Data Visualisation in Python

Module 1 : Introduction to Data visualisation with Matplotlib

Introduction to Matplotlib:-

Data visualisation is an important skill to possess for anyone trying to extract and communicate insights from data. Great business narratives and presentations often stem from brilliant visualisations that convey the key ideas in a concise and aesthetic manner. In the field of machine learning, visualisation plays a key role throughout the entire process of analysis - **to obtain relationships, observe trends and portray the final results as well**. Therefore, it is imperative that you learn and master this tool which will aid you throughout this program.

This session will help you learn how to visualise data in Python using the **Matplotlib** library.

In the previous module, you learnt how to handle numerical data using the NumPy library. You could load data in the form of arrays and then perform operations on the same. In this session, we will try to visualise the arrays using another library in Python, namely, **Matplotlib**.

Data visualisation is a crucial step in the process of data analysis; in the upcoming video, let's hear it out from Behzad how visualisation could play the most crucial role for your data toolkit.

This session will cover the following topics:

- Creating and plotting graphs
- Different chart types
- Modification of charts for better understanding and presentation

The Necessity of Data Visualisation:-

"There are three kinds of lies: lies, damned lies, and statistics." - Mark Twain

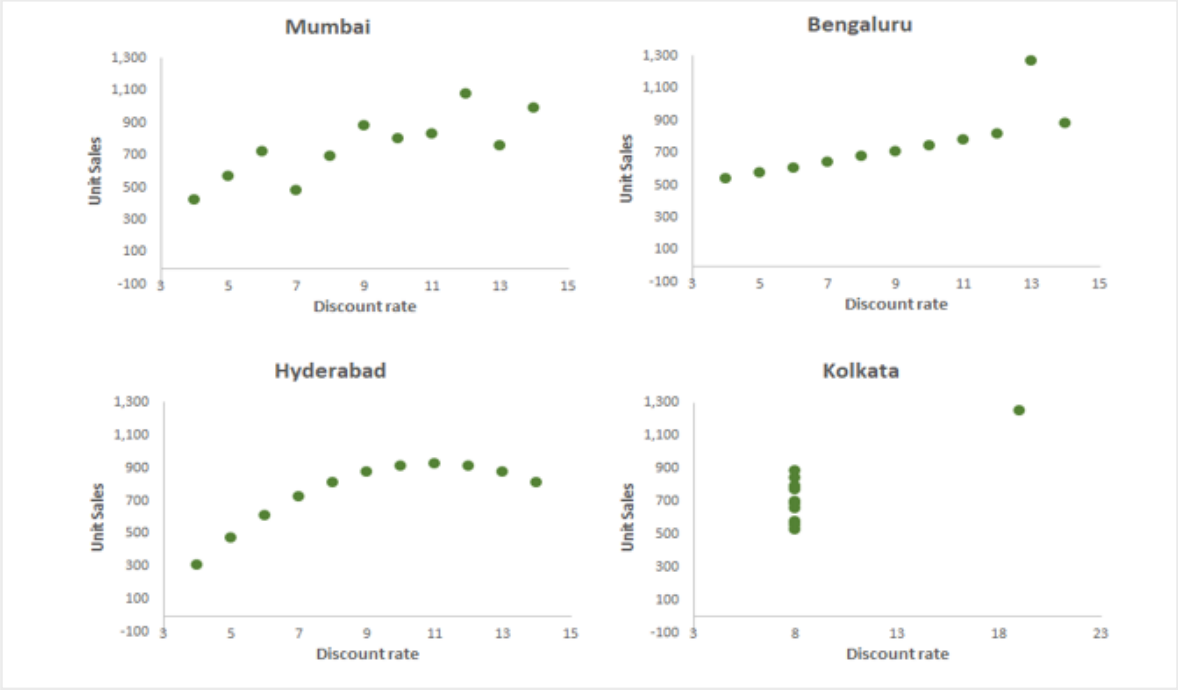
Before we get to learn about the various nuances involved in data visualisation, it is essential to appreciate why it is so important to 'look' at the data from the perspective of plots and graphs. To begin with, it is difficult for the human eye to decipher patterns from raw numbers only. Sometimes, even the statistical information summarised from the data may mislead you to wrong conclusions. Therefore, you should visualise the data often to understand how different features are behaving. Let's listen to Rahim as he demonstrates this idea using a brilliant example.

As explained in the video above, it is very easy to be deceived by the numbers and summary statistics. In the example that you saw, each of the branches had similar average sales and discount rates, and the corresponding standard deviations were similar as well, as shown in the table below.

	Mumbai		Bengaluru		Hyderabad		Kolkata	
Month	Discount	Sales	Discount	Sales	Discount	Sales	Discount	Sales
January	10	804	10	914	10	746	8	658
February	8	695	8	814	8	677	8	576
March	13	758	13	874	13	1,274	8	771
April	9	881	9	877	9	711	8	884
May	11	833	11	926	11	781	8	847
June	14	996	14	810	14	884	8	704
July	6	724	6	613	6	608	8	525
August	4	426	4	310	4	539	19	1,250
September	12	1,084	12	913	12	815	8	556
October	7	482	7	726	7	642	8	791
November	5	568	5	474	5	574	8	689
Average	**9**	**750.1**	**9**	**750.1**	**9**	**750.1**	**9**	**750.1**
Std. Dev.	**3.16**	**193.7**	**3.16**	**193.7**	**3.16**	**193.7**	**3.16**	**193.7**

Table 1

However, the patterns in the underlying data and the difference became apparent when visualised through appropriate plots.



Note : Bengaluru and Hyderabad graphs have been interchanged (bengaluru graph represents hyderabad and vice versa)

Each of the branches had actually employed a different strategy to calculate its discount rate, and the sales numbers were also quite different across all of them. It is difficult to draw this type of insight and understand the difference between each of the branches using raw numbers alone; therefore, you should utilise an appropriate visualisation technique to 'look' at the data. In the next segment, let's focus on a few examples of data visualisation.

References:

The discount and sales example that you saw above is actually a modified version of a popular dataset called the Anscombe's Quartet. As explained in the linked article (Anscombe's Quartet), the statistician Frances Anscombe constructed this example to counter the notion that **"numerical calculations are exact, but graphs are rough."**

Visualisations - Some Examples:-

In this segment, you'll be looking at some real-life examples of how data visualisation can help you derive insights out of data very quickly.

You saw how data visualisation can improve data density and improve the amount of information being conveyed. Just imagine how difficult it would be to interpret a spreadsheet with the score of each inning of each batsman being recorded along with his strike rate. In this case, data visualisation can help you figure out many insights just by looking at the plot. You can see the

visualisation below.

You can also play around with the visualisation here.

Now, let's see another example of where data visualisation can help you derive industry-relevant insights.

You saw how a treemap diagram showed how multiple companies and sectors react to the budget. You can find the interactive graph here.

An important point to remember is that visualisations should be accompanied by a voice or text narrative if possible - it improves the experience of the user drastically.

Next, let's see how visualisation can help in visual exploratory analytics. Here, you will see how data visualisation helped in understanding the connections between different software and clustering them together based on common features. You can find the visual here.

Facts and Dimensions:-

Graphics and visuals, when used intelligently and innovatively, can convey a lot more than what raw data alone can. Matplotlib serves the purpose of providing multiple functions to build graphs from the data stored in your lists, arrays, etc. So, let’s start with the first lecture on Matplotlib.

Before we start discussing different types of plots, you need to learn about the elements that help us create charts and plots effectively. There are two types of data, which are as follows:

- Facts
- Dimensions

Facts and dimensions are different types of variables that help you interpret data better. Facts are numerical data, and dimensions are metadata. Metadata explains the additional information associated with the factual variable. Both facts and dimensions are equally important for generating actionable insights from a given data set. For example, in a data set about the height of students in a class, the height of the students would be a fact variable, whereas the gender of the students would be a dimensional variable. You can use dimensions to slice data for easier analysis. In this case, the distribution of height based on the gender of a student can be studied.

In the next segment, you will start building graphical plots using Python. The first visualisation that you will try to create is a Bar Graph.

Bar Graph:-

Plots are used to convey different ideas. For example, you can use certain plots to visualise the spread of data across two variables and other plots to gauge the frequency of a label. Depending on the objective of your visualisation task, you can choose an appropriate plot. As part of this session, you will learn how to select an appropriate plot. You can download the Jupyter Notebook attached below. The same notebook will be used in the demonstrations throughout the session.

In this segment, you will learn how to create the first plot: Bar Graph. In the following video, Behzad will explain the process of creating a bar graph.

As you saw in the video above, the subpackage pyplot is used to build plots and graphs throughout this session. To load the subpackage, you need to run the following command:

import matplotlib.pyplot as plt

To recap, Matplotlib allows you to use a simple and intuitive workflow to create plots. The important Matplotlib commands used in the video above are as follows:

- plt.bar(x_component, y_component): Used to draw a bar graph
- plt.show(): Explicit command required to display the plot object

A bar graph is helpful when you need to visualise a numeric feature (fact) across multiple categories. In the example covered in the video, you plotted the sales amount (numeric feature) under three different product categories. Using the bar graph, you could easily distinguish between the performance of these categories.

Let's watch the next video in which Behzad will demonstrate how to add elements to our graph to make it more easily understandable.

In the video above, you learnt the different ways in which you can modify your chart to make it more understandable. You can use the following code to add a title and labels to your graph:

- plt.xlabel(), plt.ylabel(): Specify labels for the x and y axes
- plt.title(): Add a title to the plot object.

You can also try to make the charts more appealing by using different attributes such as font size and colour. Adding labels and a title to your plot helps the audience interpret the graphs easily and also relays the required information to the viewer.

You can use the attributes of plt.bar() to make the desired changes to the bars of a graph. For example, you can use the following code to

change the values and ticks on the x and y axes of a graph:
plt.yticks(tick_values, tick_labels)

After making all the changes demonstrated in the video, your graph will look like the one given below.

Bar Graph

Now that you have understood the basics of a bar plot, answer the following questions.

In this segment, you learnt how to build a bar graph and add or modify the required elements within it. In the next segment, you will learn about another visualisation: Scatter Plot.

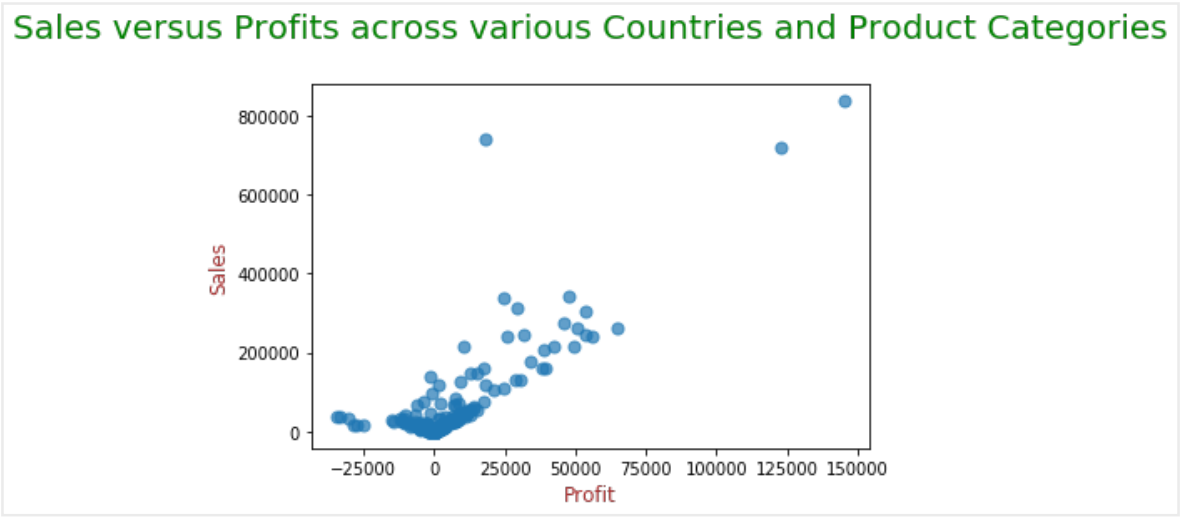
Scatter Plot:-

Scatter plot, as the name suggests, displays **how the variables are spread across the range considered**. It can be used to **identify a relationship or pattern between two quantitative variables and the presence of outliers within them**.

Let's watch the next video to understand how a scatter plot can be useful when you are dealing with two quantitative variables

You can use the following command to build a scatter plot:
plt.scatter(x_axis, y_axis)

Using this command, the data points will spread across the graph corresponding to the values on the y and x axes. The plot that was developed in the demonstration is given below.



Take a look at the rightmost point in the plot. It represents a product that made a profit of 1,50,000 units when the sales generated were 8,00,000 units. Similarly, all the points in the plot represent a product and its profit and sales values. Just by looking at the chart, can you find the products that are more lucrative to trade than others? Yes, a lucrative product has high profit value with preferably low sales value, that is the points to the right side in the plot preferably also towards the bottom.

Now that you have learnt how to read a scatter plot, let's proceed to more complex features of a scatter plot. Matplotlib offers a feature that allows you to incorporate a categorical distinction between the points plotted on a scatter plot. Using this feature, you can colour-code the points based on the category and distinguish them accordingly. Let's watch the next video to learn how to colour-code points on a scatter plot.

You can run the scatter function with the following attributes to specify the colours and labels of the categories in a data set:

```
plt.scatter(x_axis, y_axis, c = color, label = labels)
```

Here, all the information (x_axis, y_axis, colour, labels) need to be provided in the form of a list or an array. You can use this command to assign colours to categories and distinguish them accordingly.

Another feature of a scatter plot allows you to use labels to further distinguish points over another dimension variable. Suppose you have the array 'country', which indicates the country where the sales were generated. Now you want to highlight the points belonging to a particular country in the previous scatter plot.

You can use the following command to add a note (annotate) with a point in the scatter plot:

```
plt.scatter(profit[product_category == "Technology"], sales[product_category
== "Technology"],
            c= 'Green', alpha= 0.7, s = 150, label="Technology" )

plt.scatter(profit[product_category == "Office Supplies"],
sales[product_category == "Office Supplies"],
            c= 'Yellow', alpha= 0.7, s = 100, label="Office Supplies" )

plt.scatter(profit[product_category == "Furniture"], sales[product_category ==
"Furniture"],
            c= 'Cyan', alpha= 0.7, s = 50, label="Furniture" )

for xy in zip (profit[country == "India"], sales[country == "India"]):
        plt.annotate(s = "India", xy = xy)

# Adding and formatting title
plt.title("Sales versus Profits across various Countries and Product
Categories\n", fontdict={'fontsize': 20, 'fontweight' : 5, 'color' : 'Green'})

# Labeling Axes
plt.xlabel("Profit", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})

plt.legend()

plt.show()
```

After using the command to add a note, your scatter plot will look like the one given below.

Scatter Plot

As you can see in the figure above, the products that were traded in India are marked. This is how the annotate statement that was added to a point in the scatter plot helps you distinguish the data points.

In this segment, you learnt how a scatter plot helps you visualise two numeric variables. Attempt the following questions to cement the concepts that were covered in this segment.

[Note: Please use the updated matplotlib documentation here]

Matplotlib also offers multiple features to make these plots as descriptive as possible using the different dimension variables associated with the plot.bar() method.

In the next segment, you will learn about another set of graphs: Line Graph and Histogram. A line graph is used to visualise trends, and a histogram is used when you want to study the frequency distribution of a numeric variable.

Line Graph and Histogram:-

In this segment, you will learn about two new visualisation charts, which are as follows:

- Line graph
- Histogram

Line Graph

A **line graph** is used to present continuous time-dependent data. It accurately depicts the trend of a variable over a specified time period. Let's watch the next video to learn how to plot a line chart using the Matplotlib library.

You can use the following command to plot a line graph:
plt.plot(x_axis, y_axis)

Remember to be careful while using the plt.plot function. This function also helps you create a scatter plot when you tweak the syntax and specify the markers. Try to run the following code to understand the difference between the outputs of the function:

y = np.random.randint(**1**,**100**, **50**)
plt.plot(y, 'ro') # 'ro' represents color (r) and marker (o)
(if you are getting an error, check the quotation marks.)

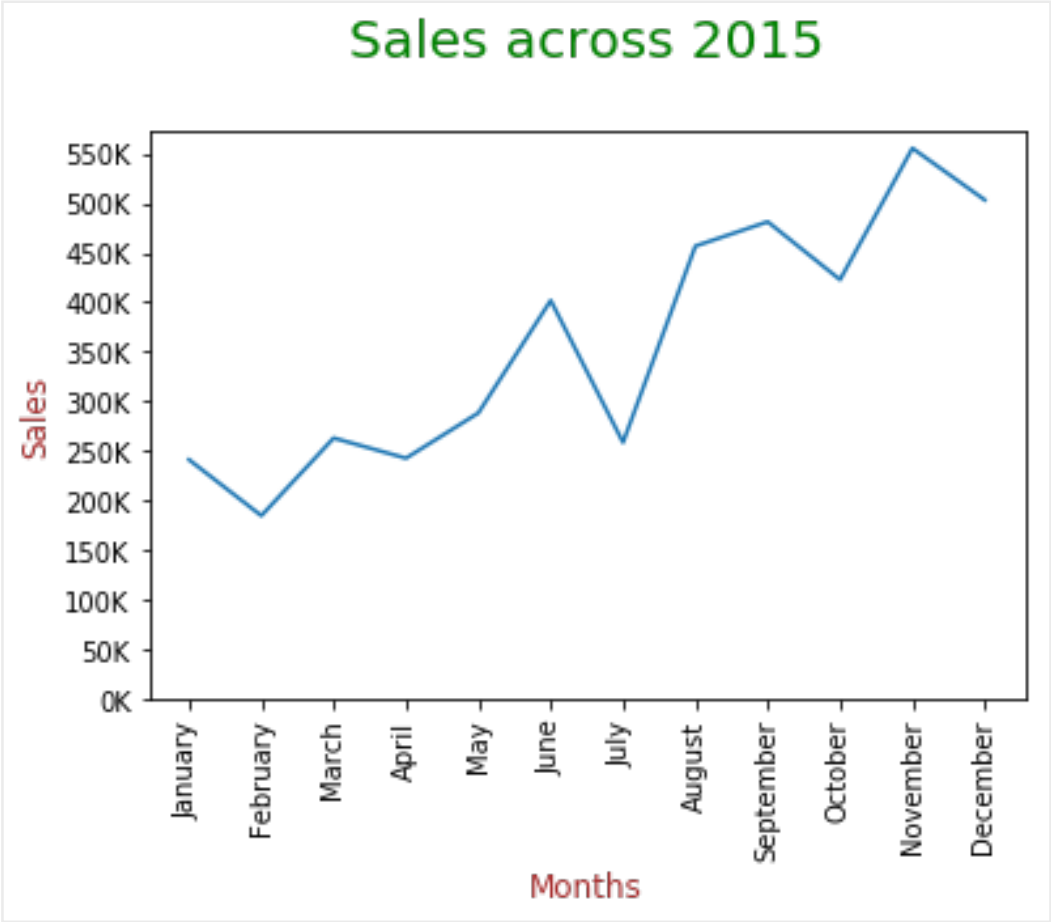
If you specify the colour and marker separately, then you will get a line plot with the points marked. Try to run the following the code for this:
plt.plot(y, 'red', marker = 'o')

A line graph can be helpful when you want to identify the trend of a particular variable. Some key industries and services that rely on line graphs include financial markets and weather forecast. Although you successfully created a line chart in the previous video, you can make certain visual improvements to create a chart that is more easily understandable. Let's watch the next video to

learn about these visual modifications.

In the video above, you learnt how to rotate the tick labels on the axes using the following command:
plt.yticks(rotation = number) #could do for xticks as well

After running the command, the chart will look like the one given below.



As you can see in the diagram, the x-ticks and y-ticks are much more readable.

To further improve the readability of the chart, you can add markers to the data points. Let's watch the next video to learn how to make modifications to add data labels.

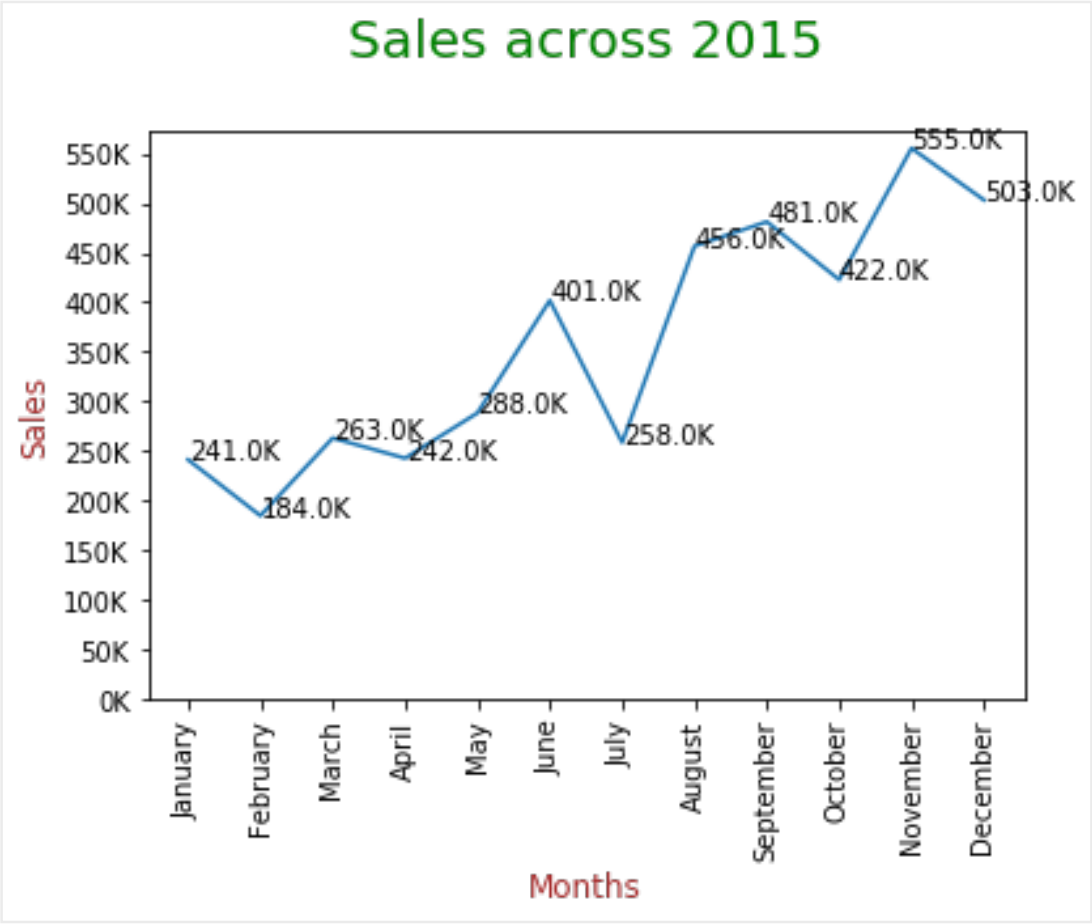
In the video above, you learnt how to use the annotate method to add data labels to the plot. The code given below was used in the previous video.

plt.plot(months, sales)

# Adding and formatting title
plt.title("Sales across 2015**\n**", fontdict={'fontsize': **20**, 'fontweight' : **5**, 'color' : 'Green'})

```
# Labeling Axes
plt.xlabel("Months", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'} )

ticks = np.arange(0, 600000, 50000)
labels = ["{}K".format(i//1000) for i in ticks]
plt.yticks(ticks, labels)

plt.xticks(rotation=90)

for xy in zip(months, sales):
    plt.annotate(s = "{}K".format(xy[1]//1000), xy = xy,  textcoords='data')

plt.show()
```

After running this code, your plot will look like the one given below.



In the earlier segment on scatter plot, you used the annotate method to add data labels to a scatter plot. Similarly, the annotate method can be used to add data labels to graphs as well. Now that you know the basics of line graphs, attempt the following questions.

Histograms

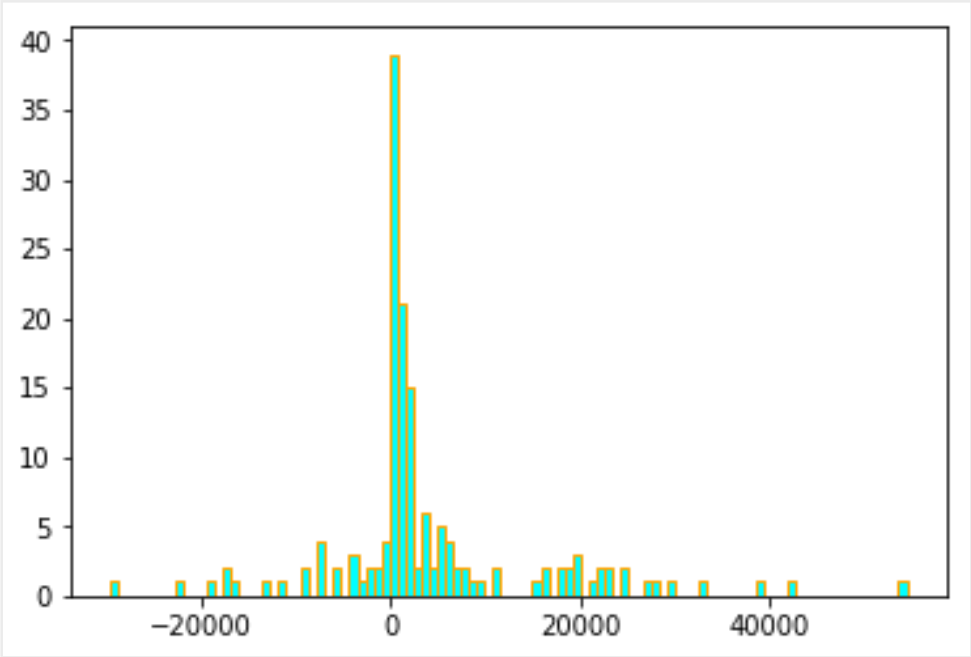
A **histogram** is a frequency chart that records the number of occurrences of an entry or an element in a data set. It can be useful when you want to understand the distribution of a given series. Let's watch the next video to learn how to plot a histogram.

As shown in the video above, you can use the following command to plot a histogram:

plt.hist(profit, bins = **100**,edgecolor='Orange',color='cyan')

plt.show()

After running this code, your histogram will look like the one given below.



The x-ticks in the histogram above are not very informative. Let's try to add more detailed x-ticks so that the data is more readable. In the next video, you will learn how to add more information to classes and x-ticks.

In the video above, you learnt how to use the hist() function to add more information to classes and x-ticks. Now that we have covered the basics of histograms, attempt the following questions.

Box Plot:-

Box plots are quite effective in summarising the spread of a large data set into a visual representation. They use percentiles to divide the data range.

The percentile value gives the proportion of the data range that falls below a

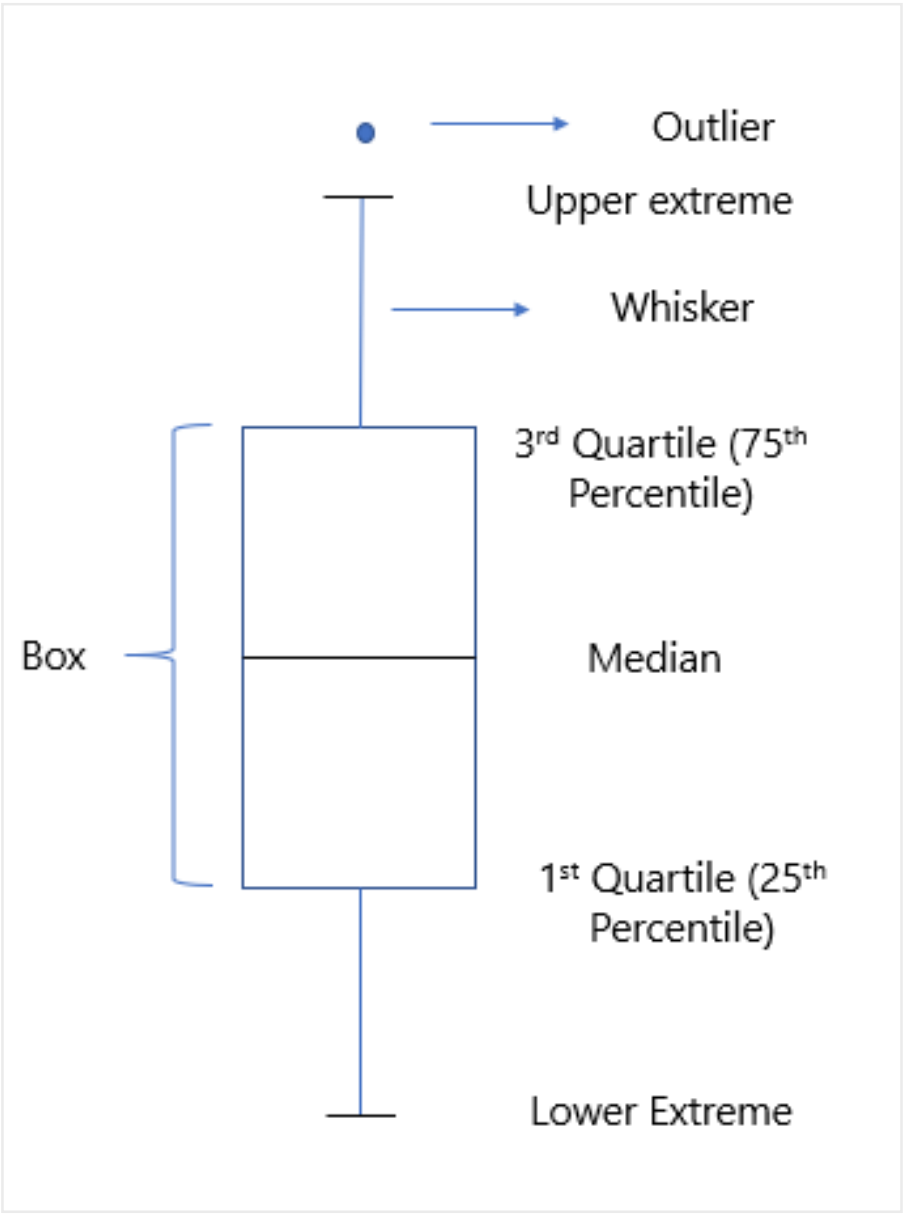
chosen data point when all the data points are arranged in the descending order. For example, if a data point with a value of 700 has a percentile value of 99% in a data set, then it means that 99% of the values in the data set are less than 700.

Let’s watch the next video to learn more about box plots.

You can use the following command to create a box plot in Python using Matplotlib:

```
plt.boxplot([ list_1, list_2])
```

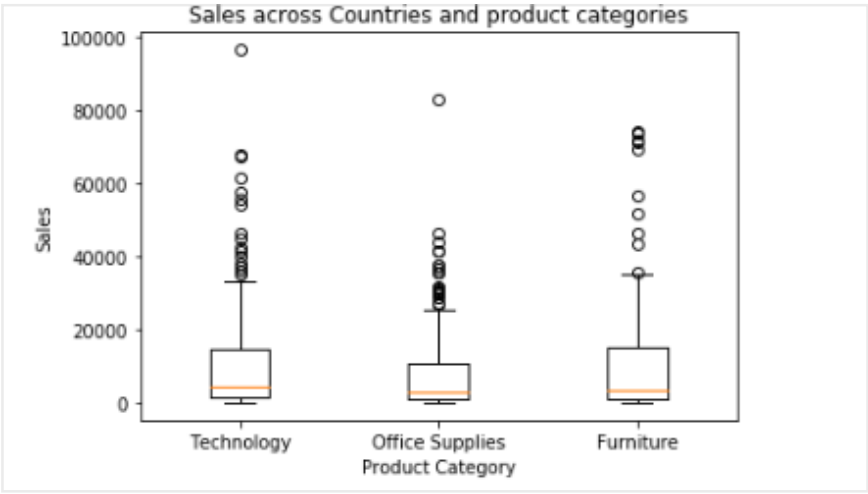
The figure below shows a typical box plot with explanations for each element in its construction.



Box plots divide the data range into three important categories, which are as follows:

- **Median value**: This is the value that divides the data range into two equal halves, i.e., the 50th percentile.
- **Interquartile range (IQR)**: These data points range between the 25th and 75th percentile values.
- **Outliers**: These are data points that differ significantly from other observations and lie beyond the whiskers.

The box plot given below was constructed in the video.



Box Plot

As you can see, there are quite a few outliers in the given data set.

Attempt the quiz given below to test your understanding of the box plots.

Feedback:
This is the correct answer. The IQR can be calculated by finding the 25th and 75th percentile values.

Feedback:
Correct Answer! A line plot and a scatterplot essentially plot the x-y relationship between two quantitative variables. They show how one variable changes with respect to another quantitative variable. To find a relationship using a line plot, one additional step that you have to complete is to first sort the elements of x-axis.

Subplots:-

In the previous segments, you learnt the basic ways to create plots.

Sometimes, it is beneficial to draw different plots on a single grid next to each other to get a better overview of the data set. For example, suppose you have some data on e-commerce purchases. If you want to analyse the number of purchases across different categories, you can create multiple bar charts for each category, for example, one for male buyers and another for female buyers. These two charts, when placed next to each other, make it easy for you to compare the buying patterns of the male and female consumers.

Different plots presented in a single plot object are commonly referred to as **subplots**. Let's watch the next video to learn how to create subplots inside a single plot in Matplotlib.

In the video above, you learnt how to create multiple plots in a single graph. Behzad added plots for Asia, USCA and the Asia-Pacific region. Let's watch the next video and continue filling up the same plot, and learn about some new features.

To recap, you can use the following Matplotlib command to create subplots in Python:

- fig, ax = plt.subplots(): It initiates a figure that will be used to comprise multiple graphs in a single chart.

Subplots are a good way to show multiple plots together for comparison. In the video above, you learnt how to plot different categories together in the same chart. Subplots offer the ability to create an array of plots, and you can create multiple charts next to each other to make it look like the elements of an array.

Let’s watch the next video to understand this with the help of an example.

You can use the following command to create an array of plots.

- plt.subplot(nrow, ncol, x): It creates a subplot. 'nrow' and 'ncol' are the dimensions of the array inside the figure, and 'x' is the position in the array.
  (You can visit this web page to understand the attributes associated with the subplot method in detail.)
- In plt.subplot()method, the numbering of subplots starts from the top-left element of the grid and moves rightward along each row. The numbering then continues to the next row from left to right. For example, suppose you have created a grid of nine subplots. The numbering would look like this:

• Subplot 1	• Subplot 2	• Subplot 3
• Subplot 4	• Subplot 5	• Subplot 6
• Subplot 7	• Subplot 8	• Subplot 9

- Suppose you need to access the subplot present in the second row and the second column (i.e., 'Subplot 5' in the table above). You can do so by using the following command:
plt.subplot(**3**, **3**, **5**)

In this segment, you learnt how to add multiple plots to the same graph and create an array of plots in the same picture. However, as you saw in an earlier video, the picture was quite small in size and the information was not easily understandable. Let’s watch the next video to learn how to modify the size of the overall plot.

In the video above, you learnt how to use the set_size_inches() function to modify the size of the plot. You also learnt how to combine both the two

techniques, plotting on the same graph and plotting an array of subplots. With this, we have covered all the basics of subplots.

Also, please note that in the last video legend for the Latin America region has been fetched incorrectly due to variable name inconsistency ("LATM has been used for legend creation instead of LATAM")

Let’s attempt the following question.

Choosing Plot Types:-

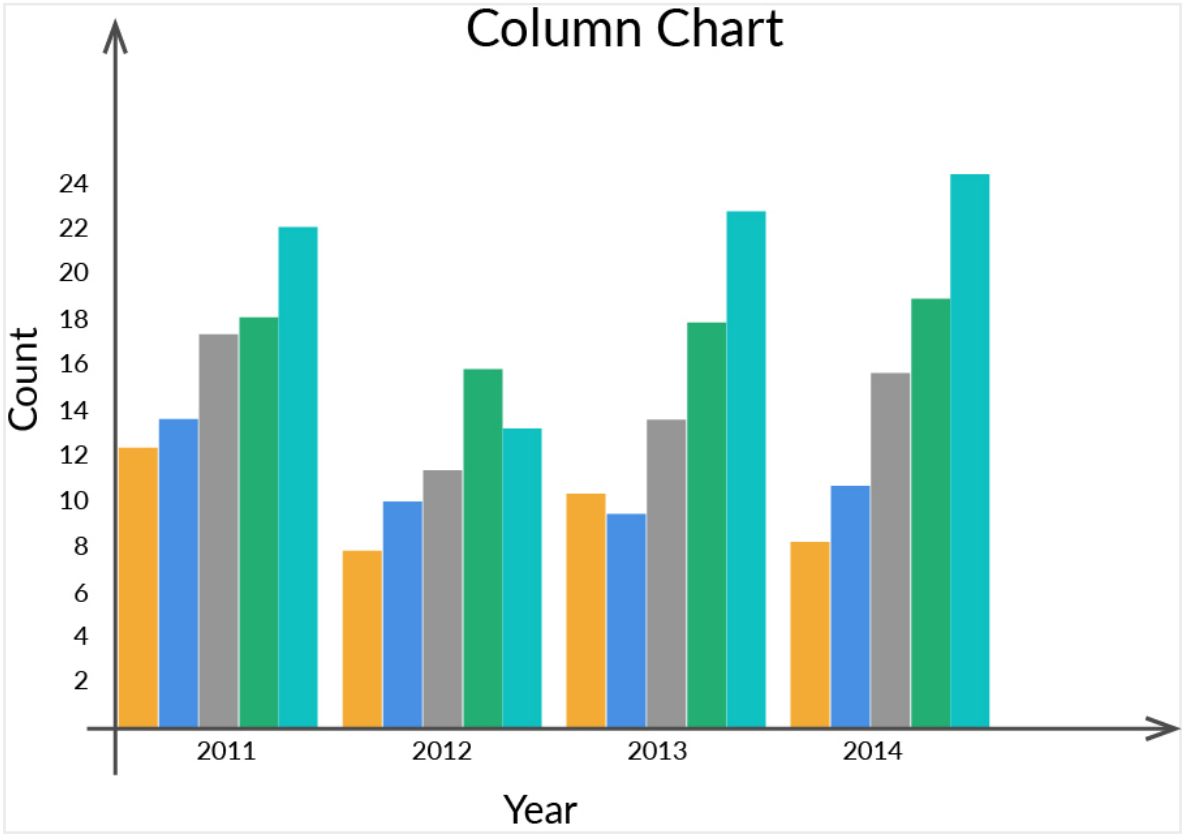
Before this segment, you have seen a lot of different types of plots. Each of the plot types is good at communicating a specific type of information. Which means, in certain situations, certain plot types are preferred over the others. So, how do you select the best possible plot type in a given situation? To answer this question, you need to first define the objective of creating a plot. A good visualisation, along with the right type of graph, presents the relationship between different variables effectively and allows you to analyse them at a quick glance. Let’s take a look at some principles that can help you select the right type of chart. Let's hear about selecting a plot type from Behzad.

Let's summarise the points discussed in the video.

Comparison

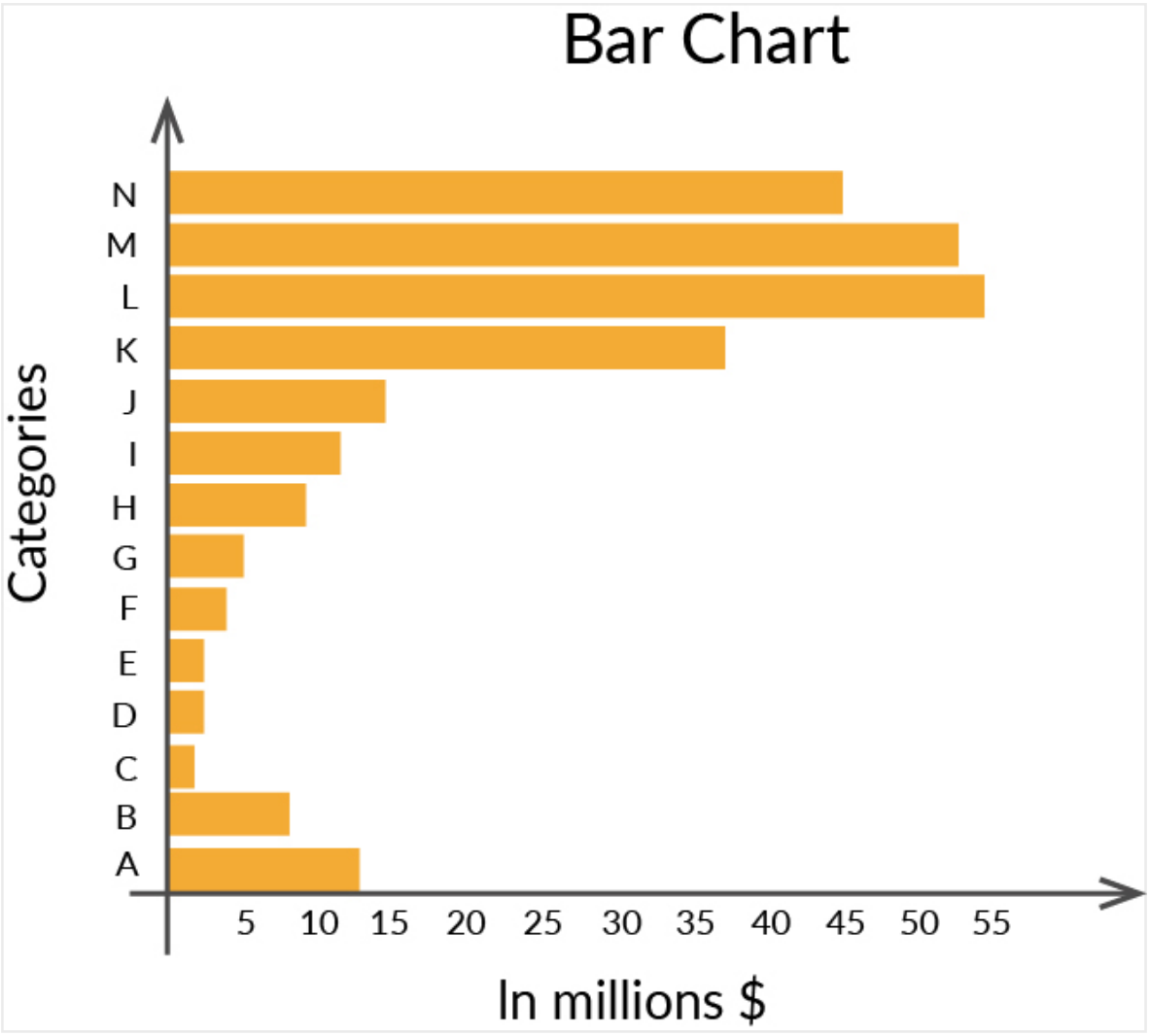
These charts can be used when you want to compare one set of values with other sets of values. The objective is to differentiate one particular set of values from the other sets, for example, quarterly sales of competing phones in the market. The following two types of charts are used to show a comparison:

1. Column chart


Column Chart
Count
24
22
20
18
16
14
12
10
8
6
4
2
2011
2012
2013
2014
Year

2.

3. Bar chart

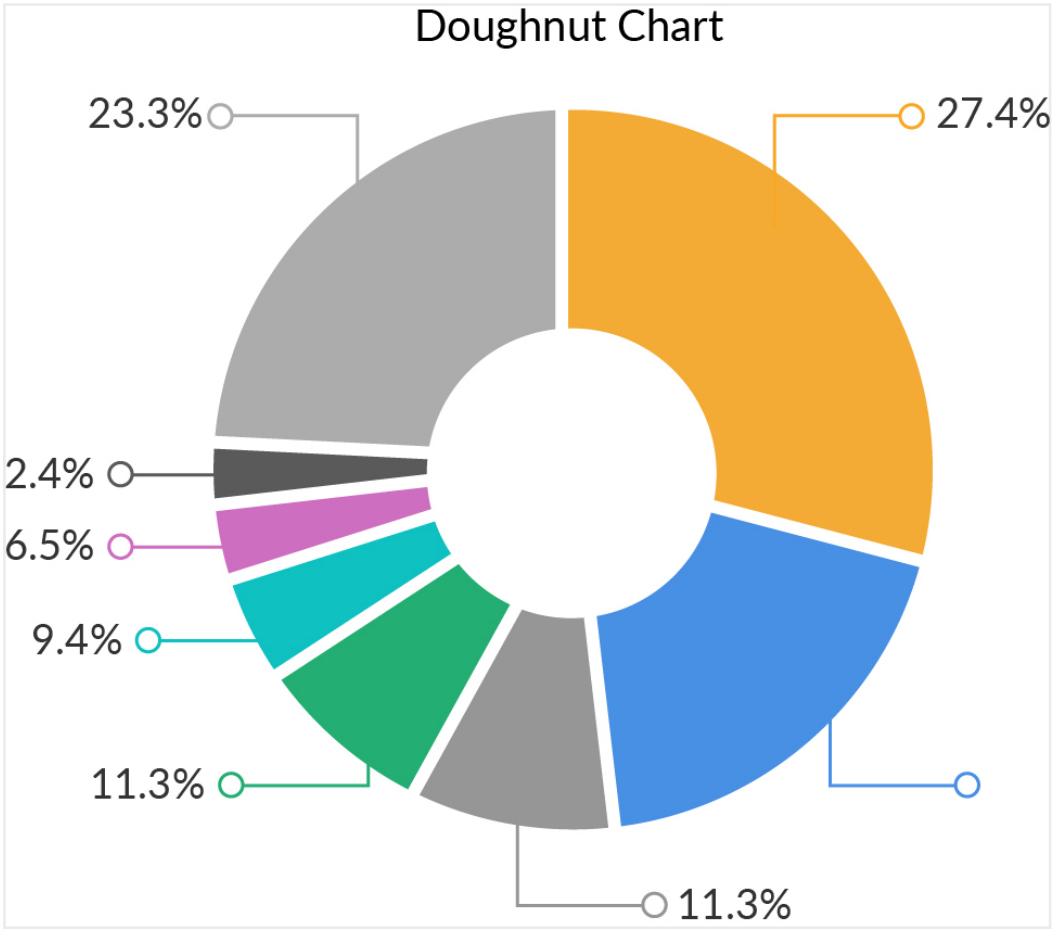


4.

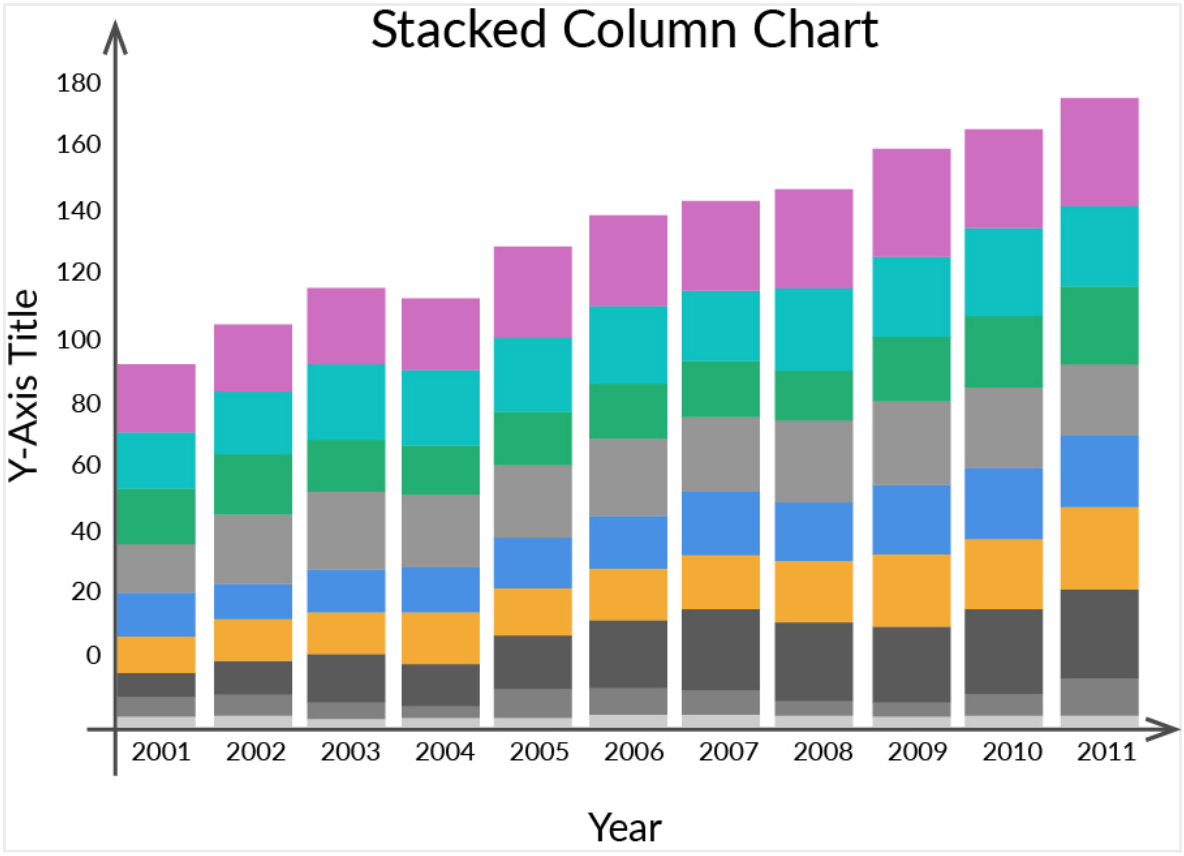
Composition

You would need to use a composition chart to display how the various elements make up the complete data. Composition charts can be static, which shows the composition at a particular instance of time, or dynamic, which shows the changes in the composition over a period of time. Two of the popular composition charts are as follows:

1. Pie/ Doughnut Chart



2. Stacked Column chart



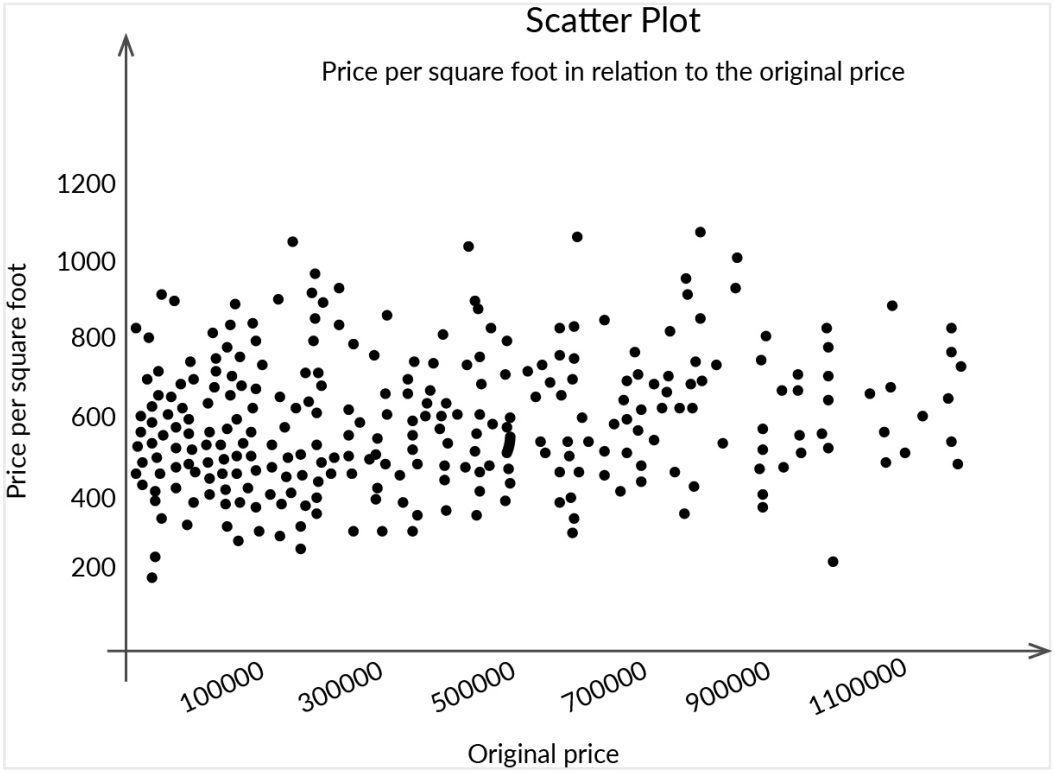
The pie chart is by far the most common way to represent static composition,

while the stacked column chart can be used to show the variation of composition over a period of time.

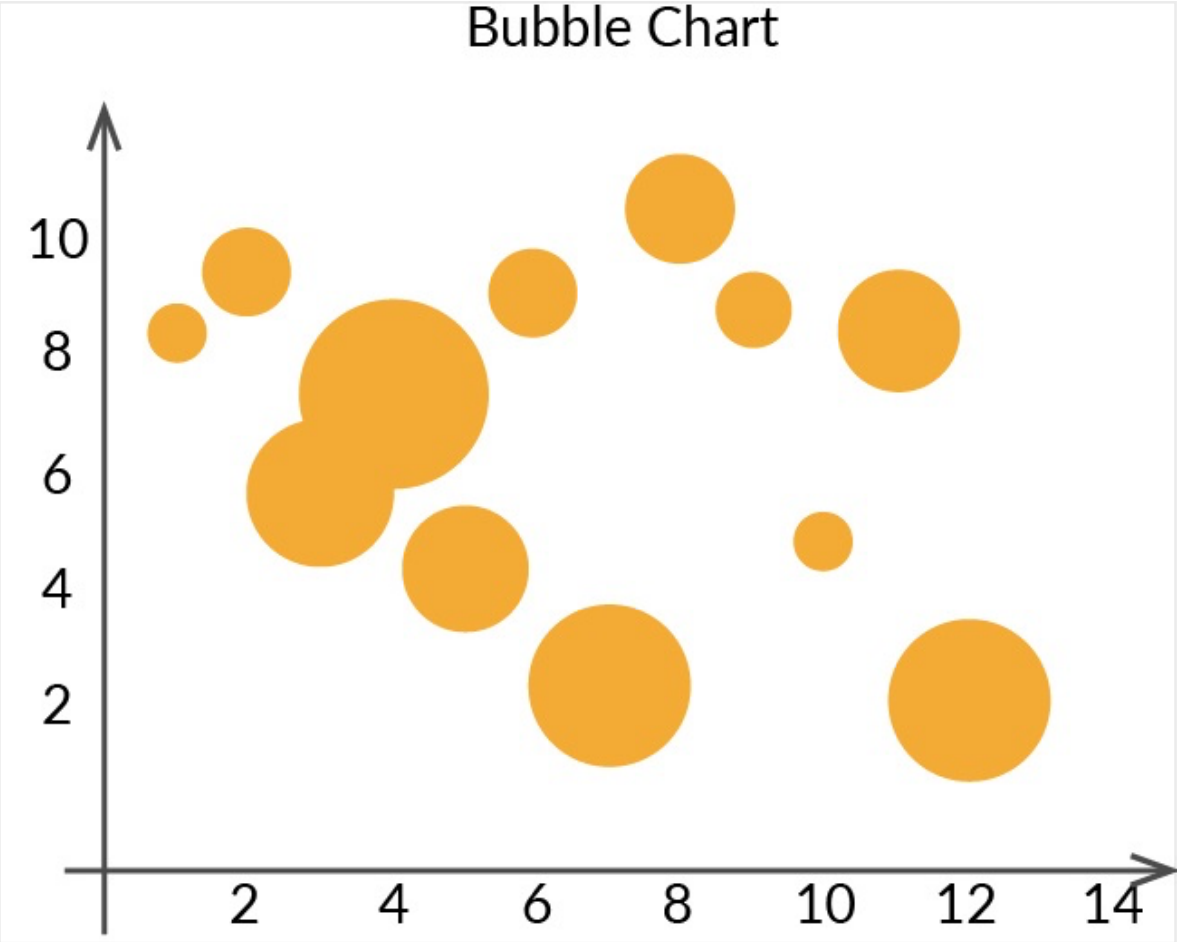
Relationship

A relationship chart helps in visualising the correlation between variables. It can help in answering questions such as ‘Is there a correlation between the amount spent on marketing and the sales revenue?’ and ‘How does the gross profit vary with the change in offers?’. Two of the most common types of charts used to visualise relationships between variables are as follows:

1. Scatter plot



2. Bubble Plot

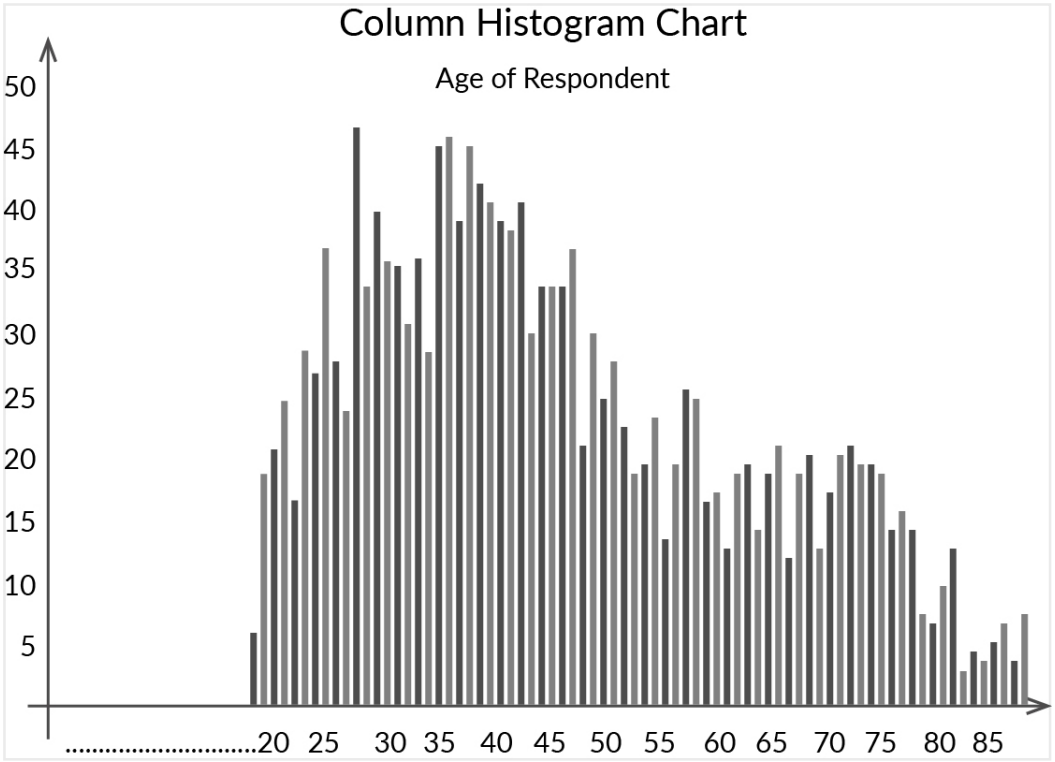


A scatter plot can help correlate two variables, whereas a bubble chart adds one more dimension, i.e., the size of the bubble (usually indicative of the frequency of occurrence of that particular data point)

Distribution

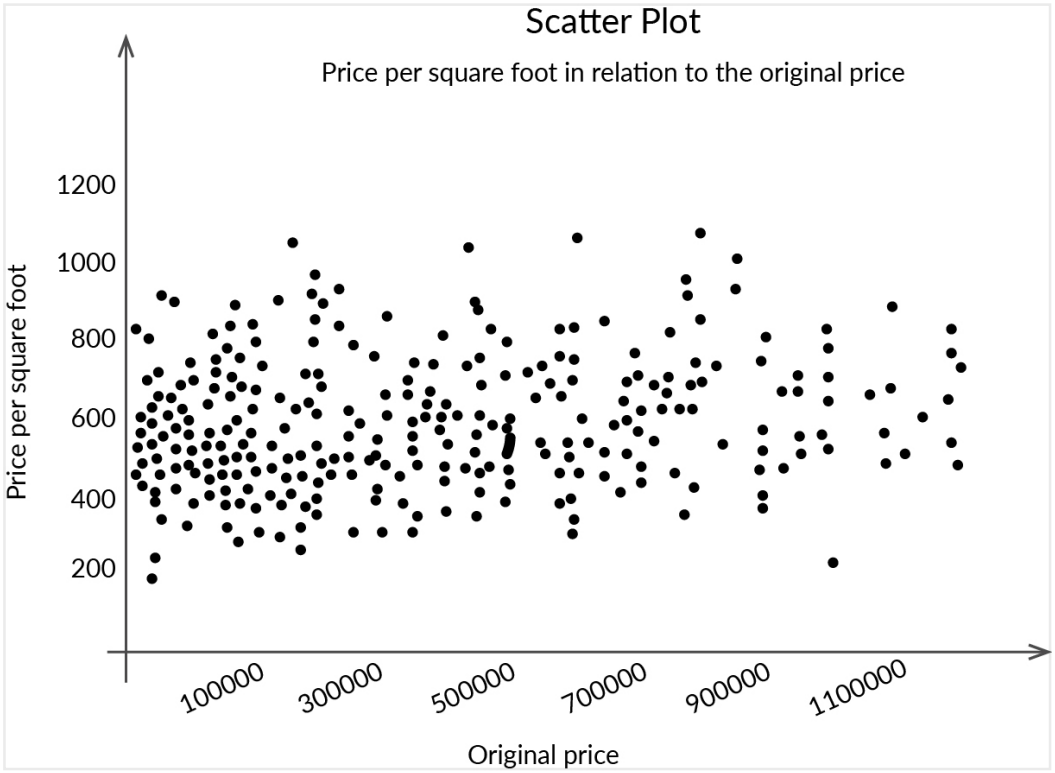
A distribution chart tries to answer the question 'How is the data distributed?'. For example, suppose you asked everyone their age in a survey. Using a distribution chart will help you visualise the distribution of ages in the data set. The distribution can be over a variable, or it can also be over a period of time. Two of the most commonly used charts for visualising distribution are as follows:

1. Histogram



2.

2. Scatter plots



Histograms are quite good at displaying the distribution of data over intervals, whereas scatter plots are good at visualising the distribution of data over two

different variables.

These are the most common objectives of data visualisation, and the aforementioned types of charts help in achieving these objectives. However, this list is not exhaustive; you can always find more goals to create a visualisation. One of the key takeaways from this segment is to observe the chart that you have prepared and check whether it serves the purpose for which it was created. If it does serve the purpose, then you have created the right type of chart. If it doesn't, then you need to try other options to make the message of the chart very clear.

You can apply your learnings from this segment in the practice notebook attached below.

Summary:-

This session introduced you to the world of data visualisation. Throughout this session, you performed visualisations using Matplotlib. Now, let's summarise all that you learnt in this session.

You started this session by understanding the importance of visualisation in interpreting data. Then you learnt about the different types of graphs and charts, namely:

- Bar chart
- Scatter plot
- Line graph
- Histogram
- Box plot

Further, you learnt about the elements that help display additional information about a plot. Also, you learnt how to fit several subplots inside a single plot object, which is useful for comparing different properties or elements of your data.

Additional References:

You can refer to the links provided below to learn more about visualisation:

1. You can go to this link from Gramener to know how data visualisation can help you derive insights about the winning political parties in India.
2. You can view this link from SocialCops to see a creative visualisation on the ageing world population.
3. You can refer to this link to learn how to choose a graph plot for your data.
4. You can refer to this link to find additional video tutorials by **Corey Shaffer** on Matplotlib
5. You can refer to this link to find **official Matplotlib Documentation**

6. You can learn more about choosing the right plot type by referring to this link.
7. You can refer to this link to find out additional information about plotting real-time data.

In the next session, you will go through a case study which will make use of the tools you learnt in this session.

Module 2 : Data Visualisation: Case Study

Introduction:-

In this session, you'll get an understanding of the importance of visualisation through certain **real-world examples** and then you'll begin your journey into the world of charts and graphs using a case study.

However, before you dive deep into visually understanding the data, it is important to work on certain basic data cleaning steps in the given dataset such that you are able to represent the visualisation and find the required insights.

Case Study: Mind Map:-

Now that you have understood why data visualisation is crucial in drawing insights, let's listen to Rahim as he gives a brief overview of this and the next session.

Here's a brief mindmap of the contents of this module.

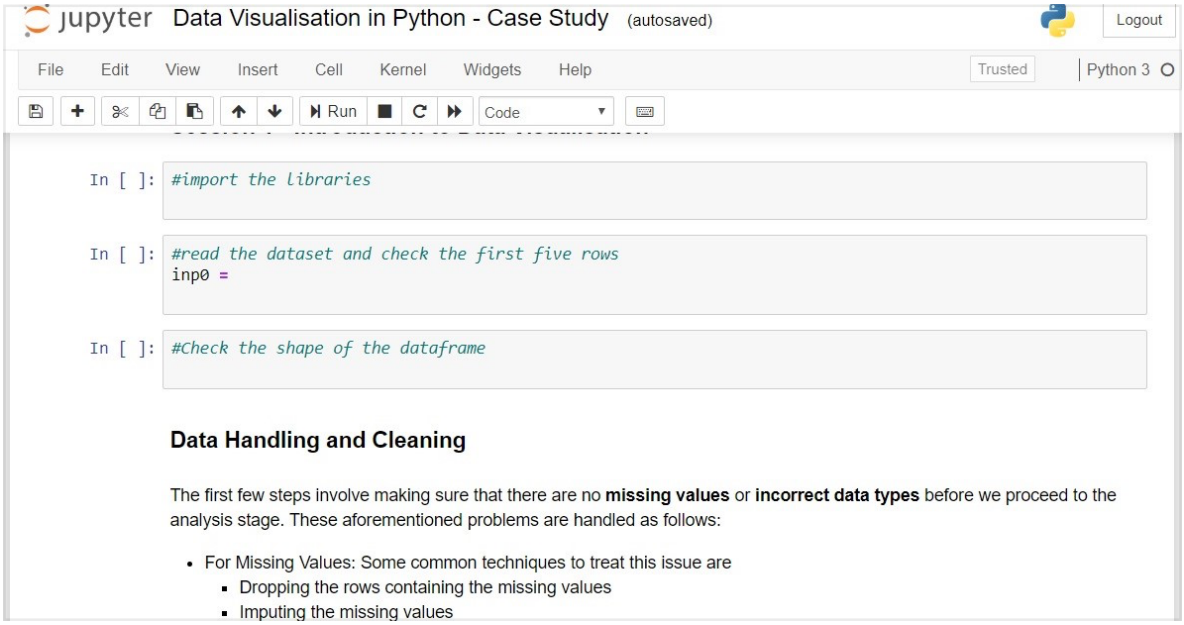
- In the next segment, you'll start working with a case study: **Analysing the Google Play Store Ratings Dataset**. Here, you'll start with some basic data wrangling techniques, which you would have learnt in the previous modules
- For the data visualisation part, you'll be revisiting the concepts of **Matplotlib**, which you learnt in the previous session. Then you'll be introduced to the popular **Seaborn** library, which gives you access to a lot of flexible and aesthetic plotting methods.
- Towards the end of the module, you'll also get a brief exposure to the library '**Plotly**', which is another visualisation library that helps you make beautiful interactive charts in Python.

HOW TO GO THROUGH THIS MODULE

For the case study demonstration, you are encouraged to code along with the SME as he explains each step in the video lectures. This will help you get a

strong grip on the various tenets for analysing datasets and to think like a data scientist in any scenario. Also, this notebook will help with your reference to revise any specific concepts with respect to this module. The code stub has been provided and you can download the same from the link below.

When you open the stub file it will look something like the image shown below (Open the image in a new tab to magnify it even further)



Code Stub File

As you can see it's a commented notebook containing the following main items

- **Commented Instructions** - For each of the coding steps that the SME will be performing while going through the case study, instructions have been provided corresponding for each step in the form of comments. Your task is to code along with the SME and fill the notebook as per the specified comments.
- **Topic Summary** - For each of the topics that are being taught in this module, a brief summary has been provided in the notebook as well. This way, before performing each step, you can have a quick look at the theory to understand what type of steps you are performing and why.

This notebook has been designed to give you ample practice as well as act as a revision resource. Therefore, it is advised that you go through the coding videos once or twice till you get a good amount of practice as this will come in handy when you move forward in this program.

For the coding practice questions in this module, you also will be

provided with a Jupyter Notebook containing instructions and lines of code. You're expected to solve the questions at the respective places in the Notebook and then choose the correct answer on the platform. The Notebook already contains the code to import datasets. In the next segment, let's go over the problem statement and the dataset of the case study.

Case Study Overview:-

As explained earlier, you'll be learning this session with the help of a case study. This will enable you to understand how data visualisation aids you in solving business problems. Let's listen to Rahim as he introduces the problem statement and our objectives for the same.

