**Introduction to data visualization**

**What is Data Visualization?**

Data visualisation is a **concept** of graphical representation of data or information using visual elements like graphs, charts and maps. This representation helps us in understanding patterns, trends, and outliers in the data.

With the increase in volume of the data, discovering patterns in the data becomes challenging. Through data visualisation, a huge chunk of complex data can be displayed to be easily comprehensible as well as pleasing to the eyes.

Data visualisation:

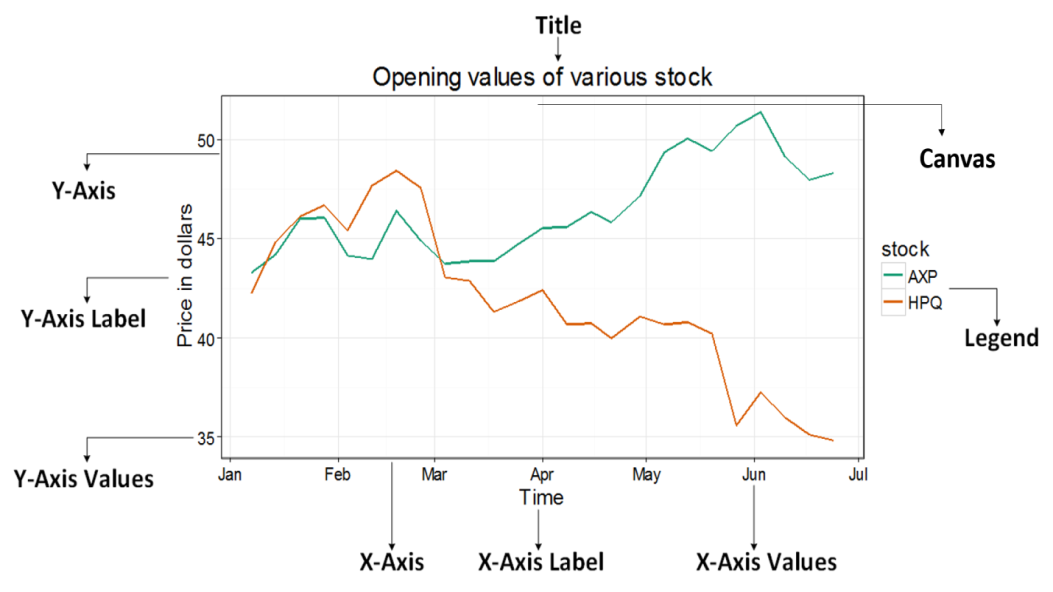
* helps in finding patterns and connections between variables
* requires less effort from the reader to understand the visuals
* condenses a large amount of information into a small space for quick analysis
* provides relevant answers and clarity on certain questions swiftly

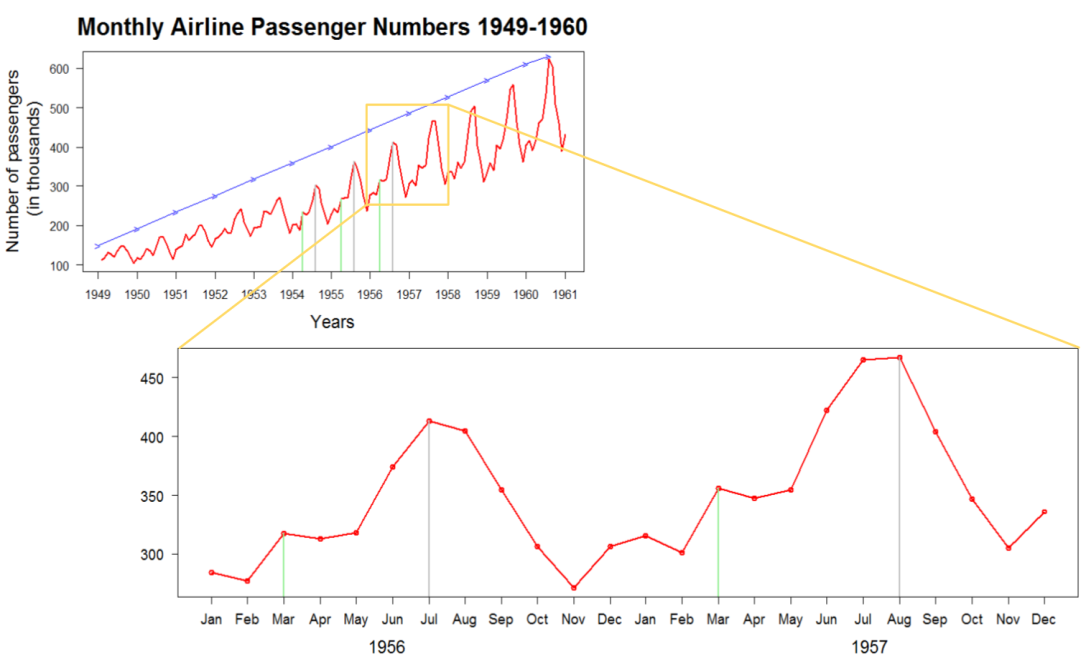
Thus, data visualisation finds applications across various domains.

One of the common graphical techniques to represent data is a plot.

**Plots** provide a pictorial representation of the relationship between two or more variables in a dataset. They are important in statistics and data analysis for deriving insights from large datasets. Some of the examples of plots include box plot, scatter plot, line chart, bar graph, histogram, etc.

Key elements of a plot are as shown below:



Applying data visualization techniques on the sample dataset the following plot is generated:  


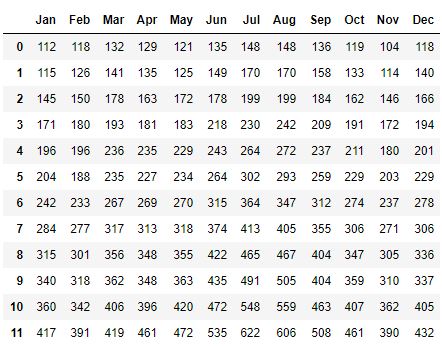
You can now observe with ease that:

* number of passengers has increased over the years (refer the first graph above)
* there is a peak in the middle of each year during the months of July and August (refer the second graph above)

Thus, visualization helped in finding the patterns quickly. This is an effective technique for finding patterns in large datasets as well.

**Why Data Visualization**

Let us understand the importance of data visualisation using a sample dataset which consists of total number of passengers traveling each month on an international airline between 1949 and 1960.

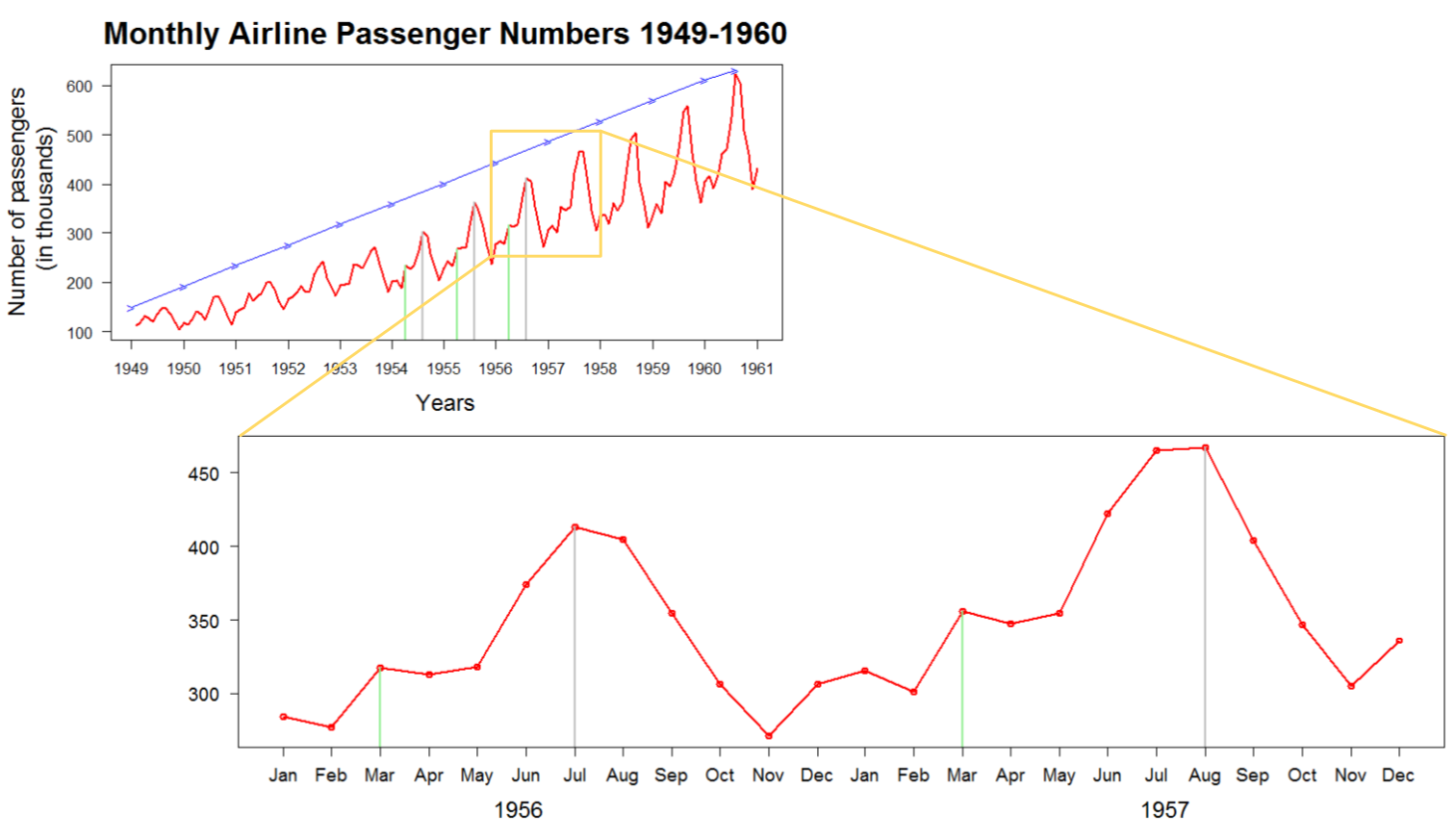


As an analyst, you would be interested in finding patterns in the given data such as:

* number of passengers travelling on an yearly basis
* repeatedly observable pattern in the number of passengers traveling every year

If the data consisted of millions of rows and hundreds of columns, then finding such patterns in it might be time-consuming, which is not desirable.

Applying data visualisation techniques on the sample dataset the following plot is generated:



**Data Visualisation Stakeholders**

There could potentially be two types of visualisations based on the types of stakeholders involved.

1. For self-consumption during data exploration, feature engineering, etc.
2. For presenting or communicating the insights (from the data) with a target audience, typically decision makers. This sort of visualisation is usually performed to prepare the final results/reports that may enable the target audience in decision making.

**Variety of Data**

In the digital world, there has been exponential growth in data collection. The visualisation and the analysis of this data can provide insights that can be used for business benefits.

The different types of data collected from various sources are as follows:

* **Temporal Data:** Data with a time component attached to it. For example, opening and closing values of stocks in a year. A plot that can represent the sequence in this data and the pattern changes over time is required.
* **Geospatial Data:** Data with a physical location as an attribute. For example, location of volcanoes around the world. A plot that can represent this data on a geographical map is required.
* **Topical Data:** Data concerned with topics. For example, feedback from customers. A plot that can represent the relationships in this data is required.
* **Network Data:** Data in the form of nodes and links between nodes. For example, social networking data. A plot that can represent the relationship between nodes is required.
* **Tree Data:** Data which is basically network data but with some hierarchy in it. For example, organisational structure. A plot that can represent the tree structure is required.

Generally, data can be of two types:

1. **Qualitative/Categorical Data:** Data that deals with characteristics and descriptions. It is further categorised as:

* **Binary:** Data that is dichotomous. For example, True/False, Yes/No, 1/0 etc.
* **Nominal:** Data with no ordering or ranking. For example, different colors, blood groups, nationality etc.
* **Ordinal:** Data with specific order or ranking. For example, height (short, medium, tall), income (low, medium, high), etc.

1. **Quantitative/Numerical Data**: Data that is numerical in nature and can be measured. It is further categorised as:

* **Discrete:** Data that can be counted (whole numbers). For example, number of floors in a building, number of students in a classroom, etc.
* **Continuous:** Data that can take any value within a range. For example, weight, mileage of a car, etc.

Thus, in data visualisation, different types of plots are required to represent the different types of available data and to fit the needs of various stakeholders.

**Visualisation Constructs**

The different types of plots that can be used for data visualisation are listed as follows:

* Box plot
* Scatter plot
* Line chart
* Bar graph
* Histogram
* Distplot
* Pie chart
* Joint plot
* Pair plot
* Heat map

Let us discuss each of them one by one.

Data analysts perform exploratory data analysis to handle missing values, outliers, etc. and analyse the relationship between the variables. Data analysts require the knowledge of statistical concepts used in data analysis to:

* select the right type of graph
* infer information like outliers, correlation of variables, redundant features etc.

In this section, few statistical concepts will be explained to help you understand different graphs.

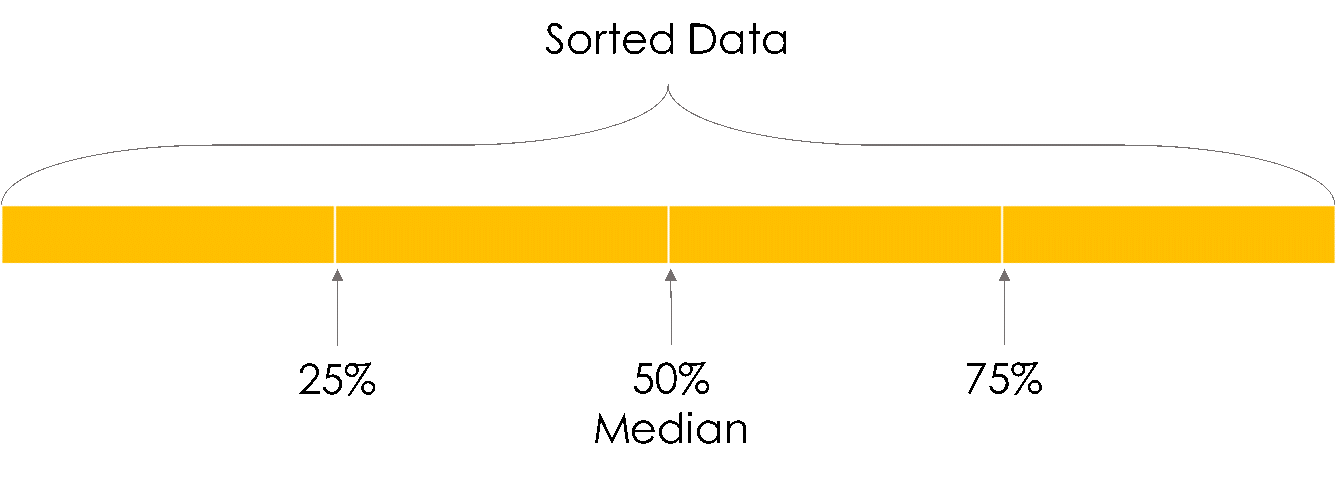
**Outliers**

Outliers are the extreme values present in the dataset. They affect the properties of data like mean and variance which are used in model building. Hence, they may impact the accuracy of the model.

So, the question that arises is, how to know if a value is an outlier? And how to deal with such values? Let us find out.

**Quartiles**

Quartiles divide the number of data points into four equal-sized groups, or quarters.



Following are the steps to find quartiles:

1. sort the dataset in ascending order
2. find median of the sorted dataset (median divides the dataset into two halves - Quartile 2 or Q2)
3. repeat step 2 with the first and second half of the data (this gives Q1 and Q3, dividing the dataset into four equal parts)

With the help of quartiles, a value called Inter-Quartile Range (IQR) can be calculated using the formula:

IQR = Q3 - Q1

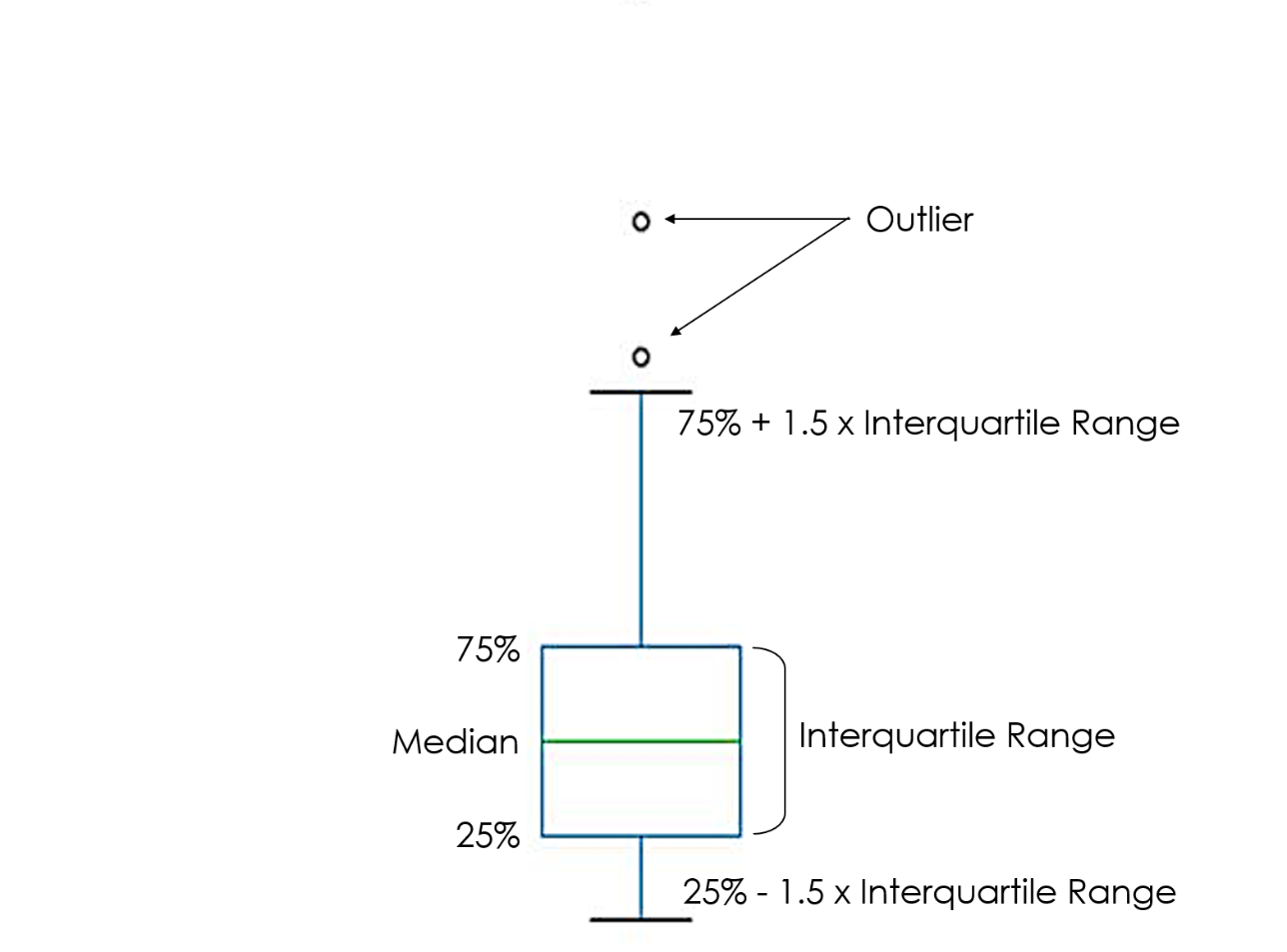
**Inter-Quartile Range**

Inter-Quartile Range also called mid-spread, H-spread, or IQR, indicates where most of the data is lying.

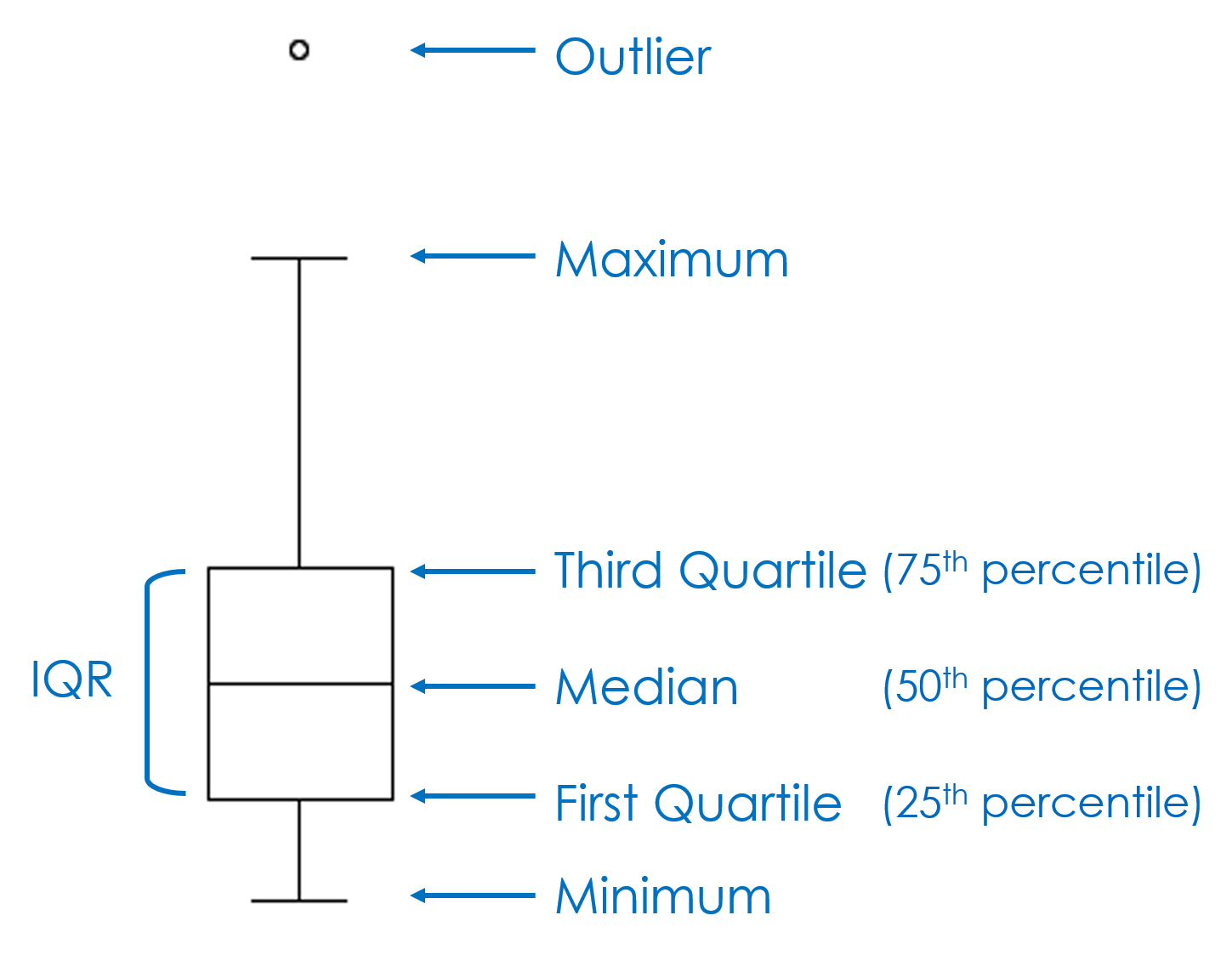
As IQR is calculated using the median, the outlying values don not affect it.  A formula is used to calculate the upper limit and lower limit of this range. Any data point lying outside these limits is an outlier.

**Upper Limit:**Q3 + (1.5 \* IQR)

**Lower Limit:** Q1 – (1.5 \* IQR)



A **box** plot gives good indication of the distribution of data about the median. Boxplots are a standardised way of displaying the distribution of data based on a five-number summary (minimum, first quartile (Q1), median, third quartile (Q3), and maximum).

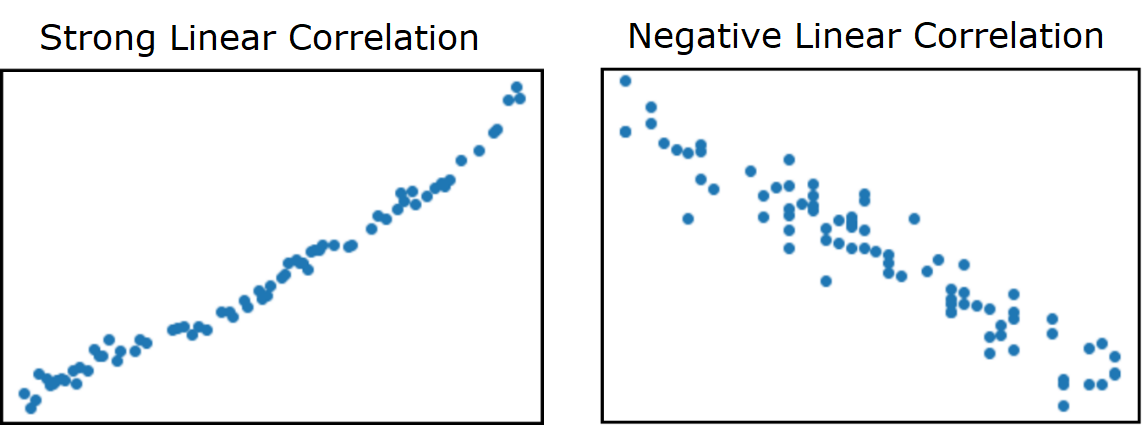


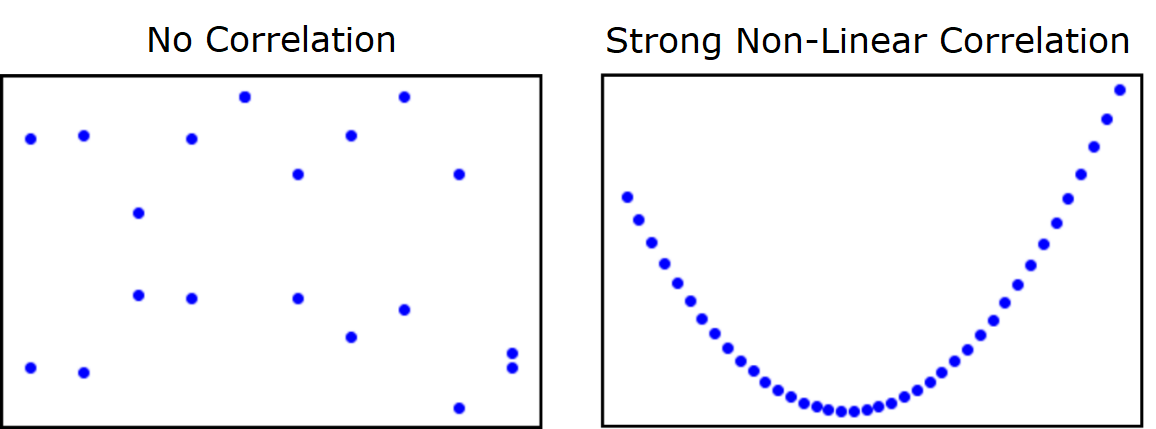
A **scatter** plot uses dots or markers to represent a value in the hyperplane. The position of each point corresponds to the value or properties of a tuple. The scatter plot is one of the simplest plots which can accept both quantitative and qualitative values, with a wide variety of applications in primitive data analysis.

