**Experiment 1- Program to demonstrate area plot and create them using Matplotlib , Histogram with Matplotlib**

**How to Create a Simple Plot with the Plot() Function**

The***matplotlib.pyplot.plot()*** function provides a unified interface for creating different types of plots.

The simplest example uses the ***plot()*** function to plot values as***x,y*** coordinates in a data plot. In this case, ***plot()***takes 2 parameters for specifying plot coordinates:

* Parameter for an array of ***X axis*** coordinates.
* Parameter for an array of ***Y axis*** coordinates.

A line ranging from x=2, y=4 through x=8, y=9 is plotted by creating 2 arrays of ***(2,8)*** and ***(4,9)***:

Program-

import matplotlib.pyplot as plt

import numpy as np

*# X axis parameter:*

xaxis = np.array([2, 8])

*# Y axis parameter:*

yaxis = np.array([4, 9])

plt.plot(xaxis, yaxis)

plt.show()

**How to Customize Plot Appearance with Marker & Linestyle**

***marker*** and ***linestyle*** are matplotlib keywords that can be used to customize the appearance of data in a plot without modifying data values.

* ***marker*** is an argument used to label each data value in a plot with a ‘**marker**‘.
* ***linestyle*** is an argument used to customize the appearance of lines between data values, or else remove them altogether.

In this example, each data value is labeled with the letter ***“o”*,** and given a dashed linestyle***“–”***:

import matplotlib.pyplot as plt

import numpy as np

xaxis = np.array([2, 12, 3, 9])

*# Mark each data value and customize the linestyle:*

plt.plot(xcoords, marker = “o”, linestyle = “--”)

plt.show(**)**

A partial list of string characters that are acceptable options for ***marker*** and***linestyle***:

“-” solid line style

“--” dashed line style

“ “ no line

“o” letter marker

**Matplotlib Scatter Plot Example**

**Matplotlib also supports more advanced plots, such as scatter plots.** In this case, the ***scatter()*** function is used to display data values as a collection of ***x,y*** coordinates represented by standalone dots.

In this example, 2 arrays of the same length (one array for X axis values and another array for Y axis values) are plotted. Each value is represented by a dot:

import matplotlib.pyplot as plt

*# X axis values:*

x = [2,3,7,29,8,5,13,11,22,33]

*# Y axis values:*

y = [4,7,55,43,2,4,11,22,33,44]

*# Create scatter plot:*

plt.scatter(x, y)

plt.show()

**Matplotlib Example: Multiple Data Sets in One Plot**

**Matplotlib is highly flexible, and can accommodate multiple datasets in a single plot**. In this example, we’ll plot two separate data sets,***xdata1*** and ***xdata2****:*

import matplotlib.pyplot as plt

import numpy as np

*# Create random seed:*

np.random.seed(5484849901)

*# Create random data:*

xdata = np.random.random([2, 8])

*# Create two datasets from the random floats:*

xdata1 = xdata[0, :]

xdata2 = xdata[1, :]

*# Sort the data in both datasets:*

xdata1.sort()

xdata2.sort()

*# Create y data points:*

ydata1 = xdata1 \*\* 2

ydata2 = 1 - xdata2 \*\* 4

*# Plot the data:*

plt.plot(xdata1, ydata1)

plt.plot(xdata2, ydata2)

*# Set x,y lower, upper limits:*

plt.xlim([0, 1])

plt.ylim([0, 1])

plt.title(“Multiple Datasets in One Plot")

plt.show()

**Matplotlib Example: Subplots**

**You can also use matplotlib to create complex figures that contain more than one plot**. In this example, multiple axes are enclosed in one figure and displayed in subplots:

import matplotlib.pyplot as plt

import numpy as np

*# Create a Figure with 2 rows and 2 columns of subplots:*

fig, ax = plt.subplots(2, 2)

x = np.linspace(0, 5, 100)

*# Index 4 Axes arrays in 4 subplots within 1 Figure:*

ax[0, 0].plot(x, np.sin(x), 'g') #row=0, column=0

ax[1, 0].plot(range(100), 'b') #row=1, column=0

ax[0, 1].plot(x, np.cos(x), 'r') #row=0, column=1

ax[1, 1].plot(x, np.tan(x), 'k') #row=1, column=1

plt.show()

#### Matplotlib Example: Histogram Plot

**A histogram is used to display frequency distributions in a bar graph.**

In this example, we’ll combine matplotlib’s histogram and subplot capabilities by creating a plot containing five bar graphs. The areas in the bar graph will be proportional to the frequency of a random variable, and the widths of each bar graph will be equal to the class interval:

import matplotlib.plot as plt

import matplotlib.ticker as maticker

import numpy as np

*# Create random variable:*

data = np.random.normal(0, 3, 800)

*# Create a Figure and multiple subplots containing Axes:*

fig, ax = plt.subplots()

weights = np.ones\_like(data) / len(data)

*# Create Histogram Axe:*

ax.hist(data, bins=5, weights=weights)

ax.yaxis.set\_major\_formatter(maticker.PercentFormatter(xmax=1.0, decimals=1))

plt.title(“Histogram Plot”)

plt.show()

**Experiment 2- Pandas Data Manipulation**

1. **Pandas**
2. **Series**
3. **Timedelta**

## How to import pandas and check the version?

import numpy as np # optional

import pandas as pd

print(pd.\_\_version\_\_)

print(pd.show\_versions(as\_json=True))

0.20.3

{'system': {'commit': None}, 'dependencies': {'pandas': '0.20.3', 'pytest': '3.2.1', 'pip': '9.0.1', 'setuptools': '36.5.0.post20170921', 'Cython': '0.26.1', 'numpy': '1.13.3', 'scipy': '0.19.1', 'xarray': None, 'IPython': '6.1.0', 'sphinx': '1.6.3', 'patsy': '0.4.1', 'dateutil': '2.6.1', 'pytz': '2017.2', 'blosc': None, 'bottleneck': '1.2.1', 'tables': '3.4.2', 'numexpr': '2.6.2', 'feather': None, 'matplotlib': '2.1.0', 'openpyxl': '2.4.8', 'xlrd': '1.1.0', 'xlwt': '1.2.0', 'xlsxwriter': '1.0.2', 'lxml': '4.1.0', 'bs4': '4.6.0', 'html5lib': '0.999999999', 'sqlalchemy': '1.1.13', 'pymysql': None, 'psycopg2': None, 'jinja2': '2.9.6', 's3fs': None, 'pandas\_gbq': None, 'pandas\_datareader': None}}

None

## **How to create a series from a list, numpy array and dict?**

Create a pandas series from each of the items below: a list, numpy and a dictionary

# Inputs

import numpy as np

mylist = list('abcedfghijklmnopqrstuvwxyz')

myarr = np.arange(26)

mydict = dict(zip(mylist, myarr))

# Solution

ser1 = pd.Series(mylist)

ser2 = pd.Series(myarr)

ser3 = pd.Series(mydict)

print(ser3.head())

a 0

b 1

c 2

d 4

e 3

dtype: int64

## **How to convert the index of a series into a column of a dataframe?**

Convert the series ser into a dataframe with its index as another column on the dataframe.

# Input

mylist = list('abcedfghijklmnopqrstuvwxyz')

myarr = np.arange(26)

mydict = dict(zip(mylist, myarr))

ser = pd.Series(mydict)

# Solution

df = ser.to\_frame().reset\_index()

print(df.head())

index 0

0 a 0

1 b 1

2 c 2

3 d 4

4 e 3

## **How to combine many series to form a dataframe?**

# Input

import numpy as np

ser1 = pd.Series(list('abcedfghijklmnopqrstuvwxyz'))

ser2 = pd.Series(np.arange(26))

# Solution 1

df = pd.concat([ser1, ser2], axis=1)

# Solution 2

df = pd.DataFrame({'col1': ser1, 'col2': ser2})

print(df.head())

col1 col2

0 a 0

1 b 1

2 c 2

3 e 3

4 d 4

**Experiment 3- Program to draw scatter plots, bubble plots, racing bar charts with Matplotlib.**

## **How to Create Scatter Plots using Matplotlib**

Scatter plots are used to depict a relationship between two variables.

For example, let’s say that you want to depict the relationship between:

* The Marks; and
* The percentile score of your class

### **Create Scatter Plot using Lists**

You can create simple lists, which will contain the values for the Unemployment Rate and the Stock Index Price:

Marks = [60, 50, 80, 75, 40, 56]

Percentile\_score = [80, 90, 50, 60, 75, 82, 50, 40]

To create the scatter plot based on the above data, you can apply the generic syntax that was introduced at the beginning of this guide. Your full Python code would look like this:

import matplotlib.pyplot as plt

Marks = [60, 50, 80, 75, 40, 56]

Percentile\_score = [80, 90, 50, 60, 75, 82, 50, 40]

plt.scatter(Marks, Percentile\_score, color='green')

plt.title('Marks Vs Percentile\_score', fontsize=14)

plt.xlabel('Marks', fontsize=14)

plt.ylabel(' Percentile\_score, fontsize=14)

plt.grid(True)

plt.show()

### **Create Scatter Plot using Pandas DataFrame**

Another way in which you can capture the data in Python is by using pandas DataFrame.

You’ll need to [install](https://datatofish.com/install-package-python-using-pip/) and then import the pandas module, in addition to the matplotlib module.

Using our example, you can then [create the pandas DataFrame](https://datatofish.com/create-pandas-dataframe/) as follows:

from pandas import DataFrame

Data = {'Unemployment\_Rate': [6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2],

'Stock\_Index\_Price': [1500,1520,1525,1523,1515,1540,1545,1560,1555,1565]

}

df = DataFrame(Data,columns=['Unemployment\_Rate','Stock\_Index\_Price'])

And here is the full Python code to display the Scatter plot using the DataFrame:

from pandas import DataFrame

import matplotlib.pyplot as plt

Data = {'Unemployment\_Rate': [6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2],

'Stock\_Index\_Price': [1500,1520,1525,1523,1515,1540,1545,1560,1555,1565]

}

df = DataFrame(Data,columns=['Unemployment\_Rate','Stock\_Index\_Price'])

plt.scatter(df['Unemployment\_Rate'], df['Stock\_Index\_Price'], color='green')

plt.title('Unemployment Rate Vs Stock Index Price', fontsize=14)

plt.xlabel('Unemployment Rate', fontsize=14)

plt.ylabel('Stock Index Price', fontsize=14)

plt.grid(True)

plt.show()

**Create Line Chart using Lists**

You may store the Years and the associated Unemployment Rates as lists:

Year = [1920,1930,1940,1950,1960,1970,1980,1990,2000,2010]

Unemployment\_Rate = [9.8,12,8,7.2,6.9,7,6.5,6.2,5.5,6.3]

Using the Line chart syntax from the beginning of this guide, your full Python code would be:

import matplotlib.pyplot as plt

Year = [1920,1930,1940,1950,1960,1970,1980,1990,2000,2010]

Unemployment\_Rate = [9.8,12,8,7.2,6.9,7,6.5,6.2,5.5,6.3]

plt.plot(Year, Unemployment\_Rate, color='red', marker='o')

plt.title('Unemployment Rate Vs Year', fontsize=14)

plt.xlabel('Year', fontsize=14)

plt.ylabel('Unemployment Rate', fontsize=14)

plt.grid(True)

plt.show()

**Create a Bar chart using Lists**

First, create the lists as follows:

Country = ['USA','Canada','Germany','UK','France']

GDP\_Per\_Capita = [45000,42000,52000,49000,47000]

Notice that the Country column contains text/strings (wrapped around quotations for each value), while the GDP\_Per\_Capita column contains numerical values without the quotations.

Since our dataset contains both text and numerical values, you’ll need to add the following syntax:

xAxis = [i + 0.5 for i, \_ in enumerate(Country)]

Without the above portion, you’ll face the following error in Python:

unsupported operand type(s) for -: ‘str’ and ‘float’

You’ll also need to incorporate the following section when depicting the bar chart:

plt.xticks([i + 0.5 for i, \_ in enumerate(Country)], Country)

When you put all the components together, your full code to create a Bar chart would look like this:

import matplotlib.pyplot as plt

Country = ['USA','Canada','Germany','UK','France']

GDP\_Per\_Capita = [45000,42000,52000,49000,47000]

xAxis = [i + 0.5 for i, \_ in enumerate(Country)]

plt.bar(xAxis, GDP\_Per\_Capita, color='teal')

plt.title('Country Vs GDP Per Capita', fontsize=14)

plt.xlabel('Country', fontsize=14)

plt.ylabel('GDP Per Capita', fontsize=14)

plt.xticks([i + 0.5 for i, \_ in enumerate(Country)], Country)

plt.show()

**Create a Bar chart using pandas DataFrame**

Using pandas DataFrame:

from pandas import DataFrame

Data = {'Country': ['USA','Canada','Germany','UK','France'],

'GDP\_Per\_Capita': [45000,42000,52000,49000,47000]

}

df = DataFrame(Data,columns=['Country','GDP\_Per\_Capita'])

And here is the full Python code to create the Bar Chart using the DataFrame:

from pandas import DataFrame

import matplotlib.pyplot as plt

Data = {'Country': ['USA','Canada','Germany','UK','France'],

'GDP\_Per\_Capita': [45000,42000,52000,49000,47000]

}

df = DataFrame(Data,columns=['Country','GDP\_Per\_Capita'])

xAxis = [i + 0.5 for i, \_ in enumerate(df['Country'])]

plt.bar(xAxis, df['GDP\_Per\_Capita'].astype(float), color='teal')

plt.title('Country Vs GDP Per Capita', fontsize=14)

plt.xlabel('Country', fontsize=14)

plt.ylabel('GDP Per Capita', fontsize=14)

plt.xticks([i + 0.5 for i, \_ in enumerate(df['Country'])], df['Country'])

plt.show()

**Experiment 4- Program related to any dataset using Pandas Dataframe methods-**

Pandas DataFrame is a 2-dimensional labeled data structure like any table with rows and columns. The size and values of the dataframe are mutable,i.e., can be modified. It is the most commonly used pandas object. Pandas DataFrame can be created in multiple ways. Let’s discuss different ways to create a DataFrame one by one.

DataFrame() function is used to create a dataframe in Pandas. The syntax of creating dataframe is:

pandas.DataFrame(data, index, columns)

where,

data: It is a dataset from which dataframe is to be created. It can be list, dictionary, scalar value, series, ndarrays, etc.

index: It is optional, by default the index of the dataframe starts from 0 and ends at the last data value(n-1). It defines the row label explicitly.

columns: This parameter is used to provide column names in the dataframe. If the column name is not defined by default, it will take a value from 0 to n-1.

**Method #1 : Creating an Empty DataFrame**

# Importing Pandas to create DataFrame

import pandas as pd

# Creating Empty DataFrame and Storing it in variable df

df = pd.DataFrame()

# Printing Empty DataFrame

print(df)

**Method #2: Creating Dataframe from Lists**

# Import pandas library

import pandas as pd

# initialize list elements

data = [10,20,30,40,50,60]

# Create the pandas DataFrame with column name is provided explicitly

df = pd.DataFrame(data, columns=['Numbers'])

# print dataframe.

df

**Using Pandas dataframe.count()**

Pandas dataframe.count() is used to count the no. of non-NA/null observations across the given axis. It works with non-floating type data as well.

Syntax: DataFrame.count(axis=0, level=None, numeric\_only=False)

Parameters:

axis : 0 or ‘index’ for row-wise, 1 or ‘columns’ for column-wise

level : If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a DataFrame

numeric\_only : Include only float, int, boolean data

Example #: Use count() function to find the number of non-NA/null value across the row axis.

# importing pandas as pd

import pandas as pd

# Creating a dataframe using dictionary

df = pd.DataFrame({"A":[-5, 8, 12, None, 5, 3],

"B":[-1, None, 6, 4, None, 3],

"C:["sam", "haris", "alex", np.nan, "peter", "nathan"]})

# Printing the dataframe

df

Example #: Use count() function to find the number of non-NA/null value across the column.

# importing pandas as pd

import pandas as pd

# Creating a dataframe using dictionary

df = pd.DataFrame({"A":[-5, 8, 12, None, 5, 3],

"B":[-1, None, 6, 4, None, 3],

"C:["sam", "haris", "alex", np.nan, "peter", "nathan"]})

# Find count of non-NA across the columns

df.count(axis = 1)

**Using Pandas Dataframe.describe() method**

Pandas describe() is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values. When this method is applied to a series of string, it returns a different output which is shown in the examples below.

Syntax: DataFrame.describe(percentiles=None, include=None, exclude=None)

Parameters:

percentile: list like data type of numbers between 0-1 to return the respective percentile

include: List of data types to be included while describing dataframe. Default is None

exclude: List of data types to be Excluded while describing dataframe. Default is None

Example #: Describing data frame with both object and numeric data type

In this example, the data frame is described and [‘object’] is passed to include parameter to see description of object series. [.20, .40, .60, .80] is passed to percentile parameter to view the respective percentile of Numeric series.

# importing pandas module

import pandas as pd

# importing regex module

import re

# making data frame

Data = pd.read\_csv("https://media.geeksforgeeks.org/wp-content/uploads/nba.csv")

# removing null values to avoid errors

data.dropna(inplace = True)

# percentile list

perc =[.20, .40, .60, .80]

# list of dtypes to include

include =['object', 'float', 'int']

# calling describe method

desc = data.describe(percentiles = perc, include = include)

# display

desc

**Using Pandas Dataframe.duplicated()**

An important part of Data analysis is analyzing Duplicate Values and removing them. Pandas duplicated() method helps in analyzing duplicate values only. It returns a boolean series which is True only for Unique elements.

Syntax:

DataFrame.duplicated(subset=None, keep='first')

Parameters:

subset: Takes a column or list of column label. It’s default value is none. After passing columns, it will consider them only for duplicates.

keep: Controls how to consider duplicate value. It has only three distinct value and default is ‘first’.

–> If ‘first’, it considers first value as unique and rest of the same values as duplicate.

–> If ‘last’, it considers last value as unique and rest of the same values as duplicate.

–> If False, it consider all of the same values as duplicates.

Example #: Returning a boolean series

In the following example, a boolean series is returned on the basis of duplicate values in the First Name column.

# importing pandas package

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("employees.csv")

# sorting by first name

data.sort\_values("First Name", inplace = True)

# making a bool series

bool\_series = data["First Name"].duplicated()

# displaying data

data.head()

# display data

data[bool\_series]

Example #: Removing duplicates

In this example, the keep parameter is set to False, so that only Unique values are taken and the duplicate values are removed from data.

# importing pandas package

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("employees.csv")

# sorting by first name

data.sort\_values("First Name", inplace = True)

# making a bool series

bool\_series = data["First Name"].duplicated(keep = False)

# bool series

bool\_series

# passing NOT of bool series to see unique values only

data = data[~bool\_series]

# displaying data

data.info()

data

**Using Pandas DataFrame.empty**

Pandas DataFrame.empty attribute checks if the dataframe is empty or not. It return True if the dataframe is empty else it return False.

Syntax: DataFrame.empty

Parameter : None

Example #: Use DataFrame.empty attribute to check if the given dataframe is empty or not.

# importing pandas as pd

import pandas as pd

# Creating the DataFrame

df = pd.DataFrame({'Weight':[45, 88, 56, 15, 71],

'Name':['Sam', 'Andrea', 'Alex', 'Robin', 'Kia'],

'Age':[14, 25, 55, 8, 21]})

# Create the index

index\_ = ['Row\_1', 'Row\_2', 'Row\_3', 'Row\_4', 'Row\_5']

# Set the index

df.index = index\_

# Print the DataFrame

print(df)

**Using Pandas dataframe.equals()**

Pandas dataframe.equals() function is used to determine if two dataframe object in consideration are equal or not. Unlike dataframe.eq() method, the result of the operation is a scalar boolean value indicating if the dataframe objects are equal or not.

Syntax: DataFrame.equals(other)

Parameters:

other : DataFrame

Example #: Use equals() function to find the result of comparison between two different dataframe objects.

# importing pandas as pd

import pandas as pd

# Creating the first dataframe

df1 = pd.DataFrame({"A":[1,5,7,8],

"B":[5,8,4,3],

"C":[10,4,9,3]})

# Creating the second dataframe

df2 = pd.DataFrame({"A":[5,3,6,4],

"B":[11,2,4,3],

"C":[4,3,8,5]})

# Print the first dataframe

df1

# Print the second dataframe

df2

Example #: Use equals() function to test for equality between two data frame object with NaN values.

# importing pandas as pd

import pandas as pd

# Creating the first dataframe

df1 = pd.DataFrame({"A":[1,2,3],

"B":[4,5,None],

"C":[7,8,9]})

# Creating the second dataframe

df2 = pd.DataFrame({"A":[1,2,3],

"B":[4,5,None],

"C":[7,8,9]})

# Print the first dataframe

df1

# Print the second dataframe

df2

**Experiment 5 - Generate Waffle chart using pyWaffle in Python and Generating Word Cloud in Python**

A Waffle Chart is a gripping visualization technique that is normally created to display progress towards goals. Where each cell in the Waffle Chart constitutes of 10 X 10 cell grid in which each cell represents one percentage point summing up to total 100%. It is commonly an effective option when you are trying to add interesting visualization features to a visual. Waffle Charts are widely used as an Excel dashboard.

For generating Waffle Chart in Python, modules needed are – matplotlib, pandas and pyWaffle.

To install these packages, run the following commands :

pip install matplotlib

pip install pandas

pip install pywaffle

Program-

# python program to generate Waffle Chart

# importing all necessary requirements

import pandas as pd

import matplotlib.pyplot as plt

from pywaffle import Waffle

# creation of a dataframe

data ={'phone': ['Xiaomi', 'Samsung',

'Apple', 'Nokia', 'Realme'],

'stock': [44, 12, 8, 5, 3]

}

df = pd.DataFrame(data)

# To plot the waffle Chart

fig = plt.figure(

FigureClass = Waffle,

rows = 5,

values = df.stock,

labels = list(df.phone)

)

**Generating Word Cloud in Python**

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

For generating word cloud in Python, modules needed are – matplotlib, pandas and wordcloud. To install these packages, run the following commands :

pip install matplotlib

pip install pandas

pip install wordcloud

The dataset used for generating word cloud is collected from UCI Machine Learning Repository. It consists of YouTube comments on videos of popular artists.

**Dataset Link :** [**https://archive.ics.uci.edu/ml/machine-learning-databases/00380/**](https://archive.ics.uci.edu/ml/machine-learning-databases/00380/)

# Python program to generate WordCloud

# importing all necessary modules

from wordcloud import WordCloud, STOPWORDS

import matplotlib.pyplot as plt

import pandas as pd

# Reads 'Youtube04-Eminem.csv' file

df = pd.read\_csv(r"Youtube04-Eminem.csv", encoding ="latin-1")

comment\_words = ''

stopwords = set(STOPWORDS)

# iterate through the csv file

for val in df.CONTENT:

# typecaste each val to string

val = str(val)

# split the value

tokens = val.split()

# Converts each token into lowercase

for i in range(len(tokens)):

tokens[i] = tokens[i].lower()

comment\_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,

background\_color ='white',

stopwords = stopwords,

min\_font\_size = 10).generate(comment\_words)

# plot the WordCloud image

plt.figure(figsize = (8, 8), facecolor = None)

plt.imshow(wordcloud)

plt.axis("off")

plt.tight\_layout(pad = 0)

plt.show()

**Experiment 6- Seaborn in Python**

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas.

Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs, so that we can switch between different visual representations for same variables for better understanding of dataset.

**Different categories of plot in Seaborn**

Plots are basically used for visualizing the relationship between variables. Those variables can be either be completely numerical or a category like a group, class or division. Seaborn divides plot into the below categories –

Relational plots: This plot is used to understand the relation between two variables.

Categorical plots: This plot deals with categorical variables and how they can be visualized.

Distribution plots: This plot is used for examining univariate and bivariate distributions

Regression plots: The regression plots in seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses.

Matrix plots: A matrix plot is an array of scatterplots.

Multi-plot grids: It is an useful approach is to draw multiple instances of the same plot on different subsets of the dataset.

**Installation**

For python environment :

pip install seaborn

**Dist plot : Seaborn dist plot is used to plot a histogram, with some other variations like kdeplot and rugplot.**

# Importing libraries

import numpy as np

import seaborn as sns

# Selecting style as white,

# dark, whitegrid, darkgrid

# or ticks

sns.set(style="white")

# Generate a random univariate

# dataset

rs = np.random.RandomState(10)

d = rs.normal(size=100)

# Plot a simple histogram and kde

# with binsize determined automatically

sns.distplot(d, kde=True, color="m")

**Line plot : The line plot is one of the most basic plot in seaborn library. This plot is mainly used to visualize the data in form of some time series, i.e. in continuous manner.**

import seaborn as sns

sns.set(style="dark")

fmri = sns.load\_dataset("fmri")

# Plot the responses for different\

# events and regions

sns.lineplot(x="timepoint",

y="signal",

hue="region",

style="event",

data=fmri)

**Experiment 8- Plotting Google Map using folium package**

Folium is built on the data wrangling strengths of the Python ecosystem and the mapping strengths of the Leaflet.js (JavaScript) library. Simply, manipulate your data in Python, then visualize it on a leaflet map via Folium. Folium makes it easy to visualize data that’s been manipulated in Python, on an interactive Leaflet map. This library has a number of built-in tilesets from OpenStreetMap, Mapbox etc.

Command to install folium module :

pip install folium

**To create a Base Map.**

# import folium package

import folium

# Map method of folium return Map object

# Here we pass coordinates of Gfg

# and starting Zoom level = 12

my\_map1 = folium.Map(location = [28.5011226, 77.4099794],

zoom\_start = 12 )

# save method of Map object will create a map

my\_map1.save(" my\_map1.html " )

**Add a circular marker with popup text.**

# import folium package

import folium

my\_map3 = folium.Map(location = [28.5011226, 77.4099794],

zoom\_start = 15)

# Pass a string in popup parameter

folium.Marker([28.5011226, 77.4099794],

popup = ' Geeksforgeeks.org ').add\_to(my\_map3)

my\_map3.save(" my\_map3.html ")

**Add a simple\_marker for parachute style marker with pop-up text.**

# import folium package

import folium

my\_map3 = folium.Map(location = [28.5011226, 77.4099794],

zoom\_start = 15)

# Pass a string in popup parameter

folium.Marker([28.5011226, 77.4099794], popup = ' sswcoe.edu.in ').add\_to(my\_map3)

my\_map3.save(" my\_map3.html ")

**Add a line to the map**

# import folium package

import folium

my\_map4 = folium.Map(location = [28.5011226, 77.4099794],

zoom\_start = 12)

folium.Marker([28.704059, 77.102490], popup = 'Delhi').add\_to(my\_map4)

folium.Marker([28.5011226, 77.4099794], popup = 'sswcoe').add\_to(my\_map4)

# Add a line to the map by using line method .

# it connect both coordinates by the line

# line\_opacity implies intensity of the line

folium.PolyLine(locations = [(28.704059, 77.102490), (28.5011226, 77.4099794)], line\_opacity = 0.5).add\_to(my\_map4)

my\_map4.save("my\_map4.html")