As you learnt in the video, the team at Google Play Store wants to develop a feature that would enable them to boost visibility for the most promising apps. Now, this analysis would require a preliminary understanding of the features that define a well-performing app. For which, you can ask questions like:

- Does a higher size or price of an app necessarily mean that it would perform better than the other apps?
- Or does a higher number of installs give a clear picture of which app would have a better rating than others?

You'll learn to use data visualisation to answer these questions and derive corresponding insights. Let's take a quick look at the dataset in the following video:

[**Note**: The dataset has already been provided to you in the earlier segment along with the Jupyter notebook file. Please go through the section on 'How to go through this module' from the Module Overview segment]

Here's a small sample of apps present in the dataset along with their corresponding headers.

App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
Google Chrome	COMMUNICATION	4.3	9643041	21516.5	1,000,000,000+	Free	0	Everyone	Communication	August 1, 2018	Varies with device	Varies with device
IMDb Movies & TV	ENTERTAINMENT	4.2	501498	12000	100,000,000+	Free	0	Teen	Entertainment	July 26, 2018	Varies with device	Varies with device
WPS Office	PRODUCTIVITY	4.5	1507205	37000	100,000,000+	Free	0	Everyone	Productivity	July 30, 2018	11.1.3	4.0 and up
Samsung Calculator	TOOLS	4.4	9602	2500	100,000,000+	Free	0	Everyone	Tools	July 5, 2018	6.0.61.5	7.0 and up
Angry Birds Rio	GAME	4.4	2610526	46000	100,000,000+	Free	0	Everyone	Arcade	July 3, 2018	2.6.9	4.1 and up
Telegram	COMMUNICATION	4.4	3128250	21516.5	100,000,000+	Free	0	Mature 17+	Communication	July 27, 2018	Varies with device	Varies with device
Home Workout - No Equipment	HEALTH_AND_FITNESS	4.8	428156	15000	10,000,000+	Free	0	Everyone	Health & Fitness	June 28, 2018	Varies with device	Varies with device

Google Playstore Apps Rating

You're advised to go through the dataset on your own and get an understanding of the various features and data points available there. From the next segment onwards, you'll begin with the hands-on demonstration in Python.

Data Handling and Cleaning: I:-

Now that you have an overall understanding of our objectives for this case study, it's time to go ahead and begin the hands-on demonstration in Python. As mentioned earlier, you're advised to code along with the SME in the following videos to get the optimum learning experience out of them.

The dataset contains a total of 10,841 apps and 13 features or columns.
The **df.info()** function showed us that some of the columns (like Rating) have missing values and some have incorrect data types associated with them. Let's discuss both of them briefly here:

- **Missing values**: You would almost always encounter data which have rows where no observation is recorded for a certain variable. These can affect the analysis process and the generated insights significantly. Some common techniques to treat this issue are
  - **Imputation**, where you replace the missing value with another estimated value
  - **Dropping** the rows containing the missing values altogether
  - or depending on the case, you can also go ahead and keep the missing values as long as they don't affect the analysis.

- **Incorrect data types**: This discrepancy mostly occurs due to some incorrect entry in the column which is stored in a format other than the desired one due to which the entire column gets misclassified. Or in some other cases, the format of the entire column is different from what we need for our analysis purposes. You either have to fix certain values or clean the entire column only to bring it to the correct format.

Missing values will affect our statistics drastically, for starters our inbuild functions of mean, sum, var etc will give incorrect results which clearly is quite dangerous. Also, you need the values to be in the numeric format of int or float to perform these operations, you cannot find the mean or median of a collection of strings, can we? Now, before you proceed to the data analysis and visualisation part, it is essential for you to remove the above discrepancies.

As explained by Rahim, you can easily use
the **isnull()** and **isnull().sum()** functions to determine the missing values in a dataset. This will give you the number of missing values corresponding to each column.

Now, there are a number of ways in which you can handle missing values. In some cases, you delete the records containing the null values. For example, the **Rating** column is our target variable, which would influence our analysis greatly as we keep progressing. Therefore, imputing values may skew our results significantly and hence we should drop them.

Now let's go ahead and check the other columns and see what actions are required for them.

Note - For the in-video question, the first option should be '4.1 and up' and not '4.1 and above'. Similarly the corresponding answer would be len(inp1[inp1['Android Ver']=="4.1 and up"])

In the case of the Android Ver column, you imputed, or you replaced the missing value with the mode for that column. Computing the mode can be done either using the **value_counts()** function or using the mode function directly.

Imputations are generally done when keeping the missing values disbars you from doing further analysis and eliminating the rows containing those values leads to some bias. The estimation is based on the mean, mode, median, etc. of the data.

In cases where there are numerical columns involved, both mean and median offer up as a good imputed value. In the case of the categorical column, mode turns out to be a decent enough imputation to carry out.

Data Handling and Cleaning: II:-

In the previous segment, you were briefly introduced to the dataset and also performed some basic data-cleaning tasks by handling the null values. The next step would be to handle the data types of the columns. It is essential that your columns are also in the correct format or else it may hamper any further analysis. To understand the motivation for doing this, try answering the following question:

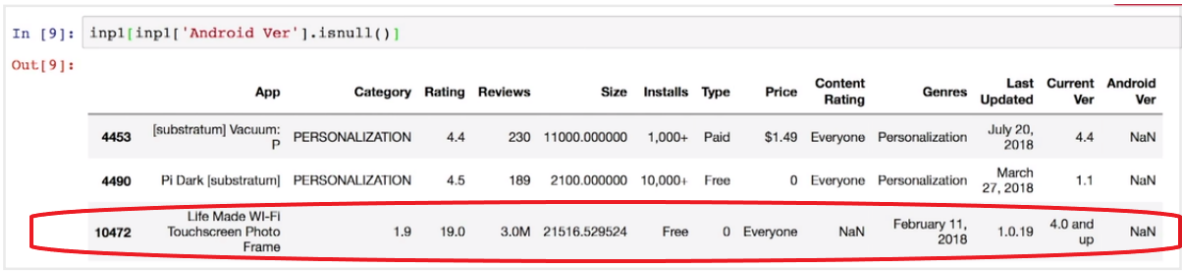
If you recall the earlier modules where you learnt about data types available in python, you know that aggregations like **average, sum or mean** cannot be performed on character variables or strings. Only numeric datatypes like **float** or **int** would allow you to calculate those values. Therefore, it is crucial that you convert all the columns having the incorrect data types to the correct ones.

The **info()** function is pretty handy here to start inspecting the datatypes of the various columns, and we can use it as a stepping stone to performing the data conversion tasks. Let's hear Rahim talk about converting the data types of columns.

As you saw in the video, the **Price** column had an additional '**$**' sign for every paid app that was visible clearly once we used the **value_counts()** function. This resulted in the column being treated as an object instead of a float-type value. So, your first task was to clean those columns.

Now, let's go ahead and inspect the **Reviews** column and perform the necessary changes to it.