Several meaningful insights can be drawn from a scatter plot, like the ones listed below:

**Finding Correlation Between Variables**

Scatter plots are often used to identify the type of correlation between variables before diving deeper into predictions. The figure below depicts the typical scatter plots indicating the type of correlations.

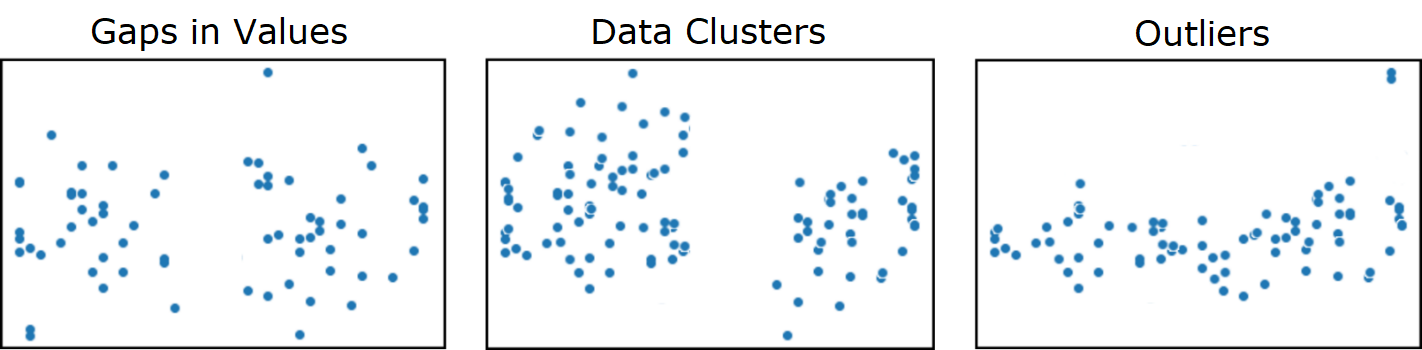




**Identifying Patterns in Data**

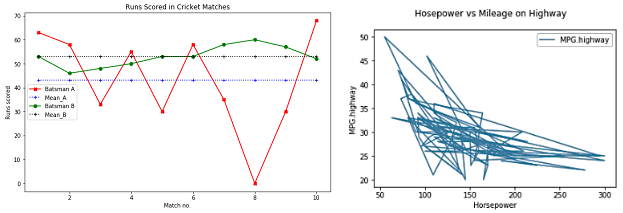
Visualising the tuples as scatter plots can be useful to spot gaps in the values and hence identify the data points crucial to the dataset. It can help draw decisive inferences about the type of predictor or classifier to be used.

The figure below depicts the typical scatter plots to help identify patterns.



A **line chart** is drawn by interconnecting all data points using straight line segments. It is used to analyse historic variations and trends in data. The individual data points are chronologically connected to obtain the patterns and draw meaningful inferences. For example, times series data.

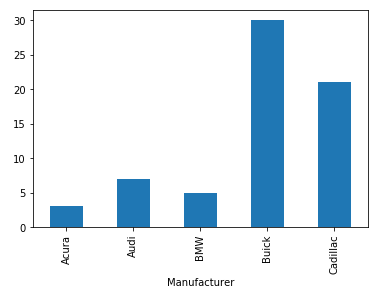
As shown below, the first figure depicts the line chart of an organized collection, while the second figure depicts the line chart of an unorganized collection. Thus, when then data is sorted the line chart is easy to infer. Whereas, when the data is not sorted a messy line chart is generated.



**A bar chart** is a graph with rectangular bars that compares different categories. Each bar represents a particular category and the length of a bar indicates the total number of values or items in that category.

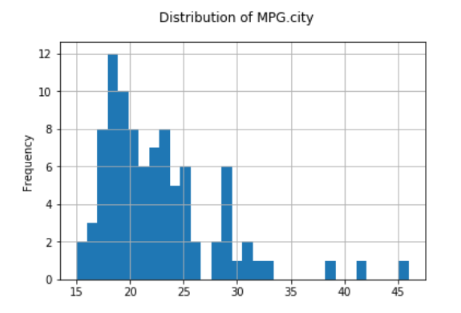
Bar charts can be plotted vertically or horizontally, but vertical bar chart is the most common type. There are multiple variations of bar charts including multiple, stacked, error bars, etc. to cater to various visualisations and presentation needs.

The figure below shows a bar chart depicting the number of cars manufactured by different companies.



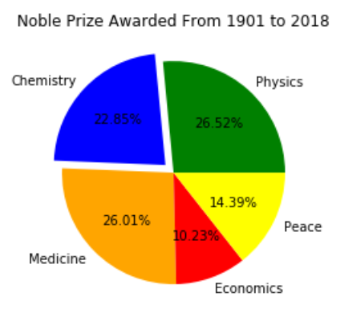
A histogram represents data as rectangular bars. Unlike the bar chart, it is used for continuous data. Each bar groups the numbers into intervals (bins) and the height of the bar is based on the number of values that fall into the corresponding intervals.

A histogram is ideally suited to obtain the frequency distribution of a given data, as depicted in the figure below.



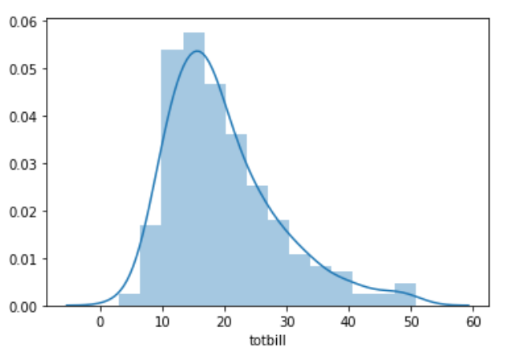
**A pie chart** divides the entire dataset into distinct groups. The chart consists of a circle split into slices and each slice represents a group. The size of the slice is proportional to the number of items in each group compared to the others.

The sum of the slices in a pie chart is always 100%, as depicted in the figure below.



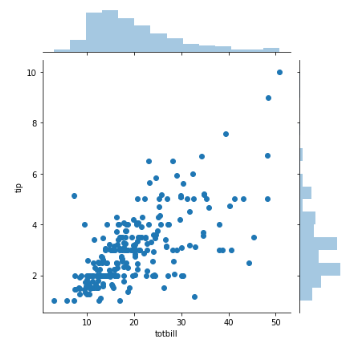
**A dist plot** or distribution plot, depicts the variation in a data distribution. It represents the overall distribution of continuous data variables.

The dist plot depicts the data by a histogram and a line in combination with it, as shown in the figure below.



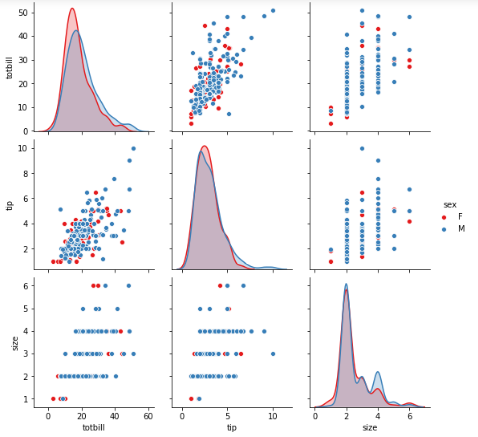
**A joint plot** is a combination of two univariate and one bivariate plots. The bivariate plot (in the center) helps in analysing the relationship between two variables. The univariate plot describes the distribution of data in each variable as a marginal plot.

A joint plot is used to quickly visualise and analyse the relationship between two variables and examine their distributions on the same plot, as shown in the figure below.



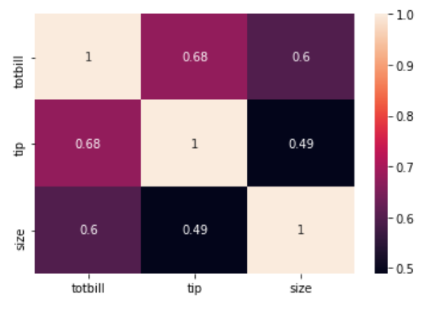
**A pair plot** depicts pairwise relationships between all the variables in a dataset in a matrix format. Each row and column in the matrix represents a variable in the dataset.

The plots present in the diagonal are univariate plots as the variables are compared with themselves and the others are bivariate scatter plots, as shown in the figure below.



**A heat map** is a graphical representation of data where similar values are depicted by the same colours. The colours vary based on the intensity of the results.

One example for heat map is to find the correlation between the variables in a dataset, as depicted in the figure below.

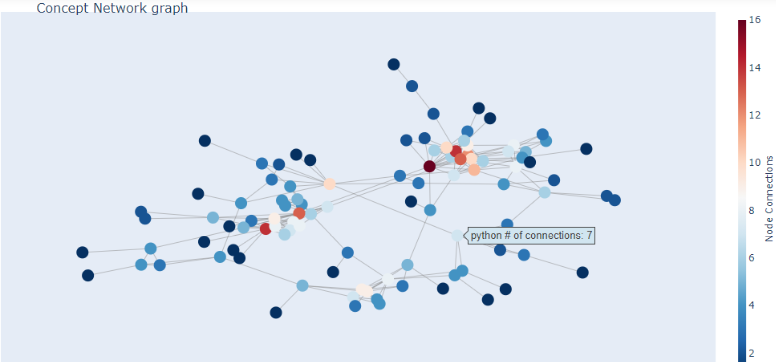


**A network** is a set of objects (called nodes or vertices) that are connected to each other. The connections between the nodes are called edges or links.

If the edges in a network are directed, i.e., pointing in only one direction, the network is called a directed network. When drawing a directed network, the edges are typically drawn as arrows indicating the direction.

If all edges are bidirectional, or undirectional, the network is an undirected network.

The figure below depicts the concept of a network graph.



**A word cloud** is a visual representation of free form text, which is like a collage. It is typically used to depict keyword metadata of websites, articles, reviews, feedbacks etc. The frequency and significance of the words are depicted by the font, font size and colour of the text in the cluster.

Words with greater significance and occurrence are depicted in a bigger and bolder font towards the central location of the cluster and other latent words occupy peripheral places with smaller fonts and faded colors. Most insignificant words, stop words, irrelevant information is eliminated from the cluster while plotting it.

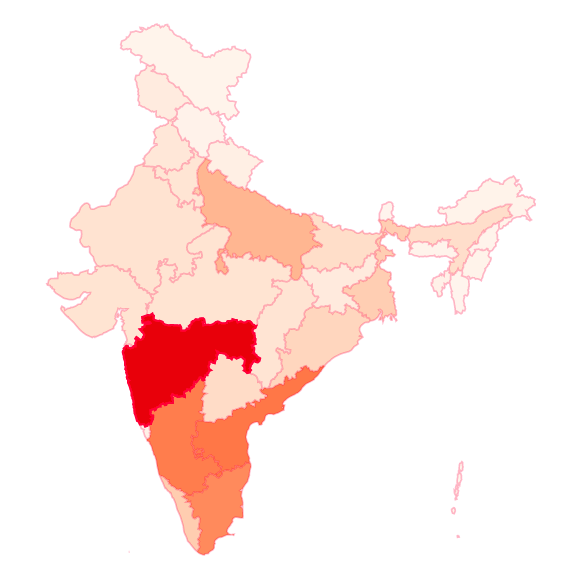
A word cloud finds its usage more in Natural Language Processing.

The picture below depicts the concept of a word cloud.



**A choropleth** map is a pictorial representation of data on a geographical map. The intensity of color in a region on the map corresponds to the respective values.

The figure below depicts the choropleth map of Covid-19 distribution in India as of 14th October, 2020. It represents the count of the spread on the given date. A deeper shade corresponds to a higher value while a lighter shade marks the safe regions.



# Data Visualization Libraries in Python

Few of the popular Python libraries used for data visualisation are listed as follows:

* Matplotlib
* Seaborn
* Plotly

Let us briefly discuss each of them.

Matplotlib is one of the most basic and popular Python libraries used for data visualisation. It is developed for imitating the plotting capabilities of MATLAB, another programming environment.

'matplotlib.pyplot' is used for two dimensional graphics in Python programming. It can be used in Python shell, scripts, web application servers, and other graphical user interface toolkits. It is heavily dependent on other Python libraries such as numpy, which is considered as a major drawback.

**Installing Matplotlib**

If you have Anaconda Navigator, the matplotlib package can be installed using the following command:

conda install matplotlib

Otherwise, navigate to the command prompt and provide the following command:

pip install matplotlib

Seaborn is a statistical data visualisation library in Python. It is integrated to work with Pandas dataframes with a more straight forward approach.

Seaborn extends the plotting capabilities of matplotlib and provides a high-level interface to generate attractive plots that are visually appealing.

**Installing Seaborn**

If you have Anaconda Navigator, the seaborn package can be installed using the following command:

conda install -c anaconda seaborn

Otherwise, navigate to the command prompt and provide the following command:

pip install seaborn

Plotly is another data visualisation library that is used to generate highly interactive plots.

**Installing Plotly**

If you have Anaconda Navigator, the plotly package can be installed using the following command:

conda install -c plotly plotly

Otherwise, navigate to the command prompt and provide the following command:

pip install plotly

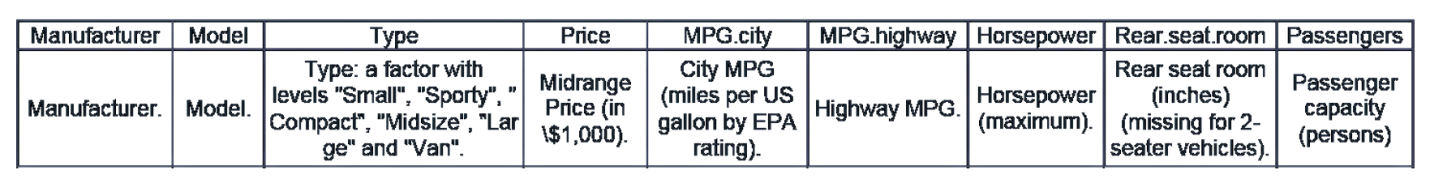
**Data visualization with Python  
Use Case:**

Let us consider a use case, where a cutomer wishes to buy a car. Following are some of the questions that the customer might have before the purchase.

* What is the price range of different cars available in the market?
* What is the range of horsepower and mileage of various cars?
* Does a car with higher horsepower give lower mileage?
* How much leg space does the car have?
* How many passengers can the car carry based on its type?

Let us use 'Cars93' dataset to answer the above questions. Click [here](https://infyspringboard.onwingspan.com/common-content-store/Shared/Shared/Public/lex_auth_013197055873867776529_shared/web-hosted/assets/Cars931611042729436.zip) to downloaded the dataset.

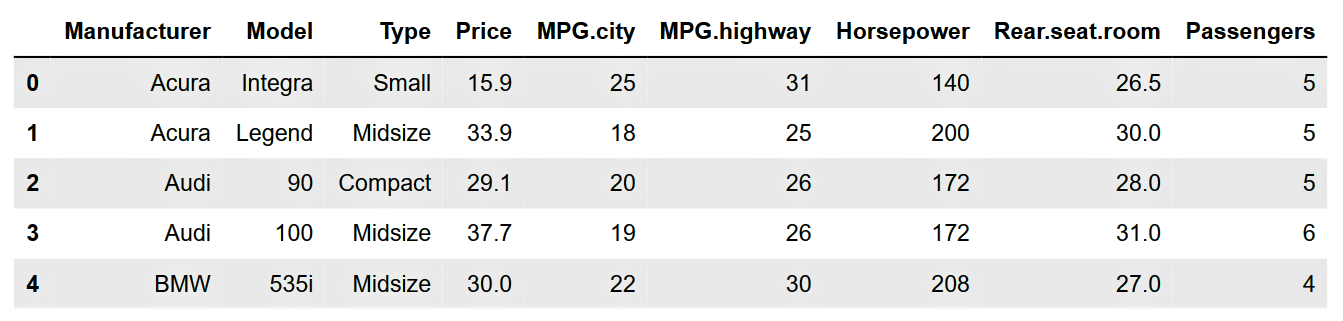
The information that the columns of this dataset contain is given below:



Now, let us import the dataset using the code given below:

1. *#Importing the necessary Libraries*
2. import numpy as np
3. import pandas as pd
4. import matplotlib.pyplot as plt
5. import matplotlib.cm as cm
6. *#Importing the required dataset*
7. cars\_df = pd.read\_csv("Data/Cars93.csv")
8. columns = ["Manufacturer","Model","Type","Price",
9. "MPG.city","MPG.highway","Horsepower","Rear.seat.room","Passengers"]
10. cars\_df[columns].head()

A sample of the dataset for the columns selected in the code above is shown below:



Let us first identify the price range of the cars in the dataset.

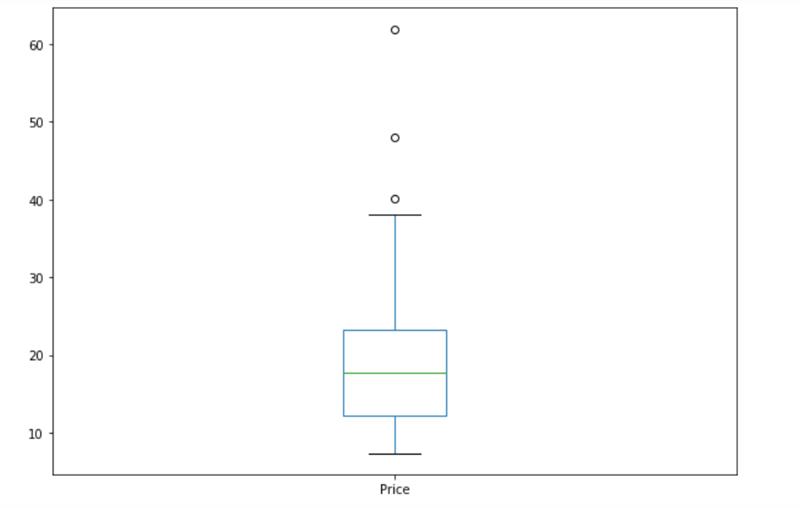
# Box Plot

The price range of the cars can be represented using a box plot. It is a graphical way of depicting the five-number summary as discussed in the Introduction to Probability and Statistics course.

Matplotlib works efficiently with dataframes and arrays. So, Pandas DataFrame can be used as they have some functionality of matplotlib built-in to create visualisations. You will notice that the same method is followed thoughtout this module.

The Python code to create a box plot for the price range of cars using cars\_df["Price"] column, is given below:

1. *#creating a box plot for the variable 'Price'*
2. cars\_df["Price"].plot(kind="box",figsize = (10,7))



plot( ) is a method of the DataFrame that takes a parameter 'kind' for displaying the appropriate visualisation. For example, the word 'box' is used to generate a box plot. Similarly, other visualisations are generated if values like bar, line, scatter, etc. are used as the 'kind' parameter.

For more details on plot features, refer to 'pandas.DataFrame.plot' section of the Pandas documentation.

The box plot is also called a whiskers plot. In case the data contains outliers, then the extreme lines (called whiskers) represent 1.5 times the IQR value from Q1 and Q3 respectively. Outliers here are the values that are outside the range described by 1.5 times IQR value from Q1 and Q3.

The skewness of data can also be identified from the visualisation of a box plot. Since the box in the above plot is towards the minimum, this data is right-skewed. Similarly, if the data is left-skewed, then the box would be towards the maximum.

Mathematically:

* Outliers < Q1 - 1.5 (IQR)
* Outliers > Q3 + 1.5 (IQR)

It can be observed that, if the outliers are excluded, then:

* minimum price of a car comes close to 10 thousand dollars and maximum price comes close to 40 thousand dollars
* most cars are priced approximately between 11 thousand dollars to 22 thousand dollars

But is car price the only parameter that needs to be considered while purchasing a car? The answer is NO.

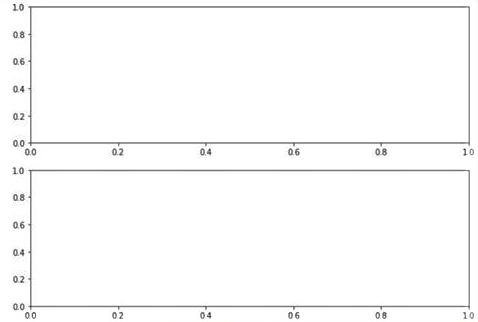
Let us tap into the other features of the cars to find the best-suited car for the customer.

 Often, there is a need to display the distribution of multiple features together for better understanding.

Let us learn how to plot the graphs alongside each other.

Let us use the subplots functionality of 'matplotlib.pyplot' to create a grid-like system based on the number of rows and columns provided, as shown in the code below:

1. *#The following lines enable us to use subplot functionality*
2. fig, (ax1, ax2) = plt.subplots(2, 1)
3. fig.set\_figwidth(10) *#setting the width for the plot*
4. fig.set\_figheight(7) *#setting the height for the plot*



The subplots method returns a figure object and axes objects and their functionality is as follows:

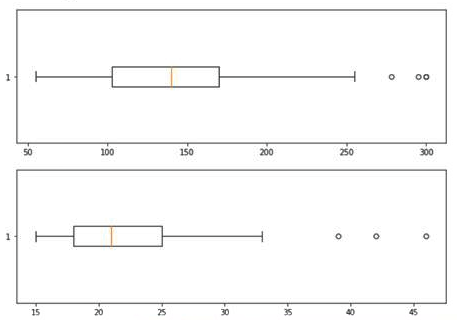
**Figure object:** It controls the structure of the plot as shown above, where the height and width of the plot needs to be set.

**Axes object:** It controls what exists in the plot like labels, data, text etc.

The number of axes objects depend on the number of elements in that grid. For example, if there are two elements, there will be two axes objects.

Let us use the axes objects to create boxplots alongside each other for the range of horsepower and mileage of cars, as shown in the code below:

1. *#The following lines enable us to use subplot functionality*
2. fig, (ax1, ax2) = plt.subplots(2, 1)
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. *#The following lines of code change the alignment from vertical to horizontal*
6. ax1.boxplot(cars\_df["Horsepower"],vert=False)
7. ax2.boxplot(cars\_df["MPG.city"],vert=False)

The output of the above code is given below:

Both horsepower and mileage are plotted together. However, it is difficult to recognise the plots that represent the respective variables.

How can the plot be improved?

Will the use of labels and titles enhance the readability of this plot?

Let us add labels to the plots to make it more concise. To do so, the axes objects can be used to set the title for each plot, as follows:

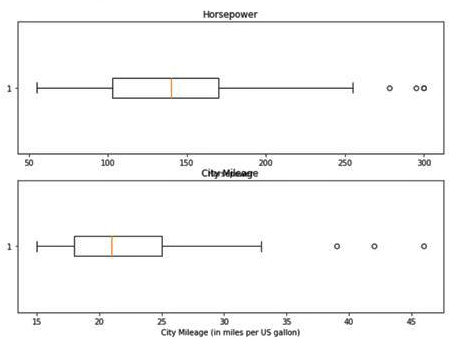
* set\_title method sets the title for each plot with default position as top of the plot
* set\_xlabel sets the text for x-axis which usually describes the nature of the axis

Similarly, there is y\_label for y-axis which will be discussed later in this course.

The code to enhance the plot is given below:

1. *#The following lines enable us to use subplot funtionality*
2. fig, (ax1, ax2) = plt.subplots(2, 1)
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. *#The following lines of code change the alignment from vertical to horizontal*
6. ax1.boxplot(cars\_df["Horsepower"],vert=False)
7. ax2.boxplot(cars\_df["MPG.city"],vert=False)
8. *#The following lines of code are used to add labels to axes and title to the graph*
9. ax1.set\_title('Horsepower')
10. ax1.set\_xlabel('Horsepower')
11. ax2.set\_title('City Mileage')
12. ax2.set\_xlabel("City Mileage (in miles per US gallon)")

The output of the above code is given below:

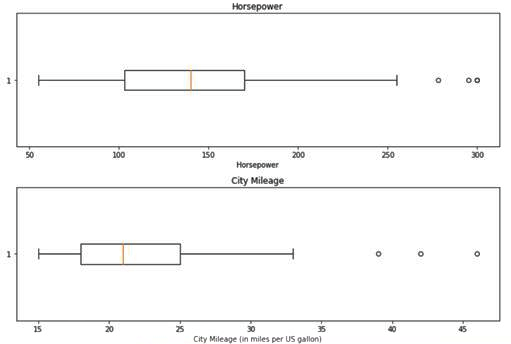


It can be seen from the above graph that adding labels to the plots made them more readable and intelligible. But there is an issue with the 'City Mileage' title as it is overlapping with the x-axis label of the first plot.

To avoid such issues, the tight\_layout method of pyplot in matplotlib is used, as shown below:

1. *#The following lines enable us to use subplot functionality*
2. fig, (ax1, ax2) = plt.subplots(2, 1)
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. *#The following lines of code change the alignment from vertical to horizontal*
6. ax1.boxplot(cars\_df["Horsepower"],vert=False)
7. ax2.boxplot(cars\_df["MPG.city"],vert=False)
8. *#The following lines of code are used to add axis labels and titles to the graph*
9. ax1.set\_title('Horsepower')
10. ax1.set\_xlabel('Horsepower')
11. ax2.set\_title('City Mileage')
12. ax2.set\_xlabel("City Mileage (in miles per US gallon)")
13. *#In case of any superimposition of the subplots, the following functions caters the aesthetics*
14. fig.tight\_layout()

This will arrange all the elements of the plot without any overlapping. The output after applying the tight layout is shown below:



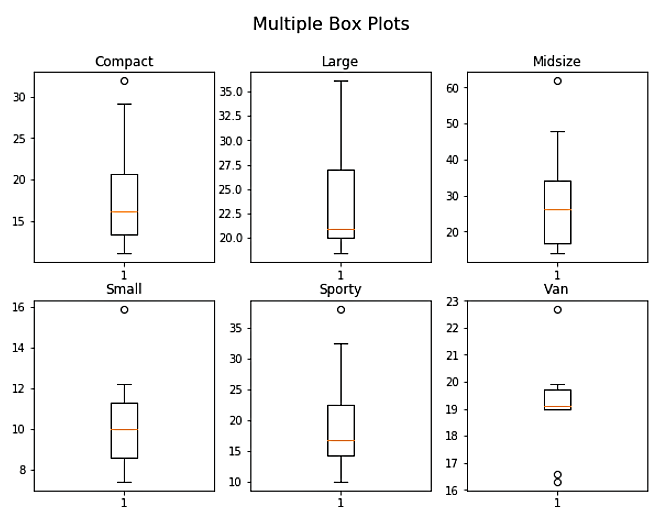
From the above graph, it can be understood that the plot on the top is for 'Horsepower' and in the bottom is for 'City Mileage'.

It can also be observed that the range of horsepower without outliers is in order of 60 – 260 while that of mileage is 15 – 34 MPG.

Let us find the price range of cars for each car type by plotting different box plots, as shown below:

1. *# Setting up the partitions, length and width of the figure*
2. fig, ax = plt.subplots(2, 3)
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. *#title*
6. fig.suptitle("Multiple Box Plots", fontsize=16)
7. *#Accessing each partition[m][n] and providing the plot and its title*
8. ax[0][0].boxplot(cars\_df["Price"][cars\_df["Type"]=="Compact"])
9. ax[0][0].set\_title('Compact')
10. ax[0][1].boxplot(cars\_df["Price"][cars\_df["Type"]=="Large"])
11. ax[0][1].set\_title('Large')
12. ax[0][2].boxplot(cars\_df["Price"][cars\_df["Type"]=="Midsize"])
13. ax[0][2].set\_title('Midsize')
14. ax[1][0].boxplot(cars\_df["Price"][cars\_df["Type"]=="Small"])
15. ax[1][0].set\_title('Small')
16. ax[1][1].boxplot(cars\_df["Price"][cars\_df["Type"]=="Sporty"])
17. ax[1][1].set\_title('Sporty')
18. ax[1][2].boxplot(cars\_df["Price"][cars\_df["Type"]=="Van"])
19. ax[1][2].set\_title('Van')

The output of the above code is given below:

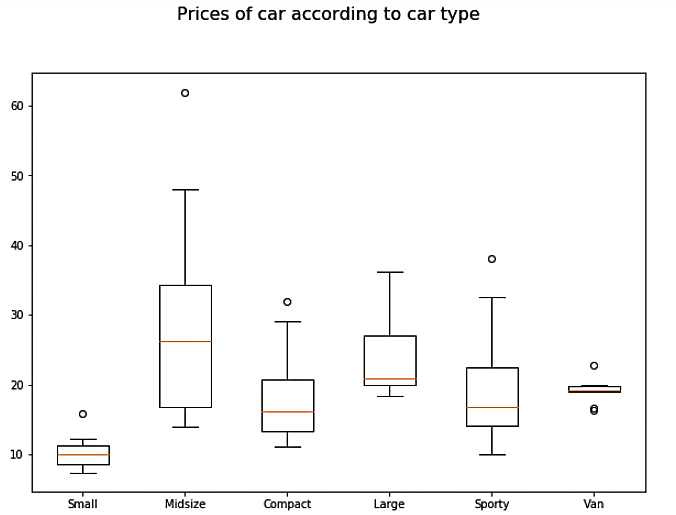


'plt.subplots' method creates a grid-like system of dimensions 2x3 and returns the figure and axis objects using which, individual subplots can be plotted. As it follows the zero-index system, the two-dimensional matrix style can be used for accessing each subplot. Here, 'set\_figwidth' and 'set\_figheight' methods of the figure object achieve the desired size of the figure and improve readability.

Until now all plots in the subplot were plotted individually, due to which they could not be compared. Therefore, let us try to plot box plots of different types of cars in one plot so that all these plots are plotted in the same canvas and are comparable.

1. *#Finding the list of unique values of 'car type'*
2. car\_type\_list = cars\_df["Type"].unique()
3. *#setting the width and height of the plot*
4. fig, ax = plt.subplots()
5. fig.set\_figwidth(10)
6. fig.set\_figheight(7)
7. *#creating a box plot for every unique car type*
8. ax.boxplot([cars\_df["Price"][cars\_df["Type"]==k] for k in car\_type\_list])
9. *#To set the position for each plots in the iteration*
10. plt.xticks([i for i in range(1,len(car\_type\_list)+1)],[k for k in car\_type\_list])
11. *#super-title*
12. fig.suptitle("Prices of car according to car type", fontsize=16, y = 1)

The output of the above code is shown below:



Now that all the boxplots are plotted successfully under the same axes, let us break down the code to see how each line helped in achieving this.

**Code Breakup**

1. First, the unique values of the Type feature need to be identified.

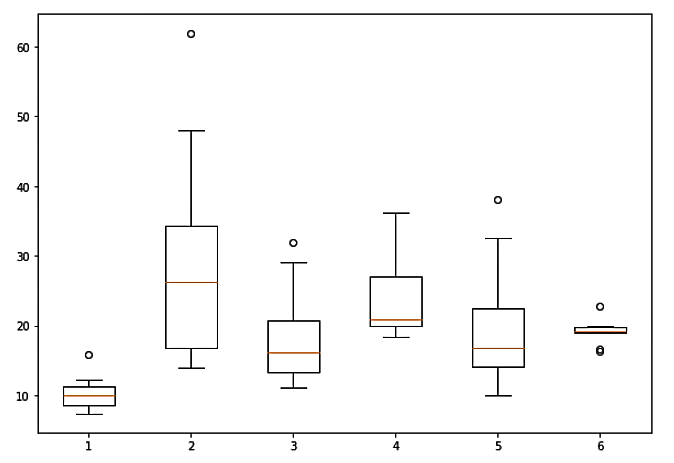
1. car\_type\_list = cars\_df["Type"].unique()
2. car\_type\_list

**Output: array(['Small', 'Midsize', 'Compact', 'Large', 'Sporty', 'Van'], dtype=object)**

2. Next, the list comprehension can be used to create a list of values which contain the data required for plotting each box plot. Each item in the list is the data associated with each type of car in the car\_type\_list.

1. car\_type\_list = cars\_df["Type"].unique()
2. fig, ax = plt.subplots()
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. ax.boxplot([cars\_df["Price"][cars\_df["Type"]==k] for k in car\_type\_list])

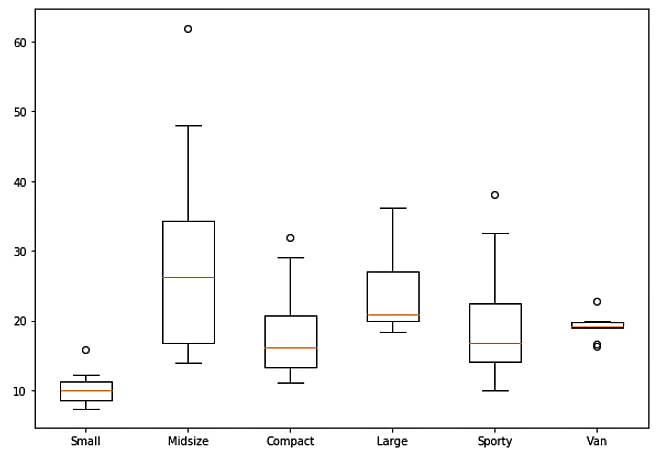
The output of the code above is shown below:



3. Finally, xticks can be assigned to the plot to determine which boxplot belongs to which type of car.

1. car\_type\_list = cars\_df["Type"].unique()
2. fig, ax = plt.subplots()
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. ax.boxplot([cars\_df["Price"][cars\_df["Type"]==k] for k in car\_type\_list])
6. plt.xticks([i for i in range(1,len(car\_type\_list)+1)],[k for k in car\_type\_list])

The output of the code above is shown below:

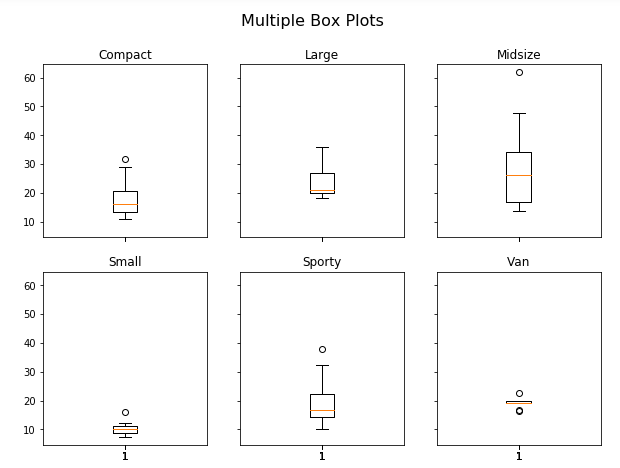


You can notice that the dimensions in the subplots method are not mentioned. In such cases, the axis object takes the default value (1, 1).

Another way to create a box plot is by using the 'grid' method. Let us modify the code in the previous page by using the 'sharey' and 'sharex' properties of the subplots, as shown below:

1. *#setting the plot, width and height*
2. fig, ax = plt.subplots(2, 3, sharey=True, sharex=True)
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. *#super-title*
6. fig.suptitle("Multiple Box Plots", fontsize=16)
7. *#accessing and creating the respective sub-plots*
8. ax[0][0].boxplot(cars\_df["Price"][cars\_df["Type"]=="Compact"])
9. ax[0][0].set\_title('Compact')
10. ax[0][1].boxplot(cars\_df["Price"][cars\_df["Type"]=="Large"])
11. ax[0][1].set\_title('Large')
12. ax[0][2].boxplot(cars\_df["Price"][cars\_df["Type"]=="Midsize"])
13. ax[0][2].set\_title('Midsize')
14. ax[1][0].boxplot(cars\_df["Price"][cars\_df["Type"]=="Small"])
15. ax[1][0].set\_title('Small')
16. ax[1][1].boxplot(cars\_df["Price"][cars\_df["Type"]=="Sporty"])
17. ax[1][1].set\_title('Sporty')
18. ax[1][2].boxplot(cars\_df["Price"][cars\_df["Type"]=="Van"])
19. ax[1][2].set\_title('Van')

The output given below has been generated by using 'sharey = True' and 'sharex = True'.



The different types of cars along with their prices can now be easily compared. Many other things can also be infered from the above visualisation. For example,

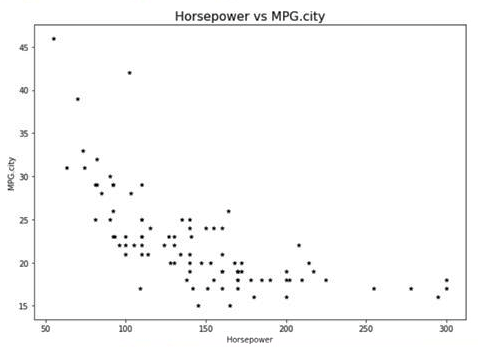
* vans have the smallest price range, whereas midsize cars have the largest price range
* small cars have the lowest price, whereas large cars have the highest price

# Scatter Plot

Now that you know the individual ranges of both horsepower and mileage, let us find if there is any relationship between them. Plotting these values against each other might reveal the presence of a relationship among them.

1. *#To Plot the data as a scatter plot*
2. ax = cars\_df.plot(["Horsepower"],["MPG.city"],kind="scatter", color = "black",marker = "\*",figsize=(10,7))
3. *#To add labels and title to the output*
4. ax.set\_xlabel("Horsepower") *#sets label for x-axis*
5. ax.set\_ylabel("MPG.city") *#sets label for y-axis*
6. ax.set\_title("Horsepower vs MPG.city",fontsize=16) *#sets title for the graph*

 The output of the above code is given below:



Since the scatter plot is a 2D plot, the x and y are passed in the same order as shown in the above code. The parameters 'color' and 'marker' can be used to adjust the colour and shape of the points shown in the plot. For example, black asterisks are used for displaying each data point.

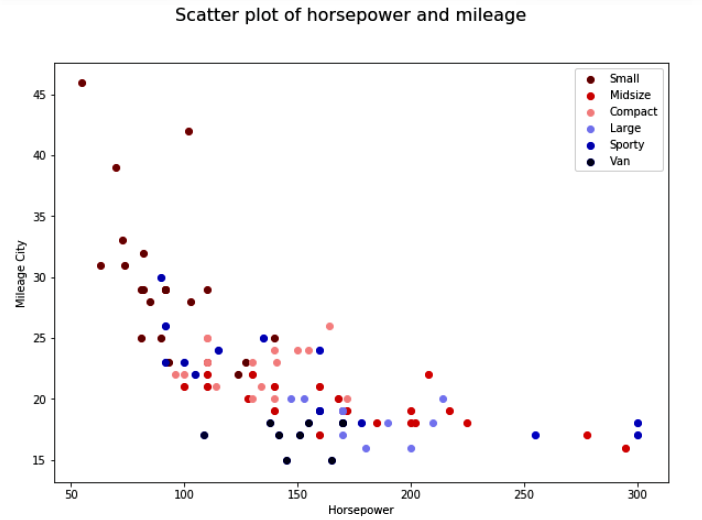
Here, it can be observed that as horsepower increases the mileage is likely to decrease.

Now let us understand the relationship between horsepower and mileage based on car type.

By now you know the unique values of the car type in the previous plots. The same list is used to create a scatter plot between horsepower and mileage.

1. fig = plt.figure()
2. fig.set\_figwidth(10)
3. fig.set\_figheight(7)
4. colors = cm.seismic\_r(np.linspace(0, 1, len(car\_type\_list))) *# We extract the colours using the 'seismic\_r' method. Here, 'r' indicates the reverse.*
5. for car\_type,c in zip(car\_type\_list,colors): *# for every car type in the car\_type\_list we plot all the points in the scatter plot*
6. x = cars\_df[cars\_df["Type"] == car\_type]["Horsepower"]
7. y = cars\_df[cars\_df["Type"] == car\_type]["MPG.city"]
8. plt.scatter(x,y,color = c,label=car\_type)
9. plt.suptitle("Scatter plot of horsepower and mileage",fontsize=16)
10. plt.xlabel("Horsepower")
11. plt.ylabel("Mileage City")
12. plt.legend()

The output of the above code is given below:



From the scatter plot it can be understood that higher the horsepower, lower the milage of a car in a city. It can also be deduced that vans give the least mileage in a city whereas small cars are the best fit though they have the least horsepower.

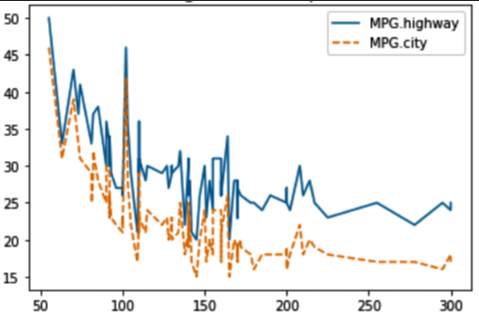
# Line Chart

A line chart is a type of chart that displays information as a series of data points connected by straight line segments.

Sometimes multiple graphs can depict the same information. Here, a connected scatter plot is created where the data is sorted based on the horsepower, to get meaningful insights from the line chart.

1. *#First sort the data to get a proper line chart*
2. cars\_df=cars\_df.sort\_values(by="Horsepower")
3. *#The following lines of code create a blank canvas to plot on*
4. fig, ax = plt.subplots()
5. fig.set\_figwidth(10)
6. fig.set\_figheight(7)
7. *#Data is fed and plotted using the following lines*
8. cars\_df.plot(ax = ax, x = "Horsepower", y = "MPG.highway", kind = "line", )
9. cars\_df.plot(ax = ax, x = "Horsepower",y= "MPG.city", kind = "line", linestyle='--')

The output of the above code is a line chart or a line graph and is shown below:



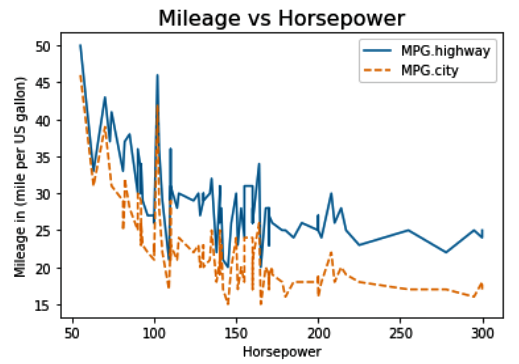
You can notice that two line charts with the same axes are plotted, unlike the box plot where subplots created multiple boxes. To do this, the axis object, returned from the subplots( ) method is passed to both the line charts using the 'ax' parameter. Note that the number of rows and columns have not been passed to the subplots method.

The plot method takes care of the colours to make each line look distinct. Also, 'linestyle' parameter can be passed to display the lines in different styles.

It can be observed that as horsepower increases the mileage is likely to decrease. It can also be deduced that there is a peak in mileage at around 100 horsepower, which was not so clear in the scatter plot created previously.

Let us enhance the readability of this chart further by replacing the labels, as shown in the code below:

1. *#The following lines of code create a blank canvas to plot on*
2. fig, ax = plt.subplots()
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. *#Data is fed and plotted using the following lines*
6. cars93\_ordered.plot(ax = ax, x = "Horsepower", y = "MPG.highway", kind = "line", )
7. cars93\_ordered.plot(ax = ax, x = "Horsepower",y= "MPG.city", kind = "line", linestyle='--')
8. *#The following part of code adds labels and titles to make the graph readable*
9. ax.set\_ylabel("Mileage in (mile per US gallon)")
10. ax.set\_title("Mileage vs Horsepower",fontsize=16)



It can be observed that the graph is not ambiguous anymore and depicts all information as intended.

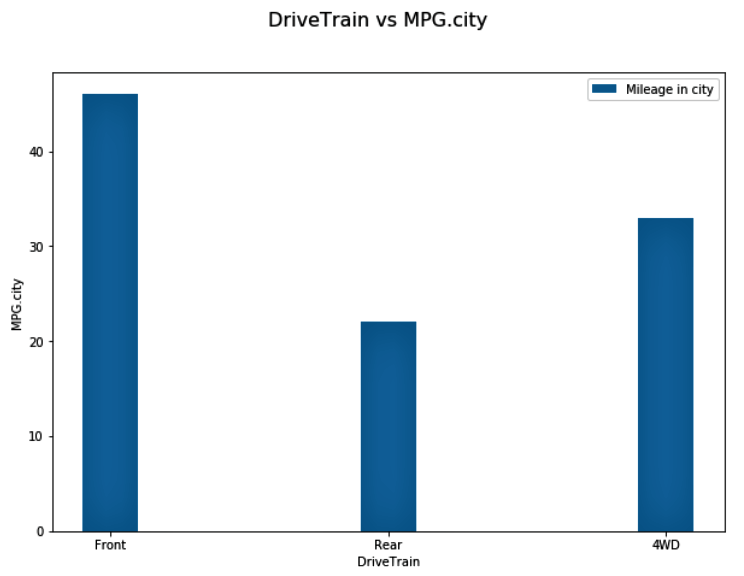
**bar chart**

Box plot, scatter plot, or line chart fail when there is a need to plot both categorical and numerical data, as they can only take numerical data. Then, how to find relationships between such types of data? Which graph can be used?

The answer to the questions above is **a bar chart** or a bar graph. A bar chart takes in two features, 'x' and 'y' as inputs. 'x' is the categorical data  plotted against 'y' which is the numerical data. The following code shows a simple bar chart between 'DriveTrain' and 'MPG.city' along with the plot aesthetics.

1. *#width,height*
2. fig = plt.figure()
3. fig.set\_figwidth(10)
4. fig.set\_figheight(7)
5. *#code to create bar chart*
6. plt.bar(cars\_df["DriveTrain"], cars\_df["MPG.city"],width=0.2,label="Mileage in city")
7. *#title and label*
8. plt.suptitle("DriveTrain vs MPG.city",fontsize=16)
9. plt.xlabel("DriveTrain")
10. plt.ylabel("MPG.city")
11. *#legend*
12. plt.legend()

The output of the above code is shown below:

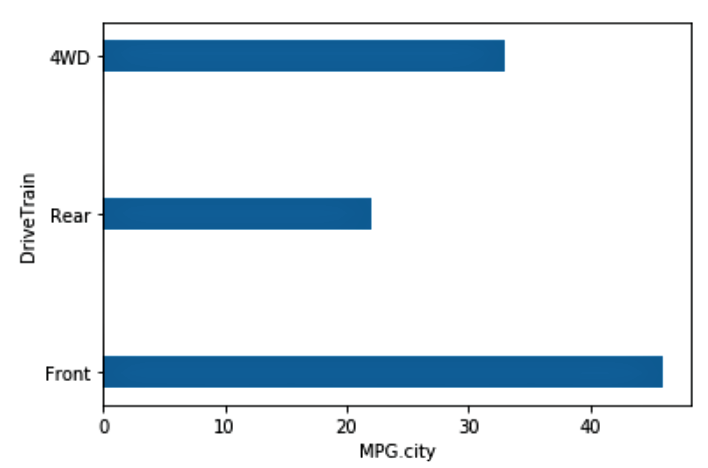


A bar chart can also be plotted horizontally using the barh() method from 'matplotlib'.

Notice that the first argument is still categorical data. Also, observe that the axes labels have been changed accordingly.

1. *#horizontal bar graph*
2. plt.barh(cars\_df["DriveTrain"], cars\_df["MPG.city"],height=0.2,label="Mileage in city")
3. plt.xlabel("MPG.city")
4. plt.ylabel("DriveTrain")

The output of the above code is shown below:



Consider the type of data with multiple categories. Assue, there is a need to plot a graph using different colours representing each category.

Let us consider 'Passengers' and 'Type' from the 'Cars93' dataset.

1. *#Use the following code snippet to filter the unique values of no. of passengers a car can carry*
2. cars\_df["Passengers"].unique()

**Output:** array([4, 5, 6, 7, 8, 2], dtype=int64

1. *#Use the following code snippet to filter the unique values of Types of car.*
2. cars\_df["Type"].unique()

**Output:** array(['Small', 'Sporty', 'Compact', 'Midsize', 'Van', 'Large'], dtype=object)

Let us find the number of each type of car for each passenger capacity. The data can be prepared for plotting by using the code below:

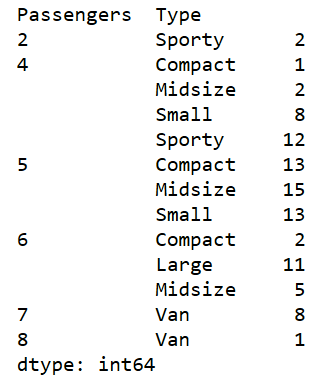
1. *#Use the following code snippet to filter the data and obtain the target columns into a separate dataframe.*
2. grouped\_data = cars\_df[["Passengers","Type"]].groupby(by = ["Passengers","Type"]).size().unstack().reset\_index()

Let us analyse the above code to understand how each of the three methods modifies the data.

1. **size()** after groupby() returns the frequency for each 'Type'.

1. grouped\_data = cars\_df[["Passengers","Type"]].groupby(by= ["Passengers","Type"]).size()

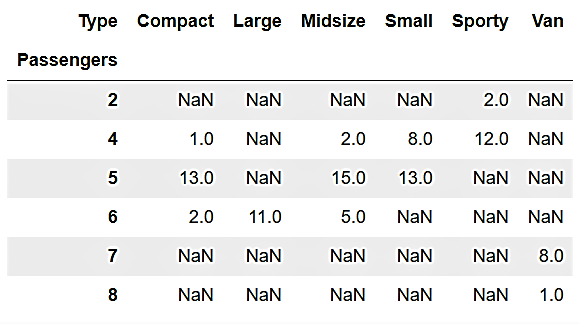
**Output:**



2. **unstack()** uses 'Passengers' as pivot and converts the above data to a dataframe as shown below:

1. grouped\_data = cars\_df[["Passengers","Type"]].groupby(by= ["Passengers","Type"]).size().unstack()

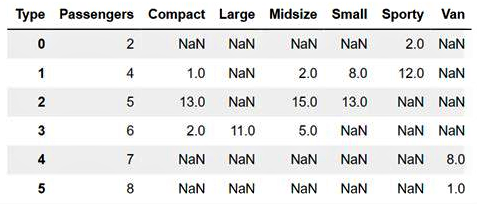
**Output:**



3. **reset\_index()** as the name suggests, resets the index of the above dataframe as shown below:

1. grouped\_data = cars\_df[["Passengers","Type"]].groupby(by= ["Passengers","Type"]).size().unstack().reset\_index()

**Output:**

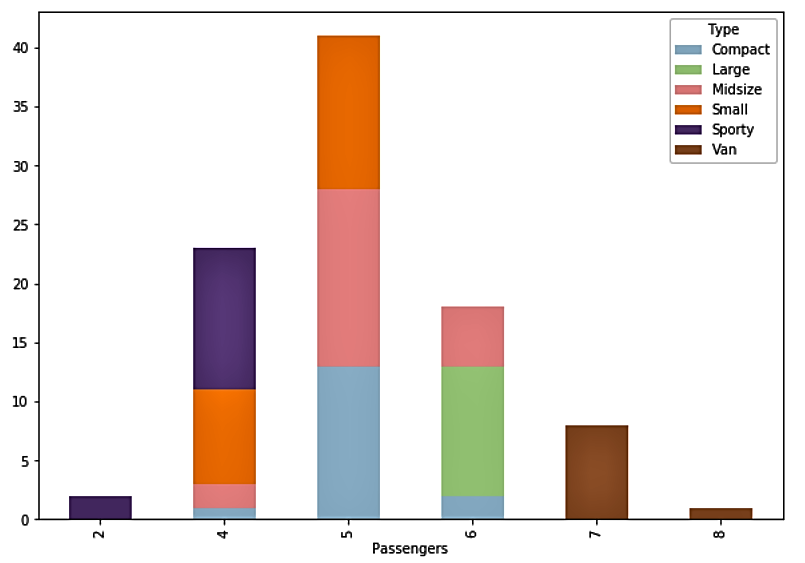


The NaN values seen in the output above will be ignored.

Let us proceed further and plot the data for each type of car using bar() as shown in the code below:

1. *#Stacked Bar Graph can be plotted using the grouped data, as follows:*
2. grouped\_data.plot(x="Passengers",kind="bar",stacked=True,colormap=cm.Paired,figsize=(10,7))

The output of the above code is shown below:



Matplotlib has built-in colormaps. Here, 'Paired' is used. Refer to 'matplotlib.cm' documentation for more colormap options.

**Note:**The three methods in the above code, size(), unstack() and reset\_index() have been utilized for demonstration purposes and are required only on a case to case basis.

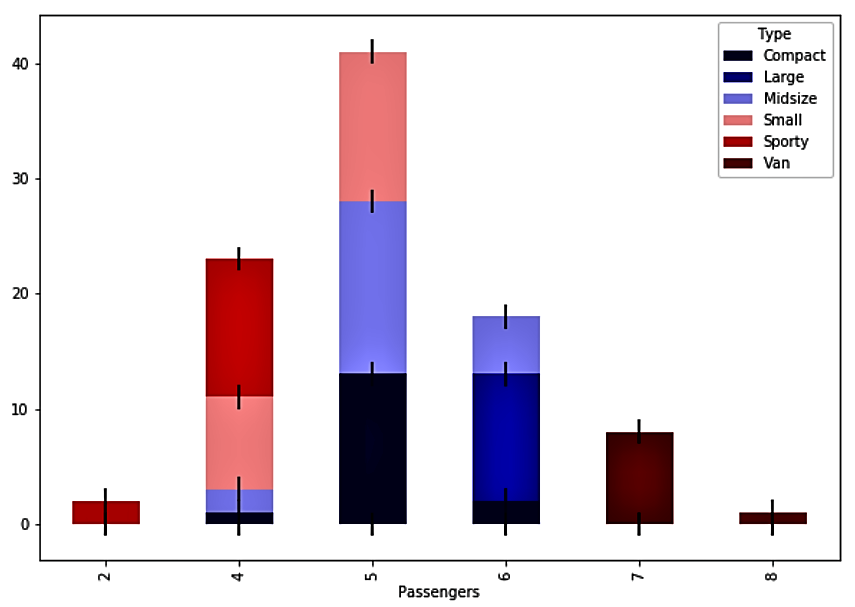
Error bars on a cartesian graph are a graphical enhancement to visualise the variability of plotted data. They are used on graphs to indicate the uncertainty in a reported measurement. A general idea of how precise a measurement is, can be obtained by using error bar charts.

Let us use the same features and data from the previous page.

Error bars can be added to the plot using 'yerr' or 'xerr' feature of bar() and barh() methods. They are used to plot the standard deviation, maximum-minimum or confidence intervals in a dataset. In the code given below, the standard deviation is plotted. Observe that a different colormap called 'seismic' has been used.

1. *#Error bars can be added to the stacked bar graph with the 'yerr' argument as follows*
2. grouped\_data.plot(x="Passengers",kind="bar",stacked=True,colormap=cm.seismic,figsize=(10,7),yerr=np.std(cars\_df["Passengers"]))

The output of the above code is the graph shown below. The black lines parallel to one of the axes are the error bars. This graph is interpreted by the length of the error bar. Longer the bar, more the deviation and shorter line indicates less deviation from the data.



**Note:** Error bars are not unique to the bar plot. They can also be used with other plots like scatter plot or line chart. You are encouraged to explore more about error bars for other plots.

Let us discuss the grouped bar chart where the bars are stacked beside each other to show the difference between multiple features used on 'x' to the values of 'y'.

Assume, there is a need to see how each type of 'DriveTrain' performs based on 'MPG.city', 'MPG.highway' and 'RPM'.

First, let us see the unique values in 'DriveTrain'.

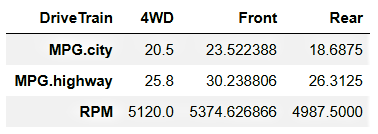
1. cars\_df["DriveTrain"].unique()

**Output:** array(['Front', '4WD', 'Rear'], dtype=object)

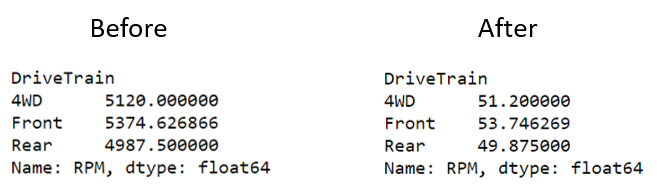
Let us group the features based on 'DriveTrain' and get their sum.

1. grouped\_cars = cars\_df[["MPG.city","MPG.highway","RPM","DriveTrain"]].groupby(by="DriveTrain").mean().T
2. grouped\_cars

**Output:**



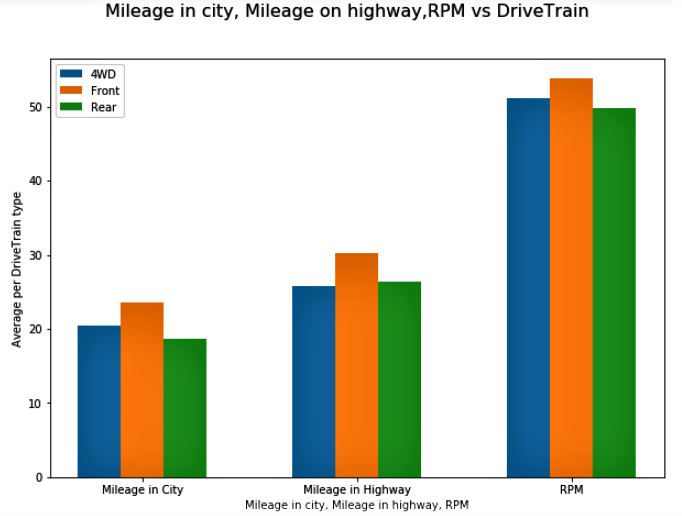
Before plotting the above features, to have an even chart, RPM needs to be scaled as it has high values. For this purpose, RPM is divided by 100.



Now, let us plot the bar chart using the code given below:

1. fig = plt.figure()
2. fig.set\_figwidth(10)
3. fig.set\_figheight(7)
4. grouped\_cars.loc['RPM'] /= 100
5. width=0.2 *# We assign the value of the width of the bar and on the number of groups.*
6. ind=list(range(len(cars\_df['DriveTrain'].unique())))
7. plt.bar([i for i in ind], height=grouped\_cars["4WD"], label="4WD", width=width)
8. plt.bar([i+width for i in ind], height=grouped\_cars["Front"], width=width, bottom=0, label="Front")
9. plt.bar([i+width\*2 for i in ind], height=grouped\_cars["Rear"], label="Rear", width=width, bottom=0)
10. plt.suptitle("Mileage in city, Mileage on highway,RPM vs DriveTrain", fontsize=16)
11. plt.xlabel("Mileage in city, Mileage in highway, RPM")
12. plt.ylabel("Average per DriveTrain type")
13. plt.xticks([i+width for i in ind],["Mileage in City","Mileage in Highway","RPM"])
14. plt.legend()

The output of the above code is shown below:



Let us analyse the code to understand the bar chart shown above, as follows:

1. The bar chart is plotted with the first argument 'x' which can be manipulated to set the position of the bar on the x-axis. Height or 'y' of the bar is the value from the 'grouped\_cars' dataframe.

1. plt.bar([i for i in ind], height=grouped\_cars["4WD"], label="4WD", width=width)
2. plt.bar([i+width for i in ind], height=grouped\_cars["Front"], width=width, bottom=0, label="Front")
3. plt.bar([i+width\*2 for i in ind], height=grouped\_cars["Rear"], label="Rear", width=width, bottom=0)

2. In bar plots, the 'xticks' of the bar must be assigned. If not specified, their values would be equal to the values assigned to 'ind' variable, shown in the code above. To position the 'xticks' at the center for each of the three grouped bars, 'i+width' is used.

1. plt.xticks([i+width for i in ind],["Mileage in City","Mileage in Highway","RPM"])

# Pie Chart

Assume, there is a need to find the number of cars based on the number of cylinders they contain.

First, let us see the unique number of cylinders a car contains.

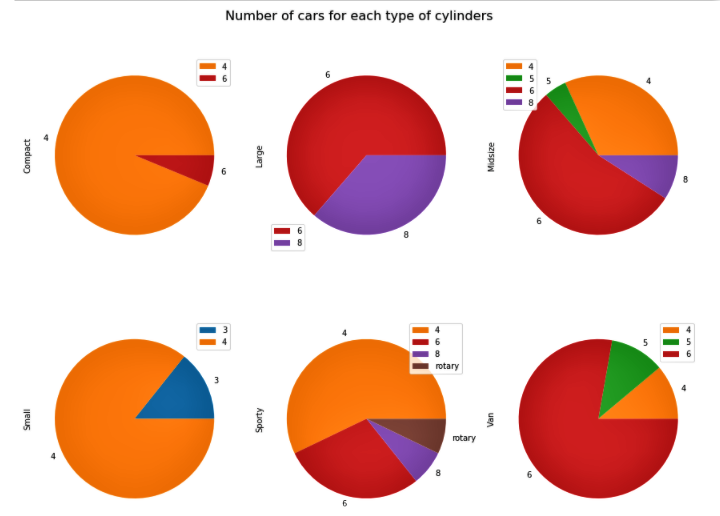
1. cars\_df["Cylinders"].unique()

**Output:** array(['4', '6', '8', '3', 'rotary', '5'], dtype=object)

For graphical representation, let us create a pie chart using the code given below:

1. grouped\_data = cars\_df[["Cylinders", "Type"]].groupby(by = ["Cylinders", "Type"]).size().unstack()
2. fig, ax = plt.subplots(2,3, figsize = (15,10))
3. grouped\_data.plot.pie(ax = ax, subplots = True, fontsize = 20)
4. fig.suptitle("Number of cars for each type of cylinders", fontsize=26, x = 1, y = 2.1)
5. fig.tight\_layout(rect=[0,0,2,2])

The output of the above code is the pie chart shown below, which clearly depicts the distribution of number of cylinders of different types of cars.



Let us analyse the code to understand the pie chart shown above, as follows:

1. The data is grouped on 'Cylinders' and 'Type' using the groupby() method.

1. grouped\_data = cars\_df[["Cylinders", "Type"]].groupby(by = ["Cylinders", "Type"]).size().unstack()

2. The pie() method returns patches, texts, and autotexts. Patches control individual slices of the pie chart. Refer to the documentation to know more about the return values.

1. fig, ax = plt.subplots(2, 3, figsize = (15,10))
2. grouped\_data.plot.pie(ax = ax, subplots = True, fontsize=20)
3. fig.suptitle("Number of cars for each type of cylinders ", fontsize=26,x = 1, y = 2.1)

3. The 'tight\_layout' is used to create a rectangle. The 'rect' feature takes in a list of values that represent the sides of the rectangle in the following order:

* left
* bottom
* right
* top

1. fig.tight\_layout(rect=[0,0,2,2])

Parameters like explode, legend, autopct, etc. can be used to make a pie chart more informative. The 'explode' parameter explodes or pops out the slice that needs to be highlighted. The 'legend' parameter gives information about what data each slice is representing; like in this scenario, the legend is representing the number of cylinders each type of car contains. The 'autopct' parameter gives the format to the labels as per the user.

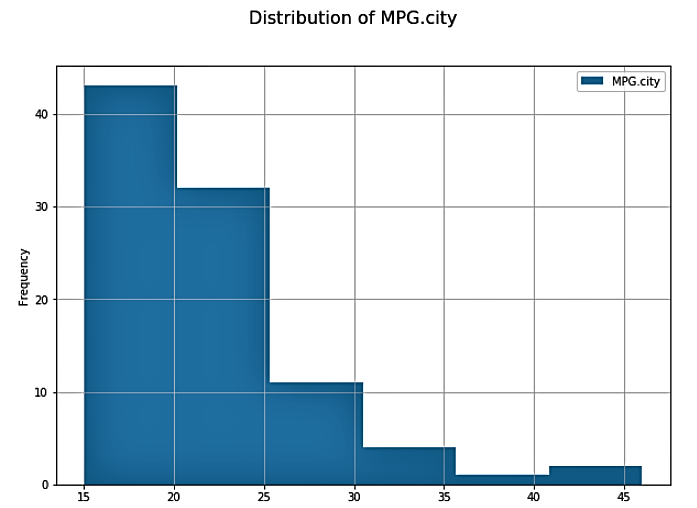
**Histogram**

A histogram is a graphical representation of the distribution of numerical data. In a histogram, the height of the bar represents the frequency in the class interval for that dataset.

Assume, there is a need to plot the range of mileage under which most cars fall.  For graphical representation, let us plot a histogram using the code given below:

1. cars\_df["MPG.city"].plot(kind="hist", grid=True, figsize=(10,7), bins=6)
2. plt.suptitle("Distribution of MPG.city", fontsize=16)
3. plt.legend()

The output of the above code is a histogram, as shown below, from which it can be inferred that most cars have a mileage between 15 and 20.

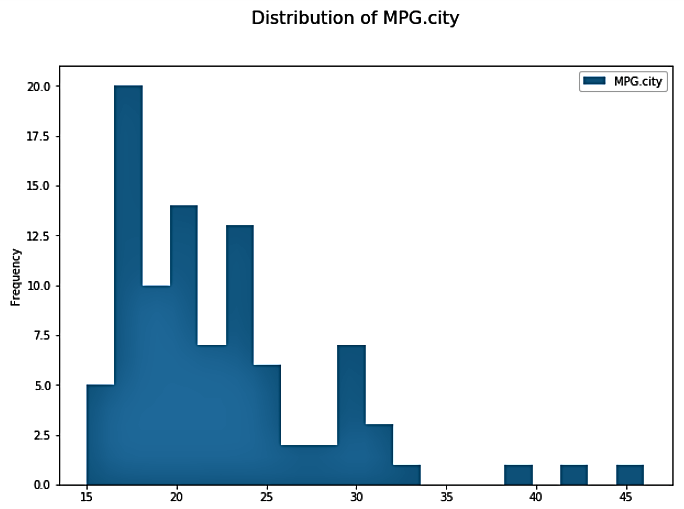


The 'grid' parameter whose default value is False, can be omitted as shown in the graph below.

To get more specific information, the class interval of the data can be changed using the 'bins' parameter. Let us increase the number of bins from 6 to 20, as shown in the code below:

1. cars\_df["MPG.city"].plot(kind="hist", bins=20, figsize=(10,7))
2. plt.suptitle("Distribution of MPG.city", fontsize=16)
3. plt.legend()

The output of the above code is a histogram, as shown below, from which it can be inferred that most cars have a mileage in the range of 16 to 18, if we consider the range to be in intervals of two.

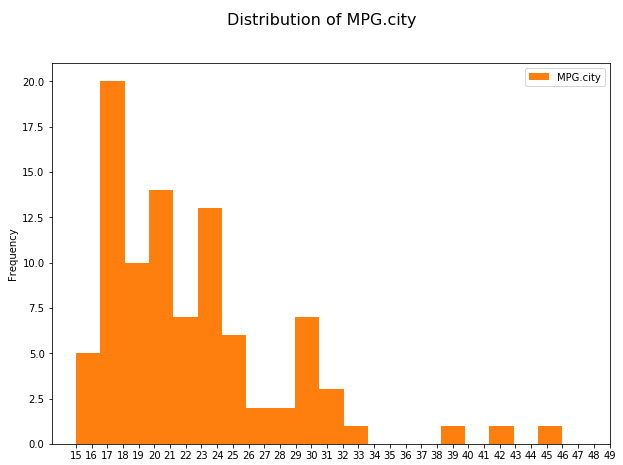


The above graph is difficult to read because of the values on the x-axis. To address this issue, custom values can be set using the xticks() method.

The colour of a histogram can be changed using the 'color' parameter. It is a built-in feature of matplotlib and can take values between C(0-9). The code below implements the above requirements as shown below:

1. cars\_df["MPG.city"].plot(kind="hist", bins=20, figsize=(10,7),color="C1")
2. plt.suptitle("Distribution of MPG.city", fontsize=16)
3. plt.xticks(range(15,50))
4. plt.legend()

The output of the above code is a histogram, as shown below, from which it can be inferred that the highest interval is between 16 and 17.

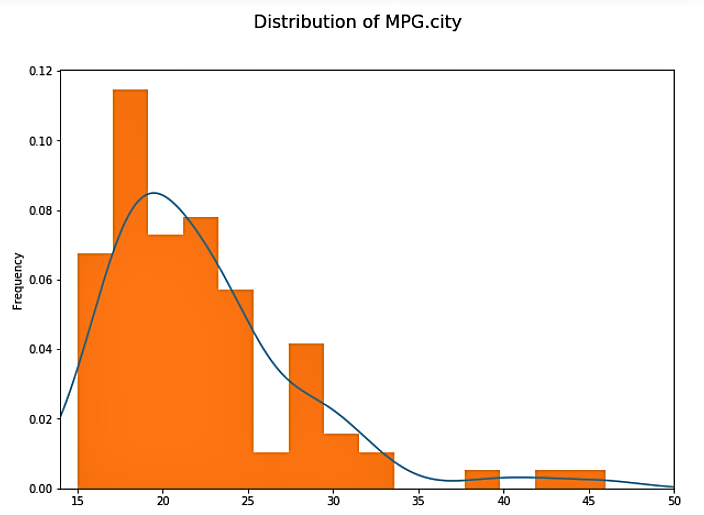


**A probability density plot** can be created by making a histogram smooth and continuous using an estimation function. It can be useful in visualising 'shape' of the data as a continuous replacement for a discrete histogram.

The 'kind = density' parameter plots the density line of the data. The 'density = True' parameter returns the probability densities of each bar of the histogram. And, the xlim( ) method is used to set the limit of the axis between 14 and 50, as shown in the code below:

1. cars\_df["MPG.city"].plot(kind="density")
2. cars\_df["MPG.city"].plot(kind="hist",bins=15,density=True,figsize=(10,7))
3. plt.suptitle("Distribution of MPG.city",fontsize=16)
4. plt.xlim(14,50)

The most common estimation function is the 'kernel density estimation' technique that lets you create a smooth curve over a given set of data, as shown in the plot below:

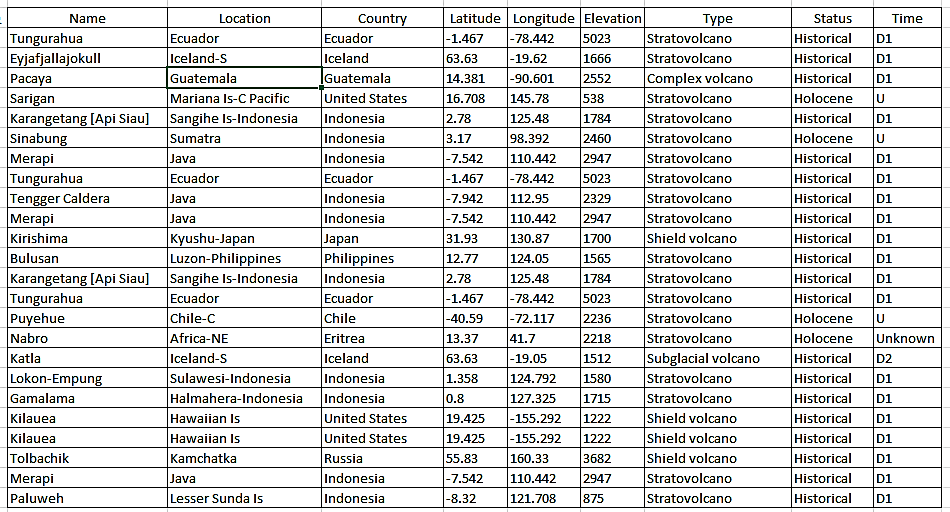


**Note:** A stacked histogram can also be created similar to a bar chart. You can explore stacking by using the parameter 'stacked = True'.

Assume, there is a need to visualise all the volcanoes around the world on a world map.

Let us consider the National Oceanic and Atmospheric Administration (NOAA) dataset, which contains data about volcanoes. It contains names of countries, longitude, latitude, and several other columns. Recall that data with location information is an example of geospatial data.

[Click here](https://infyspringboard.onwingspan.com/common-content-store/Shared/Shared/Public/lex_auth_013197568143024128565_shared/web-hosted/assets/volcanodata20101611112415803.zip) to download the dataset. A sample of the same is shown below:



Let us identify the number of volcano occurrences per year for the countries available in the dataset.

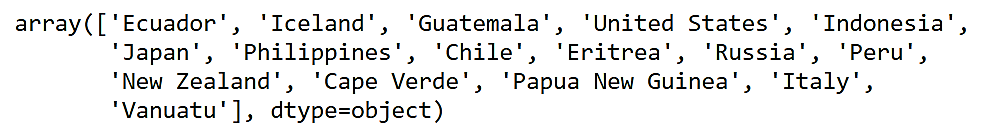
Firstly, let us import the data, using the code given below:

1. *#Importing necessary libraries*
2. import numpy as np
3. import pandas as pd
4. import matplotlib.pyplot as plt
5. *#Importing the dataset*
6. df\_world = pd.read\_csv('volcano\_data\_2010.csv')
7. df\_world

Next, let us consider the countries available in the dataset, using the code given below:

1. *#Viewing the countries in the dataframe*
2. df\_world['Country'].unique()

**Output:**



Now, let us get the count of volcanoes per country, using the code given below:

1. *#This patch of code is used to count the events of volcanic activity per country*
2. df\_countries =df\_world['Year'].groupby(df\_world['Country']).count()
3. df\_countries = df\_countries.reset\_index()
4. df\_countries.rename(columns ={'Year':'count'}, inplace = True)
5. df\_countries = df\_countries.sort\_values('count', ascending = False)
6. df\_countries.reset\_index(drop=True, inplace = True)
7. df\_countries

**Output:**



It can be observed from the above code snippet that the country-wise count of the 'Year' occurrences has been obtained. The column 'Year' is renamed as 'count' and the dataframe is sorted in the descending order of count. To reset the index of the 'df\_countries' dataframe, the reset\_index() method is used.

Let us now import matplotlib's inbuilt colormaps and plotly for the map, using the code given below:

1. *#Choropleth map is a utility of plotly, therefore we import the specific libraries*
2. import plotly.offline as po
3. import plotly.graph\_objs as go
4. from plotly.offline import init\_notebook\_mode,iplot
5. po.init\_notebook\_mode(connected=True)

In the code given below, the dict() method is used to pass the values for 'type' as 'choropleth', 'locations' as the column 'Country', and 'locationmode' as 'country names'. The value assigned to 'text' is also the column 'Country' and 'colorbar' is set with custom values for 'colorscale' in rgb() to get different shades of red.

1. *#Since the Choropleth maps receive input as dictionary, we make the following modifications to the df*
2. data = dict(type='choropleth',
3. locations = df\_countries['Country'],
4. locationmode = 'country names', z = df\_countries['count'],
5. text = df\_countries['Country'], colorbar = dict(title='Frequency'),
6. colorscale=[[0,"rgb(146, 43, 33)"],[0.35,"rgb(176,58,46)"],[0.5,"rgb(192,57,43)"],
7. [0.6,"rgb(203,67,53)"],[0.7,"rgb(231,76,60)"],[1,"rgb(205,97,85)"]],
8. autocolorscale = False,
9. reversescale = True,
10. marker = dict(
11. line = dict (
12. color = 'rgb(180,180,180)',
13. width = 0.5
14. ) ))

Let us pass the values for 'title' and 'geo' in 'layout'.

In 'geo', True or False can be assigned to 'showframe', 'showlakes', 'showcoastlines', 'showland' etc.

In 'projection', the 'type' can be set to one of the existing types ('equirectangular', 'mercator', 'orthographic', 'naturalearth', 'kavrayskiy7', 'miller', 'robinson', 'eckert4','azimuthal equal area', 'azimuthal equidistant' etc.).

Let us chose 'mercator' for the choropleth map of volcanoes, as shown in the code given below:

1. *#We prepare the data to be fed into the map plotting*
2. layout = dict(title = 'Volcanoes distribution',
3. geo = dict(showframe = False, projection = {'type':'mercator'},showlakes = False,
4. showcoastlines = True,showland = True,
5. landcolor = "rgb(229,229,229)"))

Finally, let us plot the map, using the code given below:

1. *#Plotting the choropleth map*
2. choromap = go.Figure(data = [data], layout = layout)
3. iplot(choromap, validate=False)

The output of the above code is shown below:



From the above chloropleth map it can be infered that darker the shade of red, higher the number of volcanoes in that country. For example, Indonesia has the darkest shade, since it contains 26 volcanoes.**nds-On Try Out**

Consider the credit card dataset which contains the following columns:

* **CLIENTNUM:** Primary key of the dataset
* **Attrition\_Flag:** Indicates if a customer is retained or attrited
* **Customer\_Age**: Age of the customer
* **Gender:**Gender of the customer
* **Dependent\_count:** Number of people dependent on the customer
* **Education\_Level:** Highest level of education of the customer
* **Income\_Category:** Range of income of the customer
* **Credit\_Limit:** Credit card limit
* **Total\_Revolving\_Bal:** Pending balance of the credit
* **Avg\_Purchase:** Amount of purchase made by the customer on credit card
* **Total\_Trans\_Amt:** Total transaction amount

[Click here](https://infyspringboard.onwingspan.com/common-content-store/Shared/Shared/Public/lex_auth_0126051881793126401437_shared/web-hosted/assets/CreditCardDV1611145739017.zip) to download the dataset.

Based on the above information, import relevant libraries, and perform the following steps:

1. Create a bivariate plot to find if there is a correlation between credit card limit and average purchase made on the card.
2. Visualise the distribution of values for credit card limit and average purchase made on the card. Also, identify the outliers in the data, if any.
3. Provide a visual representation of the number of customers in each income group using a bar chart.
4. Plot the frequency distribution of the total transaction amount.
5. Graphically represent the percentage of customers retained and those attrited. Highlight the latter by slicing it apart from the main pie.