Statistics For MACHINE Learning

Implement Statistical Methods Used in Machine Learning Using Python

HIMANSHU SINGH

bpb

Statistics for Machine Learning

Implement Statistical Methods used in Machine Learning using Python

Himanshu Singh

www.bpbonline.com

FIRST EDITION 2021

Copyright © BPB Publications, India

ISBN: 978-93-88511-97-1

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

My Dad

Who never stopped believing in me,
even though he never expressed.

About the Author

Himanshu Singh is currently an AI technology lead and senior NLP developer at Legato Health Technologies (An Anthem Company). Himanshu has a total of 7 years of experience, mostly in the domain of Natural Language Processing. He has written five books in the machine learning domain and is a guest faculty for machine learning and data science. Himanshu is an avid blogger and loves to read and write fiction short stories in his free time.

About the Reviewer

- **Aravind Kota** is currently working as a data scientist. He has around 3+ years of experience in the field of data science, with specialization in image and text analytics and statistical operations with Python coding. He shares his knowledge in this field through blogs, and it's important for readers to understand these concepts for further experiments.

Acknowledgments

First and foremost, I would like to thank my team. It is because of them that I got the opportunities to explore different problem statements, which has enabled me to write this book. I would especially like to thank Aravind, Bhavani, and Yunis sir.

I would also like to thank my students. Because of them, I came across all the doubts that they faced while understanding statistics. This, in turn, gave me ideas to approach this book in such a way that it clears the doubts of its readers.

Last but not least, I would like to thank my wife, Shikha. She has been a constant source of motivation for me, and without her, I would have never been able to finish the book.

Preface

This book can be considered a preliminary requirement before starting the machine learning journey in detail. One must understand that machine learning, in itself, is dependent on the concepts of statistics and mathematics. Statistical concepts are used in various areas of machine learning, like data exploration, finding the efficiency and efficacy of variables as well as models, and making visualizations. This book is designed in such a way that a reader can go through all the required concepts of statistics and then jump to understanding machine learning algorithms.

This book can be said to be having three sections. The first section starts with the basics of statistics. It covers preliminary concepts like mean, median, mode, and such and moves on to the concepts related to probability, random variables, and the like. The second section covers the complex parts of statistics, including advanced concepts like statistical tests, parametric and non-parametric tests and their applications in Python. Finally, the last section talks more about how to use various data science packages in Python and introduces readers to machine learning and some of its algorithms.

Downloading the coloured images:

Please follow the link to download the
Coloured Images of the book:

https://rebrand.ly/vqukb

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, 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.

Table of Contents

Chapter 1

Introduction to Statistics

This chapter focuses on the various parameters related to statistics. It will guide you through all the ingredients required for the statistical recipes.

Structure

- Population and sample
- Introduction to random variables
- Other variables
- Introduction to descriptive statistics
- Visualizations

Objectives

This chapter aims to provide readers with the base for statistics and statistical Python.

Population and Sample

Suppose I want to start a new service- or product-based company. The company type and the way it is operated may differ, but your company will fail if it is offering something that no one needs.

But how to know whether your offering is correct? Will people like it? Is there a need for it? There is only one answer to solving these doubts—market research.

Whenever a company launches a new product, it carries out market research to determine the product feasibility, the areas in which the product has the highest demand, the demographics that the company should target, and such. Without research, it's like shooting an arrow into the dark.

Research is not limited to business, and you can find its application in all walks of life. From politics to sports and even a movie launch are nothing without research.

Research begins with determining the target audience or target market. Suppose we are making a cosmetic product, our target market can be females over 15 years of age who live in metropolitan cities. Everyone who meets the above criteria, or any criterion that the research team makes, is considered part of our **p**opulation. A team starts its research only after they have carefully drafted the criteria to be met by the population. Once this is done, they come up with samples.

There are various reasons why we must draw samples out of our entire population. We will look at all these reasons in *Chapter 6*, but the most important reason is the inability to cover the entire population. Although we know our target population, it is next to impossible to reach each person and interview them. So, different approaches are used to draw samples of the population and apply the research. Given here is a list of the approaches to draw samples (we'll discuss all of them in detail in *Chapter 6*).

Probabilistic sampling:

- Random sampling
- Sequential random sampling
- Cluster sampling
- Stratified sampling

Non-probabilistic sampling:

- Judgment sampling
- Convenience sampling
- Snowball sampling
- Quota sampling

Introduction to Random Variables

How we do carry out the research?

We define the questions related to the research and the instruments to measure the answers. The questions can be open-ended or closed-ended. The former are ones in which specific answers do not limit us, and we can write whatever we feel like. For example:

What do you feel about the current election scenario?

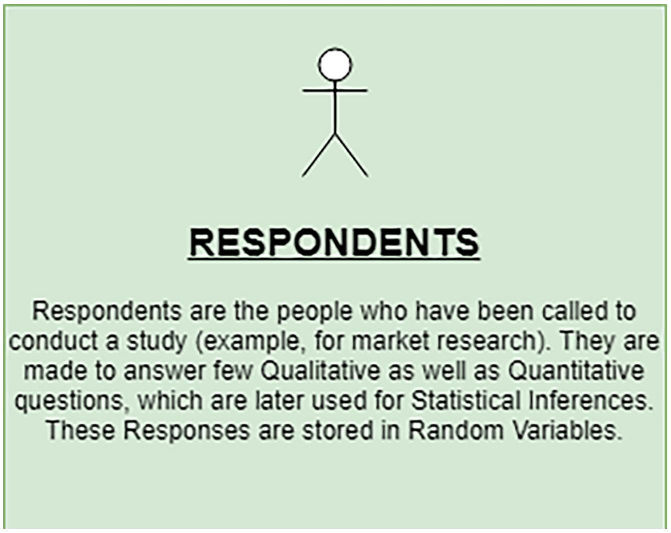
Now, the answer to this question will differ drastically for different people. Some may give positive answers, while others may give negative ones, and the language used will always differ.

When it comes to closed-ended questions, the response is limited. For example:

What is your age?

- 10-20
- 20-30
- 30-40
- 40-50
- 50+

In the preceding example, the respondent has a limited number of options to choose from. They cannot give any other input.



RESPONDENTS

Respondents are the people who have been called to conduct a study (example, for market research). They are made to answer few Qualitative as well as Quantitative questions, which are later used for Statistical Inferences. These Responses are stored in Random Variables.

Figure 1.1

Now, once a respondent has submitted their answers, we store them. This storage is called the **Random Variables**. Each Random Variable stores the answers to one question. Generally, it is used only for the Closed Ended Numerical Questions.

Random Variables are of two types based upon the questions: **discrete** and **continuous**. We will discuss Random Variable is detail in *Chapter 6*, but let's take a look at both the types.

Discrete Random Variables

Discrete Random Variables store the whole number type of values, which means all the numbers with decimals can't be stored here. For example:

We have taken four parameters in the following table. You can see that all the values are in the whole number format and cannot be mentioned in decimal format. All these parameters come under Discrete Random Variables.

Age	Siblings	No of Countries Travelled	Prizes Won
23	3	0	0
44	2	3	3
35	4	2	1
63	5	8	5
28	1	1	1
39	0	2	1

Table 1.1

Continuous Random Variables

Continuous Random Variables store the float type of values, which means all decimal numbers can be stored here. For example:

The following example has all the values in decimals. If we want, we can have the same values in two decimal places, three decimal places, and so on. It can go to infinity, as in the measurement, we cannot have any distinct value. So, all these parameters are Continuous Random Variables.

Height	Weight	CGPA	CAT Percentile
5.5	55.6	7.2	99.9
5.2	58.5	6.5	99.7
5.8	75.3	8.9	93.2

Contd…

Height	Weight	CGPA	CAT Percentile
6.3	92.6	9.2	43.2
6.1	73.3	9.8	75.5
5.11	78.6	5.4	84.9

Table 1.2

Other variables

Now that we have seen the types of Random Variables, let's look at a few others:

Numerical variables

These are the variables that store your numerical data, like age, height, and weight. They are subdivided into two types:

1. Interval variables
2. Ratio variables

Interval Variables are those that can hold any numerical data, given a range. For example, the following table has the ranges of two parameters: Temperature and Number of Gold medals won in the Olympics. Both have ranges within which various counts will occur.

Remember: Zero has a meaning in Interval Variables. As you can see in the following table, the temperature can be 0 degrees Celsius, and the number of Gold medals won can be 0 as well.

Temperature	No of Gold Medals Won
-30 to -20	0-5
-20 to -10	5-10
-10 to 0	10-15
0 to 10	15-20
10 to 20	20-25
20 to 30	25-30

Table 1.3

Ratio Variables are exactly like Interval Variables, with just one difference: there is no meaning of zero in Ratio Variables. For example, when we are talking about age, speed, or a running car, zero doesn't have any value. You can see that in the following table:

Age	Running Car Speed
1-10	1-30
10-20	30-60
20-30	60-90
30-40	90-120
40-50	120-150
50+	150+

Table 1.4

Categorical Variables

When we have categories, instead of numeric, in our data, we use Categorical Variables for its storage. They can be of the following types:

1. Nominal variables
2. Ordinal variables
3. Dichotomous or binary variables

When we have categories that cannot be rank based on each other, those variables are called **Nominal variables**. For example, when talking about gender, we can't say that male is greater than female, or vice versa. Similarly, the answers or directions you give are all examples of Nominal Variables, as given here:

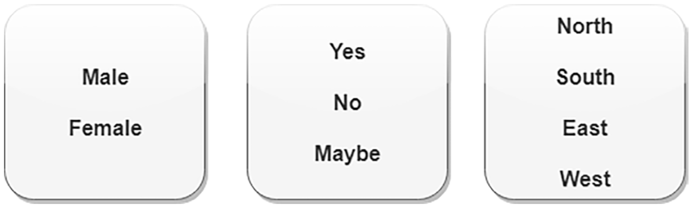


Figure 1.2

When we have categories that can be ranked, the variables can be termed as **Ordinal Variables**. For example, in all the three examples given below, we can rank the options either in ascending or descending order based on the requirement.

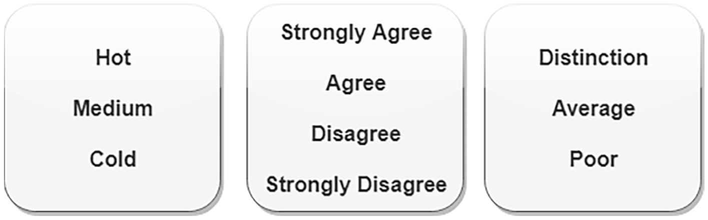


Figure 1.3

Dichotomous or Binary Variables can be considered a subset of Nominal or Ordinal. When we have only two categories that can either be ranked or not, they can be termed as **Dichotomous Variables**. For example, Male & Female, yes and no, and such are dichotomous variables.

Male
Female
Yes
No
Agree
Disagree

Figure 1.4

Introduction to Descriptive Statistics

We will be looking at descriptive statistics in detail in *Chapter 2, but* let's get introduced to it in this chapter.

When we have huge data in front of us, and we want to summarize it based on the center of the data, we use Descriptive Statistics. The first step for the analysis is to determine the center. Now, here are the most popular ways of determining the center:

1. Mean
2. Median
3. Mode

When we talk about the center of balance, which is the where the weights on either side are equal, we consider the mean of the dataset. For example, in the following diagram, the mean is at the exact center when the weights are equal on both sides, but the mean is near the corner when the weights are not equal.

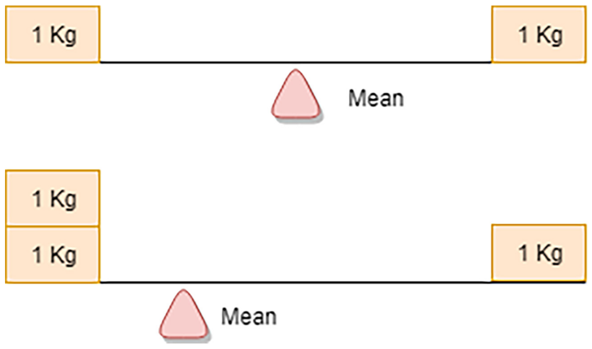


Figure 1.5

When we want to find the exact center, or that the amount of data on the right-hand side is equal to the amount of data on the left-hand side, we consider the median of the dataset. For example, in the following diagram, we can see that heights are in ascending order. Based on the order, 180cm comes in the center, so it is our median.

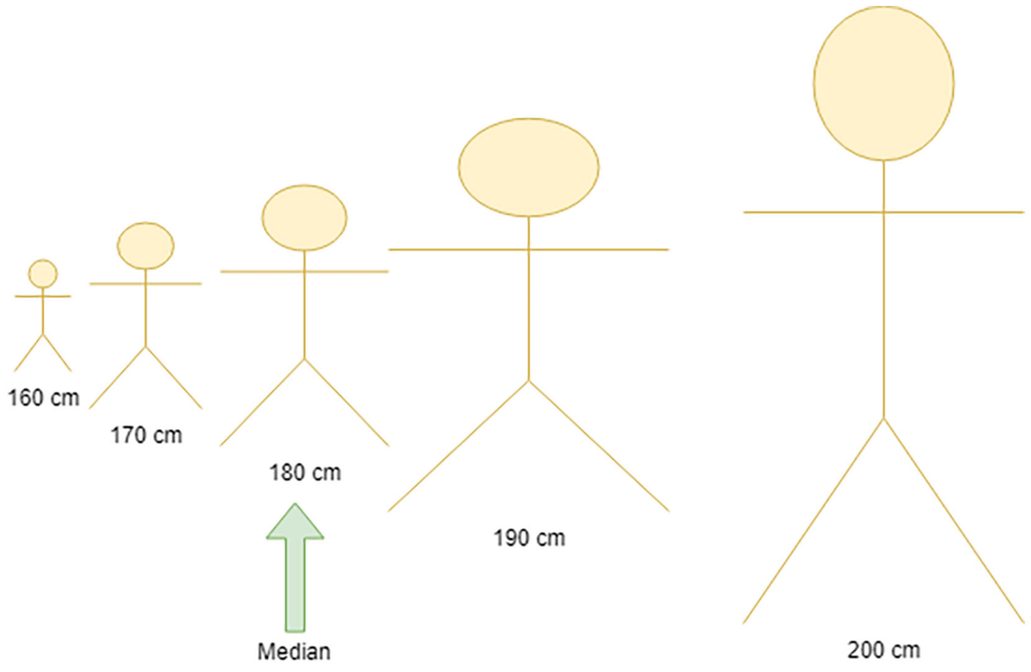


Figure 1.6

Lastly, when we only want to know the value that appears the most number of times, we use the mode of the dataset. For example:

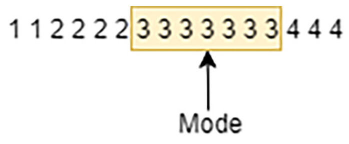


Figure 1.7

Once we know the center of the data, we can draw inferences by determining the variance and standard deviation. Both these terms summarize the distance of the data from the central point. That is, if these terms have a high value, it means that most values in the dataset are distant from the mean, and so the data is heavily dispersed. However, small values mean that the data is closer to the mean, and so the dispersion is less. For instance, in the following diagram, you can see that the data is much more distant from the mean in figure (a) as compared to figure (b). So, we can say that figure (a) has more Variance or Standard Deviation as compared to figure (b). We will talk about Variance and Standard Deviation in further detail in the next chapter.

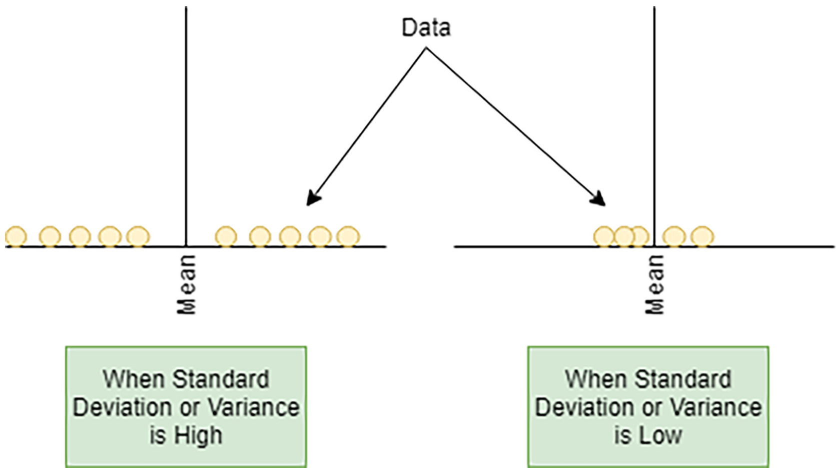
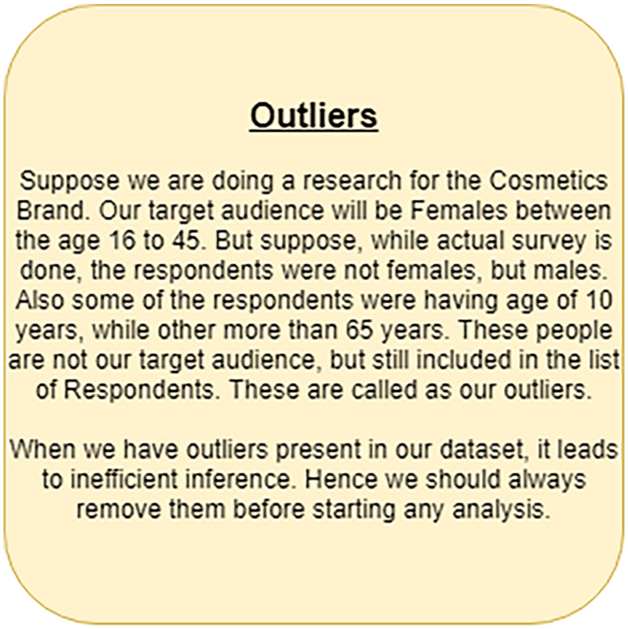

Data
Mean
Mean
When Standard Deviation or Variance is High
When Standard Deviation or Variance is Low

Figure 1.8

Lastly, Quartile and Inter-Quartile ranges help us find the outliers in our dataset.



Outliers

Suppose we are doing a research for the Cosmetics Brand. Our target audience will be Females between the age 16 to 45. But suppose, while actual survey is done, the respondents were not females, but males. Also some of the respondents were having age of 10 years, while other more than 65 years. These people are not our target audience, but still included in the list of Respondents. These are called as our outliers.

When we have outliers present in our dataset, it leads to inefficient inference. Hence we should always remove them before starting any analysis.

Figure 1.9

Quartile helps divide the dataset into four equal halves. The partition is given here:

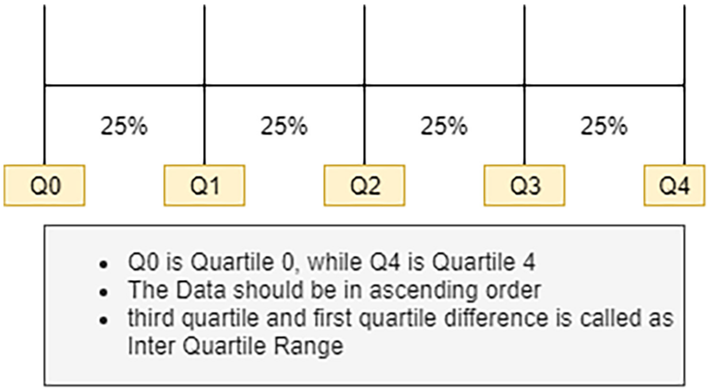

25%
25%
25%
25%
Q0
Q1
Q2
Q3
Q4
• Q0 is Quartile 0, while Q4 is Quartile 4
• The Data should be in ascending order
• third quartile and first quartile difference is called as Inter Quartile Range

Figure 1.10

With the help of Quartiles, we find the value of Inter-Quartile Ranges, which help us determine the outliers with the help of Boxplot. Given in the next section is a diagram of a Boxplot and Outliers. We will look at the concepts in detail in the next chapter.

Visualizations

Once we have the dataset and have determined the descriptive statistics, it's better to visualize everything with the help of graphs. We will be discussing all the following graphs in the next chapter. For now, let's look at how these graphs look and what their usage is:

1. **Vertical Bar Charts**
   Used for comparing discrete data.

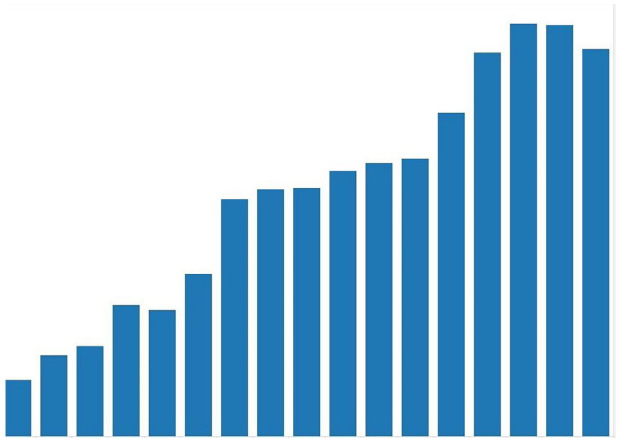


Figure 1.11

2. Stacked Bar Charts

Used for comparing two or more groups relatively.

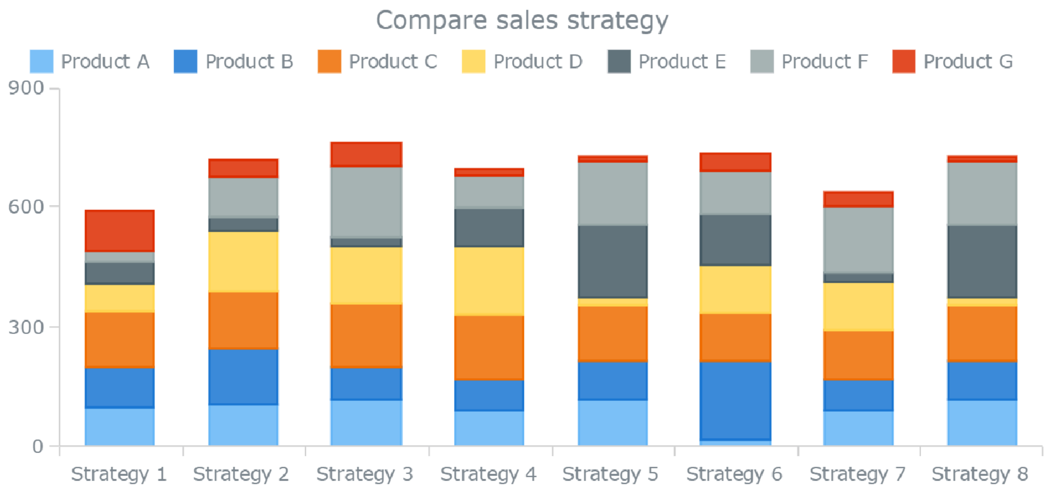


Figure 1.12

3. Histogram

Used for visualizing the frequency of a variable.

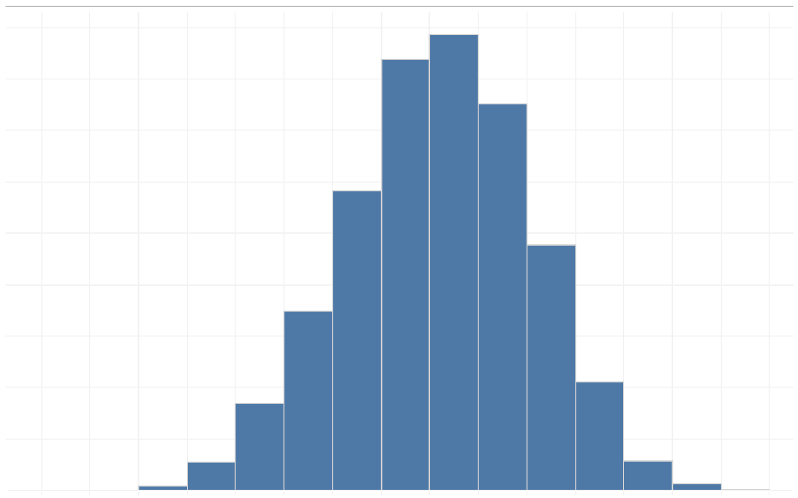


Figure 1.13

4. **Horizontal Bar**
   Similar to vertical but used when the number of categories is high.

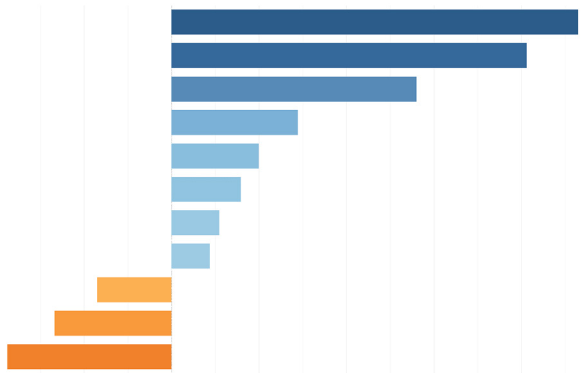


Figure 1.14

5. **Pie Charts**
   Used to visualize the proportion of data.

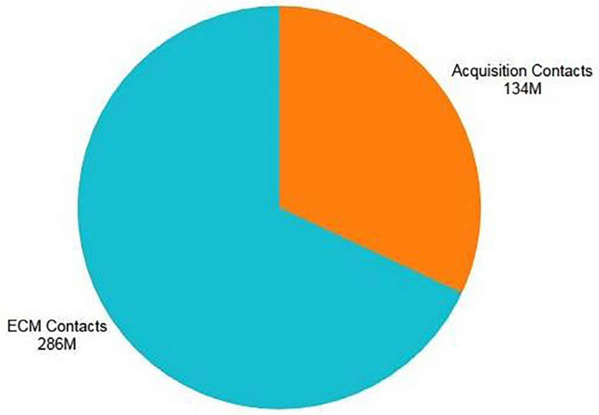


Figure 1.15

6. **Line Charts**
   Most commonly used for projections based on a time frame.

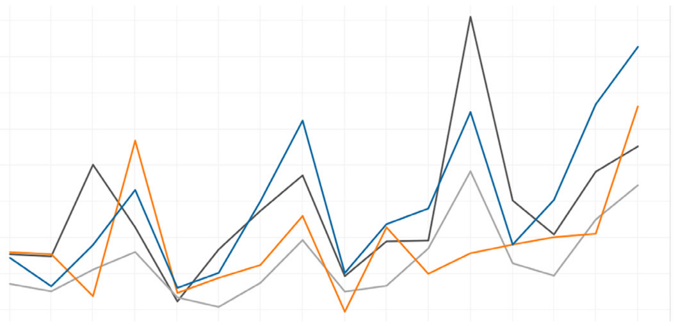


Figure 1.16

7. **Area Charts**
   Line charts also used to showcase the area within are area charts.

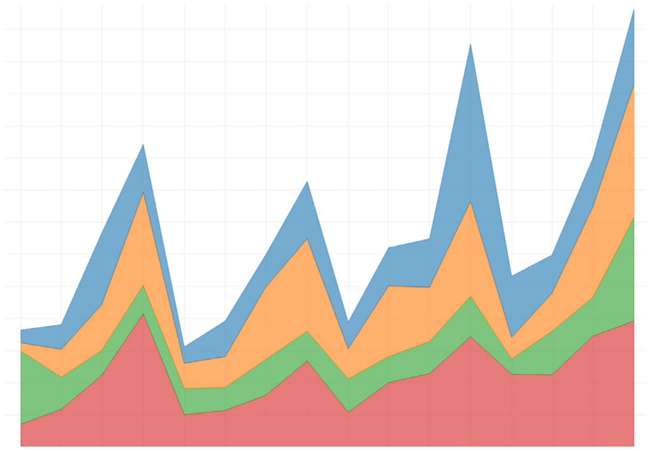


Figure 1.17

8. **Scatter Plot**
   Used to show the relationship between two variables.

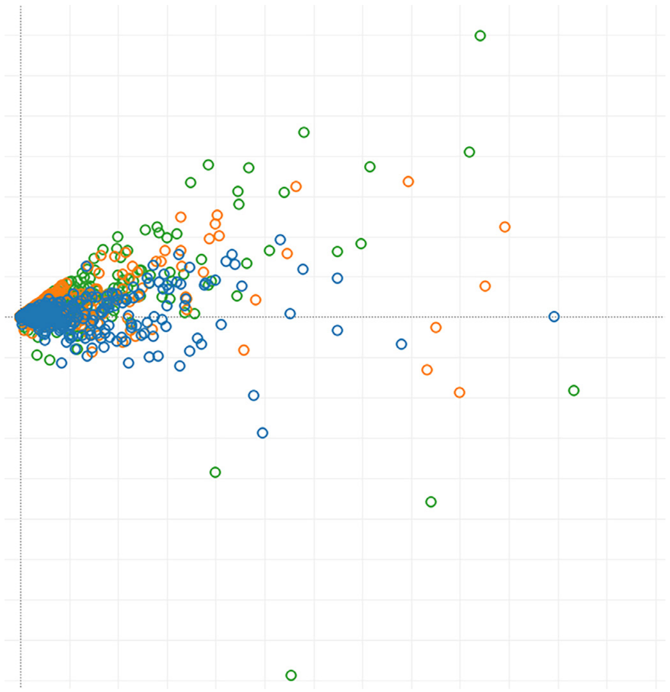


Figure 1.18

9. **Bubble Chart**

   The same as the scatter plot, but the bubble size is dependent upon the third variable.

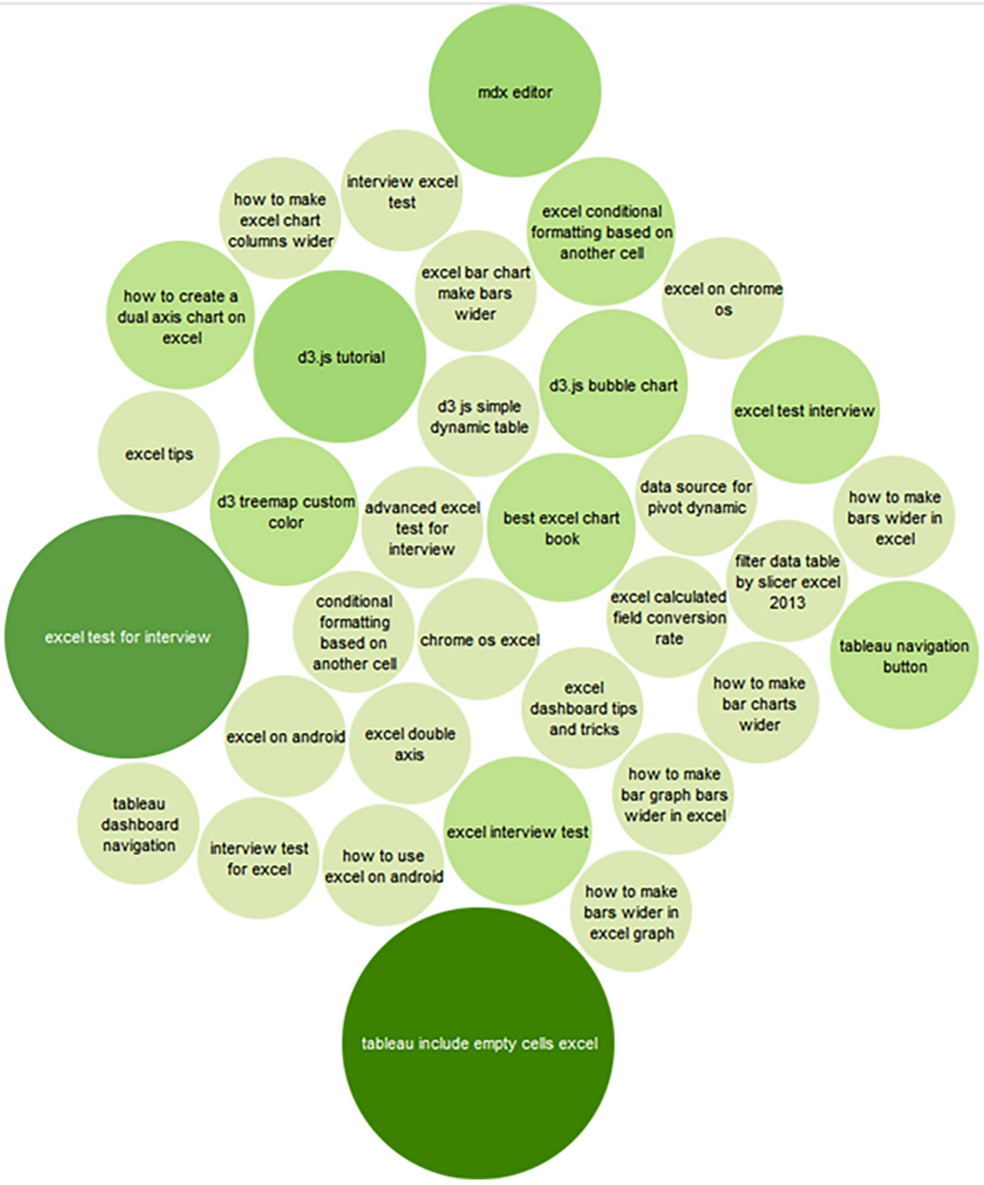


Figure 1.19

10. **Funnel Chart**

    Visualizing the different stages of a process.

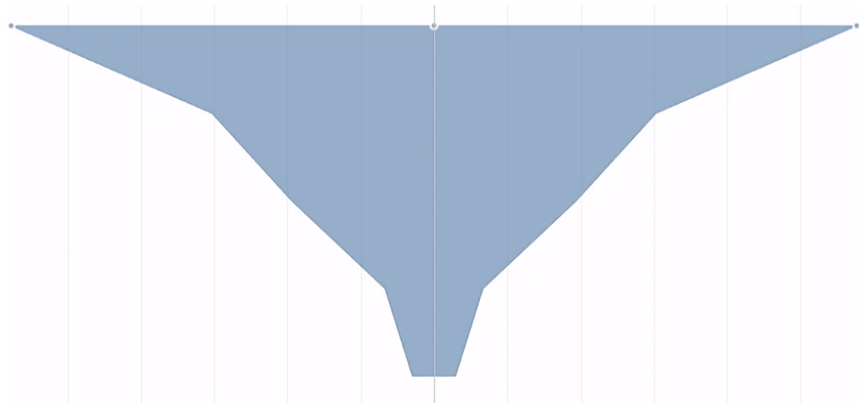


Figure 1.20

11. **Bullet Chart**

    Used to visualize performance relative to a goal.

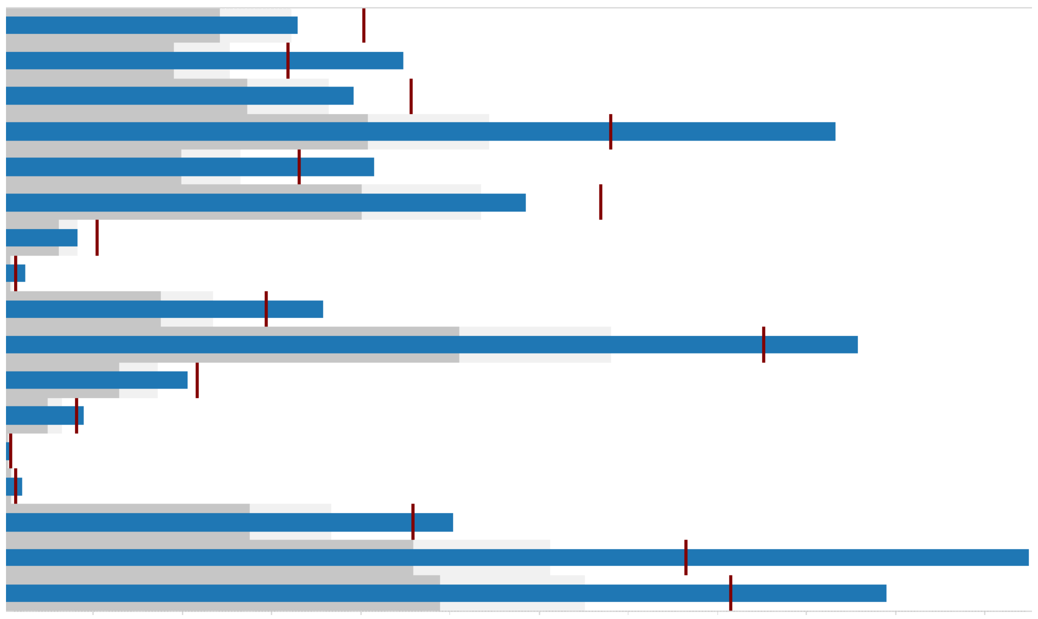
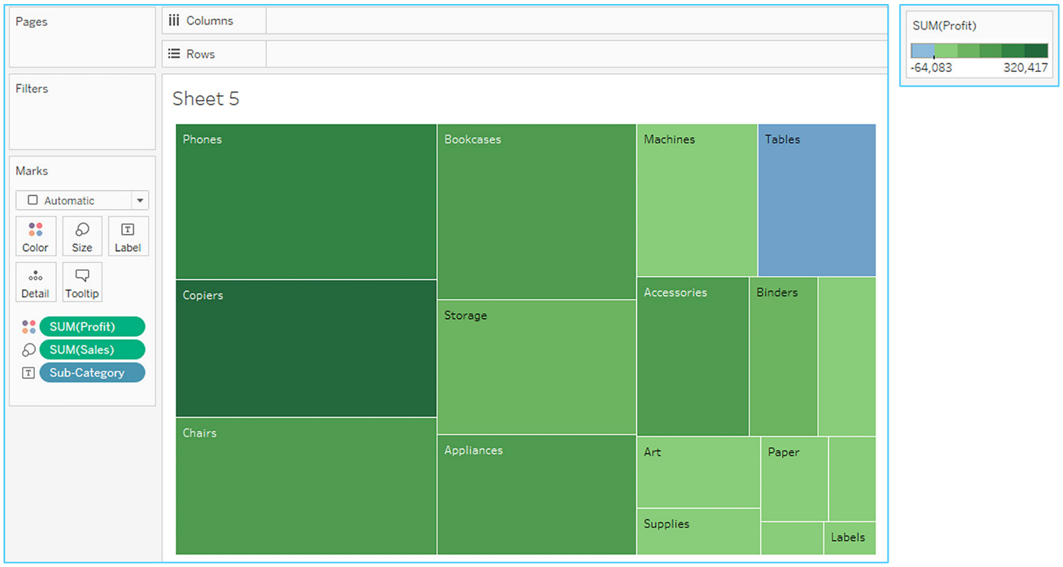


Figure 1.21

12. Heat Map

When we depict ratings with different colors, we use heat maps.



Tree Map

Figure 1.22

13. Box Plot

Used for determining the outliers.

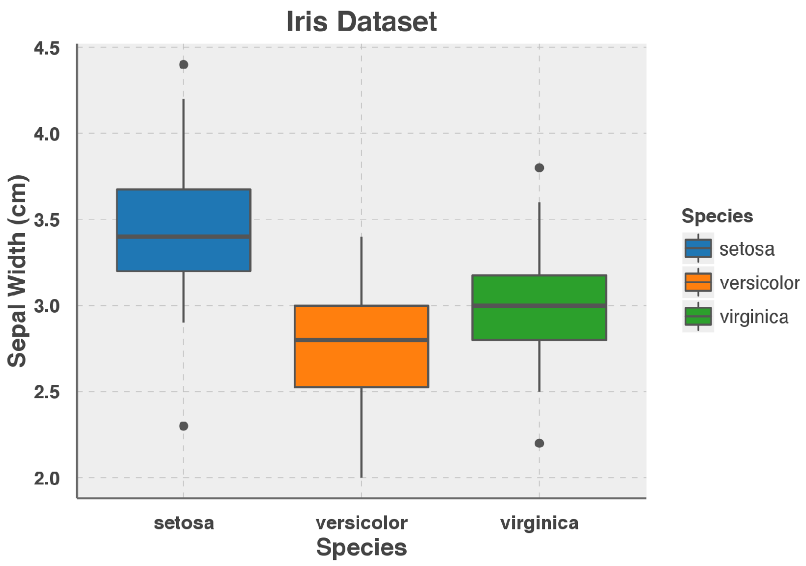


Figure 1.23

Conclusion

In this chapter, we have seen the basics of statistics. The next chapters onward, we will be diving deep into the statistical ocean and will discuss every aspect of this field that will take care of everything else dependent on statistics.

In *Chapter 2,* we will look at Descriptive or Inferential Statistics in detail. *Chapter 3* will cover Random Variables in depth. We have just been given an overview of random variables, but this chapter will cover it in greater depth.

Chapter 2
Descriptive Statistics

Statistics is the collection, presentation, analysis, and interpretation of data, and the techniques and methods for data interpretation will be covered in this chapter. This chapter is the first step to understanding the concepts related to advanced statistics.

Structure

- Measures of central tendency
- Measures of dispersion
- Strength of the relationship between variables

Objective

This chapter aims to provide the base of statistics and statistical Python to readers. It will guide them through all the ingredients for the required statistical recipes.

Measures of Central Tendency

As mentioned in the previous chapter, we summarize data using the measure of central tendency. We already discussed a gist of all the methods, but we will discuss them in detail here. We'll look at the following measures:

- Mean (Arithmetic mean)
- Median
- Mode

Mean (Arithmetic)

The **mean (or average)** is the most popular and well-known measure and can be used with both discrete and continuous data. We already discussed that the mean is the center of balance, but, mathematically speaking, mean is equal to the sum of all the values in the dataset divided by the number of values in the dataset:

$$\bar{x} = (x_1 + x_2 + x_3 + \ldots + x_n) / n$$

The preceding formula can be summarized as:

$$\bar{x} = \sum x/n$$

Mean is one of the most common measures of central tendency. It summarizes the data and plays a very important role when we talk about predictions, as we will see in linear regression. It helps minimize the prediction error, which is when the predicted value differs from the original value.

Let's take a look at the following example to understand the concept of mean:

The average marks of a group of friends are given here:

Friend	Marks
Friend 1	49
Friend 2	32
Friend 3	41
Friend 4	28
Friend 5	20

Table 2.1

The mean can be calculated as follows:

$$\textbf{Mean} = (49 + 32 + 41 + 28 + 20)/5 = 34$$

So, we can say that on average, the group got 34 marks. This value can also be used to make further inferences.

One of the disadvantages of mean is that it is easily influenced by an outlier, which is a value that "lies outside" (is much smaller or larger than) most of the other values in a dataset.

For example, in the scores 25, 29, 3, 32, 85, 33, 27, 28, both 3 and 85 are "outliers". We can visualize the outliers by using boxplot:

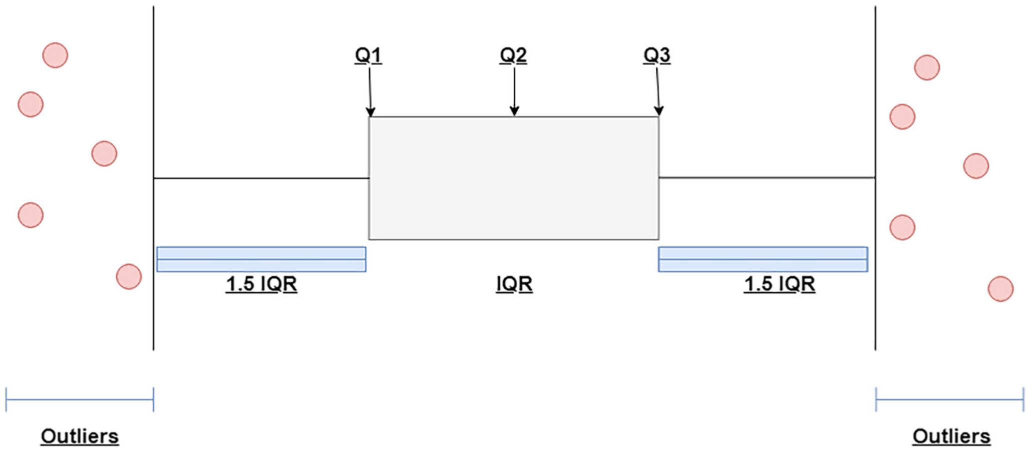


Figure 2.1

Median

As discussed in the previous chapter, the **Median** is the exact center of the data. Let's see how to calculate the median.

Given here are a list of numbers depicting the marks of the students in a subject:

3, 13, 7, 5, 21, 23, 39, 23, 40, 23, 14, 12, 56, 23, 29

The first step to calculating the median is to arrange these values in ascending order:

3, 5, 7, 12, 13, 14, 21, 23, 23, 23, 23, 29, 39, 40, 56

Now, we must find the center element. The total number of values is 15, so the eighth element will be our central element:

3, 5, 7, 12, 13, 14, 21, 23, **23**, 23, 23, 29, 39, 40, 56

So, we can say that the median in this case is 23.

What if we had only 14 elements instead of 15? In that case, we would have two middle numbers. For example, our list of numbers is:

3, 13, 7, 5, 21, 23, 23, 40, 23, 14, 12, 56, 23, 29

When we put these numbers in ascending order:

3, 5, 7, 12, 13, 14, 21, 23, 23, 23, 23, 29, 40, 56

There are now 14 numbers, so we don't have just one center. Instead, we have a pair of middle numbers:

3, 5, 7, 12, 13, 14, **21, 23**, 23, 23, 23, 29, 40, 56

In this example, the middle numbers are 21 and 23.

To find the median value, add them and divide the sum by 2:

21 + 23 = 44

Then, 44 ÷ 2 = 22

So, the median in this example is 22.

Mode

Mode is the most frequently occuring value in our dataset. It is the only measure that may not be present in the dataset. You will always find the mean or median, but mode is not necessarily present. Also, it is the only measure that can have more than one value.

Normally, the mode is used for categorical data where we wish to know the most common category.

Suppose a researcher asked their students about their color preferences. The following table records their responses:

Number of Students	Color
25	Red
15	Green
10	Yellow
40	Black

Table 2.2

Suppose Red is coded as 1, Green as 2, Yellow as 3, and Black as 4. The following chart denotes that category 4, which is Black, is liked by most people. We can validate it from our graph as well. This value is called mode.

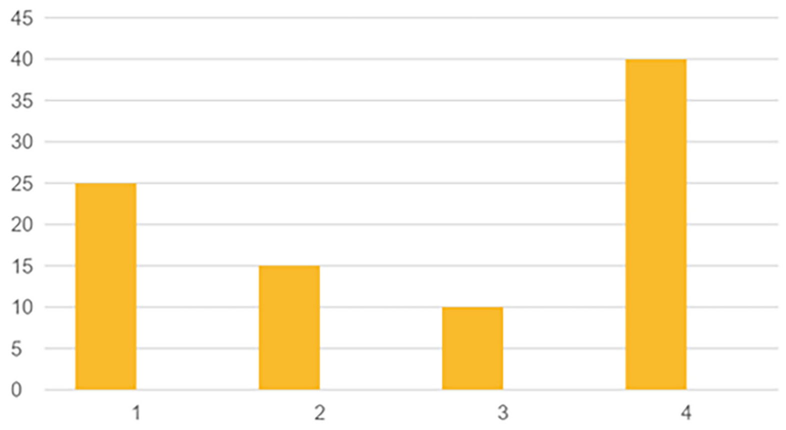


Figure 2.2

Let's look at the different types of modes:

Unimodal data

3, 5, 7, 12, 13, 14, **21, 23, 23, 23, 23**, 29, 39, 40, 56

In the preceding example, 23 is the most frequently appearing number with four instances. So, the mode of this data is 23. Since it has only one mode, it is called as **unimodal data**.

Bimodal data

3, 5, 7, 7, 7, 7, 12, 13, 14, 21, 23, 23, 23, 23, 29, 39, 40, 56

In this example, 23 and 7 both have the maximum frequency of four, so the mode is 23 and 7. Since it has two modes, the data is called **bimodal data**.

Multimodal data

3, 3, 3, 3, 5, **7, 7, 7, 7**, 12, 13, 14, 21, **23, 23, 23, 23**, 29, 39, 40, 56

In this example, 23, 7, and 3 have the maximum frequency of four, so the mode of the data is 23, 7, 3. Since it has more than two modes, the data is called **multimodal data**.

Measures of dispersion

Once we know the central tendency measures, the next thing to know about is the variations present in the data, like how distant the data is from its central tendency measure. It helps you determine whether the data is homogenous or heterogeneous.

Generally, a comparative analysis uses the measures of dispersion. We will be looking at the following measures of dispersion:

1. Range
2. Quartile
3. Inter-Quartile Range
4. Variance
5. Standard Deviation

Range

A **range** is the most common and easily understandable measure of dispersion. It is the difference between two extreme observations of the dataset. If X max and X min are the two extreme observations, then

Range = Xmax – Xmin

Quartile

The **quartiles** divide a data set into quarters. The first quartile (Q1) is the middle number between the smallest number and the median of the data. The second quartile (Q2) is the median, while the third quartile (Q3) is the middle number between the median and the largest number. We have already shown the diagram depicting this in the first chapter. The diagram of outliers in the mean section shows the interquartile range.

Standard Deviation

Standard Deviation is one of the main concepts that deal with finding the variations present in the data. It tells us the extent to which data is scattered while comparing it to the mean. To find the Standard Deviation, we first look at the variance in the data using the following formula:

$$s2 = \sum(x - \overline{x})^2/n$$

Then, we find the root of the variance to calculate the Standard Deviation:

$$\textbf{SD} = \sqrt{\textbf{variance}}$$

Let's look at the following examples to understand both the concepts:

To find the Standard Deviation of 4, 9, 11, 12, 17, 5, 8, 12, 14, we first calculate the mean of the data. After applying the formula, the mean comes out as 10.222.

Now, subtract the mean from each of the given numbers and square the resulting number. This is equivalent to the $(x - \bar{x})^2$.

X	4	9	11	12	17	5	8	12	14
$(x - \bar{x})^2$	38.7	1.49	0.60	3.16	45.9	27.3	4.94	3.16	14.3

Table 2.3

When we add all this up and divide it by the total elements, we get the value of variance, which is: 139.55/9 = 15.51.

Finally, we find the square root of this number to get the value of Standard Deviation, which comes to 3.94.

Standard Deviation vs. Variance

Standard Deviation is the square root of the variance, as mentioned earlier.

It is expressed in the same units as the mean, whereas the variance is expressed in squared units. As a result, Standard Deviation is always better when we're performing a comparative analysis, as we can visualize the comparison more efficiently than with variance.

The Strength of the relationship between variables

To find the strength of the relationship between variables, we must first know what dependent and independent variables are:

Dependent variables

A **dependent variable** is one being tested and measured in an experiment. It is 'dependent' on the independent variable. Whenever the values of independent variables are changed, dependent variables respond accordingly.

Independent variables

An **independent variable** is one that is changed in an experiment to test the effects on the dependent variables.

Let's understand these two concepts with an example:

Suppose a scientist wants to determine whether the type of pitch influences batting. The scientist is controlling the types of pitches, so they are the independent variables here. Now, how the player reacts to the different types of pitches is the dependent variable.

Now that we understand these two variables, we must see how they are related. The strength between the variables can be measured using the following concepts:

1. Covariance
2. Correlation

Let's discuss these concepts one by one.

Covariance

Covariance is a measure of how two variables will respond to the same stimuli. Let's look at the following formula:

$$\textbf{Covariance} = \sum (x - \bar{x})\,(y - \bar{y})/n$$

We can get the value of covariance between two variables using this formula. Remember, we only judge the direction of the relationship using covariance and not its magnitude. Covariance tells us how far the values are spread out from the mean and the nature of the spread. Let's look at a simple example:

Temperature	Number of Customers
98	15
87	12
90	10
85	10
95	16
75	7

Table 2.4

In the preceding table, the first column denotes the temperature in degrees Fahrenheit, while the second column denotes the number of customers visiting an ice-cream shop. Let's see how to find the covariance between them using the formula mentioned earlier.

First, you need to find the mean of each variable.

So, the mean of Temperature is (98+87+90+85+95+75)/6= 88.33, and the mean of the Number of Customers is (15+12+10+10+16+7)/6= 11.67.

Now, subtract each value from its respective mean and multiply these new values together. We get the following table:

Temperature Mean Difference	Customers Mean Difference	Product
9.67	3.33	32.2
-1.33	0.33	-0.44
1.67	-1.67	-2.79
-3.33	-1.67	5.56
6.67	4.33	28.88
-13.33	-4.67	62.25

Table 2.5

The next step is to add these, which gives us the value—125.66.

Then, we divide it by (n-1), which is = 6 - 1 = 5.

125.66/5 = 25.132

So, the covariance of this set of data is 25.132

Correlation

In covariance, we saw how a single factor could influence two different things. But, how does the changing of one factor affect the second? To answer this, we use the concept of correlation. A **correlation** between variables indicates that as one variable changes in value, the other tends to change in a specific direction.

The value of correlation varies between +1 and -1. A value of ± 1 indicates a perfect degree of association between the two variables. As the correlation coefficient value moves toward 0, the relationship between the two variables weakens. To know the direction of the relationship, we must read the signs; a + sign indicates a positive relationship and a – sign indicates a negative relationship

There are three majot types of correlation:

1. Pearson R correlation
2. Kendall rank correlation
3. Spearman rank correlation

Pearson correlation

Pearson R correlation is the most widely used correlation statistic to measure the degree of the relationship between variables. The following formula is used to calculate the Pearson r correlation:

$$p = N\sum xy - \Sigma(x)(y)$$

$$q = \sqrt{}[N\sum x^2 - \sum(x^2)]*[N\sum y^2 - \sum(y^2)]$$

Pearson r = p/q

Kendall correlation

Kendall rank correlation measures the strength of dependence between two variables. The following formula is used to calculate the value of Kendall rank correlation:

$$\tau = (nc - nd)/[½\, n\, (n - 1)]$$

Spearman correlation

Spearman rank correlation is used to measure the degree of association between two variables. The following formula is used to calculate the Spearman rank correlation:

$$\varrho = 1 - (6\sum d^2 i)/n(n^2 - 1)$$

Conclusion

In this chapter, we saw the different aspects of descriptive statistics, which are considered to be the building blocks of the domain of statistics. Once we know these concepts, it becomes a little easier to understand the ones that come under advanced statistics.

In the next chapter, we will look at some of the advanced application of statistics.

Chapter 3
Random Variables

All statistical experiments are done using Random Variables (RVs). Whether it is probability or any other kind of statistical test, RVs are very important. This chapter will take you through the different types of random variables and mathematical properties, along with examples.

Structure

- Random variables
- Joint distributions
- Independent random variables
- Marginal and continuous distribution
- Definition of mathematical expectation
- Chebyshev's inequality
- Law of large numbers

Objective

This chapter aims to provide readers a good understanding of random variables.

Random Variables

When we define the Sample Space of an experiment, we can assign a unique value to each unique element present inside it. These unique values can be represented in the form of a function called **Random Variable** or **Random Function**.

Consider an example where two coins are tossed thrice. The Sample Space then consists of:

$$S = \{HHH,HHT,HTH,HTT,THH,THT,TTH,TTT\}$$

We can assign unique numbers to the preceding Sample Space:

$$X = \{0,1,1,2,1,2,2,3\}$$

This X can be called as a random variable and can also be represented using a function. We will look at the concept of functions later in this chapter.

A random variable that contains finite or infinite members in whole number format is called a **Discrete Random Variable**. If the numbers are in real numbers format, it is called **Continuous Random Variables**. Both these functions are directly related to Discrete and Continuous Probability Distributions, which we talked about in the previous chapter. Let's look at these random variables individually.

Discrete Random Variables

As discussed in the first chapter, discrete random variables take values that can be counted. An example can explain this:

Suppose a coin is tossed twice. The sample space will consist of:

$$S = \{HH,HT,TH,TT\}$$

$$X = \{0,1,1,2\}$$

Let's find the probability of each output:

$$P(HH) = 1/4$$

$$P(HT) = 1/4$$

$$P(TH) = 1/4$$

$$P(TT) = 1/4$$

This is the probability of each element present in the sample space. However, when we talk about the random variables, HT and TH refer to the same thing, since random variables only count unique elements in the Sample Space. So, the probability of each random variable can be written as:

$$P(X=0) = 1/4$$
$$P(X=1) = 1/4 + 1/4 = 1/2$$
$$P(X=2) = \frac{1}{4}$$

So, we can represent this in tabular format, as follows:

X	f(x)
0	¼
1	½
2	¼

Table 3.1

The probability of each random variable is called a **Probability Density Function**. Let's take another example for better understanding:

A die is thrown repeatedly until a 4 is obtained. The probability density function for the number of times we throw the die is as following steps:

1. Let X be the random variable representing the number of times we throw the die.
2. P(X = 1) = 1/6 (if we only throw the die once and we get a 4 on our first throw).
3. P(X = 2) = (5/6) × (1/6) (if we throw the die twice before getting a 4, we must throw something that isn't a 6 with our first throw, the probability of which is 5/6, and we must throw a 6 on our second throw, the probability of which is 1/6).

So, in general, we can say that:

$$P(X = x) = (5/6)^{(x-1)} \times (1/6)$$

Continuous Random Variables

An uncountable random variable is called **Continuous Random Variable**. In this, the random variable can take infinite values. Suppose an individual's height is measures. There are numerous (infinite) ways of measuring height, so height can be considered a **Continuous Random Variable**.

In general, we use the following equation to find the probability distribution function of a continuous random variable:

$$\int_{all\ x} f(x)dx = 1$$

We will not get into too many details of finding the p.d.f. of continuous random variables, as it involves some complex calculus concepts, which are out of the scope of this book.

Joint Distributions

Suppose I have two random variables (discrete): X and Y. If I roll two dice, suppose X contains the value of the outcomes of dice 1, and Y contains the value of the outcomes of dice 2. So,

$$X = \{1,2,3,4,5,6\}$$

$$Y = \{1,2,3,4,5,6\}$$

We can say that X and Y are the Sample Spaces of the two random variables X and Y. Joint distributions take the ordered pair of the elements X and Y and find the probability mass function of the pairs. This can be represented using a joint probability table:

X/Y	1	2	3	4	5	6
1	1/36	1/36	1/36	1/36	1/36	1/36
2	1/36	1/36	1/36	1/36	1/36	1/36
3	1/36	1/36	1/36	1/36	1/36	1/36
4	1/36	1/36	1/36	1/36	1/36	1/36
5	1/36	1/36	1/36	1/36	1/36	1/36
6	1/36	1/36	1/36	1/36	1/36	1/36

Table 3.2

The preceding table shows the probability of the combinations of each outcome. This is how we deal with joint distributions.

When we talk about continuous random variables, we can express the probability mass function with the following equation:

$$\int_c^d \int_a^b f(x,y)dxdy = 1$$

Where x can take values from a to b, while y can take values from c to d.

Independent Random Variables

If one independent variable doesn't affect another random variable in an experiment, it is called as an independent random variable. If I roll two dice, the result of one will never affect the outcome of the other. So, the two random variables can be called independent in this scenario.

On the other hand, a dependent random variable is when one random variable affects another. Suppose two random variables are defined, one storing the outcome of one die, and the second storing the sum of the outcomes of two dice. In this scenario, one random variable will affect the second.

In a more formal scenario, we can say that a random variable is independent if they meet any one of the following conditions:

$$P(X \mid Y) = P(X) \rightarrow \text{for all the values of } x$$

$$P(X \cap Y) = P(X) * P(Y)$$

Marginal and Conditional Distributions

Now that we know joint probability distributions, we can find the marginal as well as conditional distributions from the same joint probability table.

Let's look at the joint probability table again:

X/Y	1	2	3	4	5	6	Total
1	1/36	1/36	1/36	1/36	1/36	1/36	1/6
2	1/36	1/36	1/36	1/36	1/36	1/36	1/6
3	1/36	1/36	1/36	1/36	1/36	1/36	1/6
4	1/36	1/36	1/36	1/36	1/36	1/36	1/6
5	1/36	1/36	1/36	1/36	1/36	1/36	1/6
6	1/36	1/36	1/36	1/36	1/36	1/36	1/6
Total	1/6	1/6	1/6	1/6	1/6	1/6	

Table 3.3

When we look at the total of each row/column, the value is called the marginal distribution. In the preceding table, the marked yellow cells are the marginal distributions. However, when we look at the intersection of the rows and columns, it is called the conditional distribution. As you can see in the table, the probability of getting a 2 is indicated by the intersection, which gives us 1/36.

Let’s take another example:

	Sachin	**Virat**
Men	200	120
Women	170	250

Table 3.4

Suppose this table represents the player preference of men and women. It says that 200 men like Sachin, while 120 like Virat. Similarly, 170 women like Sachin and 250 like Virat. Let‘s calculate the conditional probability of player preference in women.

Let‘s calculate the row and column total:

	Sachin	**Virat**	**Total**
Men	200	120	320
Women	170	250	420
Total	370	370	740

Table 3.5

Total number of women = 420

Let’s calculate the proportion of women who prefer each player:

Sachin $\rightarrow \frac{170}{420} = 0.4$

Virat $\rightarrow \frac{250}{420} = 0.6$

Likewise, we can calculate the conditional distribution table for everyone:

	Sachin	**Virat**
Men	0.625	0.375
Women	0.404762	0.595238

Table 3.6

Definition of Mathematical Expectation

If we can summarize all the values of random variables, either through summation or integration, the final summarized value is called the mathematical expectation. This property of a random variable is represented by $E(x)$. The formula for mathematical expectation is as follows:

$$E(x) = \sum(x_1p_1, x_2p_2, \ldots, x_np_n)$$

Let's understand this with the help of an example:

If I roll a dice, I have six possibilities. The Sample Space will consist of:

S = {1, 2, 3, 4, 5, 6}

Random variable (X) for the preceding Sample Space is = $\left\{\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right\}$

Then, the E(x) will be:

$$1 * \frac{1}{6} + 2 * \frac{1}{6} + 3 * \frac{1}{6} + 4 * \frac{1}{6} + 5 * \frac{1}{6} + 6 * \frac{1}{6}$$

So, after solving the given equation, the value of $E(x) = 3.5$

Let's understand mathematical expectation with the help of another example:

Suppose I toss a coin thrice. The Sample Space will consist of:

S = {TTT,TTH,THT,THH,HTT,HTH,HHT,HHH}

Random variable $X = \left\{\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}\right\}$

So, the expected value $E(x)$ will be:

$$0 * \frac{1}{8} + 1 * \frac{3}{8} + 2 * \frac{3}{8} + 3 * \frac{1}{8}$$

$$= \frac{3}{2}$$

Properties of Mathematical Expectation

Property 1: $E(X + Y) = E(X) + E(Y)$

The mathematical expectation of the sum of two random variables is equal to the sum of individual mathematical expectations.

Property 2: $E(XY) = E(X) * E(Y)$

The mathematical expectation of the product of two random variables is equal to the product of individual mathematical expectations.

Property 3: $E(a * f(X)) = a + E(f(X))$

The mathematical expectation of the product of a constant and the function of a random variable is equal to the sum of the constant and the mathematical expectation of the function of that random variable. The only condition is that the mathematical expectation should exist.

Property 4: $E(aX + b) = aE(X) + b$

The mathematical expectation of the sum of the product between a constant and the function of a random variable and the other constant is equal to the sum of the product between the constant and the mathematical expectation of the function of that random variable and the other constant, provided that their mathematical expectation exists.

Property 5: $E(\sum a_i X_i) = \sum a_i E(X_i)$

The mathematical expectation of the linear combination of the random variables is equal to the sum of the product between all the constant parts of the linear combination and the mathematical expectation of the total number of variables.

Chebyshev's Inequality

If I know the mean of the sample and have the complete sample data, Chebyshev's Inequality says that $1 - \frac{1}{K^2}$ Part of data from the sample must fall within K standard deviations from the mean, where K is a positive real number greater than 1.

Suppose the data is in normal distribution format. Based on the property of normal distributions, the two standard deviations will contain around 95% of the data. So, if the data follows Chebyshev's Inequality, around 95% of it will be present inside the two standard deviations of normal distribution.

For data that is not normally distributed, let's illustrate the inequality conditions:

- If K = 2, then $1 - \frac{1}{K^2} = 1 - \frac{1}{4} = 75\%$.

  So, Chebyshev's Inequality says that at least 75% of the data values of any distribution must be within two standard deviations of the mean.

- If K = 3, then $1 - \frac{1}{K^2} = 1 - \frac{1}{9} = 89\%$.

  So, Chebyshev's Inequality says that at least 89% of the data values of any distribution must be within three standard deviations of the mean.

- If K = 4, then $1-\frac{1}{K^2}=1-\frac{1}{16}=93.75\%$.

  So, Chebyshev's Inequality says that at least 93.75% of the data values of any distribution must be within two standard deviations of the mean.

Chebyshev's Inequality helps us get an insight into the data if we only know about the mean and standard deviation of the data. Let's look at an example to understand the concept.

Suppose there is a random variable X, which denotes the number of customers coming to a restaurant in a day. Suppose the mean of the data is 20, and the standard deviation is 2. Let's determine the probability that customers coming to the restaurant will be between 8 and 32:

1. Let's find the Z-Scores of 8 and 32. It's easy to find, as we know the mean and standard deviation:

$$Z(8)=\frac{|8-20|}{2}=6Z(32)=\frac{|32-20|}{2}=6$$

2. Putting the values in the Chebyshev's Inequality equation, we get:

$$P(8<X<32)=1-\frac{1}{6^2}=35/36$$

So, the probability of the number of customers being between 8 and 32 is 35/36.

Law of large numbers

If we perform the same experiment a larger number of times, we will see that the results will converge toward the expected value. This is the **law of large numbers**.

For example, if a coin is tossed 1,000,000 times, about half of the tosses will result in heads, and the other half will result in tails. The heads-to-tails ratio will be extremely close to 1:1. However, if the same coin is tossed only 10 times, the ratio will likely not be 1:1, and in fact, might come out far different, say 3:7 or even 0:10. This means that you may believe that mathematics is wrong or the experiment is not following maths when the repetition is small. However, as we keep on repeating the experiment, mathematics starts being followed. This error made when we repeat the experiment a smaller number of times is called the Gambler's fallacy.

Let's take another example of the rolling of dice. We've seen in the preceding section that the expected value of rolling dice is 3.5. If I roll the dice only thrice, the result may be far from the expected value. If the results were 1, 1, 4, the average will be 2. This is below 3.5, but if we roll the dice 100 times, we will see that the result is much closer to 3.5.

This is basically what the law of large numbers means.

Conclusion

In this chapter, we learned what a random variable is. We also studied different types, like discrete random variables and continuous random variables and saw some examples based on them. Further, we studied joint distributions along with some examples. Then, we covered mathematical expectation, along with its properties. Lastly, we covered Chebyshev's Inequality and Gambler's fallacy.

In the next chapter, we will walk through the most common yet important topic—probability.

CHAPTER 4
Probability

In this chapter, we will understand the basic concepts related to probability and their practical execution in Python. We will first look at some introductory concepts like random experiments, sample spaces, and so on. Then, we'll explore the properties of probability and some related terminologies. Finally, we'll look at conditional probability and introduce ourselves to probability distributions.

Structure

- Introduction
- Properties of probabilities
- Intersection of sets
- Some terminologies
- Conditional probabilities
- Bayes's theorem
- Probability distribution
  - o Discrete probability distribution
  - o Continuous probability distribution

Objective

This chapter will be the base for the upcoming chapters, as there is no concept in statistics that is unrelated to probability. The objective of this chapter is to help you understand the core of statistics so that learning the advanced concepts becomes easier.

Introduction

When we perform any random experiment, the chance or likelihood of one specific outcome is called **probability**. Before moving on, we must know what a random experiment is.

When we perform any kind of experiment, for example, tossing a coin or rolling a dice, they always have random elements. These components can bring a negligible or major change in the outcome of the event. So, any experiment that may have random components is called a **random experiment**. When we toss a coin or throw the dice, the friction, the speed at which they are thrown, and so on can be the random components, and so, they are random experiments.

A random experiment stores all its outcomes in a set named as sample space. So, when we toss a coin, the sample space contains **Heads** and **Tails**. When we throw a dice, it contains outcomes *{1,2,3,4,5,6}*:

$$S_{coin} = \{Heads, Tails\}$$

$$S_{dice} = \{1,2,3,4,5,6\}$$

When we are looking at a particular outcome from a sample space of random experiment, it is stored in a set called an **event**. An event can be termed as a subset of sample space. Events and sample space of a random experiment are used to determine the probability.

Suppose I want to determine the probability of an outcome of 4 when we throw a die. We already know that the sample space of throwing a die contains six members, while the event contains only 1 member (outcome of 4). So, mathematically, probability can be defined as:

$$P(A) = \frac{number\ of\ members\ \in the\ Event}{number\ of\ members\ \in the\ Sample\ Space} = \frac{number\ of\ favourable\ outcomes}{total\ number\ of\ outcomes}$$

So, the probability of an outcome of 4 becomes:

$$P(outcome\ of\ \ 4) = \frac{1}{6}$$

Some other examples of probability are given as follows:

Example 1: A die is rolled once. What is the probability of getting a prime number?

Die is rolled once, so the sample space will be:

$$S_{dice} = \{1,2,3,4,5,6\}$$

Getting a prime number will have the following event set:

$$\text{E} = \{2,3,5\}$$

So, the probability of the event set defined will be:

$$P(E) = \frac{3}{6} = 0.5$$

Example 2: In a simultaneous toss of two coins, what is the probability of getting two heads and exactly one head?

Since the coin has been thrown twice, the sample space will contain:

$$S_{coin} = \{HH,TT,HT,TH\}$$

Getting two heads has only one possible event:

$$E = \{HH\}$$

So, the probability of getting two heads = $\frac{1}{4} = 0.25$

Getting only/exactly one head has two possible events:

$$E = \{HT,TH\}$$

So, the probability of getting exactly one head = $\frac{2}{4} = 0.5$

Example 3: A ball is drawn at random from a bag containing 10 red, 4 blue, and 6 black balls. What is the probability of drawing: one red ball, one blue ball, not a black ball?

Let's solve this case by case:

Case 1: Drawing a red ball

The sample space contains the total number of balls = 10 + 4 + 6 = 20

Since there are 10 red balls, E = 10.

So, the probability of drawing one red ball = $\frac{10}{20} = 0.5$

Case 2: Drwaing one blue ball

There are four blue balls, so E = 4.

So, the probability of drawing 1 blue ball = $\frac{4}{20} = 0.2$

Case 3: Not drawing a black ball

There is a total of 14 balls that are not black, which means we just need to find the probability of drawing these 14 balls, so E = 14.

The probability of not drawing a black ball = $\frac{14}{20} = 0.7$

Given here is the Python code for determining the probability if the sample space and event set are given:

```
sample_space = int(input("Enter number of Elements in the Sample space"))

event = int(input("Enter number of Elements in the Event Set"))

probability = event/sample_space

print(probability)
```

Properties of probability

The following are some of the properties that probability must satisfy:

1. The probability of sample space should be 1
$$P(S) = 1$$
2. The probability of any event must lie between 0 and 1
$$0 \leq P(E) \leq 1$$
3. The probability of a null set should be null
$$P(\varnothing) = \varnothing$$
4. The probability of the complement of an event should have a value that you get once you subtract the probability of the event from 1
$$P(E') = 1 - P(E)$$

When we look at the mentioned properties, there are a lot of new terms to understand. So, let's start this section by understanding some of the terms related to sets.

Intersection of sets

Common elements present in two different sets can be referred to as the **intersection of two sets**.

$$A = \{2,4,5\}$$

$$B = \{1,3,4,9\}$$

$$A \cap B = \{4\}$$

The following Python code gives the intersection of any two sets stored in variables A and B:

```
A = {1,2,3,4,5,6,7,8,9}

B = {1,3,5,7,9}

print("intersection of A and B is", A.intersection(B))
```

Union of sets

Combining all the elements in two different sets is called the **union of two sets**:

$$A = \{2,4,5\}$$

$$B = \{1,3,4,9\}$$

$$A \cup B = \{1,2,3,4,5,9\}$$

The following Python code provides the union of any two sets stored in variables A and B:

```
A = {1,2,3,4,5,6,7,8,9}

B = {1,3,5,7,9}

print("Union of A and B is", A.union(B))
```

The complement of a set

If we have a set and find all the elements present in the universal set, except the given set, it is called the **complement of a set**. In the case of probability, sample space is termed as the universal set.

$$S = \{1,2,3,4,5,6,7,8,9\}$$

$$E = \{1,3,5,7,9\}$$

$$E' = \{2,4,6,8\}$$

The following Python code gives the difference of any two sets stored in variables A and B:

```
A = {1,2,3,4,5,6,7,8,9}

B = {1,3,5,7,9}

print("If A is the Universal Set, Complement of B is", A.difference(B))
```

Null set

A set containing no elements is called a **null or empty set**.

$$A = \{2,4,5\}$$

$$B = \{1,3,6,9\}$$

$$A \cap B = \varnothing$$

The following Python code finds the intersection between the two sets and provides the results in a null set, as no members are common.

```
A = {1,2,3,4,5,6,7,8,9}

C = {10,11}

print("intersection of C and A is", C.intersection(A), "\n Therefore it
is a Null Set")
```

Subset/superset

If all the elements of one set are present in another set, the former is called the **subset**, while the latter is called a **superset**.

$$S = \{1,2,3,4,5,6,7,8,9\}$$

$$E = \{1,3,5,7,9\}$$

Since S contains all the elements that E has, S is called a superset and E is called a subset.

The following Python code tells whether a set is a subset or superset of the second set given.

```
A = {1,2,3,4,5,6,7,8,9}

B = {1,3,5,7,9}
```

```
print("Is B subset of A?", B.issubset(A))

print("Is A superset of B?", A.issuperset(B))
```

Some other terminologies

Now that we have looked at the properties and the operations that can be performed on sets, let's look at a few other concepts and rules that tell us a lot about events and sets present in the sample space.

Mutually exclusive events

If two events have nothing in common or their intersection gives a null set, the events are termed **mutually exclusive events**.

$$A = \{2,4,5\}$$

$$B = \{1,3,6,9\}$$

$$A \cap B = \emptyset$$

Mutually exhaustive events

When the union of events gives us the sample space, the events are termed as **mutually exhaustive events**.

$$E_1 + E_2 + E_3 + \ldots E_n = S$$

Commutative laws

The union of two events is equal to their union reversed.

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Associative laws

The intersection or union of three events is interchangeable.

$$(A \cup B) \cup C = A \cup (B \cup C)$$

$$(A \cap B) \cap C = A \cap (B \cap C)$$

Distributive laws

The intersection and union of three events are interchangeable.

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup B \cap C = (A \cup B) \cap (A \cup C)$$

De Morgan's law

The overall complement of the union or intersection of two events is the same as the individual event's complement and union or intersection.

$$(A \cup B)' = A' \cap B'$$

$$(A \cap B)' = A' \cup B'$$

Conditional probability

For the past few years, we have observed that when it rains in January, it is followed by snowfall most of the time. Suppose it's the month of January and it's raining, and we want to find the probability of snowfall; this scenario comes under the scope of **conditional probability**.

When we have two or more related events, and we want to know the probability of one event, knowing that the second event has already occurred, it is called a **conditional probability**. We can find the probability of snowfall using the following conditional probability formula:

$$P(snowfall \mid Rain) = \frac{P(Snowfall \cap \ Rain)}{P(Rain)}$$

By looking at the data for the past few years, we can easily get the information for the days when snowfall and rain occurred together. We can also know the number of days it rained. Keeping these two pieces of information in mind, we can find the probability of snowfall if it is currently raining.

We can generalize the conditional probability formula by:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Let's understand this concept with a few more examples:

Example 1: Suppose there are a total of seven fuses, out of which two are defective. If we test one fuse at a time, randomly and without replacement, what is the probability that we find both the fuses in the first two attempts?

The probability of getting defective fuses = $\frac{2}{7}$

If I get the first fuse as defective, the probability of getting the second one as defective = $\frac{1}{6}$

We have calculated the high probability already knowing that the first fuse we drew was a defective one. So, high probability is nothing but a conditional probability. This means:

$$P(2nd\ defective\ fuse \mid 1st\ defective\ fuse) = \frac{1}{6}$$

Conditional probability will say that:

$$P(2nd\ defective\ fuse \mid 1st\ defective\ fuse) = \frac{P(2nd\ defective\ fuse \cap 1st\ defective\ fuse)}{P(1st\ defective\ fuse)}$$

This formula can be reframed as:

*P(2nd defective fuse ∩ 1st defective fuse) = P(2nd defective fuse | 1st defective fuse) * P(1st defective fuse)*

So, the probability of getting both the defective fuses together = $\frac{\frac{2}{7} * 1}{6}$

Example 2: You toss a fair coin thrice. Given that you have observed at least one heads, what is the probability that you observe at least two heads?

Suppose we have two events: $E1$ and $E2$. E1 is the event containing at least one head, and E2 is the event containing at least two heads. The sample space will contain,

S = {HHH, HHT, HTH, HTT, THH, TTT,TTH,THT}

We can see that all the outcomes have at least one head, except TTT. So, the probability of event 1 is:

$$P(E1) = \frac{7}{8}$$

Having at least two heads makes the following Event,

E2 = {HHH,HHT,HTH,THH}

$$\text{So, } P(E2) = \frac{4}{8}$$

The question asks for the probability of observing at least two heads given that we have observed at least one head. So, it's a conditional probability scenario.

$$P(E2 \mid E1) = \frac{P(E2 \cap E1)}{P(E1)} = \frac{P(E1)}{P(E1)} = \frac{4}{7}$$

Since E2 is a subset of E1, P(E2 ∩ E1) = P(E2)

In the following Python code, we'll try to solve the preceding problem. We have created a function that returns the conditional probability of two events occuring.

```
def cond_prob(known_prob, combined_prob=False, conditional_prob=False):

    if conditional_prob == False:

        return(combined_prob/known_prob)

    else:

        return(conditional_prob*known_prob)

#Question 1:

print("Probability of getting both the defective fuses together ", cond_
prob(known_prob=2/7, conditional_prob=1/6))

#Question 2:

print("Probability of observing at least two heads", cond_prob(combined_
prob = 4/8, known_prob=7/8))
```

Dependent and independent events

Suppose the probability of two events follows the following properties:

- *P(A | B) = P(A)*∨
- *P(A∨B) = P(A)*P(B)*

Then, we can say that the events are **independent**. However, they are called **dependent events** if they don't follow this property.

Example: Two cards have been drawn from the deck of 52 cards without replacing the first one. Find the probability of getting the first card as king and the second card as queen.

The total number of kings = E(Kings) = 4

The total number of cards = sample space = 52

The probability of drawing the king first = P(E1) = $\frac{4}{52}$

The total number of queens = E(Queens) = 4

The total number of cards after the king is drawn = sample space = 51

The probability of drawing queen second = P(E2) = $\frac{4}{51}$

We know that the two events are dependent since the queen can be drawn only after the king is drawn. So, the probability of getting a king first and then the queen is:

$$P(E1) * P(E2) = \frac{\frac{4}{51} * 4}{52} = \frac{4}{663}$$

The following Python code contains a method that takes the probability of two dependent events and returns the probability of the second event occuring:

```
def dep_events(event_1, event_2):

   return(event_1*event_2)


print("the probability of getting king first and then queen is", dep_
events(4/52, 4/51))
```

Bayes's theorem

As we know conditional probability, we can directly derive Bayes's theorem from that. Let's look at the derivation first, and then we'll formally define Bayes's theorem:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Similarly:

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

Based on the commutative law:

$$A \cap B = B \cap A$$

$$\therefore P(A \cap B) = P(B \cap A)$$

So, we can say that:

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)}$$

The preceding formula is the Bayes's theorem. If we have multiple causes of an event and know the probability of the occurrence of each cause and the conditional probability of the outcome of each cause, we can calculate the conditional probability of each cause as well. The detailed structure of Bayes's theorem is given as follows:

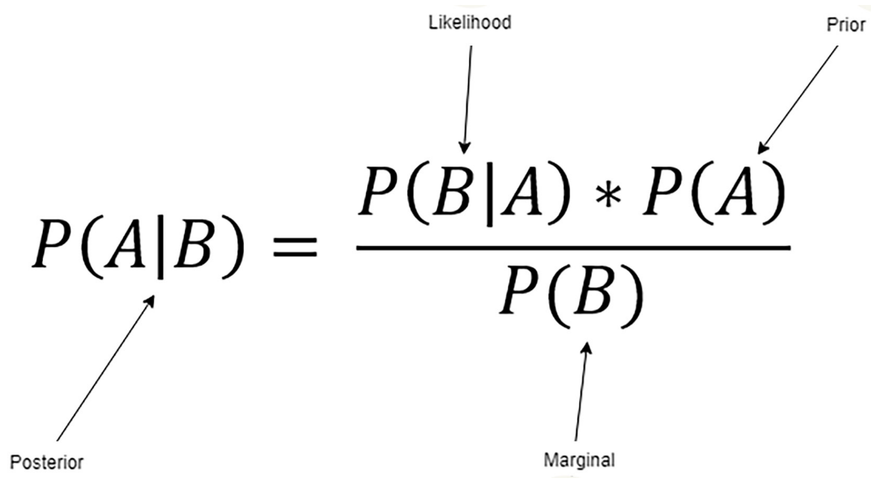


Figure 4.1

Let's understand this concept with the help of examples:

Example 1: There are two machines that manufacture bolts in a factory. Out of all the bolts, 75% are manufactured by the first machine and the remaining are produced by the second. A total of 5% of the bolts from the first machine are defective, while 8% from the second are defective. If we select a bolt at random and know that it is defective, what is the chance that it came from the first machine?

The probability of a bolt coming from the first machine: P(First Machine) = 0.75

The probability of a bolt coming from the second machine: P(Second Machine) = 0.25

If the bolt comes from the first machine, the probability of it being defective is: P(Defective | First Machine) = 0.05

If the bolt came from the second machine, the probability of it being defective is: P(Defective | Second Machine) = 0.08

Now, we can use conditional probability to determine the probability of a bolt coming from the first machine, knowing that it is defective:

$$P(FirstMachine \mid Defective) = \frac{P(Defective \mid FirstMachine) * P(FirstMachine)}{P(Defective)} = \frac{0.05 * 0.75}{P(Defective)}$$

Since 5% of the bolts that come from the first machine and 8% of those that come from the second machine are defective, the probability of a defective bolt becomes:

$$P(Defective) = (0.05 \times 0.75) + (0.08 \times 0.25) = 0.0575$$

$$\therefore P(FirstMachine \mid Defective) = \frac{0.05 * 0.75}{0.0575} = 0.385$$

Example 2: In Orange county, 51% of adults are males. One adult is randomly selected for a survey involving credit card usage. It is later learned that the selected survey subject was smoking a cigar. Also, 9.5% of males smoke cigars, whereas 1.7% of females smoke cigars (based on the data from the Substance Abuse and Mental Health Services Administration). Use this additional information to find the probability that the selected subject is a male.

The probability of a person being a male = $\frac{51}{100} = 0.51$

the probability of a person being a female = $\frac{49}{100} = 0.49$

We know that 9.5% of males smoke cigars, so the probability of a person being a male and smoking a cigar is 0.95.

Similarly, since 1.7% of females smoke a cigar, the probability of a person being a female and smoking a cigar is 0.017.

To find out the probability that a person is male and smokes a cigar, we can use conditional probability. If we apply the conditional probability formula in this example, we get:

$$P(Male \mid Cigar) = \frac{P(Cigar \mid Male) * P(Male)}{P(Cigar)} = \frac{0.095 * 0.51}{P(Cigar)}$$

We know the numerator, but the probability of a person smoking a cigar is not explicitly written in the question. However, we can deduce this information by the values we already have. Since 9.5% of males smoke cigars and 1.7% of females smoke cigars, and 51% of the population is male and 49% is female, the probability of a person smoking cigar becomes:

$$P(Cigar) = (0.095 * 0.51) + (0.017 * 0.49) = 0.0568$$

Now we know both the numerator and the denominator:

$$P(Male \mid Cigar) = \frac{0.95 * 0.51}{0.0568} 0.853$$

Probability distributions

P**robability distribution** is a function that looks at all the possible outcomes of an experiment and gives us the probability of the occurrence of each one. We have already discussed random variables and their types in the previous chapter. When we deal with the outcomes of a discrete random variable, it's called a **discrete probability distribution**. Similarly, the outcomes of a continuous random variable are called a **continuous probability distribution**.

Some of the discrete probability distributions that we'll be discussing in this chapter are:

1. Binomial distribution
2. Geometric distribution
3. Poisson distribution

Let's understand in the following section:

Binomial distribution

The visual representation of binomial distribution is as follows:

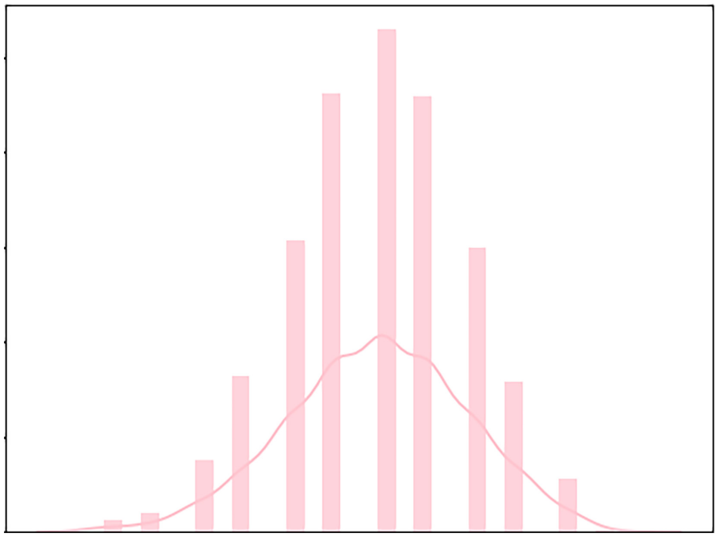


Figure 4.2

When we perform an experiment that has only two possible outcomes, defined as *success* and *failure*, the experiment is called a **Bernoulli Trial,** and the distribution of the probabilities of its outcomes is called a **binomial distribution**. If we roll a dice and look at an event of the occurrence of number 5, five is a success and anything else is a failure. So, this experiment can be an example of binomial distribution.

Mathematically, a binomial function can be represented as:

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Where:

- $0 < p < 1$
- $n = 1, 2, \ldots$
- $x = 0, 1 \ldots n$

Before looking at some examples, the following are some of the assumptions that we must know for a binomial function.

1. All the trials are independent of each other.
2. The probability of successes should be constant throughout the trials.

Example: According to an airline, flights on a certain route are on-time 85% of the time. Suppose 20 flights are randomly selected and the number of on-time flights is recorded.

(a) Find and interpret the probability that exactly 14 flights are on time.

(b) Find and interpret the probability that fewer than 14 flights are on time.

(c) Find and interpret the probability that at least 14 flights are on time.

(d) Find and interpret the probability that between 12 and 14 flights, inclusive, are on time.

a. The total number of trials = 20

Number of successes = 14

Probability of successes = 0.85

Putting all this in the binomial formula, we get:

$$P(exact 14 flights) = \binom{20}{14} 0.85^{14} (1 - 0.85)^{20-14}$$

$$0.045$$

b. The total number of trials = 20

   Number of successes = 1,2,3 … , 13

   Probability of successes = 0.85

   Putting all this in the binomial formula, we get:

$$P(14\,flights) = \left[\binom{20}{1}0.85^{1}(1-0.85)^{20-1}\right] + \left[\binom{20}{2}0.85^{2}(1-0.85)^{20-2}\right] + \left[\binom{20}{3}0.85^{3}(1-0.85)^{20-3}\right] + \ldots \left[\binom{20}{13}0.85^{13}(1-0.85)^{20-13}\right]$$

$$0.022$$

c. The total number of trials = 20

   Number of successes = 14, 15, 16 … ,20

   Probability of successes = 0.85

   Putting all this in the binomial formula, we get:

$$P(\geq 14 flights) = 1 - \mathrm{P(14flights)} = 1 - 0.022 = 0.98$$

d. The total number of trials = 20

   Number of successes = 12,13,14

   Probability of successes = 0.85

   Putting all this in the binomial formula, we get:

$$P(12\,|\,14\,flights) = \left[\binom{20}{12}0.85^{12}(1-0.85)^{20-12}\right] + \left[\binom{20}{13}0.85^{13}(1-0.85)^{20-13}\right] + \left[\binom{20}{3}0.85^{14}(1-0.85)^{20-14}\right] = 0.066$$

In the following Python code, we have used the Python statistical package SCIPY for solving the preceding binomial distribution problem:

```
import scipy.stats as stats

#Answer 1

x = stats.binom(n=20, p=0.85).pmf(14)
```

```
print("Probability of exact 14 flights on time is", x)
#Answer 2
y = stats.binom(n=20, p=0.85)
total_p = 0
for k in range(1, 14):
total_p += y.pmf(k)
print("Probability of less than 14 flights on time is", total_p)
#Answer 3
print("Probability of at least 14 flights on time is", 1-total_p)
#Answer 4
z = stats.binom(n=20, p=0.85)
total_p = 0
for k in range(12, 14+1):
total_p += y.pmf(k)
print("Probability of 12 to 14 flights on time is", total_p)
```

Geometric distribution

The visual representation of binomial distribution is as follows:

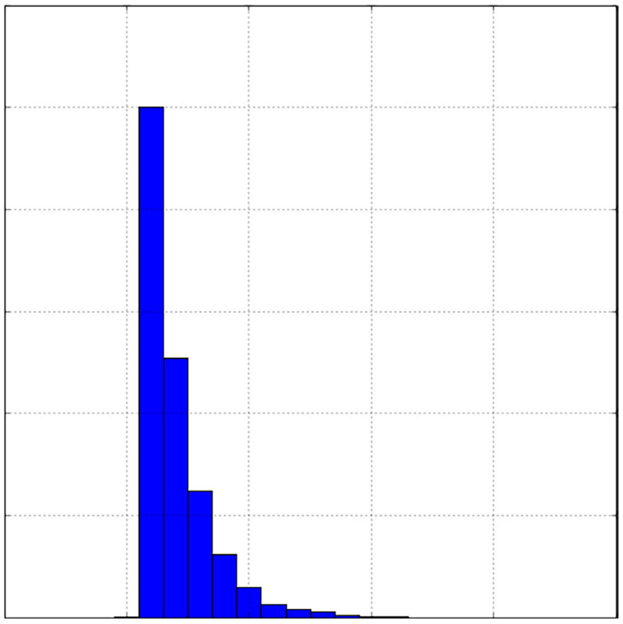


Figure 4.3

In binomial distribution, we saw that the number of trials or the number of times an experiment is performed was fixed. This means that success may or may not be obtained in a fixed number of trials. Instead, if we keep on experimenting until success is obtained rather than only a fixed number of times, this process comes under the scope of **geometric distribution**.

Mathematically, a geometric distribution can be represented by:

$$f(x) = p(1 - p)^{x-1}$$

Example: A coin has been weighted so that it has a 0.9 chance of landing on heads when flipped. What is the probability that the first time the coin lands on heads is after the third flip?

The probability of success in this question forgets the heads after the third flip. This can be represented by $P(y > 3)$.

Finding the probability of all the trials greater than three can be impossible to count. So, since all the events are independent, we can find the probabilities of all the failed trials and subtract it from 1 instead of finding the probability of all the successful trials. So:

$$P(y > 3) = 1–P(y \leq 3)$$

Based on the geometric distribution formula, we can get the solution as follows:

$$P(y > 3) = 1–P(y \leq 3) = 1–[(0.9*0.1^{1-1}) + (0.9*0.1^{2-1}) + (0.9* 0.1^{3-1})] = 0.001$$

```
import scipy.stats as stats

answer = 1 - (stats.geom(p=0.9).pmf(1)+stats.geom(p=0.9).pmf(2)+stats.
geom(p=0.9).pmf(3))

print(“The probability of landing heads after 3rd trial is: “, answer)
```

Poisson distribution

The visual representation of Poisson distribution is as follows:

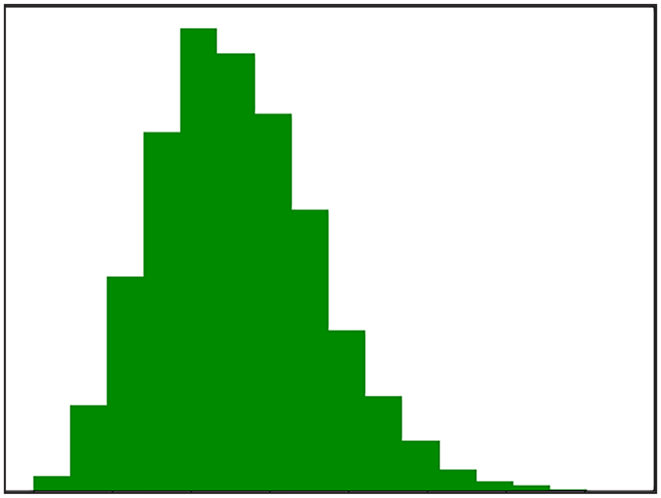


Figure 4.4

To understand the Poisson distribution, we must first know what a Poisson process is. Any experiment is termed as a Poisson process if it follows the following properties:

1. For multiple events happening, we must know the average time interval between them.
2. However, the exact time interval between any two events is unknown.
3. All the events are independent of each other.

So, if an experiment is a Poisson process, we can determine the probability of all its possible outcomes using Poisson distribution. Mathematically, a Poisson distribution can be represented as:

$$f(x) = \frac{e^{-\lambda}(\lambda)^x}{x!}$$

Where λ tells us the average time interval and x tells us about the successful event. Let's understand this with the help of an example:

A radioactive source emits 4 particles on average for 5 seconds.

a) Calculate the probability that it emits 3 particles for 5 seconds.

b) Calculate the probability that it emits at least one particle for 5 seconds.

c) During 10 seconds, what is the probability that 6 particles are emitted?

Average particles emitted (λ) = 4

a. Probability of three particles:

$$P(y=3)=\frac{e^{-4}(4)3}{3!}=0.195$$

b. Probability of at least one particle:

$$P(y\geq 1)=1-P(y=0)=1-\frac{e^{-4}(4)^0}{0!}=0.9817$$

c. Probability of 6 particles in 10 seconds (λ will become 8 for 10 seconds period, as it is 4 for 5 seconds period):

$$P(y=6)=\frac{e^{-8}(8)^6}{6!}=0.122$$

In the following code, we have used the SCIPY package in Python to solve the problem statement:

```
three_particles = stats.poisson(mu=4).pmf(3)

atleast_1 = (1 - stats.poisson(mu=4).pmf(0))

six_particles = stats.poisson(mu=8).pmf(6)

print("Probability of 3 particles is: ", three_particles)

print("Probability of at least 1 particle is: ", atleast_1)

print("Probability of 6 particles in 10 seconds time period is: ", six_
particles)
```

Now that we have looked at different types of discrete probability distributions, let's move on to continuous probability distribution.

Normal distribution

The visual representation of normal distribution is as follows:

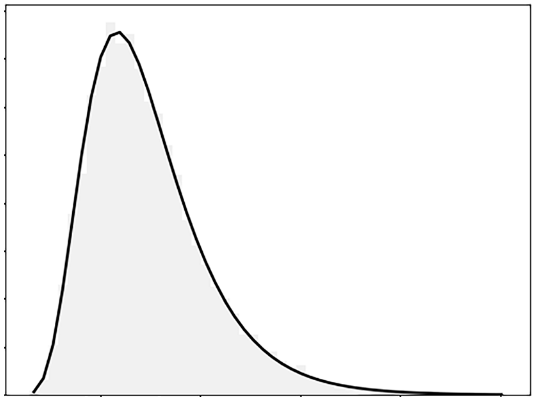


Figure 4.5

The first and the most important distribution in the list of continuous probability distributions is normal distribution, also known as **Gaussian distribution**. A normal distribution states that data present near the mean has a higher probability of occurrence as compared to the data present far from the mean. A normal distribution graph looks like a bell curve, as shown here:

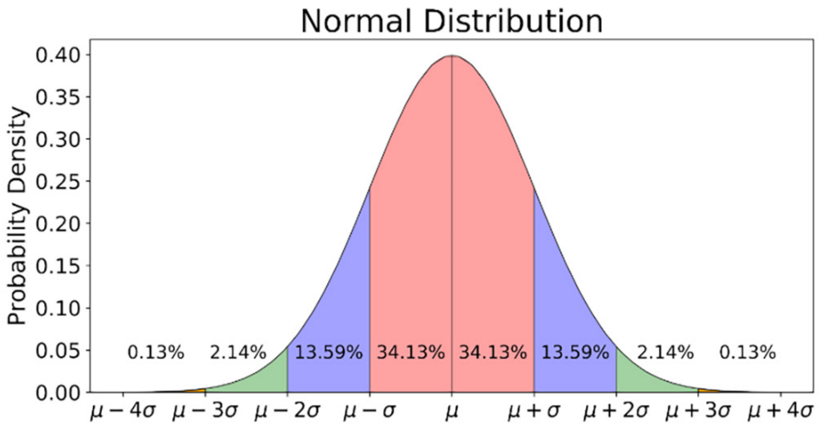


Figure 4.6

Where μ represents the mean of data, while σ represents standard deviation. Looking at the preceding graph, you can see that approximately 68% of the data lies between one standard deviation difference from the mean, approximately 95% of data lies between two standard deviation differences from the mean, and approximately 99% of data lies between three standard deviation differences from the mean.

We must understand that a normal distribution is a symmetrical distribution, as you can see in the preceding graph. However, we cannot say that all symmetrical distributions are normal. At the same time, we can convert any distribution to a normal one by transforming every value into its z-score. The formula for the conversion is:

$$z = \frac{x - \mu}{\sigma}$$

Let's understand the normal distribution and Z-Score concepts with the help of the following example.

The annual salaries of employees in a large company are distributed with an approximate mean of $50,000 and a standard deviation of $20,000.

a) **What percentage of people earn less than $40,000?**

b) **What percentage of people earn between $45,000 and $65,000?**

c) **What percentage of people earn more than $70,000?**

a. $Z = \frac{40000 - 50000}{20000} = -0.5$

The normal distribution graph looks like this:

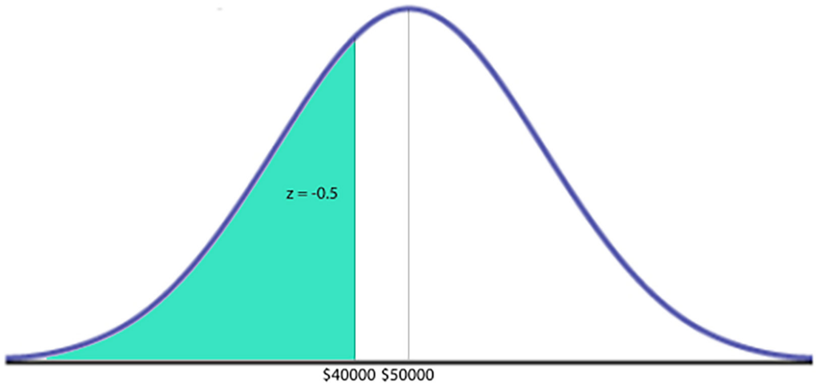
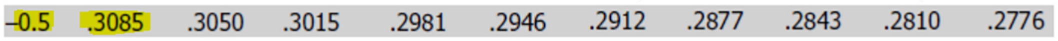


Figure 4.7

Looking at the probability value of Z = -0.5 in the Z-Table, we get:



-0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2877	.2843	.2810	.2776

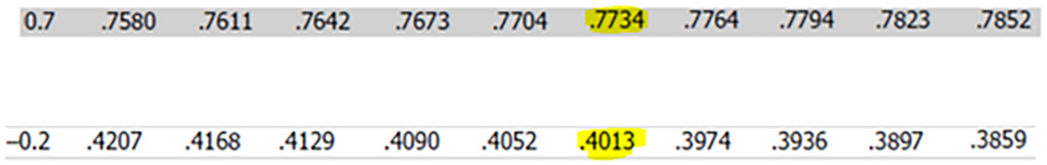
Figure 4.8

The green shaded area is = 30.85%, so 30.85% of people earn less than $40000.

b. Z-Scores for 45000 and 65000 values are:

$$Z_1 = \frac{45000 - 50000}{20000} = -0.25$$
$$Z_2 = \frac{65000 - 50000}{20000} = 0.75$$

We'll find that they are less than 0.75 and -0.25. We'll then subtract them to get the final area:



0.7	.7580	.7611	.7642	.7673	.7704	.7734	.7764	.7794	.7823	.7852
–0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897	.3859

Figure 4.9

The area less than 0.75 is = 77.34%

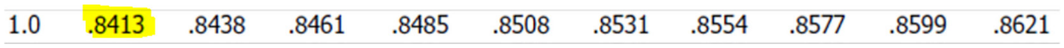
The area less than -0.25 is = 40.13%

So, common area = 77.34% – 40.13% = 37.21%

c. Z-Score for 70000 is

$$Z = \frac{70000 - 50000}{20000} = 1$$

Area to the left of z = 1 becomes 84.13%, as given here:



1.0	.8413	.8438	.8461	.8485	.8508	.8531	.8554	.8577	.8599	.8621

Figure 4.10

So, the area toward the right becomes 100 – 84.13 = 15.87%

In the following code, we have solved this problem using the SCIPY package:

```
import scipy.stats as stats

answer1 = stats.norm(50000, 20000).cdf(40000)

answer2 = stats.norm(50000, 20000).cdf(65000) - stats.norm(50000, 20000).
cdf(45000)

answer3 = stats.norm(50000, 20000).cdf(70000)
```

```
print("percent of people earning less than $40,000 is {:.5f}%".
format(answer1*100))

print("percent of people earning between $45,000 and $65,000 is {:.5f}%".
format(answer2*100))

print("percent of people earning more than $70,000 is {:.5f}%".format((1
- answer3)*100))
```

Conclusion

In this chapter, we learned the concept of probability, some related rules and laws, and some of the probability distributions that are used in the practical application of statistics. We will be using most of the concepts in this book as well as in advanced applications like machine learning. There is also a separate statistical domain that deals with Bayes's theorem, called Bayesian statistics, but all these are out of the scope of this book for now.

In the next chapter, we'll delve deeper into statistics and look at how to calculate some of the unknown parameters present in statistical problem statements. This will help us ensure that our assumptions about the population hold valid.

Chapter 5
Parameter Estimation

In this chapter, we will discuss parameter estimation and the methods to determine it. We will try to understand what a parameter is and why we need to find them. Once done, we'll be discussing the distributions of sample and some related theorems. Once we find the estimates, we will explore metrics to understand the accuracy. Finally, we will discuss two simple methods to estimate a parameter.

Structure

Given here is the structure of this chapter:

- Parameter estimation
- Point estimates
- Sampling distributions
    - o Central limit theorem
- Estimators having bias
- Variance, standard error, and mean squared error
- Point estimate methods
    - o Method of moments
    - o Maximum likelihood method
- Confidence intervals

Objective

In this chapter, we will look at the strategies and methods used to find parameters that are unknown for a population, and we will try to estimate the same through the information present in the sample. Let's start by understanding the concept of parameter estimation.

Parameter estimation

Whenever we experiment, it is majorly about drawing conclusions for the entire population and making decisions based on that. This field of statistics is called **statistical inference**, and it majorly talks about two things:

- Parameter estimation
- Hypothesis testing

In the next chapter, we will talk about hypothesis testing. This chapter is dedicated to parameter estimation and its usage.

Parameter estimation, as the name suggests, means finding the value of an unknown parameter related to an experiment. This parameter represents the population whose value we are trying to derive from a sample of the population. When we take sample data and perform experiments to compute some metrics, assuming that the metric is the best way to represent a population is called a **point estimate**. Let's understand this with the help of an example.

Suppose a company wants to find out whether the employees are satisfied with the working environment. For this, they can send across a questionnaire. But, if a few metrics require an individual's presence, and the questionnaire cannot serve the purpose, it will be very difficult to get each individual's input. If the company is an MNC, it will be next to impossible. Here, parameter estimation will come to the company's help.

What the company can do is take an intelligent sample from the entire population or all the employees. This means employees from different departments and job roles will be selected for the sample. Then, questions will be asked and the responses noted down. Finally, a group of statisticians will come up with metrics, examples of mean, or majority, etc. to help them decide whether the response can be termed as the overall response of the population. This metric will be termed as a point estimate for an employee satisfaction survey.

Point estimate – The mathematics way

The methodology of determining a few unknown parameters of the population with the information available about the samples taken from the same population is termed as **parameter estimation**. Also, when we find the best value of the parameters of interest, those values are called point estimates. Let's understand this better with a mathematical approach.

$x_1, x_2, \ldots x_n$ random variables for the experiment and its observations

The metrics that we want to conclude for this experiment can be the mean and the variance, $\overline{X}$ and σ^2. They can be termed as the functions of the preceding observations, as known as statistic. We must also know that these functions will also be random variables, just like the random variables for the observations.

An important thing to note here is that the probability distribution of a statistic will be the same as the probability distribution of the random variable and will be called the **sampling distribution**. In the following equation, θ refers to the parameter of interest for which we have to find the point estimate.

$$\theta = \mu \text{ OR } \theta = \sigma^2$$

The following image reflects the same:

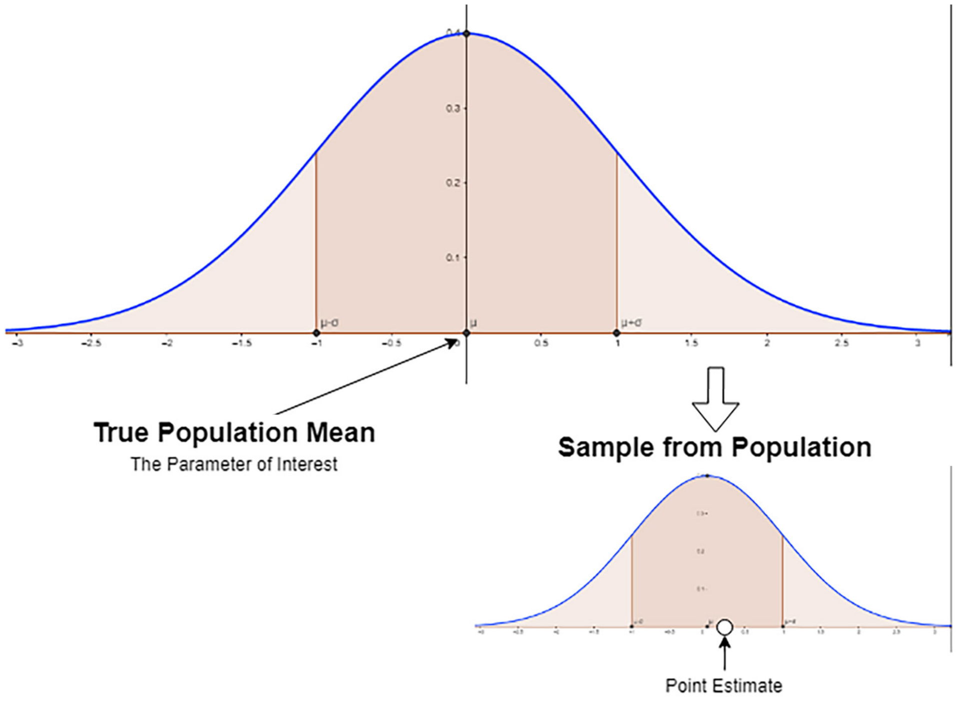


Figure 5.1: *Point estimate of the sample taken from a population*

As we have already mentioned, either the mean or variance can be the parameter of interest, and that's what is shown in the preceding equation. We can understand this with the help of another example.

Suppose we want to find whether the students who appeared for physics class tenth board examination passed. We will take a sample of marks from the entire population and assume that it follows a normal distribution, and this will be our random variable. Suppose the marks considered are:

$x_1 = 26$

$x_2 = 29$

$x_3 = 30$

$x_4 = 30$

The best parameter of interest for our problem will be the mean, so:

$$\theta = \mu = \frac{x_1 + x_2 + x_3 + x_4}{4} = 28.75$$

Assuming that the passing marks are 31, the point estimate provides the analysis going toward the failure side. However, we have taken a sample of only four points, so this will not be a valid experiment.

Sampling distributions

As we already know, when we take a sample from an entire population, that sample is termed as **random sample**. If we find the probability distribution for this random sample, it is termed as **sampling distribution**. Suppose we want to find the statistic – mean. We can take different samples from the population, represented by X_1, X_2, X_3...

Some things to notice about these random samples are:

- A random sample should be a random variable in itself
- All the random samples should have the same probability distribution

This probability distribution that we are talking about is the sampling distribution of the random variable X, and it can be seen in *Figure 5.2:*

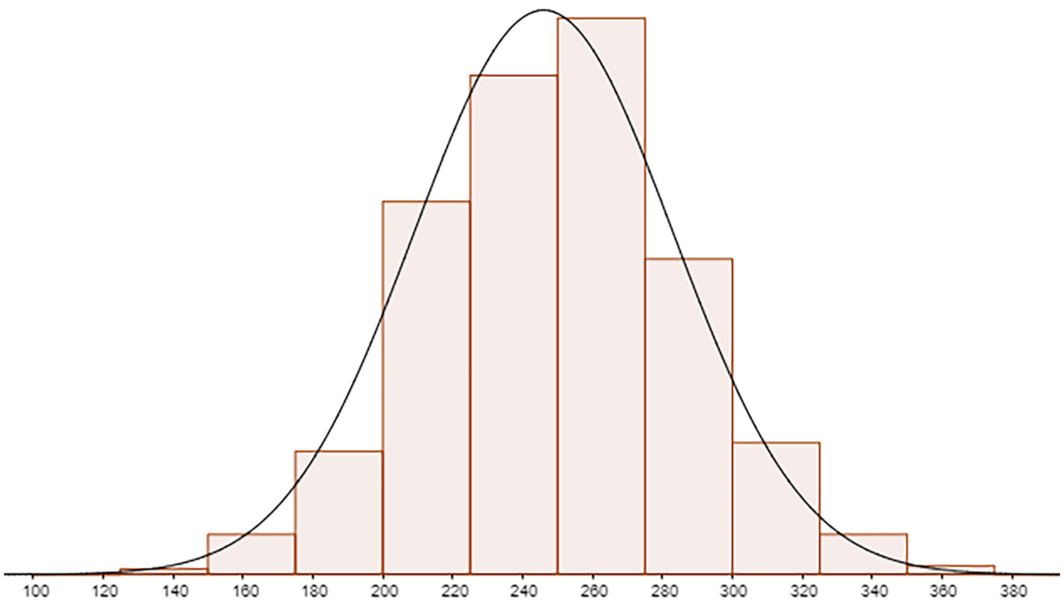


Figure 5.2: *Sampling distribution*

Central Limit Theorem

Central Limit Theorem states that the probability distribution of the sample taken from a population will have the mean approximately normally distributed with the mean of the population and variance approximately normally distributed with the variance of the population. We can see this in *Figure 5.3*:

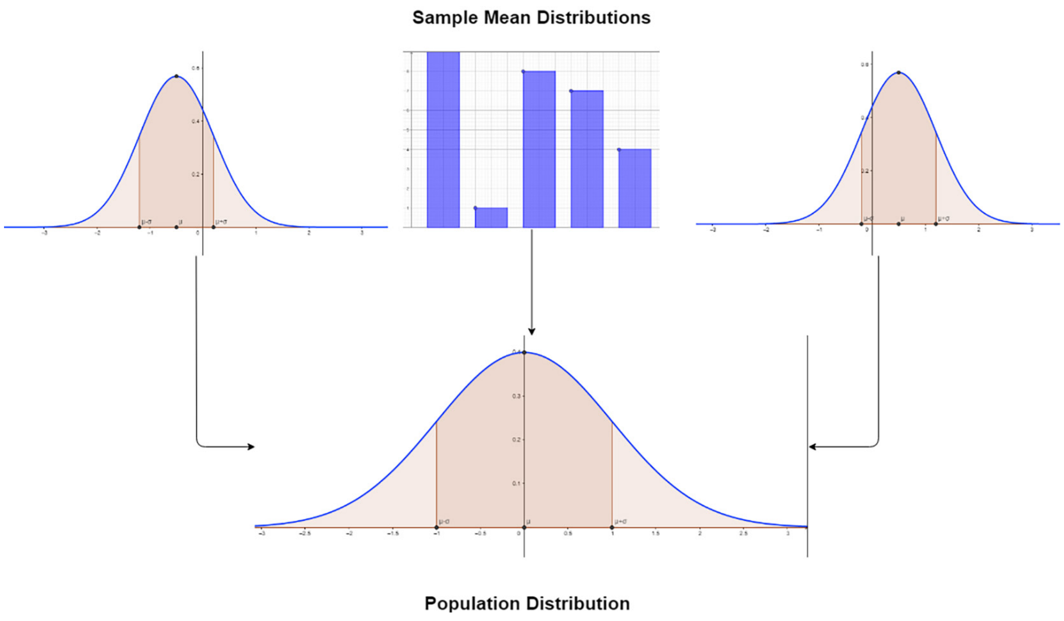


Figure 5.3: *Sample mean distribution taken from population distribution*

Suppose,

Sample mean = $\bar{X}$

Random sample size = n

Population mean = μ

Population variance = σ^2

Random sample = $X_1, X_2, X_3 \ldots X_n$

So,

$$\bar{X} = \frac{X_1 + X_2 + X_3 \ldots + X_n}{n}$$

$$\mu\bar{x} = \frac{\mu + \mu + \ldots \mu}{n} = \mu$$

$$\sigma^2\bar{x} = \frac{\sigma^2 + \sigma^2 + \ldots \sigma^2}{n^2} = \frac{\sigma^2}{n}$$

And, as per Central Limit Theorem

$$Z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Example: A group of salespersons receives benefits of $110 per week with a standard deviation of $20. If a random sample of 25 people is taken, what is the probability that the mean benefit of the salespersons will be greater than $120 per week?

In this example, we have taken a random sample of 25 people, but we don't have any idea about their probability distribution. So, we have to use the Central Limit Theorem to solve the problem:

$$\bar{X} = 120$$

$$\mu = 110$$

$$\sigma = 20$$

$$n = 25$$

$$\therefore z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}} = \frac{120 - 110}{\frac{20}{\sqrt{25}}} = 2.5$$

If we look at the z-table for the value of 2.5, we get a p-value of 0.4938. This talks only about one half of the z-curve. If we add the other half, the p-value becomes 0.9938.

So, there is a 99.38% chance that the mean will be greater than $120 a week.

The following Python code can find the solution for this problem. We have created a function that calculates the z-score of the sample, and then we will use the same to find the final value:

```
import numpy as np

defsample_z_score(mean_old_sample, mean_sample,std, size):

tmp = (mean_old_sample-mean_sample)/(std/np.sqrt(size))

   return tmp

answer = sample_z_score(120,110,20,25)
```

Estimators having bias component

Suppose θ is the parameter of interest, and we want to get an estimate for it. Let's represent this estimation of θ∅. When we use this estimation, we have to make the expected value of this estimation equal to θ. If we can do so, we call the estimation ∅ as an **unbiased Estimator**. In other words, we can say that the mean of the probability distributions of both θ and ∅ will be the same; ∅ is an unbiased Estimator. So:

$$E(\varnothing) = \theta$$

However, if there is any bias present, the preceding equation can be rephrased as:

$$E(\varnothing) - \theta = \text{bias}$$

The variance of a point estimate

Suppose there are two unbiased Estimators— φ_1 and φ_2 —for a parameter θ. Also, suppose that we get the following figure if we draw the probability distribution of these two Estimators:

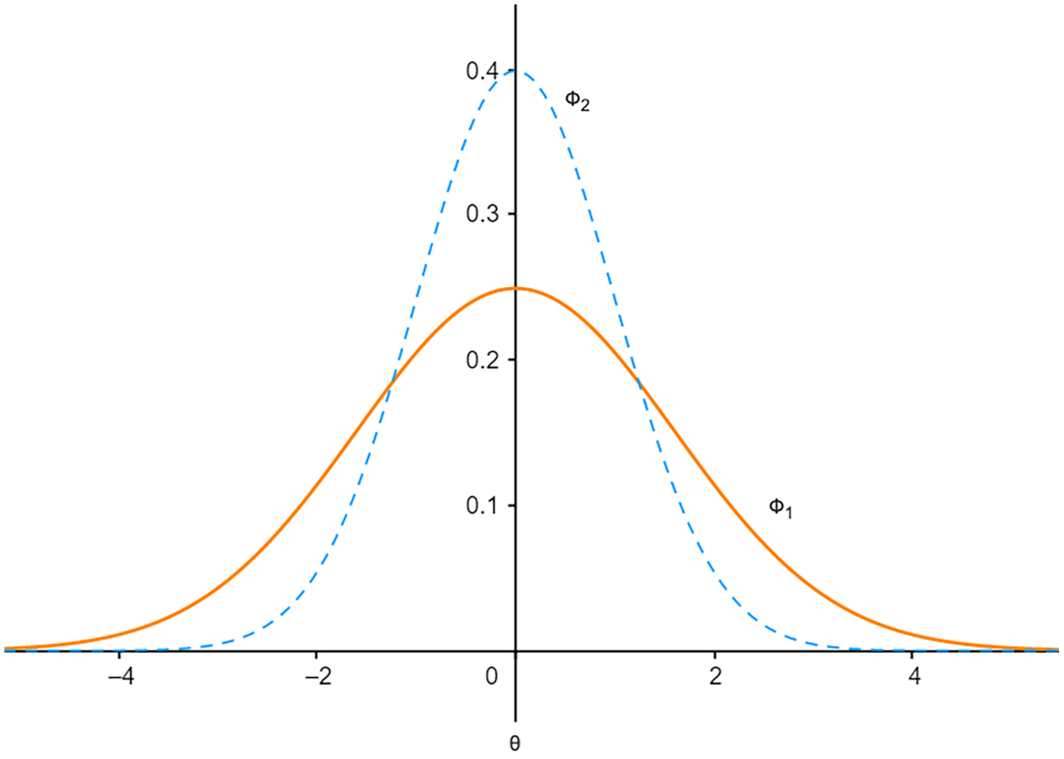


Figure 5.4: Comparing distribution of unbiased Estimators of two samples taken from the same population

The smaller variance of an unbiased Estimator means it will be closer to the distribution of θ. In the preceding diagram, we can see that the variance of φ_1 is greater than the variance of φ_2. Any unbiased Estimator having the smallest variance is termed as **Minimum Variance Unbiased Estimator** (**MVUE**). When we want to estimate something about the population, our unbiased Estimator should be MVUE.

Standard Error of Estimator

We now know that for any parameter θ, the estimates that we determine is called as the **point estimate**. But how accurate are we in the estimation? What is our precision? To know the answer to these questions, we use a metric called **Standard Error**. It is represented by:

$$\sigma_{\varphi} = \sqrt{V(\varphi)}$$

So, if the σ of the population is given, then:

$$\sigma_{\varphi} = \frac{\sigma}{\sqrt{n}}$$

However, if it is not given:

$$\sigma_{\varphi} = \frac{\sigma_{sample}}{\sqrt{n}}$$

In the following example, we'll see how precise our calculation of the point estimate will be.

Example: Suppose a sample of students got the following marks in an examination: 41.60, 41.48, 42.34, 41.95, 41.86, 42.18, 41.72, 42.26, 41.81, 42.04. What will be the precision of its point estimate of mean in terms of standard error?

In this example, the points estimate is the sample mean, $\overline{X} = 41.924$

So, the standard error would be:

$$\frac{\sigma_{sample}}{\sqrt{n}} = \frac{0.284}{\sqrt{10}} = 0.0898$$

We can say that the standard error is around 0.2% of the sample mean, which means it's pretty precise.

The following is the Python code where we have made a function that calculated the standard error. We'll use this function to calculate the standard error for the preceding example:

```
import numpy as np

def standard_error(std, size):

tmp = std/np.sqrt(size)

   return tmp

answer = standard_error(0.284,10)
```

Mean Squared Error of Estimator

Till now, we have only talked about unbiased Estimators. What if we have biased Estimators? In that case, we check the precision using **Mean Squared Error** (**MSE**) and not Standard Error. It is represented by:

$$MSE = E(\varphi - \theta)$$

Also, we can use the following formula if we want to check the relative efficiency of two biased Estimators:

$$efficiency = \frac{MSE(\varphi_1)}{MSE(\varphi_2)}$$

Methods to Determine Point Estimates

The following are the methods that we can use to find out point estimates:

1. Method of moments
2. Maximum likelihood method

We will discuss the first two estimation methods, as Bayesian methods are out of the scope of this book.

Method of Moments

The expected value of any random sample from a group of samples is termed as a moment. Mathematically, it is represented as:

$$Moment,\ E(x) = \overline{X}$$

Where *k* represents the k^{th} sample from a group of random samples $X_1, X_2, \ldots X_n$ taken from a random variable *X* having *f*(*x*) as its probability distribution function. This can be expanded as follows:

$$Moment, E(x^2) = \frac{1}{n}\sum_{i=1}^{n} x_i^k$$

Suppose we want to estimate the mean and variance point estimates for a normal distribution using the moments method. The first thing that we must know for a normal distribution is the following:

$$E(x) = \mu$$

$$E(x^2) = \mu^2 + \sigma^2$$

So:

$$\overline{X} = \mu$$

$$x^2 + \sigma^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^k$$

Simplifying the preceding two equations, we get:

$$\overline{X} = \mu$$

And:

$$\sigma^2 = \sum_{i=1}^{n} \frac{(x_i - \overline{x})^2}{n}$$

We can easily get the point estimates using the preceding two equations.

Example: 26, 33, 65, 28, 34, 55, 25, 44, 50, 36, 26, 37, 43, 62, 35, 38, 45, 32, 28, 34.

Find the point estimates of mean and variance of the preceding values, assuming they follow the normal distribution.

$$\mu = \overline{X} = \frac{(26+33+65+28+34+55+25+44+50+36+26+37+43+62+35+38+45+32+28+34)}{20} = 38.8$$

$$\sigma^2 = \sum_{i=1}^{20} \frac{(x_i 38.8)^2}{20} = 129.96$$

Maximum Likelihood Method

In this method, we first define a likelihood function, whose values are dependent on the value of θ. After this, we try to maximize this function. Whichever value of θ gives us the maximum value, we consider it as the best estimate for the population. The likelihood function can be represented as follows:

$$L(\theta) = f(x_1 : \theta).f(x_2 : \theta)\ldots f(x_n : \theta)$$

Where, x_1, x_2, …x_n are the observed values of random variable X.

Suppose we want to estimate the mean point estimate for an Exponential Distribution using the MLE method. We can define the loss function of Exponential Distribution as follows:

$$L(\lambda) = \lambda^n e^{-\lambda} \sum_{i=1}^{n} x_i$$

If we take *log* on both the sides, we can get *log* of the function:

$$\log_e L(\lambda) = n\log_e \lambda - \lambda\sum_{i=1}^{n} x_i$$

To get the maximum of the preceding function (Maximum Log-Likelihood), we will differentiate and equate it to 0:

$$d$$

$$\therefore \lambda = \frac{1}{\overline{X}}$$

So, for the exponential distribution, when we calculate the point estimate of the mean using the MLE method, the population mean estimation comes as the reciprocal of sample mean. If we take the same example discussed for the moment method and assume that the data follows exponential distribution, the population mean will be:

$$\lambda = \frac{1}{\overline{X}} = \frac{1}{38.8}$$

The following is the Python code for finding the mean (parameter estimate) of ages, given the population and the sample:

```
import numpy as np

import scipy.stats as stats

import random


#Generating Sample Population Ages have to mean like35 and 150000 data
points. Package scipy is used for dummy data generation. We have passed
the mean, the size, and the shift provided to the distribution.

population_ages1 = stats.poisson.rvs(loc=18, mu=35, size=150000)


#Generating Sample Population Ages having mean as 10 and 100000 data
points

population_ages2 = stats.poisson.rvs(loc=18, mu=10, size=100000)


# Concatenating the two samples to generate one combined sample

population_ages = np.concatenate((population_ages1, population_ages2))
```

```
#Let’s take 500 sample ages from the above population decided

sample_ages = np.random.choice(a= population_ages, size=500)


print(“The Sample Mean is”, sample_ages.mean())

print(“The Population Mean is”, population_ages.mean())

print(“The Difference between both the means is: “, population_ages.
mean() - sample_ages.mean())


std = np.std(sample_ages)

se = standard_error(std, len(sample_ages))

print(“Standard Error for the Sample is”, se)

print(“************************”)

print(“%f percentage of the mean is the Standard Error, therefore it is
quite precise” %((se)/sample_ages.mean()))
```

Confidence Intervals

Now that we know about the concepts of parameter estimation and point estimates, we must understand confidence intervals. In this section, we will understand confidence intervals and why they are important for point estimates.

As we already know, we find point estimates of the population parameter using a sample. But we don’t necessarily need to get the best sample. Maybe we have not included all the varieties of data points while taking a sample. Also, the properties of all samples will be different, so the point estimates for different samples may vary every time.

So, we use confidence intervals to solve this issue. We get a range of values that helps us decide the range of our point estimates. It also indicates a measure of the likelihood of whether the sample includes the unknown population proportion. Let’s understand this with the help of an example.

Suppose I want to know how many engineers who know machine learning also know deep learning. I selected a sample and got to know that 60 out of 80 engineers knew deep learning. So, we can say that around 75% of the engineers know deep learning. However, can we say that with surety? Because we have not interviewed all the engineers, and it may not represent the exact population estimate. So, we need to add a margin of error.

We can add a margin of 5% and say that 75% of engineers know deep learning, give and take 5%. Mathematically, we can represent this as 0.75 ± 0.5. Now, if someone asks how confident you are in stating these figures, you might say that you are 90% confident that the engineers knowing about deep learning lie between 70% to 80%. If they want you to increase your confidence, you'll need to increase your interval as well. So, you can say that you are 95% confident that 65% to 85% engineers knowing about deep learning.

Formally, a confidence level represents the samples that will contain the parameter of interest if a large number of different samples are obtained. For example, 95% confidence implies that if 100 different samples are taken, we will expect 95 of the samples to include the parameter of interest and 5 to not include it.

So, we can say that we need a confidence interval to portray our figures more effectively with a point estimate. This will help us be confident during our presentation of the estimates, and the attendees will get a much greater understanding of your findings. The following figure represents the level of confidence that we discussed:

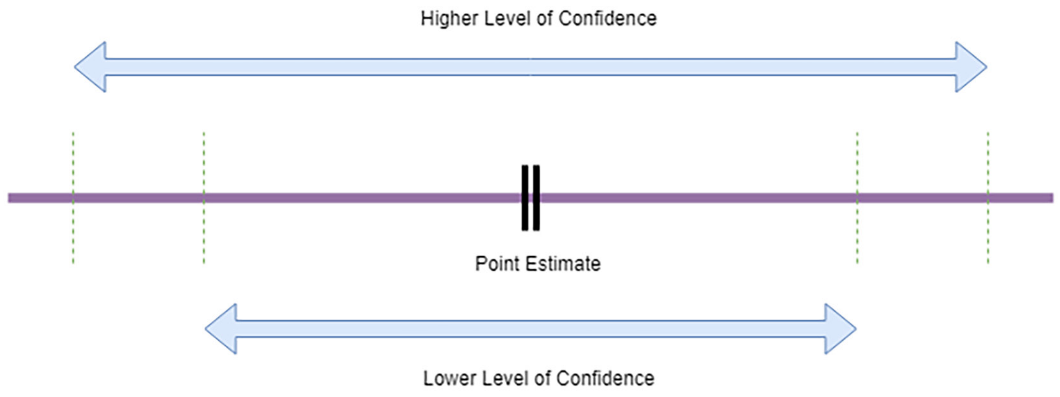


Figure 5.5: *Comparing the higher and lower level of confidence*

Conclusion

In this chapter, we saw how different methods are used to determine parameters like the mean or variance of the population if the same information is given for the sample. This chapter is important because we work with samples and not with the entire population most of the time, as gathering data for the entire population is next to impossible. Once we are done with the analysis on the sample, we would also want to know whether the same things hold for the population as well. So, we need to determine a few parameters to validate our assumptions about the sample, and that is where parameter estimation helps us.

As we are done with most of the peripheral parts of statistics, it's time to look at what every statistician uses in their daily life: hypothesis testing. From the next chapter, onward, we will discuss this concept and look at different types of statistical tests and their applications.

Chapter 6
Hypothesis Testing

In the previous chapters, we looked at how to get samples from a population and how to estimate the parameters based on the samples. But what if we want to conclude a few things about the entire population based on a selected sample? In that scenario, we use the concept of **Hypothesis Testing**. This is the use of the information present in the sample to determine whether it holds for the entire population.

In this chapter, we will look at Hypothesis Testing in detail. We will first try to understand the concept and related terminologies, and then we'll move on to understanding the different types of statistical tests. As usual, we will be trying to understand all the concepts with the help of examples and their practical applications in Python.

Structure

- Hypothesis
- Hypothesis testing
- Types of hypothesis
- Z-test
- T-test
- Chi-Square test

Objective

When it comes to statistical research, hypothesis testing plays a major role. This section will touch upon the basics of hypothesis testing and how they are applied in statistical research. We will then look at the advanced methods used during the tests.

Hypothesis

Hypothesis or **Statistical Hypothesis** is a supposition or proof to an assumption that is accepted to interpret certain events of the phenomenon that provide guidance for analysis or investigation. This means when we want to prove something, it will first be considered an assumption. We will find evidence to support or reject this assumption. Once we are done, we can determine whether our assumption was correct.

Hypothesis Testing

Hypothesis testing is the process of accepting or rejecting the statistical hypothesis using methods and tools in statistics and mathematics with programming languages like R and Python. Before we move on to understanding the testing part, we must know what a confidence interval is.

Confidence Interval

The procedure for relying on the testing outcomes is presented in a report in terms of the range of values called **Confidence Interval**. This always specifies a level of confidence regarding the procedure and the results.

This concept is the most useful for statistical inference, as most decision-making problem types, tests, or experiments in the technical environment are formulated as hypothesis testing problems. Moving forward, a close connection is established between hypothesis testing and confidence intervals. In the process of data analysis of the experiment, an individual must consider statistical hypothesis testing and confidence interval as the fundamental methods, for instance, while comparing the population mean to a specific value. The essence of these methods is crucial during the practice of complex designs and while implementing exceptional ideas.

Experiments and populations are represented as probability distributions. As a result, they can also be thought of as a statement about the probability distribution of random variables. One or more parameters of these distributions are involved in hypothesis testing.

In order to determine the facts of the statistical hypothesis, it would be recommended to examine the entire population of variables and data. Since this is unrealistic, the process is evaluated by typically examining a sample containing data and random variables from the population. If the analysis of sample data is not consistent, the hypothesis is rejected. It is important to remember that hypotheses are always statements about the population or distribution under study, and not statements about the sample.

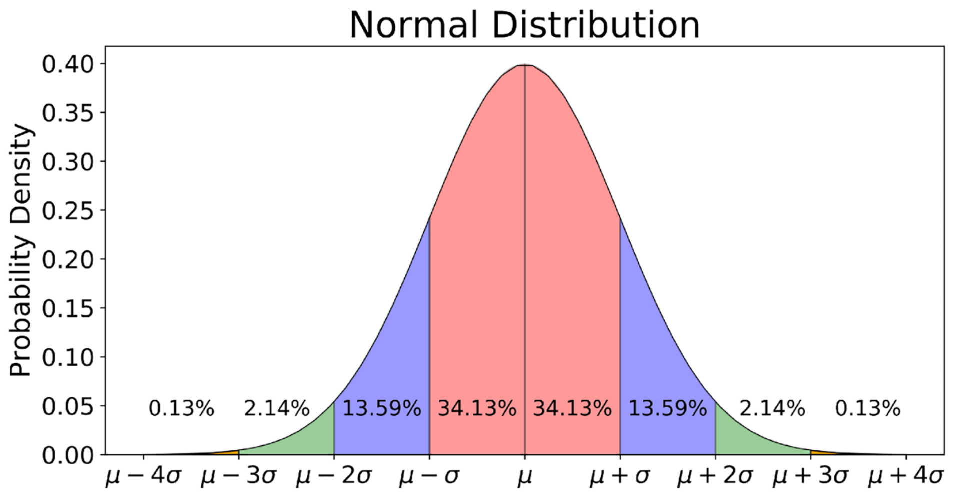


Figure 6.1: *Data partition in normal distribution*

We have seen the preceding diagram of a normal distribution curve in *Chapter 4: Probability*. We can refer to the previous example to properly understand the concept of confidence interval. Whenever the data follows a normal distribution pattern, we are sure that around 68% of data will lie in one standard deviation difference, 95% of data will lie in two standard deviation differences, and so on. This surety can be termed as the **Confidence Interval** (**CI**). In general, a CI of 0.95 refers to being 95% confident, while a CI of 0.68 refers to being 68% confident.

Types of Hypothesis

Now that we know about confidence intervals, let's proceed with the details of hypothesis testing. Before starting the testing, we must be sure about our assumption. That's why we have the following two kinds of hypothesis:

- Null hypothesis
- Alternate hypothesis

Null Hypothesis

In the process of hypothesis testing, **null hypothesis** is the standard statement that is made in the event of an exercise as there is no relationship among the groups of variables.

For instance, in the event of rolling dice, one would assume that rolling the dice would result in the showing up of number 3 half the time. This is denoted by Ho and is expressed as Ho:.

Alternative Hypothesis

Alternative hyporthesis is considered a contradictory statement to the null hypothesis and is represented as *Ha*. Alternative hypothesis states and results in providing proof to display a relation between the variables.

For instance, in the example of rolling a die, the showing up of a number depends on factors that are related to how the die is rolled. An alternative hypothesis is expressed as Ha: $\mu_1 \neq 0.5$.

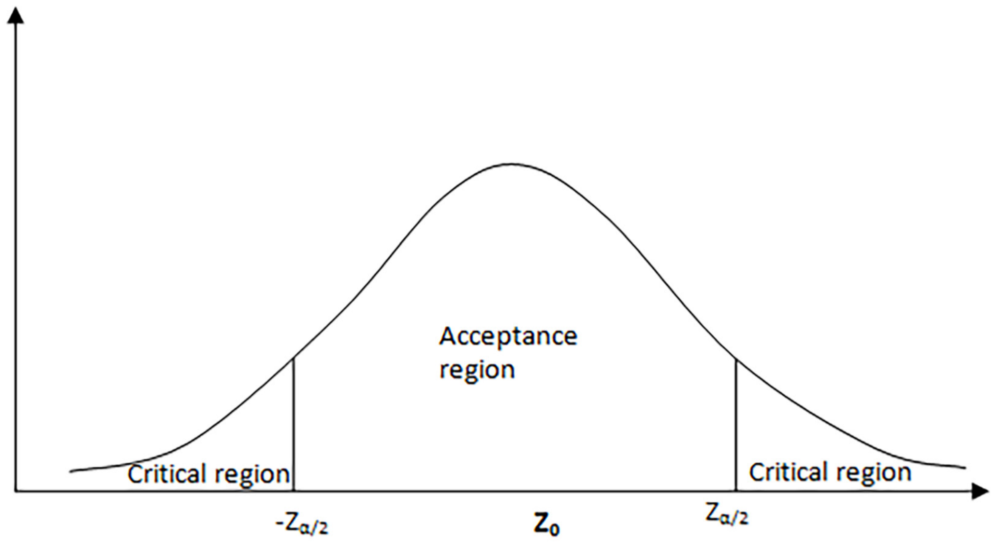


Figure 6.2: *Acceptance and Rejection Region*

The graph in the preceding figure represents the plotting of the variables and the results of the exercise. During the exercise, if the sample mean is in the range where it doesn't conflict with an alternative hypothesis, the range is termed as **Acceptance Region**. If the sample mean of the population falls in the region of the alternative hypothesis, that region is termed as **Critical Region**.

The threshold values of the boundaries are represented by Z. The normal value or the mean value is denoted by *Zo,* and boundary values are denoted by $Z\alpha/2$.

While performing tests in an environment, given a specific number of samples, it is possible to arrive at one of the following two decisions:

1. Accepting that the null hypothesis is true despite the actual result being that the null hypothesis is to be rejected. This scenario is called Type 1 Error.
2. Failing to accept the null hypothesis and accepting the alternative hypothesis despite the actual result being in favor of the null hypothesis. This is called a Type 2 Error.

These errors are called **Decision Errors**. The probability of committing a Type 1 Error is called **Significance Level** and is represented as α. A Type 2 Error occurs when one fails to reject the null hypothesis. It is termed as the power of hypothesis testing and is represented as β.

One should achieve a 100% confidence level to reject a null hypothesis despite its value. However, practically, it would be impossible to achieve 100% probability, or we can say Significance Level, so a significance level of 95% and a difference of 5% can be maintained. Similarly, the power of hypothesis is the probability of rejecting the null hypothesis when the alternative hypothesis is true.

P-Value

It is the least level of significance, which would be based on rejecting a null hypothesis. It is also considered as the calculated probability as it is a process of finding observed values or extreme values when the null hypothesis of an exercise is true.

While testing the hypothesis of an experiment, we come across situations where the mean value of a sample cannot be clearly specified. In such cases, we define the following scenarios:

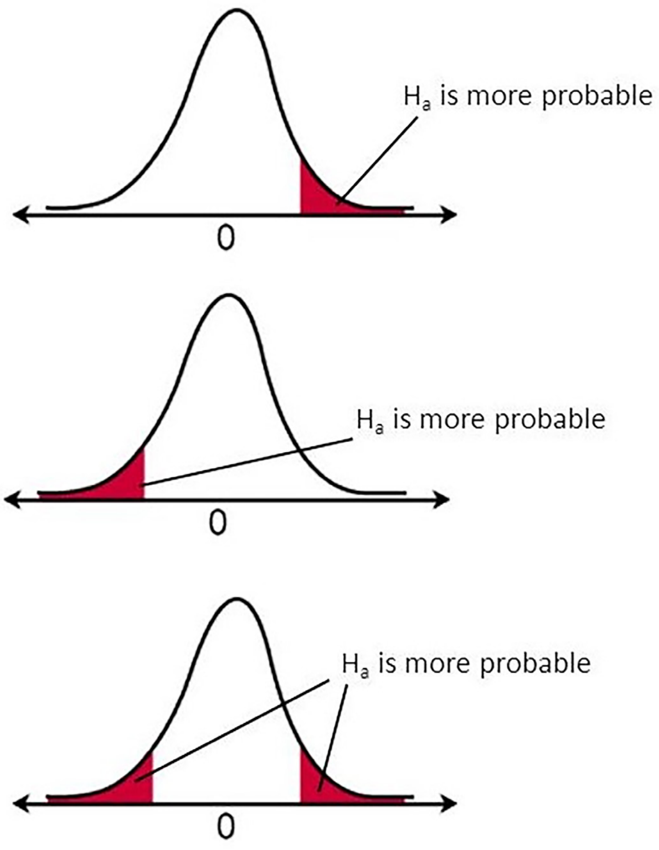


Figure 6.3: *One-tailed and two-tailed test*

One-tailed test

It is a test of a statistical experiment where the critical area of distribution is only on one side of the acceptance region. It can be greater or less than the null hypothesis. Suppose the null hypothesis of an experiment states that mean is less than or equal to 10, and the alternative hypothesis suggests that the mean is greater than 10. The rejection region for the alternative hypothesis would be on the right side of the acceptance region.

The first two graphs in *figure 6.3* represent one-tailed test.

Two-tailed test

It is a test of a statistical experiment where the critical area of distribution is on both sides of the acceptance region. It can be less than or greater than the null hypothesis.

In this case, if the null hypothesis states that the mean of the sample is equal to 10, the alternative hypothesis would be either less than 10 or greater than 10, presenting the critical region on either side of the acceptance region.

The last graph in *figure 6.3* represents a two-tailed test.

Steps in hypothesis testing

Once we are clear about one-tailed and two-tailed tests, we must know how to test our assumptions. The following is a sequential list of steps for a hypothesis testing approach:

1. Stating a parameter of interest in the problem statement
2. Defining the null hypothesis *Ho*
3. Defining the alternative hypothesis *Ha*
4. Determining a proper testing parameter
5. Stating a criterion for processing the null hypothesis and alternative hypothesis
6. Determining the necessary steps of computing the testing parameters and various hypotheses
7. Drawing conclusions

Use Case

Let's take the problem statement of rolling a die. The task is to roll the die 100 times and test the probability of a number being displayed within the duration of the experiment. Now, this experiment is repeated for a certain number of times, depending on the test being performed. Let's perform hypothetical tests for this use case and analyze how the null hypothesis and alternative hypothesis vary.

Now, let's apply some of the mentioned processes on some of the famous statistical tests. These are as follows:

1. Z-test
2. T-test (Student T-test)
3. Chi-Square test

Z-test

Z-test determines that the means of two different populations are different. It assumes that the variance of the population is known and is distributed normally.

$$Zo = \frac{X1 - \mu o}{\sigma / \sqrt{n}}$$

Figure 6.4

Where *X1* - sample mean

μo - actual mean

σ - variance

N – number of samples

Let's understand this concept with the help of an example:

Example: Boys of a certain age are known to have a mean weight of $\mu = 60$ kgs. A complaint is made that boys at a children's home are underfed. As one bit of evidence, $n = 25$ boys (of the same age) are weighed and found to have a mean weight of $x = 55$ kgs. It is known that the population's standard deviation σ is 6.5. Based on the available data, what should be concluded concerning the complaint?

$NullHypothesis(H_0) = Boysarenotunderfed = (\mu = \mu_0)$

$AlternateHypothesis(H_A) = BoysareUnderfed = (\mu \neq \mu_0)$

Now, we know that:

$$Z_0 = \frac{x - \mu_0}{\frac{\sigma}{\sqrt{n}}}$$

Based on the data provided:

$$x = 55,\ \mu_0 = 85,\ \sigma = 6.5,\ n = 25$$

So, test statistic Z_0 is:

$$\frac{55 - 85}{\frac{6.5}{\sqrt{25}}} = -23.07$$

If we decide our p-value to be 0.05, it means our confidence level will be 95%. If we look at the z-table (explained in *Chapter 4: Probability*), we get our *T-Critical* as-*1.645*. As our *Z-Statistics* is less than our *Z-Critical*, we will reject the null hypothesis. So, we can conclude that the boys are underfed, as the evidence supports this hypothesis.

Python implementation:

```
import numpy as np

import pandas as pd

import scipy

from scipy import stats

mu=85

x_=55

se=6.5

#Calculating the Z value to complete the z testing

z_stat=(x_-mu)/(se/np.sqrt(25))

#calculating the p-value

p_val=2*(1-stats.norm.cdf(z_stat))

print('Z=',z_stat,"pValue=",p_val)
```

T-test

T-test, which is also known as Student's T-test, was first formulated by *William Sealy Gosset*, who has maintained a pen name *Student*.

A T-test is a statistical test that determines the significant difference between the mean values of two groups. For performing a T-test, it is to be assumed that the data is normally distributed and the variance is unknown.

The mathematical formula for this test is as follows:

$$To = \frac{X1 - \mu o}{S/\sqrt{n}}$$

Figure 6.5

Where *X1* – sample mean

μo – actual mean

S – standard deviation

N – number of samples

There are three variants of T-test:

- One-sample T-test
- Two-sample T-test
- Paired T-test

One-sample T-test

It is used to identify the process to generate a sample of observations with a specific mean. This is a parametric test that determines whether a computed mean or statistical mean is similar to the observed mean.

Let's understand this concept with the help of an example:

Example: A car company claims that its *Super Spiffy Sedan* averages 31 mpg. You randomly select eight *Super Spiffies* from local car dealerships and test their gas mileage under similar conditions. You get the following mpg scores:

MPG: 30 28 32 26 33 25 28 30

Does the actual gas mileage for these cars deviate significantly from 31 (alpha=.05)?

$NullHypothesis(H_0) = CarsDon'tDifferinAverage = (\mu = \mu_0)$

$AlternateHypothesis(H_A) = CarsDifferinAverage = (\mu \neq \mu_0)$

Now, we know that:

$$T_0 = \frac{x - \mu_0}{\frac{s}{\sqrt{n}}}$$

Based on the preceding data:

$x = 29,\ \mu_0 = 31,\ \sigma = 2.775,\ n = 8$

So, test statistic Z_0 is:

$$\frac{29 - 31}{\frac{2.775}{\sqrt{8}}} = -2.04$$

Since our p-value is 0.05, our confidence level will be 95%. If we look at the t-table (explained in *Chapter 4: Probability*), we get our T-Critical as-2.093. As our absolute value of T-statistics is greater than our T-critical, we will reject our null hypothesis.

So, we can conclude that cars differ in their average mileage, as the evidence supports this hypothesis.

The code for computing a sample T-test is as follows:

```
import numpy as np

import pandas as pd

import scipy

from scipy import stats


x=[30,28,32,26,33,25,28,30]

mu=31

t_critical=2.093#foralphalevel0.05

x_=np.array(x).mean()

#subtract 1 from N to get unbiased estimate of sample standard deviation

N=len(x)


t_stat=(x_-mu)*np.sqrt(N)/np.array(x).std(ddof=1)

print("t-statistic:",t_stat)


#aonesamplet-testthatgivesyouthep-valuetoocanbedonewithscipyasfollows:

t,p=stats.ttest_1samp(x,mu)

print("t=",t,",p=",p)
```

Two-sample T-test

This is also known as an independent sample T-test and compares the mean of two independent samples to gather statistical evidence for determining the difference or similarity between the groups.

$$\frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Figure 6.6

We can get the t-statistic using the preceding formula, which we can later use to accept or reject the null hypothesis. Let's understand this concept with the help of an example:

Example: Yamuna has balloons from two different brands. She wants to test the durability of each brand by measuring the volume of water that can be pumped into the balloons before they burst. Here's a summary of the results:

	Brand A	**Brand B**
Mean	10.2L	11.8L
Standard deviation	1.2L	0.9L
Number of balloons	8	8

Table 6.1

Yamuna wants to use these results to carry out a two-sample *t*test to determine whether the mean volumes are significantly different for the two brands. Assume that all conditions have been met.

$NullHypothesis(H_0) = MeanVolumesaresame = (\mu = \mu_0)$

$AlternateHypothesis() = MeanVolumesaredifferent = (\mu \neq \mu_0)$

Since we already know the formula of two means *t-test*, we will be replacing it with the following values:

Based on the preceding data:

$$x_1 = 10.2, x_2 = 11.8\sigma_1 = 1.2, \sigma_2 = 0.9, n_1 = 8, n_2 = 8$$

So, the test statistic is:

$$t = \frac{(10.2 - 11.8) - 0}{\sqrt{\frac{1.2^2}{8} + \frac{0.9^2}{8}}}$$

$$= -3.016$$

Figure 6.7

If we decide our *p-value* as *0.05*, this means our confidence level will be 95%. If we look at the *t-table*, we get our `t_critical` at `1.677`. As our T-statistics is greater than our T-critical (Absolute value), we will reject the null hypothesis. So, we can conclude that the mean volume significantly differs, as the evidence supports this hypothesis.

```
import numpy as np

import pandas as pd

import scipy

from scipy import stats


t_critical=1.677#for alpha=0.05


dof_x1=7

dof_x2=7


dof=dof_x1+dof_x2


std_x1=1.2

std_x2=0.9

x1_=10.2

x2_=11.8


SE=np.sqrt(((dof_x1*std_x1**2+dof_x2*std_x2**2)/dof))


t_stat=(x2_-x1_)/var*np.sqrt(1/len(x1)+1/len(x2))

print("t-statistic",t_stat)


t,p=stats.ttest_ind(x2,x1,equal_var=True)

print("t=",t,",pvalue=",p)
```

Paired T-test

Also known as the **Dependent Sample** test, it is used to determine whether there is a similarity between two groups of observations or samples. In this test, each feature or entity is calculated twice, generating a pair of observations for hypothesis testing. In a paired T-test, generally, the same experiment is performed twice, like before and after the situation. Let's understand this concept with a dummy Python example:

```
import numpy as np

import pandas as pd

import scipy

from scipy import stats


x1=[39,45,21,11,38,36]#Examination Marks Before Vacations

x2=[22,21,13,13,49,20]#Examination Marks After Vacations

t,p=stats.ttest_rel(x2,x1)

print("t=",t,",p value=",p)
```

Chi-Square test

A chisquare(χ2) test is used to measure the computed mean with the actual mean of the observed data, assuming that the data is large enough and contains independent variables. It is formulated as follows:

$$\chi^2 = \frac{\sum (Oi - Ei)^2}{Ei}$$

Figure 6.8

Where:

Oi – observed values

Ei – expected values

Test of Goodnessoffit

Chi-Sqaure test for goodnessoffit is implemented to identify how a phenomenon's actual values differ from the expected values. This is an on-parametric test and is used to compare the sample distribution of observed values and expected probability distribution. In this test, the data used for testing is divided into groups, and the number of points that fall under the range is compared with the expected number of values in each group. It determines how well theoretical distribution like normal, binomial, or Poisson distribution fits empirical distribution.

Example: For testing with the **Chi-Square** test, let's consider the problem of rolling dice and estimate the probability of displaying two specific numbers—2, 3—if we roll the dice 20 times. Let's consider two categories: one in the case of rigged dice and the other in the case of fair dice:

```
iimport numpy as np

import pandas as pd

import scipy

from scipy import stats


a=['rigged',3,4]

b=['fair',2,5]


Observed_Values=np.append(a[1:3],b[1:3])

print(Observed_Values)


Expected_Values=np.outer(Observed_Values,1.5)#.T[0]

print(Expected_Values)

df=1

print("DegreeofFreedom:-",df)


alpha=0.05


chi_square=sum([(o_v-e_v)**2./e_vforo_v,e_vinzip(Observed_Values,Expect-
ed_Values)])


#chi_square_statistic=chi_square[0]+chi_square[1]

print("chi-square statistic:-",chi_square)


#implementing chisu qare test using scipy module

chi_square_stat=stats.chisquare(Observed_Values,[4.5,6,3,7.5])
```

```
print("ChiSquareStatistic=",chi_square_stat[0])


#critical-value

critical_value=scipy.stats.chi2.ppf(q=1-alpha,df=df)


#p-value

p_value=1-scipy.stats.chi2.cdf(x=chi_square_stat,df=df)


print('Criticalvalue=',critical_value,'Pvalue=',p_value)
```

Independence test

The Chi-Square test of independence is used to identify the relationship between two categorical variables. It is used to identify the frequency of each category for one nominal variable compared to a second variable across all the categories.

Example: Displaying specific numbers, that is, 2, 3, out of the activity of rolling dice 20 times. Let's consider two categories: one in the case of rigged dice and the other in the case of fair dice:

```
a=[‘rigged’,2,3,5]

b=[‘fair’,4,4,8]

c=[‘Sum’,6,7,13]

df=pd.DataFrame([a,b,c])


df.columns=[‘dice’,’2’,’3’,’Sum’]

df.set_index(‘dice’,inplace=True)

obs=np.array([df.iloc[0][0:2].values,df.iloc[1][0:2].values])


observed_values=np.append(df.iloc[0][0:2].values,

df.iloc[1][0:2].values)
```

```
print(“DegreesofFreedom:”,2)


exp=scipy.stats.chi2_contingency(obs)[3]

exp_values=np.append(exp[0],exp[1])


chi_squared_statistic=((observed_values-exp_values)**2/exp_values).sum()

print(‘Chi-squaredStatistic=’,chi_squared_statistic)


#implementing chi square test using scipy module

chi_stat,p_val,dof=scipy.stats.chi2_contingency(obs)[0:3]

print(‘Chi_square_stat’,chi_stat,”p-Value”,p_val,”DegreeofFreedom”,dof)
```

Conclusion

In this chapter, we learned about hypothesis testing and the different types of statistical tests. We talked about hypothesis and its types, and we explored the process where we define the null and alternate hypotheses and then try to generate enough evidence to help us accept or reject the null hypothesis.

Later, we looked at different tests. We started with T-tests and its different kinds and moved on to Z-tests. We also walked through Chi-Square tests and their applications. In the next chapter, we will talk about ANOVA tests, a type of statistical test. In this method, we will analyze the variance present in different samples.

Chapter 7
Analysis of Variance

In the chapter of hypothesis testing, we looked at different tests for comparing the means of two populations. For this, we looked at tests like t-test, z-test, chi-square test, and so on. In this chapter, we'll be exploring the mean comparison of more than two populations. This procedure is called Analysis of Variance (ANOVA). For example, if we want to find out whether different career choices like doing a Masters in Technology, getting a job, starting your own business, or doing an MBA affect average annual salary. In this problem statement, we can define the null hypothesis as *the mean salary is the same for all the career choices*. An alternative hypothesis can be that *at least one population mean salary is different from others*. We can test this hypothesis using the ANOVA approach, as there are more than two populations.

Structure

This chapter will cover the following topics, point-wise:

- Introduction to ANOVA
- ANOVA tests
    - o One-way
    - o Two-way
    - o MANOVA
- Other statistical tests related to the different ANOVA tests

Objective

The objective of this chapter is to apply hypothesis testing on three or more samples, which may or may not be related to each other.

Introduction to ANOVA

In ANOVA, the null hypothesis is always defined as the means of populations are equal, while the alternative hypothesis is defined as at least one population mean is different from the others:

$$H_0 : \mu_1 = \mu_2 = \mu_3$$

$$H_A : \mu_1 = \mu_2 \neq \mu_3 \text{ or } \mu_1 \neq \mu_2 = \mu_3$$

Here's the diagram showing null and alternative hypotheses for the three population means defined in the preceding equations:

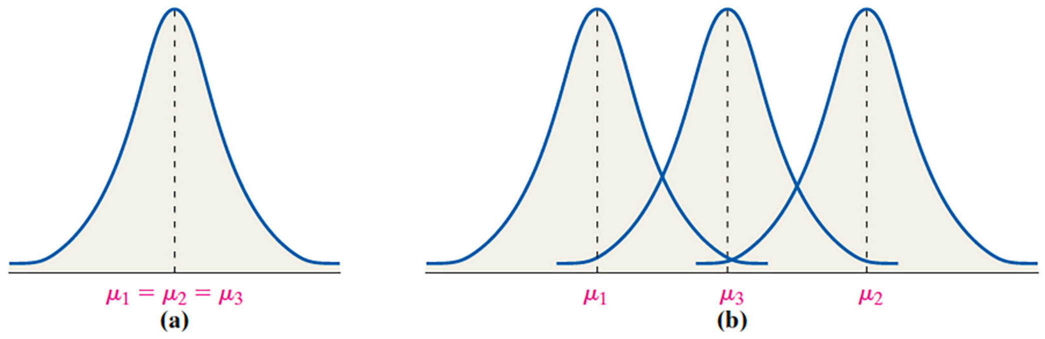


Figure 7.1: *Samples having the same means vs. samples having different means*

One-way ANOVA test

In the career choice example in the previous section, we made a comparison using a single factor *salary*. When we use a single factor for the comparison of three or more means, the test is termed as 1-factor ANOVA or one-way ANOVA test. In this section, we'll look at this test in further detail.

When we talk about one-way ANOVA tests, a few assumptions that need to be followed are:

- The sample chosen for the test should be completely random, with no bias. This design of an experiment is called randomized design.
- The response variable should be quantitative only. As we have seen in the previous example, the response variable was salary.

- The different domains we choose for the analysis are called the levels of treatment, and they should be three or more.
- The variance of the population must be similar.
- All the subjects in each group (levels) must be independent of each other.
- The population must be normally distributed.

One thing to consider while designing the experiment is that all the subjects must have similar characteristics. For instance, in the salary example, we had three career choices, but the people selected in the category of each career choice should have similar characteristics. They can be similar in age, the country in which they are living, the type of education they have completed, and so on. Once this sort of experiment is designed, we can successfully execute one-Way ANOVA over it.

As discussed earlier, we can use the ANOVA test to determine whether the means of the levels of treatment are different. However, it doesn't tell which mean is greater or smaller than the other. This can be done using the **Tukey test**, which we will talk about later in this chapter, but it cannot be determined using ANOVA.

The intuition behind the ANOVA test states that whatever randomized sample that we have got from the population is normally distributed, so we can say that each sample is coming from the populations having the same mean or at least one of the samples are coming from the population having a different mean. One thing that should always be kept in mind is that the level of variability within the samples will always be lower than the level of variability between the samples. This is because the samples are coming from different populations. Samples taken from the Ph.D. population and samples taken from the M. Tech population will have slightly different variance.

So, variability between the samples is termed as between-sample variability, and variability within a sample is termed as within-sample variability. ANOVA tries to check the significance of the difference between "**between-sample**" and "**within-sample**" variabilities. If there is a significant difference, we can say that the populations have different means; otherwise, they have the same mean. The following is the formula used to check this difference:

$$F_{statistic} = \frac{between - sample\,\mathrm{var}\,iability}{within - sample\,\mathrm{var}\,iability}$$

$F_{statistic}$ is the parameter that represents this difference. So, this test is also termed as F-test. As we assume that the population has the same mean, let it be denoted by μ and the variance be denoted by σ^2. The variance for the sample can be calculated with the following formula:

$$\sigma^2 = \frac{\Sigma(x_i - \underline{x})^2}{n-1}$$

Where, the numerator represents the sum of squares, while the denominator represents the degrees of freedom. So, we can say that the variance, in this case, is the sum of squares divided by the degrees of freedom, and it can also be called mean sum of squares. It tells us about the variability present in the data. If we can calculate this variability for within the sample and between samples, we can calculate the **F Statistic**. It can then be calculated as the ratio between the variability present in the two types of samples.

If we take the entire dataset and apply the preceding formula, it calculates the total variability present in the data. So, it can be termed as the total sum of squares. Now, let's understand how to calculate the actual value of F-Statistic in detail.

The numerator and denominator of the F-Statistic formula talk about between-sample and within-sample variability. We call the between-sample variability as Mean Square due to Treatment (MST), while within-sample variability is called Mean Square due to Error (MSE). This makes the formula revised to:

$$F_{statistic} = \frac{Mean\ Squaredue\ Treatment(MST)}{Mean\ Squaredue\ Error(MSE)}$$

If we can compute the values of MST and MSE, we can determine the value of F-Statistic, which will help us perform ANOVA result analysis. So, let's go step by step and compute these values.

Calculation of Mean Square due to Error

To compute the value of MSE, we need to compute the **Sum of Squares due to Error** (**SSE**) first. SSE is finding the variances present in the samples, giving them weightage based on the degrees of freedom, and then finding the sum of all the values. This process can be summarized using the following equation:

$$SSE = (n_1 - 1)\,\sigma_1^2 + (n_2 - 1)\,\sigma_2^2 + \ldots + (n_k - 1)\,\sigma_k^2$$

Where *n* is the total number of observations, and *k* is the total number of samples. The total degrees of freedom can be calculated by finding the difference between the total number of samples and observations, which can be given by:

$$totaldegreesoffreedom = n - k$$

We need to multiply each sample variance with its degree of freedom and find the sum of everything to get the SSE. Now, let's see how to calculate MSE. Divide the SSE by total degrees of freedom, as follows:

$$MSE = \frac{SSE}{total\deg reesoffreedom} = \frac{(n_1 - 1)\sigma_1^2 + (n_2 - 1)\sigma_2^2 + \ldots + (n_k - 1)\sigma_k^2}{n-k}$$

Calculation of Mean Square due to Treatment

Similar to MSE, we must first find the **Sum of Squares due to Treatment** (**SST**). Then, we will divide it by the total degrees of freedom to get MST. We calculate each sample's mean and overall mean, then find the square of the difference between these two means, and give them weightage based on the number of observations present in each sample to get the SST. This can be summarized using the following equation:

$$SST = n_1((\underline{x}_1 - \underline{x})^2 + n_2(\underline{x}_2 - \underline{x})^2 + \ldots + n_k(\underline{x}_k - \underline{x})^2$$

Once we get the SST, the next thing is to find the degrees of freedom for samples. This will be the total number of samples minus 1:

$$Degreesoffreedomforsample = k - 1$$

Now that we have both numerators and denominators, we can get our MST using the following equation:

$$MST = \frac{SST}{DegreesofFreedom} = \frac{n_1(\underline{x}_1 - \underline{x})^2 + n_2(\underline{x}_2 - \underline{x})^2 + \ldots + n_k(\underline{x}_k - \underline{x})^2}{k-1}$$

Now that we have both MSE and MST, we can use it to find F-Statistic. Let's understand this entire process by considering the following example:

At a community college, the mathematics department has been experimenting with four different delivery mechanisms for content in their Intermediate Algebra courses. One method is the traditional lecture (the method I), the second is a hybrid format in which half the class time is online and the other half is face-to-face (method II), the third is online (method III), and the fourth is an emporium model under which students obtain their lectures and carry out their work in a lab with an instructor available for assistance (method IV). To assess the effectiveness of the four methods, students in each approach appear for a final exam, and the results are shown in the following table. Does the data suggest that any method has a different mean score from the others?

Method I	Method II	Method III	Method IV
81	85	81	86
81	53	59	90
85	80	70	81
67	75	70	61
88	64	64	84
72	39	78	72
80	60	75	56
63	61	80	68
62	83	52	82
92	66	45	98
82	75	87	79
49	66	85	74
69	90	79	82
66	93		
74			
80			

Table 7.1

Let's look at the following steps:

1. Compute the mean of the entire dataset:

$$\underline{x} = \frac{(81+81+\ldots+80)+(85+53+\ldots+93)+(81+59+\ldots+79)+(86+90+\ldots+82)}{56} = 73.55$$

2. Compute the mean for each group:

$$\underline{x}_1' = \frac{(81+81+\ldots+80)}{16} = 74.44$$

$$\underline{x}_2' = \frac{(85+53+\ldots+93)}{14} = 70.71$$

$$\underline{x}_3' = \frac{(81+59+\ldots+79)}{13} = 71.15$$

$$\underline{x}_4' = \frac{(86+90+\ldots+82)}{13} = 77.92$$

3. Find the variance for each sample group:
   As we know, we can find the variance using this formula:

$$\sigma^2 = \frac{\sum(x_i - \underline{x})^2}{n-1}$$

The variance for each group can be denoted by:

$$\sigma_1^2 = \frac{\sum(x_i - 74.44)^2}{16-1} = 126.93$$

$$\sigma_2^2 = \frac{\sum(x_i - 70.71)^2}{14-1} = 228.07$$

$$\sigma_3^2 = \frac{\sum(x_i - 71.15)^2}{13-1} = 166.14$$

$$\sigma_4^2 = \frac{\sum(x_i - 77.92)^2}{13-1} = 134.24$$

4. The calculation of SST and SSE:

   The formulas for SST and SSE calculation are as follows:

$$SST = n_1(\underline{x}_1 - \underline{x})^2 + n_2(\underline{x}_2 - \underline{x})^2 + \ldots + n_k(\underline{x}_k - \underline{x})^2$$

$$SSE = (n_1 - 1)\sigma_1^2 + (n_2 - 1)\sigma_2^2 + \ldots + (n_k - 1)\sigma_k^2$$

   We can get the final value as:

$$SST = 16 * (74.44 - 73.55)^2 + 14 * (70.71 - 73.55)^2 + 13 * (71.15 - 73.55)^2 + 13 * (77.92 - 73.55)^2 = 448.73$$

$$\text{SSE} = (16 - 1) * 126.93 + (14 - 1) * 228.07 + (13 - 1) * 166.14 + (13 - 1) * 134.24 = 8473.42$$

5. The calculation of MST and MSE:

   We know that the formulas for MST and MSE calculation are as follows:

$$MSE = \frac{SSE}{total\deg reesoffreedom} \text{ and } MST = \frac{SST}{DegreesofFreedom}$$

   So:

$$MSE = \frac{8473.42}{52} = 162.95$$

$$MST = \frac{448.73}{3} = 149.58$$

6. Compute F-Statistic:

$$F - Statistic = \frac{MST}{MSE} = \frac{149.58}{162.95} = 0.92$$

```
#Importing Important Library

import scipy.stats as stats


#Defining our Data

method_1 = [81,81,85,67,88,72,80,63,62,92,82,49,69,66,74,80]

method_2 = [85,53,80,75,64,39,60,61,83,66,75,66,90,93]

method_3 = [81,59,70,70,64,78,75,80,52,45,87,85,79]

method_4 = [86,90,81,61,84,72,56,68,82,98,79,74,82]


#Defining the Level of Significance

alpha = 0.05


#Performing the One Way ANOVA Test

outcome = stats.f_oneway(method_1, method_2, method_3, method_4)


#Printing the Outcome

print("F-Statistic Value is %f and P-Value is %f" %(outcome.statistic,
outcome.pvalue))


#Checking the result

if outcome.pvalue<= alpha:

print( 'Null hypothesis can be rejected \n\n P-value is less than the
level of Significance \n F-statistic is greater than F-critical')

else:

print( '\nNull hypothesis cannot be rejected \n\n P-value is greater than
the level of Significance \n F-statistic is less than F-critical')
```

Decision Rule

As we saw in hypothesis testing, we accept or reject the null values based on the level of coincidence that we decide. Once we decide this coincidence level, we get a respective p-value, which is called the significance level. We will use this significance level to decide on the acceptance or rejection of the null hypothesis. After applying the F-statistic formula, we'll get a p-value for the analysis. If this p-value is greater than the level of significance, we accept the null hypothesis; else, we reject it. This is the major decision rule we use in the case of an ANOVA test.

Another way to interpret this is to compare the F-statistic value with the F-critical using the degrees of freedom and level of significance. For this, we will use the F-table given at the appendix of this book. The first thing we need to know is the degrees of freedom that we used in the numerator and denominator. We know that we used $(n - k)$ as the degrees of freedom for the denominator and $(k - 1)$ as the degrees of freedom for the numerator. We will use both of these values in the F-table to generate our F-critical. Another thing we need is the level of significance, which, let's just say, is 0.05. Once we get our F-critical, we will compare it with our F-statistic value. Then, our decision rule is revised as follows:

$$IfF - Statistic > F\text{–}critical, rejectthenullhypothesis.Else,acceptit.$$

Let's understand this rule by applying it to the example in the previous section.

In the preceding example, we got the *F-statistic* as *0.92*. If the level of significance is *0.05*, numerator degrees of freedom is *3*, and denominator degrees of freedom is *52*, we can get the value of *F-critical* as *2.78*. As *F-statistic* is less than *F-critical*, we have to accept the null hypothesis. So, we can conclude that:

"There is no difference between the mean of the samples."

We have successfully seen the application of one-way ANOVA on the samples and how to accept and reject the null hypothesis based on the decision rule. However, we still don't know which mean is different from the others in the sample if we reject the null hypothesis. This is one of the shortcomings of the ANOVA test, and we have an extension to ANOVA called the Tukey test to determine which mean differs from the lot.

Tukey test

Tukey test is used to check the pairwise mean difference. Suppose we performed the ANOVA test on three samples and rejected the null hypothesis as an outcome. Now, we want to check which of the means differ: μ_1, μ_2 or μ_3. We will use the Tukey test to check this. The starting hypothesis can be any one of the following three:

$$H_0 : \mu_1 = \mu_2$$
$$H_1 : \mu_1 \neq \mu_2$$

Or

$$H_0 : \mu_1 = \mu_3$$
$$H_1 : \mu_1 \neq \mu_3$$

Or

$$H_0 : \mu_3 = \mu_2$$
$$H_1 : \mu_3 \neq \mu_2$$

For the sake of understanding, let's proceed with the first option. For checking these hypotheses, we have Tukey statistic, which is represented by q_0. This statistic can be represented as follows:

$$q_0 = \frac{(x_2 - x_1)}{\sqrt{\frac{s^2}{2} \cdot \left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

Where,

n_1 *: Samples*$\in$ *population1*n_2 *: Samples*$\in$ *population2*s^2 *: MeanSquaredueError*

This test is also called as honestly significant difference test. As we saw in the ANOVA tests, we have to accept or reject the null hypothesis once we get the statistic value. In the case of the ANOVA test, we used F-table. But what about this test? ANOVA processes data having F-distribution, so we need to refer to F-table, but Tukey test follows the studentized range distribution, so we will use this range table to get the critical values. The table is provided in the appendix.

After getting the critical value, we'll follow the same process of null value rejection. If our statistic value is greater than the critical value, we discard the null hypothesis; else, we keep it. Let's understand this process in detail by continuing the example from the previous section.

A subset of children uses lean meats, mixed meats, or higher-fat meats. The following data represents the daily consumption of calcium (in mg) for a random sample of eight children in each category. Can we say that all the children are from the same sample?

Lean_Meats	Mixed_Meats	Higher-Fat_Meats
844.2	897.7	843.4
745	908.1	862.2
773.1	948.8	790.5
823.6	836.6	876.5

Contd…

812	871.6	790.8
758.9	945.9	847.2
810.7	859.4	772
790.6	920.2	851.3

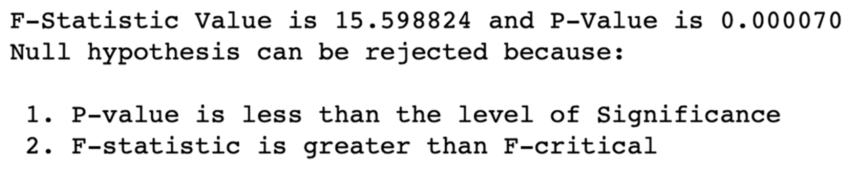
Table 7.2

We have already performed a similar test using one-way ANOVA. Let's use the following Python code to check the F-statistic this time:

```
#Importing Important Library

import scipy.stats as stats


#Defining our Data

lean = [844.2,745,773.1,823.6,812,758.9,810.7,790.6]

mixed = [897.7,908.1,948.8,836.6,871.6,945.9,859.4,920.2]

higher_fat = [843.4,862.2,790.5,876.5,790.8,847.2,772,851.3]


#Defining the Level of Significance

alpha = 0.05


#Performing the One Way ANOVA Test

outcome = stats.f_oneway(lean, mixed, higher_fat)


#Printing the Outcome

print("F-Statistic Value is %f and P-Value is %f" %(outcome.statistic,
outcome.pvalue))


#Checking the result

if outcome.pvalue<= alpha:

print( 'Null hypothesis can be rejected \n\n P-value is less than the
level of Significance \n F-statistic is greater than F-critical')

else:
```

```
print( '\nNull hypothesis cannot be rejected \n\n P-value is greater than
the level of Significance \n F-statistic is less than F-critical')
```

We get the following output:



```
F-Statistic Value is 15.598824 and P-Value is 0.000070
Null hypothesis can be rejected because:

 1. P-value is less than the level of Significance
 2. F-statistic is greater than F-critical
```

Figure 7.2

As we have rejected the null hypothesis, we can now apply the Tukey test to check which mean differs. Let's check out the steps for computing the Tukey test:

1. Compute the group mean:

   The means of different groups are:

$$\underline{x}_1 = 794.76$$
$$\underline{x}_2 = 898.58$$
$$\underline{x}_3 = 829.24$$

2. Using this, we will compute the pair-wise distance.

   The following table shows different combinations and the differences in the means:

Comparison	Mean Difference
Lean vs Mixed	898.54 – 794.76 = 103.78
Lean vs Higher Fat	829.24 – 794.76 = 34.48
Mixed vs Higher Fat	898.54 – 829.24 = 69.3

* Subtract the smaller from the larger

Table 7.3

3. Compute Tukey statistic.

   For this, we'll use the following formula:

$$q_0 = \frac{(x_2 - x_1)}{\sqrt{\frac{x^2}{2} \cdot \left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

   The mean squared error for the overall ANOVA test is given by:

$MSE = \frac{SSE}{n-k}$ = 1432.61 (One can calculate it by following the ANOVA approach)

This is the s^2 for the preceding equation, and n represents the number of samples in each group, which is equal to 8. So, we can get the following table when we substitute the above values:

Comparison	Tukey Statistic	Tukey Critical
Lean vs. Mixed	7.76	3.565
Lean vs. Higher Fat	2.58	3.565
Mixed vs. Higher Fat	5.18	3.565

Table 7.4

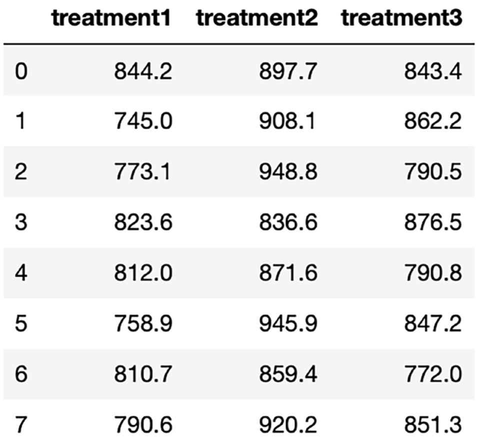
We can get the value of Tukey critical by looking at the Q Table (studentized range distribution table) with *Degree of Freedom = 21*, *Number of Groups = 3*, and *Level of Significance = 0.05*. The value derived is *3.565*. The preceding table specifies that only the first and the last combinations are greater than Tukey critical, while the middle one is less than Tukey critical, so we can summarize it as:

- Lean Meat and Mixed Meat are samples from different populations, so their means differ.
- Mixed Meat and Higher Fat meat are samples from different populations, so their means differ.
- Lean Meat and Higher Fat meat are samples from the same population, so their means do not differ.

Let's look at the Python code that performs this analysis; it provides exactly the same insights:

```
#importing necessary library
import pandas as pd

#converting our data into pandas dataframe
df = pd.DataFrame()
df['treatment1'] = lean
df['treatment2'] = mixed
df['treatment3'] = higher_fat
display(df)
```

The following is the output screen:

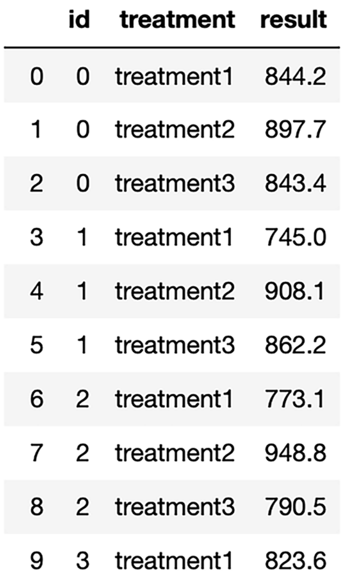


	treatment1	treatment2	treatment3
0	844.2	897.7	843.4
1	745.0	908.1	862.2
2	773.1	948.8	790.5
3	823.6	836.6	876.5
4	812.0	871.6	790.8
5	758.9	945.9	847.2
6	810.7	859.4	772.0
7	790.6	920.2	851.3

Figure 7.3

```
#stacking everything into three columns – id, treatment, result

stacked_data = df.stack().reset_index()

stacked_data = stacked_data.rename(columns={‘level_0’:  ‘id’,  ‘level_1’:
                                             ‘treatment’, 0:’result’})

display(stacked_data)
```

Following is the output:

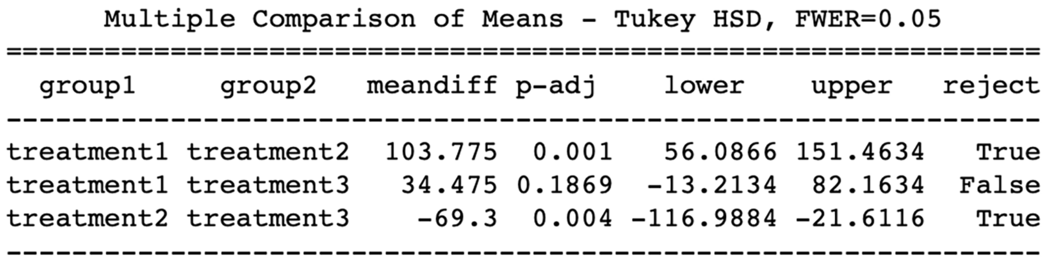


	id	treatment	result
0	0	treatment1	844.2
1	0	treatment2	897.7
2	0	treatment3	843.4
3	1	treatment1	745.0
4	1	treatment2	908.1
5	1	treatment3	862.2
6	2	treatment1	773.1
7	2	treatment2	948.8
8	2	treatment3	790.5
9	3	treatment1	823.6

Figure 7.4

```
#Applying Tukey Test

from statsmodels.stats.multicomp import pairwise_tukeyhsd, MultiComparison

MultiComp = MultiComparison(stacked_data['result'], stacked_data['treat-
ment'])

print(MultiComp.tukeyhsd().summary())
```

The following is the output screenshot:



```
     Multiple Comparison of Means - Tukey HSD, FWER=0.05
===============================================================
  group1     group2   meandiff p-adj    lower    upper   reject
---------------------------------------------------------------
treatment1 treatment2  103.775  0.001   56.0866 151.4634   True
treatment1 treatment3   34.475 0.1869  -13.2134  82.1634  False
treatment2 treatment3    -69.3  0.004 -116.9884 -21.6116   True
---------------------------------------------------------------
```

Figure 7.5

We have seen how one factor affects different samples from different populations. What if we have two factors instead of only one factor? In such cases, we will use 2-way ANOVA for our analysis. We will look at 2-way ANOVA in detail in the next section.

Two-way ANOVA

Let's take an example wherein coders love playing video games. Video games have two categories: multiplayer games and single player games, and coders have two levels: male and female coders. In this scenario, we will use the 2-way ANOVA analysis to derive a conclusion. In the current scenario, both factors have two levels, so we call it 2*X*2 Factorial Design of ANOVA. If we generalize this, we can say that if Factor 1 has 'a' levels, and Factor 2 has 'b' levels, the resulting analysis table will be an $a \times b$ factorial design.

Depending on how these factors and levels affect the response variable, we can classify them into two categories:

1. Main effects
2. Interaction effects

Main Effects

If we analyze the change in response variable when we change the levels of both Factor 1 and Factor 2 together, it's called the **main effect**.

Interaction Effects

However, if we analyze the response variable as we change the levels of the first factor and if the change in the response is different for different levels of the second factor, we say that both the factors have interaction effect between them. Let's understand this with the help of an example.

Let's first understand the main and interaction effects with the help of an example. Suppose I want to understand the effect of tuitions and online courses on exam results. The following table explains the outcome:

	Online Courses			
Tutions	No		Yes	
No	35	33	41	45
Yes	40	38	45	48

Table 7.5

Let's look at the main effect of tuitions first. For this, we will look at the means of the No and Yes values of tuitions and subtract them from each other:

$$MainEffect_{tuition} = \frac{40+38+45+48}{4} - \frac{35+33+41+45}{4} = 4.25$$

Similarly, we can check the main effect of online courses:

$$MainEffect_{online} = \frac{41+45+45+48}{4} - \frac{35+33+40+38}{4} = 8.25$$

We can interpret these values as follows:

- If we make a student take tuitions, their performance increases by 4.25 marks
- If we make a student take online courses, their performance increases by 8.25 marks

Next, we must check the interaction effect between the two factors. To find this, let's first check how a student's performance is affected when they only take tuitions and not the online course. For this, we will find the mean of the marks obtained after and before taking tuitions and then subtract them:

$$\frac{40+38}{2}-\frac{35+33}{2}=5$$

So, we can see that the marks go up by 5. Similarly, let's look at the performance when online courses are taken as well:

$$\frac{45+48}{2}-\frac{41+45}{2}=3.5$$

In this situation, we can see that the marks go up by 3.5 points. Now, if we look at the main effect of the tuition, it is 4.25 marks. The mean increase in marks is 5 when students don't take the online course, and it is 3.5 if they take the online course. So, we can say that there is some interaction effect between tuitions and online courses.

Let's see some of the requirements that our data must meet before we can continue with 2-way ANOVA analysis:

- The distribution of the population from where we have collected the sample must be normal.
- All the samples taken must be independent of each other.
- The populations from which different samples are drawn must share the same variance.

In a 2-way ANOVA test, we have three individual hypotheses testing instead of a single hypothesis testing like in one-way ANOVA. These are classified under hypothesis related to interaction and main effects, as follows:

- Is there an interaction effect between factors?
- Is there any main effect of factor 1 on the response variable?
- Is there any main effect of factor 2 on the response variable?

So, the three hypotheses testing statements are mentioned in the following table:

Null Hypothesis (H_0)	**Alternate Hypothesis (H_1)**
No Interaction Effect between factors	There is an interaction effect between factors
No Main Effect between Factor 1 and Response Variable	There is a Main Effect between Factor 1 and Response Variable
No Main Effect between Factor 2 and Response Variable	There is a Main Effect between Factor 2 and Response Variable

Table 7.6

We will always start with the hypothesis on the interaction effect. If we can reject the null hypothesis there, there is no need to check the main effects. If there is an interaction effect, it will be extremely difficult for us to understand the main effect. Let's look at the following example on the 2-way ANOVA test:

Age (years)			
Gender	18–34	35–54	55 and older
Female	180,192	205,226	218,231
Male	175,193	213,222	203,185

Table 7.7

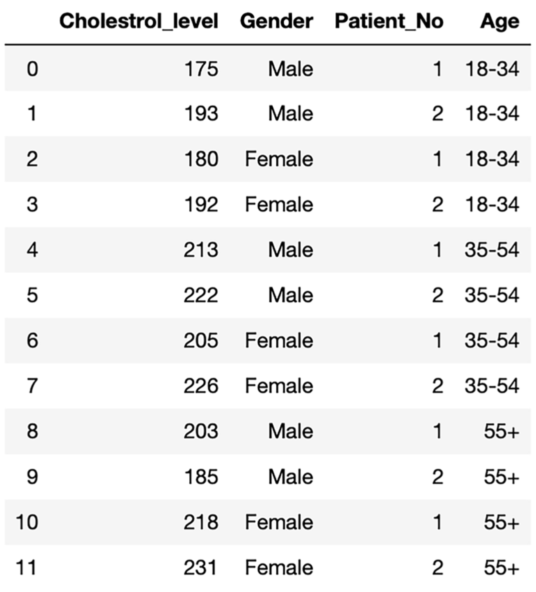
A family physician wanted to know if age and gender were factors that explained the levels of serum cholesterol (in mg/dL) in her adult patients. She randomly selects two patients for each category of data and obtains the results indicated in the preceding table. We have to check whether age and gender effect the cholesterol levels.

Let's put this data in Python:

```
#Preparing Data

data = [175,193,180,192,213,222,205,226,203,185,218,231]

gender = [“Male”,”Male”,”Female”,”Female”]*3

patient = [‘1’,’2’]*6

age = [[“18-34”]*4,[“35-54”]*4,[“55+”]*4]


age_flattened = []

for sublist in age:

   for item in sublist:

age_flattened.append(item)


#Creating a DataFrame

a=pd.DataFrame()

a[“Cholestrol_level”] = data

a[“Gender”] = gender

a[“Patient_No”] = patient

a[“Age”] = age_flattened
```

```
display(a)
```

Take a look at the following output:



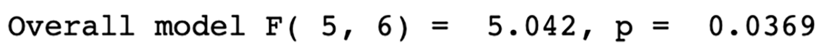
	Cholestrol_level	Gender	Patient_No	Age
0	175	Male	1	18-34
1	193	Male	2	18-34
2	180	Female	1	18-34
3	192	Female	2	18-34
4	213	Male	1	35-54
5	222	Male	2	35-54
6	205	Female	1	35-54
7	226	Female	2	35-54
8	203	Male	1	55+
9	185	Male	2	55+
10	218	Female	1	55+
11	231	Female	2	55+

Figure 7.6

Let's check for the presence of an interaction effect:

```
#Generating the Model to check for Interaction Effect

from statsmodels.formula.api import ols

model = ols('Cholestrol_level ~ C(Gender)*C(Age)', a).fit()

print(f"Overall model F({model.df_model: .0f},{model.df_resid: .0f}) =
{model.fvalue: .3f}, p = {model.f_pvalue: .4f}")
```

The following is the output:

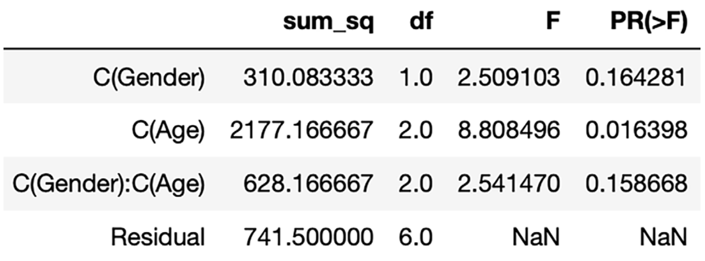


```
Overall model F( 5, 6) =  5.042, p =  0.0369
```

Figure 7.7

```
#Checking Residual Summary

res = sm.stats.anova_lm(model, typ= 2)

display(res)
```

Here’s the output:



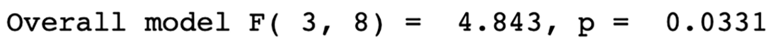
	sum_sq	df	F	PR(>F)
C(Gender)	310.083333	1.0	2.509103	0.164281
C(Age)	2177.166667	2.0	8.808496	0.016398
C(Gender):C(Age)	628.166667	2.0	2.541470	0.158668
Residual	741.500000	6.0	NaN	NaN

Figure 7.8

In the preceding diagram, the p-value of interaction effect is *C(Gender)*:*C(Age)*, which is greater than *0.05*. This means we have to accept the null value proving that there is no interaction effect in our model. Now, since there is no interaction effect, we need to check whether there is a main effect. We will check that using the following code:

```
#Checking the Model to check for Main Effect

model = ols(‘Cholestrol_level ~ C(Gender)+C(Age)’, a).fit()

print(f”Overall model F({model.df_model: .0f},{model.df_resid: .0f}) = {model.fvalue: .3f}, p = {model.f_pvalue: .4f}”)
```

The output screen looks like this:

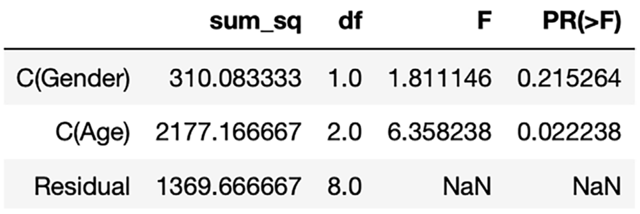


```
Overall model F( 3, 8) =  4.843, p =  0.0331
```

Figure 7.9

```
#Checking Residual Summary

res2 = sm.stats.anova_lm(model, typ= 2)

display(res2)
```

Here’s the output:



	sum_sq	df	F	PR(>F)
C(Gender)	310.083333	1.0	1.811146	0.215264
C(Age)	2177.166667	2.0	6.358238	0.022238
Residual	1369.666667	8.0	NaN	NaN

Figure 7.10

The p-value of gender is greater than *0.05*, but the p-value of age is *0.02*. This means there is a main effect coming from the age, so we will have to reject the null hypothesis. The ANOVA test here proves that the factor age does affect the mean of cholesterol levels.

Multivariate Analysis of Variance (MANOVA)

As we saw in the previous sections, ANOVA is the test of means. It tells us whether the groups we have sampled are taken from the distributions having the same means. We use MANOVA to test whether the vectors of the means from two or more groups are sampled from the same distribution. Additionally, it tells us the likelihood to select two or more random samples of means from a single source.

We use MANOVA in two major situations:

- We have a single dependent variable in ANOVA, for example, salary. We use the MANOVA test if we have more than two, correlated dependent variables and want to check the effects on different groups. If we go for ANOVA, we will have to do an individual test for each variable, which is not recommended.
- It helps us understand how independent variables show patterns of response when it comes to dependent variables.

Just like ANOVA, MANOVA cannot tell you which groups differ from the other groups based on their mean vectors. Also, it will not tell you which dependent variable is responsible for the differences in the mean vector. Let's understand MANOVA with the help of an example.

Suppose we have four different groups of patients, wherein each group is given a different kind of medication. We want to check whether all the four approaches have the same effect in the treatment of depression. Remember, we won't be able to tell which treatment is different and whether it is better or worse. The following are the details of the four groups:

- Group 1: Treatment with clinical psychotherapy and placebo drug
- Group 2: Treatment with placebo medication and cognitive psychotherapy
- Group 3: Treatment with antidepressant medication and clinical psychotherapy
- Group 4: Treatment with cognitive therapy and active medication

In this example, we have two kinds of medication—placebo and Drug—and two kinds of psychotherapy—clinical and cognitive. Suppose we want to judge the outcome based on three dependent variables:

- **Beck Depression Index** (**BDI**)
- **Hamilton Rating Scale** (**HRS**)
- **Symptom Checklist for Relatives** (**SCR**)

For simplicity, suppose there are five people in each of the four groups. If we visualize this with the help of a table, we'll get three types:

BDI Score table

Group 1	Group 2	Group 3	Group 4
12	6	12	10
10	20	13	9
16	11	14	12
9	14	12	5
11	5	11	14

Table 7.8

HRS Score table

Group 1	Group 2	Group 3	Group 4
11	7	10	5
12	14	10	7
11	15	15	9
6	13	15	13
15	5	8	10

Table 7.9

SCR Score table

Group 1	Group 2	Group 3	Group 4
9	14	9	9
10	7	7	10
11	9	6	15
13	11	11	12
14	12	13	8

Table 7.10

Just like 2-way ANOVA, MANOVA will have three effects on the preceding tables:

- Psychotherapy main effect
- Medication main effect
- Interaction effect between psychotherapy and medication

Since we only get 1x1 vector for each group in ANOVA and take the dependent variable one at a time, the vector is 3x1 for each group in MANOVA because we have three dependent variables in our example. When we talk about psychotherapy's main effect, MANOVA compares clinical and cognitive therapies and analyzes whether their mean is different, irrespective of the medication. However, all three variables are considered instead of a single variable. So, they compare the mean vectors of shape 3x1. Similarly, mean vectors are compared for medication main effect, irrespective of their psychotherapy. Finally, we check the interaction effect between psychotherapy and medication by looking at the four mean vectors of each group and comparing them with the predicted vectors from the main effects of psychology and medication.

Coming to the computation of the MANOVA test, it comprises multiple statistical tests as compared to the single F-statistic (F-Test) of the ANOVA test. These tests utilize two matrices: **H** and **E**. H refers to the Hypothesis Matrix, while E refers to the Error Matrix. It is assumed that H Matrix has *h* degrees of freedom, while E Matrix has *e* degrees of freedom. Based on these matrices, the following matrix operations are performed:

- $H * (E + H)^{-1}$
- $H * E^{-1}$
- $E * (E + H)^{-1}$

We have different eigen values for all the preceding equations. Suppose these are represented by the symbols θ_i, φ_i, λ_i. Their values are given by the following formulas:

- $\theta_i = 1 - \lambda_i = \dfrac{\phi_i}{1+\phi_i}$
- $\phi_i = \dfrac{\theta_1}{1-\theta_i} = \dfrac{1-\lambda_i}{\lambda_i}$
- $\lambda_i = 1 - \theta_i = \dfrac{1}{1+\theta_i}$

Once we know these **Eigen Values**, we can go for different MANOVA tests:

1. Wilks' Lambda test
2. Lawley Hotelling Trace

3. Pillai's Trace
4. Roy's Largest Root

Wilks' Lambda test

We can use the following equation to get to the F-distribution table and then decide whether to accept or reject the null value based on the level of significance. Once we get the F-statistic, the process is the same as with ANOVA:

$$F-Statistic=\frac{(ft-g)\left(1-\Lambda^{\frac{1}{t}}\right)}{ph\Lambda^{\frac{1}{t}}}$$

Where,

- $\Lambda=\Pi_{(j=1)}^{p}(1-\theta_j)$
- $f=e-\frac{1}{2}(p-h+1)$
- $g=\frac{ph-2}{2}$
- $t=\left\{\sqrt{\frac{p^2+h^2-4}{p^2+h^2-5}}ifp^2+h^2>01otherwise\right\}$
- $p=NumberofDependentVariables$
- $h=TotalnumberofGroups-1=m-1$

Lawley Hotelling Trace

The F-Statistic for this test can be represented as:

$$F-Statistic=\frac{T^2g}{ce}$$

Where,

- $c=\frac{a(b-2)}{b(e-p-1)}$
- $a=ph$
- $b=4+(a+2)(B-1)$
- $B=\frac{(e+h-p-1)(e-1)}{(e-p-e)(e-p)}$

Pillai's Trace

The F-Statistic for this test is represented as:

$$F - Statistic = \frac{(2n+s+1)V^s}{(2m+s+1)(s-V^s)}$$

Where,

- $V^s = \sum_{j=1}^{s} \theta_j$
- $s = (p, h)$
- $m = \frac{(|p-h|-1)}{2}$
- $n = \frac{(e-p-1)}{2}$

Roy's Largest Root

The F-Statistic for this test is represented as:

$$F - Statistic = \frac{2v_2 + 2}{2v_1 + 2} * \phi_{max}$$

Where,

- $v_1 = \frac{(|p-h|-1)}{2}$
- $v_2 = \frac{(e-p-1)}{2}$

As already mentioned, all these tests follow F-distribution. So, just like ANOVA tests, they need to be compared with the F-critical value. Reject the null hypothesis if they are greater than that, else accept it.

Some of the assumptions that need to be followed for MANOVA tests are as follows:

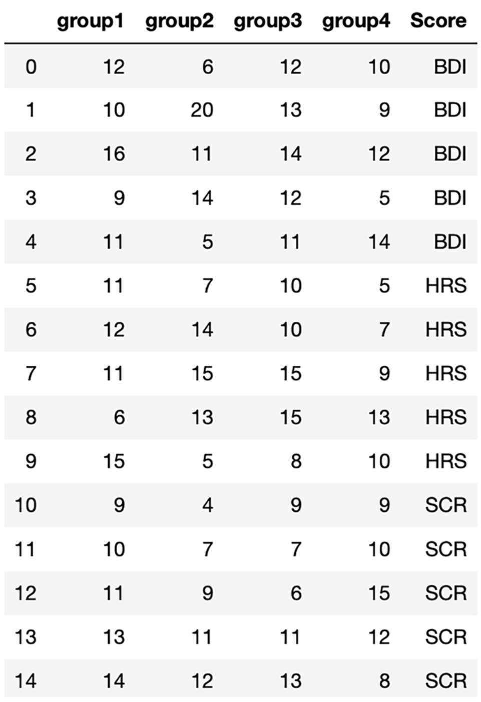
1. It doesn't work on discrete data, so the response variables must be continuous.
2. Residuals must be in normal distribution without skewness.
3. All the groups must be independent.

Take a look at the following code:

```
import pandas as pd

from statsmodels.multivariate.manova import MANOVA
```

```
group1 = [12,10,16,9,11,11,12,11,6,15,9,10,11,13,14]
group2 =[6,20,11,14,5,7,14,15,13,5,4,7,9,11,12]
group3=[12,13,14,12,11,10,10,15,15,8,9,7,6,11,13]
group4=[10,9,12,5,14,5,7,9,13,10,9,10,15,12,8]

b = pd.DataFrame()
b["group1"]=group1
b["group2"]=group2
b["group3"]=group3
b["group4"]=group4
b["Score"]=["BDI","BDI","BDI","BDI","BDI","HRS","HRS","HRS","HRS","HRS",
"SCR","SCR","SCR","SCR","SCR"]

display(b)
```

The following is the output:



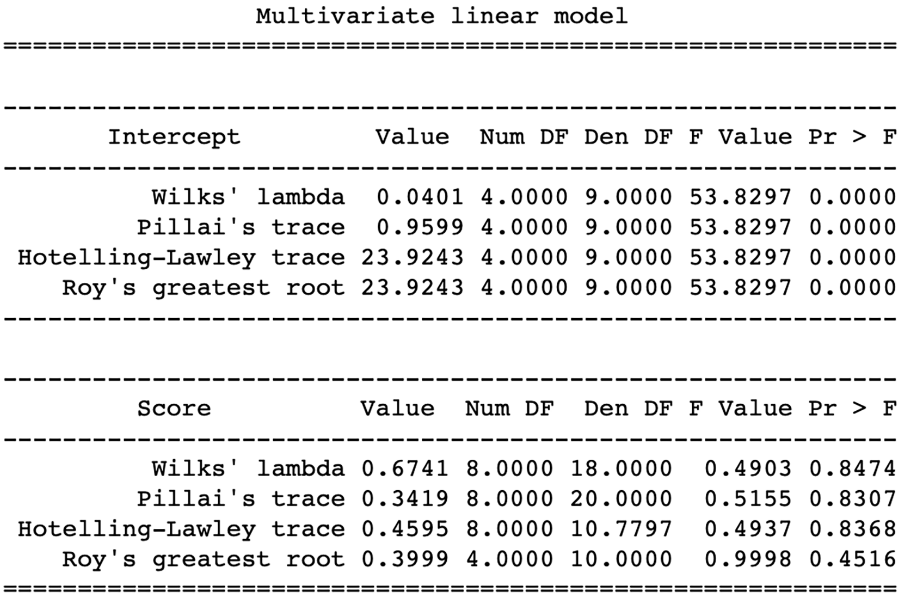
	group1	group2	group3	group4	Score
0	12	6	12	10	BDI
1	10	20	13	9	BDI
2	16	11	14	12	BDI
3	9	14	12	5	BDI
4	11	5	11	14	BDI
5	11	7	10	5	HRS
6	12	14	10	7	HRS
7	11	15	15	9	HRS
8	6	13	15	13	HRS
9	15	5	8	10	HRS
10	9	4	9	9	SCR
11	10	7	7	10	SCR
12	11	9	6	15	SCR
13	13	11	11	12	SCR
14	14	12	13	8	SCR

Figure 7.11

```
maov = MANOVA.from_formula('group1 + group2 + group3 + group4 ~ Score',
data=b)
```

```
print(maov.mv_test())
```

Here's the output screenshot:



```
                 Multivariate linear model
============================================================

------------------------------------------------------------
       Intercept         Value  Num DF Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda  0.0401 4.0000 9.0000 53.8297 0.0000
         Pillai's trace  0.9599 4.0000 9.0000 53.8297 0.0000
 Hotelling-Lawley trace 23.9243 4.0000 9.0000 53.8297 0.0000
    Roy's greatest root 23.9243 4.0000 9.0000 53.8297 0.0000
------------------------------------------------------------

------------------------------------------------------------
         Score          Value  Num DF  Den DF F Value Pr > F
------------------------------------------------------------
          Wilks' lambda 0.6741 8.0000 18.0000  0.4903 0.8474
         Pillai's trace 0.3419 8.0000 20.0000  0.5155 0.8307
 Hotelling-Lawley trace 0.4595 8.0000 10.7797  0.4937 0.8368
    Roy's greatest root 0.3999 4.0000 10.0000  0.9998 0.4516
============================================================
```

Figure 7.12

The preceding table mentions that all the four tests have p-values greater than *0.05*. This means we won't be able to reject our null hypothesis. Instead, we'll have to accept it. So, it states that there is no effect of medication or psychotherapy on the above-mentioned scores.

Conclusion

In this chapter, we saw how two or more samples perform on different levels of treatments. In one-way ANOVA, we looked at how one factor affects multiple groups. Then, we moved to two-way ANOVA, where we increased the number of factors and looked at how main and interaction effects move the analysis. Finally, we looked at the MANOVA, where the effects of the factors are seen on multivariate data. We looked at the application of every concept in Python.

In the next chapter, we will look at the application part of most of the statistical concepts we learned until now. We will explore the feature engineering done before applying any mathematical models and use some of the most-used Python packages, like `numpy`, `pandas`, and `matplotlib`.

Chapter 8
Regression

The concept of regression is used to capture the quantitative response of a group of variables on a single output variable. The output variable that captures the response is called the dependent or predicted cariable, while the other variables that help get the response are called independent or predictor variables.

Structure

There are various types of regressions, but we will be discussing the following ones in this chapter:

- Simple linear regression
- Multiple linear regression
- Polynomial regression
- Subset selection method
- Ridge regression
- Lasso regression
- ElasticNet regression
- Logistic regression

- Understanding residuals
- Multicollinearity

Objective

This is one of the advanced applications of statistics. We will apply the statistical measures learned to predict future values with different regression approaches.

Simple Linear Regression

When we measure the quantitative response Y based on one single predictor variable X, we use *Simple Linear Regression*. In Simple Linear Regression, we assume that there is an approximate linear relation between X and Y, and the equation for that linear line is given by:

$$Y = \beta_0 + \beta_1 X$$

Where β_0 represents the intercept, and β_1 represents the slope of the line. Together, they are called the **Parameters of Regression**. We are provided with data, called training data, so that we can learn the value of these unknown parameters and use their learned values to predict something new. For example, suppose we have learned the following value of the parameters:

$$\beta_0 = 30$$

$$\beta_1 = 5$$

Now, suppose X represents the height of a person, and Y is their weight. We want to determine the weight of a person if their height is 6 ft. Using the preceding parameter values, we can get the weight as follows:

$$Y = \beta_0 + \beta_1 X = 30 + 5 * 6 = 60$$

So, based on the learned values, if a person has a height of 6 ft, their weight will approximately be 60 kgs. But how exactly can we get the values of parameters?

Finding the Values of β$_0$ and β$_1$

We find the values of the parameters in such a way that if we use the equation to predict the original values present in the data, it should ideally be equal to the actual values. Taking the same example of height and weight, suppose we have the following information about 10 people:

Height	Weight
5.2	55
5.5	59
5.4	63
5.8	72
5.7	75
6.2	81
6.1	90
5.11	79
6.0	69
5.5	73

Table 8.1

Now, we have to find the values of β$_0$ and β$_1$ in such a way that the predicted output of weight is almost equal to the original column values given in the preceding table. This can be represented as follows:

$$y_i \sim \widehat{\beta}_0 + \widehat{\beta}_1 X$$

For $i = 1, 2, 3, \ldots, n$

To understand how to get the values of parameters, we must know about the residuals. Residuals, also called errors, are the values that tell us how much the predicted line differs from the original line. Or, in simpler terms, we can say that residuals tell us the extent to which the predicted output for each training example differs from the actual value. This can be represented as:

$$e_i = y_i - \hat{y}_l = y_i - \widehat{\beta}_0 - \widehat{\beta}_1 x_i$$

This distance can be represented visually, as given in the following figure:

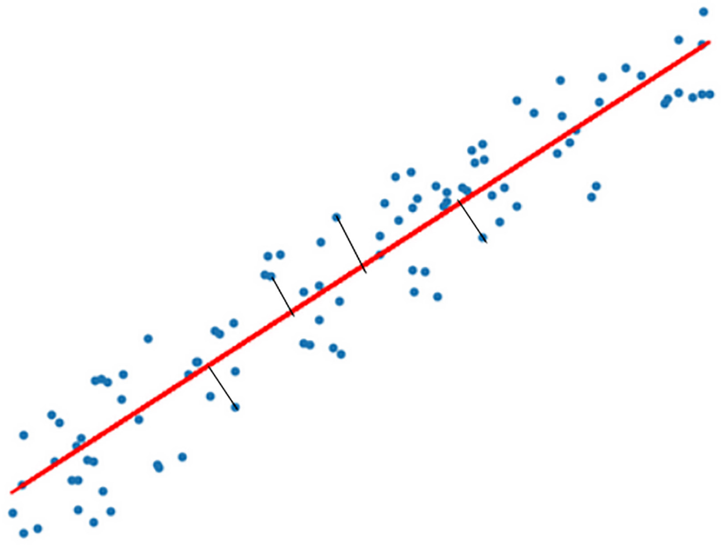


***Figure 8.1:** Fitted line and residuals*

In the preceding figure, the red line represents the predicted regression line. The blue points are the original data, while the predicted data is present on the red line. So, a straight line drawn from individual points to the residual line represents the residual, represented by some of the black lines in the preceding screenshot.

These residuals are used to find the **Residual Sum of Squares** (**RSS**), which is later used to find out the parameters:

$$RSS = e_1^2 + e_2^2 + \ldots + e_n^2$$

The values of β_0 and β_1 are chosen in such a way that they minimize the above equation of RSS. This involves a few aspects of calculus that are outside the scope of this book. After calculations, the formulas to compute the best values of β_0 and β_1 are represented by:

$$\hat{\beta}_1 = \frac{\sum_{i-1}^{n}(x_i - \underline{x})(x_i - \underline{y})}{\sum_{i=1}^{n}(x_i - \underline{x})^2}$$

$$\hat{\beta}_0 = \underline{y} - \hat{\beta}_1 \underline{x}$$

Where, $\underline{y}$ and $\underline{x}$ are the mean of dependent and independent variables. This entire process of parameter estimation is called the least squares coefficient estimates method.

Standard Error

In *Chapter 5: Parameter estimation*, we saw the concept of **Parameter Estimation**. We understood that we could get estimates about the population with knowledge of the sample. In linear regression, we want to know whether the estimation of β_0 and β_1 is closer to the population parameter. We try to understand this using the concept of **Standard Error**. It tells us how the measure of a sample parameter is different from the population parameter. Here are the formulas used to find the Standard Error for different parameters:

$$SE(\hat{\beta}_0)^2 = \sigma^2\left(\frac{1}{n} + \frac{\underline{x}^2}{\sum_{i=1}^{n}(x_i - \underline{x})^2}\right)$$

$$SE(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \underline{x})^2}$$

As the value of σ^2 is not known; we try to estimate it. The estimated value of σ^2 is called **Residual Standard Error** (**RSE**), and it can be estimated using the following formula:

$$RSE = \sqrt{\frac{RSS}{(n-2)}}$$

Confidence Intervals

Standard Errors can also be used to compute confidence intervals. A 95% confidence interval means we'll get a range of values such that there is a 95% probability that the range will contain the true value of the parameter. We can compute the 95% CI for the estimated parameters using the following equation:

$$95\%\ CI\ for\ \hat{\beta}_1 = \hat{\beta}_1 \pm 2 * SE(\hat{\beta}_1)$$
$$95\%\ CI\ for\ \hat{\beta}_0 = \hat{\beta}_0 \pm 2 * SE(\hat{\beta}_0)$$

Unimportant Variable

We can determine whether an independent variable has a relation with the dependent variable. We can use the concept of hypothesis testing to determine the relation between X and Y and continue with our analysis based on that.

We can proceed with the hypothesis testing by first declaring the null and alternate hypotheses:

$$H_0 = \textit{There is no relation between X and Y} \Rightarrow \beta_1 = 0$$

$$H_a = \textit{There is a relation between X and Y} \Rightarrow \beta_1 \neq 0$$

We will find a t-statistic using the following formula to disprove the null hypothesis:

$$T_{statistic} = \frac{\hat{\beta}_1 = 0}{SE(\hat{\beta}_1)}$$

We can find the p-value using the t-table or any statistical tool. If the p-values are less than 0.05, we can reject the null hypothesis, proving that there is a relation between the X and Y variables. Otherwise, we'll have to accept the null hypothesis.

Accuracy of Prediction

We have already seen that we can use the concept of **Residual Standard Error** to determine whether the predictions are closer to the true value. It is called the measure of the lack of fit. One of the issues with RSE is that it has the same unit as of the predicted variable, which makes it a little difficult to understand the value. So, it can be accompanied by another measure called R^2, the value that tells us about the proportion of variance explained by the model. This has a range of 0 to 1, and it is unit free as well. We can get the value of by R^2 using the following formula:

$$R^2 = 1 - \frac{RSS}{TSS}$$

Where TSS is the Total Sum of Squares, and we can get the value with this formula:

$$TSS = \sum_{i=1}^{n} (y_i - \underline{y})^2$$

We'll now look at the Python application of simple linear regression.

Note: We will use two datasets in this chapter: **Housing Price Dataset** and **Titanic Dataset**. Both the datasets are available in *kaggle.com* as well as this book's repository. Before continuing with the code application of all the regressions, we will show and explain the code for the data pre-processing steps, and then we will move on to the individual code applications.

Data Pre-processing

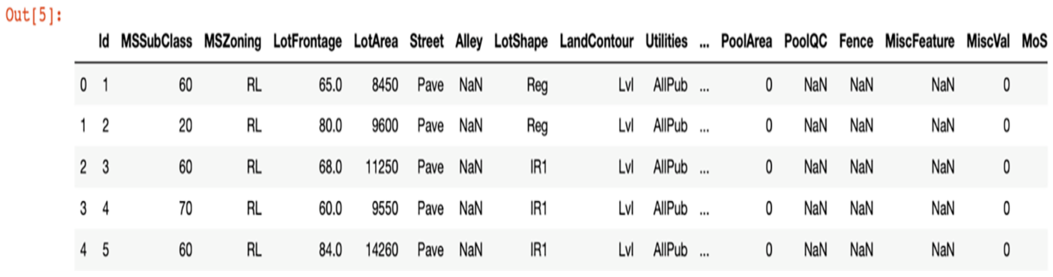
The first step is to read the dataset. We have saved the housing price dataset as `hp_train.csv` and the titanic dataset as `titanic_train.csv`. We will use a library called `pandas` to read the data files:

```
import pandas as pd

hp = pd.read_csv("hp_train.csv")

titanic = pd.read_csv("titanic_train.csv")
```

This step stores the entire information in two variables: hp and titanic. We can look at the top five rows of both the datasets using the `head()` function:

```
hp.head()
```

This gives the top five rows of the housing price dataset, as shown here:



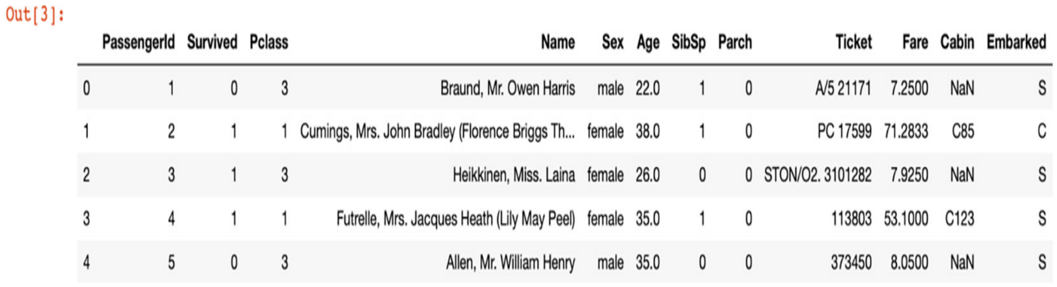
Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoS
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

Figure 8.2: *The output of head() on housing price dataset*

```
titanic.head()
```

This gives the top five rows of the titanic dataset, as shown:



Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 8.3: *The output of head() on titanic dataset*

Once we have read the dataset and stored it in the variables as a data frame, we need to remove the null values present in the data. If we keep the null values, it will lead to error, so we need to divide our dataset into a categorical and a numerical data frame before we start tackling the null values. We will replace the null values in the categorical data frame by the category that has the maximum count. Similarly, we will replace the null values in the numerical data frame by the mean of the column. We will follow one more rule saying we will delete a column that has over 60% of null values. This is not a universal rule but is generally followed by most practitioners.

We will apply the data pre-processing steps on the housing price dataset, and similar steps can be followed for the titanic dataset. Refer to this book's repository for the code applied to the titanic dataset.

```
import numpy as np
```

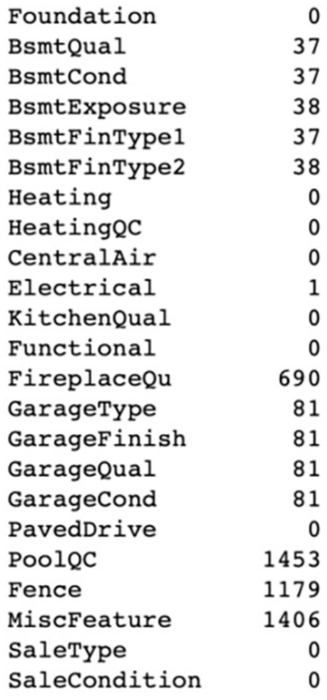
```
data_cat = data.select_dtypes(include=[object])
```

```
data_num = data.select_dtypes(include=np.number)
```

The preceding lines of code divide the data frame into two parts: `data_cat` contains categorical columns, and `data_num` contains numerical columns. Now, let's look at how many null values are present in each data frame:

```
data_cat.isna().sum()
```

You can find a list of all the columns and the number of null values present, as follows. This diagram only shows the subset of all the columns present; you can refer to the notebook in the repository for the complete output:

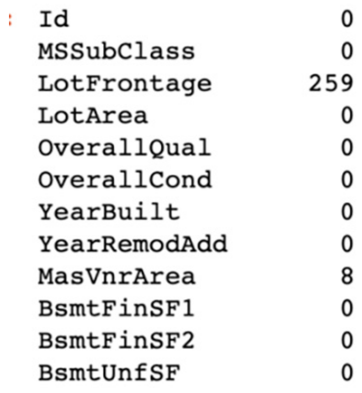


```
Foundation          0
BsmtQual           37
BsmtCond           37
BsmtExposure       38
BsmtFinType1       37
BsmtFinType2       38
Heating             0
HeatingQC           0
CentralAir          0
Electrical          1
KitchenQual         0
Functional          0
FireplaceQu       690
GarageType         81
GarageFinish       81
GarageQual         81
GarageCond         81
PavedDrive          0
PoolQC           1453
Fence            1179
MiscFeature      1406
SaleType            0
SaleCondition       0
```

Figure 8.4: *Categorical null values*

```
data_num.isna().sum()
```

Similar to the categorical columns, this code will give the following output:



```
Id                0
MSSubClass        0
LotFrontage     259
LotArea           0
OverallQual       0
OverallCond       0
YearBuilt         0
YearRemodAdd      0
MasVnrArea        8
BsmtFinSF1        0
BsmtFinSF2        0
BsmtUnfSF         0
```

Figure 8.5: *Numerical null values*

Looking at the preceding output, we can get a list of all the columns having null values. In the categorical data frame, we can find several columns having over 60% of null values. As discussed, we can delete these columns. Here's the code to delete these unnecessary columns:

```
cols_to_be_deleted = ["PoolQC","Fence","MiscFeature","Alley"]

data_cat.drop(cols_to_be_deleted, axis=1,inplace=True)
```

Once we delete these columns, we can replace the other columns with null values with the categories having maximum count:

```
cols_to_replace = ["FireplaceQu", "GarageType", "GarageFinish", "GarageQual", "GarageCond", "BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2", "Electrical","MasVnrType"]

for i in cols_to_replace:

   exec("data_cat.%s.fillna(data_cat.%s.value_counts().idxmax(),
   inplace=True)" %(i,i))
```

List `cols_to_replace` contains all the columns that have null values, and they have to be replaced. The following 'for' loop goes to each column mentioned in the list and replaces it with the categories having the maximum count. Exactly the same steps can be taken for the numerical columns:

```
cols_to_replace = ["LotFrontage", "GarageYrBlt", "MasVnrArea"]
```

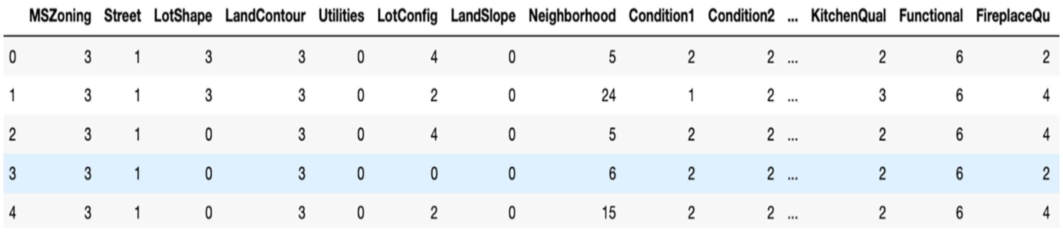
```
for i in cols_to_replace:

   exec("data_num.%s.fillna(data_num.%s.mean(), inplace=True)" %(i,i))
```

Now, we have two data frames having no null values. The next step is label encoding, a process used to represent categories with numbers. "Male, Female" can become 1,2, "Hot, Medium, Cold" can become 1,2,3 , and so on. So, for each categorical column, we will convert each category to this format. We need to do this because any model expects a numerical output, and we cannot pass strings to the model:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

data_cat_lbl = data_cat.apply(le.fit_transform)

cat = pd.DataFrame(data_cat_lbl)
```

In Scikit-Learn, we have a class called `LabelEncoder`, using which we can convert categories into numbers. After executing the preceding lines of codes, we will get a data frame having no null values, and all the strings converted into a number. We can look at the data frame using the `head()` function:

```
cat.head()
```



	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	...	KitchenQual	Functional	FireplaceQu
0	3	1	3	3	0	4	0	5	2	2	...	2	6	2
1	3	1	3	3	0	2	0	24	1	2	...	3	6	4
2	3	1	0	3	0	4	0	5	2	2	...	2	6	4
3	3	1	0	3	0	0	0	6	2	2	...	2	6	2
4	3	1	0	3	0	2	0	15	2	2	...	2	6	4

Figure 8.6: *Label encoding*

Our categorical data frame is ready, and the next thing is to process a numerical data frame. We have removed all the null values from the numerical columns. If we want, we can directly work with the same data frame, but every column has a different range of values. So, it is recommended to give a fixed range to all the columns. This can be done using two approaches: max-min normalization and standardization. After applying max-min normalization, the range of each column becomes from 0 to 1. Here's the formula using for this normalization:

$$norm = \frac{\max - X}{\max - \min}$$

Using the standardization approach, we convert the column values in the **Normal Distribution** format. So, the range becomes –∞ to +∞. The following is the z-score formula to convert the values:

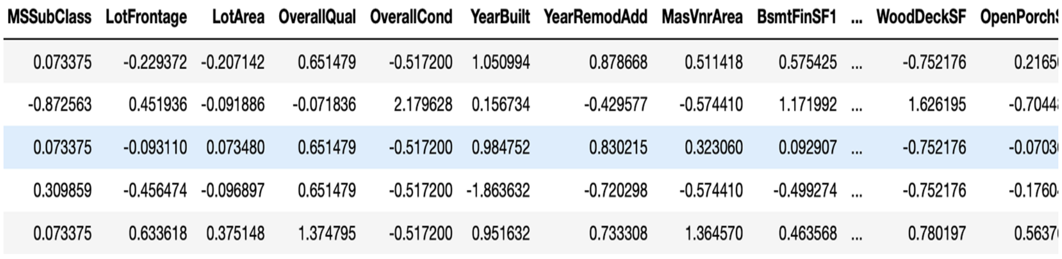
$$z - score = \frac{x - \mu}{\sigma}$$

Where μ represents the mean of the column, and σ represents the standard deviation. In this example, we will standardize our column in normal distribution format:

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

data_num_std = ss.fit_transform(data_num)

data_num_std = pd.DataFrame(data_num_std, columns=data_num.columns)
```

We can now look at the new standardized data frame using the `head()` function:

```
data_num_std.head()
```



MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...	WoodDeckSF	OpenPorch
0.073375	-0.229372	-0.207142	0.651479	-0.517200	1.050994	0.878668	0.511418	0.575425	...	-0.752176	0.2165
-0.872563	0.451936	-0.091886	-0.071836	2.179628	0.156734	-0.429577	-0.574410	1.171992	...	1.626195	-0.7044
0.073375	-0.093110	0.073480	0.651479	-0.517200	0.984752	0.830215	0.323060	0.092907	...	-0.752176	-0.0703
0.309859	-0.456474	-0.096897	0.651479	-0.517200	-1.863632	-0.720298	-0.574410	-0.499274	...	-0.752176	-0.1760
0.073375	0.633618	0.375148	1.374795	-0.517200	0.951632	0.733308	1.364570	0.463568	...	0.780197	0.5637

Figure 8.7: *Standardization*

Now, we have both categorical and numerical columns. We can combine both data frames and get the final single data frame:

```
data_final = pd.concat([data_cat_lbl, data_num_std], axis=1)
```

We'll move ahead with this data frame. Now, as our final data is ready, let's look at the application of simple linear regression on this dataset:

```
X = data_final[["MSSubClass"]]

Y = data[["SalePrice"]]
```

From the final data frame, we have selected the `MSSubClass` column as `X` and `SalePrice` column as `Y` because that is our target variable, which we'll be predicting using different regressions. One thing to be brought into notice is that the `SalePrice` column for `Y` is taken from the original dataset and not the final data frame. This is because we have standardized the final data frame, including the `SalePrice` column. So, the prediction will also be standardized if we use the standardized column, which we don't want. Next, we'll bifurcate the dataset into the train and test set. We will train the model on the train dataset and check the model performance on the test dataset:

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test   =   train_test_split(X,Y,test_size=0.1,
random_state=1234)
```

`test_size = 0.1`, in this line, we have directed Python to randomly take 10% of the total data as the training set. To control the randomness, we have defined `random_state`, which uses the concept of seeding. Next, we'll create the object of linear regression and apply the algorithm on the train set:

```
lr = LinearRegression()

lr.fit(x_train, y_train)
```

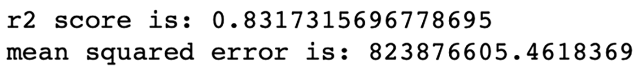
Once we have applied the algorithm, we need to make predictions on the test set:

```
pred_results =lr.predict(x_test)
```

Using these predictions, we can check the accuracy of the model:

```
from sklearn.metrics import r2_score, mean_squared_error

print("r2 score is:", r2_score(pred_results, y_test))

print("mean squared error is:", mean_squared_error(pred_results, y_test))
```

This code will give us the following output:



```
r2 score is: 0.8317315696778695
mean squared error is: 823876605.4618369
```

Figure 8.8: *Output*

So, this model gives us an R-Squared value of 83% with MSE of 823876605.

Now that we know how Simple Linear Regression operates, let's move on to the concept of Multiple Linear Regression, where we have multiple variables instead of a single predictor variable.

Multiple Linear Regression

Take a look at the following image:

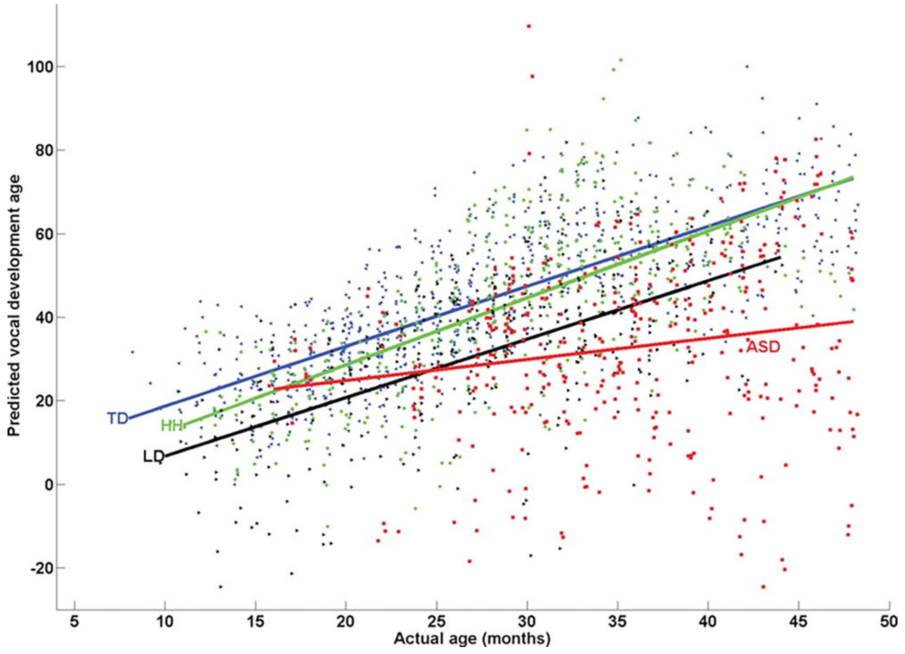


Figure 8.9: *Multiple Fitted Lines*

We have seen that we only had one predictor variable in simple linear regression, and another one was predicted one. But there are multiple predictors that help make the predictions. This is where multiple linear regression comes into the picture. However, is there any other option apart from directly moving to MLR? Well, we can make multiple simple linear models for each new predictor variable. This can give multiple predictions, but it will be difficult to come up with a way that combines the predictions of all models and provides a single prediction. Another issue is that predictions are based on the effect of individual predictors on the predicted variable. However, it can’t take into account the combined effect of all the predictor variables on the target variables. So, we can conclude that applying a simple linear model on problem statements with multiple predictors is not the right option.

Multiple linear regression is an extension of a simple linear regression model and can be represented by the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \in$$

Numerical predictions are made using the preceding equation. Just like simple linear regression, all the coefficients are found using the method of the least squares approach. In this method, we minimize the **Residual Sum of Squares** (**RSS**) equation to provide the best coefficients value. The following is the RSS equation that needs to be minimized:

$$RSS = \sum_{i=1}^{n}(y_i - \overline{y}_i)^2 = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} \ldots - \hat{\beta}_p x_{ip})^2$$

Since there are several variables, some of them may not be important. To determine which variables are unimportant, we can run an F-Statistic test on the entire model. This leads to generating different p-values for all the variables. The variables with a p-value greater than 0.05 (if we have considered a confidence interval of 95%) are the unimportant ones and can be removed from the model. However, this approach is not the best one, as some variables affect the model only minimally. If we delete them based upon the p-value, their effect will also be gone. Other methods can provide much better performance.

We can opt for the Subset Selection approach, where we can use the forward, backward, or hybrid selection approaches to select the best subset of variables. We can also go for regularization-based regression, like **Ridge Regression**, **Lasso Regression**, or **Elastic Net regression**. They penalize the variables based on their importance in the model. We will discuss all these approaches in the next few sections and determine which model is the best one using metrics like adjusted-R2, Akaike's Information Criterion, and Bayesian Information Criterion.

We'll now look at the Python application of Multiple Linear Regression. As we have already processed the data in the Simple Regression section, we will directly move to training. In this algorithm, we need all the columns instead of only one, as with simple linear regression. So, we will make a slight change in the code line and allocate all the independent variables as `X` and the duringthe `SalesPrice` variable as `Y`:

```
X = data_final.drop([“SalePrice”],axis=1)

Y = data[[“SalePrice”]]
```

Now that we got our dependent and independent variables, we can make our train and test set just like we did in the previous section:

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.1,
random_state=1234)
```

The next step is to apply multiple linear regression. We can apply both the types of regression using the `LinearRegression` class of the scikit-learn package. The next few lines of code are similar to the ones we have already seen:

```
#Initializing the algorithm
lr = LinearRegression()

#Fitting the algorithm
lr.fit(x_train,y_train)

#Predicting the results
pred_results =lr.predict(x_test)
result_df = pd.DataFrame(data={'SalePrice': pred_results})

#Checking the performance
print("r2 score is:", r2_score(pred_results, y_test))
print("mean squared error is:", mean_squared_error(pred_results, y_test))
```

Linear regression performs really well if the data follows a linear pattern, but data possess a non-linear pattern most times, and applying linear regression makes the model very bad. If we still want to use linear regression and add the capacity to tackle non-linear data, polynomial regression is the solution. This is an extension to the linear regression concept, which we'll discuss in the next section.

Polynomial Regression

The following image shows the polynomial regression curve:

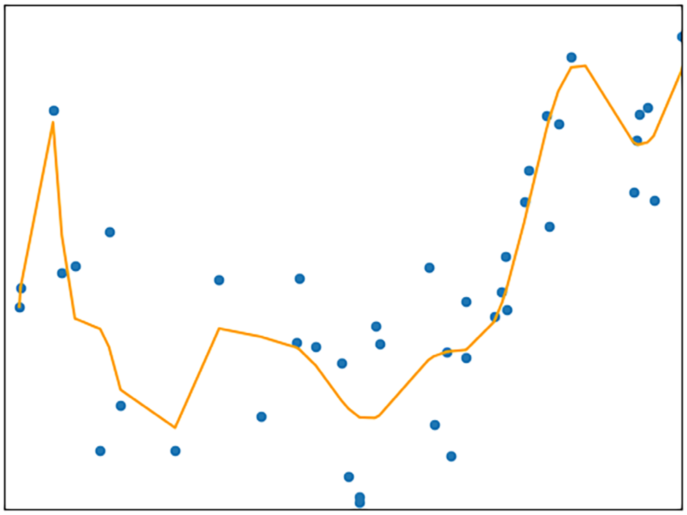


Figure 8.10: *Polynomial curve fitting*

In the previous sections, we saw that the linear regression model can only fit a straight line, regardless of the shape of the dataset. This constraint causes the model to perform poorly if the dataset is non-linear. We can try to add non-linear capability to linear regression with the help of polynomial regression.

Polynomial regression can be seen as an extension to linear regression that tries to add some extra parameters by raising the original ones to some power. This power is termed as the degree of polynomial regression. For example, a cubic polynomial regression means that three variables will be generated for each predictor variable, as follows:

Predictor variables in linear regression = X_1, X_2, X_3

Predictor variables in polynomial regression will be transformed to:

$$X_1, X_1^{\ 2}, X_1^{\ 3}, X_2, X_2^{\ 2}, X_2^{\ 3}, X_3, X_3^{\ 2}, X_3^{\ 3}$$

This method helps us create additional predictor variables and also give the linear line curve shape, providing a non-linear fit to data.

Consider the following equation:

$$y_i = \beta_0 + \beta_1 x_i + \in$$

It is revised to:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^{\ 2} + \beta_3 x_i^{\ 3} + \ldots \beta_n x_i^{\ n} + \in$$

Let's look at the application of polynomial regression on the same housing price dataset using Python. We will start directly from the application of the algorithm on the processed dataset, as in the previous section.

First, we will import the required package. As polynomial regression is not a new type of regression but an extension of linear regression, we have to import both the packages:

```
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import LinearRegression
```

The next step is to define the degree of the model. As seen in the explanation of the algorithm, a degree of 3 means three additional variables are declared and 4 means four additional variables are declared. In this example, we will continue with the degree of 3:

```
poly_deg3 = PolynomialFeatures(degree=3)
```

Also, we need to create an object of linear regression:

```
lr = LinearRegression()
```

The next step is to create new variables based on the degree. For this, we need to apply the degree 3 objects created for `PolynomialFeatures` on to the training and test set. The `fit_transform()` function helps us achieve this task:

```
X_train_poly_3 = poly_deg3.fit_transform(x_train)
```

```
X_test_poly_3 = poly_deg3.fit_transform(x_test)
```

Now that our different sets are ready, we can fit the linear regression model on this newly created variable:

```
lin_reg_poly_3 = lr.fit(X_train_poly_3, y_train)
```

Since the model is fitted on the dataset, we can move on with the predictions and check the accuracy, as we saw in the previous sections:

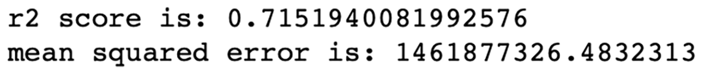
```
pred = lin_reg_poly_3.predict(X_test_poly_3)
```

```
from sklearn.metrics import r2_score, mean_squared_error
```

```
print(“r2 score is:”, r2_score(pred, y_test))
```

```
print(“mean squared error is:”, mean_squared_error(pred, y_test))
```

The final output of this code will be as follows:



```
r2 score is: 0.7151940081992576
mean squared error is: 1461877326.4832313
```

Figure 8.11

We can see that the accuracy has come down from the linear regression output, so the polynomial model is not recommended for this dataset.

Now that we know how to tackle the problem of non-linearity in the dataset, let's take the second problem faced by linear regression. Another problem with linear regression is that problems of overfitting arise as we increase the number of predictor variables. Also, there may be variables that are not helping the model but linear regression counts them as important variables. Apart from these reasons, there can be a problem with interpretability. A large number of variables cause confusion, causing estimates to have low bias but very high variance.

If we can penalize these unnecessary variables, a major part of this problem is solved. There is a series of regression algorithms with the same aim. We'll discuss some of them in this chapter:

- Subset selection
- Ridge regression
- Lasso regression
- Elastic Net regression

Let's understand these regressions sequentially.

Subset Selection Method

This is the first method to solve the problem of too many variables. As mentioned earlier, a high number of variables can cause various kinds of problems. In subset selection, we try to find the best subset of variables that can give us the least error.

In this method, we will try to determine all the possible combinations for *p* predictors and fit regression models over them. Then, based on the residual sum of squares analysis, we try to find the model that gives us the best result. The number of models required for *k* predictors out of *p* total predictors will be:

$$\binom{p}{k} models for k predictors out of p total predictors$$

This method tries to incorporate all the possible combinations to give us the best model, so it may be a time-consuming process. This approach is further divided into two kinds:

- Forward subset selection
- Backward subset selection
- Hybrid subset selection

We already discussed the forward subset selection approach, wherein we start with no predictors and then, one by one, we look at all the possible combinations of variables and the models based on them. Minimum **Residual Sum of Squares** (**RSS**) score gives the best model. In the **Backward Subset Selection** approach, we start with all the *p* predictors and make models and slowly remove predictors one by one and compare the RSS score. Again, the minimum score is the best model. In the hybrid approach, we start with 1 predictor and keep adding predictors and check the RSS. If RSS increases by adding a predictor, we will immediately dump that predictor and look at other combinations.

Ridge Regression

The following image shows ridge regression:

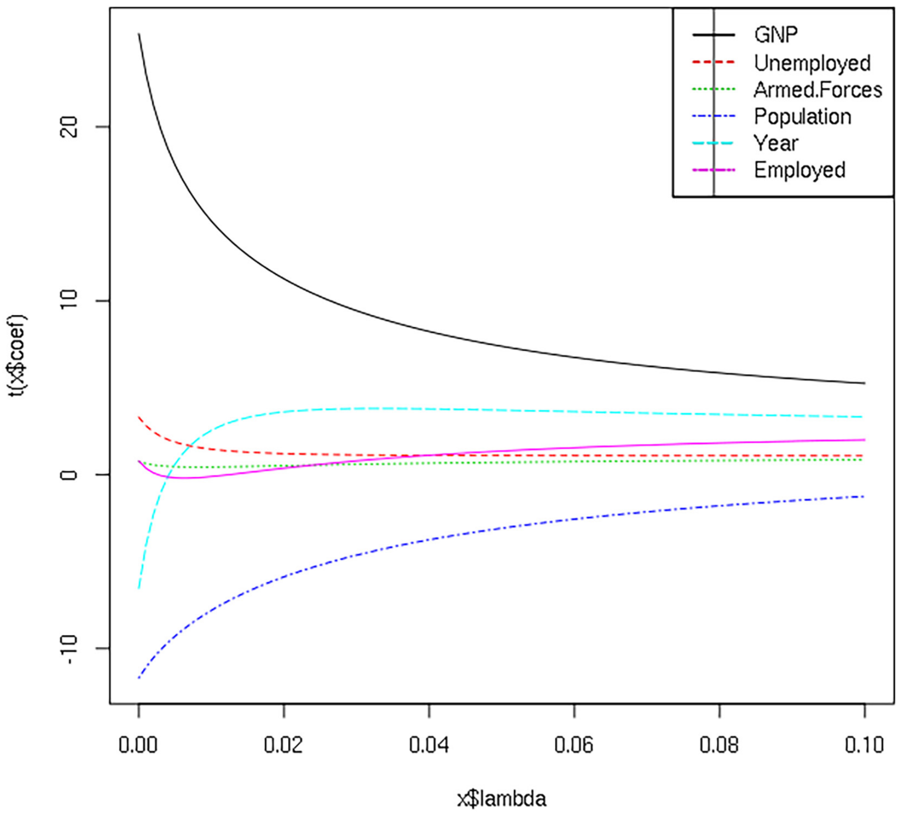


Figure 8.12: *Ridge Coefficients Saturation*

In the **subset selection method**, we found the subsets of variables that are important and omitted all the others. But, in general scenario, it may not be the case where either variable is very important, and we should keep it, or the variable is not at all important, and we should remove it. There should be a way to penalize variables based on their importance. The most important variable can be kept as is, while the others can be scaled down using a factor based on their importance. Ridge regression is one such method of achieving this.

Here, we define the importance of a variable using a tuning parameter. If the variable is very important, the value of the tuning parameter will be high, but if less important, this variable will try to shrink down the value of the variable. So, this variable is also called the shrinkage factor.

In linear regression, we find the value of its coefficients using the method of the least squares. There, we find the sum of the square of errors and then try to determine the value of the coefficients that minimize this error. Ridge regression also follows the same approach, but the function to be minimized is slightly different and is given as follows:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right) + \lambda\sum_{j=1}^{p}\beta_j^2$$

The preceding equation shows that it consists two parts. The first part, before the addition, is similar to our least squares estimation method of linear regression. Here, we try to minimize this to get the best value of the coefficients. The second part of the equation makes this approach a little different from linear regression and tries to shrink the coefficients of the variables so that the entire equation becomes minimum. This is the part that provides importance or unimportance to the variables and so, is called the shrinkage penalty. However, if we make this second part go free, it will make all the coefficients reach the zero mark, damaging the model. So, we have an additional parameter called λ This, which controls the amount of penalty that needs to be applied to the variables.

An important part of Ridge regression is that the effect of λ enables us to shrink the parameters as much as possible without reaching 0. Mathematically, the variables reach 0 at the infinity value of λ. This can be referred by:

$$\textit{as } \lambda \rightarrow \infty, \textit{ coefficients reach } 0$$

Let's look at the application of Ridge regression using Python. Again, we will use the processed dataset. The only difference in this code is that we will create the object of Ridge Regression function: `Ridge()`. All the other lines are exactly the same:

```
#Initializing the algorithm
r = Ridge()

#Fitting the model
r.fit(x_train, y_train)

#Predicting the results
pred_results =r.predict(x_test)

#Evaluating the performance
print("r2 score is:", r2_score(pred_results2, y_test))
print("mean squared error is:", mean_squared_error(pred_results2, y_
test))
```

This will give us the following output:

```
r2 score is: 0.8358301504130085
mean squared error is: 789394229.9201303
```

Figure 8.13

So, we can conclude that Ridge regression is performing better than both Multiple Linear Regression and the polynomial extension of it.

Lasso Regression

The following image shows the Lasso coefficient saturation:

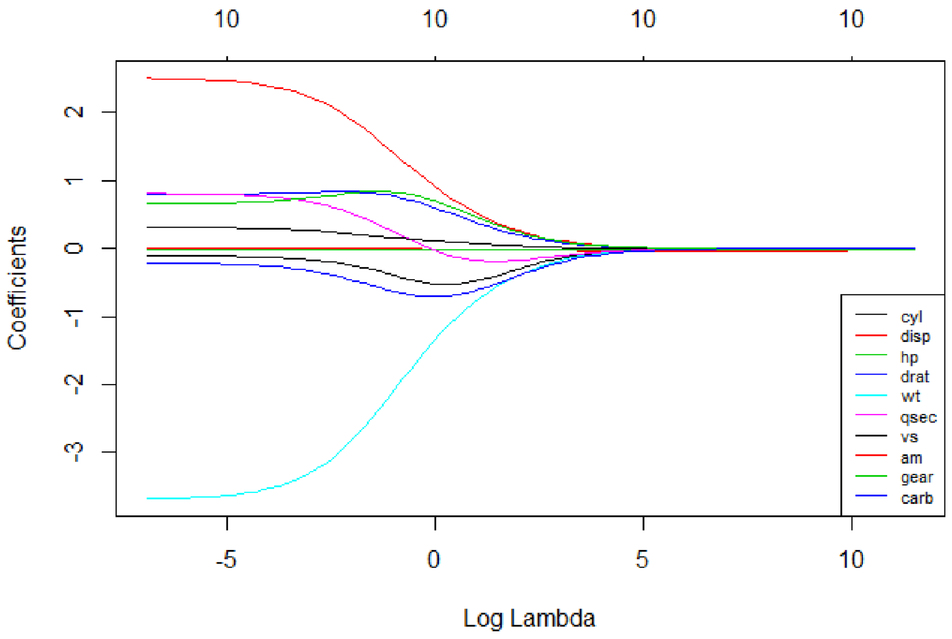


Figure 8.14: *Lasso coefficients saturation*

In the previous section, we saw how Ridge Regression helps in coefficient shrinkage of its variables, but it doesn't drop them to 0. Similarly, we saw that the Subset Selection approach directly removes the variables but doesn't penalize variables based on their importance. **Lasso regression** is like the best of both worlds. It penalizes variables like Ridge and drops them like the subset.

LASSO stands for **Least Absolute Shrinkage and Selection Operator**. To understand the process, the data should follow some assumptions similar to linear regression:

- All the predictor variables must be independent of each other
- There must be some kind of conditional dependence between the predictor and the predicted variables
- All the independent variables must be standardized

The following is the mathematical definition of a LASSO estimate to shrink the coefficients:

$$(\overline{y}, \beta) = argmin \sum_{i=1}^{n} \left(y_i - \overline{y} - \sum_{j=1} \beta_j x_{ij} \right) subjectto \sum_{j} | \beta_j | \leq t$$

Where *t* is the tuning parameter similar to the Ridge Regression. The LASSO shrinks the coefficients so that some reach zero well before others, as λ gets large. Let's see how the coefficients are affected when lambda values are changed:

- When λ = 0, no parameters are eliminated
- As λ increases, more and more coefficients are set to zero and eliminated
- As λ increases, bias increases
- As λ decreases, variance increases

Let's look at the application of LASSO regression using Python:

```
#Initializingthealgorithm
ls=Lasso()

#Fittingthealgorithm
ls.fit(x_train,y_train)

#Predictionofresults
pred_results1=ls.predict(x_test)
result_df=pd.DataFrame(data={'SalePrice':pred_results})

#Checkingtheperformance.
print("r2scoreis:",r2_score(pred_results1,y_test))
print("meansquarederroris:",mean_squared_error(pred_results1,y_test))
```

```
r2 score is: 0.8357629507141869
mean squared error is: 789810522.8567586
```

Figure 8.15

As we can see in the output of the preceding code, LASSO provides a similar output to that of Ridge Regression. In this case, both the models give us better results, and we can choose any of them.

ElasticNet Regression

The following image is about elasticnet regression:

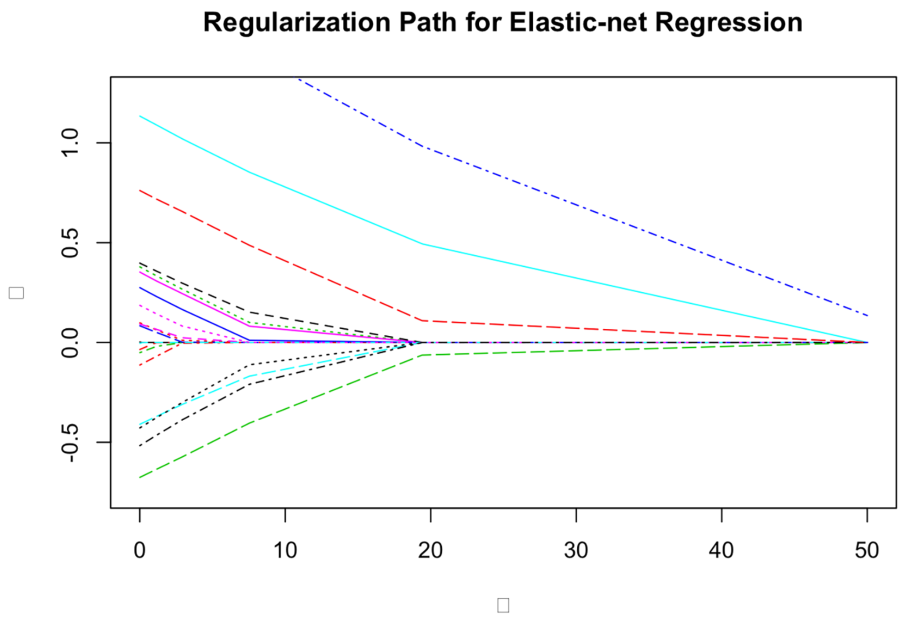


Figure 8.16: *ElasticNet coefficients saturation*

We have seen how LASSO regression solves the disadvantages of a subset or ridge regression. It is considered a good regression algorithm, but LASSO also faces some disadvantages, which elasticnet tries to solve. Before moving to the elasticnet, let's look at some of the disadvantages faced by LASSO regression:

- If the number of variables is higher than the number of observations, , in this scenario, LASSO fails because it can only select n variables before it saturates.
- If there are variables having a very high correlation between them, LASSO selects only one of them, leaving the other as is. This can cause issues.

- Again, if there is a very high correlation among predictors, LASSO's performance becomes majorly dominated by ridge regression.

Elasticnet provides the solution to these problems where it automatically selects the variables, performs continuous shrinkage, and also selects from the group of highly correlated variables.

The following the same assumptions of LASSO and Ridge, and Elasticnet regression can be mathematically defined as follows:

$$L(\lambda_1, \lambda_2, \beta) = | y - X\beta |^2 + \lambda_2 | \beta |^2 + \lambda_1 | \beta |$$

where,

$$| \beta |^2 = \textstyle\sum_{j=1}^{p} \beta_j^2$$

$$| \beta |^2 = \sum_{i=1}^{p} | \beta_j |$$

The preceding equation is minimized to get the coefficient values. Here, Elasticnet has two parameters instead of just one. These parameters are used for the shrinkage, and together, they are termed as the elasticnet penalty. Mathematically, this penalty can be represented as follows:

$$(1 - \alpha) | \beta | + \alpha | \beta |^2$$

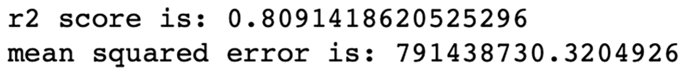
For different values of α, elasticnet performs differently. Let's look at some of its versions:

- For the value of $\alpha = 1$, elasticnet performs exactly like ridge regression
- At $\alpha = 0$, it performs like LASSO regression
- Between these values, it has the combined features of both ridge and LASSO

Let's look at the Python application of elasticnet:

```
#Initializing The Algorithm
ENet=ElasticNet()

#Fitting The Algorithm
ENet.fit(x_train,y_train)

#Predicting The Results
pred_results=ENet.predict(x_test)
result_df=pd.DataFrame(data={'SalePrice':pred_results})
```

```
#Checking The Performance
fromsklearn.metricsimportr2_score,mean_squared_error
print("r2scoreis:",r2_score(pred_results,y_test))
print("mean squared error:",mean_squared_error(pred_results,y_test))
```



```
r2 score is: 0.8091418620525296
mean squared error is: 791438730.3204926
```

Figure 8.17

As seen in the output, the results are not similar to those of Ridge and LASSO, but they are still better than the polynomial features. So, we can go with Ridge or LASSO.

One thing to remember here is that all these functions are customizable. We have not given any parameters, and Python has come up to these results automatically by finding the best in the model. Instead, we can give our own parameters and drive the model in the desired direction to get better results. But we must note that the results may also be poor. You can read the documentation of the scikit-learn package to understand how to explore these parameters.

All the approaches we saw earlier are majorly used for numerical predictions. Regression fails when it comes to classification problems, we need to predict the best category instead of predicting the actual numbers. Let's understand why linear regression fails when it comes to categorical predictions, and then we will understand how logistic regression solves the problem.

Logistic Regression

The following image shows the representation of classification problems:

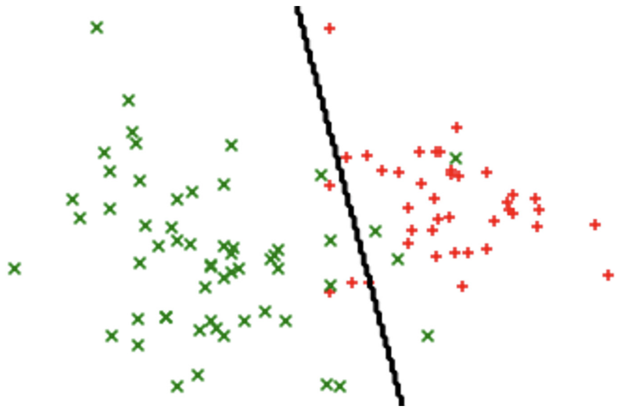


***Figure 8.18:** Classification problems*

Let's suppose that we want to predict whether a student will be an average performer, the best performer, or the worst performer. The first step to solving this problem is to label these categories. Suppose we provide the labels as follows:

Best performer	1
Average performer	2
Worst performer	3

Table 8.2

When we apply linear regression, the model predicts a numerical value. It may predict 4.6, or 2.5, and so on. In this scenario, the predictions may not fall into one or the other category, and we have to depend too much on the approximations. Suppose we change the dummy coding to the following:

Best performer	3
Average performer	2
Worst performer	1

Table 8.3

In this case, the model will change and give different types of results. In reality, these numbers only refer to the categories and do not mean anything else, but linear regression takes it as a numerical value and makes the predictions.

Another issue is that linear regression has different ranges. Let's look at the following image:

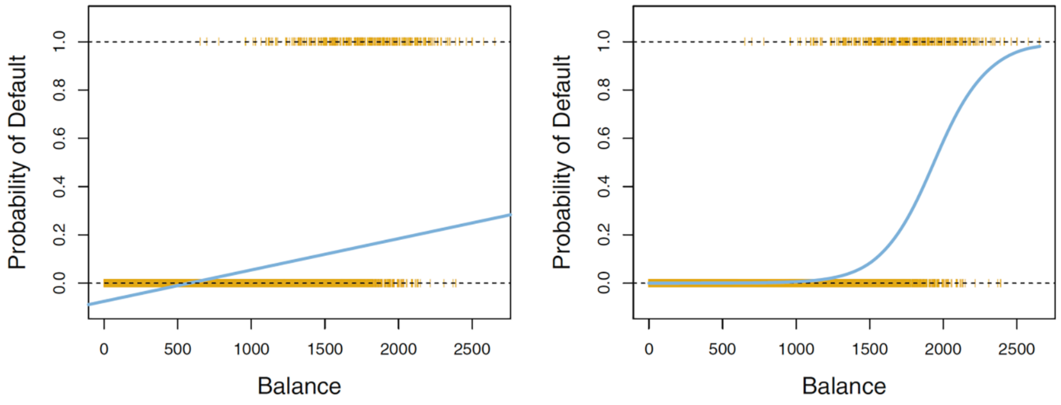


Figure 8.19: *Linear regression vs. logistic regression curve*

Suppose we want to predict whether a person will default in paying back the loan. On the left-hand side in the image is the output of linear regression. If we want to make the analysis by counting the predictions as probabilities, you can see that the linear regression line is deviating to the negative side as well. Probabilities can never be negative. Also, the linear line crosses 1 many a time, again defying the properties of probability.

But if we get the output curve as in the right-hand side image, it gives better probability predictions. This S-shaped curve is termed as the **Logistic Regression Model**.

So, we can say that Logistic Regression models the probability that Y belongs to a particular category instead of modelling the response Y directly. If we try to understand this based on conditional probability, we can suffice the algorithm with the following equation:

$$P(default = Yes \mid Occupation)$$

This equation tells us about the probability of a person defaulting if we know their occupation. Similarly, we can have multiple pieces of information about the person and get the output probability of them defaulting. This is the major intuition behind logistic regression.

The logistic regression equation is derived from linear regression, and the revised format is denoted by the following equation. This equation is also called the logistic function:

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

If we draw the curve of this equation, we'll get an S-shaped curve, as illustrated in the `figure()`. This curve is also called the Sigmoid curve. The preceding equation can be modified further to gives us the following equation:

$$\frac{p(X)}{1 + p(X)} = e^{\beta_0 + \beta_1 X}$$

The left-hand side of this equation, $\frac{p(X)}{1+p(X)}$, is called the odds. The range of the odd is 0 to ∞. If we take the same example of default, a nod value closer to 0 means less probability of defaulting, while a higher odd value denotes a higher probability of defaulting.

If we further modify the equation by taking the logarithm on both the sides of the equation, we have an equation termed as the log-odds equation or log it function. The equation is as follows:

$$\log\left(\frac{p(X)}{1-p(X)}\right)=\beta_0+\beta_1 X$$

Estimation of Parameters

Using the training data provided, we can estimate the values of β0 and β1 in the logistic function equation. We do this using the Maximum Likelihood Estimation approach. The following is the likelihood function equation that needs to be maximized to get the best value of the coefficients:

$$l(\beta_0,\beta_1)=\prod_{i:y_i=1} p(x_i)\prod_{i':y_{i'}}(1-p(x_{i'}))$$

In linear regression, we used the method of least squares to get the estimation of the coefficients. That method is a special case of the preceding function of maximum likelihood estimation.

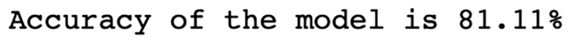
Let's look at the application of Logistic Regression on the Titanic dataset. This dataset talks about all the people who were on the ship at the time of the accident. This includes different features like age, gender, name, ticket, and so on. The dependent variable of this dataset is the survived column, which tells us whether the person survived the accident. We will use this dataset for the logistic regression to predict the same, that is, whether a person survived.

We have already seen how to pre-process a dataset. In this section, we'll look at the pre-processed dataset. To know how we processed it, refer to the jupyter notebook of this chapter. As seen in the previous sections, we need to create the object of logistic regression, fit the model, predict the values on the test set, and then check the accuracy. Let's look at the code for this:

```
#InitializingtheAlgorithm
lr=LogisticRegression()

#FittingtheModel
lr.fit(x_train,y_train)

#Predictions
pred=lr.predict(x_test)
```

```
#CheckingthePerformance
fromsklearn.metricsimportaccuracy_score
print("Accuracyofthemodelis{0:.2f}%".format(accuracy_score(pred,y_
test)*100))
```

Here, we look at the accuracy score instead of the R2 score or MSE. This tells us how many predictions made on the test dataset were correct. The following is the output:



```
Accuracy of the model is 81.11%
```

Figure 8.20

We got the accuracy of 81.11% for this model, which is decent enough.

Understanding Residuals

Till now, we covered the different types of regression models and their execution in Python, but it is important to understand the model's adequacy along with its performance. This means understanding how efficiently we can generalize our model to the entire population. For this, we try to understand and visualize residuals in further detail.

The following assumptions should be followed when we talk about linear regression or any of its versions:

1. Dependent variable and all the independent variables must have a linear relationship
2. The residual should have zero mean and constant variance
3. All the residuals must not be correlated
4. Residuals must be normally distributed

If the model does not follow these assumptions, it is not adequate. So, we must check the validity of these assumptions before making a decision related to the model. Small variations in following the assumptions are allowed, but large variations can lead to unstable models. This means if we take different samples, the model result for each sample will be different, and it may also give the opposite conclusion. Understanding residuals is important to ensure that we're well on track on all the assumptions.

As mentioned earlier, residuals are defined as the difference between the predictions and the original value. Mathematically, they can be defined as the deviation between the data and the fit. We can say that whatever response the model is unable to explain is termed as a residual.

A few properties of residuals can be:

- They have zero mean
- Their approximate average variance can be estimated using:

$$MS_{\mathrm{Res}} = \frac{\sum_{i=1}^{n}(e_i - \bar{e})}{n-p}$$

We can directly work with the residuals, or we can scale them down beforehand. This scaling can prove helpful in finding the observations that are outliers, which are somewhat different from the normal data. Standardized residuals are those that can be scaled down using the following formula:

$$d_i = \frac{e_i}{\sqrt{MS_{\mathrm{Res}}}}$$

If we use the following formula instead of using the preceding one, it is called Studentized Residuals:

$$r_i = \frac{e_i}{\sqrt{MS_{\mathrm{Res}}\left[1-\left(\frac{1}{n}+\frac{(x_i-\bar{x})^2}{s_{xx}}\right)\right]}}$$

Both these methods are perfect for finding outliers. Two other methods called **PRESS Residuals** and **Rstudent residuals** are also good choices, but they are outside the scope of this book. Now, let's look at how we can analyze residuals to derive a conclusion about the model.

Plotting the residuals gives us a lot of insights. Normal probability plots are used to determine whether the normality assumtions are being followed, as a confidence interval, prediction interval, and various statistics depend on this assumption. After plotting this diagram, finding that the tails are thicker and heavier may affect the least squares estimation. Heaviness tends to pull the least squares estimates towards their direction, making the model biased.

In this diagram, we consider Studentized Residuals. The following is the Python code to make a normal probability plot. This code is applied to the output of multiple line arregression code that we saw in this chapter.

Let's import the important packages first. Scipy contains most of the statistical and mathematical algorithms, while `matplotlib` is used for drawing plots:

```
importscipy

importmatplotlib.pyplotasplt
```

Residuals are the difference between the predictions and the original values. Let's calculate the residuals based on the predictions:

```
resid=pred_results-y_test
```

The normal probability plot is generally a plot of cumulative frequencies of the residuals. In the next step, we'll calculate the cumulative frequency, and then we'll plot them using `matplotlib`:

```
counts,start,dx,_=scipy.stats.cumfreq(resid,numbins=20)

x=np.arange(counts.size)*dx+start


plt.plot(x,counts,'ro')

plt.xlabel('Value')

plt.ylabel('CumulativeFrequency')


plt.show()
```

We can see the following output:

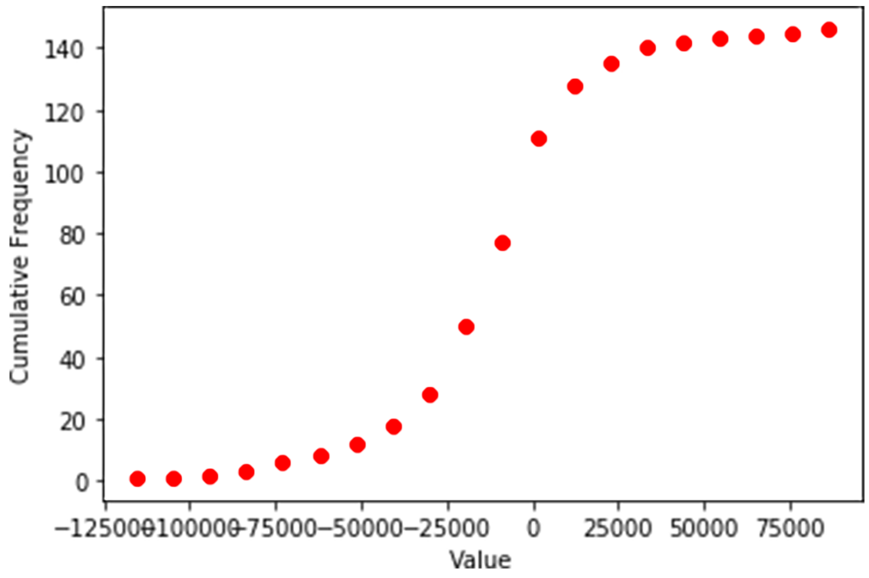
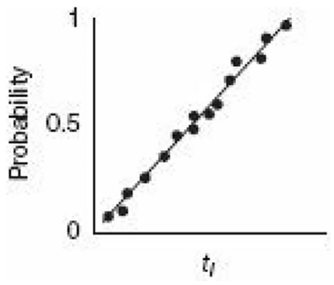
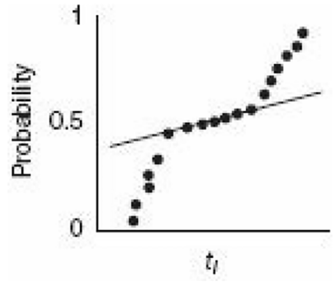


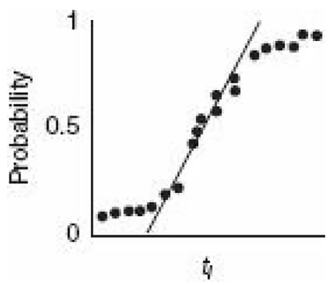
Figure 8.21: *Residual plot*

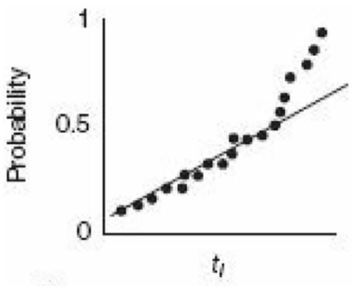
This output shows that the residuals are light-tailed. A normal probability plot can vary, and the following are some of the versions. When approximately all the points lie on the straight line, it is an ideal normal probability plot. When you can see that the points are gradually increasing on the tails, it is a light-tailed plot. However, if the increase is sudden, it is a heavy-tailed plot. Finally, if the data is dispersed too much in the top or bottom part of the plot, it represents skewness in the data.





Ideal plot light -tailed distribution





Heavy-tailed distribution - Positive skew

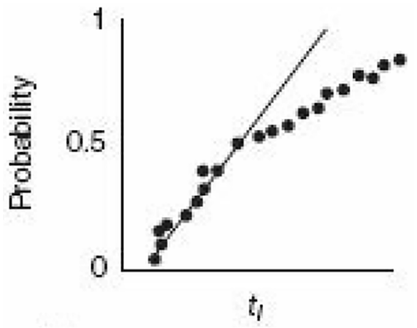


Figure 8.22: *NegativeSkew*

Patterns of Residuals

We can also plot residuals normally and gain a lot of insights. The following are the different versions of residual plots that we can see in a normal scenario. The first image illustrates the best residual plot. Ideally, a residual plot should show no pattern. The second image shows that the variance of the residuals increases with time, which is the pattern of **heteroscedasticity**. One can apply a suitable transformation of data to apply linear regression in this scenario. The third image shows a double bow pattern, which means the variance of a binomial proportion near 0.5 is greater than the one near 0 or 1. We can apply suitable transformation sto apply linear regression here as well. Finally, the last image shows non-linearity, where no transformation of variables can happen. We can either go for regression having polynomial features or some other non-linear algorithms.

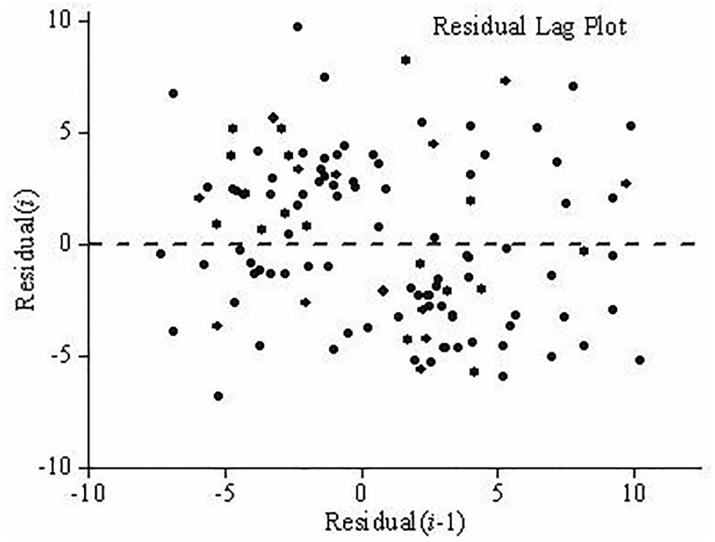


Figure 8.23: *Scattered residual plot*

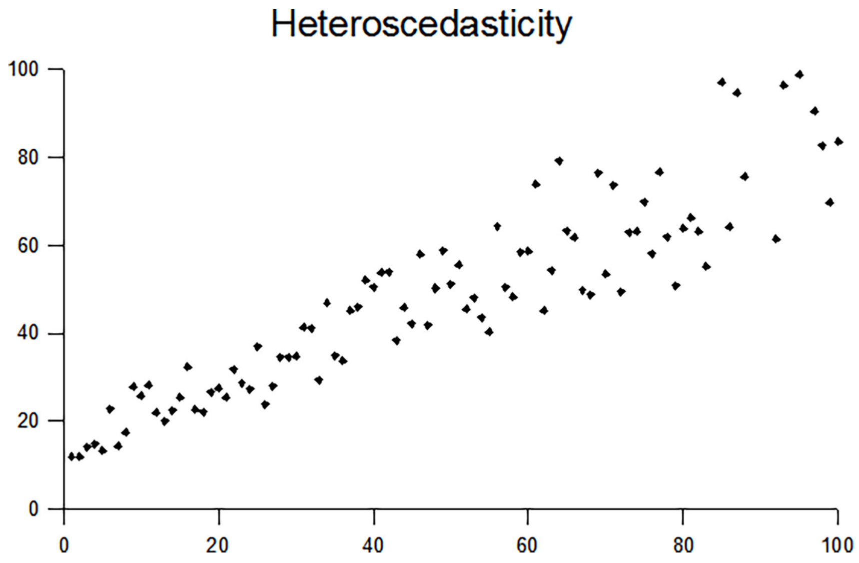


Figure 8.24: *Heteroscedastic residual plot*

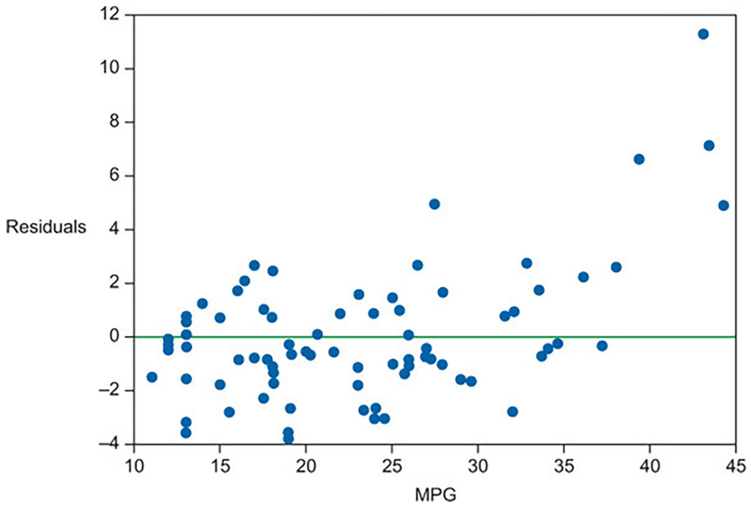


Figure 8.25: *Double bow residual plot*

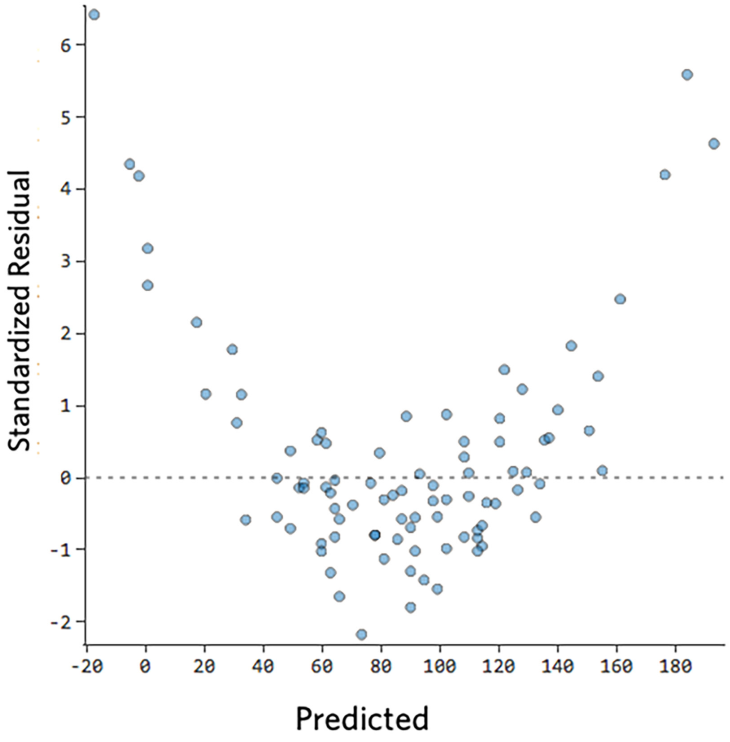


Figure 8.26: *Non-linear residual plot*

Another issue the model may have to deal with is **Multicollinearity**, which will be covered in the next section.

Multicollinearity

If a subset of independent variables is correlated to each other, it can cause our model to perform badly. This situation wherein some variables are correlated is called **multicollinearity**. If the variables have no relationship with each other, the set is termed as that of sorthogonal variables.

As this is an ideal condition where all the variables are orthgonal, some relationships are allowed to make it near ideal. However, when these relationships are very strong, sometimes even perfectly linear, we need to deal with this problem of multicollinearity.

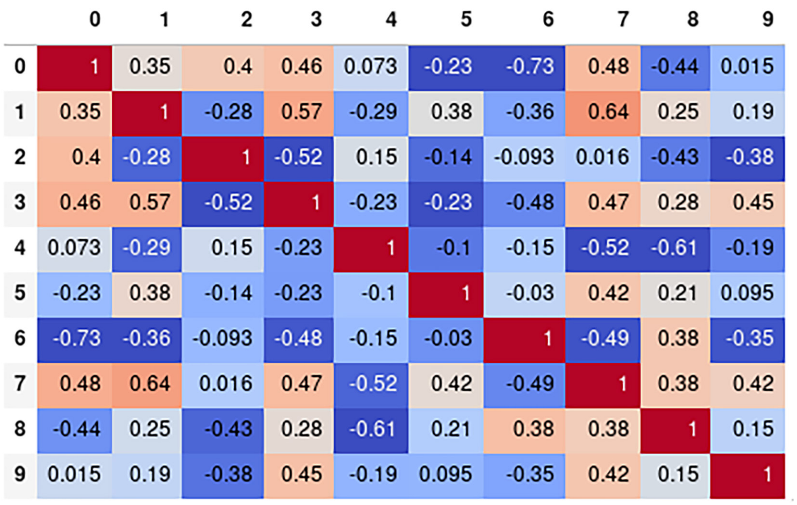
There are multiple reasons why multicollinearity may exist in a dataset:

1. The person who collected the data didn't do the sampling perfectly.
2. We can only apply some experiments on specific genres, for example, analysis done only on older people. This type of constraint on the model or population can cause multicollinearity to seep in.
3. We can add some extra parameters in the model, which may be correlated to other variables, for example, adding polynomial features. Sometimes, we may have a high number of independent variables as compared to the total number of observations, which may give rise to multicollinearity.

Some of the recommendations given to solve the issue of multicollinearity are:

- Take only the subset of independent variables. We can use the subset selection approach as well.
- We can make use of shrinkage parameters and related algorithms like a ridge, LASSO, or elasticnet regression.
- We can make a dimensionality reduction of the data and then apply regression over it.

How can we determine the presence of multicollinearity in a dataset? One approach is to make a correlation matrix, wherein all independent variable combinations are analyzed in the form of a matrix. The diagonal means correlation of the variables with themselves, so the value is always 1. So, we only take care of off-diagonal members. Variables showing very high correlation with each other can be dropped off. The following is one type of correlation matrixoutput in Python:



	0	1	2	3	4	5	6	7	8	9
0	1	0.35	0.4	0.46	0.073	-0.23	-0.73	0.48	-0.44	0.015
1	0.35	1	-0.28	0.57	-0.29	0.38	-0.36	0.64	0.25	0.19
2	0.4	-0.28	1	-0.52	0.15	-0.14	-0.093	0.016	-0.43	-0.38
3	0.46	0.57	-0.52	1	-0.23	-0.23	-0.48	0.47	0.28	0.45
4	0.073	-0.29	0.15	-0.23	1	-0.1	-0.15	-0.52	-0.61	-0.19
5	-0.23	0.38	-0.14	-0.23	-0.1	1	-0.03	0.42	0.21	0.095
6	-0.73	-0.36	-0.093	-0.48	-0.15	-0.03	1	-0.49	0.38	-0.35
7	0.48	0.64	0.016	0.47	-0.52	0.42	-0.49	1	0.38	0.42
8	-0.44	0.25	-0.43	0.28	-0.61	0.21	0.38	0.38	1	0.15
9	0.015	0.19	-0.38	0.45	-0.19	0.095	-0.35	0.42	0.15	1

Figure 8.27: *Correlation matrix*

The following is the Python code to make a correlation matrix:

```
importmatplotlib.pyplotasplt

f=plt.figure(figsize=(19,15))
```

Finding the correlation matrix of all the columns won't be visually appealing, so we'll work with a subset of the data:

```
data_sub=data_final.iloc[:,0:8]
```

Now that we have the data, we need to find each column's correlation with every other column. This can be done using the `corr()` function provided by pandas. We can plot these values in a matrix format using the `finctionmatshow()` provided by `matplotlib`:

```
plt.matshow(data_sub.corr(),fignum=f.number)

plt.xticks(range(data_sub.shape[1]),data_sub.columns,fontsize=14,rota-
tion=45)

plt.yticks(range(data_sub.shape[1]),data_sub.columns,fontsize=14)

cb=plt.colorbar()

cb.ax.tick_params(labelsize=14)

plt.title('CorrelationMatrix',fontsize=16)
```

We can get the following output from this code:

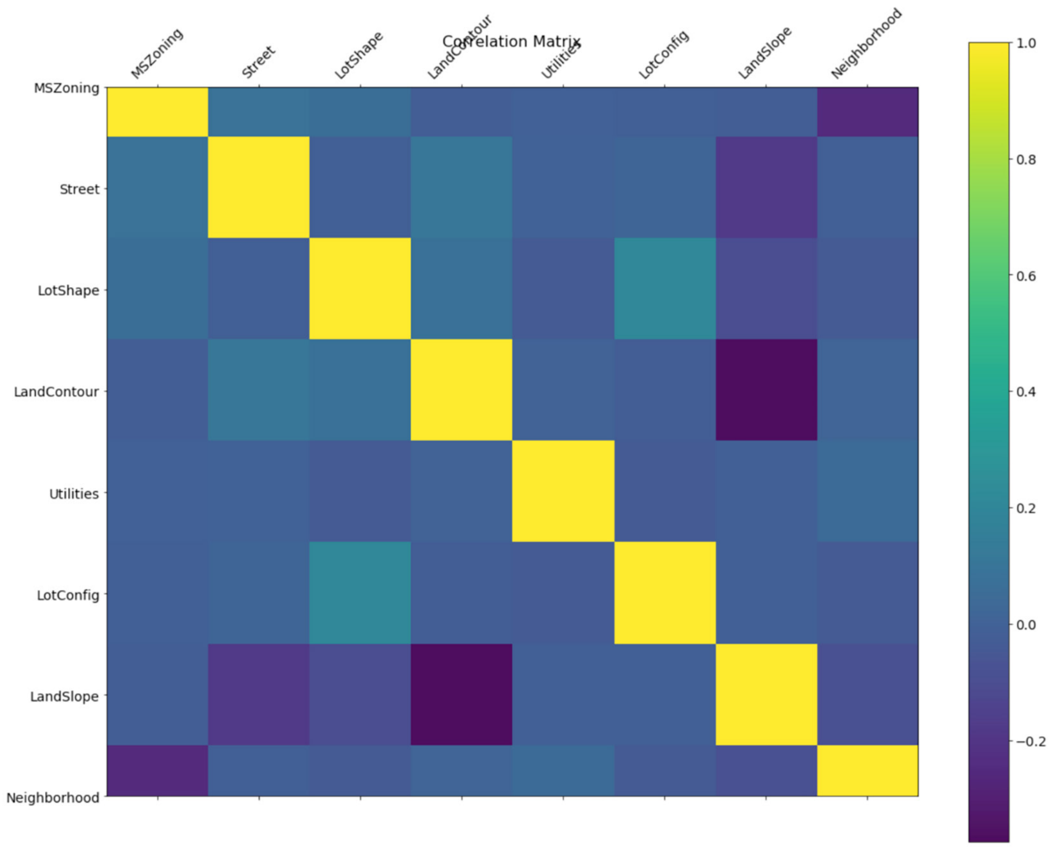


Figure 8.28

We can deduce that *LotShape* and *LotConfig* have a high correlation. Similarly, *LandContour* and *Street* also have a high correlation.

The second method to check for multicollinearity is a mathematical function called **VarianceInflationFactor** (**VIF**). It determines the effect of multicollinearity among the independent variables on the variance. It can be represented by this formula:

$$VIF = \frac{1}{1 - R^2 i}$$

In this formula, we take each predictor and make a linear regression model with all the remaining independent variables. If VIF's value is greater than 5 or 6, we can conclude that multicollinearity is present.

Here's the Python code for finding VIF. The function to find VIF is stored in the `statsmodels` package, inside the module stats and `outliers_influence`, as `variance_inflation_factor`:

```
fromstatsmodels.stats.outliers_influenceimportvariance_inflation_factor

fromstatsmodels.tools.toolsimportadd_constant
```

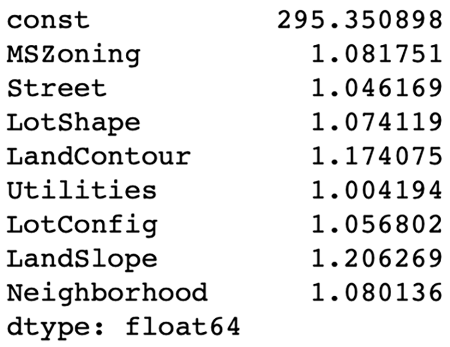
We will add a column of ones that will help us find the regression line between the independent variables:

```
X=add_constant(data_sub)
```

Now, we can convert it into pandas series and show the results:

```
pd.Series([variance_inflation_factor(X.values,i)

foriinrange(X.shape[1])],

index=X.columns)
```

I have taken a subset of the entire data frame. We can see the following output on the notebook:



```
const           295.350898
MSZoning          1.081751
Street            1.046169
LotShape          1.074119
LandContour       1.174075
Utilities         1.004194
LotConfig         1.056802
LandSlope         1.206269
Neighborhood      1.080136
dtype: float64
```

Figure 8.29

We can see that the VIF is well under control.

Conclusion

This chapter covered the different types of regressions. We started with simple linear regression having a single feature and moved to multiple linear regression having multiple features To tackle non-linearity in data, we extended linear regression by adding **Polynomial Features**. We also discussed the importance of variables by simple approaches like subset selection, regularization approaches like Ridge and LASSO regression, and hybrid approach like ElasticNet regression. We wrapped up the chapter with the most important part of regression—residuals.

In the next chapter, we will look at the different methods to analyze data and its applications in Python using Pandas and Numpy.

CHAPTER 9
Data Analysis Using Python

Till now, we have seen different types of statistical measures that can be applied to data. However, the data that reaches us lacks proper format most of the time. If it has the proper format, it needs to undergo certain pre-processing steps before a statistical measure is applied,. Also, we need visualizations to understand the data perfectly. For all these things, we must understand a few core packages in Python that have made the life of data scientists very easy. In this chapter, we will cover three such packages: Pandas, Numpy, and Matplotlib.

Structure

- Pandas
- Importing and reading a CSV sheet
- Basic exploration of data
- Converting a Python data structure to data frame
- Numerical description of a data frame
- Adding conditions in Pandas
- Extending extractions – `loc` and `iloc`
- Tackling null values

- Concatenating data frames
- Merging data frames
- Reading and writing Excel sheets
- Exploring groupby
- Binning in Pandas
- Pandas series
- Numpy
    - o Creating null vector
    - o Indexing
    - o Reshaping a Numpy Array
    - o Generating random values using Numpy
    - o Descriptive statistics using Numpy
    - o Mathematical operations using Numpy
    - o Other important features in Numpy

Objectives

Pandas is used for Excel-like analysis inside Python. It is very interactive and makes us understand the data pretty effectively. Numpy is used for all kinds of mathematical calculations. It is extremely fast, which is why it is also the core of Pandas. Finally, we will be discussing **Matplotlib**, which will help us make different kinds of visualizations in Python.

All these things are important because either it could be Machine Learning or Deep Learning, all three packages have a very important role, and one cannot avoid going further without knowing them. Since this book covers the foundations of machine learning, we cannot proceed further without understanding them.

Pandas

As mentioned earlier, Pandas make data analysis similar to that of Excel. However, instead of writing complicated VBA code, you do it in Python, and you can do much more than in Excel. We will be working on the Jupyter Notebook (Anaconda Distribution) throughout this chapter. Generally, Pandas comes with the distribution, but we can install it with the following commands if it's not present:

```
pip install pandas
```

```
conda install pandas
```

After installation, we can perform several operations using it. Let's explore pandas in the next few sections and understand how it can be used.

Importing and Reading a CSV Sheet

The first step will be to import the pandas package inside the Python environment. After importing it, we can use the `read_csv` method of pandas to read a CSV file. We will use the `titanic` and `house price` dataset throughout this chapter. Let's read the titanic dataset first:

```
import pandas as pd
```

```
data = pd.read_csv("titanic_train.csv")
```

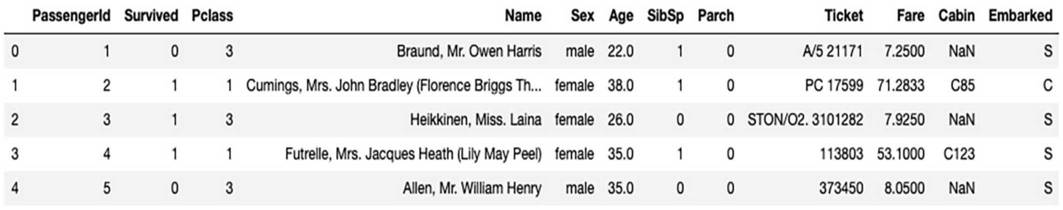
This will store all the rows and columns in the CSV sheet inside the variable data.

Basic Exploration of Data

Now that we have read the CSV file, let's look at what is stored inside the variable. For this, we can use two functions provided by pandas: `head()` and `tail()`. Where `head()` gives us the top five rows present in the data, `tail()` gives us the last five rows.

```
data.head()
```

The following is the screenshot:

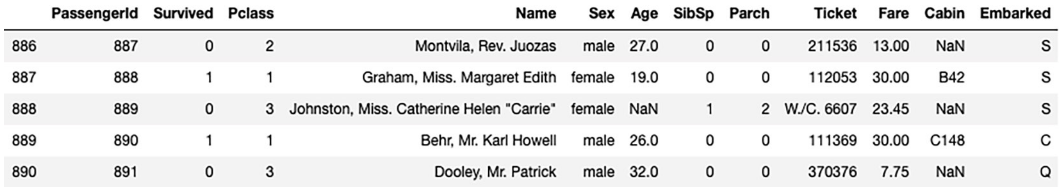


	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 9.1: *The output of the head() function*

```
data.tail()
```

Here's the screenshot:



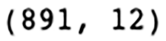
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

Figure 9.2: *The output of the tail() function*

We can get a higher number of rows by putting custom values as parameters. For example, we can use `head(10)` to get the first 10 rows or `tail(10)` to get the last 10 rows. To know the total number of rows and columns, we can use the shape method:

```
data.shape
```

The following is the output:



```
(891, 12)
```

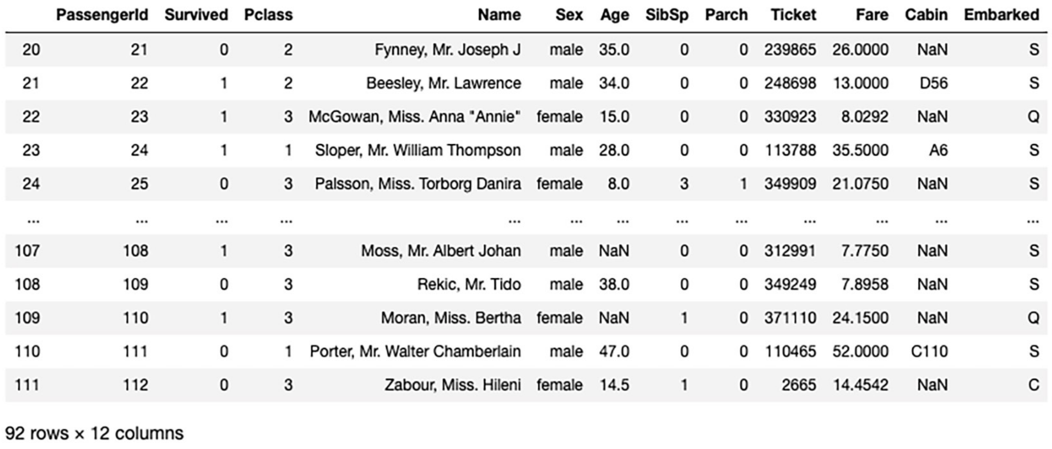
Figure 9.3: *The output of data.shape*

You can see that the output says that there are 891 rows and 12 columns in total.

We have seen how to get the first or bottom few rows, but what about if we want to get rows from between? Just like Python Lists or Tuples, Pandas supports the slicing concept, which can be used to get rows from between. Suppose we want data from row number 20 to row number 111, we will use:

```
data[20:112]
```

This will give us the required output. Remember, whenever we give slices, the lower limit is included, but the upper limit is not. So, we will get data from 20 to 111, as seen in the following screenshot:



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
20	21	0	2	Fynney, Mr. Joseph J	male	35.0	0	0	239865	26.0000	NaN	S
21	22	1	2	Beesley, Mr. Lawrence	male	34.0	0	0	248698	13.0000	D56	S
22	23	1	3	McGowan, Miss. Anna "Annie"	female	15.0	0	0	330923	8.0292	NaN	Q
23	24	1	1	Sloper, Mr. William Thompson	male	28.0	0	0	113788	35.5000	A6	S
24	25	0	3	Palsson, Miss. Torborg Danira	female	8.0	3	1	349909	21.0750	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
107	108	1	3	Moss, Mr. Albert Johan	male	NaN	0	0	312991	7.7750	NaN	S
108	109	0	3	Rekic, Mr. Tido	male	38.0	0	0	349249	7.8958	NaN	S
109	110	1	3	Moran, Miss. Bertha	female	NaN	1	0	371110	24.1500	NaN	Q
110	111	0	1	Porter, Mr. Walter Chamberlain	male	47.0	0	0	110465	52.0000	C110	S
111	112	0	3	Zabour, Miss. Hileni	female	14.5	1	0	2665	14.4542	NaN	C

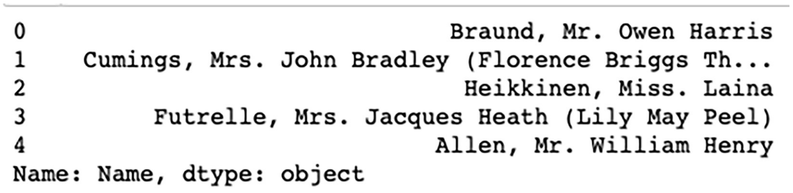
92 rows × 12 columns

Figure 9.4: *The output of data[20:112]*

Now, if we can get a subset of rows, can we get a subset of columns? Yes, of course! Suppose we want the first five index names, the last five index names, and from 3 to 10 index names. For this, we can write:

```
data['Name'].head()
```

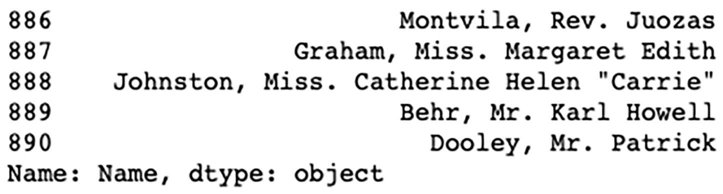
The following is the output:



```
0                             Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2                              Heikkinen, Miss. Laina
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                            Allen, Mr. William Henry
Name: Name, dtype: object
```

Figure 9.5: The output of head()

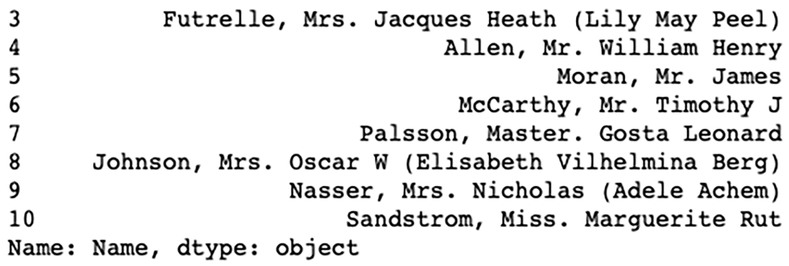
```
data['Name'].tail()
```



```
886                       Montvila, Rev. Juozas
887                Graham, Miss. Margaret Edith
888    Johnston, Miss. Catherine Helen "Carrie"
889                       Behr, Mr. Karl Howell
890                         Dooley, Mr. Patrick
Name: Name, dtype: object
```

Figure 9.6: The output of tail()

```
data['Name'][3:11]
```

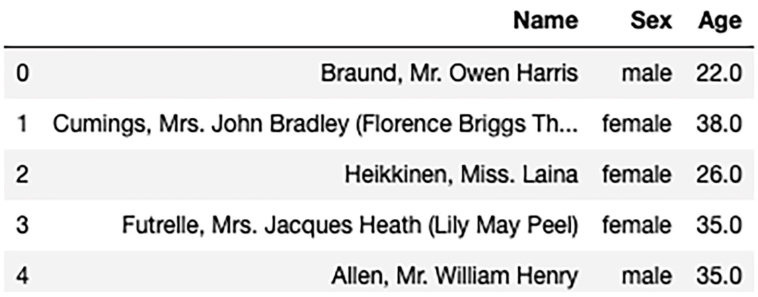


```
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                              Allen, Mr. William Henry
5                                      Moran, Mr. James
6                               McCarthy, Mr. Timothy J
7                        Palsson, Master. Gosta Leonard
8     Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)
9                   Nasser, Mrs. Nicholas (Adele Achem)
10                      Sandstrom, Miss. Marguerite Rut
Name: Name, dtype: object
```

Figure 9.7: The output of slicing

What if we want more than one column? In that case, we can write it as follows:

```
data[['Name', 'Sex', 'Age']].head()
```



	Name	Sex	Age
0	Braund, Mr. Owen Harris	male	22.0
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	Heikkinen, Miss. Laina	female	26.0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
4	Allen, Mr. William Henry	male	35.0

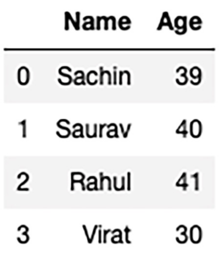
Figure 9.8: *The output of the combination of head() and column selection*

Converting a Python Data Structure to Data Frame

Remember, in Pandas, the preceding table is called a data frame. What if we don't have an Excel/CSV sheet and have a normal array like structures: list, tuples, dictionaries, and such. Here's how we can convert them into a data frame:

```
name = [‘Sachin’, ‘Saurav’, ‘Rahul’, ‘Virat’]

age = [39, 40, 41, 30]


zipped = list(zip(name,age))


crkt_data = pd.Dataframe(zipped, columns = [‘Name’, ‘Age’])

crkt_data
```

If we break down the preceding code, we define two lists—name and age—which tell us about some cricketers' names and ages. Before converting it into a data frame, we must zip them together to make it easy for pandas. Finally, we use the `Data frame` method, which converts the list to a data frame. We provide the column names for each list as `Name` and `Age`. The output is given as follows:



	Name	Age
0	Sachin	39
1	Saurav	40
2	Rahul	41
3	Virat	30

Figure 9.9: *The output of crkt_data*

Similarly, we can covert a dictionary into a data frame. The output for the following code will be the same as mentioned earlier:

```
dt = {‘Name’ : [‘Sachin’, ‘Saurav’, ‘Rahul’, ‘Virat’], ‘Age’ : [39, 40, 41, 30]}
```

```
crkt_data = pd.Dataframe(dt)
```

```
crkt_data
```

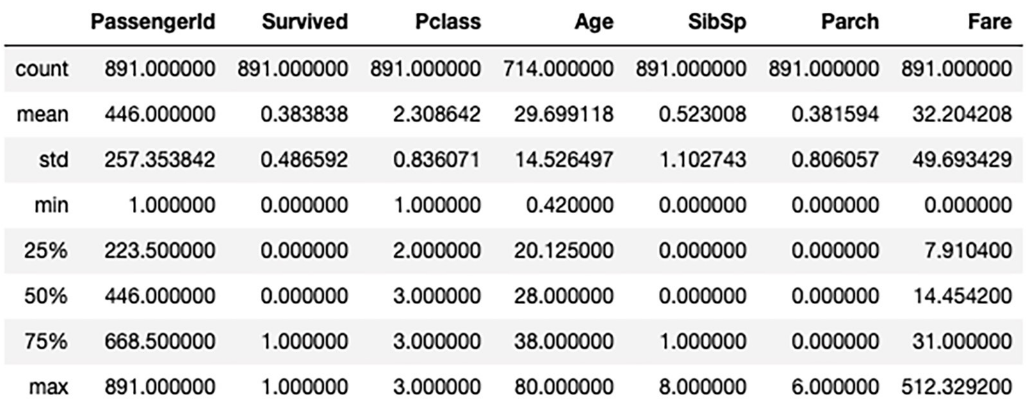
Numerical Description of a Data Frame

We all know that any data frame can be divided into two parts:

- Columns having categories or text – Categorical columns
- Columns having numbers – Numerical columns

We can use the `describe()` method of pandas to describe all the numerical columns present inside a data frame. If we apply this function on the titanic dataset, we'll get the following:

```
data.describe()
```

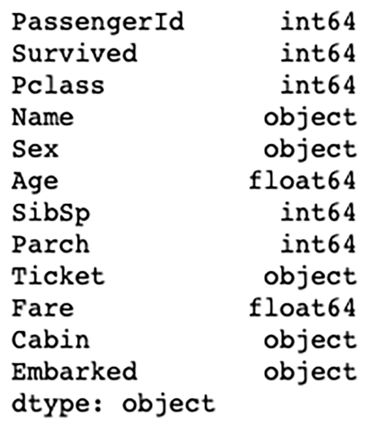


	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Figure 9.10: *The output of describe()*

As an output for all the numerical columns, we get the total count (excluding the null values), mean of the column, standard deviation, minimum and maximum, first and third quartile, and the median. For knowing the data type of all the columns, we can use the `dtypes` method:

```
data.dtypes
```



```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Figure 9.11: *The output of dtypes()*

If we want, we can extract specific descriptive statistics for specific columns. For example, we can get a total count for the `survived` column and mean for the `fare` column:

```
data['Survived'].count()
```

```
data['Fare'].mean()
```

It provides 891 and 32.204 as the output. Similarly, other functions include `min()`, `max()`, and `std()`. They can be applied to specific columns or the entire data.

Adding Conditions in Pandas

Suppose we want answers for the following questions:

- How many females are present in the dataset?
- How many passengers had fares over $30?
- What is the average age of the passengers who died?
- What were the top ten names of the passengers who were female, above the age of 30, and died?

To answers these questions, we will look at conditional pandas. Let's answer them one by one.

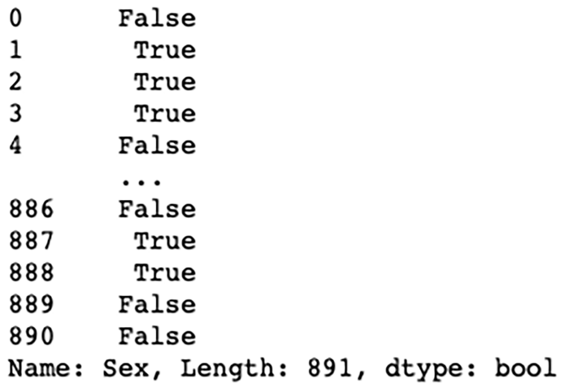
How many females are present in the dataset?

The following code gives us the answer:

```
data[data.Sex == 'female']['Sex'].count()
```

To understand the preceding code, let's break it down into three parts. The condition put as a slice, the column name provided after slice, and the count function. The first part—condition—checks for all the positions in a data frame where the `Sex` column value is `female`. If the condition matches, it gives `true`, else it gives `false`. If we only execute that line, we can understand what is explained here:

```
data.Sex == 'female'
```

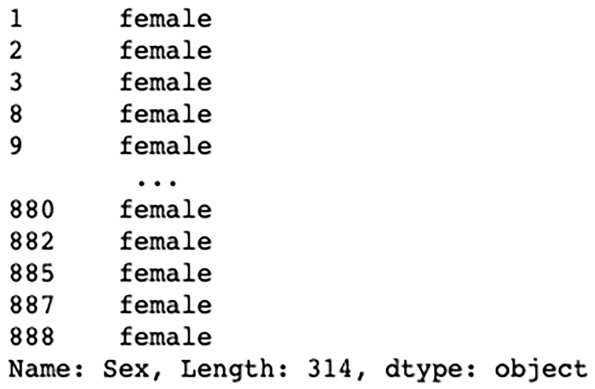


```
0      False
1       True
2       True
3       True
4      False
       ...
886    False
887     True
888     True
889    False
890    False
Name: Sex, Length: 891, dtype: bool
```

Figure 9.12: *The output of applying conditions*

Only the rows that come out as `True` are used in the analysis, and the ones with `False` aren't. Coming to the second part, we only want one column out of all those that come out as true—`Sex`. This gives the following output:

```
data[data.Sex == 'female']['Sex']
```



```
1      female
2      female
3      female
8      female
9      female
        ...
880    female
882    female
885    female
887    female
888    female
Name: Sex, Length: 314, dtype: object
```

Figure 9.13: *The output of applying condition and selecting a column*

Finally, we need to count all the females, which gives us an output of `314`.

How many passengers had fares over $30?

```
data[data.Fare> 30][‘PassengerId’].count()
```

As we saw in the previous question, the same logic will be followed here. First, we'll look at all the rows where the fare is greater than 30. We will extract the `Passenger ID` of wherever it is `True`, and then we'll count the number. This will give us an output of `234`.

What is the average age of the passengers who died?

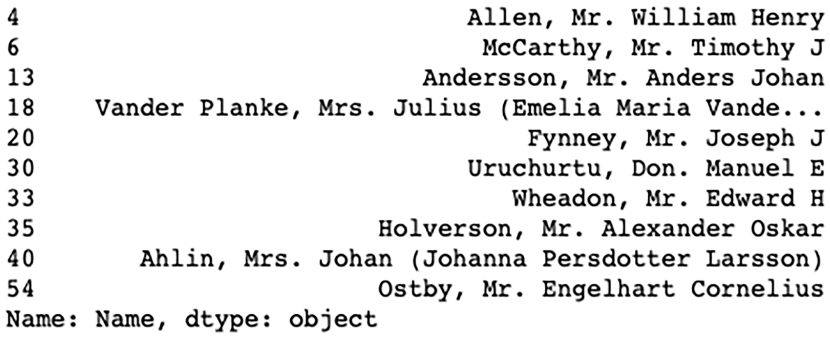
```
data[data.Survived == 0][‘Age’].mean()
```

This will also follow the same logic, but we will use the `mean` function instead of the `count` function, as we want to determine the average age. This gives us an output of `30.63`.

What were the top ten names of the passengers who were female, above the age of 30, and died?

```
data[(data.Age> 30) & (data.Survived == 0)][‘Name’].head(10)
```

Here, we have used multiple conditions. The first condition returns all the people over the age of 30, while the second only returns the people who didn't survive. After the check is performed, we only want the `Name` column to be provided as output. Finally, we want only the first 10 names, so we will use the custom head function. We will get the following output:



```
4                                Allen, Mr. William Henry
6                                 McCarthy, Mr. Timothy J
13                             Andersson, Mr. Anders Johan
18    Vander Planke, Mrs. Julius (Emelia Maria Vande...
20                                  Fynney, Mr. Joseph J
30                             Uruchurtu, Don. Manuel E
33                                 Wheadon, Mr. Edward H
35                          Holverson, Mr. Alexander Oskar
40       Ahlin, Mrs. Johan (Johanna Persdotter Larsson)
54                          Ostby, Mr. Engelhart Cornelius
Name: Name, dtype: object
```

Figure 9.14: *The output of multiple conditions*

Extending Extractions – loc and iloc

Pandas provides `loc()` and `iloc()`, are the two functionalities that, which can be used for any kind of subset selection from a data frame. While loc is a row-based subset selection method, iloc is an index-based subset selection method. Let's start by exploring `iloc()`.

We can get the specific rows by providing a single number as index – `data.iloc[0]` or `data.iloc[1]`, which gives us first and second rows, respectively. Similarly, we can get first and second column by writing `data.iloc[:,0]` or `data.iloc[:,0]`.

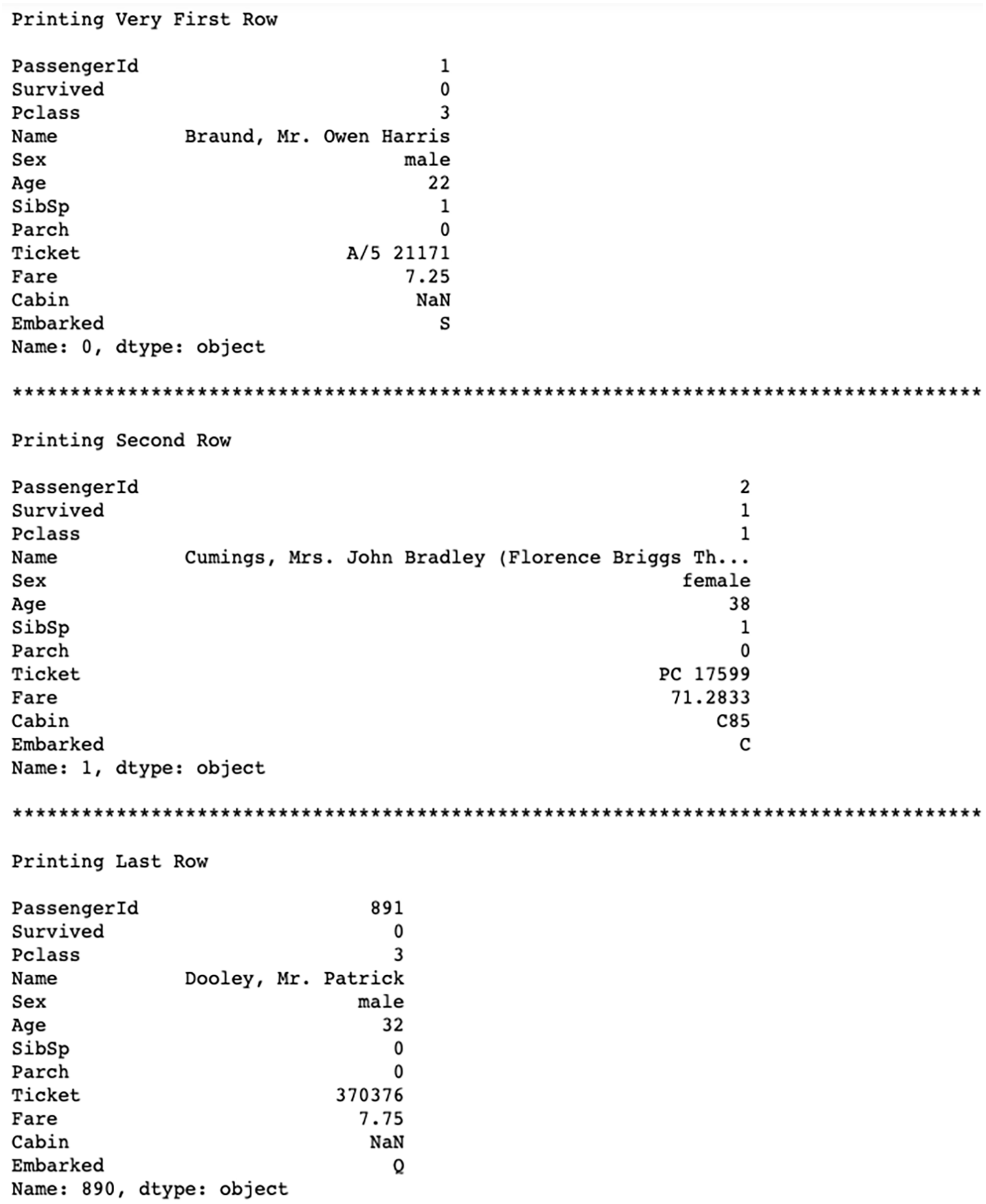
So, in the index brackets, the section before the comma represents rows, while the section after the comma represents columns. Let's look at the complete code to understand this.

Understanding the iloc() Function

Take a look at the following code:

```
print("Printing Very First Row\n")

print(data.iloc[0])

print("\n***************************************************************
*********************\n")

print("Printing Second Row\n")

print(data.iloc[1])

print("\n***************************************************************
*********************\n")

print("Printing Last Row\n")

print(data.iloc[-1])

print("\n***************************************************************
*********************\n")

print("Printing First Column\n")

print(data.iloc[:,0])

print("\n***************************************************************
*********************\n")

print("Printing Second Column\n")

print(data.iloc[:,1])

print("\n***************************************************************
*********************\n")

print("Printing Last Columns\n")
print(data.iloc[:,-1])
```

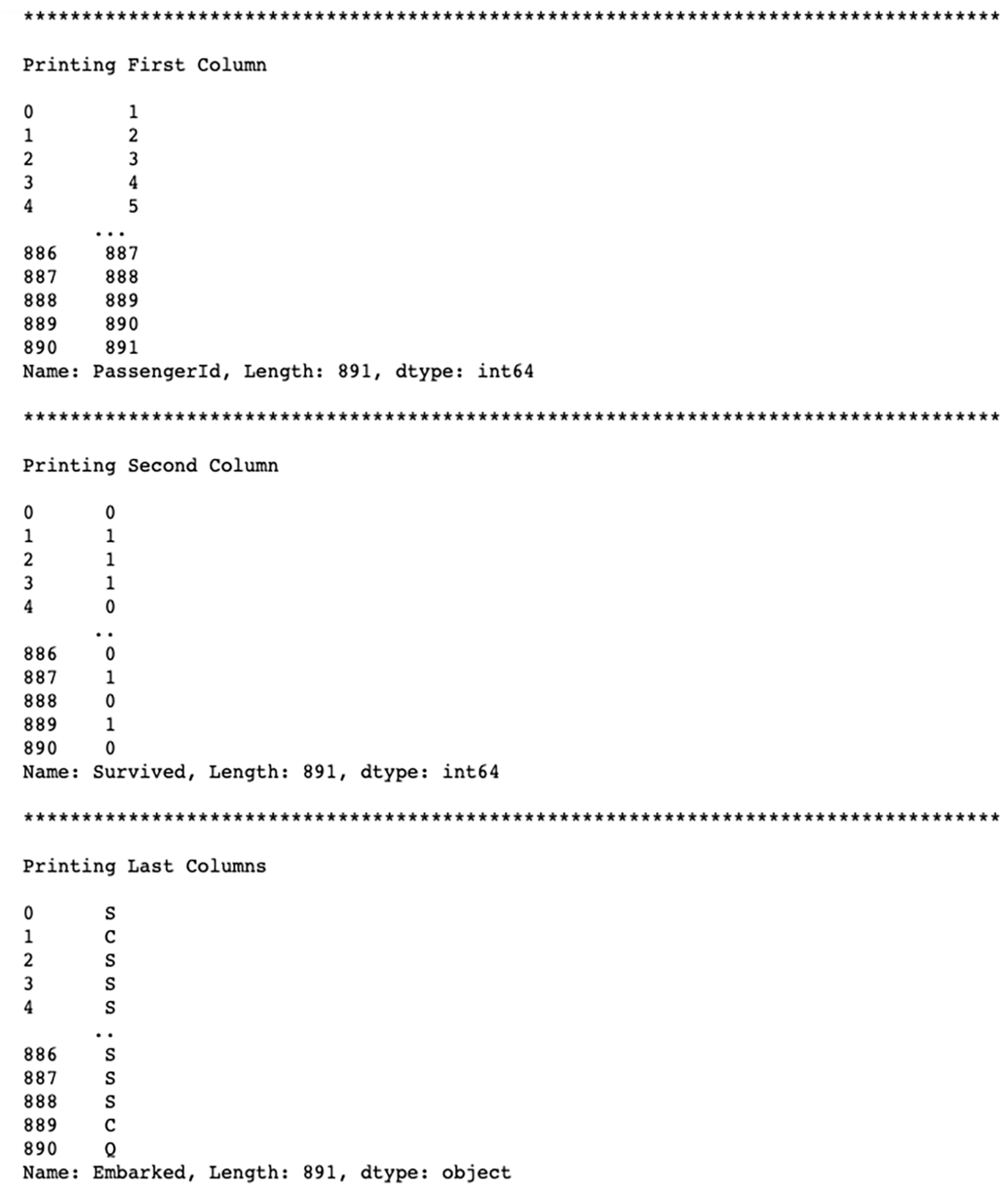
This will give us the following output:



```
Printing Very First Row

PassengerId                        1
Survived                           0
Pclass                             3
Name           Braund, Mr. Owen Harris
Sex                             male
Age                               22
SibSp                              1
Parch                              0
Ticket                     A/5 21171
Fare                            7.25
Cabin                            NaN
Embarked                           S
Name: 0, dtype: object

************************************************************************************

Printing Second Row

PassengerId                                                  2
Survived                                                     1
Pclass                                                       1
Name           Cumings, Mrs. John Bradley (Florence Briggs Th...
Sex                                                     female
Age                                                         38
SibSp                                                        1
Parch                                                        0
Ticket                                                PC 17599
Fare                                                   71.2833
Cabin                                                      C85
Embarked                                                     C
Name: 1, dtype: object

************************************************************************************

Printing Last Row

PassengerId                    891
Survived                         0
Pclass                           3
Name           Dooley, Mr. Patrick
Sex                           male
Age                             32
SibSp                            0
Parch                            0
Ticket                      370376
Fare                          7.75
Cabin                          NaN
Embarked                         Q
Name: 890, dtype: object
```

Figure 9.15: *The output of using iloc()*

And the following is the succeeding output:



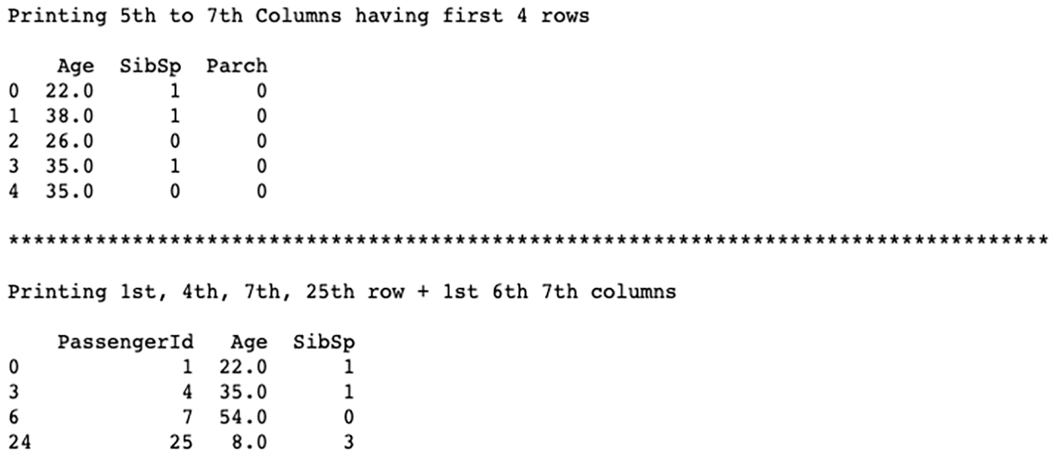
```
*********************************************************************************

Printing First Column

0        1
1        2
2        3
3        4
4        5
      ...
886    887
887    888
888    889
889    890
890    891
Name: PassengerId, Length: 891, dtype: int64

*********************************************************************************

Printing Second Column

0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64

*********************************************************************************

Printing Last Columns

0      S
1      C
2      S
3      S
4      S
      ..
886    S
887    S
888    S
889    C
890    Q
Name: Embarked, Length: 891, dtype: object
```

Figure 9.16: *The output of using iloc()*

We can also get a combination of a subset of rows as well as columns, all at once:

```
print(“Printing 5th to 7th Columns having first 4 rows\n”)

print(data.iloc[0:5, 5:8])

print(“\n***************************************************************
*********************\n”)

print(“Printing 1st, 4th, 7th, 25th row + 1st 6th 7th columns\n”)

print(data.iloc[[0,3,6,24], [0,5,6]])
```

The output is as follows:



```
Printing 5th to 7th Columns having first 4 rows

    Age  SibSp  Parch
0  22.0      1      0
1  38.0      1      0
2  26.0      0      0
3  35.0      1      0
4  35.0      0      0

************************************************************************************

Printing 1st, 4th, 7th, 25th row + 1st 6th 7th columns

    PassengerId   Age  SibSp
0             1  22.0      1
3             4  35.0      1
6             7  54.0      0
24           25   8.0      3
```

Figure 9.17: *The output of multiple operations using iloc()*

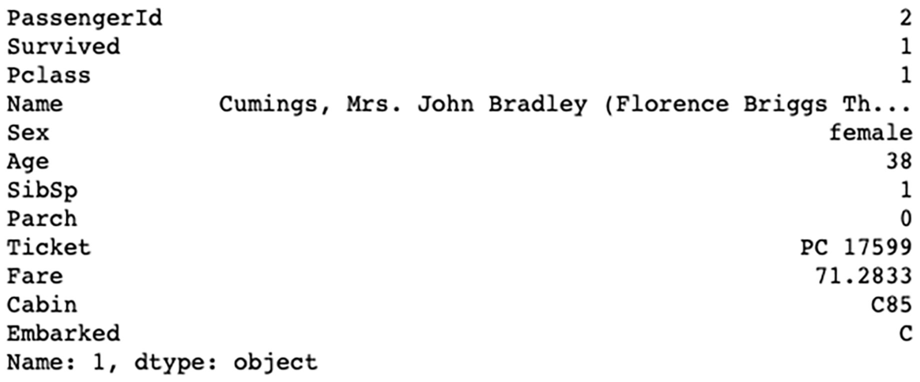
Now, let us look at the loc function.

Understanding the loc() Function

The `loc()` function can be used to directly access a particular index in the data frame:

```
data.loc[1]
```

This will give us the output of the second row of the data frame, shown as follows:



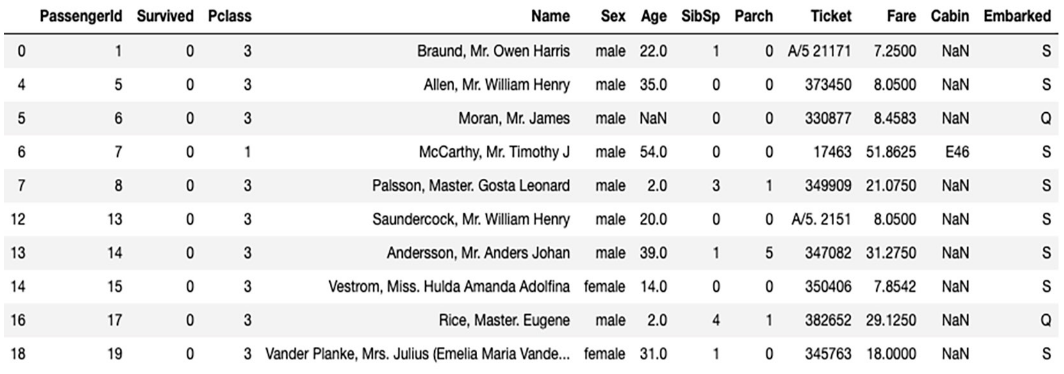
```
PassengerId                                                     2
Survived                                                        1
Pclass                                                          1
Name           Cumings, Mrs. John Bradley (Florence Briggs Th...
Sex                                                        female
Age                                                            38
SibSp                                                           1
Parch                                                           0
Ticket                                                   PC 17599
Fare                                                      71.2833
Cabin                                                         C85
Embarked                                                        C
Name: 1, dtype: object
```

Figure 9.18: *The output of loc()*

We can also return a subset of a data frame based on a condition using `loc()`:

```
data.loc[data.Survived == 0].head(10)
```

The preceding code selects all the rows where the survived column has the value of 0 and returns the first 10 rows, displayed as follows:



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
12	13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	NaN	S
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	NaN	S
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0	0	350406	7.8542	NaN	S
16	17	0	3	Rice, Master. Eugene	male	2.0	4	1	382652	29.1250	NaN	Q
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female	31.0	1	0	345763	18.0000	NaN	S

Figure 9.19: *The output of combining conditions with loc()*

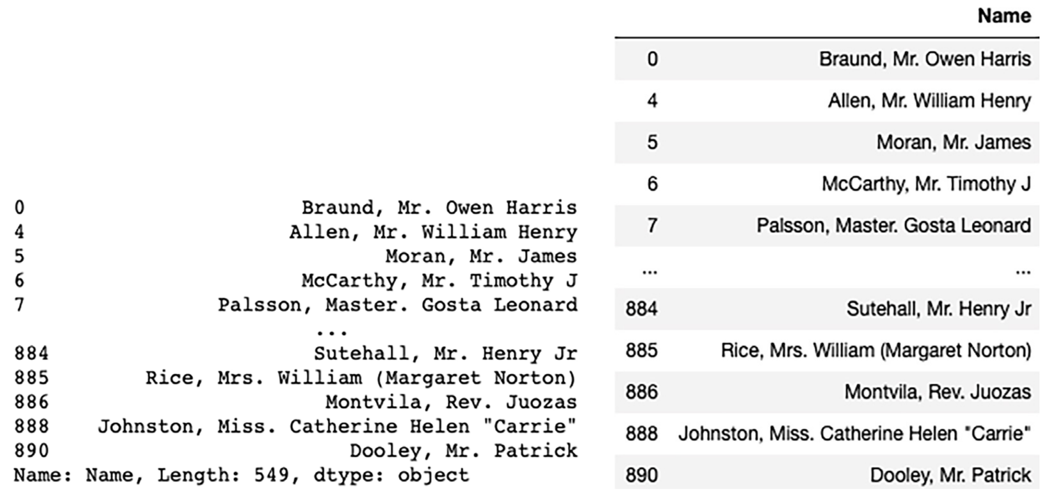
Based on the output of the preceding code, we can return a separate column as well. If we pass the column name as a string, the output is returned a string. However, if it is passed as a list, it comes as a data frame:

```
data.loc[data.Survived == 0], 'Name']
```

vs

```
data.loc[data.Survived == 0], ['Name']]
```

The output of these two sets of code is as follows:



```
0                        Braund, Mr. Owen Harris
4                       Allen, Mr. William Henry
5                               Moran, Mr. James
6                        McCarthy, Mr. Timothy J
7                 Palsson, Master. Gosta Leonard
                          ...
884                        Sutehall, Mr. Henry Jr
885          Rice, Mrs. William (Margaret Norton)
886                          Montvila, Rev. Juozas
888       Johnston, Miss. Catherine Helen "Carrie"
890                            Dooley, Mr. Patrick
Name: Name, Length: 549, dtype: object
```

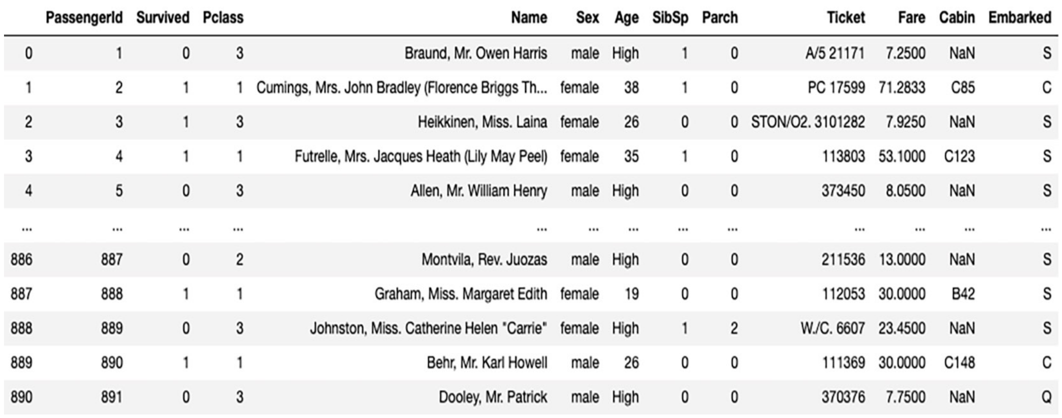
	Name
0	Braund, Mr. Owen Harris
4	Allen, Mr. William Henry
5	Moran, Mr. James
6	McCarthy, Mr. Timothy J
7	Palsson, Master. Gosta Leonard
...	...
884	Sutehall, Mr. Henry Jr
885	Rice, Mrs. William (Margaret Norton)
886	Montvila, Rev. Juozas
888	Johnston, Miss. Catherine Helen "Carrie"
890	Dooley, Mr. Patrick

Figure 9.20: *The output of column selection*

We can use the preceding functionalities of Pandas to make changes to the values of the data frames. For example, we change the value of `Age` to `High` wherever the value of `Survived` is `0`. Then, we can use the `loc()` function as follows:

```
data.loc[data.Survived == 0, [‘Age’]] = ‘High’
```

This gives us the following output:



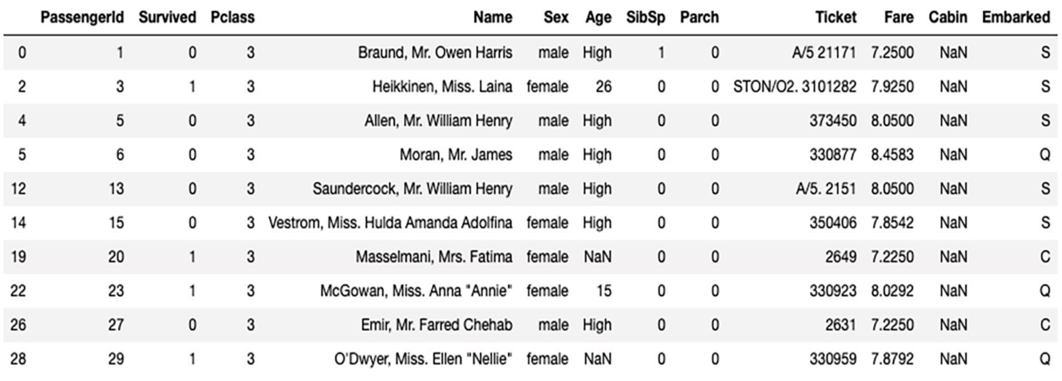
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	High	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	High	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	High	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	High	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	High	0	0	370376	7.7500	NaN	Q

Figure 9.21: *Assigning Values*

One last thing to note here is that whenever `iloc()` or `loc()` gives some output, they copy the index as it is and do not restart them. You can see this in the following code:

```
data.loc[data.Fare< 10].head(10)
```

If you look at the output, you'll find that the sequence of indices is not right, as given in the following screenshot:



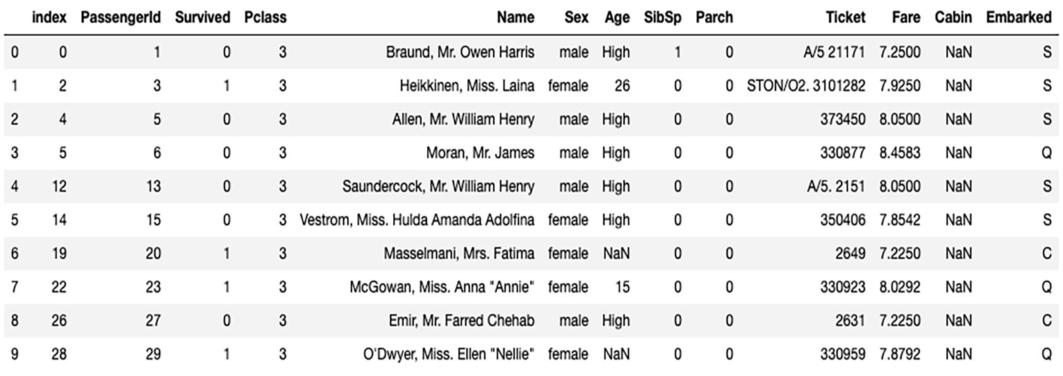
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	High	1	0	A/5 21171	7.2500	NaN	S
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	High	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	High	0	0	330877	8.4583	NaN	Q
12	13	0	3	Saundercock, Mr. William Henry	male	High	0	0	A/5. 2151	8.0500	NaN	S
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	High	0	0	350406	7.8542	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C
22	23	1	3	McGowan, Miss. Anna "Annie"	female	15	0	0	330923	8.0292	NaN	Q
26	27	0	3	Emir, Mr. Farred Chehab	male	High	0	0	2631	7.2250	NaN	C
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.8792	NaN	Q

Figure 9.22: *Checking index*

To solve this problem and restart the index, we can use the `reset_index()` method of pandas:

```
data.loc[data.Fare< 10].reset_index().head(10)
```

It provides the following output. Later, we can drop the unnecessary column created.



	index	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	0	1	0	3	Braund, Mr. Owen Harris	male	High	1	0	A/5 21171	7.2500	NaN	S
1	2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.9250	NaN	S
2	4	5	0	3	Allen, Mr. William Henry	male	High	0	0	373450	8.0500	NaN	S
3	5	6	0	3	Moran, Mr. James	male	High	0	0	330877	8.4583	NaN	Q
4	12	13	0	3	Saundercock, Mr. William Henry	male	High	0	0	A/5. 2151	8.0500	NaN	S
5	14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	High	0	0	350406	7.8542	NaN	S
6	19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C
7	22	23	1	3	McGowan, Miss. Anna "Annie"	female	15	0	0	330923	8.0292	NaN	Q
8	26	27	0	3	Emir, Mr. Farred Chehab	male	High	0	0	2631	7.2250	NaN	C
9	28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	NaN	0	0	330959	7.8792	NaN	Q

Figure 9.23: *Resetting index*

Tackling Null Values

Null values are one of the most important parts of data processing. If we are unable to tackle these values, we won't be able to build a model. Almost all the models don't expect null values in the data, so we must process all the null values before giving the data to the model.

There are several approaches to deal with null values. In this section, we will look at the following ways:

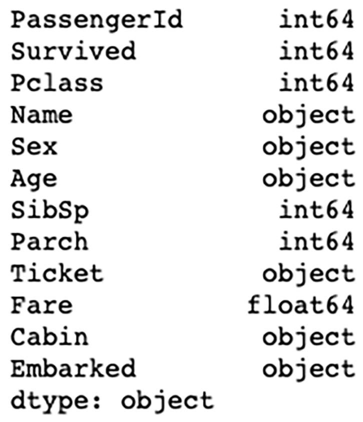
1. Mean/median/mode
2. Backward fill
3. Forward fill
4. Interpolate
5. Custom

Let's understand all these approaches and look at their practical implementation in Python.

The first thing we must know is that a categorical feature's null value is tackled differently, and a numerical one is tackled differently. We cannot replace categorical null values with mean, and replacing numerical with the mode is not feasible either. That said, some approaches, like backward / forward fill, can be adopted in both. So, we must first create two data frames: one containing only categorical, and another containing only numerical. The following code tells us about the different `datatypes` present in the data frame:

```
data.dtypes
```

This will give us the following output:



```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age             object
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Figure 9.24: *The output of dtypes()*

We can see that the categorical columns have object datatype, while the others are int and float. Let's split them now:

```
importnumpy as np

data_cat = data.select_dtypes(object)
```

```
data_num = data.select_dtypes(np.number)
```

The `select_dtypes` method selects a particular data type column from the data frame. For categorical ones, we have passed object datatype as the parameter. When it comes to numerical, we can pass `int64` and `float64` one by one and concatenate the data frame, but that will be an unnecessary step. So, we will take help from the `numpy` package, where we can select all the numerical datatypes by writing `np.number`. Now, all our categorical columns are present in `data_cat` and numerical in `data_num`. We can take a look at the columns:

```
data_cat.columns
```

The following is the output:

```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

```
data_num.columns
```

Here's the output:

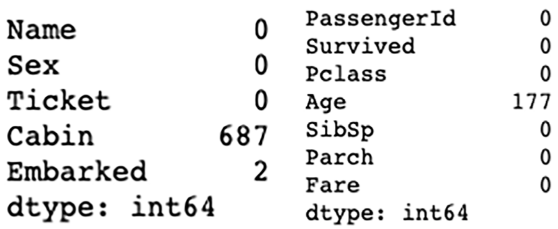
```
Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare'], dtype='object')
```

The next step is to look at the number of null values present in each column. For this, we will use two functions:

```
data_cat.isna().sum()
```

```
data_num.isna().sum()
```

We get the following output:



```
Name          0
Sex           0
Ticket        0
Cabin       687
Embarked      2
dtype: int64
```

```
PassengerId      0
Survived         0
Pclass           0
Age            177
SibSp            0
Parch            0
Fare             0
dtype: int64
```

Figure 9.25: *Checking null values()*

As we can see, we need to tackle the **Cabin** and **Embarked** columns in the categorical columns, and we need to tackle the **Age** column in numerical. Let's start with **Embarked** and fill all the null values with the category having the maximum count (Mode):

```
data_cat['Embarked'].fillna(data_cat['Embarked'].value_counts().idxmax(),
inplace=True)
```

In the preceding code, we first selected the column we want to remove null values from and then used the `fillna()` pandas helper function, which is used to treat null values. As we want the mode of the column, we will use two other functions: `value_counts()` to tell us how often each category occurs, and `idxmax()` to return the category with the maximum counts. Another parameter—inplace —tells pandas to make changes directly in the original data frame. If we keep it as false, it will provide an output but won't make any changes in the original frame.

Coming to the `Cabin` column, we have seen that a total of 891 rows are present in the data frame. Out of those, 687 are null in this column, which means it has over 80% of null values. As it will not help us by filling the null values, it is better to drop this column. We will use the `drop()` function of pandas to drop this column:

```
data_cat.drop([‘Cabin’], axis=1, inplace=True)
```

In the preceding code, `axis=1` is used to delete a column. If we kept `axis = 0`, it would try to delete a row. This finishes our categorical null values removal. Now, let's switch to numerical null values in the `Age` column. Let's remove the null values by`mean()`:

```
data_num[‘Age’].fillna(data_num[‘Age’].mean(), inplace=True)
```

We can use other methods as well. Backward fill will look at a null value, see what value occurs after that, and fill the null value with it. Similarly, the forward fill method will look at a null value, see what value occurs before that, and replace the null value with it. The following is the code for it. Remember, we can use only one code: the preceding one or the following one:

```
data_cat[‘Embarked’].ffill(inplace=True)
```

```
data_cat[‘Embarked’].bfill(inplace=True)
```

Similarly, in numerical columns, we can use the interpolate method as well. There are several ways in which we can interpolate:

- linear
- time
- index
- values
- nearest
- zero
- slinear
- quadratic

- cubic
- barycentric
- krogh
- polynomial
- spline
- piecewise_polynomial
- from_derivatives
- pchip
- akima

All these methods have their own advantages and their own specific domain of application. Their explanation is out of the scope of this book, but the code remains the same. Let's look at two methods—linear and polynomial—to replace the null values in the `Age` column:

```
data_num['Age'].interpolate('polynomial', order=2, inplace=True)
```

```
data_num['Age'].interpolate('linear', inplace=True)
```

We have seen various ways of tackling null values. One last approach can be to drop the rows containing null values, which can be done using the `dropna()` function and providing `axis=0`:

```
data.dropna(axis=0, inplace=True)
```

Concatenating Data Frames

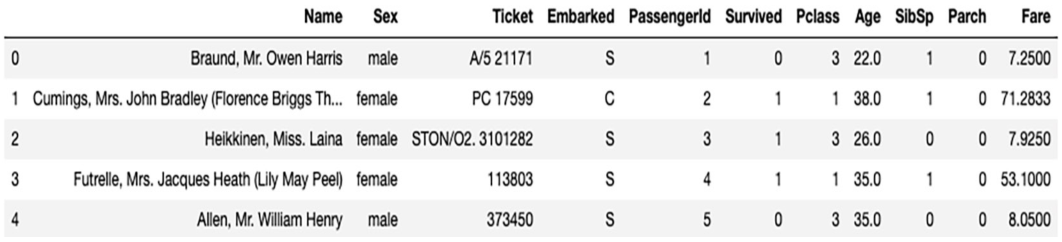
In the previous section, we tackled the null values of categorical and numerical data frames. The next step is to join them to get a single data frame using the `concat()` function of Pandas.

While joining two data frames, we must know whether we are going to join columns or rows. In our case, we will be joining columns, but there may be scenarios where we have to join rows, as the columns will be the same in both the data frames. All this can be done by providing a value of 0 or 1 to the axis:

```
data_final = pd.concat([data_cat, data_num], axis=1)
```

```
data_final.head()
```

This concatenates the two data frames and gives us the output, as shown in the following screenshot:

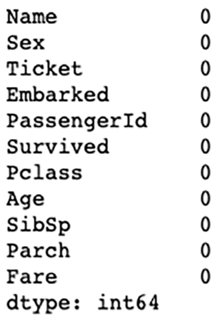


	Name	Sex	Ticket	Embarked	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	Braund, Mr. Owen Harris	male	A/5 21171	S	1	0	3	22.0	1	0	7.2500
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	PC 17599	C	2	1	1	38.0	1	0	71.2833
2	Heikkinen, Miss. Laina	female	STON/O2. 3101282	S	3	1	3	26.0	0	0	7.9250
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	113803	S	4	1	1	35.0	1	0	53.1000
4	Allen, Mr. William Henry	male	373450	S	5	0	3	35.0	0	0	8.0500

Figure 9.26: *Data frame concatenation*

We can recheck the null values as well:

```
data_final.isna().sum()
```



```
Name           0
Sex            0
Ticket         0
Embarked       0
PassengerId    0
Survived       0
Pclass         0
Age            0
SibSp          0
Parch          0
Fare           0
dtype: int64
```

Figure 9.27

As we can see, we have taken care of all the null values now.

Merging Data Frames

If you know about SQL, this section is similar to that. Just like joining two tables in SQL, we can join two data frames in Pandas using the following approaches:

- Left Join
- Right Join
- Outer Join
- Inner Join

Before looking at the code, let's understand what these joins are.

Left Join

In **Left join**, each and every row from the left table will be taken, but only those that are common to a particular column that we pass as a foreign key will be taken from the right table. The following venn diagram explains the process:

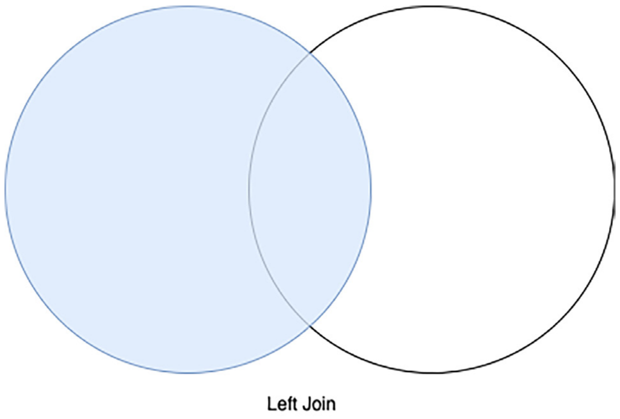


Figure 9.28: *Left join*

Right Join

This is exactly like Left Join, but **right join** takes the right table as the main table instead of the left one. We can see that in the following venn diagram:

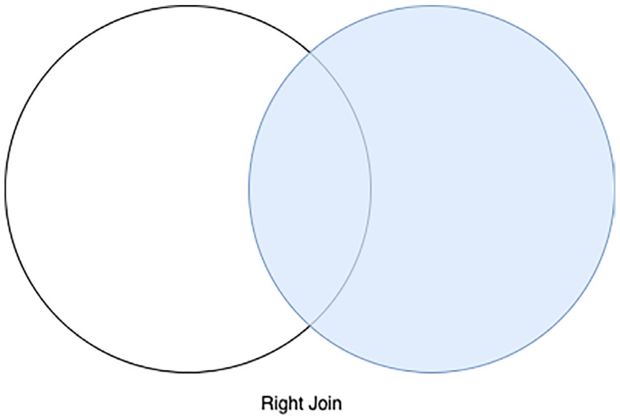


Figure 9.29: *Right join*

Outer Join

An **outer join** returns everything from both the tables, as shown in the following venn diagram:

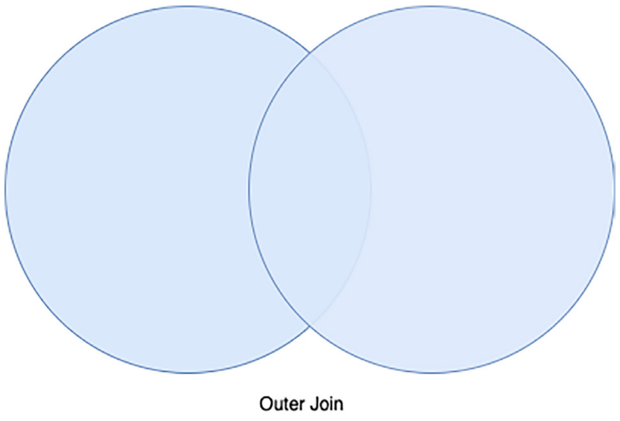


Figure 9.30: *Outer join*

Inner Join

Inner join only returns the common elements from both the tables, as illustrated in the following diagram:

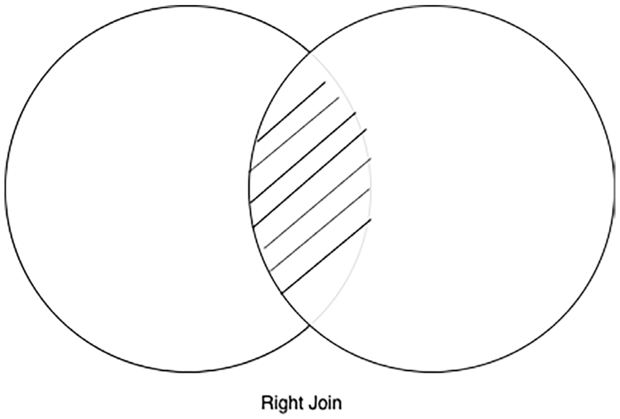


Figure 9.31: *Inner join*

Let's create a few dummy data frames and apply all the above-mentioned joins. Suppose there is a class of 10 students having ID saved in the following list:

```
student_id = [0,1,2,3,4,5,6,7,8,9,10]
```

Let their names be stored in another list:

```
student_name = ['A', 'B', 'C', 'D', 'E', 'F', 'G','H','I', 'J']
```

Suppose there is a club in the school with 20 students, including the above ones. The ID of sports students is stored in the following list:

```
student_id_sports = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
20]
```

The sports that they play are stored on another list:

```
sports = [‘Cricket’, ‘Football’, ‘BasketBall’, ‘Football’,’Cricket’,
‘Football’, ‘BasketBall’, ‘Football’,’Cricket’, ‘Football’, ‘BasketBall’,
‘Football’,’Cricket’, ‘Football’, ‘BasketBall’, ‘Football’,’Cricket’,
‘Football’, ‘BasketBall’, ‘Football’,’Cricket’, ‘Football’, ‘BasketBall’,
‘Football’]
```

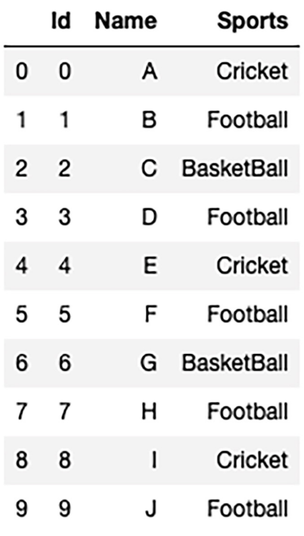
We will create two data frames: one for the class and another for the sports club:

```
student = pd.Data frame(zip(student_id, student_name), columns=[‘Id’,
‘Name’])
```

```
sports_df = pd.Dataframe(zip(student_id_sports, sports), columns=[‘Id’,
‘Sports’])
```

Now that our dummy data frames are created, let’s apply the four joins:

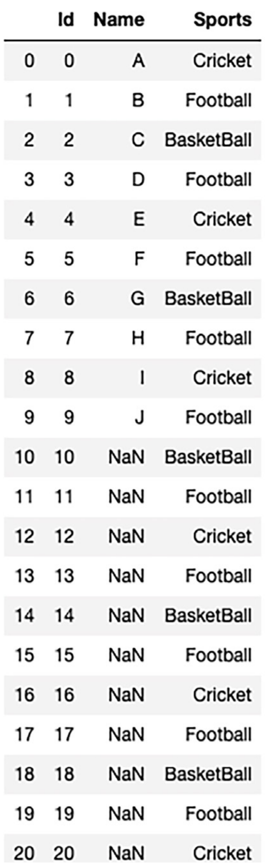
```
student.merge(sports_df, how=’inner’, on=’Id’)
```



	Id	Name	Sports
0	0	A	Cricket
1	1	B	Football
2	2	C	BasketBall
3	3	D	Football
4	4	E	Cricket
5	5	F	Football
6	6	G	BasketBall
7	7	H	Football
8	8	I	Cricket
9	9	J	Football

Figure 9.32

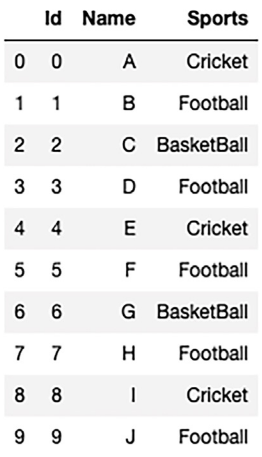
```
student.merge(sports_df, how=’outer’, on=’Id’)
```



	Id	Name	Sports
0	0	A	Cricket
1	1	B	Football
2	2	C	BasketBall
3	3	D	Football
4	4	E	Cricket
5	5	F	Football
6	6	G	BasketBall
7	7	H	Football
8	8	I	Cricket
9	9	J	Football
10	10	NaN	BasketBall
11	11	NaN	Football
12	12	NaN	Cricket
13	13	NaN	Football
14	14	NaN	BasketBall
15	15	NaN	Football
16	16	NaN	Cricket
17	17	NaN	Football
18	18	NaN	BasketBall
19	19	NaN	Football
20	20	NaN	Cricket

Figure 9.33

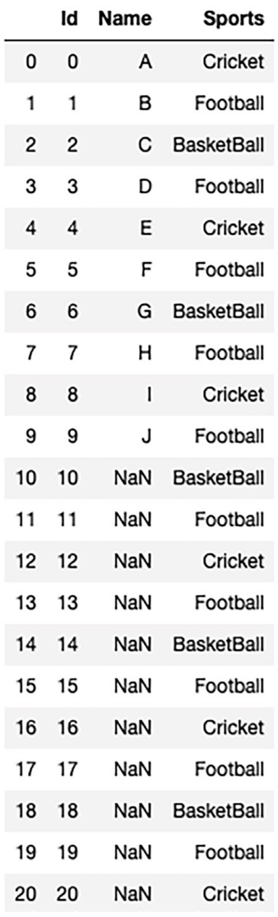
```
student.merge(sports_df, how='left', on='Id')
```



	Id	Name	Sports
0	0	A	Cricket
1	1	B	Football
2	2	C	BasketBall
3	3	D	Football
4	4	E	Cricket
5	5	F	Football
6	6	G	BasketBall
7	7	H	Football
8	8	I	Cricket
9	9	J	Football

Figure 9.34

```
student.merge(sports_df, how='right', on='Id')
```



	Id	Name	Sports
0	0	A	Cricket
1	1	B	Football
2	2	C	BasketBall
3	3	D	Football
4	4	E	Cricket
5	5	F	Football
6	6	G	BasketBall
7	7	H	Football
8	8	I	Cricket
9	9	J	Football
10	10	NaN	BasketBall
11	11	NaN	Football
12	12	NaN	Cricket
13	13	NaN	Football
14	14	NaN	BasketBall
15	15	NaN	Football
16	16	NaN	Cricket
17	17	NaN	Football
18	18	NaN	BasketBall
19	19	NaN	Football
20	20	NaN	Cricket

Figure 9.35

As we can see in all the outputs, the joins have been performed and uncommon data is filled with null values.

Reading and Writing Excel Sheets

Till now, we have seen how to work with a CSV sheet, but there we may be required to work using Excel sheets. For this, Pandas provides a functionality called `read_excel()`.

After reading an Excel sheet, we can perform exactly the same operations that we did with a CSV sheet. We will first use the preceding function to read an Excel sheet. We have converted the `titanic` dataset from CSV to Excel using Microsoft Excel; let's now read this Excel sheet:

```
import pandas as pd

data = pd.read_excel('titanic_train.xlsx')
```

You can now look at the data frame using the `head()` function. It may be possible that Pandas may ask for a dependent package to be installed before reading an Excel sheet. You just need to pip install them. One of those packages is `xlrd` (`pip install xlrd`).

Suppose we have multiple sheets in one Excel sheet; how to read them? We have divided the titanic dataset into two parts: the first 400 rows are saved in sheet 1, which is named `train_1`, and the remaining are in sheet 2, which is named `train_2`. Let's see how to read them:

```
data1 = pd.read_excel('titanic_train.xlsx', sheet_name='train_1')

data2 = pd.read_excel('titanic_train.xlsx', sheet_name='train_2')
```

Just like CSV, we can export the results of a data frame using the `to_excel()` function. We can add sheets as well. Let's save the titanic data frame in three sheets of a new Excel file:

```
# Initialize the Excel Writer Engine

writer = pd.ExcelWriter('three_sheets.xlsx', engine='xlsxwriter')


# Save Data frames to Different Sheets

df1.to_excel(writer, index=False, sheet_name='train_1')

df2.to_excel(writer, index=False, sheet_name='train_2')

df3.to_excel(writer, index=False, sheet_name='train_3')


# Add the sheets to the engine

workbook = writer.bookworksheet = writer.sheets['train_1']

workbook = writer.bookworksheet = writer.sheets['train_2']

workbook = writer.bookworksheet = writer.sheets['train_3']


# Write on the Disk

writer.save()
```

You may need to install the engine xlsxwriter (`pip install xlsxwriter`).

Exploring Groupby

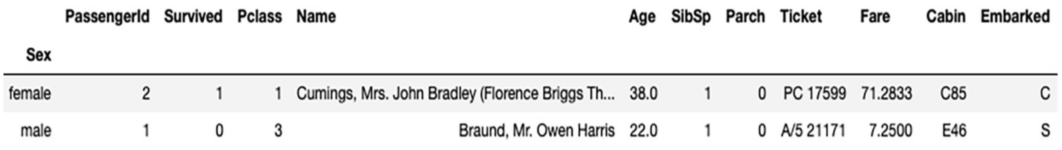
If we want to group data based on a particular column value or row value, we can use the `groupby()` function. We'll take the titanic dataset and group the data by gender. We can do so with the following code:

```
df = data.groupby('Sex')
```

This will group the data by gender. We can look at the first row of each gender by using the `first()` function:

```
df.first()
```

It will give us an output like the following:

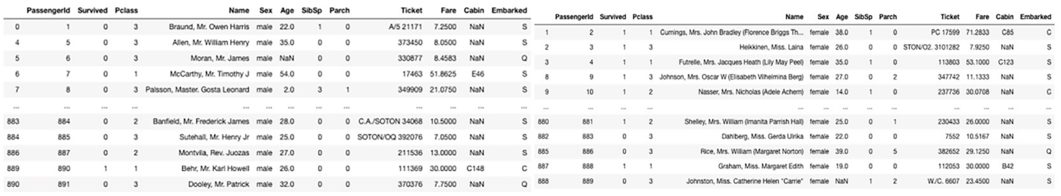


```
        PassengerId  Survived  Pclass  Name                                          Age   SibSp  Parch  Ticket     Fare     Cabin  Embarked
Sex
female            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  38.0      1      0  PC 17599   71.2833  C85           C
  male            1         0       3                          Braund, Mr. Owen Harris  22.0      1      0  A/5 21171   7.2500  E46           S
```

Figure 9.36: *Using groupby()*

We can get specific groups output as well:

```
df.get_group('male')
df.get_group('female')
```



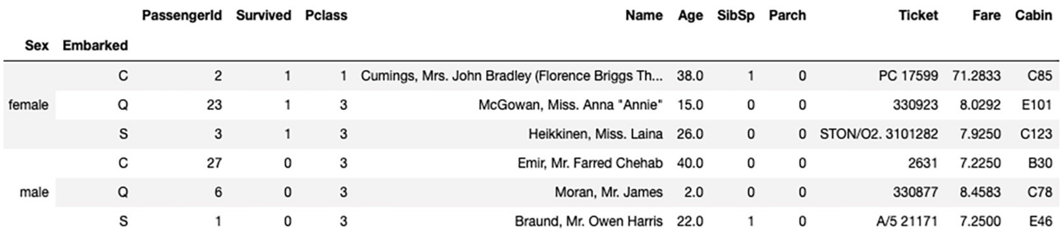
```
     PassengerId  Survived  Pclass                          Name   Sex   Age  SibSp  Parch             Ticket     Fare  Cabin  Embarked
  0            1         0       3       Braund, Mr. Owen Harris  male  22.0      1      0          A/5 21171   7.2500    NaN         S
  4            5         0       3      Allen, Mr. William Henry  male  35.0      0      0             373450   8.0500    NaN         S
  5            6         0       3             Moran, Mr. James  male   NaN      0      0             330877   8.4583    NaN         Q
  6            7         0       1       McCarthy, Mr. Timothy J  male  54.0      0      0              17463  51.8625    E46         S
  7            8         0       3 Palsson, Master. Gosta Leonard  male   2.0      3      1             349909  21.0750    NaN         S
...          ...       ...     ...                           ...   ...   ...    ...    ...                ...      ...    ...       ...
883          884         0       2     Banfield, Mr. Frederick James  male  28.0      0      0   C.A./SOTON 34068  10.5000    NaN         S
884          885         0       3         Sutehall, Mr. Henry Jr  male  25.0      0      0  SOTON/OQ 392076   7.0500    NaN         S
886          887         0       2         Montvila, Rev. Juozas  male  27.0      0      0             211536  13.0000    NaN         S
889          890         1       1           Behr, Mr. Karl Howell  male  26.0      0      0             111369  30.0000   C148         C
890          891         0       3             Dooley, Mr. Patrick  male  32.0      0      0             370376   7.7500    NaN         Q
```

```
     PassengerId  Survived  Pclass                                               Name     Sex   Age  SibSp  Parch             Ticket     Fare  Cabin  Embarked
  1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1      0           PC 17599  71.2833    C85         C
  2            3         1       3                              Heikkinen, Miss. Laina  female  26.0      0      0  STON/O2. 3101282   7.9250    NaN         S
  3            4         1       1          Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1      0             113803  53.1000   C123         S
  8            9         1       3    Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)  female  27.0      0      2             347742  11.1333    NaN         S
  9           10         1       2              Nasser, Mrs. Nicholas (Adele Achem)  female  14.0      1      0             237736  30.0708    NaN         C
...          ...       ...     ...                                                ...     ...   ...    ...    ...                ...      ...    ...       ...
880          881         1       2        Shelley, Mrs. William (Imanita Parrish Hall)  female  25.0      0      1             230433  26.0000    NaN         S
882          883         0       3                        Dahlberg, Miss. Gerda Ulrika  female  22.0      0      0               7552  10.5167    NaN         S
885          886         0       3           Rice, Mrs. William (Margaret Norton)  female  39.0      0      5             382652  29.1250    NaN         Q
887          888         1       1                     Graham, Miss. Margaret Edith  female  19.0      0      0             112053  30.0000    B42         S
888          889         0       3        Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1      2         W./C. 6607  23.4500    NaN         S
```

Figure 9.37: *Using groupby()*

We can use `groupby` for multiple columns as well:

```
df = data.groupby(['Sex', 'Embarked'])
df.first()
```



```
                 PassengerId  Survived  Pclass                                               Name   Age  SibSp  Parch            Ticket     Fare  Cabin
Sex     Embarked
female  C                  2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  38.0      1      0          PC 17599  71.2833    C85
        Q                 23         1       3                         McGowan, Miss. Anna "Annie"  15.0      0      0            330923   8.0292   E101
        S                  3         1       3                             Heikkinen, Miss. Laina  26.0      0      0  STON/O2. 3101282   7.9250   C123
male    C                 27         0       3                             Emir, Mr. Farred Chehab  40.0      0      0              2631   7.2250    B30
        Q                  6         0       3                                   Moran, Mr. James   2.0      0      0            330877   8.4583    C78
        S                  1         0       3                            Braund, Mr. Owen Harris  22.0      1      0         A/5 21171   7.2500    E46
```

Figure 9.38: *Using groupby()*

Binning in Pandas

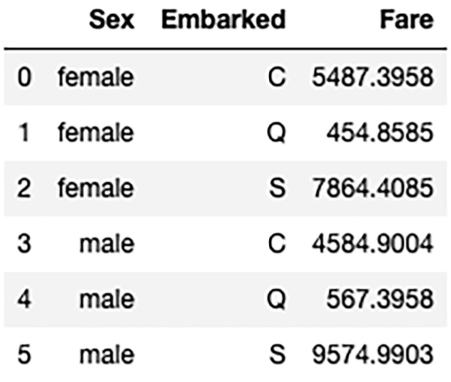
Binning is a concept used to convert a numerical continuous data into discrete categories. This is necessary when we want to create buckets. For example, it is always better to take bins of ages like 10-20, 21-30, 31-40, and so on, instead of taking the ages individually. This sort of creation of buckets can be done using two Pandas functions: `cut()` and `qcut()`.

In the titanic dataset, we have a column for fare. Instead of taking it as a number, we can convert it into a bin. Let's bin it on the basis of its frequency on **Gender and Embarked** type. In the previous section, we grouped the data by gender and embarked. Let's now try to find the perfect bins:

```
z = df['Fare'].sum().reset_index()
```

```
z
```

This will summarize/aggregate the data based on the sum of fare for each gender and each embarked type, as displayed in this screenshot:



	Sex	Embarked	Fare
0	female	C	5487.3958
1	female	Q	454.8585
2	female	S	7864.4085
3	male	C	4584.9004
4	male	Q	567.3958
5	male	S	9574.9903

Figure 9.39

Now, we can plot a histogram of this table to find the buckets:

```
z['Fare'].plot(kind='hist')
```

This will give us the following plot:

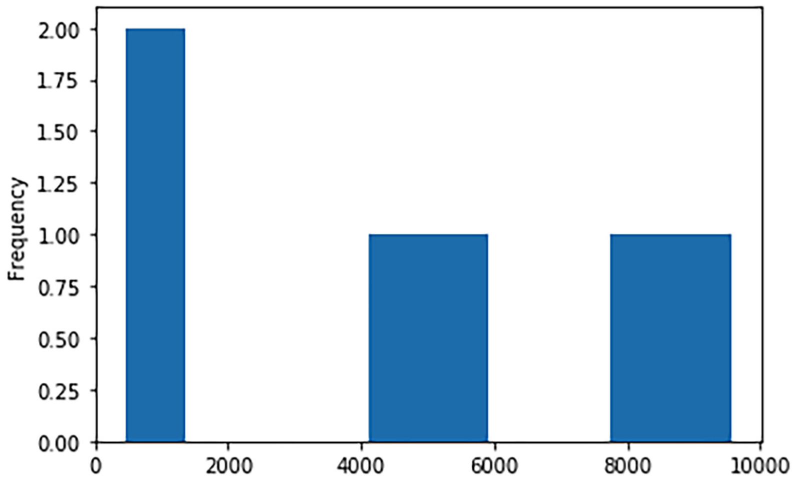


Figure 9.40: *Histogram*

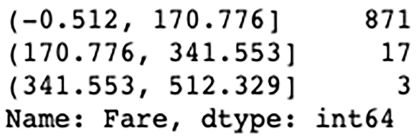
As shown in the diagram, we can bin the `Fare` column into three buckets using the `cut()` function. Let's first look at the coding application, and then we will explore the functions:

```
pd.cut(data['Fare'], 3)
```

In the mentioned code, we have broken the Fare column into three bins. We can see how many of them are in each bin using the `value_counts` function:

```
pd.cut(data['Fare'], 3).value_counts()
```

This will give us the following results:



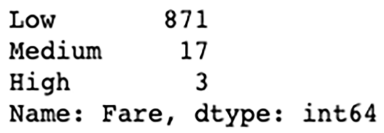
```
(-0.512, 170.776]     871
(170.776, 341.553]     17
(341.553, 512.329]      3
Name: Fare, dtype: int64
```

Figure 9.41

We can also name these bins:

```
pd.cut(data['Fare'], 3, labels = ['Low', 'Medium', 'High']).value_counts()
```

Here's the output:



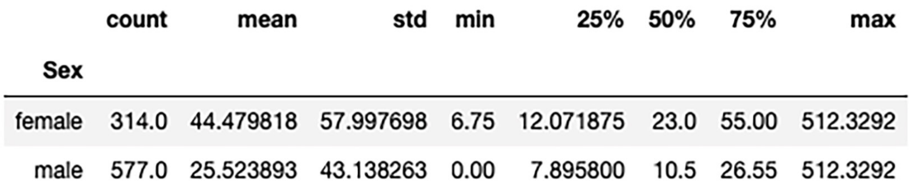
```
Low       871
Medium     17
High        3
Name: Fare, dtype: int64
```

Figure 9.42

In the preceding coding example, we have just written 3, and pandas have automatically defined the three bins. These bins have a constant size but each one may not have the same number of data points. Apart from this, we can provide custom bins using the `cut()` function. Let's bin the Fare column based on gender:

```
df = data.groupby([‘Sex’])[‘Fare’].describe()

df
```

The following is the output:



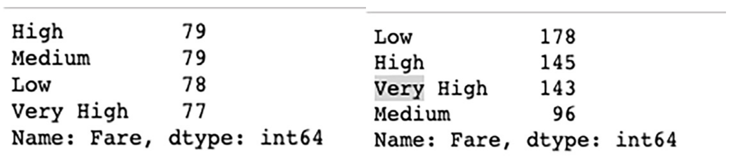
Sex	count	mean	std	min	25%	50%	75%	max
female	314.0	44.479818	57.997698	6.75	12.071875	23.0	55.00	512.3292
male	577.0	25.523893	43.138263	0.00	7.895800	10.5	26.55	512.3292

Figure 9.43

In the preceding code, we described the data for both the genders. Based on this, we will be doing the splits:

```
Female_Fare = data[data[‘Sex’]==’female’][‘Fare’]

pd.cut(Female_Fare, [6.75, 12.08, 23.1, 56, 513], labels = [‘Low’, ‘Medium’, ‘High’, ‘Very High’]).value_counts()


Male_Fare = data[data[‘Sex’]==’male’][‘Fare’]

pd.cut(Male_Fare, [0, 8, 11, 27, 513], labels = [‘Low’, ‘Medium’, ‘High’, ‘Very High’]).value_counts()
```

The preceding code split both Fares for male and Fares for females. This is the output:



```
High          79
Medium        79
Low           78
Very High     77
Name: Fare, dtype: int64
```

```
Low          178
High         145
Very High    143
Medium        96
Name: Fare, dtype: int64
```

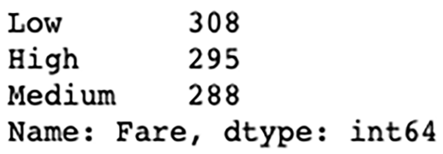
Figure 9.44

You can see that the number and distribution are different.

`When it comes to binning, qcut()` has different functionality. It bins the data in such a way that an equal amount of numbers is present in each bucket. Let's use the same data frame we used earlier and apply `qcut()` over it:

```
pd.qcut(data['Fare'], 3, labels = ['Low', 'Medium', 'High']).value_
counts()
```

This is the output:



```
Low       308
High      295
Medium    288
Name: Fare, dtype: int64
```

Figure 9.45

When we applied `pd.cut()` on this column, we got numbers like 871 in **Low**, 17 in **Medium**, and 3 in **High**. However, you can see that the numbers are almost equal in `qcut()`. So, we can conclude that we can use `pd.cut()`if we want to make custom bins, but we `pd.qcut() is more` suitable if we want equally distributed bins.

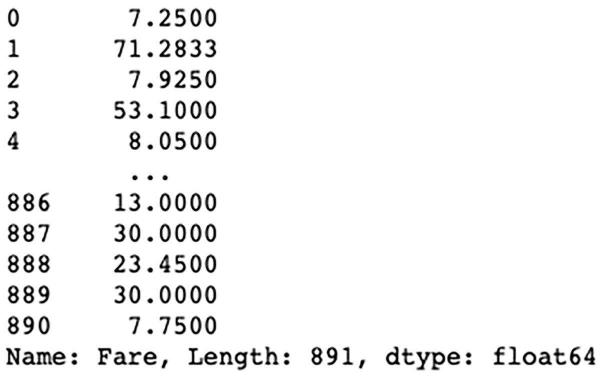
Pandas Series

Till now, we have worked with data frames in Pandas, but if we talk about an individual column in that data frame, the values present inside it are considered a series. It can be considered as an array having indexes, which may be sequential or random based on the way a series is extracted. However, the index is definitely **hashable,** which allows us to access the members inside a series.

So, in the `titanic` dataset, the Fare and `Age` columns are series, and so are all the other columns. There are two methods to extract the series from a data frame:

```
data['Fare'] or data.Fare
```

This will give us the following output:

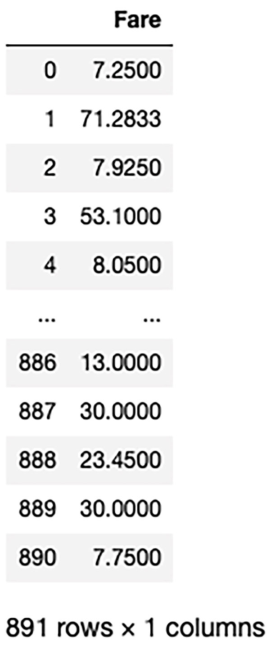


```
0       7.2500
1      71.2833
2       7.9250
3      53.1000
4       8.0500
         ...
886    13.0000
887    30.0000
888    23.4500
889    30.0000
890     7.7500
Name: Fare, Length: 891, dtype: float64
```

Figure 9.46

However, we get a data frame if we use double square brackets instead of a single one. Instead, you can look at the difference between how they look:

```
data[[‘Fare’]]
```



	Fare
0	7.2500
1	71.2833
2	7.9250
3	53.1000
4	8.0500
...	...
886	13.0000
887	30.0000
888	23.4500
889	30.0000
890	7.7500

891 rows × 1 columns

Figure 9.47

We can create a series simply by converting a list into a series, as follows:

```
names = [‘Sachin’, ‘Saurav’, ‘Rahul’]

df = pd.Series(names)

df
```

```
0    Sachin
1    Saurav
2     Rahul
dtype: object
```

Figure 9.48

We can perform several operations of Series, some of which we have already seen. Here's the table explaining those operations:

Operations:

Function	Usage
add()	Adding two Series
sub()	Subtracting two Series
mul()	Multiplying two Series
div()	Dividing two Series
sum()	Finding the sum of Series
prod()	Finding the product of Series
mean()	Finding the average of Series
pow()	Fiding the exponential power of Series
abs()	Finding the absolute value of Series
cov()	Finding the covariance between two Series

Table 9.1

Functions:

FUNCTION	DESCRIPTION
combine_first()	Combines two series into one
count()	Provides the number of non-null observations
size()	Returns the total number of elements
is_unique()	Checks whether the values in the object are unique
idxmax()	Extracts the index positions of the highest values in a Series
idxmin()	Extracts the index positions of the lowest values in a Series
sort_values()	Sorts the values in ascending or descending order
sort_index()	Sorts the series by the index instead of its values
astype()	Changes the data type of a series
tolist()	Converts a series to a list

Contd…

unique()	Sees the unique values in a particular column
nunique()	Provides the count of unique values
value_counts()	Count the number of the times each unique value occurs in a Series
apply()	Python function applied to every series value

Table 9.2

NumPy

NumPy is a short name given to Numerical Python. This package is used to perform any kind of numerical operation supporting the multi-dimensional data format. It is the most powerful library present, and it has a usage in almost all machine learning fields. Even the Pandas package we saw in the previous section has a base of NumPy. In this section, we'll explor this package and use most of its important features.

Creating Null Vector

We can create a numpy array of a particular dimension, having nothing but zeroes inside it. This is useful when you know the output dimension of a particular model but don't know what exactly the values may be. This concept is also useful during sparse matrix computations.

Indexing

The concept of indexing is similar to that of any data structures present in Python. Just like a list, we can extract a single value by providing an index. We can create a sequential numpy array having a start and end range using the `arange()` function:

```
a = np.arange(10)

print(a)

print(a[5])
```

This will return the sixth element of the array. Now, just like lists, we can get a range of values as well:

```
print(a[0:4])
```

This will give us the first five elements. We can also create an array of multiple arrays using the `np.array()` function:

```
a = np.array([[1,2,3], [4,5,6], [7,8,9]])

print(a)
```

This array contains three arrays, as we can see in the output. If we apply this to the index here, it will give us the array present inside as the value. Later, we can apply another indexing to get the exact values. Consider this example:

```
print(a[1])
```

This will give us the array at index 1.

```
print(a[1][0])
```

This will extract the array present at the first index and then extract the 0th index value from the extracted array. We can also provide negative indexes:

```
print(a[-1])
```

This will give us the first element from the right side of the array. We can also provide step functions to extract arrays:

```
a = np.arange(20)

print(a)

print(a[-6:19:1])
```

This will start on the right side and go to the sixth element from the right. Then, it will go to number nineteenth, which is index number 18. The last number 1 tells us about the step. Here, the step value is 1, so the difference will be of 1. If the step value would have been 2, every alternate value would have been extracted.

Using the concept of indexing, we can reverse the array using the following notation:

```
reversed_a = a[:,:,-1]

print(reversed_a)
```

Reshaping a Numpy Array

NumPy array has dimensions. You can imagine them as a matrix. So, if a matrix is of dimension 10x10, we can reshape it to 2x50, 25x4. Similarly, we can reshape a `numpy` array using the `reshape()` function. Let's create an array with 100 elements and then reshape it:

```
arr = np.arange(1,100)

shape10x10 = arr.reshape(10,10)

shape25x4= arr.reshape(25,4)
```

```
shape50x2= arr.reshape(50,2)


print(shape10x10)

print(shape25x4)

print(shape50x2)
```

Generating Random Values Using Numpy

Generating random numbers is extremely important in the field of statistics and data science, and Numpy provides a strong functionality to generate these numbers. Various methods can help us do this:

- Generating random integers – `random.randint()`
- Generating random numbers following normal distribution – `random.normal()`
- Generating random numbers following uniform distribution – `random.uniform()`
- Shuffling the existing array – `random.shuffle()`
- Seeding – `seed()`

```
a = np.random.randint(1,100,100)

b = np.random.normal(100)

c = np.random.uniform(100)


print(a)

print(b)

print(c)


z = np.arange(1,100)

print(np.random.shuffle(z))
```

Descriptive statistics using Numpy

We can get the descriptive statistics of an array using `numpy`. The following are the measures that we can find:

- Mean – `np.mean()`
- Median – `np.median()`
- Range – `np.ptp()`
- Percentile – `np.percentile()`
- Variance – `np.var()`
- Standard deviation – `np.std()`
- Minimum – `np.min()`
- Maximum – `np.max()`

```
a = np.random.randint(1,100,100)

print(np.mean(a))

print(np.median(a))

print(np.ptp(a))

print(np.var(a))

print(np.std(a))

print(np.min(a))

print(np.max(a))
```

Mathematical Operations Using Numpy

The different types of mathematical operations we can perform using NumPy are mentioned here:

- We can use the `np.dot()` function to find the dot product of two arrays
- We can use the `np.abs()` function to find the absolute values of each elements of an array
- We can use the following functions for element-wise mathematical operations:
    - `np.add()` - for element-wise addition
    - `np.divide()` – for element-wise division
    - `np.negative()` – for element-wise conversion to negative sign
    - `np.multiply()` – for element-wise multiplication
- We can find element-wise square root using the `np.sqrt()` function
- We can sort an array in ascending or descending order using `np.sort()`

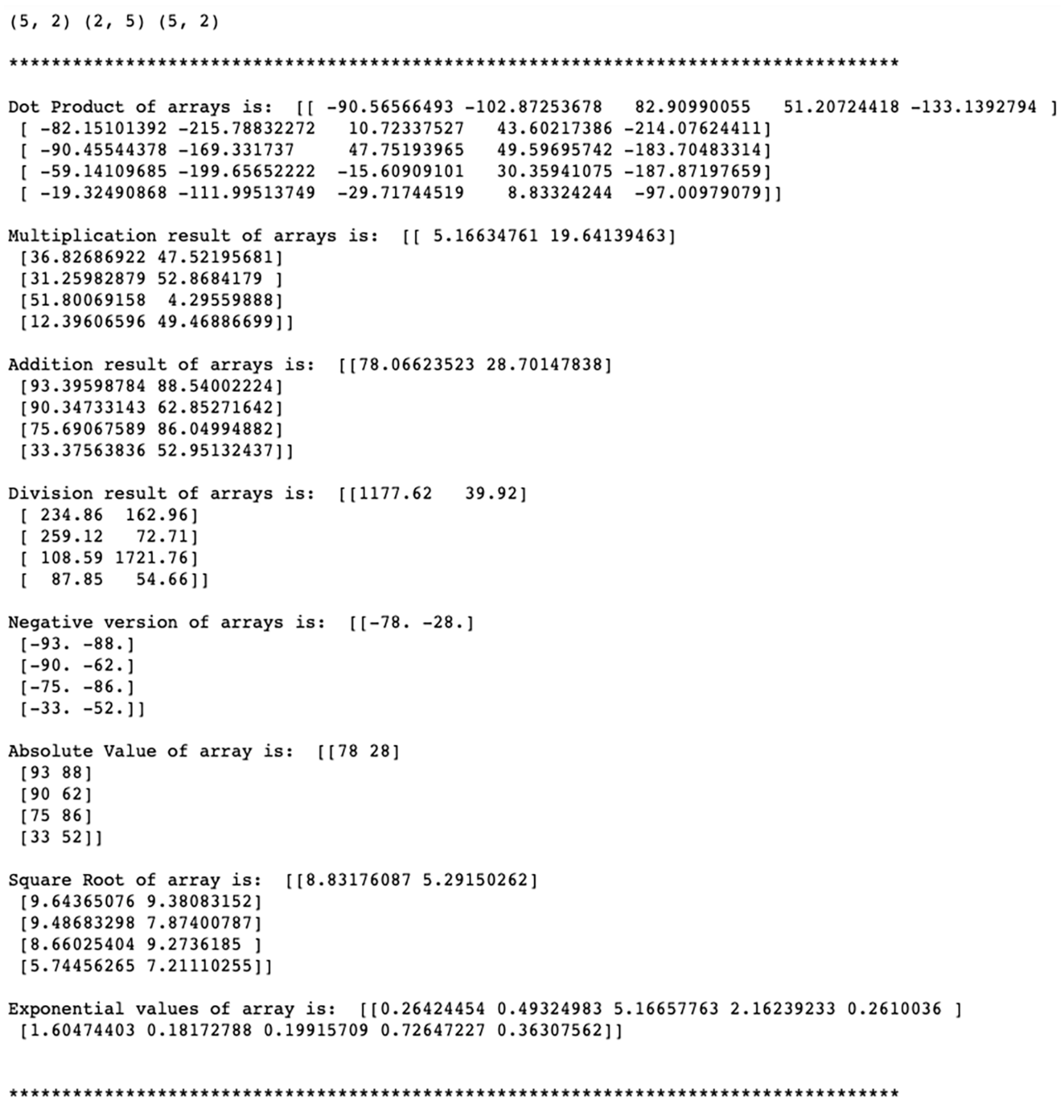
- We can find the exponent of a number or members in an array using the `np.exp()` function
- We can find the lowest integer nearest to a decimal number using `np.ceil()`
- We can find the highest integer nearest to a decimal number using the `np.floor()` function
- We can truncate all the numbers after the decimal with the `np.trunc()` function
- We can find unique elements present in an array using the `np.unique()` function

Let's now look at the application of all these operations in Python:

```
# Generating some random arrays
a = np.random.randint(1, 100, 10).reshape(5,2)
b = np.random.normal(size=10).reshape(2,5)
c = np.random.uniform(size=10).reshape(5,2)
z = np.array([2,2,2,4,4,4,6,6,6,7])

# Checking the Shape of the array
print(a.shape, b.shape, c.shape)

# Performing Mathematical operations on above arrays
print("\n*************************************************************
*********************\n")
print("Dot Product of arrays is: ", np.dot(a,b), "\n")
print("Multiplication result of arrays is: ", np.multiply(a,c), "\n")
print("Addition result of arrays is: ", np.add(a,c), "\n")
print("Division result of arrays is: ", np.around(np.divide(a,c),2), "\n")
print("Negative version of arrays is: ", np.negative(a,c), "\n")
print("Absolute Value of array is: ", np.abs(a), "\n")
print("Square Root of array is: ", np.sqrt(a), "\n")
print("Exponential values of array is: ", np.exp(b), "\n")

# Other important operations on arrays
print("\n*************************************************************
*********************\n")
```

```
print("Sorting the array: ", np.sort(a), "\n")
print("Finding the floor: ", np.floor(b), "\n")
print("Finding the ceil: ", np.ceil(b), "\n")
print("Truncating the array: ", np.trunc(b), "\n")
print("Finding unique numbers: ", np.unique(z), "\n")
```

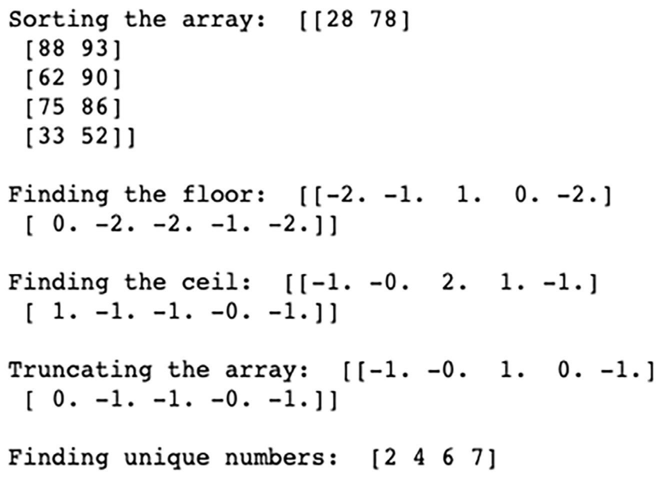
The output of the above code is given as follows:



```
(5, 2) (2, 5) (5, 2)

*********************************************************************************

Dot Product of arrays is:  [[ -90.56566493 -102.87253678   82.90990055   51.20724418 -133.1392794 ]
 [ -82.15101392 -215.78832272   10.72337527   43.60217386 -214.07624411]
 [ -90.45544378 -169.331737     47.75193965   49.59695742 -183.70483314]
 [ -59.14109685 -199.65652222  -15.60909101   30.35941075 -187.87197659]
 [ -19.32490868 -111.99513749  -29.71744519    8.83324244  -97.00979079]]

Multiplication result of arrays is:  [[ 5.16634761 19.64139463]
 [36.82686922 47.52195681]
 [31.25982879 52.8684179 ]
 [51.80069158  4.29559888]
 [12.39606596 49.46886699]]

Addition result of arrays is:  [[78.06623523 28.70147838]
 [93.39598784 88.54002224]
 [90.34733143 62.85271642]
 [75.69067589 86.04994882]
 [33.37563836 52.95132437]]

Division result of arrays is:  [[1177.62   39.92]
 [ 234.86  162.96]
 [ 259.12   72.71]
 [ 108.59 1721.76]
 [  87.85   54.66]]

Negative version of arrays is:  [[-78. -28.]
 [-93. -88.]
 [-90. -62.]
 [-75. -86.]
 [-33. -52.]]

Absolute Value of array is:  [[78 28]
 [93 88]
 [90 62]
 [75 86]
 [33 52]]

Square Root of array is:  [[8.83176087 5.29150262]
 [9.64365076 9.38083152]
 [9.48683298 7.87400787]
 [8.66025404 9.2736185 ]
 [5.74456265 7.21110255]]

Exponential values of array is:  [[0.26424454 0.49324983 5.16657763 2.16239233 0.2610036 ]
 [1.60474403 0.18172788 0.19915709 0.72647227 0.36307562]]


*********************************************************************************
```

Figure 9.49

The following image is the succeeding output:



```
Sorting the array:  [[28 78]
 [88 93]
 [62 90]
 [75 86]
 [33 52]]

Finding the floor:  [[-2. -1.  1.  0. -2.]
 [ 0. -2. -2. -1. -2.]]

Finding the ceil:  [[-1. -0.  2.  1. -1.]
 [ 1. -1. -1. -0. -1.]]

Truncating the array:  [[-1. -0.  1.  0. -1.]
 [ 0. -1. -1. -0. -1.]]

Finding unique numbers:  [2 4 6 7]
```

Figure 9.50

Other important features in Numpy

The following are the important features in Numpy:

- Padding values in an array to increase the size so that it matches. A fixed width can be done using the `np.pad()` function.
- `np.nan()` helps us find the null values present in an array or manually put null values in the array.
- `np.dtype()` is used to check the data type of an array.
- `np.astype()` is used to convert the array from one data type to another.
- `np.linspace()` is used to convert a range into an equal distance with smaller stops.
- `np.argmax()` helps return the index of the maximum element present in an array.
- `np.argmin()` helps return the index of the minimum element present in an array.
- `np.argsort()` is used to return the index of the sorted array.
- If any one element in an array is `True`, the `np.any()` function returns `True`, else `False`.
- `np.flat()`is used to convert the array into one dimension.
- `np.shape()` gives us the dimension of the array, which is the total number of rows and columns.

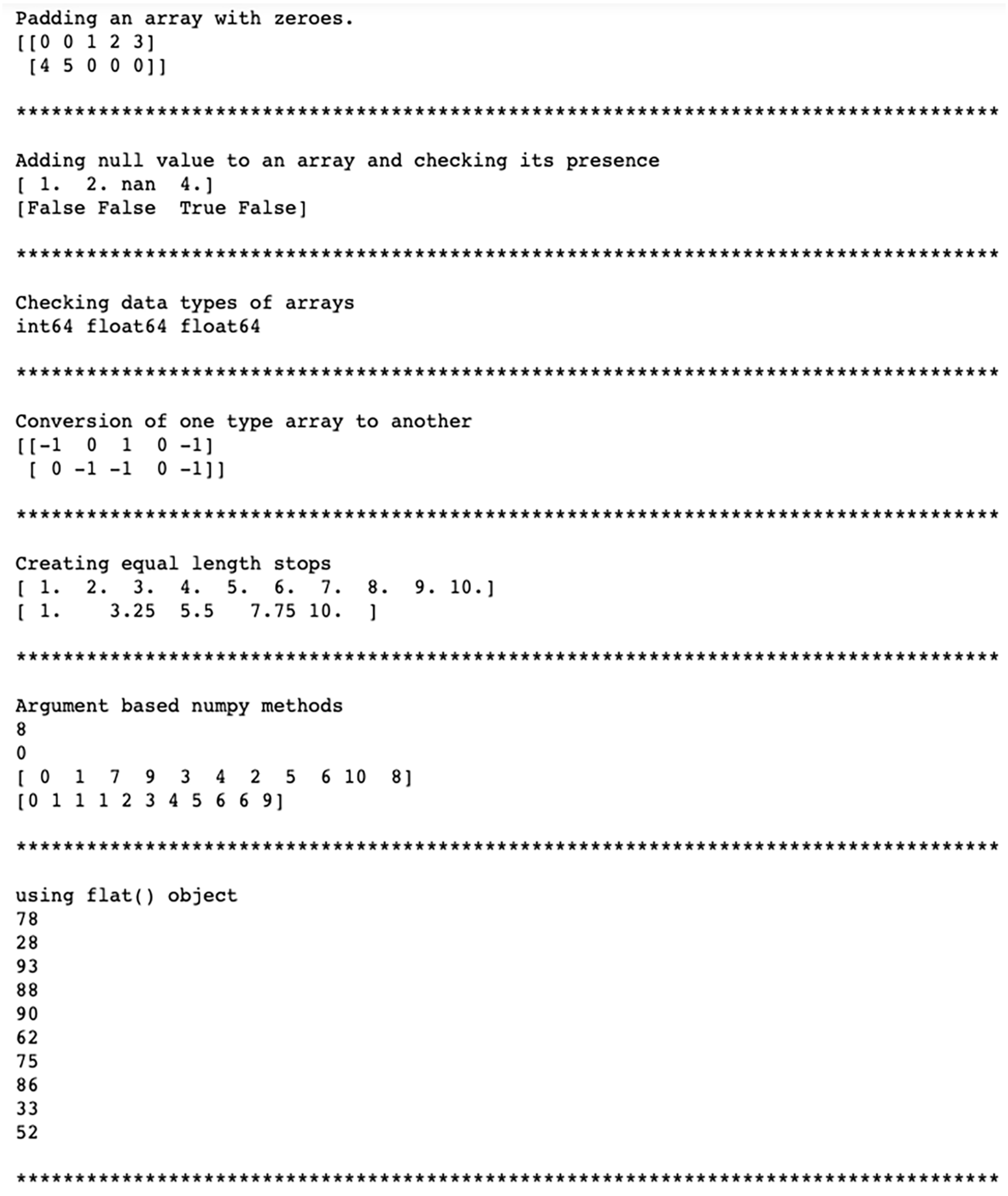
- `np.asarray()` coverts a list or any data structure into an array.
- `np.where()` is used to check a condition inside an array, just like conditional statements in Python.

Let’s look at the application of these functions in Python:

```
# Padding an array with zeroes.
print(“Padding an array with zeroes.”)
A = np.array([1,2,3,4,5])
print(np.pad(A, (2, 3), ‘constant’).reshape(2,5))
print(“\n***************************************************************
*********************\n”)

# Adding null value to an array and checking its presence
print(“Adding null value to an array and checking its presence”)
B = np.array([1,2,np.nan, 4])
print(B)
print(np.isnan(B))
print(“\n***************************************************************
*********************\n”)

# Checking data types of arrays
print(“Checking data types of arrays”)
print(a.dtype, b.dtype, c.dtype)
print(“\n***************************************************************
*********************\n”)

# Conversion of one type array to another
print(“Conversion of one type array to another”)
print(b.astype(int))
print(“\n***************************************************************
*********************\n”)

# Creating equal length stops
print(“Creating equal length stops”)
print(np.linspace(1,10,10))
print(np.linspace(1,10,5))
print(“\n***************************************************************
*********************\n”)

# Argument based numpy methods
print(“Argument based numpy methods”)
t = [0, 1,4,2,3,5,6,1,9,1,6]
print(np.argmax(t))
print(np.argmin(t))
print(np.argsort(t))
print(np.sort(t))
print(“\n***************************************************************
*********************\n”)

# using flat() object
print(“using flat() object”)
for i in a.flat:
print(i)
print(“\n***************************************************************
*********************\n”)

# Finding shape of arrays
print(“Finding shape of arrays”)
print(a.shape, b.shape, c.shape)
print(“\n***************************************************************
*********************\n”)

# Converting a list to array and reshaping it
print(“Converting a list to array and reshaping it”)
ls = [1,2,3,4,5,6,7,8,9,0]
print(np.asarray(ls))
print(np.asarray(ls).reshape(2,5))
print(“\n***************************************************************
*********************\n”)
```

```
# Checking for a condition inside an array
print("Checking for a condition inside an array")
print(np.where(b<0, 'negative', 'positive'))
```

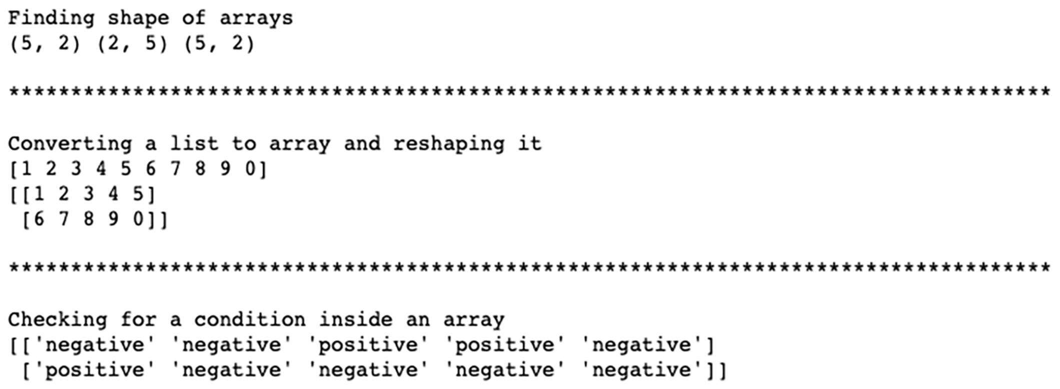
The preceding code gives this output:



```
Padding an array with zeroes.
[[0 0 1 2 3]
 [4 5 0 0 0]]

*************************************************************************************

Adding null value to an array and checking its presence
[ 1.  2. nan  4.]
[False False  True False]

*************************************************************************************

Checking data types of arrays
int64 float64 float64

*************************************************************************************

Conversion of one type array to another
[[-1  0  1  0 -1]
 [ 0 -1 -1  0 -1]]

*************************************************************************************

Creating equal length stops
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[ 1.    3.25  5.5   7.75 10.  ]

*************************************************************************************

Argument based numpy methods
8
0
[ 0  1  7  9  3  4  2  5  6 10  8]
[0 1 1 1 2 3 4 5 6 6 9]

*************************************************************************************

using flat() object
78
28
93
88
90
62
75
86
33
52

*************************************************************************************
```

Figure 9.51

The following is the succeeding output screen:



```
Finding shape of arrays
(5, 2) (2, 5) (5, 2)

************************************************************************************

Converting a list to array and reshaping it
[1 2 3 4 5 6 7 8 9 0]
[[1 2 3 4 5]
 [6 7 8 9 0]]

************************************************************************************

Checking for a condition inside an array
[['negative' 'negative' 'positive' 'positive' 'negative']
 ['positive' 'negative' 'negative' 'negative' 'negative']]
```

Figure 9.52

Conclusion

In this chapter, we learned the usage of different Python libraries. These libraries are used along with the `scipy` library, which we covered in the previous chapters in the field of statistics, machine learning, and artificial intelligence. One should be very comfortable using these libraries because statistics is the foundation for understanding the core of advanced fields like machine learning. Also, these libraries will help you apply all the advanced concepts in Python to solve problem statements.

The next chapter will provide an introduction to machine learning and use some of the libraries to execute a few algorithms.

Chapter 10
Non-Parametric Statistics

In the last few chapters, we looked at different types of statistical tests like t-test, z-test, and ANOVA test. All these tests had one thing in common: an assumption about the underlying distribution of the population. However, if we are unable to find out about the distribution, we move on to use the field of statistics called **Non-Parametric Statistics**.

Structure

- Test for randomness
- Sign test
- Wilcoxon test
- Mann Whitney test
- Spearman rank correlation test
- Kruskal Wallis test

Objective

In this chapter, we will look at some of the statistical tests in the domain of non-parametric statistics. Let's start this chapter by going through the algorithms one by one.

The test for randomness

In statistics, we talk a lot about taking data at random. This way, we try to remove any kind of bias in our analysis. That said, are we really sure that the data we are working on is random? For this, we'll have a non-parametric test called test for randomness. Let's understand this test with the help of an example.

Suppose we ordered pizzas from a nearby restaurant and have recently observed that the quality of the packaging is not good, due to which the quality of the pizzas is getting affected. Is this happening randomly, or is an employee of the restaurant doing it deliberately? To find this, let's apply the test of randomness on 16 consecutive pizza deliveries.

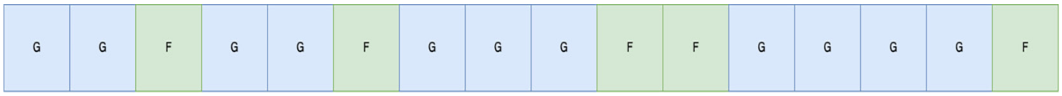


Figure 10.1: *Consecutive pizza deliveries*

The preceding sequence tells about 16 consecutive pizza deliveries, where *G* represents Good Pizzas and *F* represents Faulty ones. To start the test, we must know two related terminologies: Run and Length.

Run: The same events following each other, considering that all are mutually exclusives.

Length: Number of events in a single run.

If we break down the deliveries based on these terminologies:

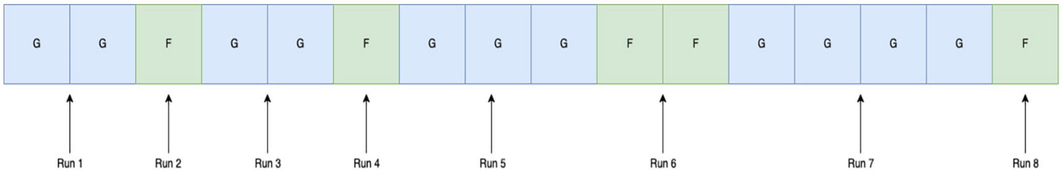


Figure *10.2: Depiction of runs*

In the preceding diagram, you can see that there are 8 runs, with *Run 7* having a maximum length of *4*. Let's understand the hypothesis for the test of randomness before applying the actual measure.

$$NullHypothesis = H_0 = TheEventsareRandom$$

$$AlternateHypothesis = H_A = TheEventsarenotRandom$$

Notations that we'll use for the test are as follows:

$$n{:}TotalSample$$

$$n_1{:}ObservationsofFirstType$$

$$n_2{:}ObservationsofSecondType$$

$$r{:}NumberofRuns$$

Let's represent our pizza example with the preceding notation:

$$n = 16$$

$$n_1 = 11$$

$$n_2 = 5$$

$$r = 8$$

The test statistic for the test of randomness can be defined in the following ways:

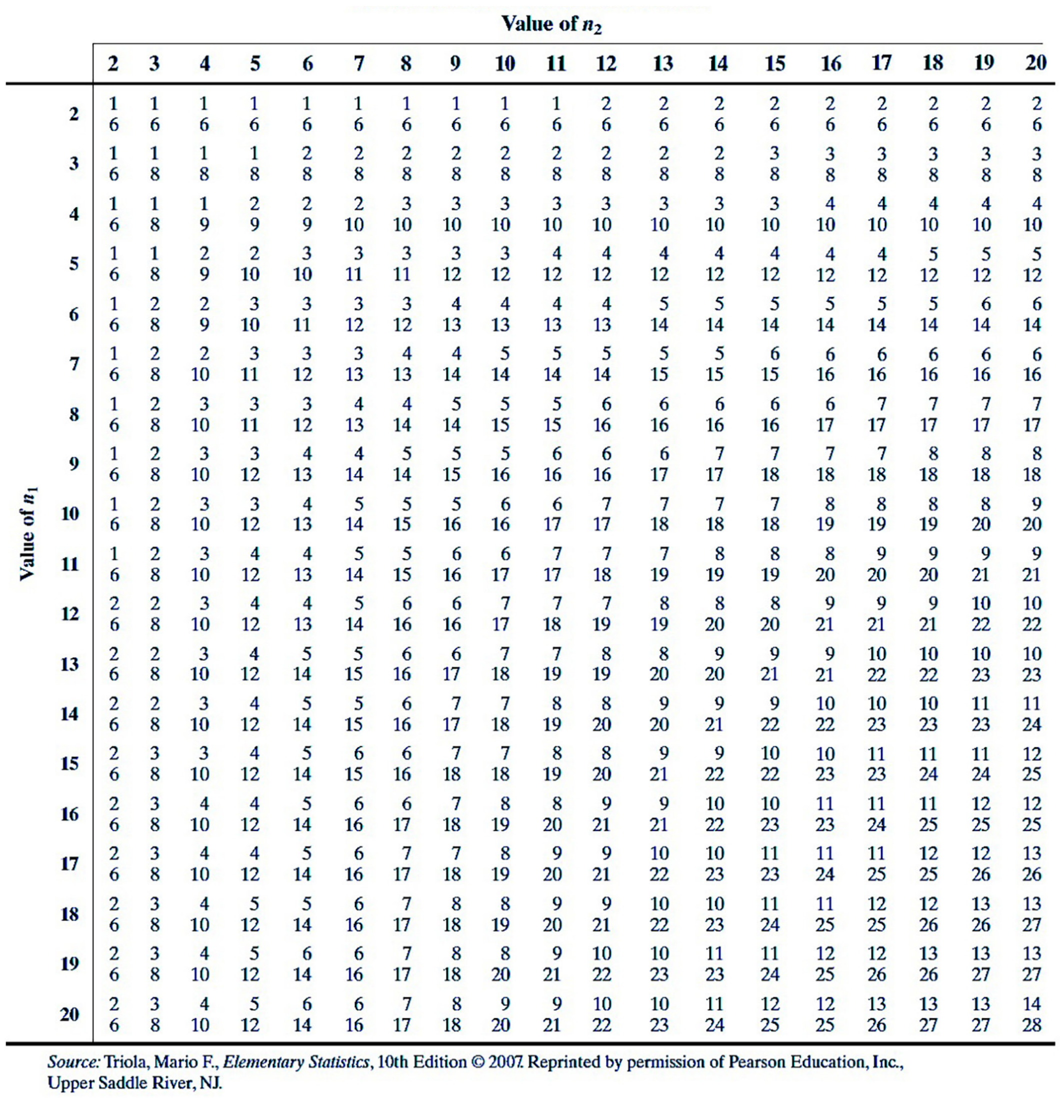
$$if n_1 \le 20 \wedge n_2 \le 20 then Test Statistic Z_0 = r$$

$$if n_1 > 20 \wedge n_2 > 20 then Test Statistic Z_0 = \frac{(r - \mu_r)}{\sigma_r}$$

$$where \mu_r = \frac{2n_1n_2}{n} + 1,$$

$$\sigma_r = \sqrt{\left(\frac{2n_1n_2(2n_1n_2 - n)}{n^2(n-1)}\right)}$$

For the preceding example, both n_1 and n_2 are less than *20*, so our test statistic will be *8*. We will use the following run test table to find the critical test statistic range:



	Value of n_2																		
Value of n_1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
2	1, 6	1, 6	1, 6	1, 6	1, 6	1, 6	1, 6	1, 6	1, 6	1, 6	2, 6	2, 6	2, 6	2, 6	2, 6	2, 6	2, 6	2, 6	2, 6
3	1, 6	1, 8	1, 8	1, 8	2, 8	2, 8	2, 8	2, 8	2, 8	2, 8	2, 8	2, 8	2, 8	3, 8	3, 8	3, 8	3, 8	3, 8	3, 8
4	1, 6	1, 8	1, 9	2, 9	2, 9	2, 10	3, 10	3, 10	3, 10	3, 10	3, 10	3, 10	3, 10	3, 10	4, 10	4, 10	4, 10	4, 10	4, 10
5	1, 6	1, 8	2, 9	2, 10	3, 10	3, 11	3, 11	3, 12	3, 12	4, 12	4, 12	4, 12	4, 12	4, 12	4, 12	4, 12	5, 12	5, 12	5, 12
6	1, 6	2, 8	2, 9	3, 10	3, 11	3, 12	3, 12	4, 13	4, 13	4, 13	4, 13	5, 14	5, 14	5, 14	5, 14	5, 14	5, 14	6, 14	6, 14
7	1, 6	2, 8	2, 10	3, 11	3, 12	3, 13	4, 13	4, 14	5, 14	5, 14	5, 14	5, 15	5, 15	6, 15	6, 16	6, 16	6, 16	6, 16	6, 16
8	1, 6	2, 8	3, 10	3, 11	3, 12	4, 13	4, 14	5, 14	5, 15	5, 15	6, 16	6, 16	6, 16	6, 16	6, 17	7, 17	7, 17	7, 17	7, 17
9	1, 6	2, 8	3, 10	3, 12	4, 13	4, 14	5, 14	5, 15	5, 16	6, 16	6, 16	6, 17	7, 17	7, 18	7, 18	7, 18	8, 18	8, 18	8, 18
10	1, 6	2, 8	3, 10	3, 12	4, 13	5, 14	5, 15	5, 16	6, 16	6, 17	7, 17	7, 18	7, 18	7, 18	8, 19	8, 19	8, 19	8, 20	9, 20
11	1, 6	2, 8	3, 10	4, 12	4, 13	5, 14	5, 15	6, 16	6, 17	7, 17	7, 18	7, 19	8, 19	8, 19	8, 20	9, 20	9, 20	9, 21	9, 21
12	2, 6	2, 8	3, 10	4, 12	4, 13	5, 14	6, 16	6, 16	7, 17	7, 18	7, 19	8, 19	8, 20	8, 20	9, 21	9, 21	9, 21	10, 22	10, 22
13	2, 6	2, 8	3, 10	4, 12	5, 14	5, 15	6, 16	6, 17	7, 18	7, 19	8, 19	8, 20	9, 20	9, 21	9, 21	10, 22	10, 22	10, 23	10, 23
14	2, 6	2, 8	3, 10	4, 12	5, 14	5, 15	6, 16	7, 17	7, 18	8, 19	8, 20	9, 20	9, 21	9, 22	10, 22	10, 23	10, 23	11, 23	11, 24
15	2, 6	3, 8	3, 10	4, 12	5, 14	6, 15	6, 16	7, 18	7, 18	8, 19	8, 20	9, 21	9, 22	10, 22	10, 23	11, 23	11, 24	11, 24	12, 25
16	2, 6	3, 8	4, 10	4, 12	5, 14	6, 16	6, 17	7, 18	8, 19	8, 20	9, 21	9, 21	10, 22	10, 23	11, 23	11, 24	11, 25	12, 25	12, 25
17	2, 6	3, 8	4, 10	4, 12	5, 14	6, 16	7, 17	7, 18	8, 19	9, 20	9, 21	10, 22	10, 23	11, 23	11, 24	11, 25	12, 25	12, 26	13, 26
18	2, 6	3, 8	4, 10	5, 12	5, 14	6, 16	7, 17	8, 18	8, 19	9, 20	9, 21	10, 22	10, 23	11, 24	11, 25	12, 25	12, 26	13, 26	13, 27
19	2, 6	3, 8	4, 10	5, 12	6, 14	6, 16	7, 17	8, 18	8, 20	9, 21	10, 22	10, 23	11, 23	11, 24	12, 25	12, 26	13, 26	13, 27	13, 27
20	2, 6	3, 8	4, 10	5, 12	6, 14	6, 16	7, 17	8, 18	9, 20	9, 21	10, 22	10, 23	11, 24	12, 25	12, 25	13, 26	13, 27	13, 27	14, 28

Source: Triola, Mario F., *Elementary Statistics*, 10th Edition © 2007. Reprinted by permission of Pearson Education, Inc., Upper Saddle River, NJ.

Figure 10.3: *Table for test of randomness*

In the preceding table with the consecutive values, we get the critical range as (*4,12*). Since our test statistic is in the range (as z_0 is *8*), we will the null hypothesis. So, we can say that the defects in pizza delivery are totally random and not due to a particular reason.

Let's take another randomness test example with a larger sample and look at how to use Python to perform the test.

The following table indicates the percentage change in the stock price of a company. *N* represents a negative change, while *P* represents a positive change. The data is taken for *45* consecutive days. Test the hypothesis that the stock price fluctuates randomly on a daily basis. Take the level of significance value as *0.05*.

Date	Return	Date	Return	Date	Return
2/24	N	3/17	P	4/8	P
2/25	N	3/18	P	4/9	P
2/26	N	3/19	N	4/10	P
2/27	N	3/20	P	4/13	N
3/2	P	3/23	N	4/14	N
3/3	P	3/24	N	4/15	N
3/4	N	3/25	N	4/16	N
3/5	P	3/26	N	4/17	N
3/6	N	3/27	P	4/20	P
3/9	P	3/30	P	4/21	P
3/10	N	3/31	N	4/22	N
3/11	N	4/1	N	4/23	N
3/12	P	4/2	P	4/24	N
3/13	N	4/6	P	4/27	N
3/16	P	4/7	P	4/28	N

Figure 10.4: *Stock prices for 45 days*

Step 1: Find the values of n, n_1, n_2:

$$n = 45$$

$$n_1 = CountofPositivePercentageChanges = 19$$

$$n_2 = CountofNegativePercentageChanges = 26$$

$$r = 19$$

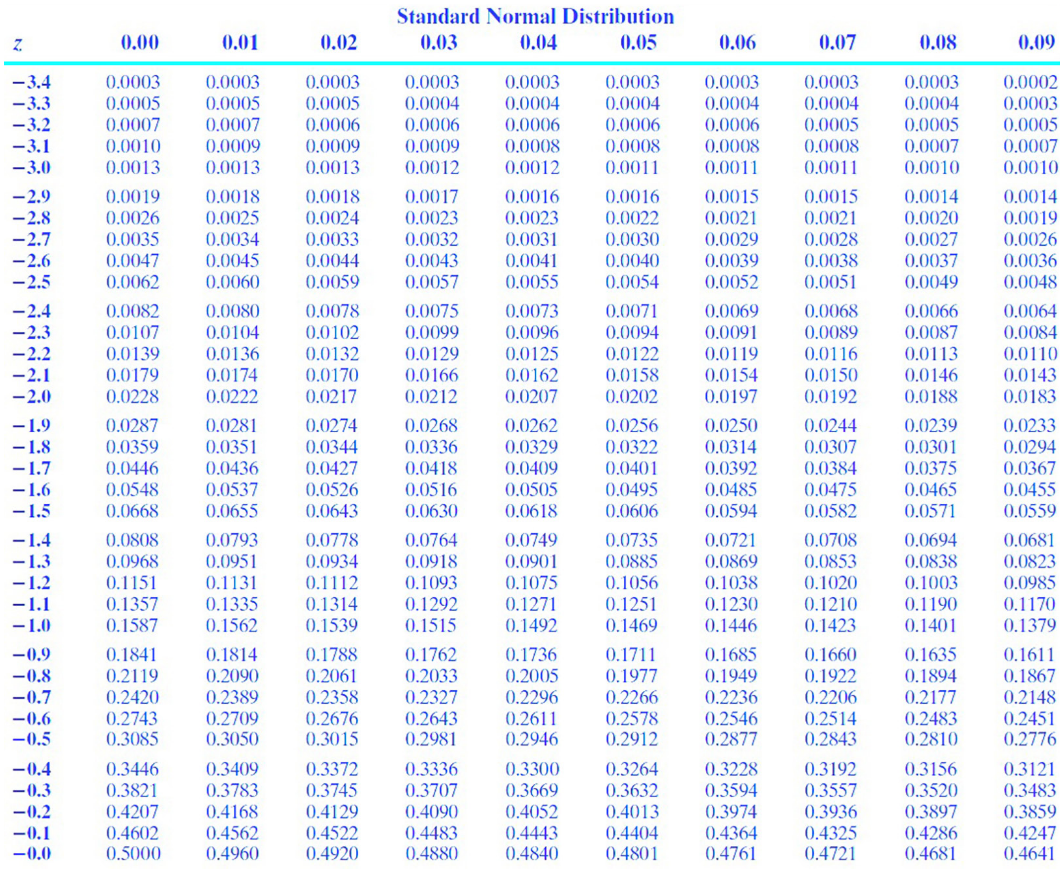
As n_2 is greater than 20; we will use a large sample test statistic. So, let's first determine the value of mean and variance:

$$\mu_r = \frac{2n_1n_2}{n} + 1 = \frac{2\times19\times26}{45} + 1 = 22.96$$

$$\sigma_r = \sqrt{\left(\frac{2n_1n_2(2n_1n_2 - n)}{n^2(n-1)}\right)} = \sqrt{\left(\frac{2\times19\times26(2\times19\times26)}{45^2(45-1)}\right)} = \sqrt{\frac{931684}{89100}} = 3.23$$

$$Z_0 = \frac{(r - \mu_r)}{\sigma_r} = \frac{(19 - 22.96)}{3.23} = -1.23$$

Let's look at the **Z_critical** from the *Standard Normal Distribution* table in the following screenshot:



Standard Normal Distribution

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
−3.4	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0002
−3.3	0.0005	0.0005	0.0005	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0003
−3.2	0.0007	0.0007	0.0006	0.0006	0.0006	0.0006	0.0006	0.0005	0.0005	0.0005
−3.1	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
−3.0	0.0013	0.0013	0.0013	0.0012	0.0012	0.0011	0.0011	0.0011	0.0010	0.0010
−2.9	0.0019	0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014	0.0014
−2.8	0.0026	0.0025	0.0024	0.0023	0.0023	0.0022	0.0021	0.0021	0.0020	0.0019
−2.7	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0026
−2.6	0.0047	0.0045	0.0044	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036
−2.5	0.0062	0.0060	0.0059	0.0057	0.0055	0.0054	0.0052	0.0051	0.0049	0.0048
−2.4	0.0082	0.0080	0.0078	0.0075	0.0073	0.0071	0.0069	0.0068	0.0066	0.0064
−2.3	0.0107	0.0104	0.0102	0.0099	0.0096	0.0094	0.0091	0.0089	0.0087	0.0084
−2.2	0.0139	0.0136	0.0132	0.0129	0.0125	0.0122	0.0119	0.0116	0.0113	0.0110
−2.1	0.0179	0.0174	0.0170	0.0166	0.0162	0.0158	0.0154	0.0150	0.0146	0.0143
−2.0	0.0228	0.0222	0.0217	0.0212	0.0207	0.0202	0.0197	0.0192	0.0188	0.0183
−1.9	0.0287	0.0281	0.0274	0.0268	0.0262	0.0256	0.0250	0.0244	0.0239	0.0233
−1.8	0.0359	0.0351	0.0344	0.0336	0.0329	0.0322	0.0314	0.0307	0.0301	0.0294
−1.7	0.0446	0.0436	0.0427	0.0418	0.0409	0.0401	0.0392	0.0384	0.0375	0.0367
−1.6	0.0548	0.0537	0.0526	0.0516	0.0505	0.0495	0.0485	0.0475	0.0465	0.0455
−1.5	0.0668	0.0655	0.0643	0.0630	0.0618	0.0606	0.0594	0.0582	0.0571	0.0559
−1.4	0.0808	0.0793	0.0778	0.0764	0.0749	0.0735	0.0721	0.0708	0.0694	0.0681
−1.3	0.0968	0.0951	0.0934	0.0918	0.0901	0.0885	0.0869	0.0853	0.0838	0.0823
−1.2	0.1151	0.1131	0.1112	0.1093	0.1075	0.1056	0.1038	0.1020	0.1003	0.0985
−1.1	0.1357	0.1335	0.1314	0.1292	0.1271	0.1251	0.1230	0.1210	0.1190	0.1170
−1.0	0.1587	0.1562	0.1539	0.1515	0.1492	0.1469	0.1446	0.1423	0.1401	0.1379
−0.9	0.1841	0.1814	0.1788	0.1762	0.1736	0.1711	0.1685	0.1660	0.1635	0.1611
−0.8	0.2119	0.2090	0.2061	0.2033	0.2005	0.1977	0.1949	0.1922	0.1894	0.1867
−0.7	0.2420	0.2389	0.2358	0.2327	0.2296	0.2266	0.2236	0.2206	0.2177	0.2148
−0.6	0.2743	0.2709	0.2676	0.2643	0.2611	0.2578	0.2546	0.2514	0.2483	0.2451
−0.5	0.3085	0.3050	0.3015	0.2981	0.2946	0.2912	0.2877	0.2843	0.2810	0.2776
−0.4	0.3446	0.3409	0.3372	0.3336	0.3300	0.3264	0.3228	0.3192	0.3156	0.3121
−0.3	0.3821	0.3783	0.3745	0.3707	0.3669	0.3632	0.3594	0.3557	0.3520	0.3483
−0.2	0.4207	0.4168	0.4129	0.4090	0.4052	0.4013	0.3974	0.3936	0.3897	0.3859
−0.1	0.4602	0.4562	0.4522	0.4483	0.4443	0.4404	0.4364	0.4325	0.4286	0.4247
−0.0	0.5000	0.4960	0.4920	0.4880	0.4840	0.4801	0.4761	0.4721	0.4681	0.4641

Figure 10.5: *Table for normal distribution*

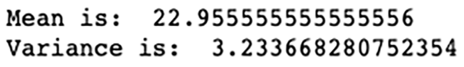
For a *0.05* level of significance and the value of Z_0 as *-1.23*, we get p-value to be *0.1093*. This value is greater than *0.05*, so *we have to accept the null hypothesis*. So, we can conclude that the fluctuation in the stock market is random.

Let's solve this problem using Python. Here's the code:

```
# Importing important libraries

import numpy as np

fromscipy.stats import norm
```

```
# Defining the methods required
def mean(n1, n2, n):
return ((2*n1*n2)/n)+1

def variance(n1,n2,n):
returnnp.sqrt((2*n1*n2*(2*n1*n2-n))/(n**2*(n-1)))

def statistic(r, mean, variance):
   return (r-mean)/variance

# Calculating Mean and Variance
m = mean(19,26,45)
print("Mean is: ", m)

v = variance(19,26,45)
print("Variance is: ", v)
```

The following is the output:



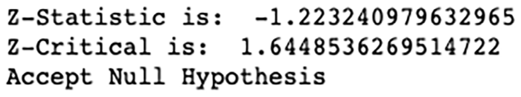
```
Mean is:  22.955555555555556
Variance is:  3.233668280752354
```

Figure 10.6

```
z_statistic = statistic(19,m,v)
print("Z-Statistic is: ", z_statistic)

# define level of significance
a = 0.05

# Getting Critical Value
value = norm.ppf(1-a)
```

```
print(“Z-Critical is: “, value)


if z_statistic>value:

print(“Reject Null Hypothesis”)

else:

print(“Accept Null Hypothesis”)
```

We get the following output:



```
Z-Statistic is:  -1.223240979632965
Z-Critical is:  1.6448536269514722
Accept Null Hypothesis
```

Figure 10.7

Sign Tests

You can say that sign tests are equivalents of t-test or z-test when we talk about non-parametric statistics. These tests can be sub-divided into three individual tests:

- One-sample sign test
- Two-sample dependent sign test or Wilcoxon test
- Two-sample independent sign test or Mann-Whitney test

One-sample Sign Test

In all the sign tests, we use the median as the parameter for hypothesis testing. When we talk about a single sample sign test, the rules of the test statistic are defined as follows:

For a two-tailed test:

$$H_0 : M = M_0$$

$$H_A : M \neq M_0$$

For a left-tailed test:

$$H_0 : M = M_0$$

$$H_A : M < M_0$$

For a right-tailed test:

$$H_0 : M = M_0$$

$$H_A : M > M_0$$

In all the preceding cases, *M* represents the median. To perform a sample test, we rank all the observations in ascending order. Now, the observations that are lower than the median are considered negative, while the ones greater than the median are considered positive. So, we must calculate the total positive and negative observations. Based on this number, we define the test statistics for the different types of tests that we mentioned earlier:

- For a two-tailed test, the test statistic will be the positive or negative observations having the lowest count.
- For left-tailed test, the test statistic will be the count of positive observations.
- For right-tailed test, the test statistic will be the count of negative observations.

The above-mentioned values of the test statistic only hold valid when the total number of observations is less than or equal to *25*. However, if there are more observations, they follow the normal distribution table, so the test statistic becomes as follows:

$$z_0 = \frac{(k + 0.5) - \frac{n}{2}}{\sqrt{\frac{n}{2}}}$$

Here, *k* represents the positive or negative observations having the lowest count.

Let's understand one-hand sign test with the help of an example. Suppose we know that the median pH level of rain at a particular place is *5.25*. A biologist thinks that the pH of the rain occurring at that place has increased. To test this hypothesis, she obtains a random sample of rain for 15 days. The following table shows the pH values she obtained. Is her hypothesis true?

5.31	5.19	5.55	5.38	5.37
5.19	5.26	5.29	5.27	5.19
5.27	5.36	5.22	5.28	5.24

Source: National Atmospheric Deposition Program

Figure 10.8: *pH values of rain*

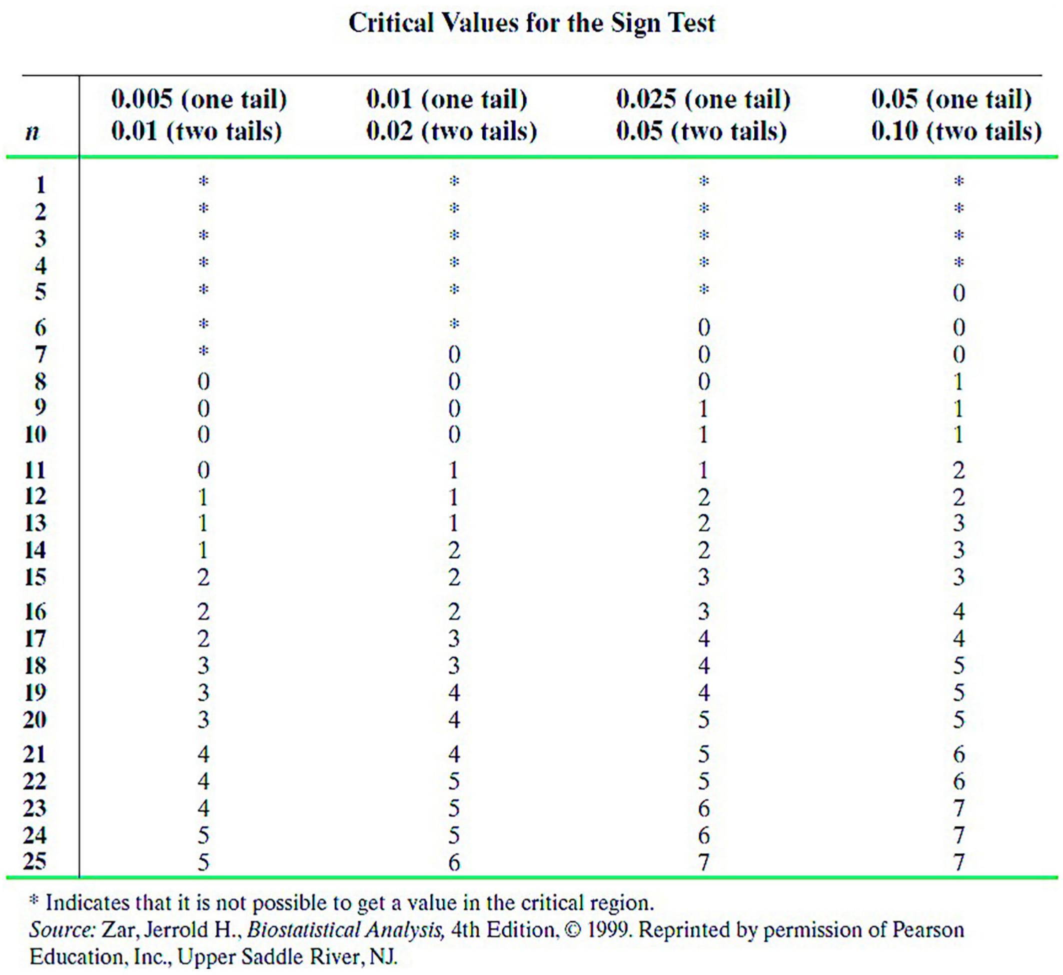
Here are the null and alternate hypothesis for the problem:

$$H_0 = pHisatsamelevel$$
$$H_A = pHisincreased$$
$$n = 15 \because smallsampletest$$
$$M_0 = 5.25$$

Observations less than the median (Negative ones) = 5

Observations greater than the median (Positive ones) = 10

We need to check whether the pH has increased, which means it will be a right-tailed test. For a right-tailed test, the test statistic will be the count of negative observations, which is 5. Now, we'll look at the following table to validate the hypothesis:



Critical Values for the Sign Test

n	0.005 (one tail) 0.01 (two tails)	0.01 (one tail) 0.02 (two tails)	0.025 (one tail) 0.05 (two tails)	0.05 (one tail) 0.10 (two tails)
1	*	*	*	*
2	*	*	*	*
3	*	*	*	*
4	*	*	*	*
5	*	*	*	0
6	*	*	0	0
7	*	0	0	0
8	0	0	0	1
9	0	0	1	1
10	0	0	1	1
11	0	1	1	2
12	1	1	2	2
13	1	1	2	3
14	1	2	2	3
15	2	2	3	3
16	2	2	3	4
17	2	3	4	4
18	3	3	4	5
19	3	4	4	5
20	3	4	5	5
21	4	4	5	6
22	4	5	5	6
23	4	5	6	7
24	5	5	6	7
25	5	6	7	7

* Indicates that it is not possible to get a value in the critical region.
Source: Zar, Jerrold H., *Biostatistical Analysis,* 4th Edition, © 1999. Reprinted by permission of Pearson Education, Inc., Upper Saddle River, NJ.

Figure 10.9: *Table for sign test*

For $n = 15$ and 0.05 as the level of significance, we get the critical statistic as 3. So, our test statistic is greater than the critical statistic (4>3). So, we have to accept the null hypothesis, which means there is no conclusive evidence to say that the pH level has increased.

Wilcoxon Test

Wilcoxon matched pair sign test is a test for the difference in the medians of two dependent samples. This test also comes under the sign test, as the approach is similar and involves considering the signed samples.

This test involves the following steps:

1. Find the difference between the two observations.
2. Rank the absolute value of the difference in ascending order.
3. If the difference is positive, give a positive sign to the rank. If the difference is negative, give it a negative sign.
4. All the observations with 0 as the difference are removed from the samples.
5. If the rank of two or more observations is the same, we'll calculate the mean of the ranks.
6. Finally, we will calculate T_+ and T_-, which is nothing but the sum of positive and negative ranks.

The test statistic of the Wilcoxon test depends upon the type of test we are performing and the number of observations. Just like one sample test, the three types of tests we can do are as follows:

Two-tailed test:

$$H_0 : M = M_0$$

$$H_A : M \neq M_0$$

Left-tailed test:

$$H_0 : M = M_0$$

$$H_A : M < M_0$$

Right-tailed test:

$$H_0 : M = M_0$$

$$H_A : M > M_0$$

If the observations are ≤ 30 then, for two-tailed tests, the test statistics is the lowest of T_+ and T_-. For left-tailed, it's T_+ and for right-tailed, it's T_-. However, if the observations are higher, the test statistic becomes:

$$z_0 = \frac{T - \frac{n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}}$$

The following example uses the Wilcoxon approach to solve the hypothesis. In this example, a physiotherapist wants to prove the hypothesis that exercising increases flexibility. He trains 12 individuals (randomly selected) and then obtains the data about their flexibility before and after the program. Is his hypothesis true? Take the level of significance as *0.05*. The following table depicts the values that he noted down:

Subject	Before	After	Subject	Before	After
1	18.5	19.25	7	19.75	19.5
2	21.5	21.75	8	15.75	17
3	16.5	16.5	9	18	19.25
4	21	20.25	10	22	19.5
5	20	22.25	11	15	16.5
6	15	16	12	20.5	20

Figure 10.10: *Subject flexibility values*

Since the number of samples is less than 30, we'll be using the smaller sample Wilcoxon test. Let's first define the null and alternate hypotheses:

$$H_0 = Excerciseprogramhasnoeffectonflexibility$$

$$H_A = Exerciseprogramhaseffectonflexibility$$

Let's start the analysis by calculating the difference and ranking them:

Subject	Before	After	Difference	Signed Rank	Final Rank
1	18.5	19.25	-0.75	-4	-4.5
2	21.5	21.75	-0.25	-1	-1.5
4	21	20.25	0.75	4	4.5
5	20	22.25	-2.25	-10	-10
6	15	16	-1	-6	-6
7	19.75	19.5	0.25	1	1.5

Subject	Before	After	Difference	Signed Rank	Final Rank
8	15.75	17	-1.25	-7	-7.5
9	18	19.25	-1.25	-7	-7.5
10	22	19.5	2.5	11	11
11	15	16.5	-1.5	-9	-9
12	20.5	20	0.5	3	3

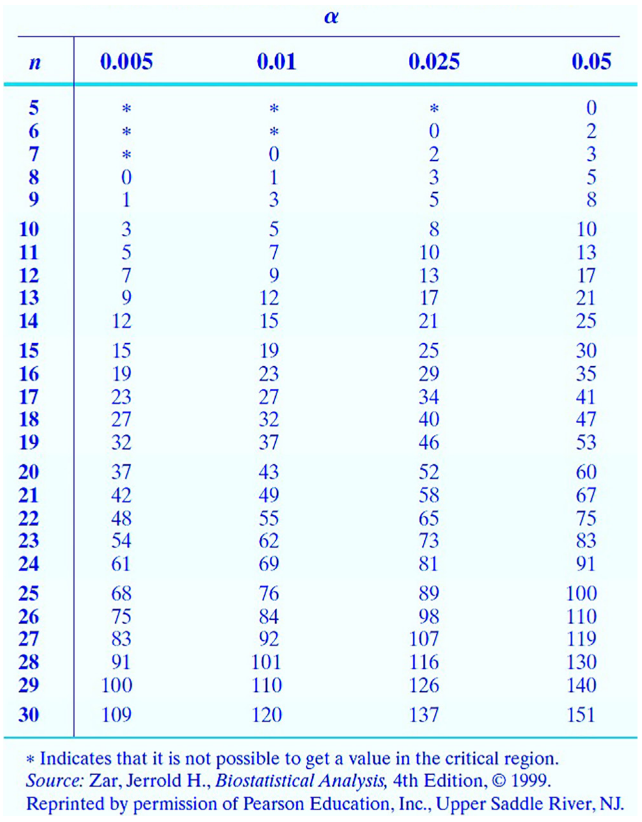
Table 10.1: *Wilcoxon test intermediate values*

Once we have the final ranks, we must calculate the T_+ and T_-. One thing to note here is that we have removed subject 3 since the difference came as 0.

$$T_{+ = 20}$$

$$T_{- = 46}$$

As we know that it is a two-tailed test, we will consider T_+ as out test statistic with the value of *20*. To test this, we will get a critical statistic using the following table:



	α			
n	**0.005**	**0.01**	**0.025**	**0.05**
5	*	*	*	0
6	*	*	0	2
7	*	0	2	3
8	0	1	3	5
9	1	3	5	8
10	3	5	8	10
11	5	7	10	13
12	7	9	13	17
13	9	12	17	21
14	12	15	21	25
15	15	19	25	30
16	19	23	29	35
17	23	27	34	41
18	27	32	40	47
19	32	37	46	53
20	37	43	52	60
21	42	49	58	67
22	48	55	65	75
23	54	62	73	83
24	61	69	81	91
25	68	76	89	100
26	75	84	98	110
27	83	92	107	119
28	91	101	116	130
29	100	110	126	140
30	109	120	137	151

* Indicates that it is not possible to get a value in the critical region.
Source: Zar, Jerrold H., *Biostatistical Analysis*, 4th Edition, © 1999.
Reprinted by permission of Pearson Education, Inc., Upper Saddle River, NJ.

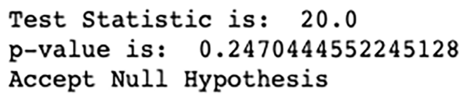
Figure 10.11: *Table for Wilcoxon test*

As our n value is *12* and the level of significance is *0.05*, the critical value will be *17*. Since our *test statistic value is greater than the critical value (20>17)*, we have to accept the null hypothesis. So, we conclude that the exercise program has no effect on flexibility based on this data.

The following is the solution to the same problem using Python:

```
before = [18.5,21.5,21,20,15,19.75,15.75,18,22,15,20.5]
after = [19.25,21.75,20.25,22.25,16,19.5,17,19.25,19.5,16.5,20]

fromscipy.stats import wilcoxon

t, p = wilcoxon(after, before)

print(“Test Statistic is: “, t)
print(“p-value is: “, p)

level_of_significance = 0.05

if p>level_of_significance:
print(“Accept Null Hypothesis”)
else:
print(“Reject Null Hypothesis”)
```

We get the following output:



```
Test Statistic is:  20.0
p-value is:  0.2470444552245128
Accept Null Hypothesis
```

Figure 10.12

Mann Whitney Test

The **Mann Whitney test** is used instead of the Wilcoxon test if the samples are independent of each other. Here are the steps for applying this test:

- Combine both samples into one.
- Rank them is ascending order.

- Find the mean of the ranks, if they are equal (just like in Wilcoxon test).

It follows the same hypothesis tests: two-tailed, left-, and right-tailed, but the test statistic changes. For a sample size of less or equal to *25*, it is as follows:

$$T = S - \frac{n_1(n_1+1)}{2}$$

For samples greater than *25*, we have:

$$Z_0 = \frac{T - \frac{n_1 n_2}{2}}{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

To test the critical value, we use the following conditions for accepting the null hypothesis:

- For two-tailed test: $w_{1-\frac{a}{2}} \leq n_1 n_2 - w_{\frac{a}{2}}$
- For left-tailed test: $T < w_a$
- For right-tailed test: $w_{1-a} \leq n_1\, n_2 - w_a$

Let's understand this with the help of an example. A researcher wants to determine whether the median amount of calcium in the rainwater at his place is different from the median amount of calcium in the rainwater at his wife's place. He collects 22 weeks of data for his place and 20 weeks data for his wife's place, which is depicted in the following table. Is he right in his hypothesis?

Lincoln County Calcium Level (mg/L)				**Clarendon County Calcium Level (mg/L)**			
0.11	0.41	0.19	0.33	0.06	0.12	0.14	0.10
0.09	0.33	0.67	0.20	0.09	0.29	0.14	0.21
0.21	0.20	0.75	0.42	0.14	0.10	0.12	0.16
0.09	0.22	0.19	0.25	0.16	0.41	0.08	0.13
0.07	0.34	0.30	0.47	0.03	0.08	0.09	0.12
0.30	0.46						

Source: National Atmospheric Deposition Program

***Figure 10.13:** Calcium levels of two places*

Let's define our hypothesis:

$$H_0 = Nodifference \in Medianamountofcalcium$$
$$H_A = Difference \in Medianamountofcalcium$$

So, it will be a two-tailed test. As the first step, let's combine these samples and rank them:

Calcium Level	Rank	Calcium Level	Rank	Calcium Level	Rank
0.03	1	0.13	16	0.3	31.5
0.06	2	0.14	18	0.3	31.5
0.07	3	0.14	18	0.33	33.5
0.08	4.5	0.14	18	0.33	33.5
0.08	4.5	0.16	20.5	0.34	35
0.09	7.5	0.16	20.5	0.41	36.5
0.09	7.5	0.19	22.5	0.41	36.5
0.09	7.5	0.19	22.5	0.42	38
0.09	7.5	0.2	24.5	0.46	39
0.1	10.5	0.2	24.5	0.47	40
0.1	10.5	0.21	26.5	0.67	41
0.11	12	0.21	26.5	0.75	42
0.12	14	0.22	28		
0.12	14	0.25	29		
0.12	14	0.29	30		

***Table 10.2:** Mann Whitney test intermediate values*

As we can clearly see, our observations are more than 20, which means it will be a large sample test. So, we have to use the following formula to calculate the test statistic:

$$Z_0 = \frac{T - \frac{n_1 n_2}{2}}{\sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}}$$

Firstly, we will calculate *T*, which is given by:

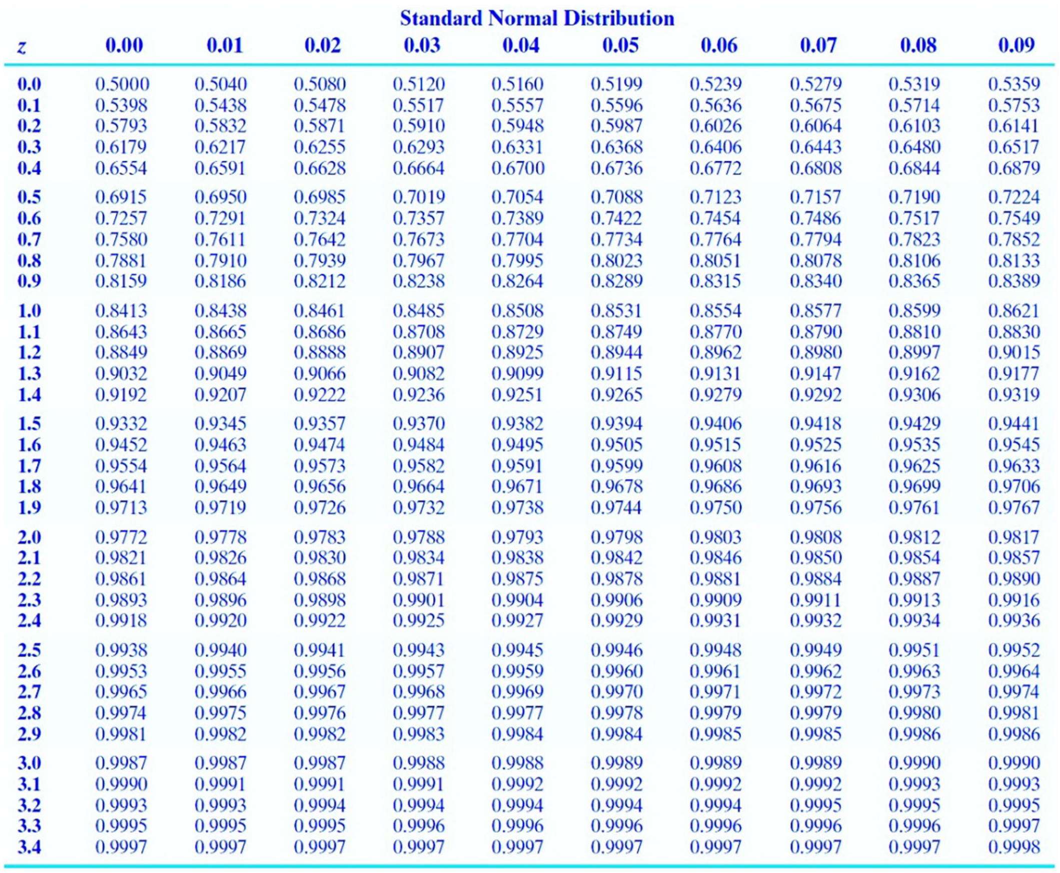
$$T = S - \frac{n_1(n_1 + 1)}{2}$$

Here, *S* is the sum of ranks. Since it is a two-tailed test, we will sum up the ranks for both the classes (Clarendon and Lincoln) and use the higher of the two. So, *S1 = 609* and *S2 = 294*. This means *S = 609*.

$$\therefore T = 609 - \frac{22(22+1)}{2} = 356$$

$$\therefore Z_0 = \frac{356 - \dfrac{22 * 20}{2}}{\sqrt{\dfrac{22 * 20(22+20+1)}{12}}} = \frac{136}{\sqrt{1576.67}} = 3.43$$

So, our test statistic comes to *3.43*. Let's find our critical value now. For this, we'll have to use the following z-table:



Standard Normal Distribution

z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
3.1	0.9990	0.9991	0.9991	0.9991	0.9992	0.9992	0.9992	0.9992	0.9993	0.9993
3.2	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
3.3	0.9995	0.9995	0.9995	0.9996	0.9996	0.9996	0.9996	0.9996	0.9996	0.9997
3.4	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998

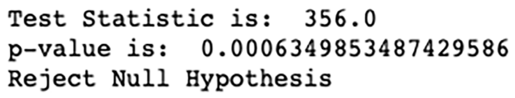
Figure 10.14: *Table for normal distribution*

For our statistic value of *3.43*, the p-value comes to *0.9997*. It covers more than 99% of the area in the curve, and we will have to reject . So, we conclude that there is a difference in the median amount of calcium in the rainwater of the two areas.

Here's the solution to this problem using Python:

```
Lincoln_Calcium = [0.11, 0.41, 0.19, 0.33, 0.09, 0.33, 0.67, 0.20, 0.21,
0.20, 0.75, 0.42, 0.09, 0.22, 0.19, 0.25, 0.07, 0.34, 0.30, 0.47, 0.30,
0.46]

Clarendon_Calcium = [0.06, 0.12, 0.14, 0.10,0.09, 0.29, 0.14, 0.21,0.14,
0.10, 0.12, 0.16,0.16, 0.41, 0.08, 0.13,0.03, 0.08, 0.09, 0.12]

fromscipy.stats import mannwhitneyu

t, p = mannwhitneyu(Lincoln_Calcium, Clarendon_Calcium,alternative='two-
sided')

print("Test Statistic is: ", t)
print("p-value is: ", p)

level_of_significance = 0.05

if p>level_of_significance:
print("Accept Null Hypothesis")
else:
print("Reject Null Hypothesis")
```

Following is the output:



```
Test Statistic is:  356.0
p-value is:  0.0006349853487429586
Reject Null Hypothesis
```

Figure 10.15

Spearman Rank Correlation Test

This non-parametric test is used to find the strength of the relationship between the two samples. There are three tests that contribute to the Spearman test:

- Two-tailed tests where we just need to prove that there is a correlation between the two samples, which can be positive or negative.
- One-tailed test to test whether the samples are positively associated.
- One-tailed test to test whether the samples are negatively associated.

The test statistic for the Spearman test is provided by the following formula:

$$r = 1 - \frac{6\Sigma d_i^2}{n(n^2 - 1)}$$

Let's understand this with the help of an example. A sociologist believes that birth rate decreases as the per capita personal incomes of states increases. She randomly selects eight states in the District of Columbia and obtains the following data:

State	Per Capita Personal Income ($)	Birthrate
Delaware	39,817	14.1
Maryland	48,285	13.9
District of Columbia	66,000	15.1
Virginia	43,874	14.1
West Virginia	32,219	12.1
North Carolina	34,453	14.5
South Carolina	31,799	14.3
Georgia	33,786	15.8
Florida	37,780	13.1

Source: Statistical Abstract of the United States

Figure 10.16: *Per capita income*

Let's define our hypothesis:

$$H_0 = Noassociationbetweenpersonalincome \wedge birthrate$$

$$H_A = personalincome \wedge birthratearenegativelycorrelated$$

First, let's calculate the test statistic:

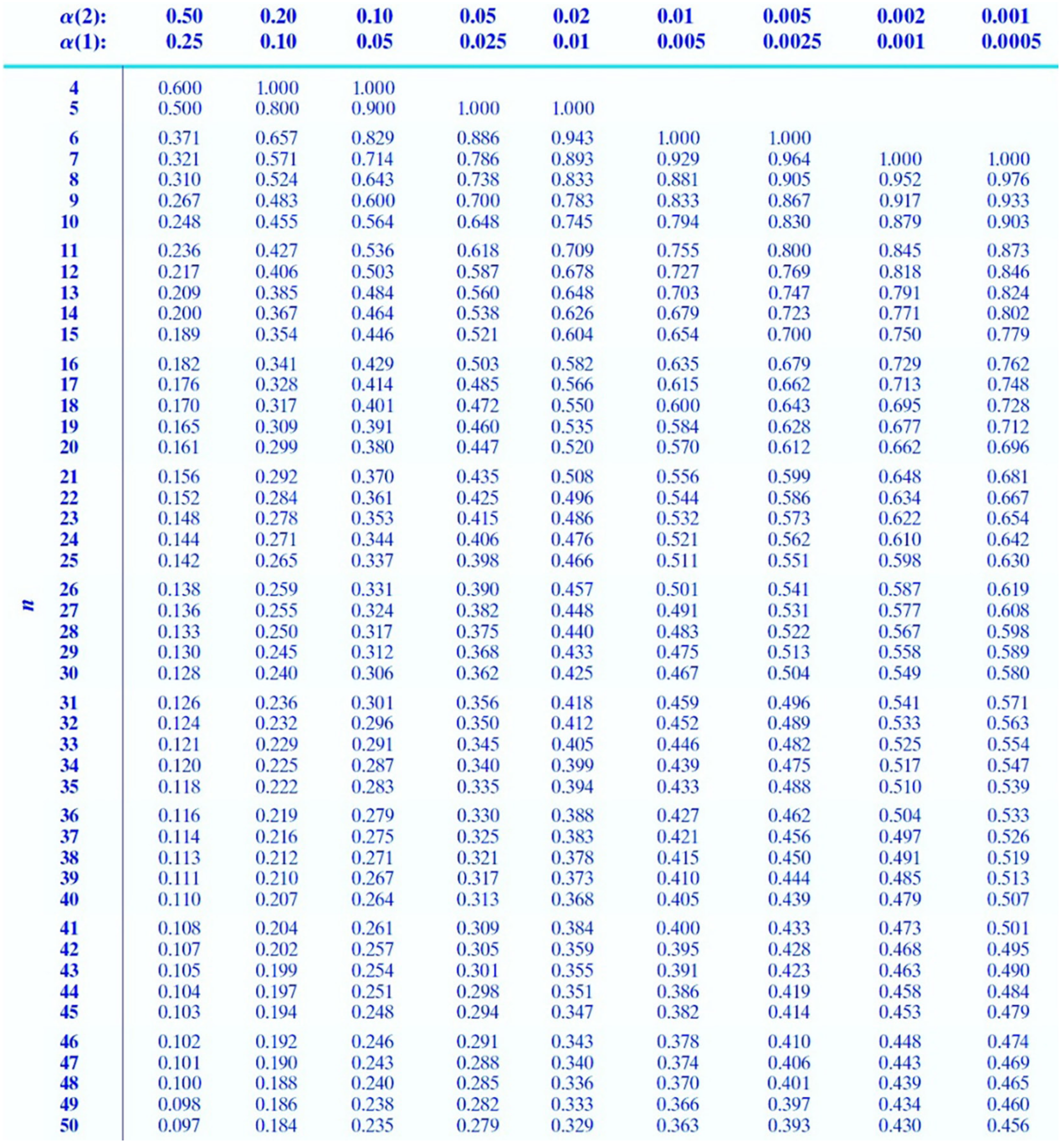
Personal Income	Birth Rate	Rank PI	Rank BR	d	d^2
48,285	13.9	8	3	5	25
39,817	14.1	6	4.5	1.5	2.25
66,000	15.1	9	8	1	1
43,874	14.1	7	4.5	2.5	6.25
32,219	12.1	2	1	1	1
34,453	14.5	4	7	-3	9
31,799	14.3	1	6	-5	25
33,786	15.8	3	9	-6	36
37,780	13.1	5	2	3	9

Table 10.3: *Spearman correlation test intermediate values*

$$\sum d_i^2 = 114.5$$

$$\therefore r = 1 - \frac{6 * 114.5}{9(81-1)} = 1 - \frac{687}{720} = 0.046$$

Now, we'll use the following table to derive a conclusion about the test statistic:



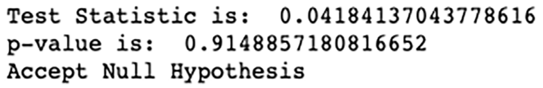
	$\alpha(2)$:	0.50	0.20	0.10	0.05	0.02	0.01	0.005	0.002	0.001
	$\alpha(1)$:	0.25	0.10	0.05	0.025	0.01	0.005	0.0025	0.001	0.0005
	4	0.600	1.000	1.000						
	5	0.500	0.800	0.900	1.000	1.000				
	6	0.371	0.657	0.829	0.886	0.943	1.000	1.000		
	7	0.321	0.571	0.714	0.786	0.893	0.929	0.964	1.000	1.000
	8	0.310	0.524	0.643	0.738	0.833	0.881	0.905	0.952	0.976
	9	0.267	0.483	0.600	0.700	0.783	0.833	0.867	0.917	0.933
	10	0.248	0.455	0.564	0.648	0.745	0.794	0.830	0.879	0.903
	11	0.236	0.427	0.536	0.618	0.709	0.755	0.800	0.845	0.873
	12	0.217	0.406	0.503	0.587	0.678	0.727	0.769	0.818	0.846
	13	0.209	0.385	0.484	0.560	0.648	0.703	0.747	0.791	0.824
	14	0.200	0.367	0.464	0.538	0.626	0.679	0.723	0.771	0.802
	15	0.189	0.354	0.446	0.521	0.604	0.654	0.700	0.750	0.779
	16	0.182	0.341	0.429	0.503	0.582	0.635	0.679	0.729	0.762
	17	0.176	0.328	0.414	0.485	0.566	0.615	0.662	0.713	0.748
	18	0.170	0.317	0.401	0.472	0.550	0.600	0.643	0.695	0.728
	19	0.165	0.309	0.391	0.460	0.535	0.584	0.628	0.677	0.712
	20	0.161	0.299	0.380	0.447	0.520	0.570	0.612	0.662	0.696
	21	0.156	0.292	0.370	0.435	0.508	0.556	0.599	0.648	0.681
	22	0.152	0.284	0.361	0.425	0.496	0.544	0.586	0.634	0.667
	23	0.148	0.278	0.353	0.415	0.486	0.532	0.573	0.622	0.654
	24	0.144	0.271	0.344	0.406	0.476	0.521	0.562	0.610	0.642
	25	0.142	0.265	0.337	0.398	0.466	0.511	0.551	0.598	0.630
	26	0.138	0.259	0.331	0.390	0.457	0.501	0.541	0.587	0.619
n	27	0.136	0.255	0.324	0.382	0.448	0.491	0.531	0.577	0.608
	28	0.133	0.250	0.317	0.375	0.440	0.483	0.522	0.567	0.598
	29	0.130	0.245	0.312	0.368	0.433	0.475	0.513	0.558	0.589
	30	0.128	0.240	0.306	0.362	0.425	0.467	0.504	0.549	0.580
	31	0.126	0.236	0.301	0.356	0.418	0.459	0.496	0.541	0.571
	32	0.124	0.232	0.296	0.350	0.412	0.452	0.489	0.533	0.563
	33	0.121	0.229	0.291	0.345	0.405	0.446	0.482	0.525	0.554
	34	0.120	0.225	0.287	0.340	0.399	0.439	0.475	0.517	0.547
	35	0.118	0.222	0.283	0.335	0.394	0.433	0.488	0.510	0.539
	36	0.116	0.219	0.279	0.330	0.388	0.427	0.462	0.504	0.533
	37	0.114	0.216	0.275	0.325	0.383	0.421	0.456	0.497	0.526
	38	0.113	0.212	0.271	0.321	0.378	0.415	0.450	0.491	0.519
	39	0.111	0.210	0.267	0.317	0.373	0.410	0.444	0.485	0.513
	40	0.110	0.207	0.264	0.313	0.368	0.405	0.439	0.479	0.507
	41	0.108	0.204	0.261	0.309	0.384	0.400	0.433	0.473	0.501
	42	0.107	0.202	0.257	0.305	0.359	0.395	0.428	0.468	0.495
	43	0.105	0.199	0.254	0.301	0.355	0.391	0.423	0.463	0.490
	44	0.104	0.197	0.251	0.298	0.351	0.386	0.419	0.458	0.484
	45	0.103	0.194	0.248	0.294	0.347	0.382	0.414	0.453	0.479
	46	0.102	0.192	0.246	0.291	0.343	0.378	0.410	0.448	0.474
	47	0.101	0.190	0.243	0.288	0.340	0.374	0.406	0.443	0.469
	48	0.100	0.188	0.240	0.285	0.336	0.370	0.401	0.439	0.465
	49	0.098	0.186	0.238	0.282	0.333	0.366	0.397	0.434	0.460
	50	0.097	0.184	0.235	0.279	0.329	0.363	0.393	0.430	0.456

Figure 10.17: *Table for Spearman correlation*

As we are performing a one-tailed test, let’s look at the value for $n=9$ and $a=0.05$. This gives us a value of 0.7. As we find that the test statistic is lesser than this value ($0.046<0.7$), we have to accept the null hypothesis.

So we conclude that there is no negative correlation between birth rate and personal income. Here’s the solution to the same problem using Python:

```
personal_income = [48285,39817,66000,43874,32219,34453,31799,33786,3778
0]

birth_rate = [13.9,14.1,15.1,14.1,12.1,14.5,14.3,15.8,13.1]


fromscipy.stats import spearmanr


t, p = spearmanr(personal_income, birth_rate)


print(“Test Statistic is: “, t)

print(“p-value is: “, p)


level_of_significance = 0.05


if p>level_of_significance:

print(“Accept Null Hypothesis”)

else:

print(“Reject Null Hypothesis”)
```



```
Test Statistic is:  0.04184137043778616
p-value is:  0.9148857180816652
Accept Null Hypothesis
```

Kruskal Wallis test

This is the last non-parametric test we will discuss in this chapter. This test is used to determine whether k independent samples come from the same underlying distribution. So, the test hypothesis can be defined as follows:

$$H_0 = Distibutionsofthepopulationisidentical$$

$$H_A = Distributionofthepopulationsisnotidentical$$

The procedure to apply the Kruskal Wallis test is as follows:

- Rank all the observations in ascending order
- If the values are tied, find the mean of the ranks
- Find the sum of ranks and then use the test statistics

$$TestStatistic = H = \frac{12}{N(n+1)} \Sigma 1/n_i \left[R_i - \frac{n_i(n+1)}{2} \right]^2$$

Where:

R: Sum of Ranks for i^th sample

N: Total number of observations

k: Number of populations being compared

The Kruskal Wallis test is always a right-tailed test. Let's understand it with the help of an example.

An obstetrician wants to determine whether the distribution of births is the same for the periods January to March, April to June, July to September, and October to December. She randomly selected 10 days within each time frame and obtained the results shown in the following table:

January–March	April–June	July–September	October–December
10,456	10,799	11,465	11,261
11,574	11,743	12,475	11,406
12,321	11,657	12,454	11,627
11,661	11,608	12,193	8,706
8,521	10,982	11,244	12,022
11,621	9,184	12,081	11,930
11,321	12,357	11,387	11,054
7,706	12,251	9,055	11,431
10,872	11,916	12,319	12,618
10,837	11,212	12,927	10,824

Source: National Center for Health Statistics

Figure 10.18: *The distribution of births*

The hypothesis can be defined as follows:

$$H_0 : Thedistributionofbirthforthedefinedperiodissame$$

$$H_A : Thedistributionofbirthforthedefinedperiodisnotsame$$

Now, let's find the sum of ranks for each period by computing the following table:

A: Jan to March	B: Apr to Jun	C: July to Sept	D: Oct to Dec	Ra	Rb	Rc	Rd
10,456	10,799	11,465	11,261	6	7	20	15
11,574	11,743	12,475	11,406	21	27	38	18
12,321	11,657	12,454	11,627	35	25	37	24
11,661	11,608	12,193	8,706	26	22	32	3
8,521	10,982	11,244	12,022	2	11	14	30
11,621	9,184	12,081	11,930	23	5	31	29
11,321	12,357	11,387	11,054	16	36	17	12
7,706	12,251	9,055	11,431	1	33	4	19
10,872	11,916	12,319	12,618	10	28	34	39
10,837	11,212	12,927	10,824	9	13	40	8

Table 10.4: *Kruskal Wallis test intermediate values*

R_a, R_b, R_c, R_d represent the ranks for each period. Let's find the sum of each rank now:

$$\sum of\ R_a = 149$$

$$\sum of\ R_b = 207$$

$$\sum of\ R_c = 267$$

$$\sum of\ R_d = 197$$

N is *40*, and each sample contains *10* observations. So, the test statistic becomes:

$$H = \frac{12}{N(N+1)} \sum \frac{1}{n_i} \left[R_i - \frac{n_i(n+1)}{2} \right]^2 = 5.16$$

For the Kruskal Wallis test, we use the Chi-square distribution table, which is given as follows:

Degrees of Freedom	Area to the Right of Critical Value									
	0.995	**0.99**	**0.975**	**0.95**	**0.90**	**0.10**	**0.05**	**0.025**	**0.01**	**0.005**
1	—	—	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	12.017	14.067	16.013	18.475	20.278
8	1.344	1.646	2.180	2.733	3.490	13.362	15.507	17.535	20.090	21.955
9	1.735	2.088	2.700	3.325	4.168	14.684	16.919	19.023	21.666	23.589
10	2.156	2.558	3.247	3.940	4.865	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	17.275	19.675	21.920	24.725	26.757
12	3.074	3.571	4.404	5.226	6.304	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.042	19.812	22.362	24.736	27.688	29.819
14	4.075	4.660	5.629	6.571	7.790	21.064	23.685	26.119	29.141	31.319
15	4.601	5.229	6.262	7.261	8.547	22.307	24.996	27.488	30.578	32.801
16	5.142	5.812	6.908	7.962	9.312	23.542	26.296	28.845	32.000	34.267
17	5.697	6.408	7.564	8.672	10.085	24.769	27.587	30.191	33.409	35.718
18	6.265	7.015	8.231	9.390	10.865	25.989	28.869	31.526	34.805	37.156
19	6.844	7.633	8.907	10.117	11.651	27.204	30.144	32.852	36.191	38.582
20	7.434	8.260	9.591	10.851	12.443	28.412	31.410	34.170	37.566	39.997
21	8.034	8.897	10.283	11.591	13.240	29.615	32.671	35.479	38.932	41.401
22	8.643	9.542	10.982	12.338	14.041	30.813	33.924	36.781	40.289	42.796
23	9.260	10.196	11.689	13.091	14.848	32.007	35.172	38.076	41.638	44.181
24	9.886	10.856	12.401	13.848	15.659	33.196	36.415	39.364	42.980	45.559
25	10.520	11.524	13.120	14.611	16.473	34.382	37.652	40.646	44.314	46.928
26	11.160	12.198	13.844	15.379	17.292	35.563	38.885	41.923	45.642	48.290
27	11.808	12.879	14.573	16.151	18.114	36.741	40.113	43.195	46.963	49.645
28	12.461	13.565	15.308	16.928	18.939	37.916	41.337	44.461	48.278	50.993
29	13.121	14.256	16.047	17.708	19.768	39.087	42.557	45.722	49.588	52.336
30	13.787	14.953	16.791	18.493	20.599	40.256	43.773	46.979	50.892	53.672
40	20.707	22.164	24.433	26.509	29.051	51.805	55.758	59.342	63.691	66.766
50	27.991	29.707	32.357	34.764	37.689	63.167	67.505	71.420	76.154	79.490
60	35.534	37.485	40.482	43.188	46.459	74.397	79.082	83.298	88.379	91.952
70	43.275	45.442	48.758	51.739	55.329	85.527	90.531	95.023	100.425	104.215
80	51.172	53.540	57.153	60.391	64.278	96.578	101.879	106.629	112.329	116.321
90	59.196	61.754	65.647	69.126	73.291	107.565	113.145	118.136	124.116	128.299
100	67.328	70.065	74.222	77.929	82.358	118.498	124.342	129.561	135.807	140.169

Figure 10.19: *Table for Kruskal Wallis test*

We'll find the degrees of freedom represented by *k-1*. In our example,*k* is *4*, so the degrees of freedom will be *3*. Using this with a level of significance of *0.05*, we get a critical value as *7.815*.

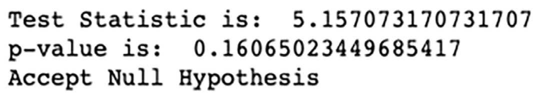
As our test statistic is less than the critical value (*5.16 < 7.815*), we have to accept our null hypothesis. This means that the distribution of birth for all the periods is the same.

The following is the solution to this problem using Python:

```
period_A = [10456,11574,12321,11661,8521,11621,11321,7706,10872,10837]

period_B = [10799,11743,11657,11608,10982,9184,12357,12251,11916,11212]

period_C = [11465,12475,12454,12193,11244,12081,11387,9055,12319,12927]
```

```
period_D = [11261,11406,11627,8706,12022,11930,11054,11431,12618,10824]

fromscipy.stats import kruskal

t, p = kruskal(period_A,period_B,period_C,period_D)

print("Test Statistic is: ", t)
print("p-value is: ", p)

level_of_significance = 0.05

if p>level_of_significance:
print("Accept Null Hypothesis")
else:
print("Reject Null Hypothesis")
```

The following is the output:



```
Test Statistic is:  5.157073170731707
p-value is:  0.16065023449685417
Accept Null Hypothesis
```

Figure 10.20

Conclusion

In this chapter, we saw the different types of non-parametric tests. These tests, though not majorly used in the development scenarios, have a deep application in the research domain. It's not always the case that we know a great deal about the population, so knowing these tests help validate your hypothesis. After knowing the parametric tests, readers must understand non-parametric tests since a scenario may look like it can be solved using the former, but the latter is a better choice.

In the next chapter, we will introduce readers to machine learning and discuss some of the famous algorithms from the domain and their execution in Python.

Chapter 11
Introduction to Machine Learning

We have discussed different concepts of statistics and their applications. Machine learning is one of the biggest applications of statistics and mathematics. A machine tries to mimic human behavior by learning from the information provided. This task of learning is completed using the discussed statistical and mathematical concepts. In this chapter, we will look at a few machine learning concepts and their applications in Python. We will be taking a dataset of Titanic disasters and perform the algorithms that we will look at.

Structure

- Machine learning
    a. Supervised ML
        i. K-Nearest neighbour algorithm
        ii. Naïve Bayes algorithm
        iii. Decision trees
        iv. Ensemble trees
        v. Support vector ,machines

   b. Unsupervised ML
      i. K-Means clustering
      ii. Hierarchical clustering
      iii. Decomposition
- Python application

Objective

The main objective of this chapter is to provide readers a brief overview of what is done once we understand the concept of statistics. Machine learning is the most trending field worldwide, so an introduction to it will set the right path for readers.

Machine Learning

Many get confused between machine learning and automation. Many people believe that machine learning is used if a machine is doing something automatically. However, both the fields are totally different. If we provide a set of rules and make a software system operate on their basis, it comes under the purview of automation. Here, the machine doesn't have an intelligence of its own. Whatever it is doing is based on those rules.

Machine learning, on the other hand, is the application of artificial intelligence, where we don't provide rules and regulations for a system to operate. We provide the machine with a lot of information and expect it to do things with the data based on the provided learning strategy. Once the machine has learned whatever it can, it will make decisions based on the experience with the data and not based on the rules.

In machine learning, the system looks at the data and tries to find hidden patterns in it. The more hidden patterns it can obtain, the better it learns from the data and uses it in a real-world application. Machine learning has various domains, but we will quickly look at two of them in this chapter:

- Supervised machine learning
- Unsupervised machine learning

We will not go through the details of each algorithm in this chapter, but we will give you a gist of the intuitions behind each one.

Supervised Learning

If we already know what has happened in the past and want to see if something similar happens in the future, what will be the reaction? This comes under the domain of supervised machine learning. Whatever happened in the past was dependent on a few factors. If the same factors are present in the future, we need to know what kind of response we will get. The factors are called independent factors (or independent variables), while the outcome is called a dependent factor (or dependent variable).

In machine learning terms, we can say that all the incidents of the past, combined together, contribute to our training dataset. All the supervised learning algorithms learn from the dataset. A training dataset consists of both dependent and independent variables, and the algorithm tries to learn the pattern in the data and correlate it with the outcome. When we get a new combination of independent variables, the trained model tries to check whether the new combination resembles the pattern learned previously, and it predicts the dependent variable based on that.

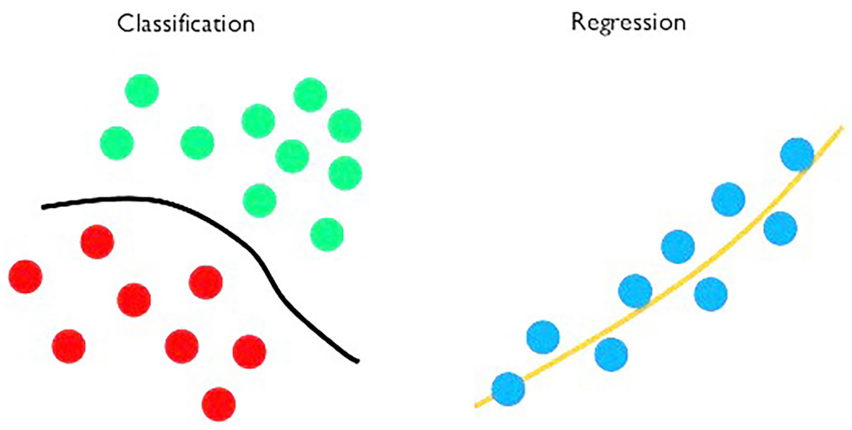


Figure 11.1: *Graphical difference between the regression and classification problems*

Suppose we want to predict whether it will rain tomorrow. We noted the weather for the past 30 days. We looked at the pressure, temperature, humidity, cloud coverage, and so on for each of those 30 days, and then we checked whether it rained. We combined all this information into a training set (we will look at how to make a training set in the next section) and trained a supervised learning model on it. If we provide the pressure, temperature, humidity, and cloud coverage for the next day, our trained model will try to predict whether it will rain. This is a supervised learning model intuition.

We will discuss the following supervised learning algorithms in this chapter:

1. K-Nearest neighbour algorithm

2. Naïve Bayes algorithm
3. Decision trees
4. Ensemble trees
5. Support vector machines

We'll cover all these algorithms in the next few sections.

K-Nearest Neighbour

Take a look at the following illustration:

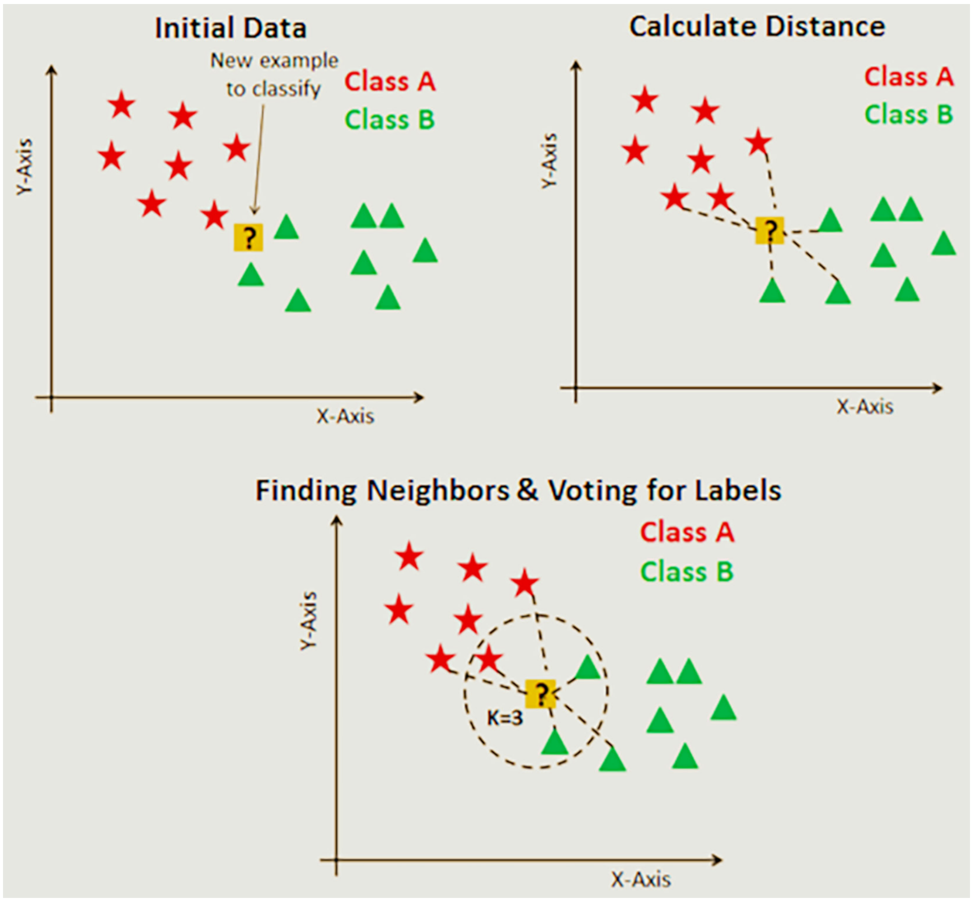


Figure 11.2: *A visual representation of the K-Nearest neighbour algorithm*

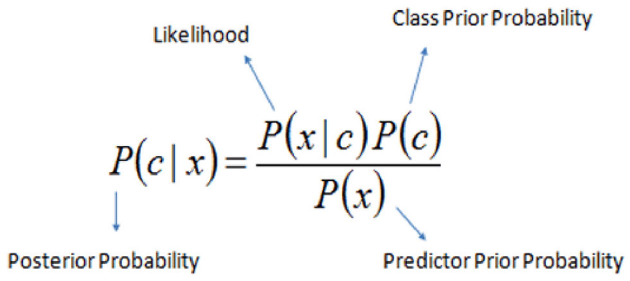
The intuition behind the K-Nearest Neighbour (KNN) algorithm is that two things should fall in the same category if they have the same features. To find this similarity, the KNN algorithm uses the concept of calculation of distance (generally, **Euclidean Distance** (**ED**)). The formula for ED is:

$$E_d = \sqrt{(x \mid \mid 2 - x_1)^2 + (y_2 - y_1)^2}$$

The closest points are considered to be belonging to the same label of the dependent variable. *k* defines the number of observations to look at and compares their distance. If we say *k=20, 20* neighbours will be looked at, and the one with the minimum distance will be the prediction category.

Naïve Bayes Theorem

This is an algorithm used for classification problems. Before understanding this theorem, we must know the difference between classification and regression problems. When predicting categories, this comes under the domain of classification, while it comes under the domain of regression if we want to predict actual numerical values. So, in the previous example, predicting whether it will rain is a classification problem, while predicting the amount of rainfall in '*mm*' is a regression problem. So, the naïve bayes algorithm is used for classification problems:



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 11.3: *Naïve Bayes Formula*

This algorithm uses the bayes formula for the prediction, given as follows:

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)}$$

For example, if it is raining and we want to determine whether it will snow, the formula becomes:

$$P(snow \mid rain) = \frac{P(rain \mid snow) * P(snow)}{P(rain)}$$

Decision trees

Decision trees are one of the most important algorithms in supervised learning approaches. They follow the same approach as humans follow in real life, asking the right questions and performing pros and cons analysis. However, an algorithm uses the concept of entropy to make this analysis because it doesn't know what questions should be asked. Randomly, a decision is made: for example, analyzing whether it will rain if cloud coverage is more v/s less. The output is checked for the number of instances of the occurrence of rain in both the categories: more and less cloud coverage.

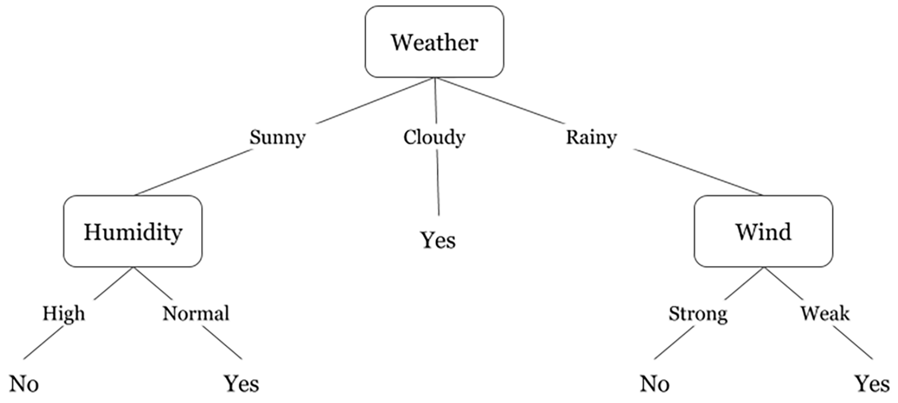


Figure 11.4: *Sample decision tree*

If we get more instances of rain in higher cloud coverage as compared to lower cloud coverage, the split is important. This decision is made using the formula of entropy, which judges the homogeneity and heterogeneity of splits. The formula of entropy is as follows:

$$Entropy(S) = \sum -p_i \log_2 p_i$$

Ensemble trees

Ensemble tree uses the concept of combining multiple decision trees to derive an output. Majorly, they are of three types:

- **Bagging**: This refers to combining the results of multiple decision trees made on random samples taken from the data. Each decision tree gives a prediction, and we consider the mean or a majority vote of all the predictions to come to a final one. The sampling is done using the bootstrapping method, which means sampling with replacement.

- **Boosting**: Here, we combine multiple decision trees, but we transfer the mistakes done by one tree to the next one so that it doesn't repeat the same mistakes instead of finding the mean or majority vote. This process is continued until the last tree is reached, and the output of the last tree is considered as the final one.
- **Random forest**: Random forest is a bagging approach, but we also collect samples of the features (columns) instead of only taking samples from the observations.

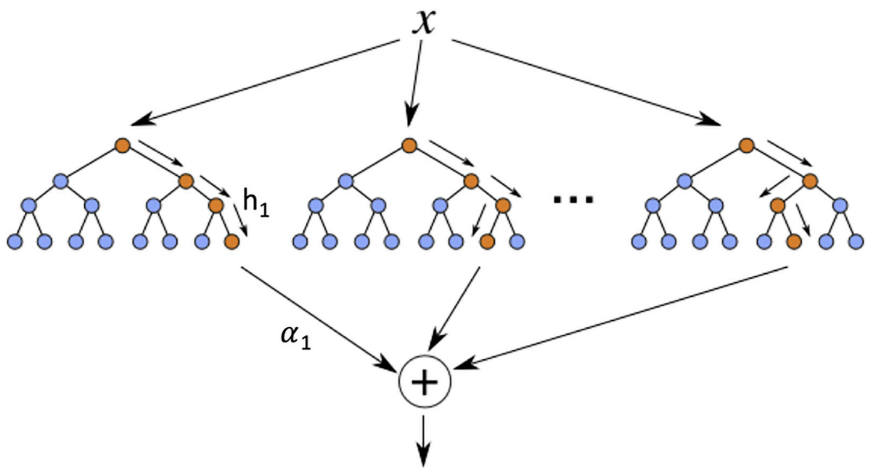


Figure 11.5: *Graphical representation of ensemble trees*

Support Vector Machines

Support vector machines look similar to the concept of linear regression, but we have a plane called **Maximum Margin Hyper Plane** instead of using a line to differentiate between the classes. Also, the approach is not similar to linear regression when it comes to finding the plane. We have to find a plane that maximizes the distance between two points of two classes, which are called support vectors. This helps update two parameters, given by the following formula:

$$y = wX + b$$

Where, w and b are the parameters whose values provide us with the maximum margin. Also, SVMs can be applied in non-linear datasets using the concept of **Kernel-trick**, which is not possible in the case of linear regression.

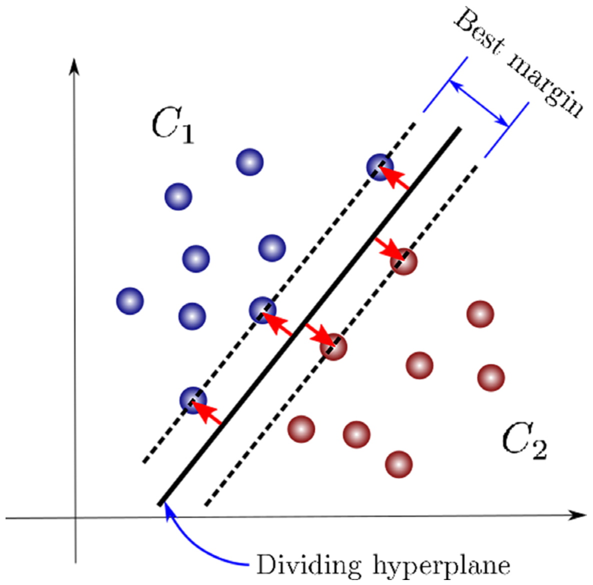


Figure 11.6: *Graphical representation of SVM classification*

Python application

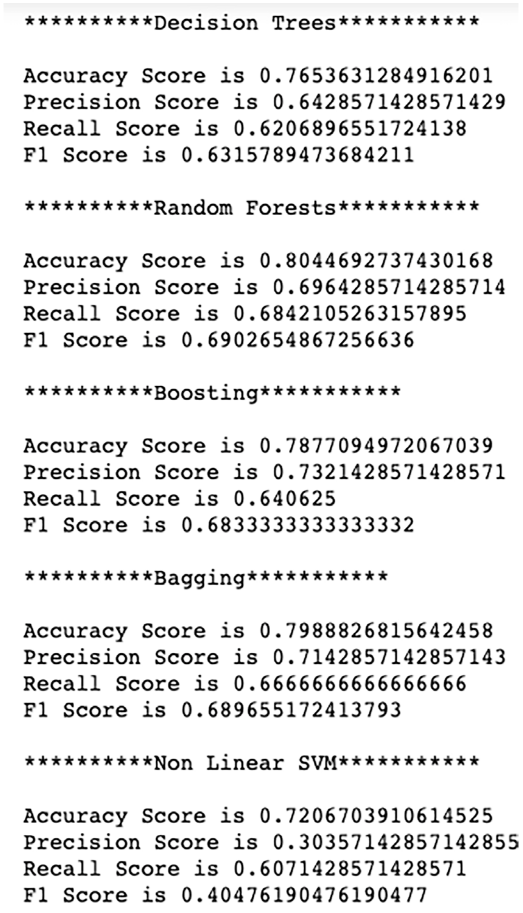
For all the quick algorithms we covered in the previous few sections, let's look at their Python application on the same Titanic dataset that we saw in the previous chapters. We'll be predicting whether a person will survive based on the other columns. Here's the Python code for this problem:

```
import pandas as pd

import numpy as np


titanic_data = pd.read_csv("titanic_train.csv")

titanic_data.head()


# Removing unnecessary columns

titanic_data.drop(["PassengerId", "Name", "Ticket", "Cabin"], axis=1,
inplace=True)


# Making two dataframes. One having categories, other having numbers

tit_cat = titanic_data.select_dtypes([object])

tit_num = titanic_data.select_dtypes([np.number])
```

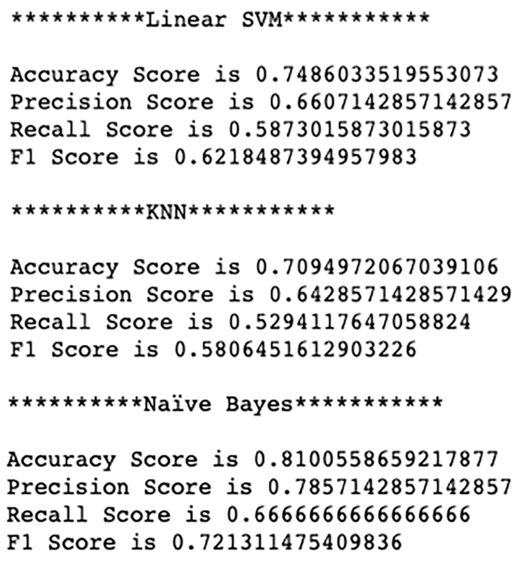
```
tit_cat.head()
tit_num.head()

# Checking Null Values
tit_cat.isna().sum()
tit_num.isna().sum()

# Filling Null Values
tit_cat.Embarked.fillna(tit_cat.Embarked.value_counts().idxmax(),
inplace=True)
tit_num.Age.fillna(tit_num.Age.mean(), inplace=True)

# Converting Categorical Columns to Label Numbers
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
tit_cat = tit_cat.apply(le.fit_transform)

# Combining categorical and numerical dataframes
titanic_processed = pd.concat([tit_cat, tit_num], axis=1)

# Getting dependent and independent features
X = titanic_processed.drop([“Survived”], axis=1)
Y = titanic_processed[[‘Survived’]]

# Getting Train and Test Set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2)
len(X_train), len(X_test), len(Y_train), len(Y_test)

# Initializing all the algorithms
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
BaggingClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

dt = DecisionTreeClassifier()
rf = RandomForestClassifier()
adb = AdaBoostClassifier()
bg = BaggingClassifier()
nlsvm = SVC()
lsvm = LinearSVC()
knn = KNeighborsClassifier()
gnb = GaussianNB()

# Fitting the models
dt.fit(X_train, Y_train)
rf.fit(X_train, Y_train)
adb.fit(X_train, Y_train)
bg.fit(X_train, Y_train)
nlsvm.fit(X_train, Y_train)
lsvm.fit(X_train, Y_train)
knn.fit(X_train, Y_train)
gnb.fit(X_train, Y_train)

# Prediction of Test Set
pred1 = dt.predict(X_test)
pred2= rf.predict(X_test)
pred3 = adb.predict(X_test)
pred4 = bg.predict(X_test)
pred5 = nlsvm.predict(X_test)
pred6 = lsvm.predict(X_test)
pred7 = knn.predict(X_test)
pred8 = gnb.predict(X_test)
```

```
# Checking the Model Performance
from sklearn.metrics import accuracy_score, precision_score, recall_
score, f1_score
algo = ['Decision Trees', 'Random Forests', 'Boosting', 'Bagging', 'Non
Linear SVM', 'Linear SVM', 'KNN', 'Naïve Bayes']
for p,i in enumerate([pred1, pred2, pred3, pred4, pred5, pred6, pred7,
pred8]):
    print(f"\n**********{algo[p]}***********\n")
    print("Accuracy Score is", accuracy_score(i,Y_test))
    print("Precision Score is", precision_score(i,Y_test))
    print("Recall Score is", recall_score(i,Y_test))
    print("F1 Score is", f1_score(i,Y_test))
```

The output for this code is as follows:



```
**********Decision Trees***********

Accuracy Score is 0.7653631284916201
Precision Score is 0.6428571428571429
Recall Score is 0.6206896551724138
F1 Score is 0.6315789473684211

**********Random Forests***********

Accuracy Score is 0.8044692737430168
Precision Score is 0.6964285714285714
Recall Score is 0.6842105263157895
F1 Score is 0.6902654867256636

**********Boosting***********

Accuracy Score is 0.7877094972067039
Precision Score is 0.7321428571428571
Recall Score is 0.640625
F1 Score is 0.6833333333333332

**********Bagging***********

Accuracy Score is 0.7988826815642458
Precision Score is 0.7142857142857143
Recall Score is 0.6666666666666666
F1 Score is 0.689655172413793

**********Non Linear SVM***********

Accuracy Score is 0.7206703910614525
Precision Score is 0.30357142857142855
Recall Score is 0.6071428571428571
F1 Score is 0.40476190476190477
```



```
**********Linear SVM***********

Accuracy Score is 0.7486033519553073
Precision Score is 0.6607142857142857
Recall Score is 0.5873015873015873
F1 Score is 0.6218487394957983

**********KNN***********

Accuracy Score is 0.7094972067039106
Precision Score is 0.6428571428571429
Recall Score is 0.5294117647058824
F1 Score is 0.5806451612903226

**********Naïve Bayes***********

Accuracy Score is 0.8100558659217877
Precision Score is 0.7857142857142857
Recall Score is 0.6666666666666666
F1 Score is 0.721311475409836
```

Figure 11.7: *Output of Supervised Learning Code*

Unsupervised Learning

As the name suggests, **unsupervised** refers to the lack of any means to guide the model. In contrast to the supervised learning approaches, we don't have the target variable using which we can find the hidden patterns. So, using unsupervised learning, we look at and group the data based on the similarities. Let's consider an example.

Suppose we have people from different countries attending a conference, and you want to tailor your branding strategies based on an individual's native country. However, you fogot to ask their country in the form that they filled out. You can read all the forms using the supervised learning approach and group the ones that show almost the same features. This strategy may help you group together people from the same countries by examining the way they filled out the form. This approach comes under the supervised learning domain. Here are the algorithms we will cover in this section:

- K-Means clustering
- Hierarchical clustering
- Decomposition

K-Means Clustering

This approach is similar to the K-Nearest Neighbour algorithm that we went through in the supervised learning section. Here, we try to find out similar groups by looking at the distances between them. The closer the points, the higher are the chances of them belonging to a particular group.

In this algorithm, we use the method of centroid and Euclidean distance. We tell the model about the number of groups we need to find beforehand. Based on this information, the model defines the same number of centroids randomly. Then, the initial clusters are created once the Euclidean distances between the points and the centroid are found. Then, we again find the centroid and repeat the process till we get the final groups.

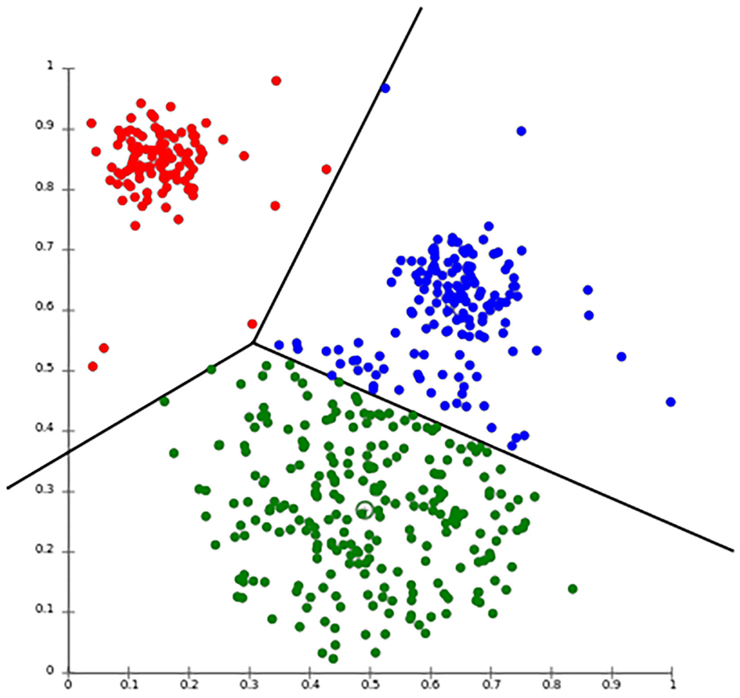


Figure 11.8: Graphical representation of K-Means clustering

Hierarchical Clustering

This algorithm has the same usage as K-Means clustering, but the approach is different. We start by determining the two closest points and group them as one. Then, we proceed by finding the next set of two closest groups and group them as one. We repeat this until we get the desired number of clusters.

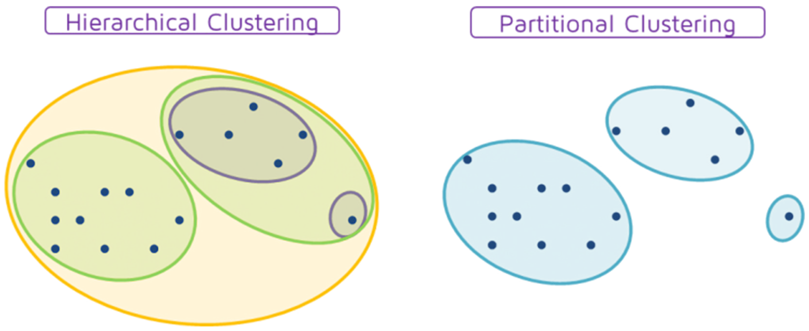


***Figure 11.9:** Graphical representation of hierarchical clustering*

Principal Component Analysis

This algorithm is used to reduce the number of features we have. Suppose we want to predict the stock price. They depend on an infinite number of factors. Suppose we can gather 1,000 features (columns) on which the price depends. Making a model with 1,000 features will be computationally very challenging, and it will also increase the variance in the data. So, the results will not be satisfactory, and that's where PCA comes into the picture.

Principal Component Analysis (**PCA**) is the algorithm using which we try to find how much each column explains the target variable. We make the first component, which considers all the columns that explain the target variable most. Then, we make the second component, which contains the second-highest amount of information about the target variable. We repeat this until we reach the manually-decided threshold.

In the preceding example, we expect and assume that 1,000 features will explain the target variable completely. However, suppose we can get 50 principal components that can explain around 95% of the target variable variance. Then, it is always better to use 50 columns as compared to 1,000. PCA achieves this using the concepts of **Eigen Values** and **Eigen Vectors**. As we are only aiming to introduce ML algorithms, exploring the mathematics is outside the scope of this book.

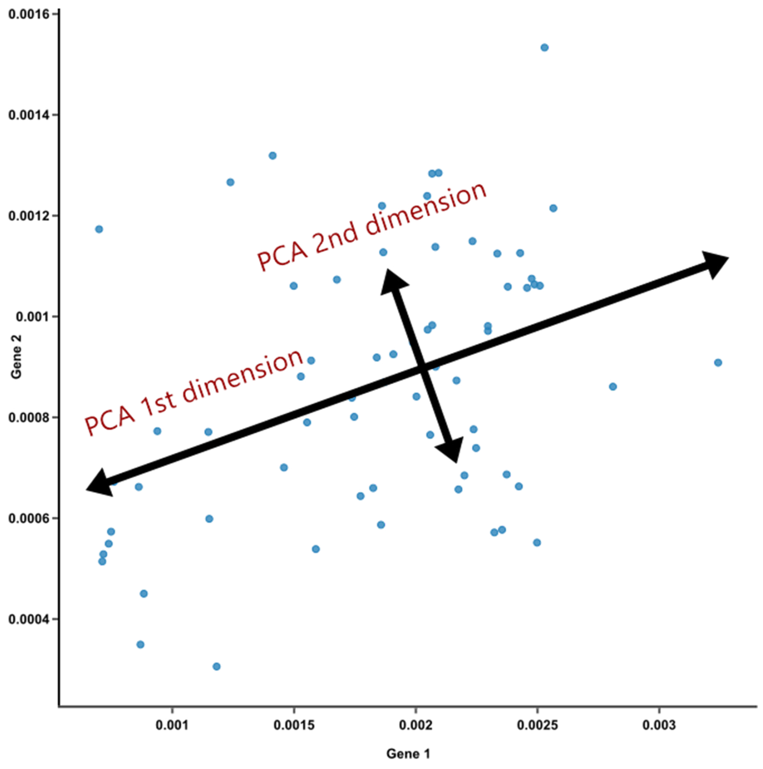


Figure 11.10: *Graphical representation of PCA*

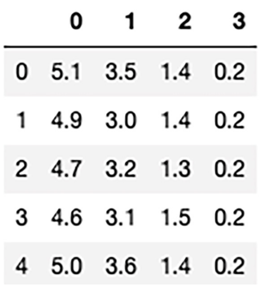
Let's look at the Python applications of the unsupervised learning algorithms that we discussed:

```
import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import numpy as np

from sklearn.metrics import silhouette_score

import numpy as np

from scipy.spatial.distance import cdist, pdist

from matplotlib import pyplot as plt
```

```
# Reading iris data and dropping target column
data = pd.read_csv('Data/iris.data', delimiter=',', header=None)
data = data.drop([4], axis=1)
display(data.head())

# Defining train and test set
data_train = data.iloc[0:100, :]
data_test = data.iloc[100:150, :]

# Let's look at the range of k values
k_range = range(1,14)

# fit k-means for different values of k
k_means_var = [KMeans(n_clusters=k).fit(data_train) for k in k_range]

# find some metrics to decide the best k
# Step1: Calculate the Centroids
centroids = [X.cluster_centers_ for X in k_means_var]
# Step2: Find Euclidean Distance
k_euclid = [cdist(data_train, cent, 'euclidean') for cent in centroids]
# Step3: Find Minimum Distance
dist = [np.min(ke,axis=1) for ke in k_euclid]
# Step4: Calculate the Sum of Squares
wcss = [sum(d**2) for d in dist]
tss = sum(pdist(data_train)**2)/data_train.shape[0]
bss = tss - wcss

# Displaying elbow curve to find the best value
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(k_range, bss/tss*100, ‘b*-’)
ax.set_ylim((0,100))
plt.grid(True)
plt.xlabel(‘n_clusters’)
plt.ylabel(‘Percentage of variance explained’)
plt.title(‘Variance Explained vs. k’)
plt.show()

# Best value comes out as 2, therefore apply k-means for 2 clusters
k_means = KMeans(n_clusters=2)
k_means.fit(data_train)

# Let’s check the cluster for new data
k_means.predict(data_test)

# Check the cluster efficiency using Silhouette Score
labels = k_means.labels_
print(“The Silhouette Score is: “, silhouette_score(data_train, labels,
metric=’euclidean’))
```

Following is the output:



	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Figure 11.11: *Top 5 rows of our Data*

In the following screenshot, **Elbow** can be seen at point 2, which means the best number of clusters is 2:

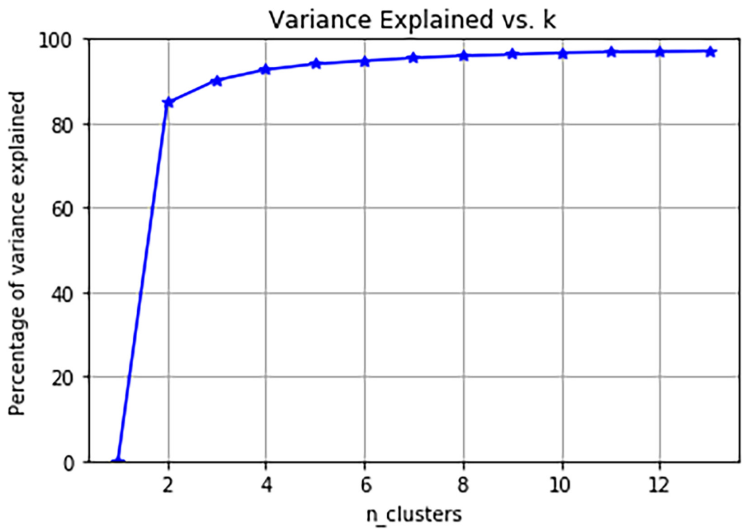
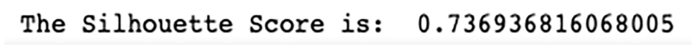


Figure 11.12

So, the higher the **Silhouette Score**, the better the cluster:



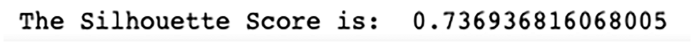
```
The Silhouette Score is:  0.736936816068005
```

Figure 11.13

```
# Let us now apply Hierarchical Clustering and check the Silhouette

from sklearn.cluster import AgglomerativeClustering


# Initializing Hierarchical Clustering Algorithm

clustering = AgglomerativeClustering(linkage=”ward”, n_clusters=2)


# Fitting on Train Set

clustering.fit(data_train)


# Predict the clusters

y = clustering.fit_predict(data_train)
```

```
print("The Silhouette Score is: ", silhouette_score(data_train, y, metric='euclidean'))
```

The following is the output:



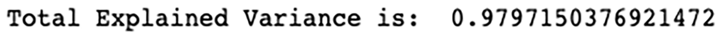
```
The Silhouette Score is:  0.736936816068005
```

Figure 11.14

In the preceding code block, we have tried to group the data into two clusters using both the K-Means clustering and hierarchical clustering methods. The silhouette score in both the cases is same, as we are using Euclidean distance as the distance measure. Now, let's look at the application of PCA:

```
from sklearn.decomposition import PCA


# Define the Components Required

pca = PCA(n_components=2)


# Apply them on Data

principalComponents = pca.fit_transform(data_train)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])


finalDf = pd.concat([principalDf, pd.read_csv('Data/iris.data', delimiter=',', header=None)[[4]]], axis = 1)


# Visualize the old and new data frames

pd.read_csv('Data/iris.data', delimiter=',', header=None).head()

finalDf.head()


# Check the total Variance Explained

print("Total Explained Variance is: ", np.sum(pca.explained_variance_ratio_))
```

Here's the screenshot:



```
Total Explained Variance is:  0.9797150376921472
```

Figure 11.15

The preceding code shows that we explained 98% of the variance present in the data using only two components instead of four columns. That's the power of PCA. The only problem is that we lose the explaining ability of the model, as we don't know the extent to which each column is contributing.

Conclusion

In this chapter, we saw some of the ML algorithms that data scientists come across in their daily life. We wrapped up the book by talking about some famous Python libraries and machine learning, as statistics may feel incomplete by itself. Hypothesis testing may seem to find lesser usage in practical machine learning, but it is extremely important, as the data we get to make predictions represents a sample. So, hypothesis tests are important for comparing samples with the population.

Now that we're done with all the chapters, readers can move on to exploring the exhaustive field of machine learning and understanding the advanced applications in detail.

Index