[Note: At 0:34, the SME mistakenly says **3.9 M** instead of **3.0M**]



```
In [9]: inp1[inp1['Android Ver'].isnull()]
Out[9]:
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
4453	[substratum] Vacuum: P	PERSONALIZATION	4.4	230	11000.000000	1,000+	Paid	$1.49	Everyone	Personalization	July 20, 2018	4.4	NaN
4490	Pi Dark [substratum]	PERSONALIZATION	4.5	189	2100.000000	10,000+	Free	0	Everyone	Personalization	March 27, 2018	1.1	NaN
10472	Life Made WI-FI Touchscreen Photo Frame	1.9	19.0	3.0M	21516.529524	Free	0	Everyone	NaN	February 11, 2018	1.0.19	4.0 and up	NaN

In the previous segment, you had fixed the row **'Life Made Wi-Fi'** where all the values had shifted to the right. There you had a value of **3.0M** under the **'Reviews'** column due to which the entire column got treated like an object. Since that row got removed, you no longer had any issues with the **'Reviews'** column and all you had to do was convert it into an ***int*** type.

Then, you saw the issue with the Installs column and observed the problem with it - the values contain both the ',' and '+' symbols, which led to this misrepresentation as an object type. Try solving it on your own and answer the question given below:

Sanity Checks:-

"I became insane, with long intervals of horrible sanity." - EA Poe

Once you've completed the basic data cleaning and data handling tasks, the next step is to ensure that the data that is available with us 'makes sense'. What it means is that the data needs to be factually correct apart from being of the correct data type.

For example, on a test where you can score between 0 and 100, it is not possible for a student to score 110 marks. Therefore, if such discrepancies occur in a data set, then you need to take care of them accordingly. So, in order to quickly check whether the data in the columns is rational and makes sense, you need to perform the so-called sanity checks.

As you saw in the video, three essential sanity checks were performed on the data:

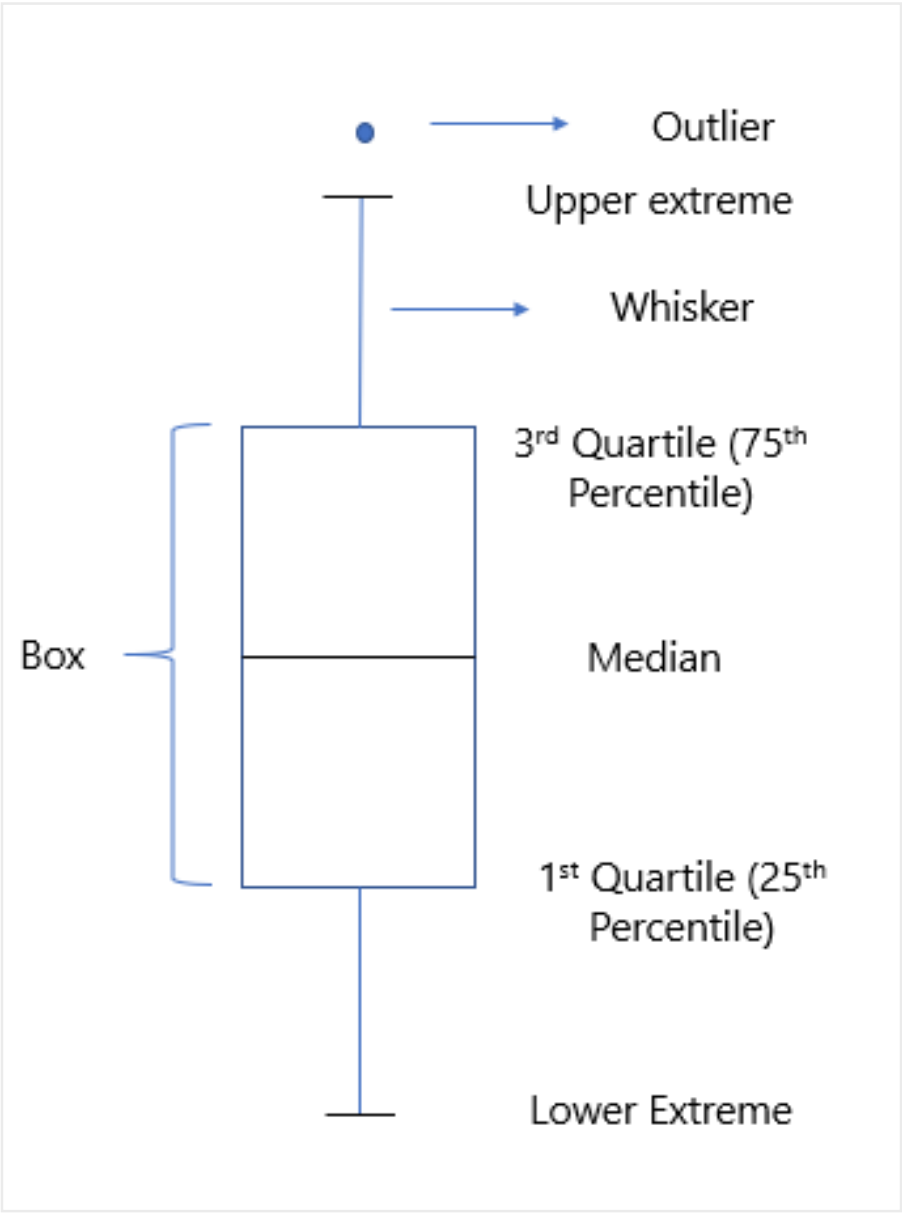
- Rating is between 1 and 5 for all the apps.
- Number of Reviews is less than or equal to the number of Installs.
- Free Apps shouldn't have a price greater than 0.

The first and third conditions were satisfied, whereas the second condition was not satisfied with some records. When you inspected those records, you realised that those apps were likely junk apps and therefore you should ideally remove those records. As mentioned, you are free to apply sanity checks to the dataset on your own as long as they're logically sound.

Outliers Analysis with Boxplots:-

Now that you have performed the sanity checks, it's time to finally turn our attention to identifying and removing extreme values or **outliers** from the dataset. These values can tilt our analysis and often provide us with a biased perspective of the data available. This is where you'll start utilising visualisation to achieve your tasks. And the visualisation best suited for this is the **box plot**. You've already learnt about box plots in the previous session. Here's a brief refresher that recalls the basic concepts as well as explains the workings of the same in identifying outliers.

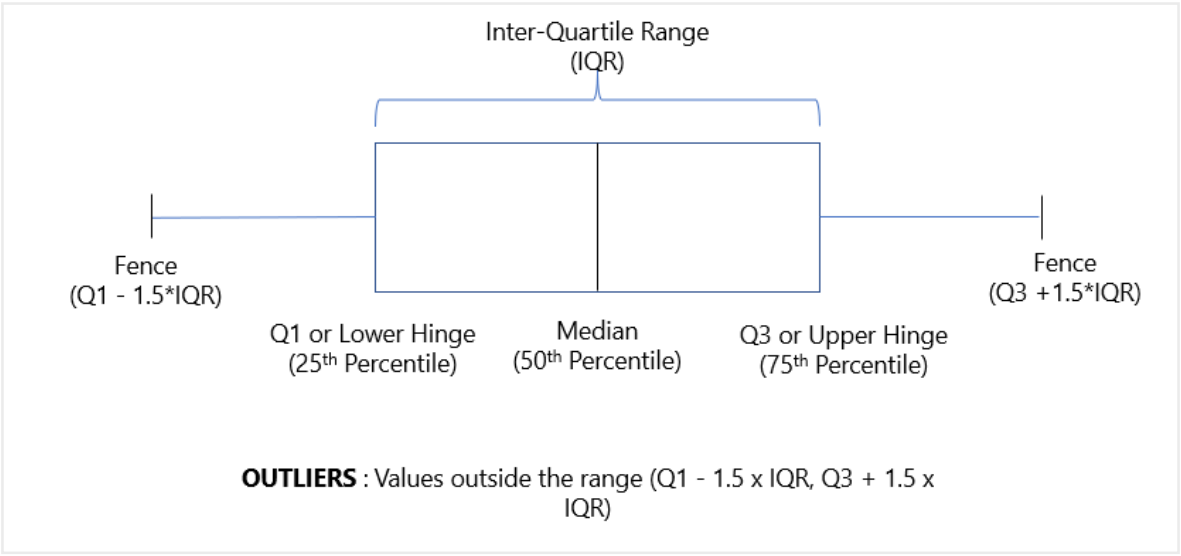
As you saw in the video, a box plot can be described as a representation of the spread of the numerical data for a particular variable. As a matter of fact, it is perhaps the best way to explain the spread of a variable that is numeric in nature. The following diagram shows the various attributes of a box plot:



If you're having trouble visualising a box plot, you can take a look at this simple example to get an idea on how box plots are created.

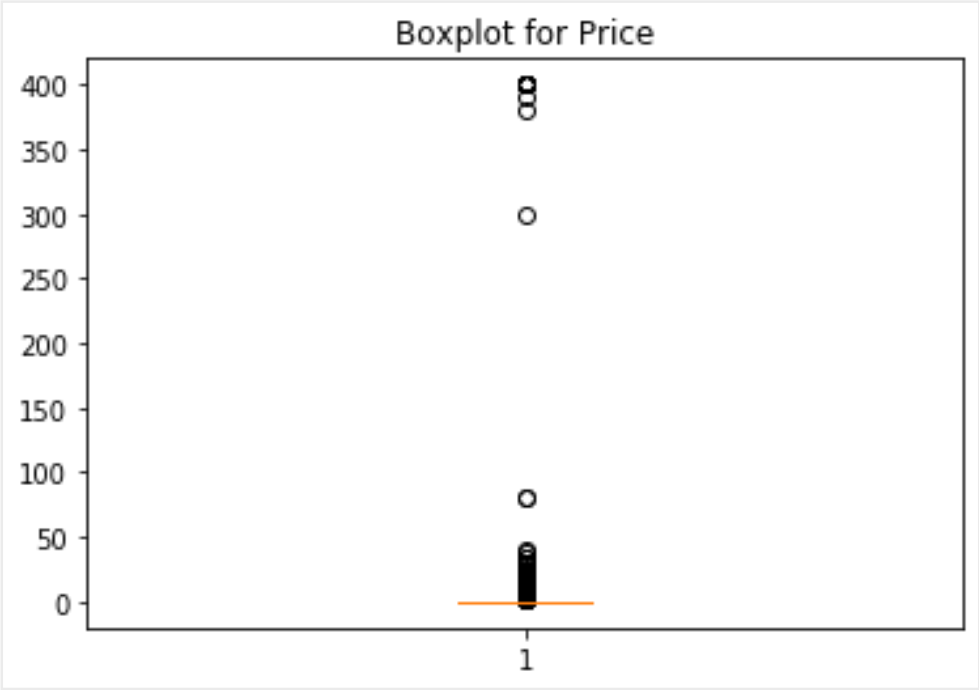
As you might have learnt earlier, the 'maximum' and 'minimum' values, which are represented by the fences of the box plot, are given by the formula **Q3 + 1.5*IQR** and **Q1-1.5*IQR**, respectively. Any value lying outside this range would be treated as an outlier.

[Here **IQR** or the interquartile range denotes the values that lie between the 25th and 75th percentiles.]

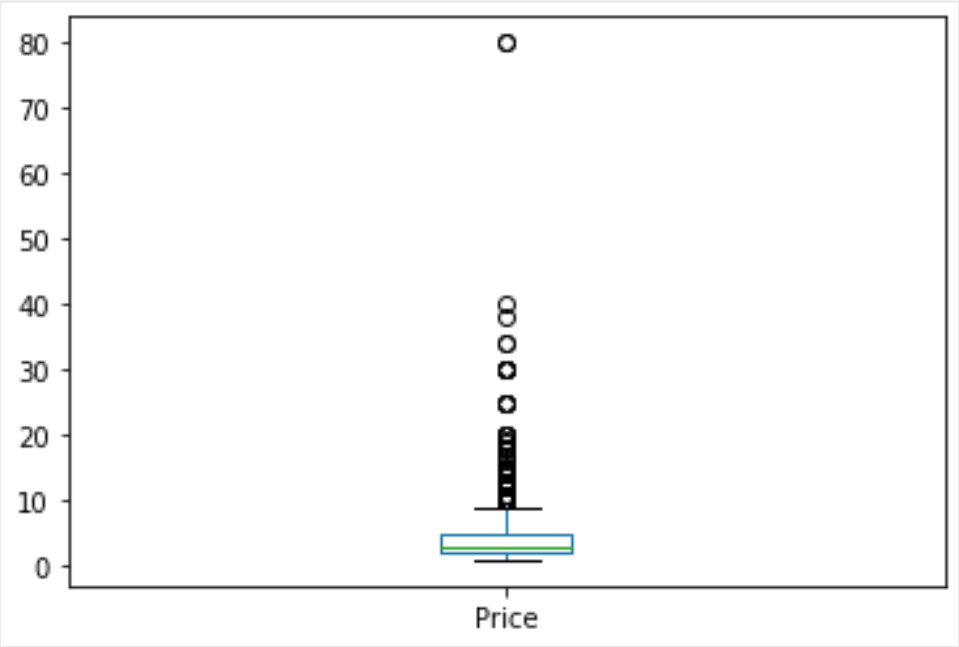


Here are some questions to test your understanding of boxplots before we get into the analysis part for our case study

When you created a box plot for the prices of all the apps using the Matplotlib library (check the official documentation), you observed some really stark outliers.



On further inspection, you observed that a lot of junk apps with 'I am rich' string having those outlier values. Removing them and then inspecting only the paid apps using the pandas boxplot tool (check the official documentation here) gave us the following view:



After inspecting the entries having Price greater than 30, you went ahead and removed them as well. Finally, you were left with 9338 records.

So, the two major takeaways from outlier analysis are as follows:

- Outliers in data can arise due to genuine reasons or because of dubious entries. In the latter case, you should go ahead and remove such entries immediately. Use a boxplot to observe, analyse and remove them.
- In the former case, you should determine whether or not removing them would add value to your analysis procedure.

In this segment, the focus was on detecting outliers. In the next segment, let's visualise the distribution using histograms.

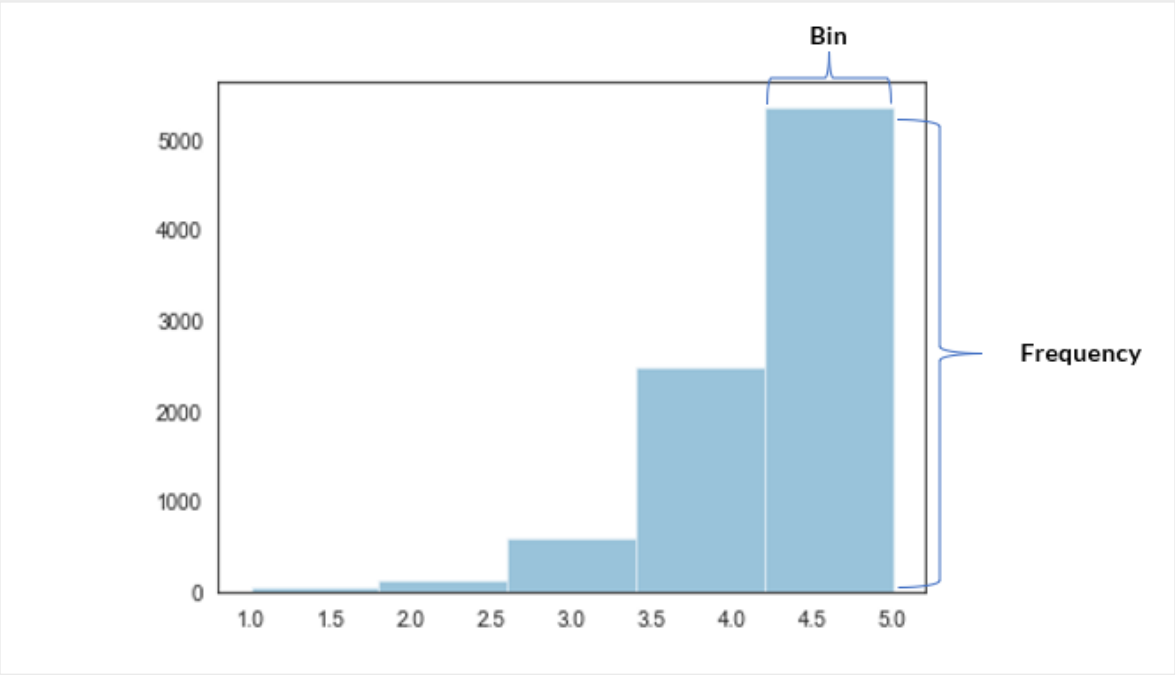
Additional Notes:

- Several definitions exist for outliers for different contexts so that the correct procedure is followed for removing them. You can go through this link for more information.
- Box plots are utilised not just for outlier analysis, but can also be used to compare a certain numeric variable across different categories. You'll learn about this method in the next session where we start analysing the data for insights using Seaborn.

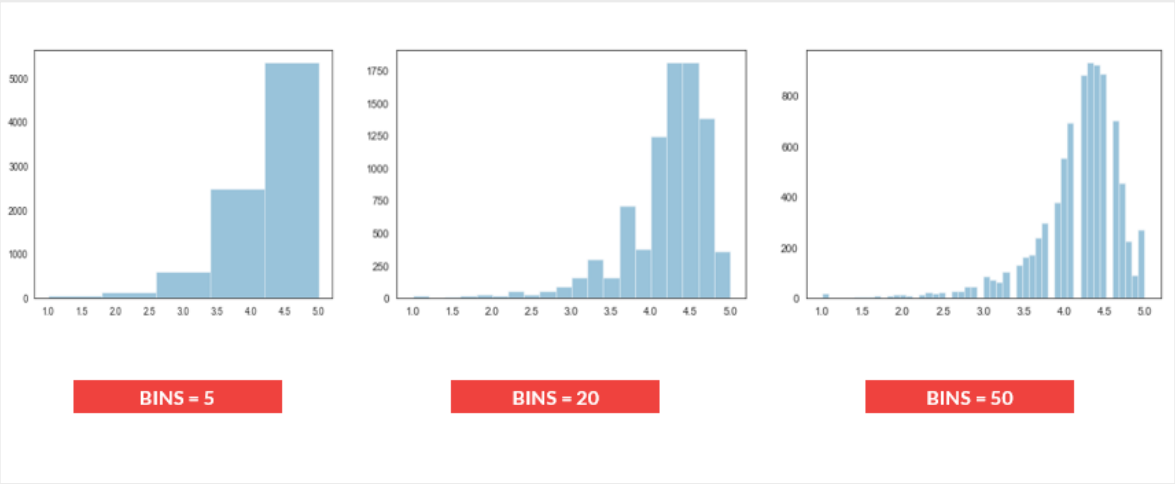
Histograms:-

In the previous segment, you learnt about one way of analysing a numeric variable. There is another way in which you can gauge the spread of quantitative value, and that is through the method of histograms, which you would have learnt already in the previous module in the session on Matplotlib.

Now, as you saw in the video, histograms generally work by bucketing the entire range of values that a particular variable takes to specific **bins**. After that, it uses vertical bars to denote the total number of records in a specific bin, which is also known as its frequency.



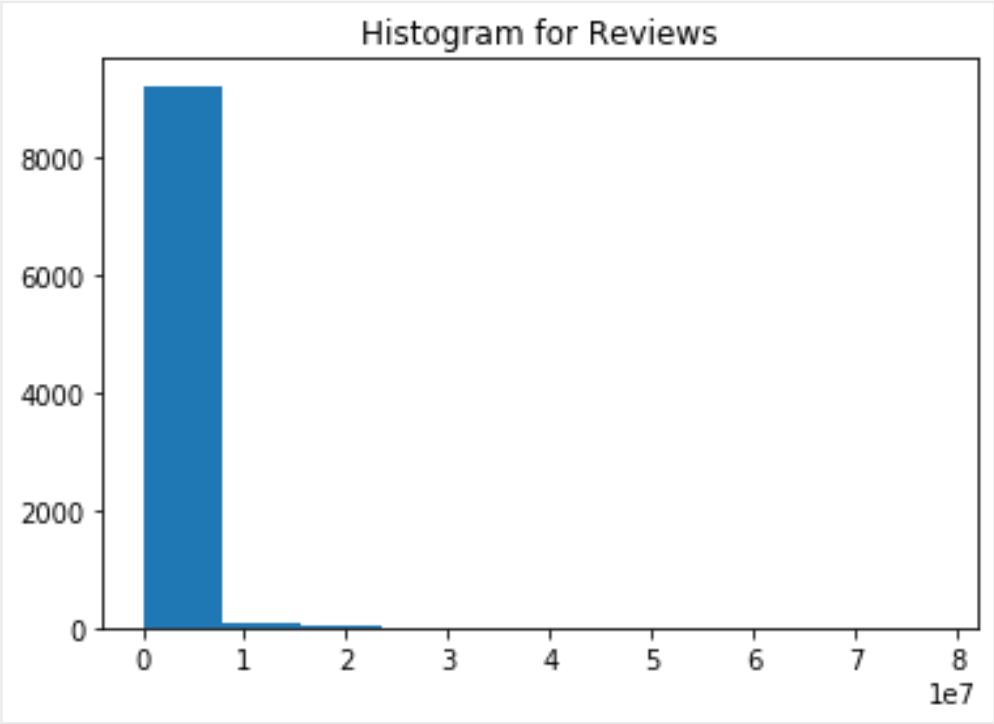
You can manipulate the number of bins to increase or reduce the granularity of your analysis:

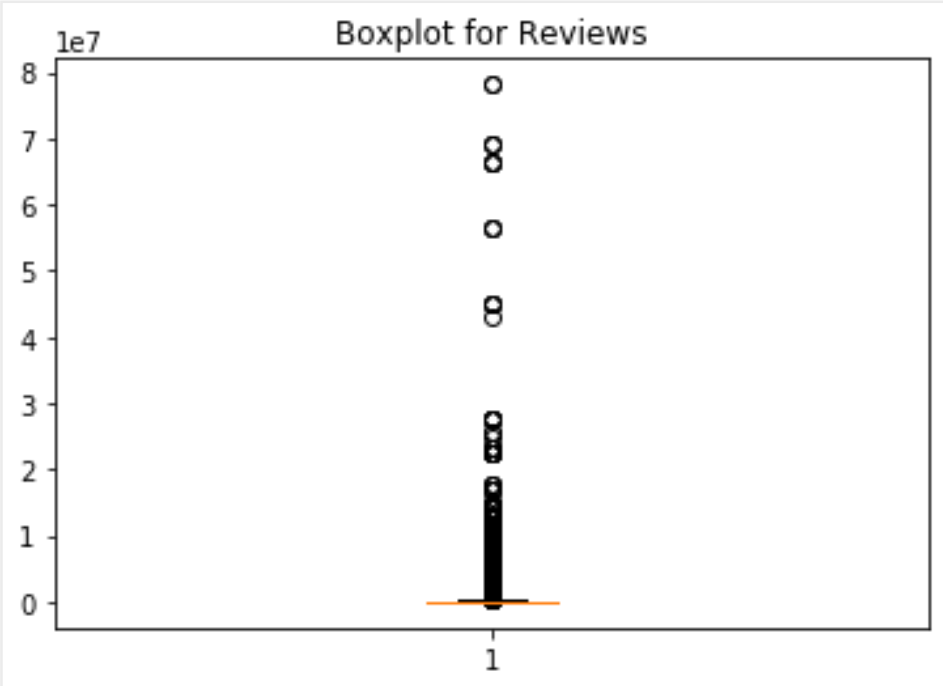


As you can observe, increasing the bins to 20 gives a more in-depth analysis than the one with only 5 bins. You can keep on increasing the number of bins as per your requirement and make sure that there are enough bins to predict the trends in the data correctly. For example, the final image with 50 bins is too granular and even has gaps in-between; this indicates that no records occur in such a fine-tuned bin range, and therefore it is of not much use to us.

Now that you've understood about binning in general, let's go ahead and plot histograms for the Reviews column. Here, you'll also use a box plot in conjunction with the histogram to obtain insights.

Here you utilised both a histogram and a box plot to solve the problem.[In general, when you're analysing a single numeric variable, both box plots and histograms come in handy]. The key results that you obtained were as follows:





This revealed that lots of pre-installed and superstar apps are present in the data. These apps won't be useful for our analysis since they already have skewed statistics (an extremely high number of installs and reviews), which is

not common for a majority of the rest of the apps. Hence, you took a qualifier of 1 million reviews and removed all the apps having more reviews.

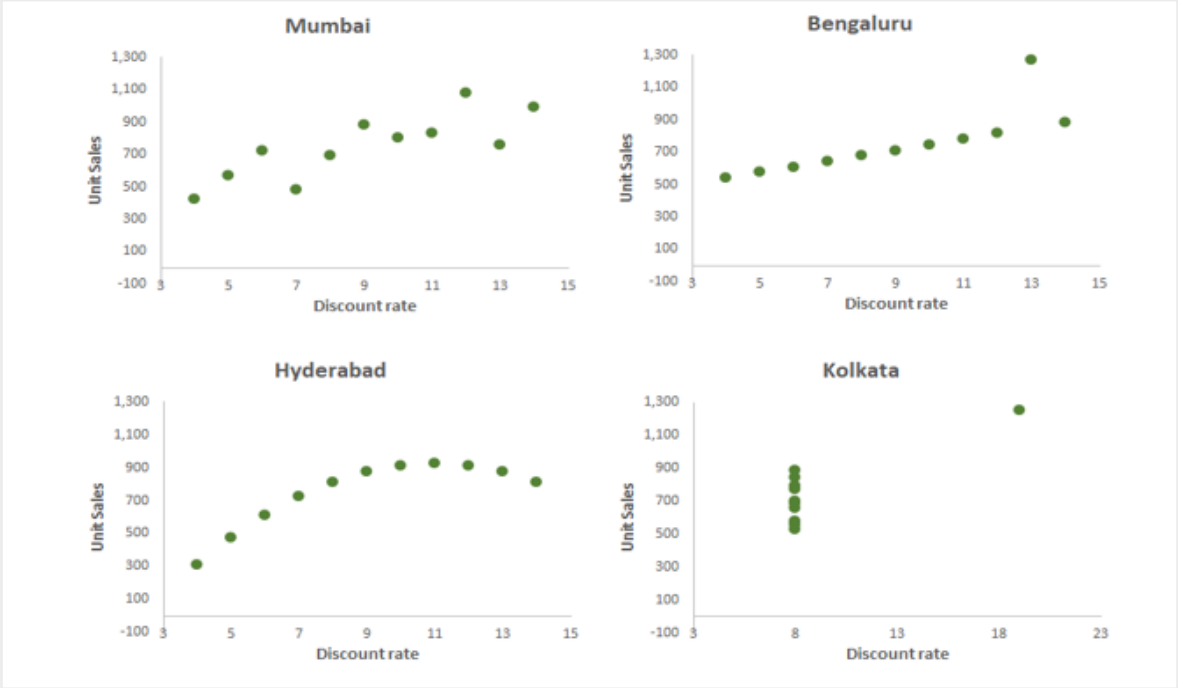
Now, go ahead and analyse the Installs and the Size columns as well in the following questions
Note that you can check the video solutions for verifying your approach.

Summary:-

You may decide to go through the solutions to the previous segment of practice problems before we summarize the learnings from the session.

Here's a summary of what you have learnt so far in this very intense session:

- First, you began with understanding the necessity of visualisation using the example of Anscombe's Quartet. There, you saw four different data sets having the same exact summary statistics but completely different visual plots.



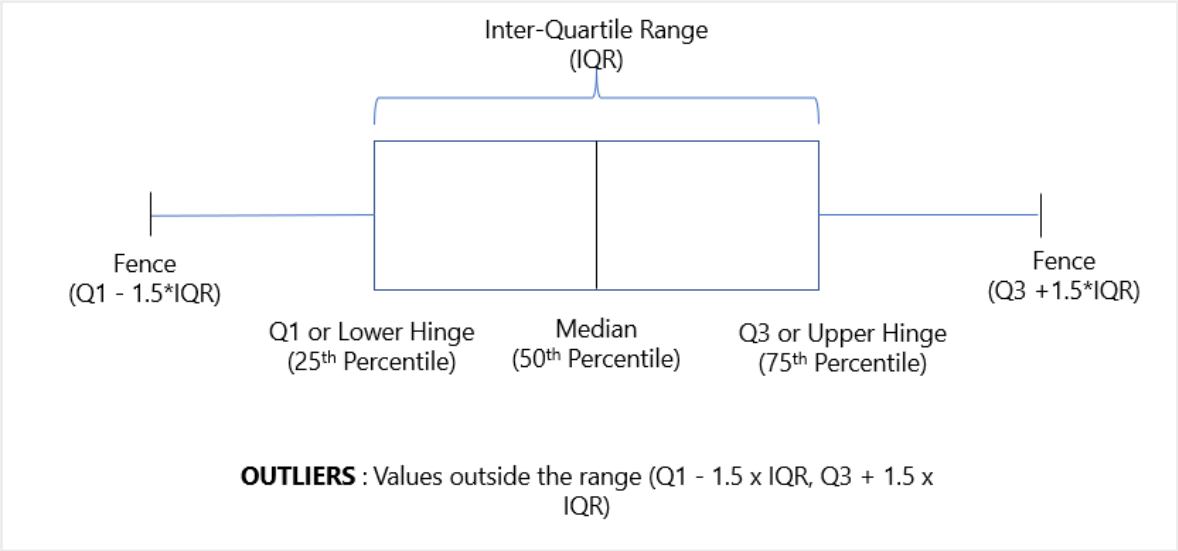
- After that, you saw some examples of data visualisation which enabled you to understand how effective visualisations can portray insights elegantly.

- Next, you began your data visualisation journey with the help of a case study. Here, you had to analyse the ratings of various Google Play Store apps in order to determine the features that decide whether an

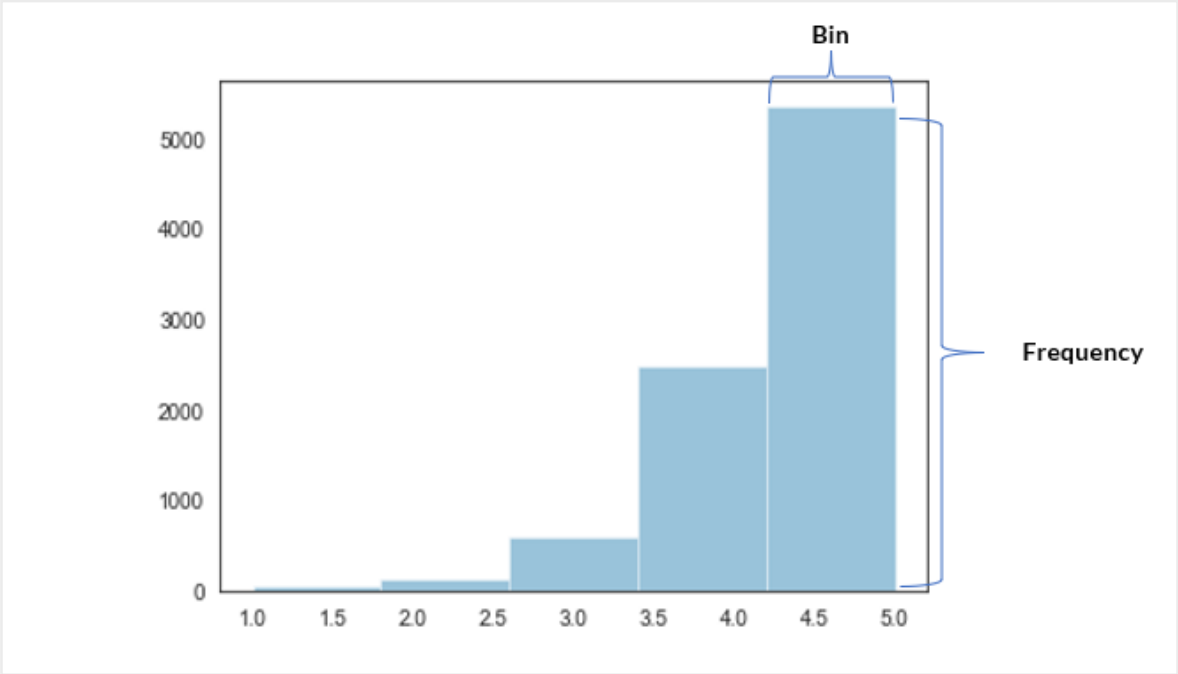
app is performing well.

- In the beginning, you performed some basic data-handling and cleaning tasks using your knowledge of Python libraries, which you have learnt earlier. You converted data types, cleaning certain columns containing erroneous entries, and even deleted records having a lot of null values.

- After that, you learnt how visualisation tools like box plots and histograms can help you with certain data-cleaning and analysis tasks. Box plots are required for understanding the spread of a numeric variable and identifying extreme values or outliers.

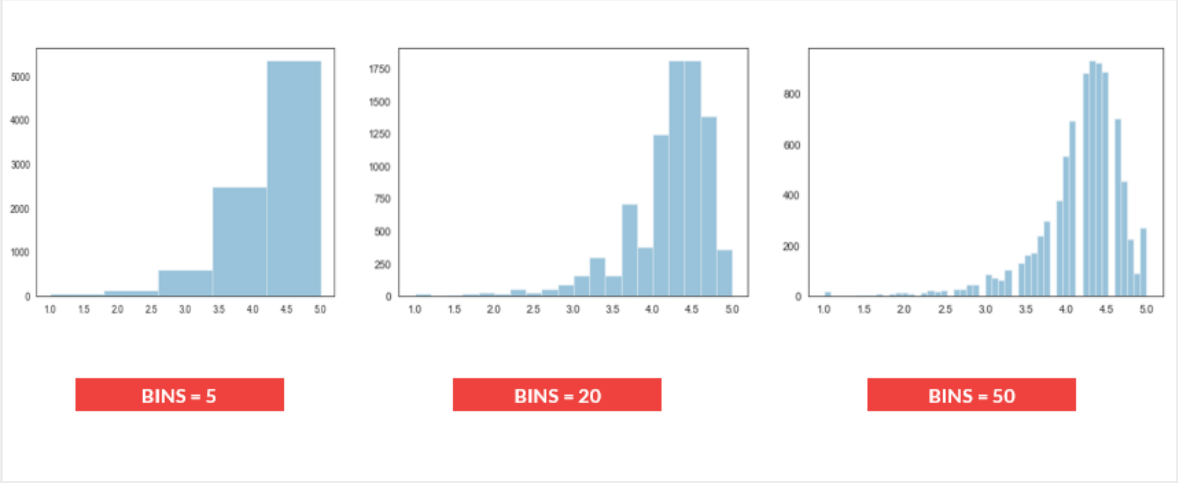


- After that, you learnt about histograms, and how you can use them for analysing a numeric variable by dividing them to certain bins and then finding the frequency, or the number of records in each bin.



-

- Then you also learnt how to manipulate the number of bins to obtain more granular results as shown in the figure below:



-

- You performed the above analysis for the Reviews, Size and Installs columns, and along with the box plot, you decided to keep only certain records for analysis.

Module 3 : Data Visualisation with Seaborn

Introduction:-

Welcome to the session on data visualisation with Seaborn library.

In the previous session, you understood the importance of data visualisation and were introduced to the various types of visual aids available to us for portraying insights. After that, you have started the visualisation journey with a case study: analysing the Google Play Store Apps rating data set. Here, you learnt a couple of key data-handling and cleaning tasks that need to be performed before you can start analysing your data and communicate insights with charts and graphs. You also learnt that box plots and histograms are quite useful not only for communicating insights but also for some basic data-cleaning procedures like outlier analysis.

In this session

You will learn about another library, Seaborn, which is used predominantly to create beautiful and aesthetic statistical plots in Python. Let's listen to Rahim as he introduces this library and explains its various features.

You can go through the official Seaborn documentation to gain further understanding of this library. You will learn about the importance of each visualisation as you go through the case study and derive the insights. **You will have to continue working with the same notebook that you had used earlier for the case study part and this needs to be worked on along with the videos to ensure that you are answering the questions along with videos.**

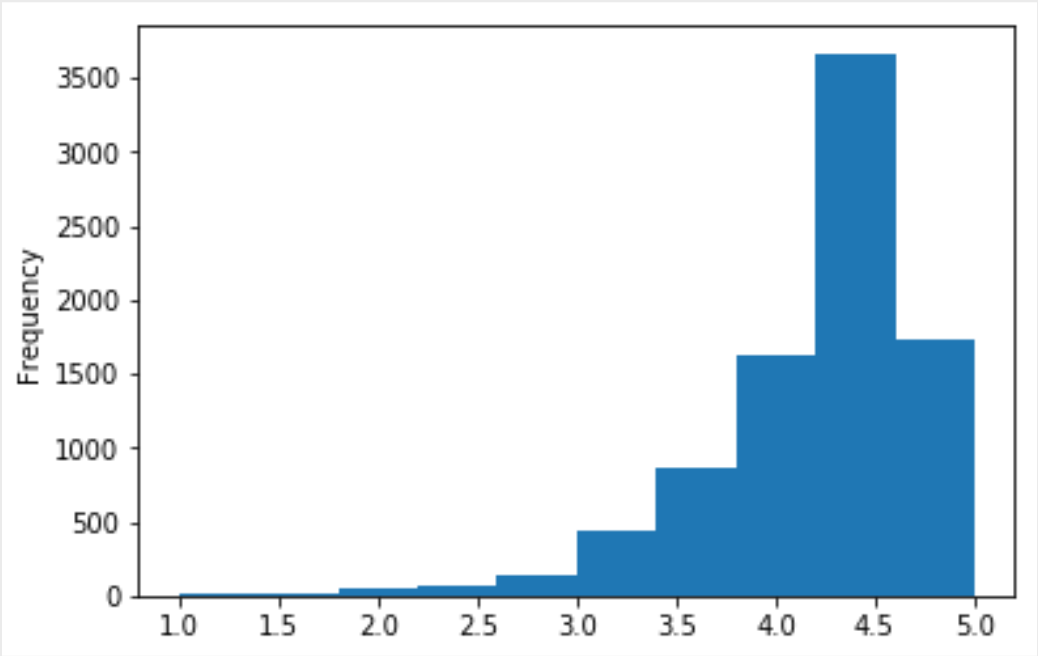
Distribution Plots:-

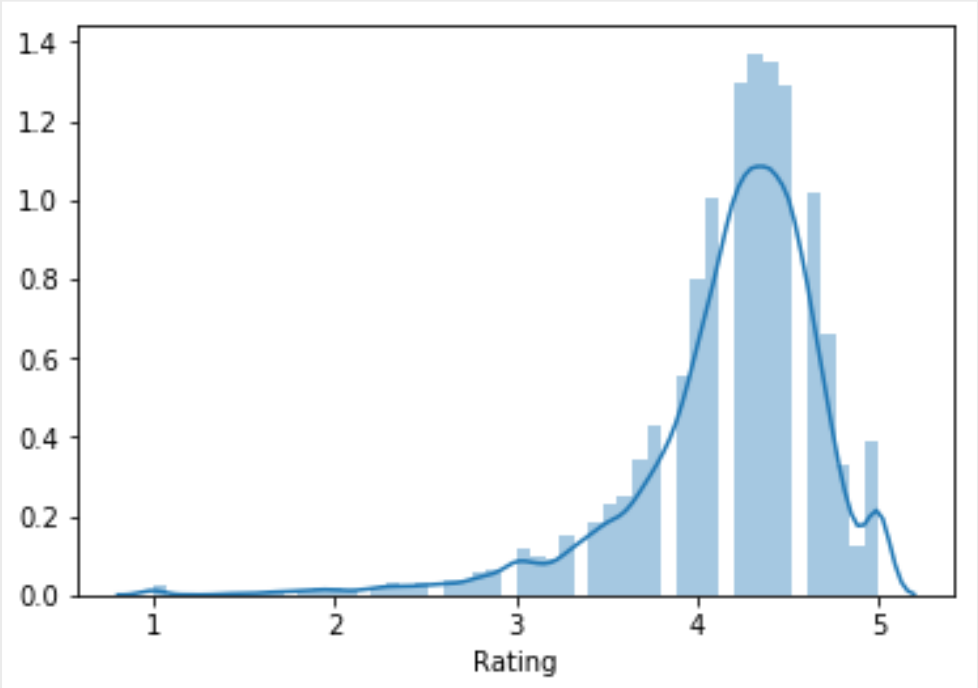
In the previous session, you learnt about the basic data-handling and data-cleaning tasks that were essential to be performed. In this session, you will begin the journey with Seaborn library and start extracting insights. Recall that the **target variable for this case study is the Rating column**. The main task is to analyse this column and compare it with other variables to observe how the ratings change through different categories.

First, you'll learn how to build a **distribution plot** for the **'Rating'** column, which is pretty similar to the histograms that you saw earlier in matplotlib.

So, you have plotted a distribution plot to check the distribution of ratings using both the Matplotlib function and the Seaborn functions. In the latter case, you must have noticed that instead of the hist command, you are now using a **distplot** or a **distribution plot**.The corresponding Seaborn command is ***sns.distplot(inp1.Rating)***.

You can go through distplot's documentation here to learn more about the various parameters that can be used. Notice that this view is quite different from the histogram plot that we had obtained earlier in Matplotlib.





The difference arises due to the fact that instead of calculating the '**frequency**', the **distplot** in Seaborn directly computes the **probability density** for that rating bucket. And the curve (or the **KDE** as noted in the documentation for Seaborn) that gets drawn over the distribution is the approximate **probability density curve**.*

Coming back to the visualisation, the bars that get plotted in both the cases are proportional. For example, the maximum frequency occurs around the 4-4.5 bucket in the histogram plotted by matplotlib. Similarly, the maximum density also lies in the 4-4.5 bucket in the distplot.

The advantage of the distplot view is that it adds a layer of probability distribution without any additional inputs and preserves the same inter-bin relationship as in the Matplotlib version. This statistical view of things is also far more informative and aesthetic than the earlier one.

You are expected to go through the Seaborn documentation from the link given above and answer the following questions.

So, after changing the number of bins to 20, you were able to observe that most of the ratings lie in the 4-5 range. This is quite a useful insight, which highlights the peculiarities of this domain, as mentioned by Rahim. If people dislike an app, they don't generally wait to give it bad ratings; rather, they go ahead and remove it immediately. Therefore, the average ratings of the apps are pretty high.

Also, you learnt about some more customisations that can be done on the same view. You can change the colour of the view and even use Matplotlib functionalities on top of Seaborn to make your graphs more informative.

Additional Notes:

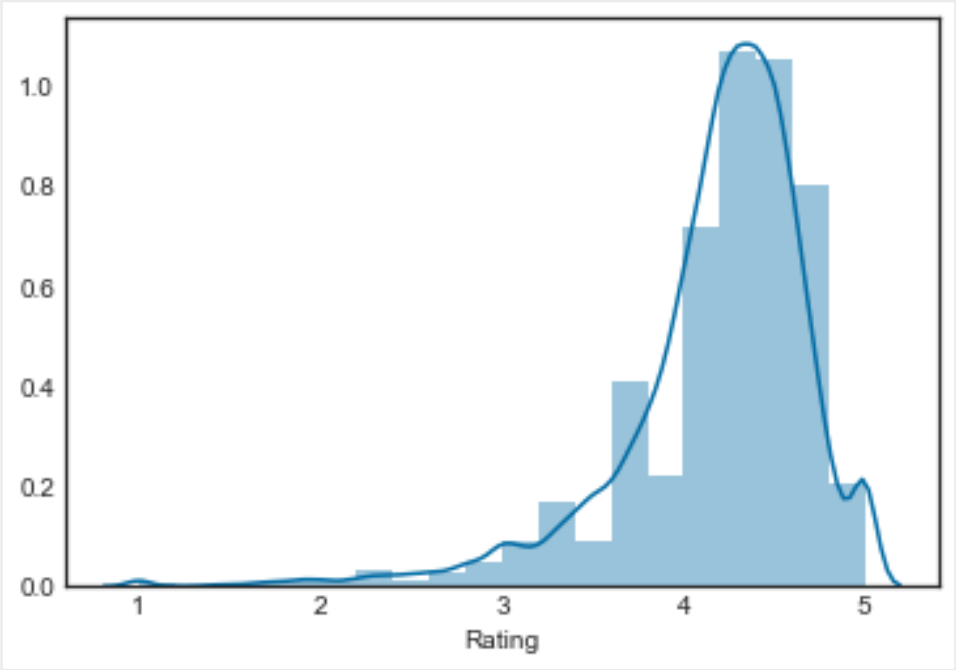
1. ***The terms "**Probability Density**" and "**Probability Density Curve**" may seem a bit alien to you right now if you do not have the necessary statistical background. But don't worry, you will learn about them in a future module on Inferential Statistics. However, if you're still curious, you can take a look at this link for further understanding.
2. Another chart analogous to the histogram is the **countplot**. It essentially plots the frequency of values for a categorical variable. Basically, the values are the same as when you take a value_counts() for that variable. Take a look at its documentation to understand how it is implemented.

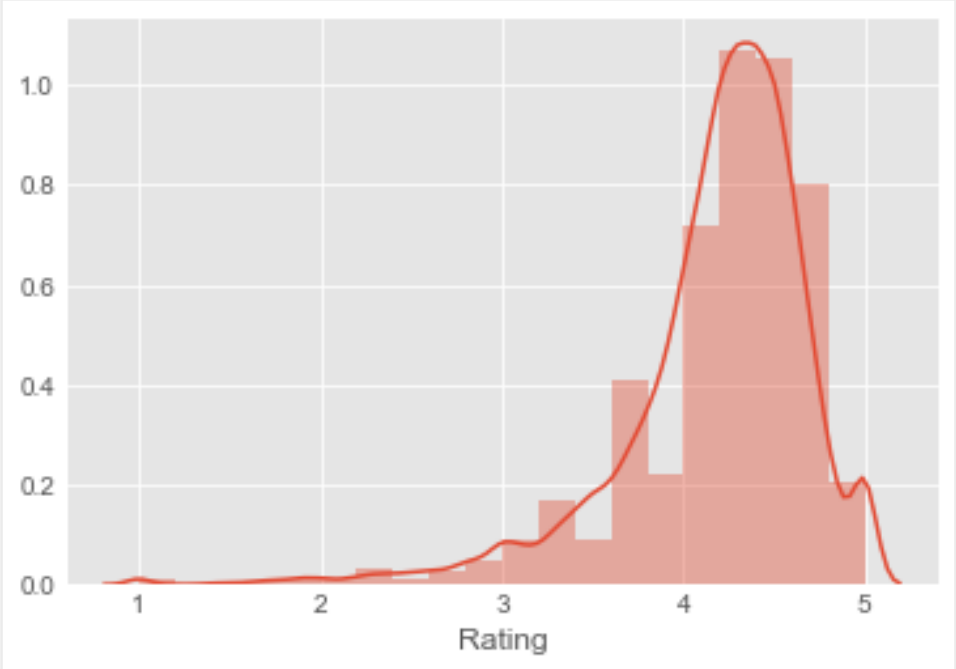
Now that you are reasonably proficient in creating a distplot and performing some basic customisations, in the next segment, let's dive even deeper into the different ways in which you can customise a plot.

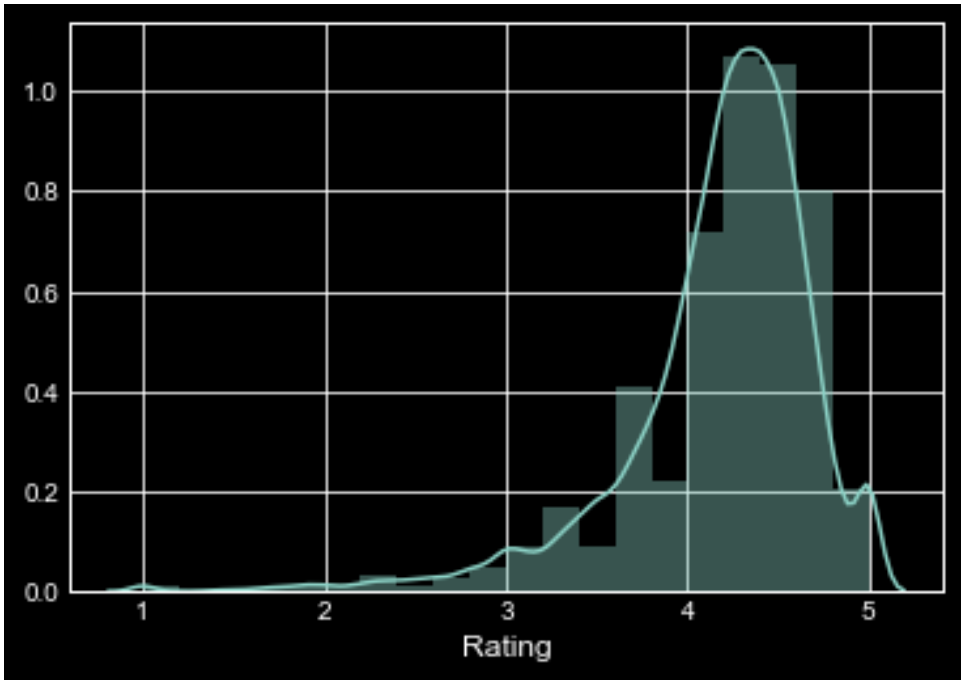
Styling Options:-

As discussed earlier, one of the biggest advantages of using Seaborn is that you can retain its aesthetic properties and also the Matplotlib functionalities to perform additional customisations. Before we continue with our case study analysis, let's study some styling options that are available in Seaborn.

As you just learnt, you can use several styling options by using the **sns.set_style() function**. This gives you control over the way the axes and grid are presented. Here's the link to its official documentation. Given below are certain style options that you can use for Seaborn in conjunction with the original customisations.


1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
Rating

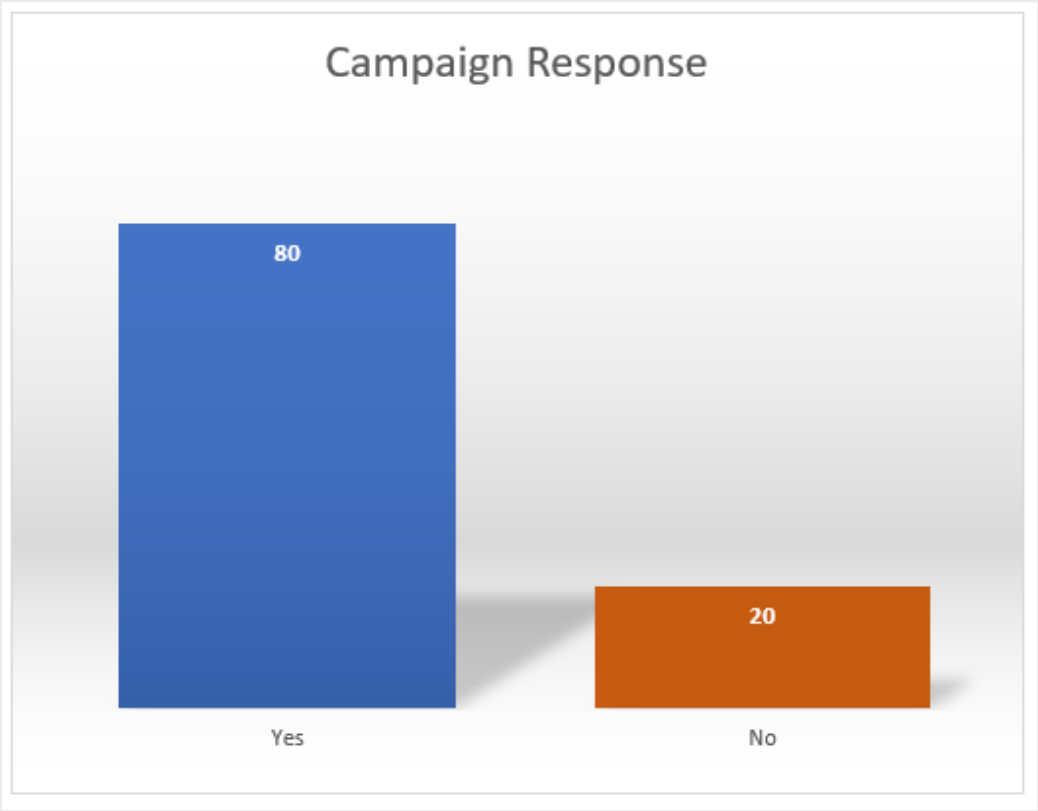

1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
Rating

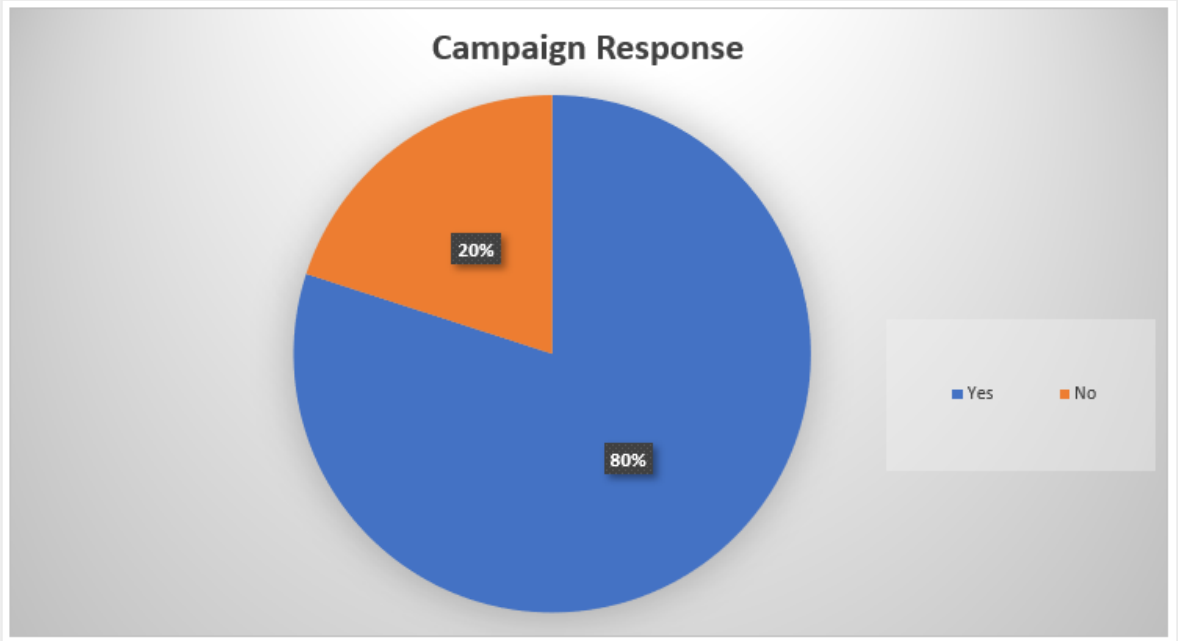


You should go ahead and try out the styling options and select the one that suits you the best and stick to it for the rest of the case study demonstration. In the next segment, you will explore the case study data using pie charts and bar graphs from the seaborn library.

Pie - Chart and Bar Chart:-

In the earlier visualisations, you're dealing only with numeric variables. Now you'll step into analysing the categorical variables and see how the Ratings vary across each of them. Note that in the case of categorical variables, you need to use aggregates or measures like sum, average and median to plot the visualisations. And then use plots like a bar chart or pie chart to portray those relationships. They are as follows:





You're already familiar with how to create bar plots in matplotlib. Here, you'll see how you can create bar plots and pie charts directly from the pandas series as well. Go through the documentation for both pie charts and bar plots. You'll also be doing a couple of data handling tasks here. So let's dive in.

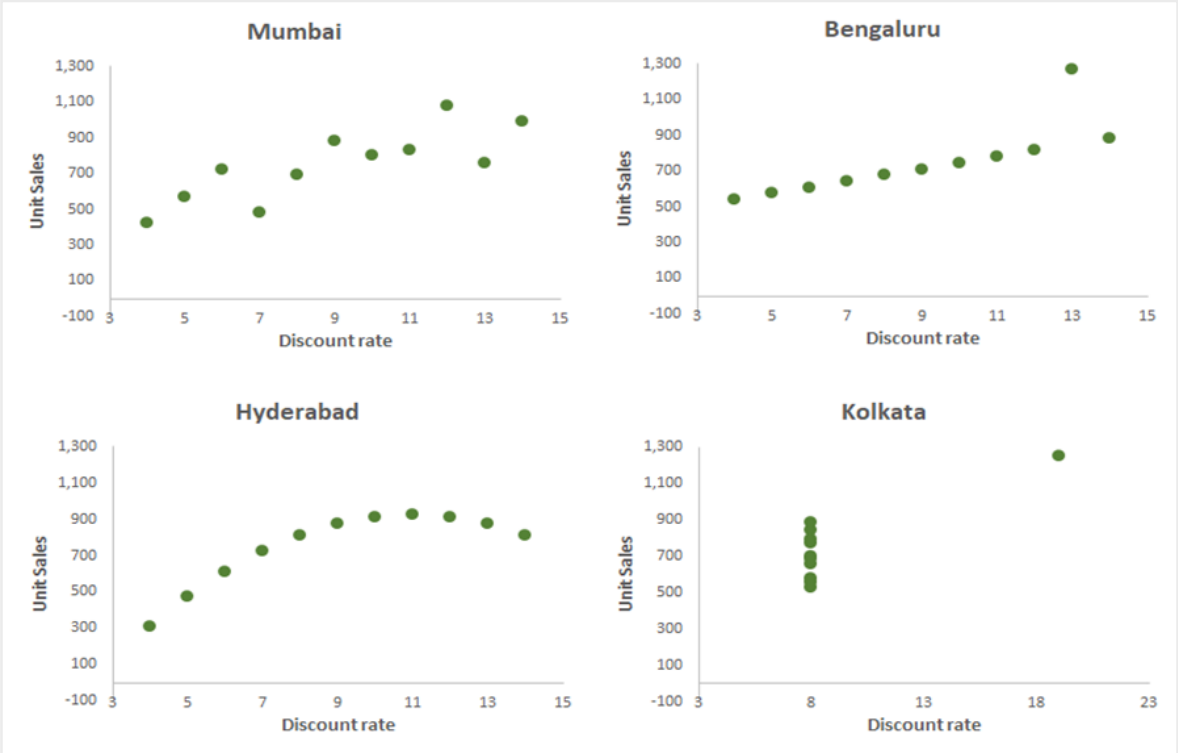
Scatter Plots:-

Previously, you had dealt with only a single numeric column and therefore used

either a box-plot or a histogram to portray the insights visually. What about two numeric columns, say Rating and Size? If you want to **plot the relationship between two numeric variables**, you will be using something known as a **scatter plot**. In the following video, Rahim will explain the situations when a scatter plot can be used.

Scatter plots are perhaps one of the most commonly used as well one of the most powerful visualisations you can use in the field of machine learning. They are pretty crucial in revealing relationships between the data points and you can generally deduce some sort of trends in the data with the help of a scatter plot.

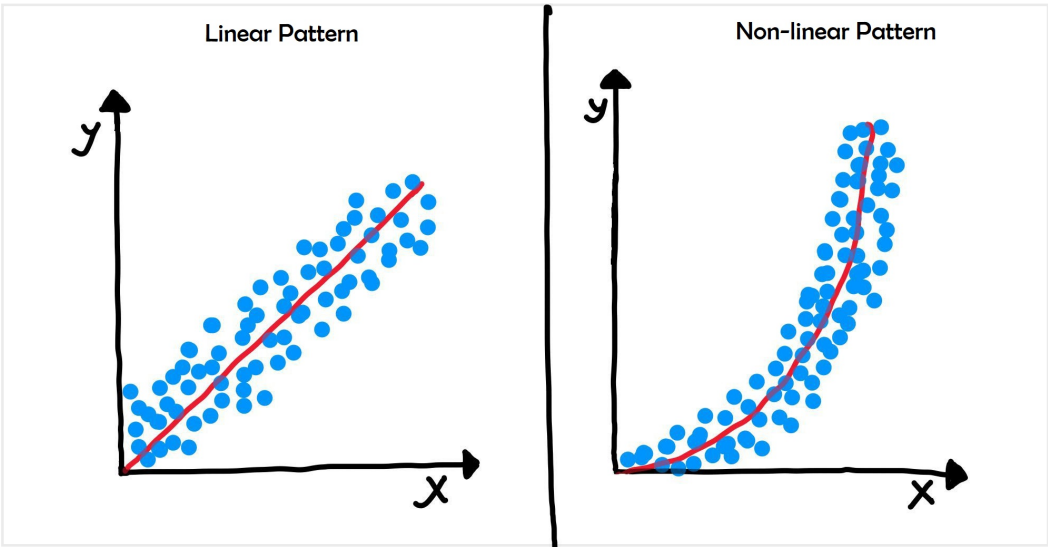
The "Sales and Discount" example that you had seen earlier at the beginning of the module is an example of a scatter plot (technically these are 4 different scatter plots, each of them showing a different city)



Applications of scatter plots in machine learning

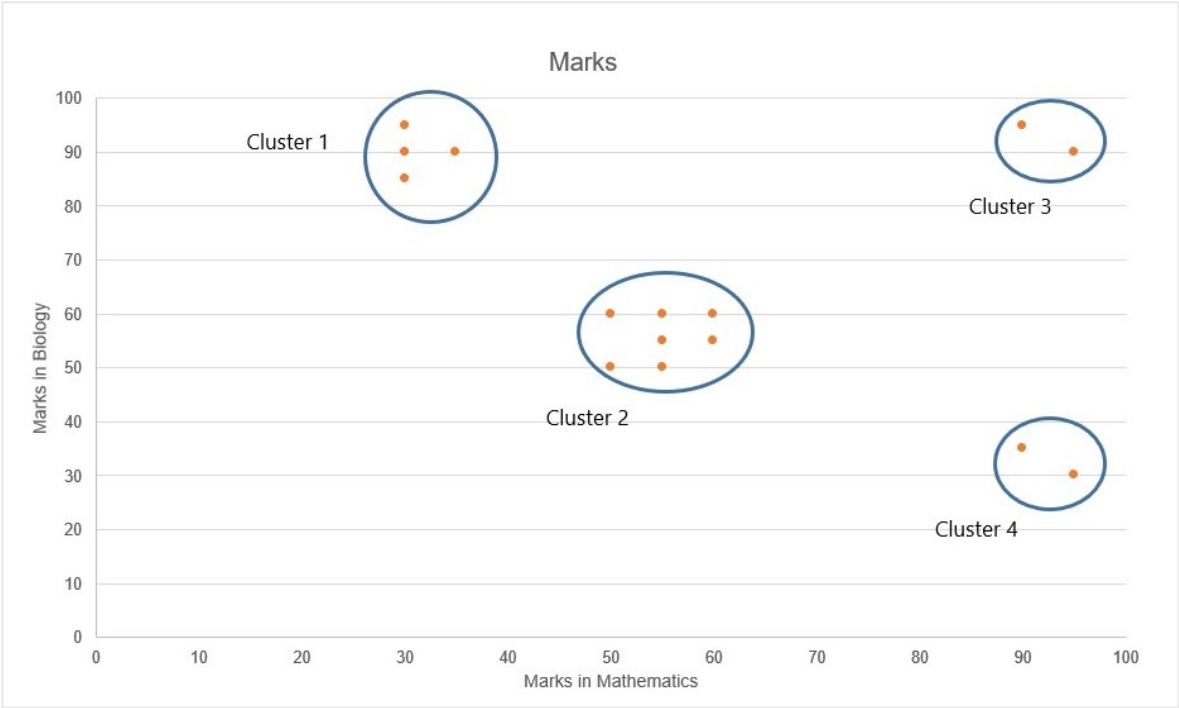
Even though you'll be learning about them in greater detail in future modules, it is good to know certain use cases where a scatter plot is immensely productive in the field of machine learning:

- **Observing trends between numeric variables**: Because scatter plots can reveal patterns in the data, they're a necessity in linear regression problems where you want to determine whether making a linear model, i.e. using a straight line to predict something makes sense or not. Check out the diagram given below.



- Making a linear model between x and y makes complete sense in the first case rather than the second one.

- **Observing natural clusters in the data:** In simple terms, clustering is the act of grouping similar entities to clusters. For example, let's say you have a group of students who have recently taken a test in Maths and Biology. Plotting a scatter plot of their marks in the two subjects reveals the following view:

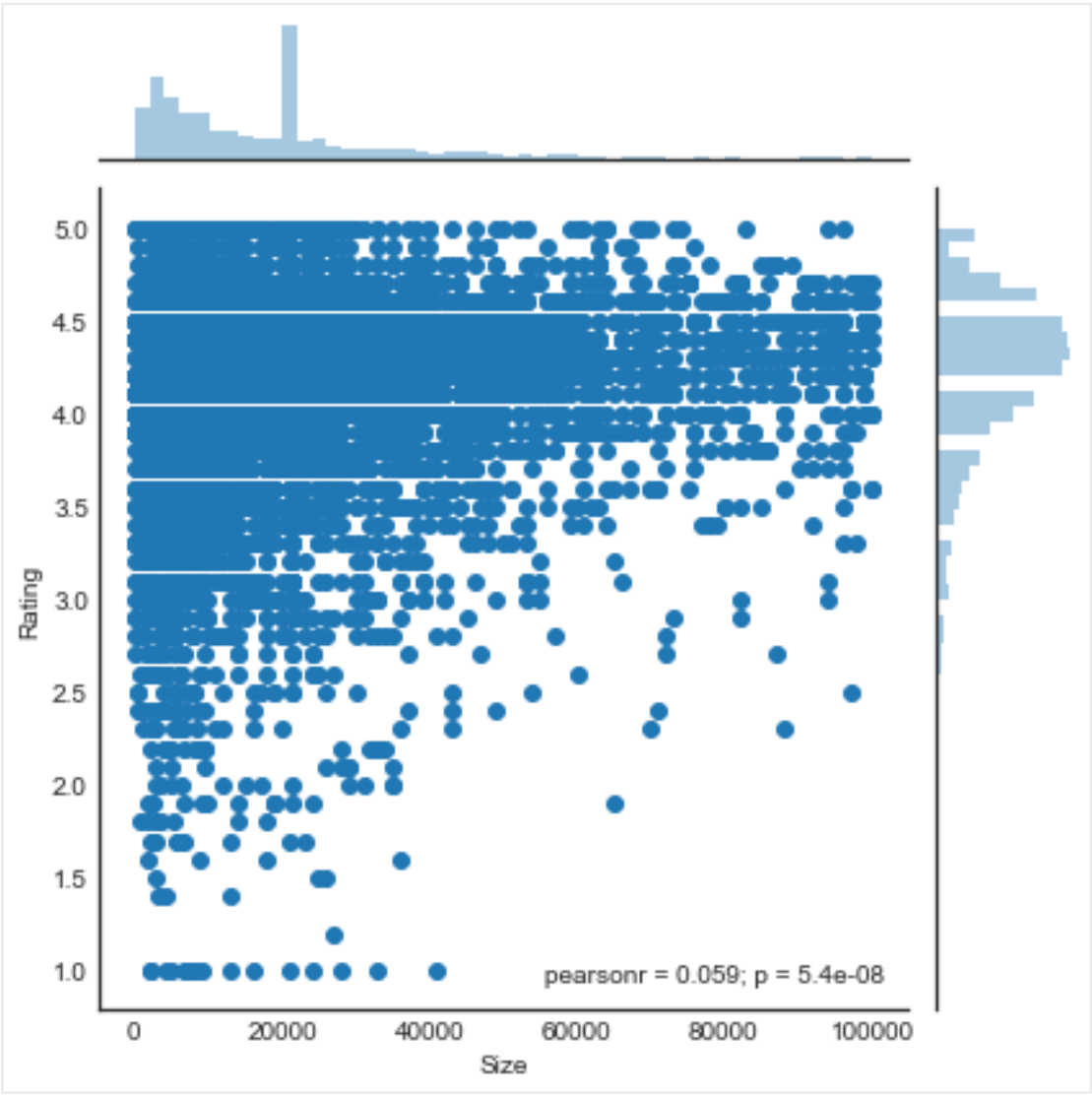


> You can clearly group the students to 4 clusters now. Cluster 1 are students who score very well in Biology but very poorly in Maths, Cluster 2 are students who score equally well in both the subjects and so on.

Now coming back to our problem, we're discussing plotting the scatter plot between Rating and Size. You already know how to do this in matplotlib using **pyplot.scatter()** function. In seaborn, we have the **sns.scatterplot()** which is pretty intuitive and similar to its matplotlib counterpart. You are advised to go through its official documentation to get an understanding of how the various parameters work.

However, in this case, you'll be using something called a JointPlot which combines the functionality of a scatter plot and also adds additional statistical information to it. Let's watch the next video to understand this further.

You utilised the jointplot() functionality of seaborn to plot it and observed the following results:



Important Note

In newer versions of seaborn, the Pearson r and the p value metrics may not be visible since they have been deprecated. We're suggesting the following workarounds. Please use them as per your seaborn version

Method 1 *(before seaborn 0.11)***:** You would have to annotate those values manually by importing the *scipy.stats* library and passing an additional parameter called *stat_func* in the jointplot code. Please check the code below to get a better understanding

```
#Import this library
import scipy.stats as stats

#Change the code to the following
sns.jointplot(inp1.Size, inp1.Rating, stat_func = stats.pearsonr )
```

Method 2 *(before seaborn 0.11)*: Here's a StackOverflow answer that describes another similar way to achieve the same thing

Method 3 *(seaborn 0.11 and above)*: For seaborn versions 0.11.0 and above, the above two methods won't work. Please use the following code snippet instead.

```
import scipy.stats as ss
#str1 will hold the value of Pearson r
str1 = 'Pearson r = {}'.format(ss.pearsonr(inp1.Size,inp1.Rating)[0].round(3))

#str2 will hold the p value
str2 = 'p value = {}'.format(ss.pearsonr(inp1.Size,inp1.Rating)[1].round(9))
plt.figure(figsize = (10,10))
sns.jointplot(inp1.Size, inp1.Rating)

#Manually align the position of str1 and str2
plt.text(-1250,1.25,
str1,horizontalalignment='center',verticalalignment='center',fontsize = 11 )
plt.text(-1250,1.0,str2,horizontalalignment='center',verticalalignment='center',f
ontsize = 11)
plt.show()
```

In addition to the normal scatter plot, the jointplot also adds the histogram of the respective columns to the mix as well. In this way, you can get an idea of the spread of the variables being discussed and therefore, make more succinct conclusions and gather insights from the data.

[Also, if you notice, there is the "Pearson r" and "p value" statistics information available to you as well. You'll be learning more about them in an upcoming module.*]

The syntax of jointplot is pretty similar to both the scatter plot syntaxes from seaborn and matplotlib. Take a look at the official documentation to learn more about the parameters.

The major insight that you got from the scatter plot is that there is a very weak trend between size and ratings, i.e. you cannot strongly say that higher size means better ratings.

Check the documentation of Jointplots and answer the following

question:

Regplots

When you are introduced to the seaborn library, it was mentioned that seaborn provides automatic estimation and plotting for regression setups for different kind of variables. Now regression would be dealt with in detail in future modules. However, it's good to know how seaborn uses a modified version of the scatter plots, also known as regplots to achieve this for now. Let's hear from Rahim as he explains this feature

In the next segment, let's learn to plot multiple charts together.

Additional Notes

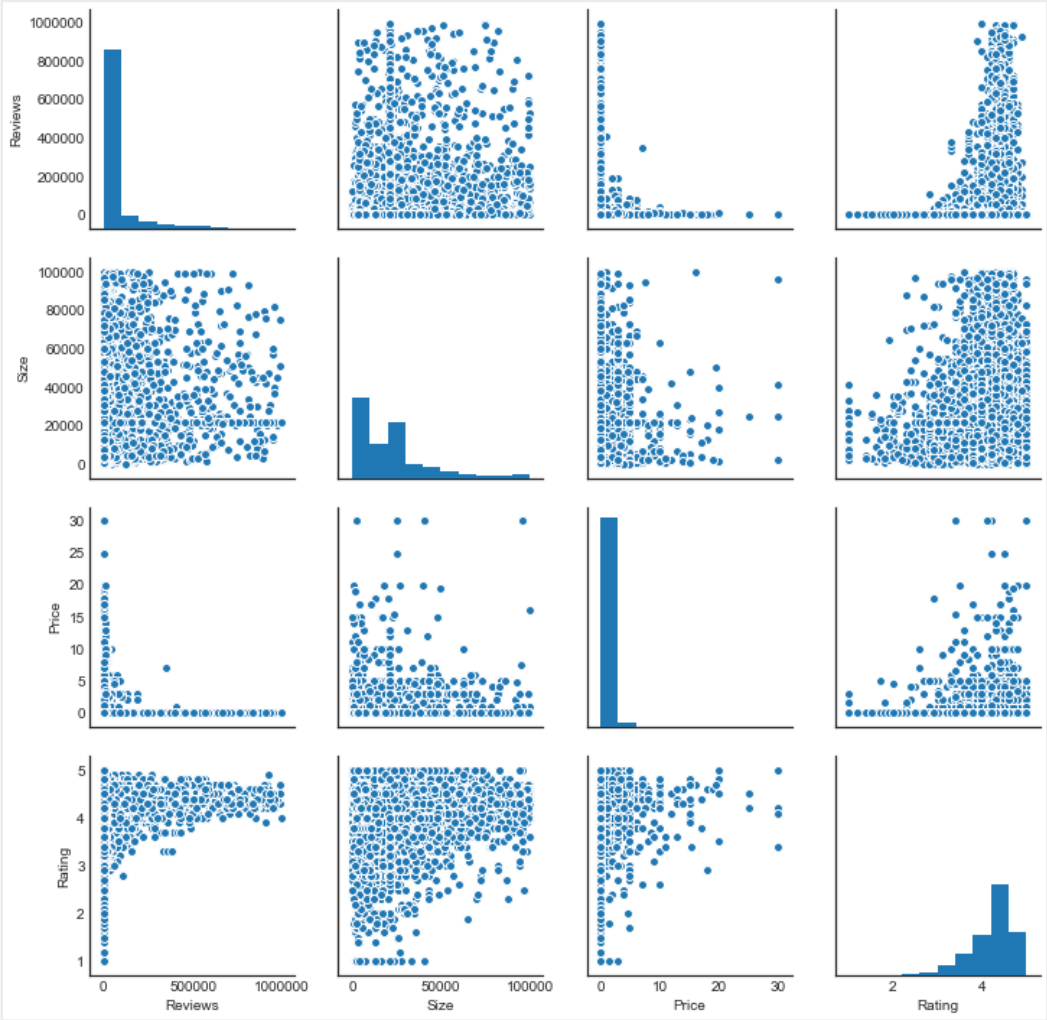
- * In case you're curious, **Pearson's r** value is a metric to measure the correlation between 2 numerical entities. You can read more about it in the following link.
- Scatter plots can show the trends for only 2 numeric variables. For understanding the relationships between 3 or more, you need to use other visualisations.
- Here's an article discussing the utilities of scatter plots.

Pair Plots:-

In the past segments, you learnt how to create scatter plots and joint points using matplotlib and seaborn. Using the sns.jointplot() you also saw how to create reg plots that provide regression functionality on the top of the scatter plot and histograms that are already available.

Now, in case there are 4-5 numeric variables that you want to analyse, making a jointplot for every 2 numeric variables is a bit tedious. To overcome this limitation, let's learn another functionality, the pair **plots**.

First, you took a subset of the entire dataframe - [Reviews, Size, Price and Rating] for the pairplot. Then you simply used the sns.pairplot() function to plot it. Check its official documentation for understanding its parameters.



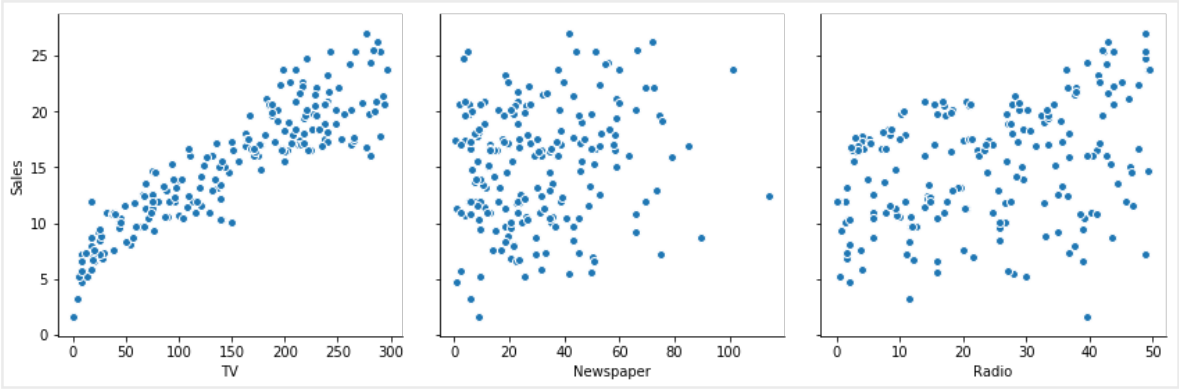
As you can see for every two numeric variables, the pairplot creates a scatter-plot whereas in the case for the diagonal ones, where the same variable is being considered twice, a histogram is shown.

Here, you're able to make certain inferences in conjunction with the ones made in earlier segments, like how Reviews and Price have an inverse relationship as the L-shaped scatter plot represents. Now, compared to the previous jointplot, you observe that the statistical information is a bit less(no Pearson coefficient to explain the correlation between the 2 variables) but nevertheless having a bird's eye view of all the numeric variables at once has its own advantages.

Application in Machine Learning

- Pairplots instantly give you the relationship between one numeric variable with the rest of the numeric variables. This is pretty useful in identifying relationships between the target variable and the rest of the features.
- For example, say you want to predict how your company's **sales** are affected by budgets allocated to three different types of advertisement channels - **TV, Newspaper and Radio**. In order to choose, you need to create a pair plot containing profits and the three

different budgets as the variables. Let's say the scatterplots of profits vs the three variables that you obtained from the pair plot are as follows (Click on the image to magnify it):



It is clearly visible that the left-most factor or budget allocated to TV is the most prominently related to the company's Sales since you can clearly ascertain a trend between them - ***increase in budgets for TV ads leads to more sales,*** whereas the points are scattered quite randomly in the latter two cases.

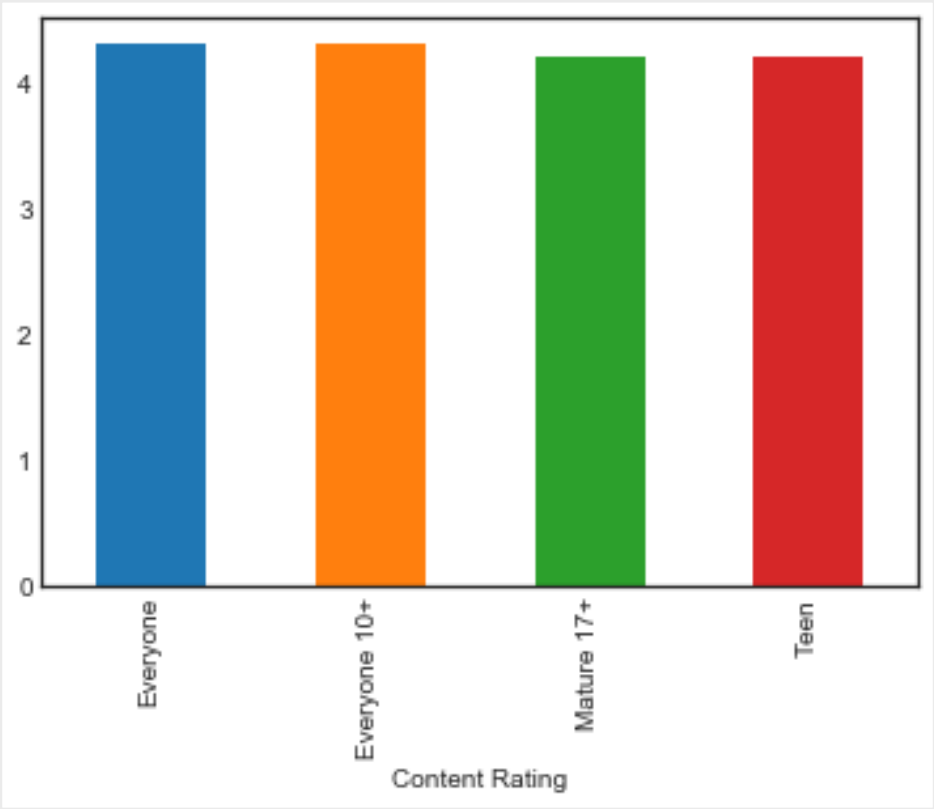
In the next segment, let's learn ways to add more information to bar graphs.

[**Note**: This concept of utilising pairplots to find insights will be dealt with in detail in future modules on EDA and Regression.]

Revisiting Bar Graphs and Box Plots:-

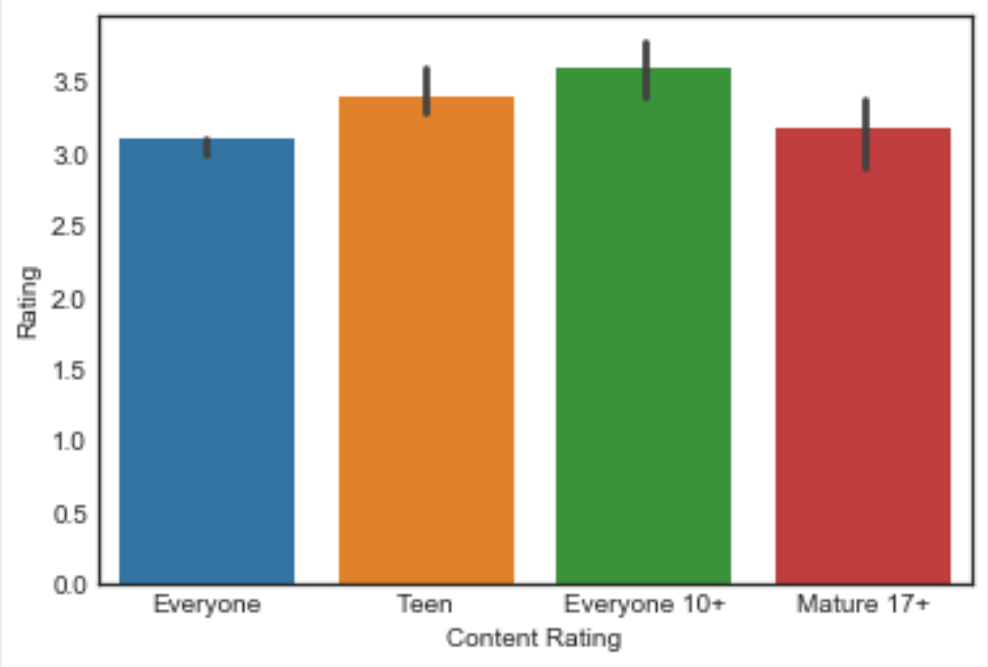
In the earlier sessions, you learnt how bar graphs and box plots can be utilised for analysing the numerical and categorical variables in your data. Now, you'll learn some additional customisations that Seaborn provides along with certain use cases where those functionalities come in handy. For this demonstration, you'll be taking a look at the Content Rating module.

Since taking just the average did not give us any insight, we decided to use the median metric. Here, you observed that the median value also did not prove to be a good differentiator for the categories being analysed.



Now, this is where you utilised Seaborn's **estimator function** to create bar graphs for different metrics (other than the median and mean) as you did earlier. In this case, you used the value at the 5th percentile to compare the categories and utilised the following estimator function for it: estimator=**lambda** x: np.quantile(x,**0.05**)

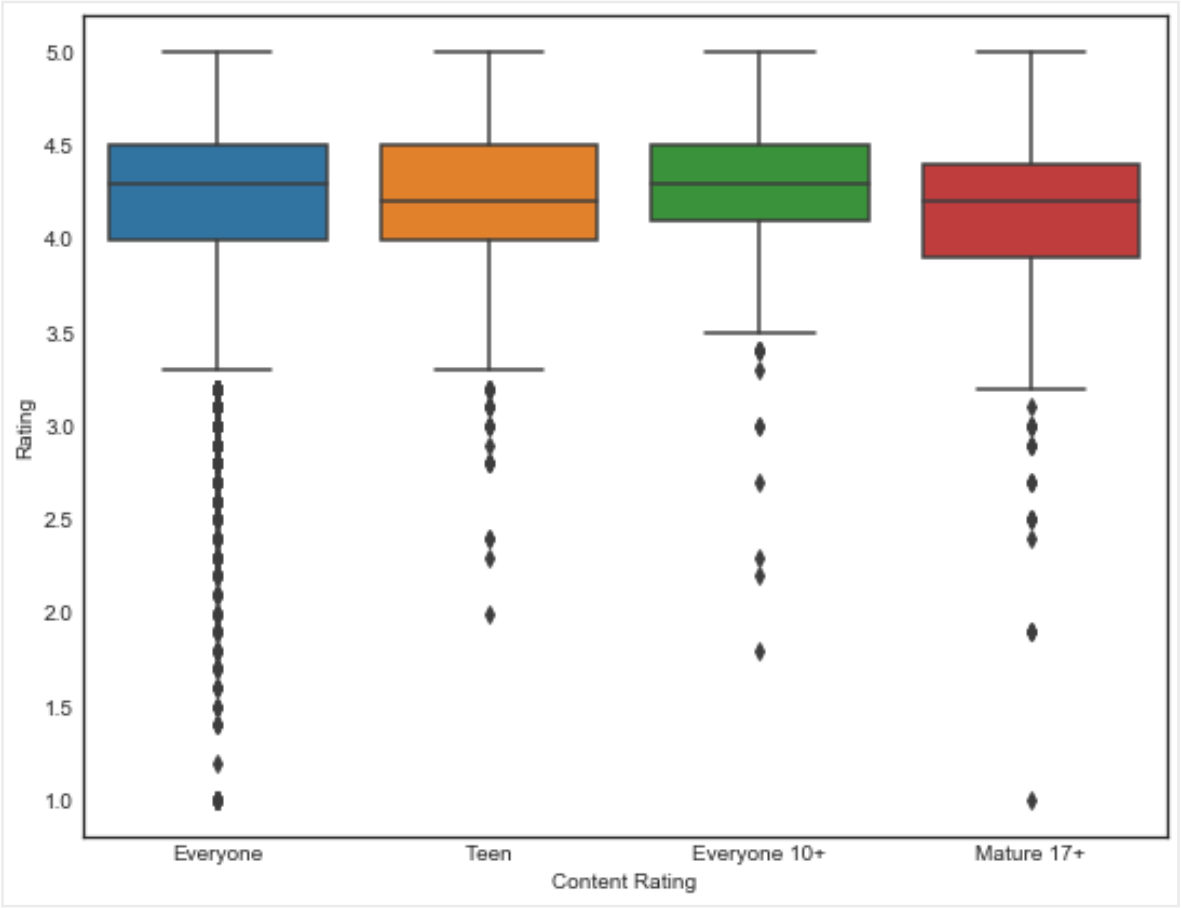
This yielded the following view:



Here, you can see some clear differences popping up: “Everyone 10+” has the highest rating at the 5th percentile (3.5), followed by “Teen” (around 3.3) and then “Everyone” & “Mature 17+”(around 3).
Using the estimator function, you can observe the values at different percentiles and compare the different categories.

Now, you must be wondering, rather than observing at specific percentiles, why not visualise the entire spread of ratings for each category using a box plot? Well, if you did , then good job! You’re thinking in the right direction. Rahim will be discussing that in the next video.

The following is the box plot of ratings for all different categories:



Here, you get a bird’s eye view of the spread of ratings for the different categories: median, 75th percentiles, fences, etc. The immediate insight that you obtained from the above view are:

- That “Everyone” category has the highest number of ratings in the lower percentiles as compared to the other categories.
- The median values are all comparable, which was discovered in the previous views as well.
- The upper fences for all the categories get capped at 5.0, whereas there are some observable differences in the lower fences.

Comparing Box Plots

As you saw in the above visualisation, comparing box plots of a particular measure for different categories helps you analyse the consistency and difference in spread between all the given variables. The **IQR or the inter-quartile range** serves a very useful purpose here in doing the same. Here's a video that explains how to compare different box plots to determine the most consistent performance.

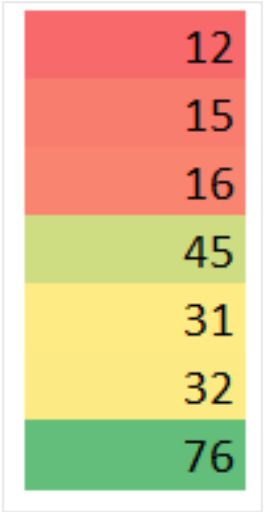
In the next segment, let's learn to plot heat maps. Heat maps help visualise the values in a matrix by colour coding them based on their values.

Additional Notes

- In the first use case of box plots, you observed how they can be used to identify and remove outliers from the data. In this segment, you understood how box plots can enable you to analyse a numerical variable across several categories. These two are the most prominent use cases of box plots that you'll be encountering from time to time as you proceed in this program.
- As you saw in the video, utilising the **groupby** function, the bar graph can be used to compare the mean, median, sum and several other metrics.

Heatmaps:-

If you've used MS Excel, then you must have come across conditional formatting, which utilises colour intensities to indicate the magnitude of the numerical value.



Heat maps also utilise the same concept of using colours and colour intensities to visualise a range of values. You must have seen heat maps in cricket or

football broadcasts on television to denote the players' areas of strength and weakness.

A heat map can be created as long as you have a rectangular grid of values. For the demonstration, you'll be seeing how to create a heat map for **Ratings/Size/ Content Rating**.

In the above video, you were introduced to the concept of **binning,** where you convert a numeric variable to a categorical variable by bucketing a specific range of values. This is pretty useful during analyses where you can create useful buckets and analyse how some other variable changes across those buckets.
One of the most common examples of binning happens in demographic survey datasets (like Census or Market research surveys) that contain the **Age** column, where people can be categorised as **Under-12**, **12-17**, **18-24** and so on.
Despite the actual age of the person being a numeric value, it's much easier to analyse across buckets and gather insights(*like asking how many people in the 12-17 age bucket have gone to school, how many of them prefer a particular brand over the other and so on*).

For binning purposes, you utilised the **pd.qcut** method, which divided the entire Size column to the following buckets on the basis of the percentiles. Note that pd.qcut takes percentile values in decimals, as in 20th percentile becomes 0.2, 40th percentile becomes 0.4 and so on.

Percentile Range	Binned Category
(0,20]	Very Low (**VL**)
(20,40]	Low(**L**)
(40,60]	Medium (**M**)
(60,80]	High(**H**)
(80,100]	Very High(**VH**)

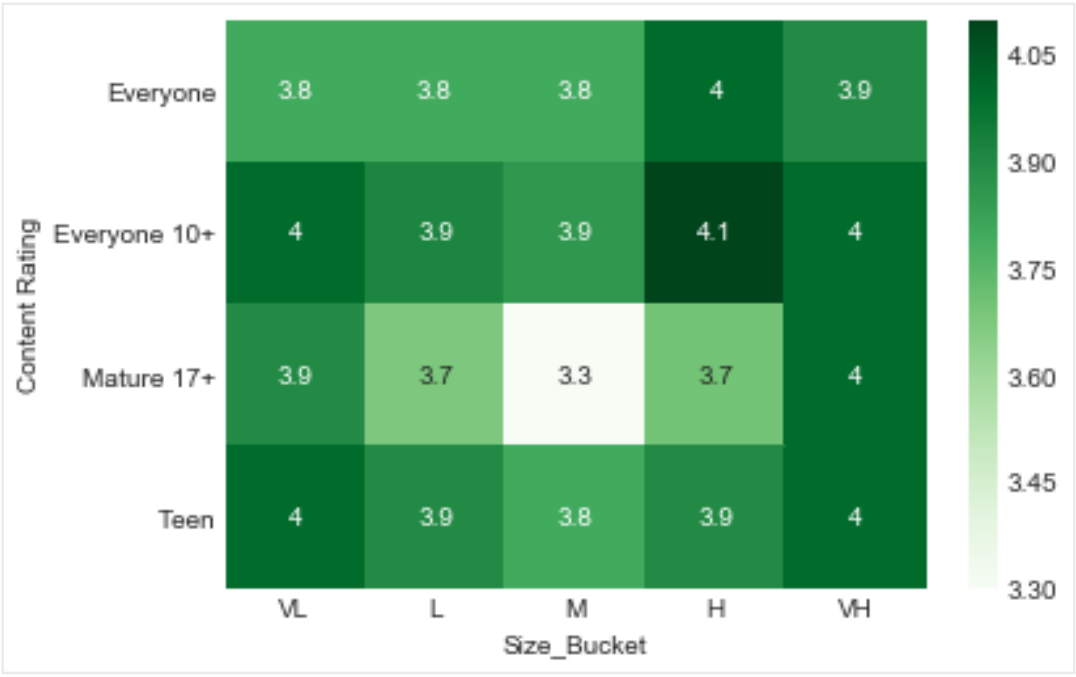
The above bins were now used to create the new column called **Size_Bucket** which stored the binned categories corresponding to the size of each app. Now finally when you prepare the pivot table (corresponding to the aggregation at 20th percentile for ratings), you'll get a grid as follows..

Size_Bucket	VL	L	M	H	VH
Content Rating					
Everyone	3.8	3.80	3.80	4.0	3.9
Everyone 10+	4.0	3.92	3.86	4.1	4.0
Mature 17+	3.9	3.68	3.30	3.7	4.0
Teen	4.0	3.90	3.80	3.9	4.0

..which is exactly what you need to create a heatmap!

Now that the pivot table has been created, let's go ahead and create the heatmap.

Once you've created a rectangular grid (either provided or made using the pivot table method taught earlier), use the sns.heatmap() function and pass the grid dataframe as the parameter.Mention some parameters like (cmap = "Greens", annot=True) to enhance its readability.
The final heat map that you obtained looked like this:



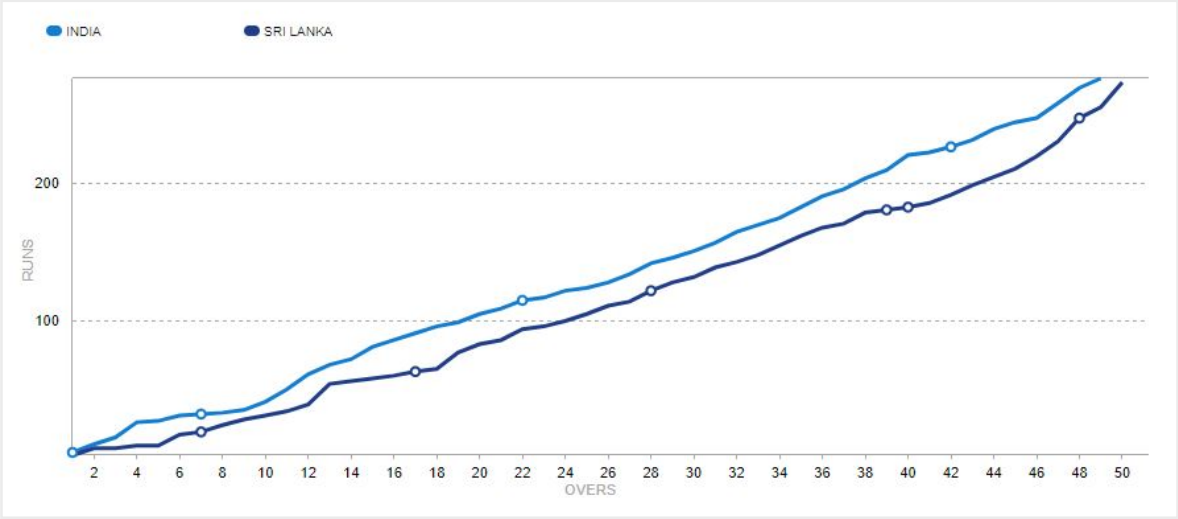
Note *- There's an additional question in the notebook where instead of Content Rating you'll be analysing Review Buckets using the q.cut approach mentioned above.*

Additional Notes:

- Heat maps are predominantly used in machine learning problems to visualise a Correlation Matrix, a grid that shows the correlation between any two quantitative variables. As mentioned in the additional notes of previous segments, understanding the correlation between variables is crucial for building and evaluating any ML model. You'll learn more about them in the upcoming modules.

Line Charts:-

Are you a fan of cricket? Then you must have observed those worm graphs depicting the pace at which the two teams were playing in a match at a particular instance of time. Here's an example of a worm graph, depicting the iconic World Cup 2011 final.



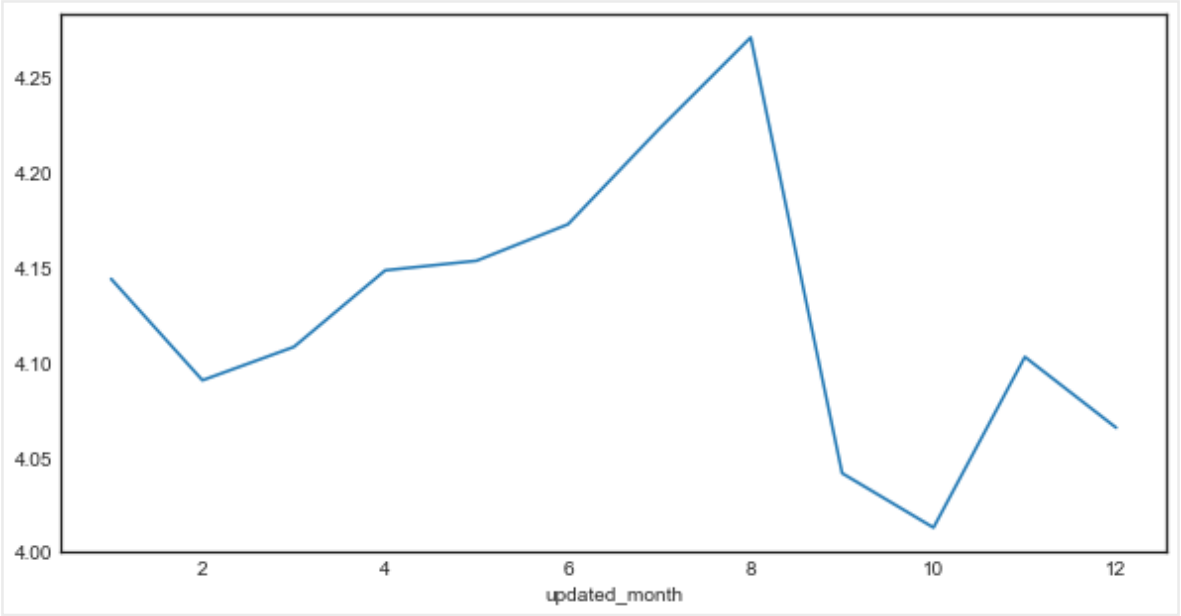
This graph is an example of a line graph (when drawn at such a scale it resembles a “crawling worm”). In the earlier session on matplotlib, you learnt what is a line graph and its main uses. Its main feature is that it utilises continuous time-dependent data to accurately depict the trend of a variable. In the next video, let’s see how you can build a line chart for the case study.

[Note: Earlier you used the parse_time parameter to index the date-time fields. In this demo, you’ll be using another pandas function pd.to_datetime. You’re advised to check its documentation before watching this video.]

Here are the steps you followed to create the line chart:

- You converted the date column to a date_time object using **pd.to_datetime**.
- After that, you found the average rating for each month using a groupby.
- Then, you used the plot function of matplotlib to create a line chart.

The following is the line chart obtained:



Though not quite significant, there is indeed some improvement in the ratings during the months of July-August. Note that, here, **we have assumed the Last Updated month to be the one in which all the reviews and ratings are coming from the users**.

Additional Notes:

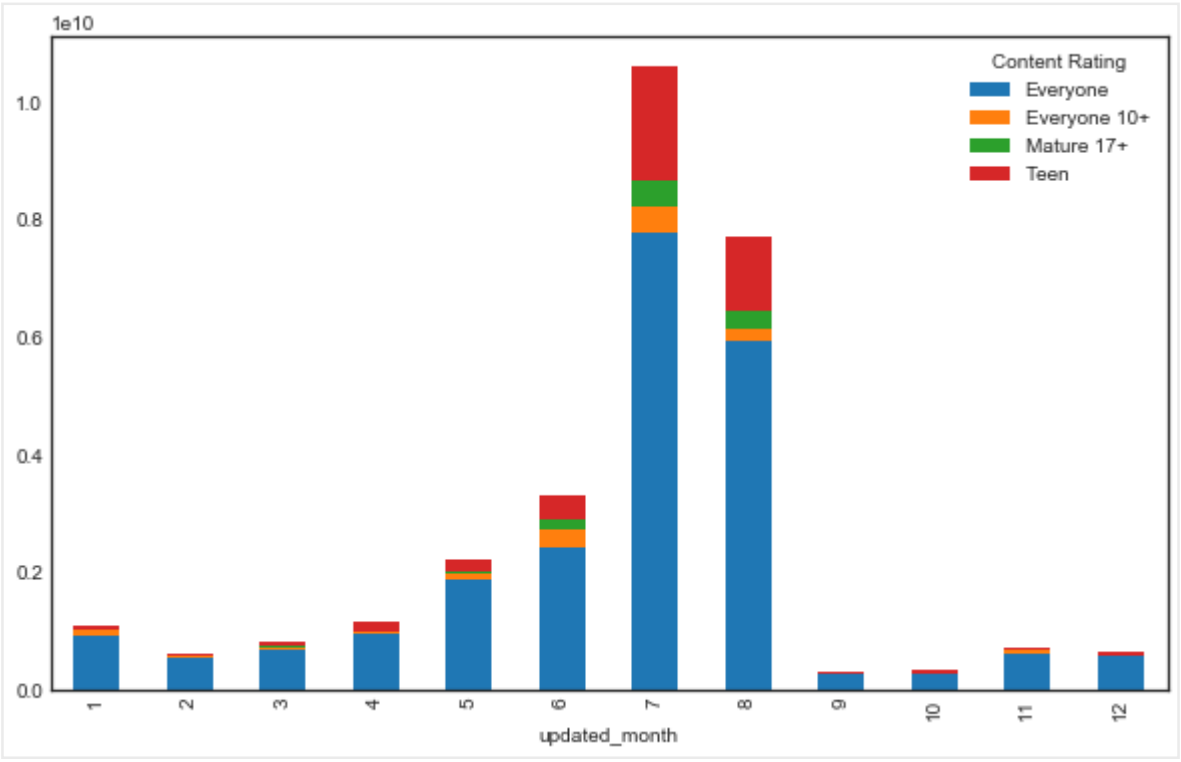
- Line charts are more or less utilised only for time-series data. Therefore, you'll be using them predominantly while working on forecasting and other time series models.

Stacked Bar Charts:-

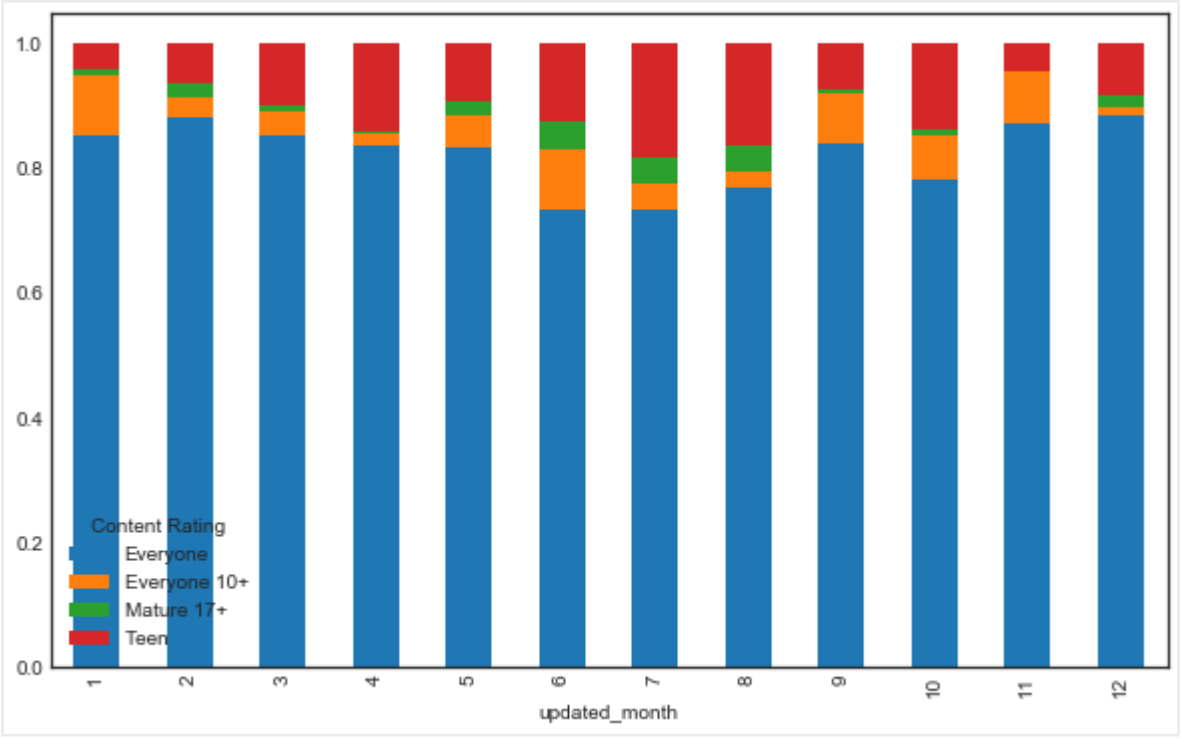
Earlier, you learnt how to analyse categorical variables using a bar chart. You can also add another categorical variable to the mix and analyse it even further with the help of a stacked bar chart. In the following demonstration, you'll be creating a stacked bar chart by comparing Installs across the different months and different categories of Content Rating.

To create a stacked bar chart, you need to follow these steps:

- First, create a pivot table with the updated month and Content Rating as the rows and columns and the "values" parameter set to the number of Installs.
- Now, plot a stacked bar chart by using the plot() function from matplotlib. Also, set the stacked parameter as True.



Here, even though you can say that the months June-Aug has the highest number of installs, it is quite difficult to infer anything about the different Content Rating categories. To overcome this, you set all of the different types(Content Rating)of installs in proportion to their monthly installs:



Case Study Summary:-

So, this marks the end of the case study. Here's what you've learnt in the past two sessions.

Summary of case study:

- First, you did a fair bit of data handling and cleaning - cleaning junk records, adding missing values, changing data types, remove outliers, etc.
- When you analysed the ratings using the histogram, you saw that they are skewed towards higher ratings.
- Using a bar chart, you saw that most of the apps belong to the Everyone category.
- You also observed a weak trend between the ratings and the size of the app, using a scatter-plot. You also briefly forayed to reg plots to understand its nuances.
- Using a pair-plot, you were able to see multiple scatter plots and draw several inferences, for example, price and rating having very weak trend, reviews and price being inversely related and so on.
- After that, you utilised estimator functions along with bar plots as well as box plots to observe the spread of ratings across the different Content Rating Categories. Here, your main observation was that Everyone category has a lot of apps having very low ratings.
- Finally, you created a heat map comparing the ratings across different Reviews and Content Rating buckets.

Plotly:-

Plotly is a Python library used for creating interactive visual charts. You can take a look at how you can use it to create aesthetic looking plots with a lot of user-friendly functionalities like hover, zoom, etc.

[First, you need to install Plotly on your local machine. Take a look at this link for instructions as well as documentation for the library.]

So, as you saw in the video, Plotly offers a lot of interactive utilities to the user. You should definitely experiment and learn about the different types of visualisations that are available here.

This marks the end of the case study. You can go ahead and download the completed sample notebook from the link provided below: