VISHNU
UNIVERSAL LEARNING

Data Analytics

VISHNU
UNIVERSAL LEARNING

Data Analytics Unit-4

UNIT-4: Exploratory Data Analysis (10 Periods)

Data Visualization, Reading and getting data (External Data): Using CSV files, XML files, Web Data, JSON files, Databases, Excel files. Charts and Graphs: Histograms, Boxplots, Bar Charts, Line Graphs, Scatter plots, Pie Charts

VISHNU
UNIVERSAL LEARNING

Flow of Presentation

- Data Visualization,
- Reading and getting data (External Data): Using CSV files,
- XML files,
- Web Data,
- JSON files, Databases, Excel files.
- Charts and Graphs: Histograms, Boxplots, Bar Charts, Line Graphs, Scatter plots, Pie Charts

VISHNU
UNIVERSAL LEARNING

What is Data Visualization?

- Data visualization is the graphical representation of information and data.

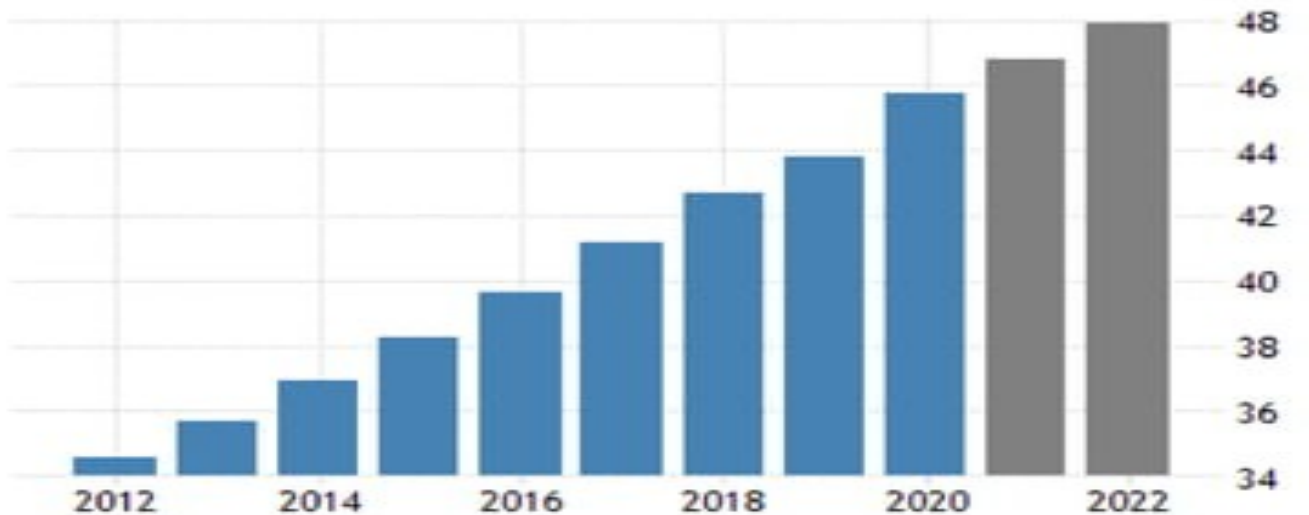
- By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends and patterns in data.

- In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions.

The advantages and benefits of data visualization

- Data visualization is another form of visual art that grabs our interest and keeps our eyes on the message.
- If we can see something, we internalize it quickly.

- When we see a chart, we quickly see trends and outliers. A good visualization tells a story, removing the noise from data and highlighting the useful information.



VISHNU
UNIVERSAL LEARNING

Application of Data Visualization

- Presenting statistics
- Mapping
- To show change over time
- To Compare Values
- To Show Connections

Common general types of data visualization:

- Charts
- Graphs
- Maps
- Infographics

What Are Data Visualization Tools?

- Some of the best data visualization tools include Google Charts, Tableau, Grafana, Chartist, FusionCharts, Datawrapper, Infogram, and ChartBlocks etc. These tools support a variety of visual styles, be simple and easy to use, and be capable of handling a large volume of data.
- Modern data visualisation tools and advanced software are on the market. A data visualisation tool is a software that is used to visualise data. The features of each tool vary, but at their most basic, they allow you to input a dataset and graphically alter it. Most, but not all, come with pre-built templates for creating simple visualisations.

Reading and getting data

- Using CSV files

A **CSV (Comma Separated Values)** file is a form of plain text document which uses a particular format to organize tabular information. CSV file format is a bounded text document that uses a comma to distinguish the values. Every row in the document is a data log. Each log is composed of one or more fields, divided by commas. It is the most popular file format for importing and exporting spreadsheets and databases.

Reading a CSV File

- There are various ways to read a CSV file that uses either the CSV module or the pandas library.

Reading and getting data

USing csv.reader():

```
import csv
 # opening the CSV file
with open('Giants.csv', mode ='r')as file:
  # reading the CSV file
  csvFile = csv.reader(file)
   # displaying the contents of the CSV file
  for lines in csvFile:
        print(lines)
```

Reading and getting data

Using pandas.read_csv() method:

```
import pandas
 # reading the CSV file
csvFile = pandas.read_csv('Giants.csv')
 # displaying the contents of the CSV file
print(csvFile)
```

VISHNU
UNIVERSAL LEARNING

Reading and Writing XML Files in Python

- **Extensible Markup Language**, commonly known as XML is a language designed specifically to be easy to interpret by both humans and computers altogether. The language defines a set of rules used to encode a document in a specific format. In this article, methods have been described to read and write XML files in python.
- **Note:** In general, the process of reading the data from an XML file and analyzing its logical components is known as **Parsing**. Therefore, when we refer to reading a xml file we are referring to **parsing the XML document**.
- Two libraries that could be used for the purpose of xml parsing are:
- BeautifulSoup used alongside the lxml parser
- Elementtree library.

Using BeautifulSoup alongside with lxml parser

- Beautiful Soup supports the HTML parser included in Python's standard library, but it also supports a number of third-party Python parsers. One is the lxml parser (used for parsing XML/HTML documents).

pip install beautifulsoup4

pip install lxml

```xml
<?xml version="1.0" encoding="utf-8"?>
<saranghe>
 <child name="Frank" test="0">
  FRANK likes EVERYONE
 </child>
 <unique>
  Add a video URL in here
 </unique>
 <child name="Texas" test="1">
  TEXAS is a PLACE
 </child>
 <child name="Frank" test="2">
  Exclusively
 </child>
 <unique>
  Add a workbook URL here
 </unique>
 <data>
  Add the content of your article here
  <family>
   Add the font family of your text here
  </family>
  <size>
   Add the font size of your text here
  </size>
 </data>
</saranghe>
```

```
from bs4 import BeautifulSoup
# Reading the data inside the xml
# file to a variable under the name
# data
with open('dict.xml', 'r') as f:
        data = f.read()
# Passing the stored data inside
# the beautifulsoup parser, storing
# the returned object
Bs_data = BeautifulSoup(data, "xml")
# Finding all instances of tag
# `unique`
b_unique = Bs_data.find_all('unique')
print(b_unique)
# Using find() to extract attributes
# of the first instance of the tag
b_name = Bs_data.find('child', {'name':'Frank'})
print(b_name)
# Extracting the data stored in a
# specific attribute of the
# `child` tag
value = b_name.get('test')
print(value)
```

OUTPUT:

```
[<unique>
  Add a video URL in here
 </unique>, <unique>
  Add a workbook URL here
 </unique>]
<child name="Frank" test="0">
  FRANK likes EVERYONE
 </child>
0
```

Writing an XML File

- Writing a xml file is a primitive process, reason for that being the fact that xml files aren’t encoded in a special way. Modifying sections of a xml document requires one to parse through it at first. In the below code we would modify some sections of the aforementioned xml document.

```
from bs4 import BeautifulSoup

# Reading data from the xml file
with open('dict.xml', 'r') as f:
        data = f.read()

# Passing the data of the xml
# file to the xml parser of
# beautifulsoup
bs_data = BeautifulSoup(data, 'xml')

# A loop for replacing the value
# of attribute `test` to WHAT !!
# The tag is found by the clause
# `bs_data.find_all('child', {'name':'Frank'})`
for tag in bs_data.find_all('child', {'name':'Frank'}):
        tag['test'] = "WHAT !!"


# Output the contents of the
# modified xml file
print(bs_data.prettify())
```

```
<?xml version="1.0" encoding="utf-8"?>
<saranghe>
 <child name="Frank" test="WHAT !!">
  FRANK likes EVERYONE
 </child>
 <unique>
  Add a video URL in here
 </unique>
 <child name="Texas" test="1">
  TEXAS is a PLACE
 </child>
 <child name="Frank" test="WHAT !!">
  Exclusively
 </child>
 <unique>
  Add a workbook URL here
 </unique>
 <data>
  Add the content of your article here
  <family>
   Add the font family of your text here
  </family>
  <size>
   Add the font size of your text here
  </size>
 </data>
</saranghe>
```

Reading and getting web data

- Requests is a versatile HTTP library in python with various applications. One of its applications is to download a file from web using the file URL.
  Installation: First of all, you would need to download the requests library. You can directly install it using pip by typing following command:
- pip install requests

```
# imported the requests library
import requests
image_url =
"https://www.python.org/static/community_logos/python-logo-master-
v3-TM.png"

# URL of the image to be downloaded is defined as image_url
r = requests.get(image_url) # create HTTP response object

# send a HTTP request to the server and save
# the HTTP response in a response object called r
with open("python_logo.png",'wb') as f:

    # Saving received content as a png file in
    # binary format

    # write the contents of the response (r.content)
    # to a new file in binary mode.
    f.write(r.content)
```

This small piece of code written above will download the following image from the web.

Download large files

- The HTTP response content (**r.content**) is nothing but a string which is storing the file data. So, it won't be possible to save all the data in a single string in case of large files. To overcome this problem, we do some changes to our program:
- Since all file data can't be stored by a single string, we use **r.iter_content** method to load data in chunks, specifying the chunk size. r = requests.get(URL, stream = True)
  Setting **stream** parameter to **True** will cause the download of response headers only and the connection remains open. This avoids reading the content all at once into memory for large responses. A fixed chunk will be loaded each time while **r.iter_content** is iterated.

```
import requests
file_url =
"http://codex.cs.yale.edu/avi/db-book/db4/slide-dir/ch1-2.pdf"

r = requests.get(file_url, stream = True)

with open("python.pdf","wb") as pdf:
        for chunk in r.iter_content(chunk_size=1024):

                # writing one chunk at a time to pdf file
                if chunk:
                        pdf.write(chunk)
```

Reading and getting JSON files

- JSON (**J**ava**S**cript **O**bject **N**otation) is a popular data format used for representing structured data. It's common to transmit and receive data between a server and web application in JSON format.

In Python, JSON exists as a string. For example:

p = '{"name": "Bob", "languages": ["Python", "Java"]}‘

- It's also common to store a JSON object in a file.

Import json Module

To work with JSON (string, or file containing JSON object), you can use Python's json module. You need to import the module before you can use it.

```
Import  json
```

Python JSON to dict

```
import json

person = '{"name": "Bob", "languages": ["English", "French"]}'
person_dict = json.loads(person)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}
print( person_dict)

# Output: ['English', 'French']
print(person_dict['languages'])
```

Python read JSON file

```
{"name": "Bob",
"languages": ["English", "French"]
}
```

Here's how you can parse this file:

```
import json

with open('path_to_file/person.json', 'r') as f:
  data = json.load(f)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}
print(data)
```

Writing JSON to a file

```
import json

person_dict = {"name": "Bob",
"languages": ["English", "French"],
"married": True,
"age": 32
}

with open('person.txt', 'w') as json_file:
  json.dump(person_dict, json_file)
```

```
# Python program to read
# json file

import json

# Opening JSON file
f = open('data.json')

# returns JSON object as
# a dictionary
data = json.load(f)

# Iterating through the json
# list
for i in data['emp_details']:
        print(i)

# Closing file
f.close()
```

Output:

```
{'emp_name': 'Shubham', 'email': 'ksingh.shubh@gmail.com', 'job_profile': 'intern'}
{'emp_name': 'Gaurav', 'email': 'gaurav.singh@gmail.com', 'job_profile': 'developer'}
{'emp_name': 'Nikhil', 'email': 'nikhil@geeksforgeeks.org', 'job_profile': 'Full Time'}
```

Reading and getting databases

How to Connect to a SQL Database using Python

- Python has several libraries for connecting to SQL databases, including pymysql, psycopg2, and sqlite3.
- First, we need to install the pymysql library using pip:

pip install pymysql

import the pymysql library and connect to the MySQL database using the following code:

```
import pymysql conn = pymysql.connect( host='localhost', user='root', password='password', db='mydatabase', charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor )
```

In the code above, we first import the pymysql library. Then, we use the connect() function to establish a connection to the MySQL database.

We need to provide the following parameters to the connect() function:

host: the hostname or IP address of the MySQL server
user: the username used to authenticate with the MySQL server
password: the password used to authenticate with the MySQL server
db: the name of the database to connect to
charset: the character set to use for the connection
cursorclass: the type of cursor to use for the connection (in this case, we use the DictCursor cursor, which returns rows as dictionaries)

How to Insert Data into a SQL Database using Python

```
try:
    with conn.cursor() as cursor:
        # Create a new record
        sql = "INSERT INTO `users` (`email`, `password`)
VALUES (%s, %s)"
        cursor.execute(sql, ('john@example.com',
'mypassword'))

    # Commit changes
    conn.commit()

    print("Record inserted successfully")
finally:
    conn.close()
```

How to Update Data in a SQL Database using Python

```
try:
    with conn.cursor() as cursor:
        # Update a record
        sql = "UPDATE `users` SET `password`=%s WHERE
`email`=%s"
        cursor.execute(sql, ('newpassword',
'john@example.com'))

    # Commit changes
    conn.commit()

    print("Record updated successfully")
finally:
    conn.close()
```

How to Delete Data from a SQL Database using Python

```
try:
    with conn.cursor() as cursor:
        # Delete a record
        sql = "DELETE FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('john@example.com',))

    # Commit changes
    conn.commit()

    print("Record deleted successfully")
finally:
    conn.close()
```

How to Read Data from a SQL Database using Python

```
try:
    with conn.cursor() as cursor:
        # Read data from database
        sql = "SELECT * FROM `users`"
        cursor.execute(sql)

        # Fetch all rows
        rows = cursor.fetchall()

        # Print results
        for row in rows:
            print(row)
finally:
    conn.close()
```

Reading and getting data from excel files

- An Excel spreadsheet document is called a workbook which is saved in a file with **.xlsx** extension. The first row of the spreadsheet is mainly reserved for the header, while the first column identifies the sampling unit. Each workbook can contain multiple sheets that are also called a worksheets. A box at a particular column and row is called a cell, and each cell can include a number or text value. The grid of cells with data forms a sheet.
- The active sheet is defined as a sheet in which the user is currently viewing or last viewed before closing Excel.

Reading from an Excel file

First, you need to write a command to install the **xlrd** module.

- pip install xlrd

```
# Import the xlrd module
import xlrd

# Define the location of the file
loc = ("path of file")

# To open the Workbook
wb = xlrd.open_workbook(loc)
sheet = wb.sheet_by_index(0)

# For row 0 and column 0
sheet.cell_value(0, 0)
```

Explanation: In the above example, firstly, we have imported the xlrd module and defined the location of the file. Then we have opened the workbook from the excel file that already exists.

VISHNU
UNIVERSAL LEARNING

Reading an excel file using Python using Pandas

Example -

```
# import pandas lib as pd
import pandas as pd
 # read by default 1st sheet of an excel file
dataframe1 = pd.read_excel('book2.xlsx')
 print(dataframe1)
```

Reading an excel file using Python using openpyxl

```
import openpyxl

# Define variable to load the dataframe
dataframe = openpyxl.load_workbook("Book2.xlsx")

# Define variable to read sheet
dataframe1 = dataframe.active

# Iterate the loop to read the cell values
for row in range(0, dataframe1.max_row):
    for col in dataframe1.iter_cols(1, dataframe1.max_column):
        print(col[row].value)
```

VISHNU
UNIVERSAL LEARNING

Reading an excel file using Python using Xlwings

```
# Python3 code to select
# data from excel
import xlwings as xw

# Specifying a sheet
ws = xw.Book("Book2.xlsx").sheets['Sheet1']

# Selecting data from
# a single cell
v1 = ws.range("A1:A7").value
# v2 = ws.range("F5").value
print("Result:", v1, v2)
```

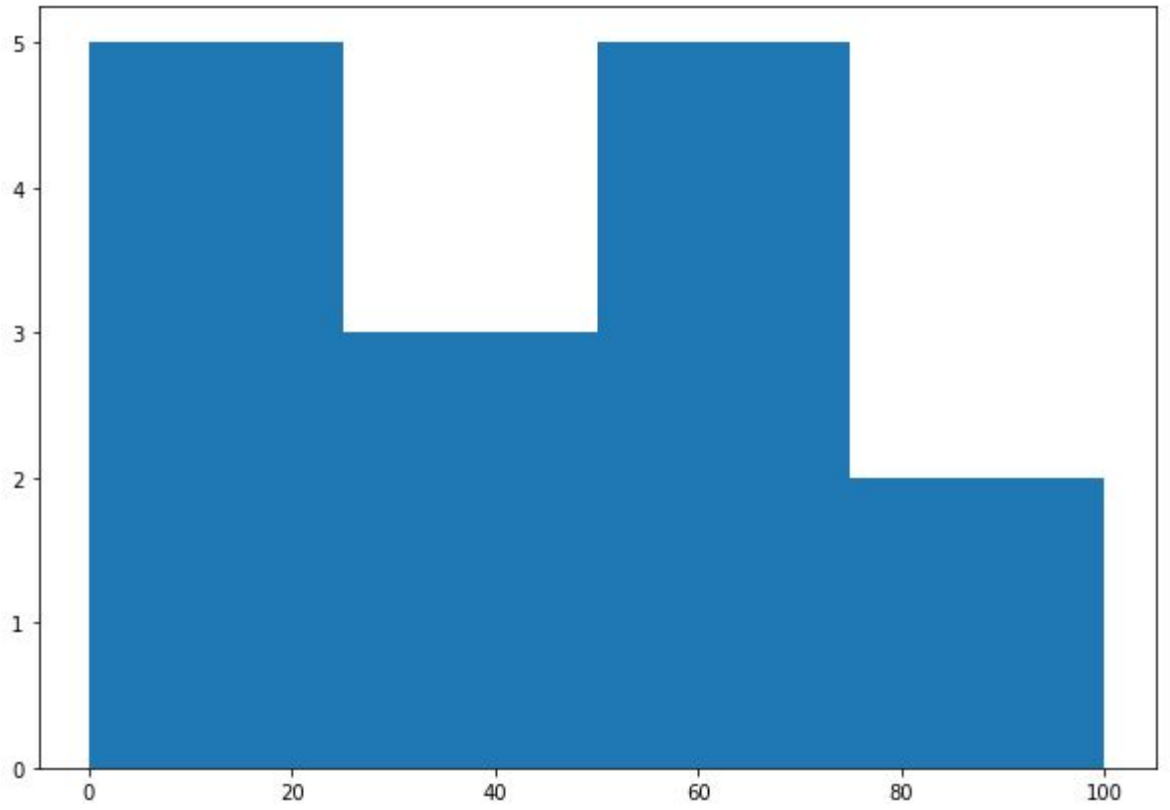
Histogram

- A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution.It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

Creating a Histogram

- To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals.Bins are clearly identified as consecutive, non-overlapping intervals of variables.The matplotlib.pyplot.hist() function is used to compute and create histogram of x.

```
from matplotlib import pyplot as plt
import numpy as np


# Creating dataset
a = np.array([22, 87, 5, 43, 56,
              73, 55, 54, 11,
              20, 51, 5, 79, 31,
              27])

# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(a, bins = [0, 25, 50, 75, 100])

# Show plot
plt.show()
```



Box Plot

- A **Box Plot** is also known as **Whisker plot** is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.

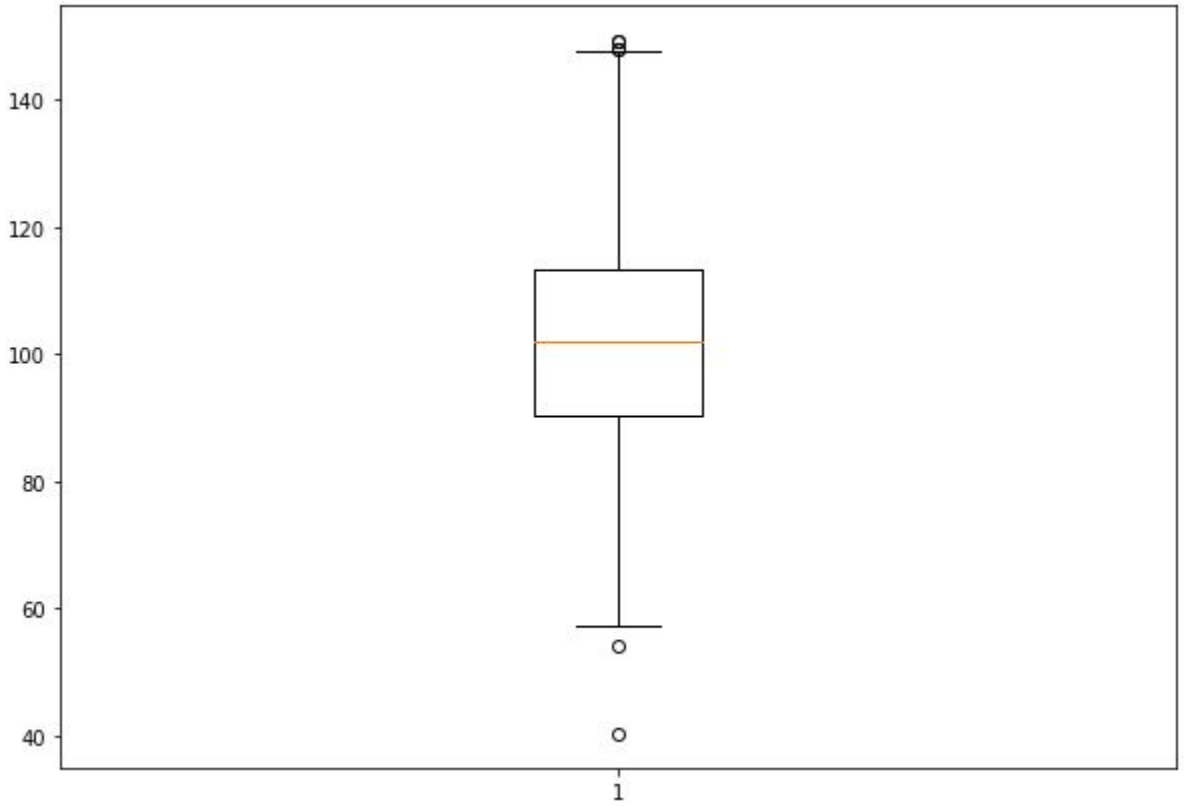
Creating Box Plot

- The matplotlib.pyplot module of matplotlib library provides boxplot() function with the help of which we can create box plots.

Syntax:

- matplotlib.pyplot.boxplot(data, notch=None, vert=None, patch_artist=None, widths=None)

```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np


# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)

fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(data)

# show plot
plt.show()
```



Bar Plot

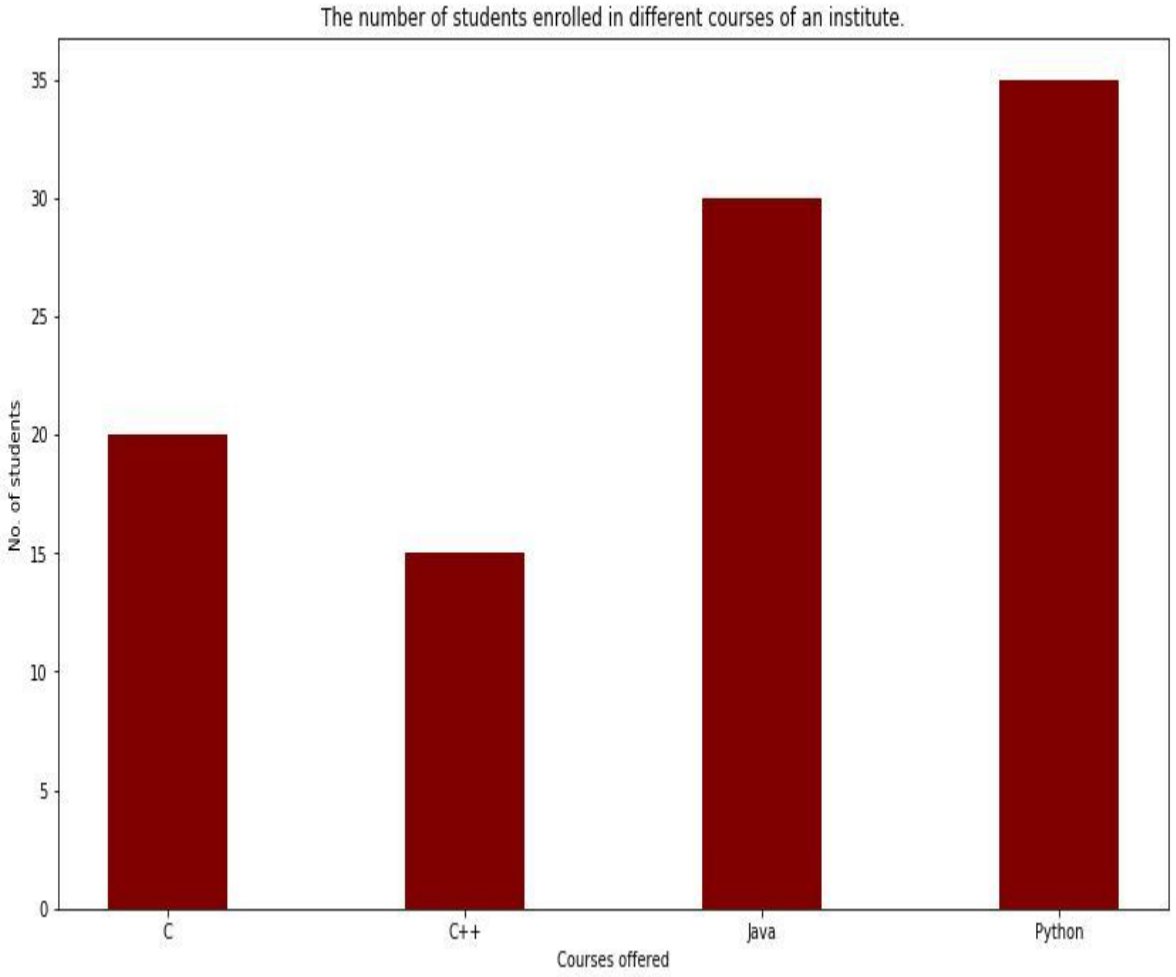
- A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons between the discrete categories. One of the axis of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.

Creating a bar plot

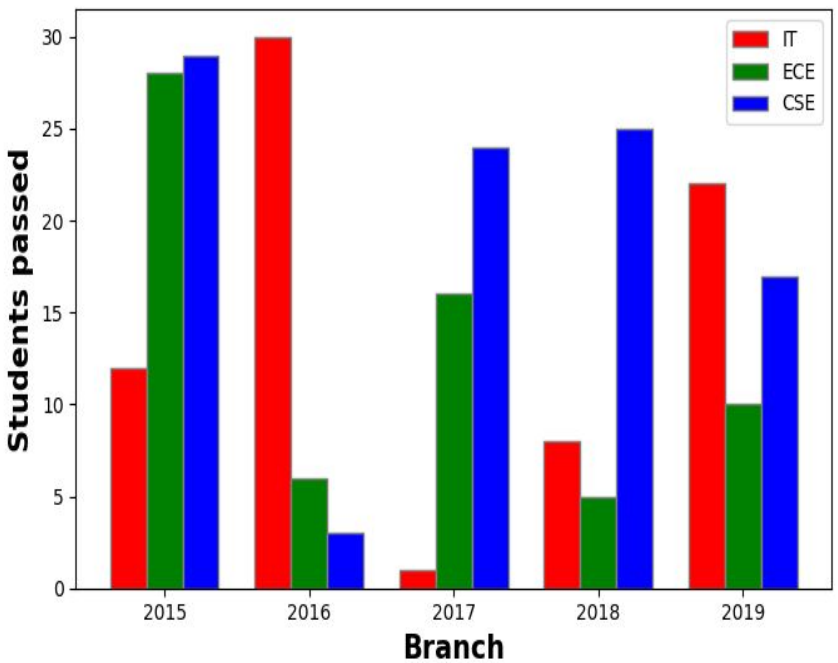
- The **matplotlib** API in Python provides the bar() function which can be used in MATLAB style use or as an object-oriented API. The syntax of the bar() function to be used with the axes is as follows:-

plt.bar(x, height, width, bottom, align)

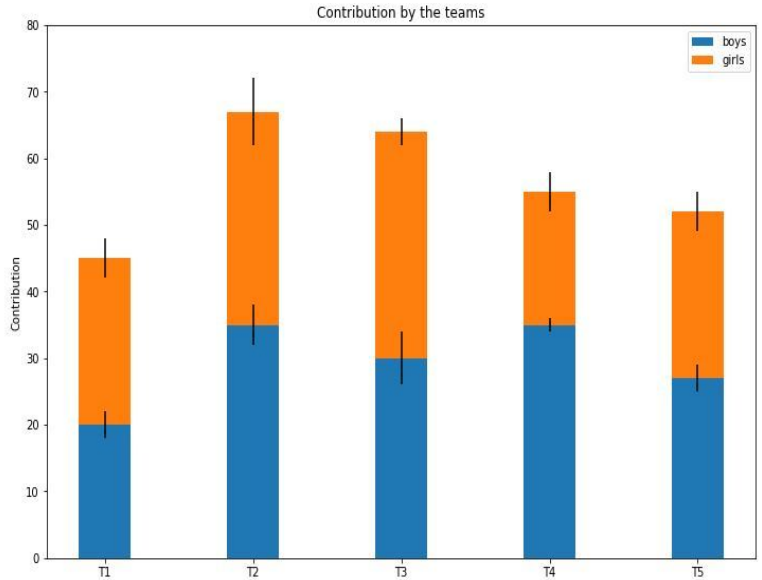
```
import numpy as np
import matplotlib.pyplot as plt


# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
        'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
        width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```



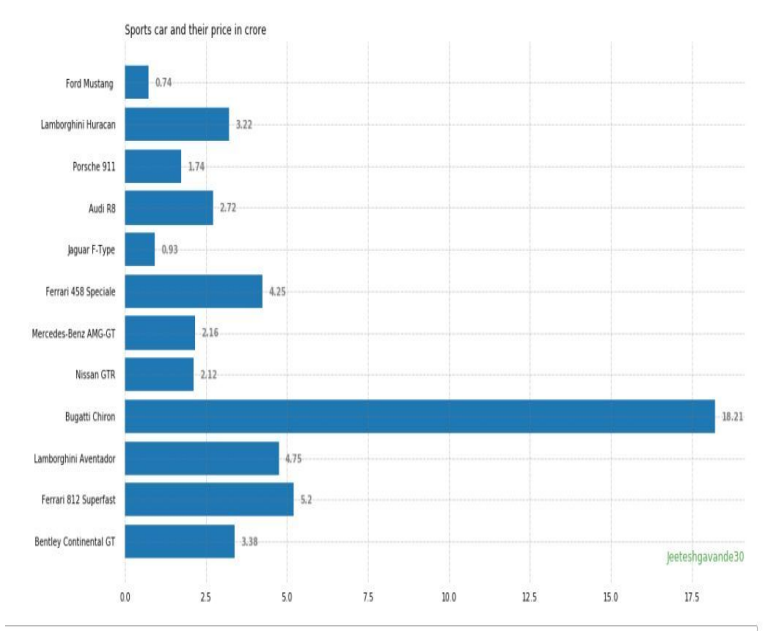
Multiple bar plots



Stacked bar plot



Horizontal bar plot



Line chart

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

```
ax = plt.axes() ax.plot(x, np.sin(x)) ax.set(xlim=(0, 10), ylim=(-2, 2),
  xlabel='x', ylabel='sin(x)', title='A Simple Plot');
```

Scatter plots

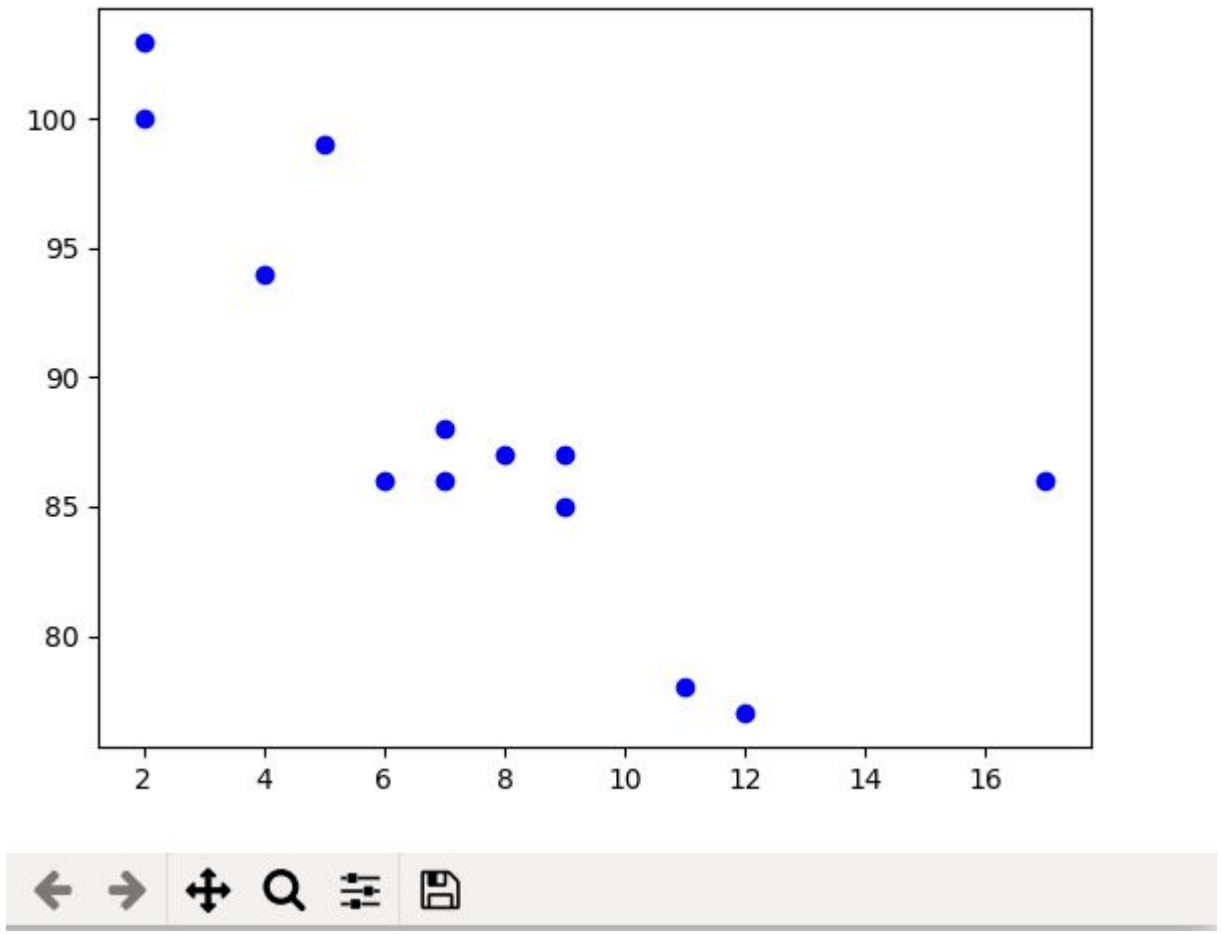
- Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them.
  The **scatter()** method in the matplotlib library is used to draw a scatter plot. Scatter plots are widely used to represent relation among variables and how change in one affects the other.

Syntax

The syntax for scatter() method is given below:

- matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

```
import matplotlib.pyplot as plt


x =[5, 7, 8, 7, 2, 17, 2, 9,
    4, 11, 12, 9, 6]

y =[99, 86, 87, 88, 100, 86,
    103, 87, 94, 78, 77, 85, 86]

plt.scatter(x, y, c ="blue")

# To show the plot
plt.show()
```



Pie Chart

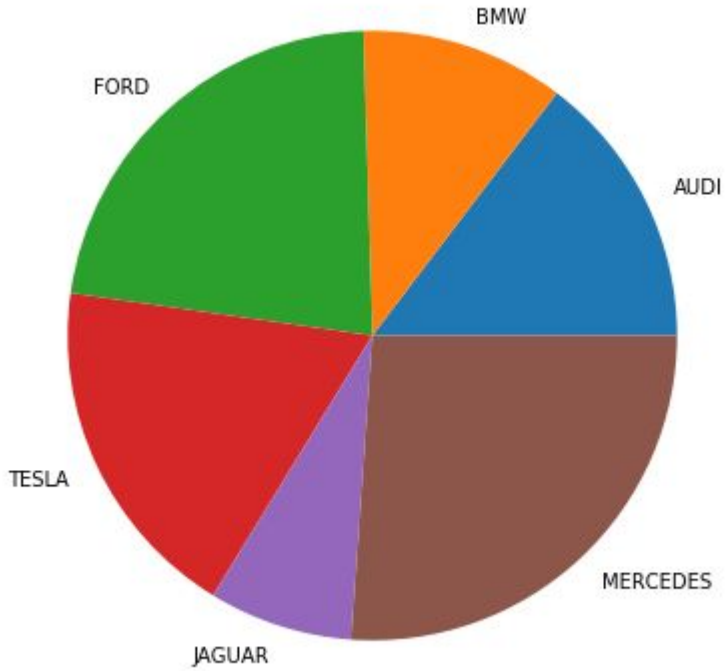
- A **Pie Chart** is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data. The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges. The area of the wedge is determined by the length of the arc of the wedge. The area of a wedge represents the relative percentage of that part with respect to whole data. Pie charts are commonly used in business presentations like sales, operations, survey results, resources, etc as they provide a quick summary.

Creating Pie Chart

Matplotlib API has pie() function in its pyplot module which create a pie chart representing the data in an array.

- **Syntax:** matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)

```
# Import libraries
from matplotlib import pyplot as plt
import numpy as np


# Creating dataset
cars = ['AUDI', 'BMW', 'FORD',
        'TESLA', 'JAGUAR', 'MERCEDES']

data = [23, 17, 35, 29, 12, 41]

# Creating plot
fig = plt.figure(figsize =(10, 7))
plt.pie(data, labels = cars)

# show plot
plt.show()
```



THANK YOU