Matplotlib

Visualising data using plots and charts helps in gaining more insights into the data and also aids in presenting it. Graphics and visuals, if used intelligently and innovatively, can convey a lot more insights than what raw data alone can convey. Matplotlib serves the purpose of providing multiple functions to build graphs from the data stored in your lists, arrays, etc.

Facts and Dimensions

- Facts are numerical data.
- Dimensions are metadata, which is data that explains some other data, associated with fact variables.

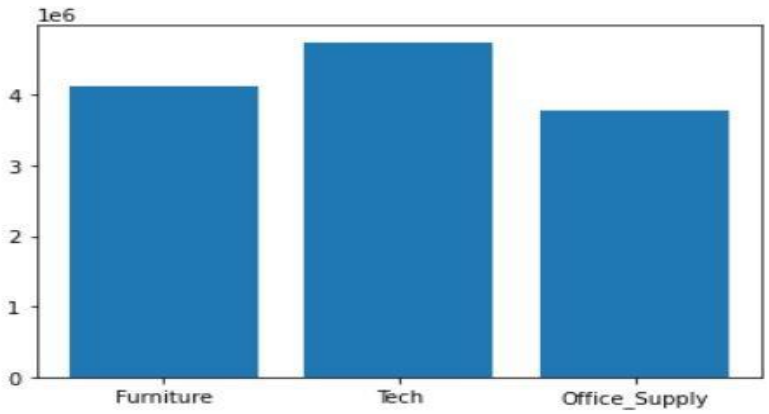
Bar Graph Visualisation Link to data

```
# importing numpy and the pyplot package of
matplotlib import numpy as np import
matplotlib.pyplot as plt
```

```
# Creating an array with product categories product_category =
np.array(['Furniture', 'Tech', 'Office_Supply']) sales =
np.array([4110451.90, 4744557.50, 3787492.52])
```

```
plt.bar(product_category, sales)
plt.show()
```

Output:



Input:

```
# plotting the bar graph with product categories on x-axis and sales
amount of y-axis
plt.bar(product_category, sales) # adding title to the graph plt.title(
"Sales across Product Categories", fontdict = {'fontsize': 20,
'fontweight': 5, 'color': 'Green'})
# labeling axes changing color of the bars in the bar graph plt.xlabel(
"Product_Categories", fontdict = {'fontsize': 10,     'fontweight':
3, 'color': 'Blue'}) plt.ylabel("Sales", fontdict = {'fontsize': 10,
'fontweight': 3, 'color':
'Blue'})
# necessary command to display the created graph
plt.show()
```

```
tick_labels = ["0L", "10L", "20L", "30L", "40L", "50L", "60L", "70L"
     ] plt.yticks(tick_values, tick_labels)
```

Output:

Scatter plot Visualisation

A scatter plot, as the name suggests, displays the scatter of the data. It can be helpful in checking for any relationship pattern between two quantitative variables and detecting the presence of outliers within them.

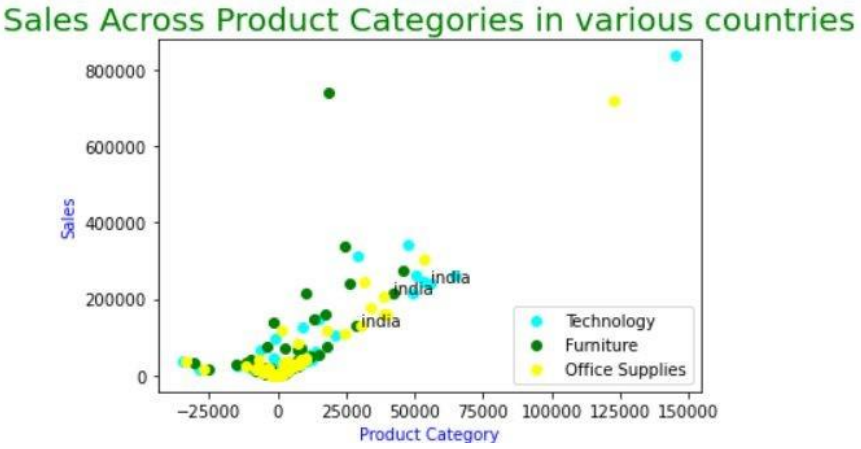
Matplotlib also offers a feature that allows incorporating a categorical distinction between the points plotted on a scatter plot. You can colour-code the points based on the category and distinguish them from each other.

Another feature of a scatter plot is that the points can be further distinguished over another dimension variable using labels. In the example we are using here, you have another array called 'country' that tells you the country where the sales were made. Suppose you want to highlight the points belonging to a particular country in the figure given above. Let's see how you can do that.

Input:

```
product_categories = np.array(["Technology", "Furniture",
"Office Supplies"]) colors = np.array(["cyan", "green", "yellow"
])
# plotting the scatter plot with color coding the points belonging
to different categories  for color, category in zip(colors,
product_categories):     sales_category = sales[product_category ==
category]     profit_category = profit[product_category == category]
plt.scatter(profit_category, sales_category, c = color, label =
category)
# labeling points that belong to country "India" for xy in
zip(profit[country == "India"], sales[country == "India"]):
    plt.annotate(s = "india", xy = xy)
# Adding and formatting title
plt.title("Sales Across Product Categories in various countries",
fontdict={'fontsize': 20, 'fontweight' : 5, 'color' : 'Green'})
# Labeling Axes
plt.xlabel("Product Category", fontdict={'fontsize': 10, 'fontweight' : 3,
'color' : 'Blue'}) plt.ylabel("Sales", fontdict={'fontsize': 10,
'fontweight' : 3,       'color' :
'Blue'})
# Adding legend for interpretation of points
plt.legend()
plt.show()
```

Output:



Line Graph and 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.

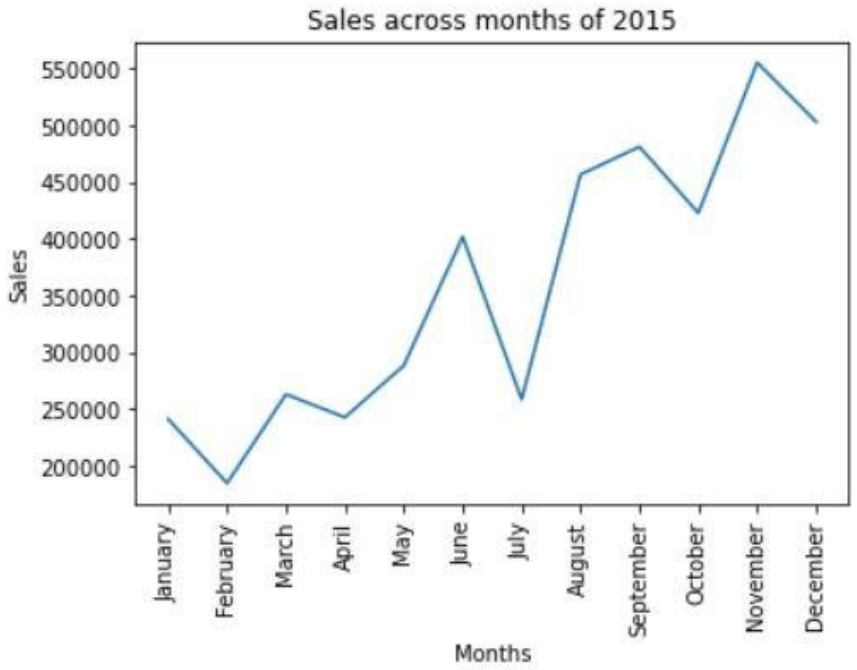
Input:

```
# importing the required libraries import numpy as np import
matplotlib.pyplot as plt # Sales data across months months = np.array([
'January', 'February', 'March', 'April', 'May', 'June',
'July', 'August', 'September', 'October', 'November', 'December'])
sales = np.array([241268.56, 184837.36, 263100.77, 242771.86, 288401.05
,
401814.06, 258705.68, 456619.94, 481157.24, 422766.63, 555279.03  ,
503143.69])
```

```
# plotting a line chart
plt.plot(months, sales)
# adding title to the
chart
```

```
plt.title("Sales across months of 2015")
# labeling the axes plt.xlabel("Months"
) plt.ylabel("Sales")
# rotating the tick values of x-
axis plt.xticks(rotation= 90) #
displaying the created plot
plt.show()
```

Output:



A line graph is extremely helpful when you want to understand the trend of a variable. Some key industries and services that rely on line graphs are financial markets, weather forecast, etc.

➔ Histogram

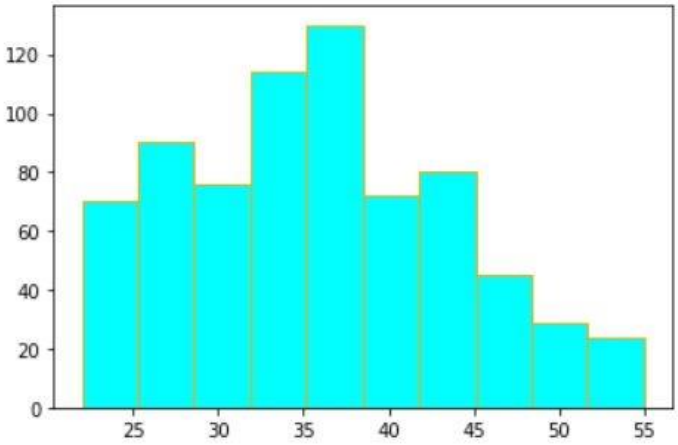
A histogram is a frequency chart that records the occurrence of an entry or element in a data set. It is useful when you want to understand the distribution of a given series.
Input:

```
# importing the required libraries - numpy, matplotlib.pyplot
import numpy     as np
```

```
import matplotlib.pyplot as plt

# data corresponding to age of the employees in the company age
= np.array([23, 22, 24, 24, 23, 23, 22, 23, 24,......])
```

```
# plotting a histogram plt.hist(age, bins = 10, edgecolor
= "orange", color = "cyan") plt.show()
```

Output:



Box Plot Visualisation

Box plots are extremely effective in summarising the spread of large data into a visual representation. They take the help of percentiles to divide a 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 descending order.
The figure given below summarises what a boxplot represents.

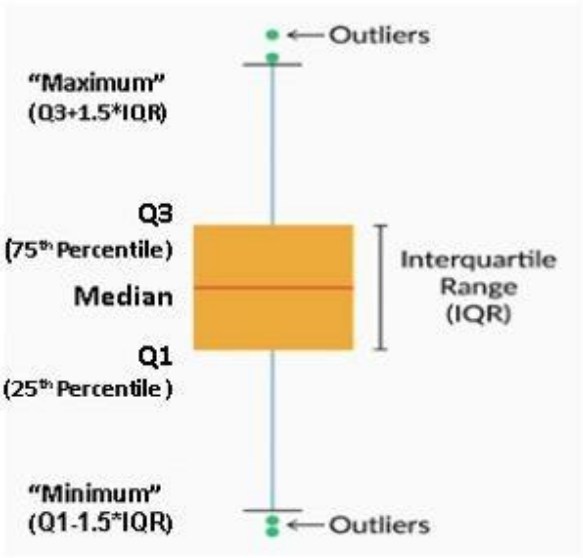


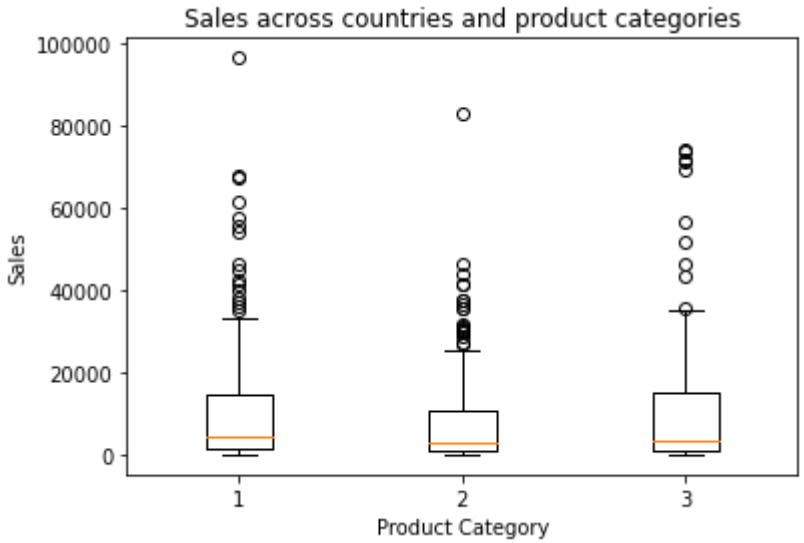
Fig: Boxplot

Input:

```
import numpy as np import

matplotlib.pyplot as plt

# Creating arrays with sales in different countries across each category:
'Furniture', 'Technology' and 'Office Supplies' sales_technology =

np.array ([1013.14, 8298.48, 875.51, 22320.83, ......])

sales_office_supplies = np.array ([1770.13, 7527.18, 1433.65, 423.3,

......]) sales_furniture = np.array ([981.84, 10209.84,
156.56, .....])
```

```
# plotting box plot for each category plt.boxplot([sales_technology,
sales_office_supplies, sales_furniture])
# adding title to the graph plt.title("Sales across
countries and product categories")

# labeling the axes plt.xlabel(

"Product Category") plt.ylabel("Sales"

)

# Replacing the x ticks with respective category
#plt.xticks((1,2,3), ["Technology","Office_supplies", "Furniture"])
plt.show()
```

Output:



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, that is, the 50th percentile.
- Interquartile range (IQR): These are data points between the 25th and 75th percentile values.
- Outliers: These are data points that differ significantly from other observations and lie beyond the whiskers.

Subplots

Sometimes, it is beneficial to draw different plots on a single grid next to each other in order to get a better overall view. Different plots presented in a single plot object are commonly referred to as subplots.

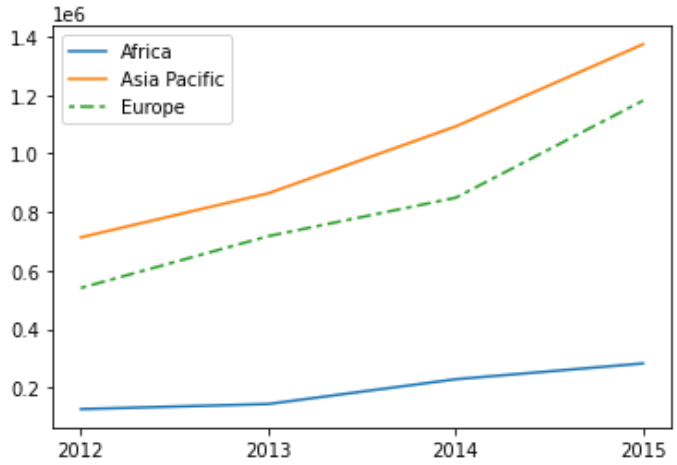
Input:

```
# importing numpy and the pyplot package of matplotlib import
numpy as np import matplotlib.pyplot as plt# Data for sales made
by the company across three regions over the years 2012 to 2015
years = np.array(['2012', '2013', '2014', '2015']) sales_Africa =
np.array([127187.27, 144480.70, 229068.79, 283036.44])
sales_Asia_Pacific = np.array([713658.22, 863983.97, 1092231.65,
1372784.40])
sales_Europe = np.array([540750.63, 717611.40, 848670.24,
1180303.95])
```

Single Chart:

```
# Plotting the trend of sales in African region over the 4 years #
Check the array names before plotting to avoid error fig, ax =
plt.subplots() # It initiates a figure which will be used to
comprise multiple graphs in a single chart.
africa,= ax.plot(years, sales_Africa) africa.set_label("Africa")
 aspac,= ax.plot(years,
sales_Asia_Pacific) aspac.set_label("Asia
Pacific")
 europe,= ax.plot(years,
sales_Europe) europe.set_label(
"Europe") europe.set_dashes([2, 2, 5
, 2])

plt.legend() plt.show()
```

Output:



Subplots are a good way to show multiple plots together for comparison. They provide the ability to create an array of plots, and you can have multiple charts next to each other such as the elements of an array as given below.

Multiple Charts:

Input:

```
# Plotting the trend of sales in all three regions over the 4 years
using subplots fig, ax = plt.subplots(ncols = 3, sharex = True)
africa,= ax[0].plot(years, sales_Africa) africa.set_label("Africa")
 aspac,= ax[1].plot(years,
sales_Asia_Pacific) aspac.set_label("Asia
Pacific")
 europe,= ax[2].plot(years,
sales_Europe) europe.set_label("Europe"
) europe.set_dashes([2, 2, 5, 2])
 plt.legend([africa, aspac, europe],  ["Africa", "Asia
Pacific", "Europe"]) plt.show()
```

Output: