Ex: 1
Date:

Working with numpy arrays

Aim:

To write a python program to create an array using numpy package.

Algorithm:

Step1: Start
Step2: Import numpy package.
Step3: Create a list and assign values to it.
Step4: Create array using numpy.
Step5: Display array values.
Step6: Stop

Program:

```
import numpy as np

list = [1,2,3,4]


sample = np.array(list)

print("created list %s"% list)

print("Numpy array in python %s"% sample)


print("Creating new list of array\n")

new = np.array([(1,2,3),(4,5,6)])

print('Squre Root of {}'.format(new))

print(np.sqrt(new))
```

Execution:

```
created list [1, 2, 3, 4]

Numpy array in python [1 2 3 4]

Creating new list of array
```

Squre Root of [[1 2 3]

[4 5 6]]

[[1. 1.41421356 1.73205081]

[2. 2.23606798 2.44948974]]

Result:

Ex: 2
Date:

Basic plots using Matplotlib

Aim:

To write a python program to make a basic plots by using matplotlib.

Algorithm:

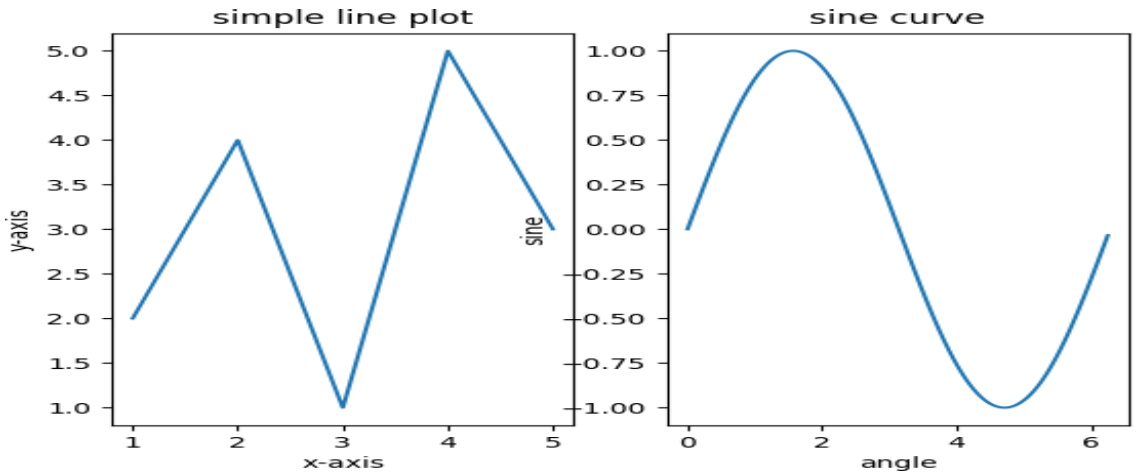
Step1: Start
Step2: Import matplotlib package.
Step3: Create a basic variables x and y.
Step4: Assign x and y with a common values.
Step5: Plot the points and display.
Step6: Stop

Program:

```
import matplotlib.pyplot as plt

import numpy as np

import math


# Sample data

x = [1, 2, 3, 4, 5]

y = [2, 4, 1, 5, 3]


# Create a figure1 and axis

plt.subplot(1,2,1)

plt.xlabel('x-axis')

plt.ylabel('y-axis')

plt.title('simple line plot')

plt.plot(x,y)
```

```
# Create a figure2 and axis
plt.subplot(1,2,2)
x=np.arange(0,(math.pi)*2,0.05)
y=np.sin(x)
plt.plot(x,y)
plt.xlabel('angle')
plt.ylabel('sine')
plt.title('sine curve')


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

Execution:



Result:

Ex: 3
Date:

Working with Pandas data frames

Aim:

To write a python program to create a data frames by using Pandas package.

Algorithm:

Step1: Start
Step2: Import pandas package.
Step3: Create a variable and assign some dictionary data values to it.
Step4: By using DataFrame() function create a data frame.
Step5: Display the data set values.
Step6: Stop

Program:

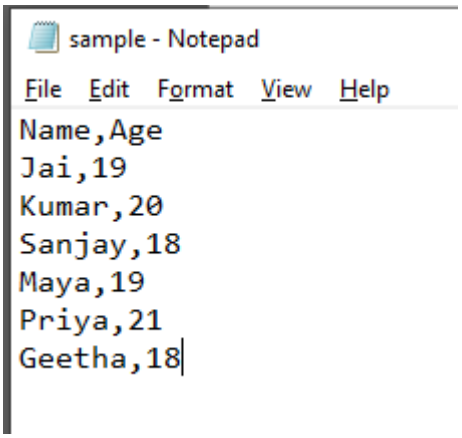
(i) Pandas data frames

