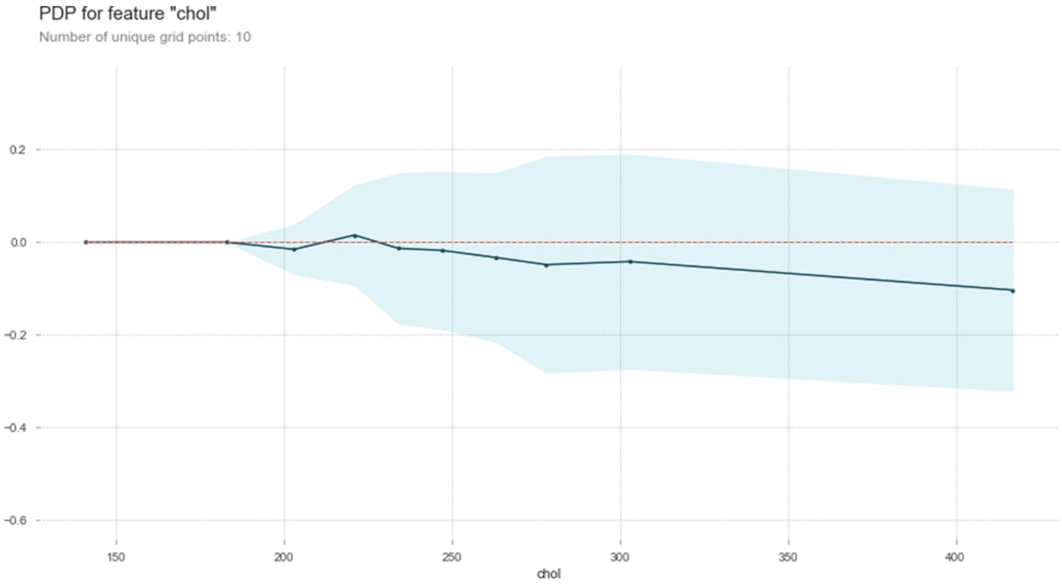


Figure 5.63: *One of the bottom features*

"chol" is a continuous feature that can be understood by the data points from the above plot. We can see the partial dependence is negative on one side, and on the positive side, it did not score a high value.

For more understanding of PDP, it's recommended to visit the footnotes and use it as a start to explore this area of study.

SHAP (SHapley Additive exPlanations)[43]

SHAP is a game-theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classical Shapley values from game theory and their related work[44 45 46].

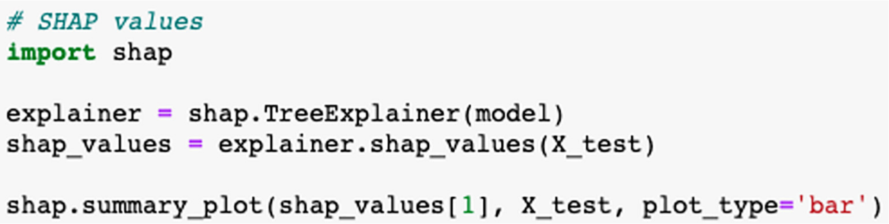
[43] "Explainers — SHAP latest documentation." https://shap.readthedocs.io/en/latest/.

[44] "A Unified Approach to Interpreting Model Predictions - NIPS ...." https://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.

[45] "Explainable machine-learning predictions for the ... - Nature." 10 Oct. 2018, https://www.nature.com/articles/s41551-018-0304-0.

[46] "From local explanations to global understanding with ... - Nature." 17 Jan. 2020, https://www.nature.com/articles/s42256-019-0138-9.

Here we will be using a python package "`shap`"[47] to find the Shapley values and plot some distributions.



```
# SHAP values
import shap

explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

shap.summary_plot(shap_values[1], X_test, plot_type='bar')
```

Figure 5.64: Code Snippet for SHAP value

The above code snippet gives us the Shapley value[48] for all the features. SHAP Values helps to show the impact of each feature.

First, take a look at the output of the code snippet and see the mean of SHAP Values.

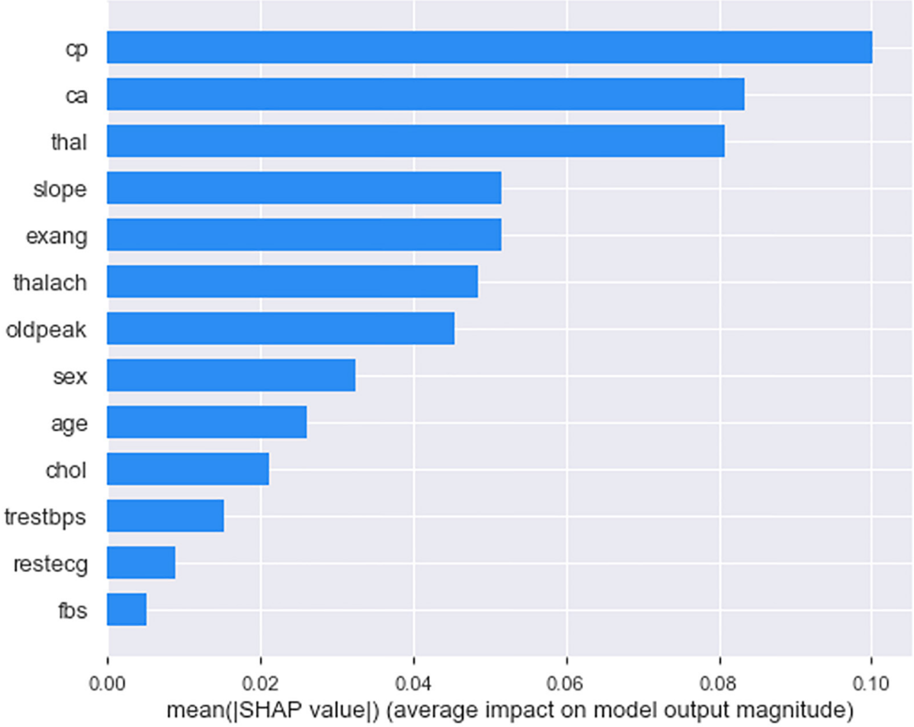


Figure 5.65: *Average Impact using SHAP value*

[47] "slundberg/shap: A game theoretic approach to ... - GitHub." https://github.com/slundberg/shap.

[48] "Shapley value - Wikipedia." https://en.wikipedia.org/wiki/Shapley_value.

The above table of ordered SHAP values is extremely similar to Feature Importance from ELI5. SHAP values interpret the impact of having a certain value for a given feature in comparison to the prediction we would make if that feature took some baseline value.

After knowing the SHAP values, now let us do a summary plot for the same.

```
shap.summary_plot(shap_values[1], X_test)
```

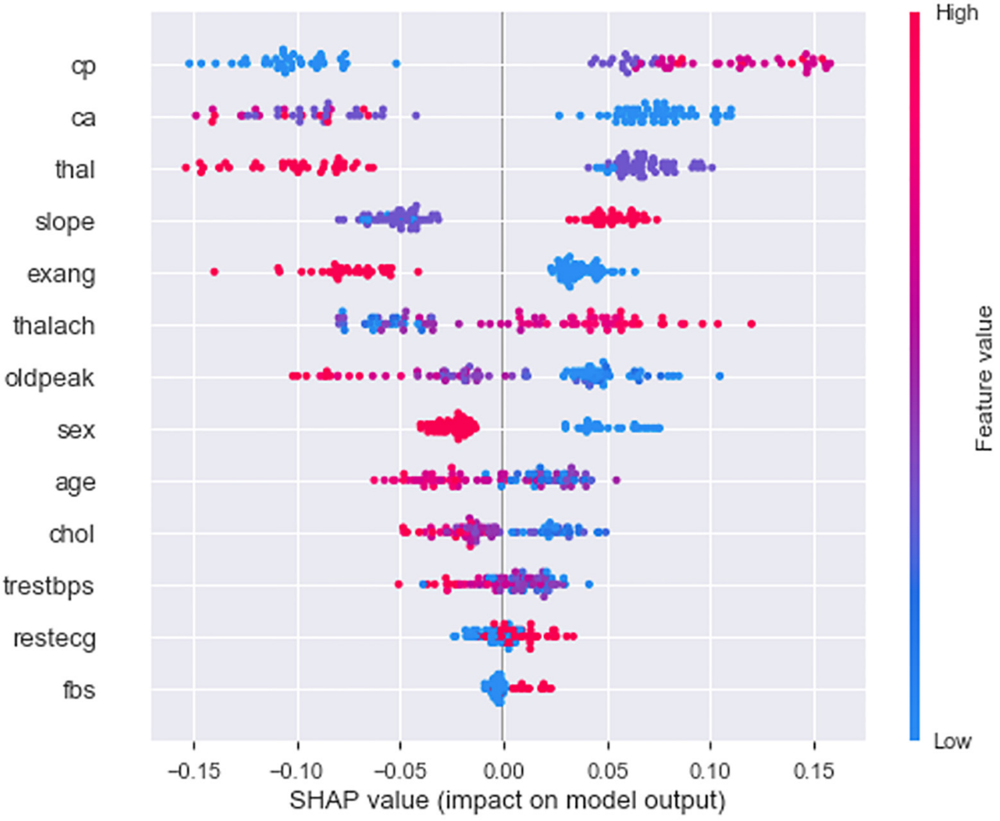
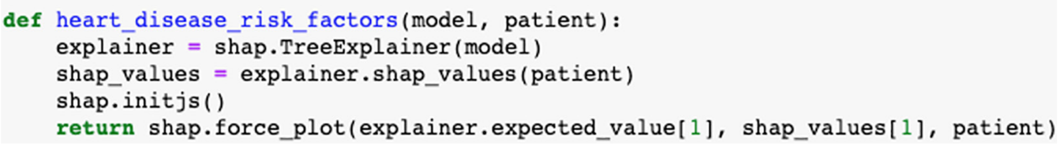


***Figure 5.66:** SHAP value summary*

We can see that the feature "cp" has a clear division, and the lower values are bad (low), which is represented in blue, and the good (high) ones are represented in red.

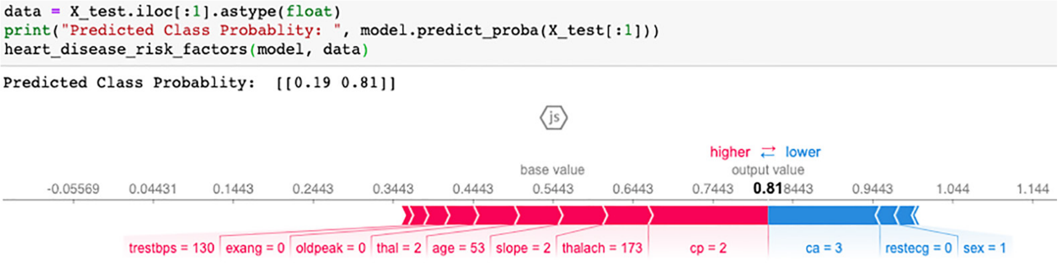
If the features have clear separation, then it has a good feature for prediction. Let's write a code snippet to perform a forced plot and see which features influence the prediction with its influence score.



```
def heart_disease_risk_factors(model, patient):
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(patient)
    shap.initjs()
    return shap.force_plot(explainer.expected_value[1], shap_values[1], patient)
```

Figure 5.67: *Function to generate force plot*

That is, the SHAP values of all features sum up to explain why my prediction was different from the baseline. This allows us to decompose a prediction in a graph like this:



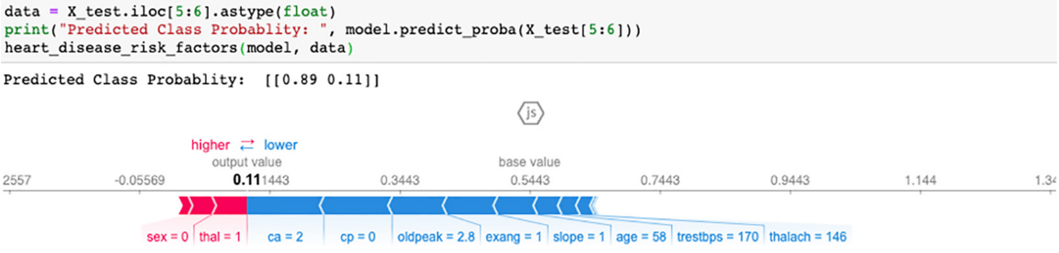
```
data = X_test.iloc[:1].astype(float)
print("Predicted Class Probablity: ", model.predict_proba(X_test[:1]))
heart_disease_risk_factors(model, data)
```

```
Predicted Class Probablity:  [[0.19 0.81]]
```

Figure 5.68: *Positive Class*

When an input is given to a model, it will have a probabilistic output. Now, if we want to explain which features had its contribution to this probabilistic score, then this is one of the best methods to implement. We can see from the above image that red indicated higher value, i.e., it has a positive effect on the prediction. For the given data, we can see "`cp`" has a hugely positive effect, and "`ca`" has a huge negative effect. With this, we can use the data dictionary and say create a reason for the output "`0.81.`"

Let us see one more example which will yield a negative prediction.



```
data = X_test.iloc[5:6].astype(float)
print("Predicted Class Probablity: ", model.predict_proba(X_test[5:6]))
heart_disease_risk_factors(model, data)
```

```
Predicted Class Probablity:  [[0.89 0.11]]
```

Figure 5.69: *Negative Class*

Similarly, here "`ca`" and "`cp`" hurts the prediction, and only "`thal`" and "`sex`" has a positive affect over the prediction "`0.11`". These are some of the methods which are used to explain the prediction of an ML model.

With these examples, we will end this explainable AI section. If someone is interested in diving deep in this topic, they can go through all the footnotes and search the relevant topics over search engines to get further insights.

Further reading

- Correlation
    - o Kendall's τ
    - o Phik (ϕk)
    - o Cramér's V (ϕc)
    - o Recoded
- Pipeline
- XGBoost
- LightGBM
- Hyperparameter Optimization
- Naive Bayes' Classifiers
    - o Multinomial Naive Bayes
    - o Bernoulli Naive Bayes
    - o Complement Naive Bayes
    - o Categorical Naive Bayes
- SHAP
- PDP
- Individual Conditional Expectation (ICE)
- Accumulated Local Effect (ALE)
- LIME
- Ethics of AI

Conclusion

This chapter, we have covered a multitude of topics starting from Ensemble Learning Method, Grid Searching to Explainable AI. Ensemble Learning is one of the most popular techniques which is widely used in Machine Learning. Learning ML has no end to it, but it is advised to implement concepts and algorithms to get insight from them.

After going through all the chapters, one can take up real-time data and work on it. Staring from data fetching, data scrubbing, data cleaning to making Machine Learning modeling. Everyone should remember that working with data will take more than 60% to 70% of the project's bandwidth, and the rest is given to modeling, model evaluating, and model tuning. In addition to this, we have seen explainable AI and talked a bit about the ethics of AI.

It is advised to the readers that, they should go through all the footnotes and recommendations for better understanding. I have used Jupyter Notebook for all the chapters, and it is a good practice to write down all the code and execute then, but all the code is available on GitHub:

https://github.com/bpbpublications/Machine-Learning-Cookbook-with-Python.

With all these concepts covered, we will end this chapter and the book.