EXPERIMENT -1

DATE:

BASIC PYTHON PROGRAM

AIM: To crate a python code on the following:

(1) Program to demonstrate different number data type in Python.
(2) Program to perform different arithmetic operation on numbers in Python.
(3) Program to create, concatenate and print a string and accessing sub-string.
(4) Program to verify whether the given number is odd number or even number.
(5) Program to print the day of the week.
(6) Program to print the multiplication table of a given integer..

SOURCE CODE:

1.
```
i=7
c=24+8j
f=7.01
s='HELLO EVERYONE!!'
# NOTE: boolean has truth values that are case sensitive Ex: True (T is caps!)
b= True
print("the value of c is:",i,'\nits type is:',type(i))
print("the value of c is:",f,'\nits type is:',type(f))
print("the value of c is:",c,'\nits type is:',type(c))
print("the value of c is:",s,'\nits type is:',type(s))
print("the value of c is:",b,'\nits type is:',type(b))
```

OUTPUT:

```
the value of c is: 7
its type is: <class 'int'>
the value of c is: 7.01
its type is: <class 'float'>
the value of c is: (24+8j)
its type is: <class 'complex'>
the value of c is: HELLO EVERYONE!!
its type is: <class 'str'>
the value of c is: True
its type is: <class 'bool'>
```

2.
```
a=10; b=3
print("addition of a:",a,"&b:",b,"is:",a+b)
print("substraction of a:",a,"&b:",b,"is:",a-b)
print("multiplication of a:",a,"&b:",b,"is:",a*b)
print("division of a:",a,"&b:",b,"is:",a/b)
print("floor divison of a:",a,"&b:",b,"is:",a//b)
print("moduli of a:",a,"&b:",b,"is:",a%b)
print("exponent of a:",a,"&b:",b,"is:",a**b)
```

OUTPUT:

```
addition of a: 10 &b: 3 is: 13
substraction of a: 10 &b: 3 is: 7
multiplication of a: 10 &b: 3 is: 30
division of a: 10 &b: 3 is: 3.3333333333333335
floor divison of a: 10 &b: 3 is: 3
moduli of a: 10 &b: 3 is: 1
exponent of a: 10 &b: 3 is: 1000
```

3.
```
pi=3.14
s= "Venkata"
v= "Subhramanyam"
print("the value of s is:",s)
print("the value of v is:",v)
string_add = s+v
print("after concatenating s and v the string is:",s+v)
text = 'The value of pi is ' + str(pi)
print("NOTE: variables after '+' operator must be converted to string before using them as
strings\n otherwise value will be considered as its class type")
print(text)
```

OUTPUT:

```
the value of s is: Venkata
the value of v is: Subhramanyam
after concatenating s and v the string is: VenkataSubhramanyam
NOTE: variables after '+' operator must be converted to string before using them as strings
 otherwise value will be considered as its class type
The value of pi is 3.14
```

4.
```
n = int(input('n =? '))
if n%2==0: print (n, "is an even number")
else : print (n, "is an odd number")
```

OUTPUT:

```
n =? 3
3 is an odd number
```

5.

```
n = int(input('n (1-7)=? '))
if n==1: print ("Monday")
elif n==2: print ("Tueday")
elif n==3: print ("Wednesday")
elif n==4: print ("Thursday")
elif n==5: print ("Friday")
elif n==6: print ("Saturday")
elif n==7: print ("Sunday")
else : print ("Please enter a number between 1 to 7")
```

OUTPUT:

```
n (1-7)=? 5
Friday
```

6.

```
n = int(input('n=? '))
i = 1
while (i <= 10) :
print (n ,'X', i, '=', n*i)
i = i + 1
print ('done')
```

OUTPUT:

```
n=? 5
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50
done
```

RESULT:

Basic program in python was successfully exceuted.

EXPERIMENT -2

DATE:

BASIC STATSTICAL FUNCTION

AIM:

>Program to first mean, median and mode.

>Standard deviation.

>Program data distribution.

>Program to normal distribution

SOURCE CODE:

1.
```
import numpy
speed = [99, 86, 88, 11, 86, 103, 87, 94, 78, 77, 85, 86]
x = numpy, mean(speed)
print(x)
y = numpy.median(speed)
print(y)
z = numpy.mode(speed)
print(z)
w = numpy range(speed)
print(w)
```

OUTPUT:

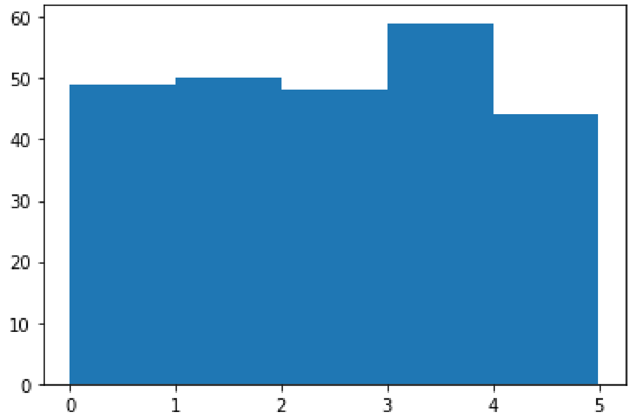
```
 89.76923076923077
87.0
ModeResult(mode=array([86]), count=array([3]
```

2.
```
import numpy
speed =[86,87,86,87,85,88]
x = numpy.std(speed)
print(x)
```

OUTPUT: 0.9035079029052513

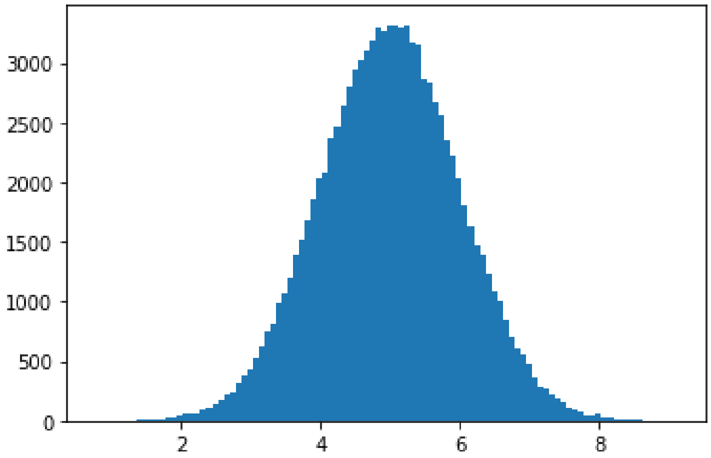
```
3.  from numpy import random
    x = random.choice([3,5,7,9] p=[2.1,0.3,0.6,0.6] size(3.5))
    print(x)
```

OUTPUT:



```
4.  from numpy import random
    import matpotlin.pyplot ass plt
    import seaborn as sns
    sns.displot(random.normal(size = 1000), hist=False)
    plt.show()
```

OUTPUT:



RESULT:

Program for finding mean, median, mode, data distribution, standard deviation and normal distribution was executed successfully

EXPERIMENT -3

DATE:

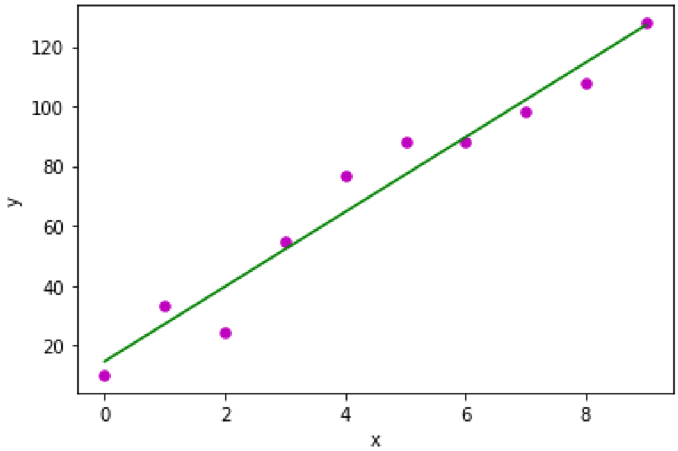
LINEAR REGRESSION

AIM: To crate a python code on Implementation of linear regression in python

```
#LINEAR REGRESSION 1
import numpy as np
import matplotlib.pyplot as plt
 def estimate_coef(x, y):
    # number of observations/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
     # calculating 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 the actual points as scatter plot
    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")
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()
# observations / data
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([10, 33, 24, 55, 77, 88, 88, 98, 108, 128])
  # estimating 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)
```

OUTPUT:

```
Estimated coefficients:
b_0 = 14.527272727272731
b_1 = 12.527272727272727
```



RESULT: Program for linear regression was successfully executed.

EXPERIMENT -4

DATE:

LINEAR REGRESSION USING DATASET

AIM: To crate a python code on Implementation of linear regression using dataset in python

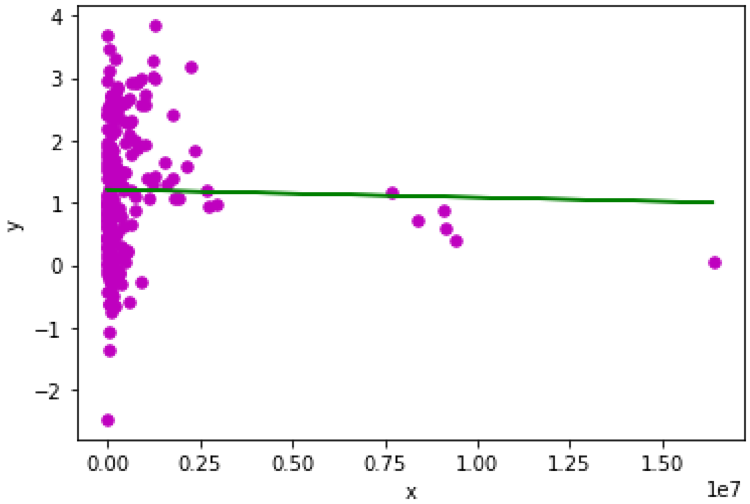
Sourcecode:

```
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
import io
import pandas as pd
data=files.upload()

data=pd.read_csv('/content/Country Population.csv')
data.head()

df = pd.read_csv("Country Population.csv")

def estimate_coef(x, y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    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):
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color = "g")
    plt.xlabel('x')
    plt.ylabel('y')


    plt.show()


x = df['Area_km']
y = df['Yearly_Change']

# estimating 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)
```

OUTPUT:

```
Estimated coefficients:
b_0 = 1.2083619983544096
b_1 = -1.249978632507629e-08
```



RESULT: Program for linear regression for the datasets from csv was successfully executed

EXPERIMENT -5

DATE:

MULTI LINEAR REGRESSION

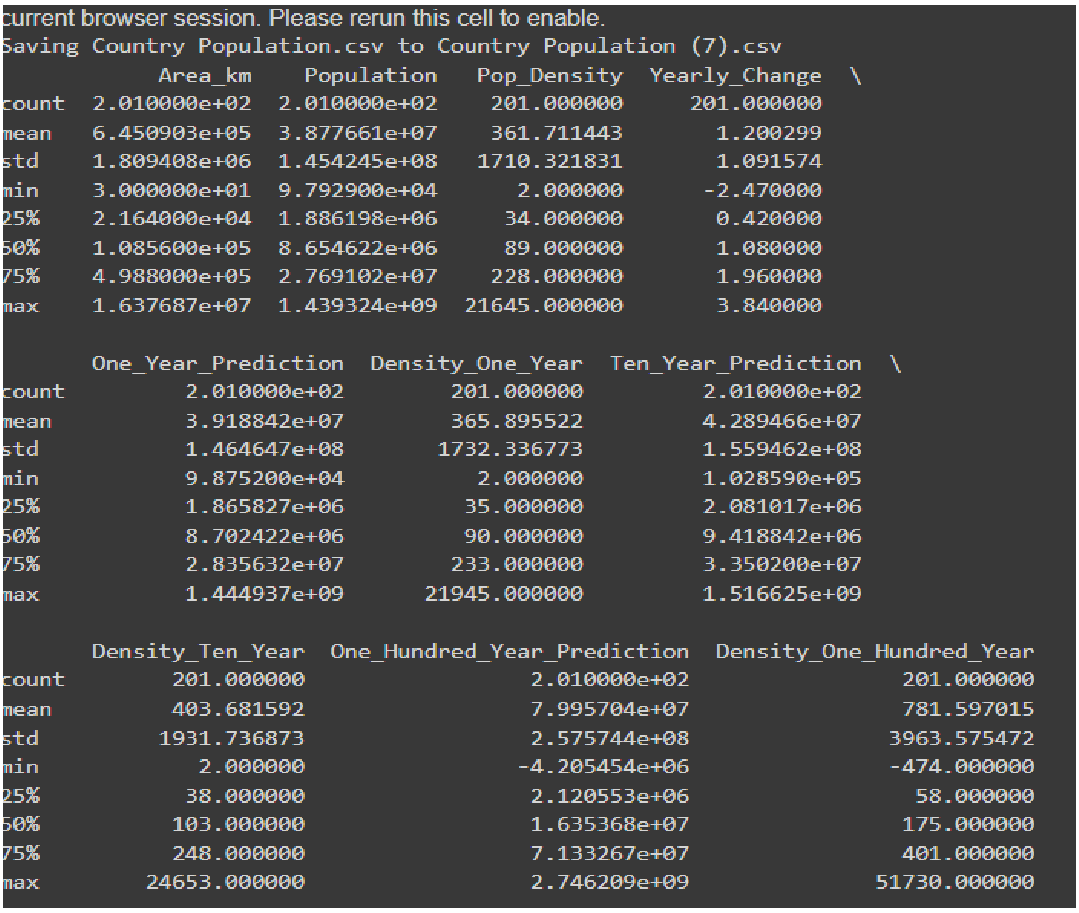
AIM: To crate a python code on Implementation of Multi linear regression in python

Source code:

```
from google.colab import files

uploaded = files.upload() #upload baseball.csv


import io

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from sklearn import linear_model

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.model_selection import train_test_split


df = pd.read_csv("Country Population.csv")

df.columns = ['Country','Area_km','Population','Pop_Density','Yearly_Change','One_Year_Prediction','Density_One_Year','Ten_Year_Prediction','Density_Ten_Year','One_Hundred_Year_Prediction','Density_One_Hundred_Year']

print(df.describe())

X = df.iloc[:,df.columns != 'Area_km']

Y = df.iloc[:, 0]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state= 0)


model = linear_model.LinearRegression()

model.fit(X_train, Y_train)

coeff_df = pd.DataFrame(model.coef_, X.columns, columns=['Coefficient'])

print(coeff_df)


y_pred = model.predict(X_test)
```

```
df = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred})
print(df.head(10))
df.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
rmsd = np.sqrt(mean_squared_error(Y_test, y_pred))
r2_value = r2_score(Y_test, y_pred)
print("Intercept: \n", model.intercept_)
print("Root Mean Square Error \n", rmsd)
print("R^2 Value: \n", r2_value)
```

OUTPUT:



```
current browser session. Please rerun this cell to enable.
Saving Country Population.csv to Country Population (7).csv
            Area_km    Population   Pop_Density  Yearly_Change  \
count  2.010000e+02  2.010000e+02    201.000000     201.000000
mean   6.450903e+05  3.877661e+07    361.711443       1.200299
std    1.809408e+06  1.454245e+08   1710.321831       1.091574
min    3.000000e+01  9.792900e+04      2.000000      -2.470000
25%    2.164000e+04  1.886198e+06     34.000000       0.420000
50%    1.085600e+05  8.654622e+06     89.000000       1.080000
75%    4.988000e+05  2.769102e+07    228.000000       1.960000
max    1.637687e+07  1.439324e+09  21645.000000       3.840000

       One_Year_Prediction  Density_One_Year  Ten_Year_Prediction  \
count         2.010000e+02        201.000000         2.010000e+02
mean          3.918842e+07        365.895522         4.289466e+07
std           1.464647e+08       1732.336773         1.559462e+08
min           9.875200e+04          2.000000         1.028590e+05
25%           1.865827e+06         35.000000         2.081017e+06
50%           8.702422e+06         90.000000         9.418842e+06
75%           2.835632e+07        233.000000         3.350200e+07
max           1.444937e+09      21945.000000         1.516625e+09

       Density_Ten_Year  One_Hundred_Year_Prediction  Density_One_Hundred_Year
count        201.000000                 2.010000e+02                201.000000
mean         403.681592                 7.995704e+07                781.597015
std         1931.736873                 2.575744e+08               3963.575472
min            2.000000                -4.205454e+06               -474.000000
25%           38.000000                 2.120553e+06                 58.000000
50%          103.000000                 1.635368e+07                175.000000
75%          248.000000                 7.133267e+07                401.000000
max        24653.000000                 2.746209e+09              51730.000000
```

RESULT: Program for multilinear regression was successfully executed.

\

EXPERIMENT -6

DATE:

K- MEANS CLUSTERING

AIM:

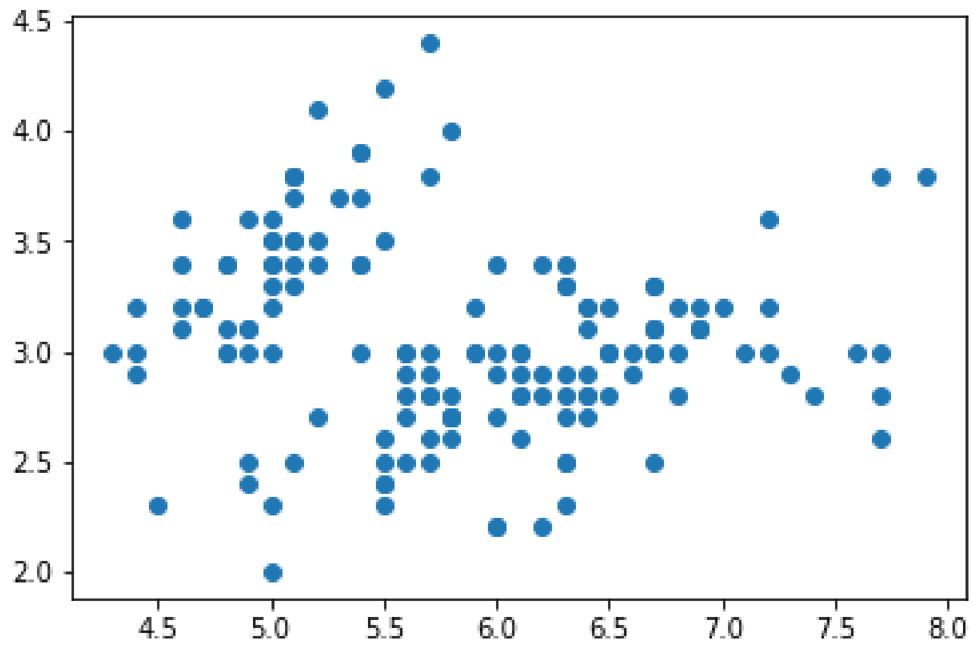
To crate a python code on Program for K- MEANS CLUSTERING

SOURCECODE:

```
import numpy as np

import matplotlib.pyplot as plt

from google.colab import files

import io

import pandas as pd

data=files.upload()

data=pd.read_csv('/content/iris.csv')

data.head()

mport pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

data = pd.read_csv('iris.csv')

print(data.head())

plt.scatter(data['Sepal.Length'],data['Sepal.Width'])

plt.show()
```

OUTPUT:

```
   Unnamed: 0  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
0           1           5.1          3.5           1.4          0.2  setosa
1           2           4.9          3.0           1.4          0.2  setosa
2           3           4.7          3.2           1.3          0.2  setosa
3           4           4.6          3.1           1.5          0.2  setosa
4           5           5.0          3.6           1.4          0.2  setosa
```



RESULT:

Program for K – means clustering was successfully executed.

EXPERIMENT -7

DATE:

COMBINING THE DATA SET

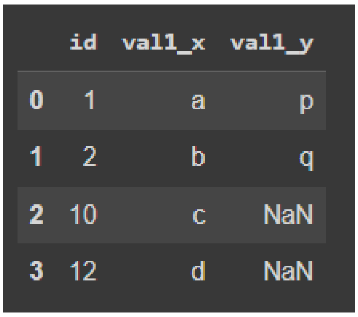
AIM: To crate a python code on Program for Combining the data set

Sourcecode:

inner join:

```
import pandas as pd
a = pd.DataFrame()
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}
a = pd.DataFrame(d)
b = pd.DataFrame()
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)
df = pd.merge(a, b, on='id', how='inner')
df
```

OUTPUT:

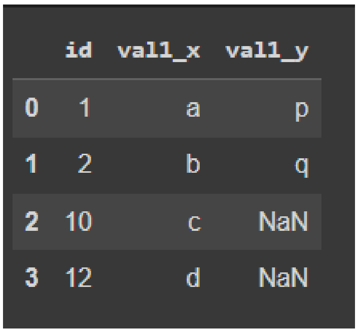


```
   id  val1_x  val1_y
0   1       a       p
1   2       b       q
2  10       c     NaN
3  12       d     NaN
```

LEFT JOIN:

```
import pandas as pd
a = pd.DataFrame()
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}
a = pd.DataFrame(d)
b = pd.DataFrame()
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)
df = pd.merge(a, b, on='id', how='left')
df
```

OUTPUT:

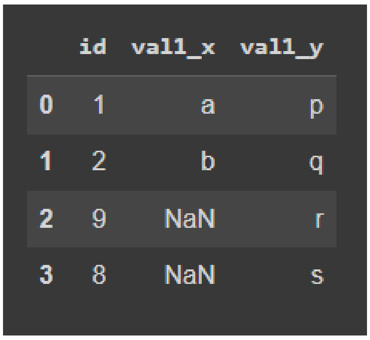


```
   id  val1_x  val1_y
0   1       a       p
1   2       b       q
2  10       c     NaN
3  12       d     NaN
```

RIGHT JOIN:

```
import pandas as pd
a = pd.DataFrame()
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}
a = pd.DataFrame(d)
b = pd.DataFrame()
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)
df = pd.merge(a, b, on='id', how='right')
df
```

OUTPUT:

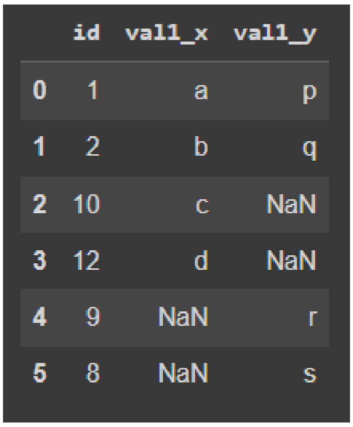


```
   id  val1_x  val1_y
0   1       a       p
1   2       b       q
2   9     NaN       r
3   8     NaN       s
```

LEFT JOIN:

```
import pandas as pd
a = pd.DataFrame()
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}
a = pd.DataFrame(d)
b = pd.DataFrame()
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)
df = pd.merge(a, b, on='id', how='outer')
df
```

OUTPUT:



	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN
4	9	NaN	r
5	8	NaN	s

RESULT:

Program for combining the data set was successfully executed.

EXPERIMENT -8

DATE:

FILE OPERATIONS

AIM: To crate a python code on Program for FILE OPERATIONS

Source code:

```
import pandas as pd
pd = open("demofile_1.txt", "rt")
print(pd.read())
#Return the 5 first characters of the file:
pd = open("demofile_1.txt", "r")
print(pd.read(5))
#Read one line of the file:
pd = open("demofile_1.txt", "r")
print(pd.readline())
By calling readline() two times, you can read the two first lines:
pd = open("demofile_1.txt", "r")
print(pd.readline())
print(pd.readline())
#Close the file when you are finish with it:
pd = open("demofile_1.txt", "r")
print(pd.readline())
pd.close()
#Remove the file "demofile.txt":
import os
os.remove("demofile_1.txt")
#Check if file exists, then delete it:
import os
if os.path.exists("demofile_1.txt"):
  os.remove("demofile_1.txt")
else:
  print("The file does not exist")
```

OUTPUT:

```
Hello world
Hello
Hello world
Hello world
Hello world
#removes the file
The file does not exist
```

RESULT: Hence, the file operations – opening file, read or write, closing the file, file exists implemented successfully

EXPERIMENT -9

DATE:

REGULAR EXPRESSIONS

AIM: To crate a python code on Program for REGULAR EXPRESSIONS

a) Write a Python program to check that a string contains only a certain set of characters (in this case a-z, A-Z and 0-9)

b) Write a Python program that matches a string that has an a followed by zero or one 'b'.

c) Write a Python program that matches a string that has an 'a' followed by anything, ending in 'b'.

d) Write a Python program that matches a word containing 'z'.

e) Write a Python program to separate and print the numbers of a given string.

Source code:

a)

```
import re
def is_allowed_specific_char(string):
    charRe = re.compile(r'[^a-zA-Z0-9]')
    string = charRe.search(string)
    return not bool(string)
print(is_allowed_specific_char("ABCDEFabcdef123450"))
print(is_allowed_specific_char("*&%@#!}{"))
```

output:

```
True
False
```

b)import re

```
def text_match(text):
        patterns = 'ab?'
        if re.search(patterns,  text):
                return 'Found a match!'
        else:
                return('Not matched!')
print(text_match("ab"))
print(text_match("abc"))
print(text_match("abbc"))
print(text_match("cb"))
```

output:

```
Found a match!
Found a match!
Found a match!
Not matched!
```

c)

```
import re
def text_match(text):
        patterns = 'a.*?b$'
        if re.search(patterns,  text):
                return 'Found a match!'
        else:
                return('Not matched!')
print(text_match("aabbbbd"))
print(text_match("aabAbbbc"))
print(text_match("accddbbjjjb"))
```

output:

```
Not matched!
Not matched!
Found a match!
```

d)import re

```
def text_match(text):
        patterns = '\w*z.\w*'
        if re.search(patterns,  text):
                return 'Found a match!'
        else:
                return('Not matched!')
print(text_match("The quick brown fox jumps over the lazy dog."))
print(text_match("Python Exercises."))
```

output:

```
Not matched!
Not matched!
```

e)import re

```
text = "The following example creates an ArrayList with a capacity of 50 elements. Fo
ur elements are then added to the ArrayList and the ArrayList is trimmed accordingly.
"
for m in re.finditer("\d+", text):
    print(m.group(0))
    print("Index position:", m.start())
```

output:

```
50
Index position: 62
```

RESULT: hence the regular expressions implemented successfully in the python

EXPERIMENT -10

DATE:

PRINCIPAL COMPONENT ANALYSIS

AIM: To crate a python code on Program for PRINCIPAL COMPONENT ANALYSIS

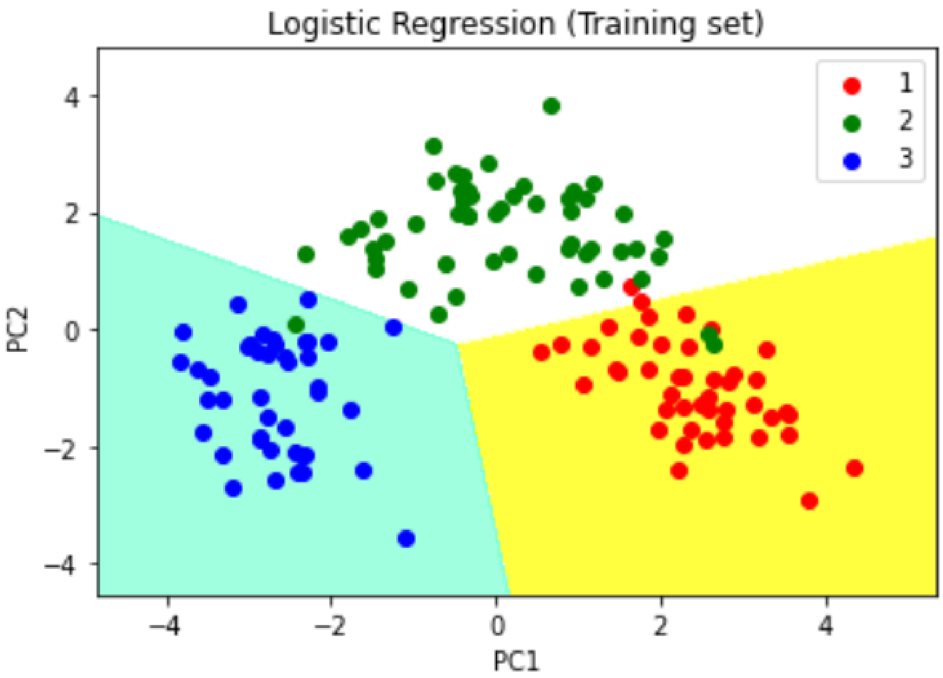
SOURCECODE:

```
# importing required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# importing or loading the dataset
dataset = pd.read_csv(&#39;/content/Wine.csv&#39;)

# distributing the dataset into two components X and Y
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values
# Splitting the X and Y into the
# Training set and Testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test
_size = 0.2, random_state = 0)
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Applying PCA function on training
# and testing set of X component
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
```

```
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_

# Fitting Logistic Regression To the training set


from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
# Predicting the test set result using
# predict function under LogisticRegression
y_pred = classifier.predict(X_test)
# making confusion matrix between
# test set of Y and predicted value.
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
# Predicting the training set
# result through scatter plot
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
        stop = X_set[:, 0].max() + 1, step = 0.01),
        np.arange(start = X_set[:, 1].min() - 1,
        stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
    X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
    cmap = ListedColormap((&#39;yellow&#39;, &#39;white&#39;, &#39;aquamarine&#39;))
)
```

```
plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],

        c = ListedColormap((&#39;red&#39;, &#39;green&#39;, &#39;blue&#39;))(i), l
abel

 = j)

plt.title(&#39;Logistic Regression (Training set)&#39;)

plt.xlabel(&#39;PC1&#39;) # for Xlabel

plt.ylabel(&#39;PC2&#39;) # for Ylabel

plt.legend() # to show legend


# show scatter plot

plt.show()
```

OUTPUT:



RESULT: Program for principal component analysis was successfully executed.

EXPERIMENT -11

DATE:

IMPLEMENTATION OF GRAPH USING ARRAY

AIM: To crate a python code on Program for implementation of graph using array in python

SOURCECODE:

```
# Adjacency Matrix representation of a graph
class Graph:
    # self represents the instance of the class.
    # By using the “self” keyword we can access the
    # attributes and methods of the class in python.
    # It binds the attributes with the given arguments.
    # This constructor is used to initialize the adjacecny matrix
    # with 0.
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                     for row in range(vertices)]

    # This function prints the adjacency matrix of the graph
    # Due to two nested loops, it is O(V^2)
    def printGraph(self):
        print(&quot;\nAdjacency Matrix:&quot;)
        for i in range(self.V):
            for j in range(self.V):
                print(self.graph[i][j], end = &quot; &quot;)
            print()
    # This function is used to add an edge
    # between vertices v and w.
    # This implementation is for undirected graph
    def addEdge(self, v, w):
        print(&quot;Adding an edge between&quot;, v , &quot;and&quot;, w , &quot;and between&quot;, w , &quot;and&quot;, v)
        self.graph[v][w] = 1
        self.graph[w][v] = 1

    # This function is used to add a
    # vertex to the graph.
    def addVertex(self, v):
        self.V += 1
        for i in range(self.V):
            self.graph[i].append(0)
        self.graph.append([0 for column in range(self.V)])

if __name__ == &quot;__main__&quot;:
    # Initialize the graph with 5 vertices
    g = Graph(5)
    # An edge between 0 and 1 and between 1 and 0 will be created
    g.addEdge(0, 1)
    # An edge between 0 and 2 and between 2 and 0 will be created
    g.addEdge(0, 2)

    # An edge between 1 and 2 and between 2 and 1 will be created
    g.addEdge(1, 2)
    # An edge between 2 and 0 and between 2 and 0 will be created
    g.addEdge(2, 0)
```

```
# An edge between 2 and 3 and between 3 and 2 will be created
g.addEdge(2, 3)
# An edge between 3 and 4 and between 4 and 3 will be created
g.addEdge(3, 4)
g.printGraph()
```

OUTPUT:

```
Adding an edge between 0 and 1 and between 1 and 0
Adding an edge between 0 and 2 and between 2 and 0
Adding an edge between 1 and 2 and between 2 and 1
Adding an edge between 2 and 0 and between 0 and 2
Adding an edge between 2 and 3 and between 3 and 2
Adding an edge between 3 and 4 and between 4 and 3
Adjacency Matrix:
0 1 1 0 0
1 0 1 0 0
1 1 0 1 0
0 0 1 0 1
0 0 0 1
```

RESULT:

Program for Implementation of Graph using Array was successfully executed.

EXPERIMENT -12

DATE:

MINIMUM SPANNING TREE – Prim’s Algorithm

AIM: To crate a python code on Program for MINIMUM SPANNING TREE – Prim’s Algorithm

SOURCE CODE:

```
INF = 9999999
        # number of vertices in graph
        N = 5
        #creating graph by adjacency matrix method
        G = [[0, 19, 5, 0, 0],
            [19, 0, 5, 9, 2],
            [5, 5, 0, 1, 6],
            [0, 9, 1, 0, 1],
            [0, 2, 6, 1, 0]]
        selected_node = [0, 0, 0, 0, 0]
        no_edge = 0
        selected_node[0] = True
        # printing for edge and weight
        print(&quot;Edge : Weight\n&quot;)
        while (no_edge &lt; N - 1):

            minimum = INF
            a = 0
            b = 0
            for m in range(N):
                if selected_node[m]:
                    for n in range(N):
                        if ((not selected_node[n]) and G[m][n]):
                            # not in selected and there is an edge
                            if minimum &gt; G[m][n]:
                                minimum = G[m][n]
                                a = m
                                b = n
            print(str(a) + &quot;-&quot; + str(b) + &quot;:&quot; + str(G[a][b]))
            selected_node[b] = True
    no_edge += 1
```

OUTPUT

```
Edge : Weight
0-2:5
2-3:1
3-4:1
4-1:2
```

RESULT:

Program for minimum spanning tree – prim’s algorithm in python was successfully executed

EXPERIMENT -13 **DATE:**

BOOTSTRAP

AIM: To crate a python code on Program for Boot strap in python

SOURCE CODE:

```
import numpy as np
import random
x = np.random.normal(loc= 300.0, size=1000)
print(x)
# Mean of 1000 entries
print(np.mean(x))
sample_mean = []
for i in range(50):
  y = random.sample(x.tolist(), 4)
  # 50 Samples, each of size 4
  print(y)
  avg = np.mean(y)
  # Mean of samples
  print(y)
  sample_mean.append(avg)
print(np.mean(sample_mean))
print(np.mean(x))
print(np.mean(sample_mean))
```

OUTPUT:

```
[302.17297421 299.96828161 297.02889639 300.8049516  299.43552456
 299.87351557 301.51786043 300.08441156 299.15384533 301.02074884]
300.10610101015504
[300.8049515972285, 301.0207488447083, 299.1538453265068, 301.5178604313821]
[300.8049515972285, 301.0207488447083, 299.1538453265068, 301.5178604313821]
[299.1538453265068, 301.0207488447083, 300.8049515972285, 301.5178604313821]
[299.1538453265068, 301.0207488447083, 300.8049515972285, 301.5178604313821]
[302.1729742099062, 299.435524561513, 300.08441155749676, 301.0207488447083]
[302.1729742099062, 299.435524561513, 300.08441155749676, 301.0207488447083]
[299.96828161195157, 299.435524561513, 299.8735155684807, 297.02889639237696]
[299.96828161195157, 299.435524561513, 299.8735155684807, 297.02889639237696]
[297.02889639237696, 299.96828161195157, 299.435524561513, 299.1538453265068]
[297.02889639237696, 299.96828161195157, 299.435524561513, 299.1538453265068]
[299.96828161195157, 297.02889639237696, 299.435524561513, 300.08441155749676]
[299.96828161195157, 297.02889639237696, 299.435524561513, 300.08441155749676]
[299.96828161195157, 299.1538453265068, 302.1729742099062, 299.435524561513]
[299.96828161195157, 299.1538453265068, 302.1729742099062, 299.435524561513]
[299.8735155684807, 301.5178604313821, 299.96828161195157, 297.02889639237696]
[299.8735155684807, 301.5178604313821, 299.96828161195157, 297.02889639237696]
[300.8049515972285, 299.8735155684807, 299.435524561513, 301.0207488447083]
[300.8049515972285, 299.8735155684807, 299.435524561513, 301.0207488447083]
[302.1729742099062, 300.8049515972285, 301.0207488447083, 300.08441155749676]
[302.1729742099062, 300.8049515972285, 301.0207488447083, 300.08441155749676]
[299.1538453265068, 301.5178604313821, 302.1729742099062, 297.02889639237696]
[299.1538453265068, 301.5178604313821, 302.1729742099062, 297.02889639237696]
[299.1538453265068, 301.0207488447083, 299.8735155684807, 297.02889639237696]
[299.1538453265068, 301.0207488447083, 299.8735155684807, 297.02889639237696]
[300.8049515972285, 301.5178604313821, 299.435524561513, 302.1729742099062]
[300.8049515972285, 301.5178604313821, 299.435524561513, 302.1729742099062]
[300.8049515972285, 301.0207488447083, 299.435524561513, 299.8735155684807]
[300.8049515972285, 301.0207488447083, 299.435524561513, 299.8735155684807]
[299.1538453265068, 301.5178604313821, 301.0207488447083, 299.96828161195157]
[299.1538453265068, 301.5178604313821, 301.0207488447083, 299.96828161195157]
[301.0207488447083, 297.02889639237696, 299.1538453265068, 300.08441155749676]
[301.0207488447083, 297.02889639237696, 299.1538453265068, 300.08441155749676]
[300.8049515972285, 299.8735155684807, 297.02889639237696, 301.5178604313821]
[300.8049515972285, 299.8735155684807, 297.02889639237696, 301.5178604313821]
[300.8049515972285, 300.08441155749676, 297.02889639237696, 299.435524561513]
[300.8049515972285, 300.08441155749676, 297.02889639237696, 299.435524561513]
[301.0207488447083, 299.96828161195157, 297.02889639237696, 301.5178604313821]
[301.0207488447083, 299.96828161195157, 297.02889639237696, 301.5178604313821]
[297.02889639237696, 301.5178604313821, 300.08441155749676, 299.96828161195157]
[297.02889639237696, 301.5178604313821, 300.08441155749676, 299.96828161195157]
[299.96828161195157, 302.1729742099062, 297.02889639237696, 300.08441155749676]
[299.96828161195157, 302.1729742099062, 297.02889639237696, 300.08441155749676]
[299.8735155684807, 299.435524561513, 300.08441155749676, 301.0207488447083]
[299.8735155684807, 299.435524561513, 300.08441155749676, 301.0207488447083]
[300.8049515972285, 301.5178604313821, 299.1538453265068, 299.435524561513]
[300.8049515972285, 301.5178604313821, 299.1538453265068, 299.435524561513]
[301.0207488447083, 299.96828161195157, 300.8049515972285, 299.435524561513]
[301.0207488447083, 299.96828161195157, 300.8049515972285, 299.435524561513]
[299.1538453265068, 297.02889639237696, 299.96828161195157, 299.8735155684807]
[299.1538453265068, 297.02889639237696, 299.96828161195157, 299.8735155684807]
[299.8735155684807, 299.435524561513, 297.02889639237696, 299.96828161195157]
```

```
[299.8735155684807, 299.435524561513, 297.02889639237696, 299.96828161195157]
[299.1538453265068, 299.96828161195157, 299.435524561513, 301.5178604313821]
[299.1538453265068, 299.96828161195157, 299.435524561513, 301.5178604313821]
[299.96828161195157, 301.0207488447083, 301.5178604313821, 300.8049515972285]
[299.96828161195157, 301.0207488447083, 301.5178604313821, 300.8049515972285]
[299.96828161195157, 299.8735155684807, 301.0207488447083, 300.08441155749676]
[299.96828161195157, 299.8735155684807, 301.0207488447083, 300.08441155749676]
[299.1538453265068, 302.1729742099062, 301.0207488447083, 299.8735155684807]
[299.1538453265068, 302.1729742099062, 301.0207488447083, 299.8735155684807]
[299.435524561513, 301.5178604313821, 300.8049515972285, 302.1729742099062]
[299.435524561513, 301.5178604313821, 300.8049515972285, 302.1729742099062]
[301.0207488447083, 297.02889639237696, 299.96828161195157, 300.8049515972285]
[301.0207488447083, 297.02889639237696, 299.96828161195157, 300.8049515972285]
[300.8049515972285, 299.435524561513, 299.1538453265068, 302.1729742099062]
[300.8049515972285, 299.435524561513, 299.1538453265068, 302.1729742099062]
[299.1538453265068, 299.8735155684807, 299.96828161195157, 300.8049515972285]
[299.1538453265068, 299.8735155684807, 299.96828161195157, 300.8049515972285]
[301.0207488447083, 299.8735155684807, 300.8049515972285, 299.435524561513]
[301.0207488447083, 299.8735155684807, 300.8049515972285, 299.435524561513]
[302.1729742099062, 297.02889639237696, 299.8735155684807, 299.1538453265068]
[302.1729742099062, 297.02889639237696, 299.8735155684807, 299.1538453265068]
[299.8735155684807, 300.8049515972285, 301.5178604313821, 300.08441155749676]
[299.8735155684807, 300.8049515972285, 301.5178604313821, 300.08441155749676]
[299.1538453265068, 301.5178604313821, 300.8049515972285, 302.1729742099062]
[299.1538453265068, 301.5178604313821, 300.8049515972285, 302.1729742099062]
[302.1729742099062, 301.0207488447083, 301.5178604313821, 299.1538453265068]
[302.1729742099062, 301.0207488447083, 301.5178604313821, 299.1538453265068]
[300.8049515972285, 299.1538453265068, 302.1729742099062, 297.02889639237696]
[300.8049515972285, 299.1538453265068, 302.1729742099062, 297.02889639237696]
[300.8049515972285, 300.08441155749676, 299.1538453265068, 297.02889639237696]
[300.8049515972285, 300.08441155749676, 299.1538453265068, 297.02889639237696]
[300.08441155749676, 297.02889639237696, 299.8735155684807, 301.5178604313821]
[300.08441155749676, 297.02889639237696, 299.8735155684807, 301.5178604313821]
[297.02889639237696, 301.5178604313821, 299.8735155684807, 302.1729742099062]
[297.02889639237696, 301.5178604313821, 299.8735155684807, 302.1729742099062]
[299.8735155684807, 300.08441155749676, 301.5178604313821, 301.0207488447083]
[299.8735155684807, 300.08441155749676, 301.5178604313821, 301.0207488447083]
[301.0207488447083, 299.8735155684807, 302.1729742099062, 300.08441155749676]
[301.0207488447083, 299.8735155684807, 302.1729742099062, 300.08441155749676]
[302.1729742099062, 300.08441155749676, 299.8735155684807, 300.8049515972285]
[302.1729742099062, 300.08441155749676, 299.8735155684807, 300.8049515972285]
[302.1729742099062, 299.1538453265068, 300.08441155749676, 299.96828161195157]
[302.1729742099062, 299.1538453265068, 300.08441155749676, 299.96828161195157]
[301.5178604313821, 297.02889639237696, 299.435524561513, 299.8735155684807]
[301.5178604313821, 297.02889639237696, 299.435524561513, 299.8735155684807]
[299.435524561513, 299.96828161195157, 301.5178604313821, 300.8049515972285]
[299.435524561513, 299.96828161195157, 301.5178604313821, 300.8049515972285]
[300.8049515972285, 299.435524561513, 301.5178604313821, 301.0207488447083]
[300.8049515972285, 299.435524561513, 301.5178604313821, 301.0207488447083]
300.0888454400969
300.10610101015504
300.0888454400969
```

RESULT:

Program For Boot strap in python was successfully executed