```
import pandas as pd
data = {'Word':['happy','apple','blue','gloomy'],
    'Meaning':['happy','fruit','color','sad']}
df = pd.DataFrame(data)
print(df)
```

(ii) Reading CSV files with pandas

Creating a csv file by using notepad or any other text editor.

Save the file as any-name.csv.




```
import pandas as pd
df = pd.read_csv(' any-name.csv ')
print(df.head())
print(df.tail())
print(df.info())
```

Execution:

(i)

```
    Word Meaning
0   happy  happy
1   apple  fruit
2    blue    color
3  gloomy    sad
```

(ii)

```
    Name  Age
0    Jai  19
1   Kumar 20
2 Sanjay  18
3    Maya  19
4   Priya  21
```

```
    Name  Age
1  Kumar  20
2 Sanjay  18
3    Maya  19
4   Priya    21
5  Geetha  18
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 2 columns):
 #  Column  Non-Null Count  Dtype
```

Result:

Ex: 4
Date:

Frequency distributions, averages and variability

Aim:

To write a python program to find frequency distributions, averages and variability.

Algorithm:

Step1: Start
Step2: Import numpy package.
Step3: Import pandas package.
Step4: Assign data to created variables.
Step5: Solve the values and display.
Step6: Stop

Program:

```
import numpy as np

import pandas as pd

list = [2,4,4,4,5,5,7,9]

data={'Grade':['A','A','A','B','B','B','B','C','D','D'],

    'Age':[18,18,18,19,19,20,18,18,19,19],

    'Gender':['M','M','F','F','F','M','M','F','M','F']}

df = pd.DataFrame(data)

print(df)


#Find freqency of each letter grade

print('\nFind freqency of each letter grade')

print(pd.crosstab(index=df['Grade'],columns='count'))


#Fiding average, variance, standard deviation

print('\nFiding average, variance, standard deviation for')
```

```
print(list)
print('Average :',np.average(list))
print('Variance :',np.var(list))
print('Standard Deviation :',np.std(list))
```

Execution:

```
  Grade Age Gender
0     A   18      M
1     A   18      M
2     A   18      F
3     B   19      F
4     B   19      F
5     B   20      M
6     B   18      M
7     C   18      F
8     D   19      M
9     D   19      F
```

Find freqency of each letter grade

```
col_0 count
Grade
A         3
B         4
C         1
D         2
```

Fiding average, variance, standard deviation for

```
[2, 4, 4, 4, 5, 5, 7, 9]
Average : 5.0
```

Variance : 4.0

Standard Deviation : 2.0

Result:

Ex: 5
Date:

Normal Curves, Correlation and scatter plots, correlation coefficient

Aim:

To write a python program to calculate correlation, correlation coefficient and normal curves.

Algorithm:

Step1: Start
Step2: Import required librery
Step3: Make normal curves and calculate correlation.
Step4: Collect sample data to calculate correlation coefficient.
Step5: Assign the datas to x and y variable.
Step6: Plot the points.
Step7: Display the graphs (i),(ii)and(iii).
Step8: Stop

Program:

(i) Plotting normal distribution

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm


x=np.arange(-3,3,0.001)

plt.plot(x,norm.pdf(x,0,1))

plt.show()
```

(ii) Plot multiple normal distributions

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm


x=np.arange(-5,5,0.001)
```

```
plt.plot(x,norm.pdf(x,0,1),'--',label='μ:0, σ:1')
plt.plot(x,norm.pdf(x,0,1.5),'-.',label='μ:0, σ:1.5')
plt.plot(x,norm.pdf(x,0,2),'-',label='μ:0, σ:2')
plt.legend()
plt.show()
```

(iii) Plotting a scatter plot

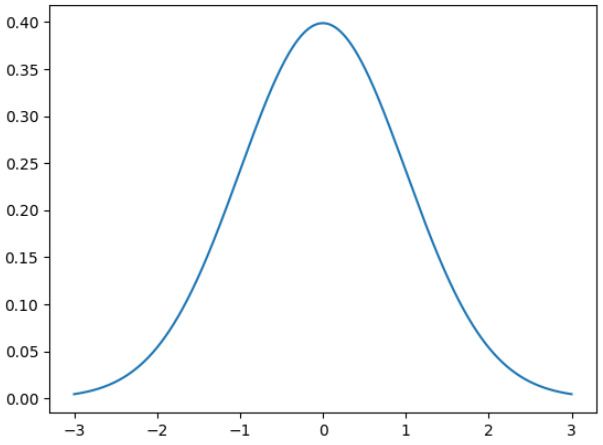
```
import numpy as np
import matplotlib.pyplot as plt

x,y,scale = np.random.randn(3,50)
fig,ax = plt.subplots()
ax.scatter(x=x,y=y,c=scale,s=np.abs(scale)*500)
ax.set(title='Scatter plot')
plt.show()
```

(vi) Calculation of the Pearson's correlation between two variables

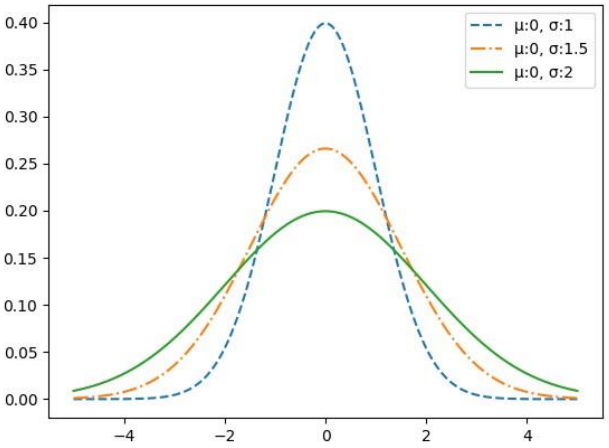
```
from numpy.random import randn
from numpy.random import seed
from scipy.stats import pearsonr
#seed random number generator
seed(1)
#data
data1 = 20*randn(1000) +100
data2 = data1 + (10 * randn(1000)+50)
#calculate pearson's correlation
corr,_=pearsonr(data1,data2)
print('Pearson correlation: %.3f' % corr)
```

Execution:

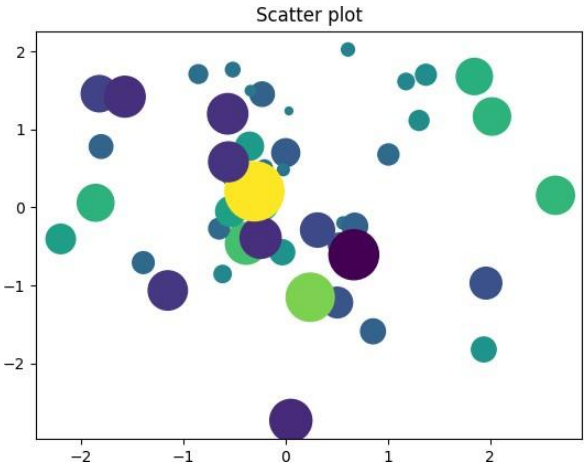
(i)



(ii)



(iii)



Result:

Ex: 6
Date:

Regression

Aim:

To write a python program calculate regression.

Algorithm:

Step1: Start
Step2: Import numpy and matplotlib.
Step3: Create a function coef(x,y) and calculate cross-deviation and deviation about x
Step4: And calculate regression coefficients.
Step5: Derive predicted response vector to
Step6: Create plot_regression_line(x,y,b) to plot values.
Step7: Plot the values and display.
Step8: Stop

Program:

```
import numpy as np

import matplotlib.pyplot as plt


def estimate_coef(x,y):

  #No.of points

  n=np.size(x)


  #mean of x and y vector

  m_x=np.mean(x)

  m_y=np.mean(y)


  #calculating cross-deviation and deviation about x

  SS_xy=np.sum(y*x) - n*m_y*m_x

  SS_xx=np.sum(x*x) - n*m_x*m_x


  #calculation regression coefficients

  b_1=SS_xy / SS_xx
```

```
    b_0=m_y - b_1 * m_x

    return (b_0, b_1)

def  plot_regression_line(x,y,b):
    #plotting actual points as scatter plots
    plt.scatter(x,y,color='m', marker='o',s=30)

    #predicted response vector
    y_pred=b[0] + b[1]*x

    #plotting the regression line
    plt.plot(x,y_pred,color='g')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

def main():
    #data
    x=np.array([0,1,2,3,4,5,6,7,8,9])
    y=np.array([1,3,2,5,7,8,8,9,10,12])

    #estimation coefficients
    b=estimate_coef(x,y)
    print("Estimated coefficients:\nb_0 = {} \nb_1 = {}".format(b[0],b[1]))

    #plotting regression line
    plot_regression_line(x,y,b)
```

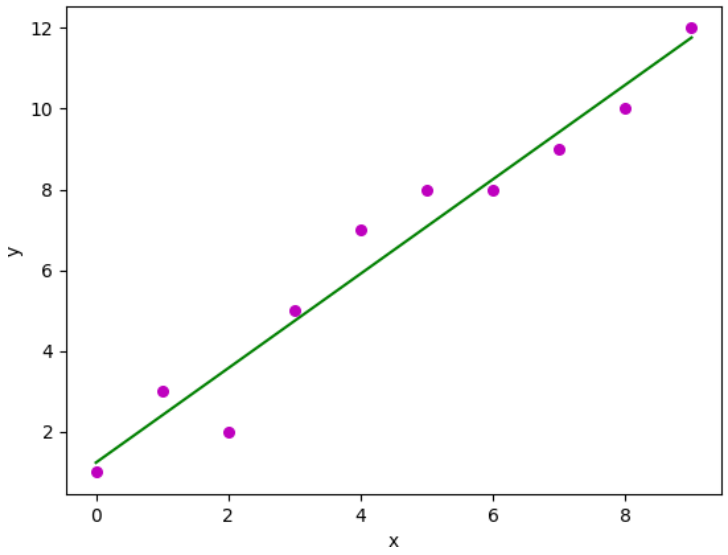
```
if __name___== '_main_':
    main()
```

Execution:

Estimated coefficients:

b_0 = 1.2363636363636363

b_1 = 1.1696969696969697



Result:

Ex: 7
Date:

Z-Test

Aim:

To write a python program to make a Z-test.

Algorithm:

Step1: Start
Step2: Import ztest from statsmodels.stats.weightstats.
Step3: Collecting IQ datas of 20 patients.
Step4: Assigning those values to data.
Step5: Display ztest(data,value=100).
Step6: Collects data from city A and B.
Step7: Display ztest(cityA,cityB,value=0)
Step8: Stop

Program:

```
from statsmodels.stats.weightstats import ztest as ztest

#enter IQ level for 20 patients

data=[88,92,94,94,96,97,97,97,99,99,105,109,109,110,112,112,113,114,115]

#perform one sample z-test

print('Z-Test I')

print(ztest(data, value=100))

cityA=[78,89,92,94,94,96,97,97,97,99,99,105,109,110,112,112,113,114,115]

cityB=[88,89,92,92,94,94,96,97,97,97,99,99,105,109,110,112,113,114,115]

print('\nZ-Test II')

print(ztest(cityA,cityB,value=0))
```

Execution:

```
Z-Test I

(1.378696666763784, 0.1679882976520375)

Z-Test II

(0.16977083200593462, 0.8651903665846945)
```

Result:

Ex: 8
Date:

T-Test

Aim:

To write a python program to make a T-test.

Algorithm:

Step1: Start
Step2: Import numpy and scipy.
Step3: Calculate standard deviation.
Step4: Assign standard deviation value to var_x.
Step5: Calculate variance to get std
Step6: Assign variance to var_y.
Step7: By using stats module calculated standard deviation, p and t.
Step8: Stop

Program:

```
import numpy as np

from scipy import stats


N = 10

#Gaussian distributed data with mean=2 and var=1

x=np.random.rand(N)+2

#Gaussian distributed data with mean=0 and var=1

y=np.random.randn(N)

#calculating standard deviation

#calculating variance to get std

var_x = x.var(ddof=1)

var_y = y.var(ddof=1)

#standard deviation

SD = np.sqrt((var_x + var_y) / 2)

print('Standard Deviation =',SD)

#Calculating the T-Statistics
```

```
tval = (x.mean() - y.mean()) / (SD * np.sqrt(2/N))
#compaing with critical T-Value
#Degrees of freedom
dof=2*N-2
#p-value after compaison with the T-Statistics
pval = 1-stats.t.cdf(tval,df=dof)
print('t = '+str(tval))
print('p = '+str(2*pval))
#Cross checking using the internal function from scipy package
tval2,pval2 = stats.ttest_ind(x,y)
print('t = '+str(tval2))
print('p = '+str(pval2))
```

Execution:

```
Standard Deviation = 0.7194173256540722
t = 7.307006005934893
p = 8.687336403578882e-07
t = 7.307006005934891
p = 8.68733640421676e-07
```

Result:

Ex: 9
Date:

ANOVA

Aim:

To write a python program to make a Anova analysis.

Algorithm:

Step1: Start
Step2: Import required library.
Step3: Import seaborn to customize style.
Step4: Import csv dataset named Diet_Dataset.csv
Step5: Display CSV file data.
Step6: Display header datas in CSV file.
Step7: Plot x and y which are mata-data age and pdf.
Step8: Stop

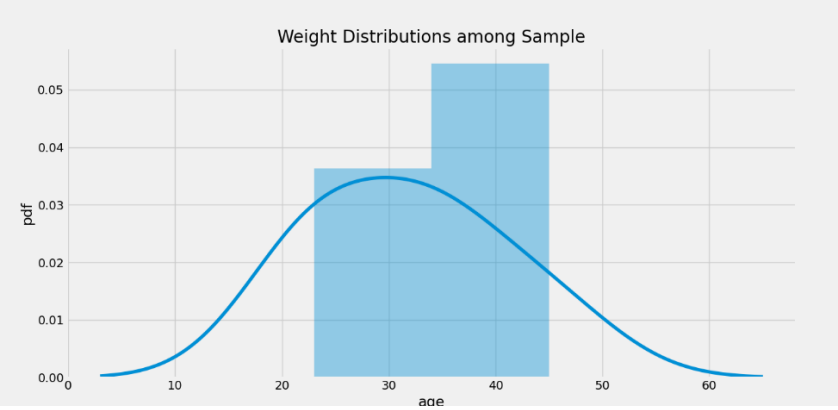
Program:

```
import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm

from statsmodels.formula.api import ols

import seaborn as sns

import numpy as np

import pandas.tseries

plt.style.use('fivethirtyeight')

mydata=pd.read_csv('Diet_Dataset.csv')

print(mydata.head())

print('\nThe total number of rows in the dataset:',mydata.size)

print('\n',mydata.gender.unique())

print(mydata[mydata.gender==' '])

f,ax=plt.subplots(figsize=(11,9))

plt.title('Weight Distributions among Sample')

plt.ylabel('pdf')
```

```
sns.distplot(mydata.age)
plt.show()
```

Execution:

```
   person gender age height
0     23        34    344
1     32        45    233
2      2     0  23    234
3      3     0  34    345
4     22     0  23    344
The total number of rows in the dataset: 20
[' ' '0']
   person gender  age  height
0     23        34    344
1     32        45    233
```



Result:

Ex: 10
Date:

Building and Validating linear models

Aim:

To write a python program to building and validating linear models.

Algorithm:

Step1: Start
Step2: From pandas import read_csv, autocorrelation_plot and DataFrame.
Step3: Import statsmodels.tsa.arima_model.
Step4: Import a dataset from csv.csv file.
Step5: Create a function parser() to calculate date-time.
Step6: Read csv.csv file and assign the data to series variable.
Step7: Display series.
Step8: Plot series.
Step9:Display graph.
Step10: Stop

Program:

```
from pandas import read_csv

from matplotlib import pyplot

from pandas.plotting import autocorrelation_plot

from pandas import DataFrame

from statsmodels.tsa.arima_model import ARIMA


#Importing Data

def parser(x):

   return datetime.strptine('198"+x, "W-%a')


series = read_csv("csv.csv")#, header=0, index_col=0, squeeze=True)

print(series.head())


#ploting in series

series.plot()
```

#autocorrelation

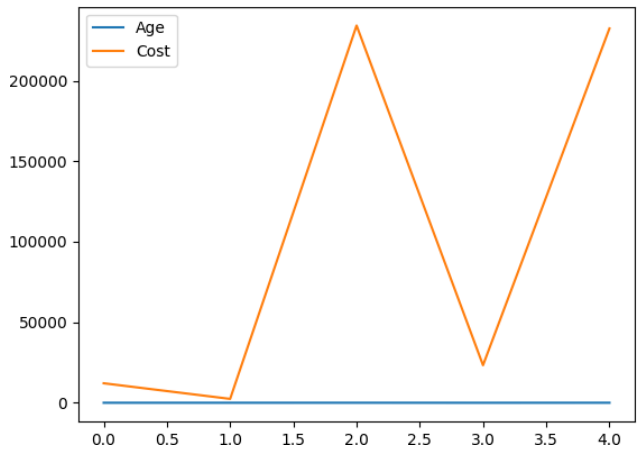
pyplot.figure()

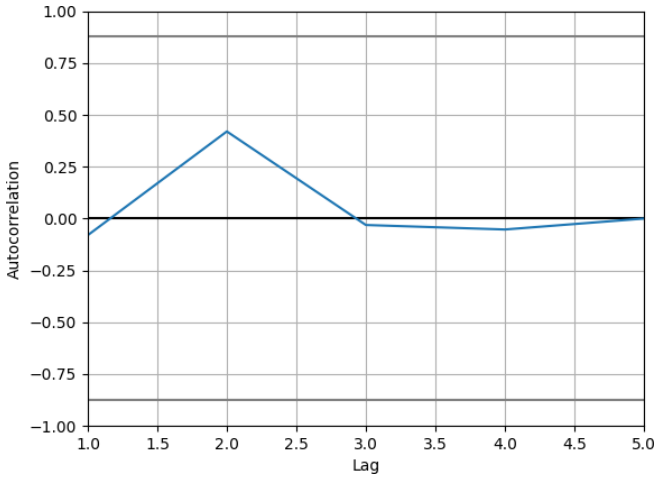
autocorrelation_plot(series)

pyplot.show()

Execution:

	Age	Cost
0	12	12121
1	12	2423
2	22	234234
3	3	23324
4	23	232422





Result: