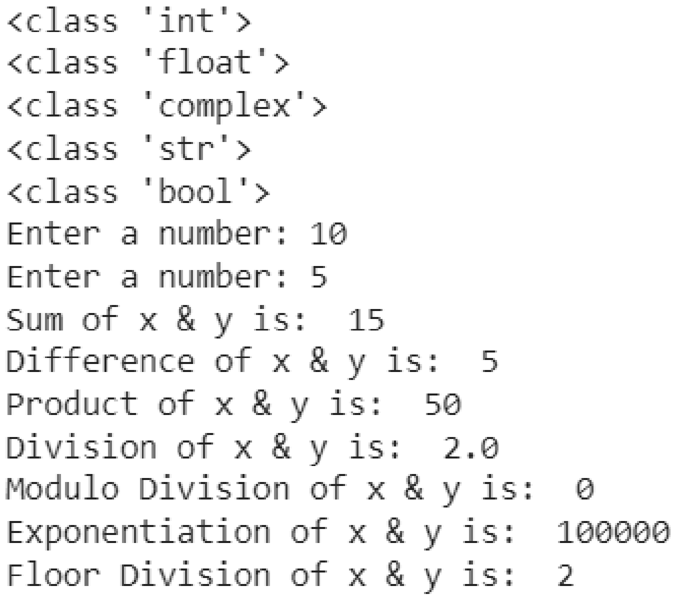
MACHINE LEARNING USING PYTHON LAB PRACTICAL

1)Write a program to demonstrate Python datatypes and variables

```
a=101
print(type(a))
b=0.101
print(type(b))
c=(2+3j)
print(type(c))
d="CBIT"
print(type(d))
i=True
print(type(i))
x=int(input("Enter a number: "))
y=int(input("Enter a number: "))
print("Sum of x & y is: ",(x+y))
print("Difference of x & y is: ",(x-y))
print("Product of x & y is: ",(x*y))
print("Division of x & y is: ",(x/y))
print("Modulo Division of x & y is: ",(x%y))
print("Exponentiation of x & y is: ",(x**y))
print("Floor Division of x & y is: ",(x//y))
```

OUTPUT

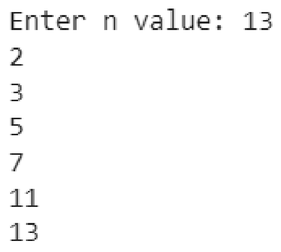


```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'str'>
<class 'bool'>
Enter a number: 10
Enter a number: 5
Sum of x & y is:  15
Difference of x & y is:  5
Product of x & y is:  50
Division of x & y is:  2.0
Modulo Division of x & y is:  0
Exponentiation of x & y is:  100000
Floor Division of x & y is:  2
```

2)Write a Python program to print the prime numbers up to ‘n’

```
n=int(input("Enter n value: "))
i=2
while(i<=n):
  c=0
  for j in range(1,i+1):
    if(i%j==0):
      c=c+1
  if(c==2):
    print(i)
  i=i+1
```

OUTPUT

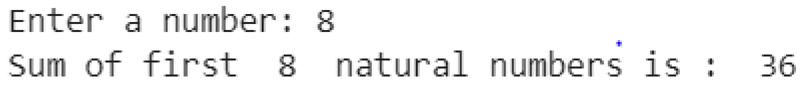


```
Enter n value: 13
2
3
5
7
11
13
```

3)Write a Python program to find the sum of ‘n’ natural numbers using recursion functions

```
def sumuptoN(n):
  if (n==0):
    return 0
  else:
    return n+sumuptoN(n-1)

n=int(input("Enter a number: "))
if(n<0):
  print("Invalid number")
else:
  s=sumuptoN(n)
  print("Sum of first ",n," natural numbers is : ",s)
```

OUTPUT

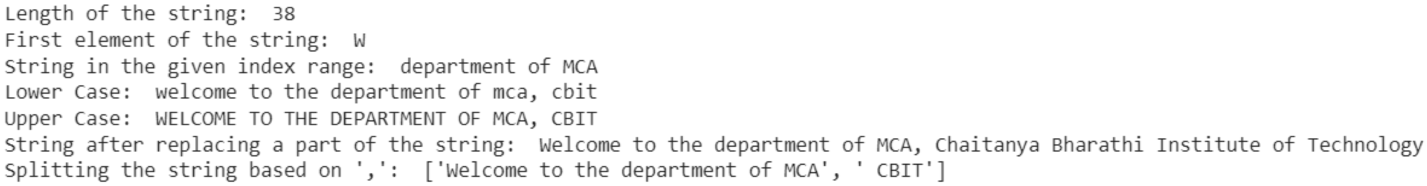


```
Enter a number: 8
Sum of first  8  natural numbers is :  36
```

4)Write a Python program to demonstrate Strings

```
a = "Welcome to the department of MCA, CBIT"
print("Length of the string: ",len(a))
print("First element of the string: ",a[0])
print("String in the given index range: ",a[15:32])
print("Lower Case: ",a.lower())
print("Upper Case: ",a.upper())
print("String after replacing a part of the string: ",a.replace("CBIT","Chaitanya Bharathi Institute
of Technology"))
print("Splitting the string based on ',': ",a.split(","))
```

<u>OUTPUT</u>



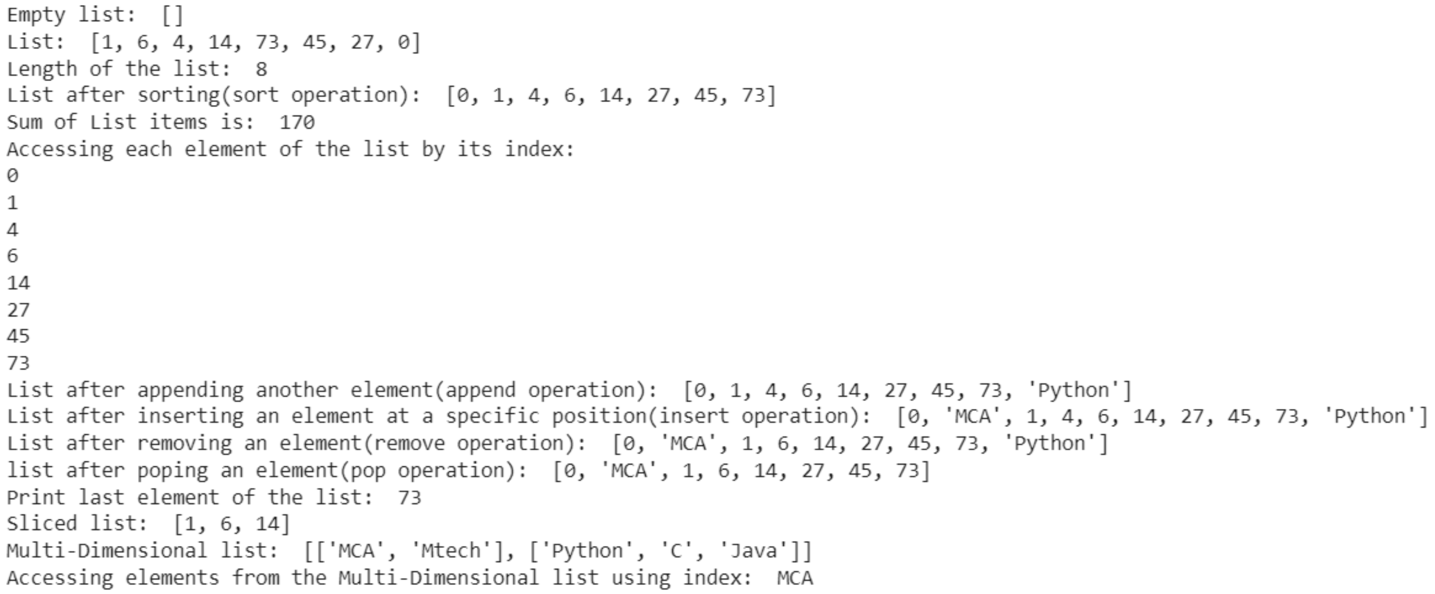
```
Length of the string:  38
First element of the string:  W
String in the given index range:  department of MCA
Lower Case:  welcome to the department of mca, cbit
Upper Case:  WELCOME TO THE DEPARTMENT OF MCA, CBIT
String after replacing a part of the string:  Welcome to the department of MCA, Chaitanya Bharathi Institute of Technology
Splitting the string based on ',':  ['Welcome to the department of MCA', ' CBIT']
```

5)Write a Python program to demonstrate Lists

```
list=[]
print("Empty list: ",list)
list=[1,6,4,14,73,45,27,0]
print("List: ",list)
print("Length of the list: ",len(list))
list.sort()
print("List after sorting(sort operation): ",list)
print("Sum of List items is: ",sum(list))
print("Accessing each element of the list by its index: ")
for i in list:
  print(i)

list.append("Python")
print("List after appending another element(append operation): ",list)
list.insert(1,"MCA")
print("List after inserting an element at a specific position(insert operation): ",list)
list.remove(4)
print("List after removing an element(remove operation): ",list)
list.pop()
print("list after poping an element(pop operation): ",list)

print("Print last element of the list: ",list[-1])

list1=list[2:5]
```

```
print("Sliced list: ",list1)

list2=[["MCA","Mtech"],["Python","C","Java"]]
print("Multi-Dimensional list: ",list2)
print("Accessing elements from the Multi-Dimensional list using index: ",list2[0][0])
```

OUTPUT

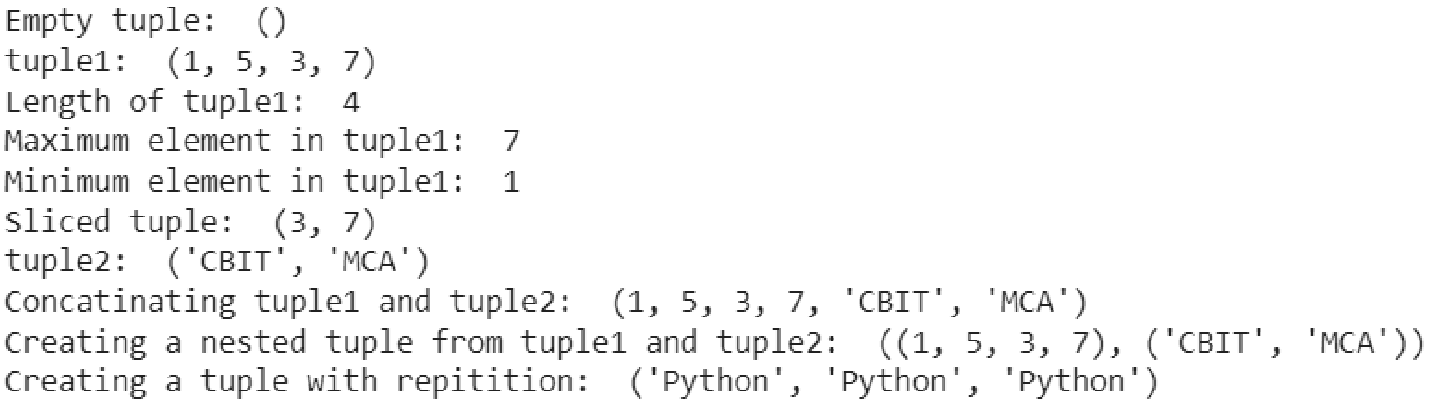


```
Empty list:  []
List:  [1, 6, 4, 14, 73, 45, 27, 0]
Length of the list:  8
List after sorting(sort operation):  [0, 1, 4, 6, 14, 27, 45, 73]
Sum of List items is:  170
Accessing each element of the list by its index:
0
1
4
6
14
27
45
73
List after appending another element(append operation):  [0, 1, 4, 6, 14, 27, 45, 73, 'Python']
List after inserting an element at a specific position(insert operation):  [0, 'MCA', 1, 4, 6, 14, 27, 45, 73, 'Python']
List after removing an element(remove operation):  [0, 'MCA', 1, 6, 14, 27, 45, 73, 'Python']
list after poping an element(pop operation):  [0, 'MCA', 1, 6, 14, 27, 45, 73]
Print last element of the list:  73
Sliced list:  [1, 6, 14]
Multi-Dimensional list:  [['MCA', 'Mtech'], ['Python', 'C', 'Java']]
Accessing elements from the Multi-Dimensional list using index:  MCA
```

6)Write a Python program to demonstrate Tuples

```
tuple=()
print("Empty tuple: ",tuple)

tuple1=(1,5,3,7)
print("tuple1: ",tuple1)
print("Length of tuple1: ",len(tuple1))
print("Maximum element in tuple1: ",max(tuple1))
print("Minimum element in tuple1: ",min(tuple1))
print("Sliced tuple: ",tuple1[2:4])

tuple2=("CBIT","MCA")
print("tuple2: ",tuple2)
print("Concatinating tuple1 and tuple2: ",tuple1+tuple2)

tuple3 = (tuple1, tuple2)
print("Creating a nested tuple from tuple1 and tuple2: ",tuple3)

tuple4=('Python',)*3
print("Creating a tuple with repitition: ",tuple4)
```

OUTPUT

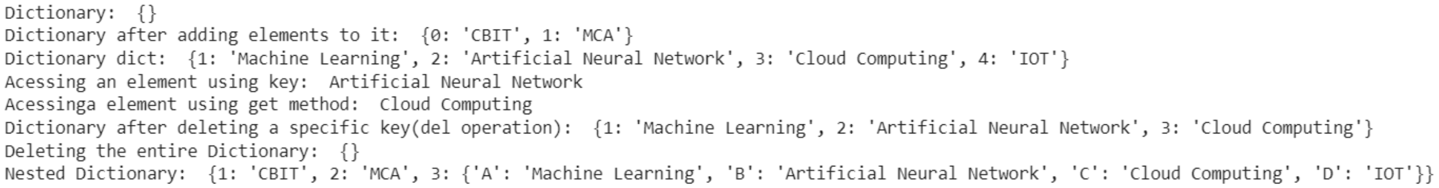


```
Empty tuple:  ()
tuple1:  (1, 5, 3, 7)
Length of tuple1:  4
Maximum element in tuple1:  7
Minimum element in tuple1:  1
Sliced tuple:  (3, 7)
tuple2:  ('CBIT', 'MCA')
Concatinating tuple1 and tuple2:  (1, 5, 3, 7, 'CBIT', 'MCA')
Creating a nested tuple from tuple1 and tuple2:  ((1, 5, 3, 7), ('CBIT', 'MCA'))
Creating a tuple with repitition:  ('Python', 'Python', 'Python')
```

7)Write a Python program to demonstrate Dictionaries

```
Dictionary={}
print("Dictionary: ",Dictionary)
Dictionary[0]='CBIT'
Dictionary[1]='MCA'
print("Dictionary after adding elements to it: ",Dictionary)

dict={1: 'Machine Learning', 2: 'Artificial Neural Network', 3: 'Cloud Computing', 4:'IOT'}
print("Dictionary dict: ",dict)
print("Acessing an element using key: ",dict[2])
print("Acessinga element using get method: ",dict.get(3))
del dict[4]
print("Dictionary after deleting a specific key(del operation): ",dict)
dict.clear()
print("Deleting the entire Dictionary: ",dict)

dict1={1: 'CBIT', 2: 'MCA', 3:{'A' : 'Machine Learning', 'B' : 'Artificial Neural Network', 'C' : 'Cloud Computing', 'D' : 'IOT'}}
print("Nested Dictionary: ",dict1)
```

OUTPUT

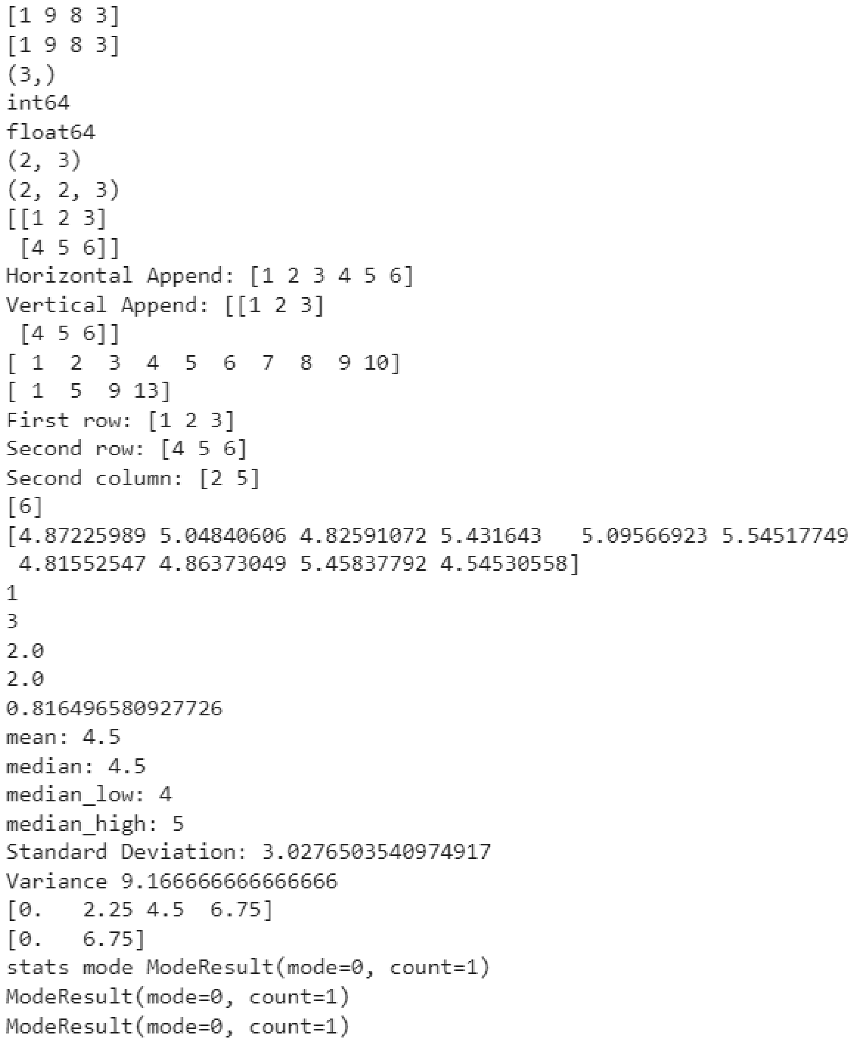


```
Dictionary:  {}
Dictionary after adding elements to it:  {0: 'CBIT', 1: 'MCA'}
Dictionary dict:  {1: 'Machine Learning', 2: 'Artificial Neural Network', 3: 'Cloud Computing', 4: 'IOT'}
Acessing an element using key:  Artificial Neural Network
Acessinga element using get method:  Cloud Computing
Dictionary after deleting a specific key(del operation):  {1: 'Machine Learning', 2: 'Artificial Neural Network', 3: 'Cloud Computing'}
Deleting the entire Dictionary:  {}
Nested Dictionary:  {1: 'CBIT', 2: 'MCA', 3: {'A': 'Machine Learning', 'B': 'Artificial Neural Network', 'C': 'Cloud Computing', 'D': 'IOT'}}
```

8)Write a Python program to demonstrate Packages and Libraries

```
import statistics as st
import numpy as np
myPythonList = [1,9,8,3]
numpy_array_from_list = np.array(myPythonList)
print(numpy_array_from_list)
a = np.array([1,9,8,3])
print(a)
a = np.array([1,2,3])
print(a.shape)
print(a.dtype)
b = np.array([1.1,2.0,3.2])
print(b.dtype)
c = np.array([(1,2,3),
(4,5,6)])
print(c.shape)
d = np.array([
[[1, 2,3], [4, 5, 6]],
[[7, 8,9],[10, 11, 12]]
])
print(d.shape)
np.zeros((2,2), dtype=np.int16)
np.ones((1,2,3), dtype=np.int16)
e = np.array([(1,2,3), (4,5,6)])
print(e)
e.reshape(3,2)
d.flatten()
f = np.array([1,2,3])
g = np.array([4,5,6])
print('Horizontal Append:', np.hstack((f, g)))
print('Vertical Append:', np.vstack((f, g)))
print(np.arange(1, 11))
print(np.arange(1, 14, 4))
print('First row:', e[0])
print('Second row:', e[1])
print('Second column:', e[:,1])
print(e[1, 2:3])
normal_array = np.random.normal(5, 0.5, 10)
print(normal_array)
print(np.min(f))
print(np.max(f))
print(np.mean(f))
print(np.median(f))
print(np.std(f))
x=[0,1,2,3,4,5,6,7,8,9]
print('mean:',st.mean(x))
print('median:',st.median(x))
print('median_low:',st.median_low(x))
print('median_high:',st.median_high(x))
print('Standard Deviation:',st.stdev(x))
print('Variance',st.variance(x))
q1=np.percentile(x,np.arange(0,100,25))
q3=np.percentile(x,np.arange(0,100,75))
print(q1)
print(q3)

from scipy import stats
print('stats mode',stats.mode(x))
print(stats.mode(x,axis=0))
print(stats.mode(x,axis=None))
```

OUTPUT

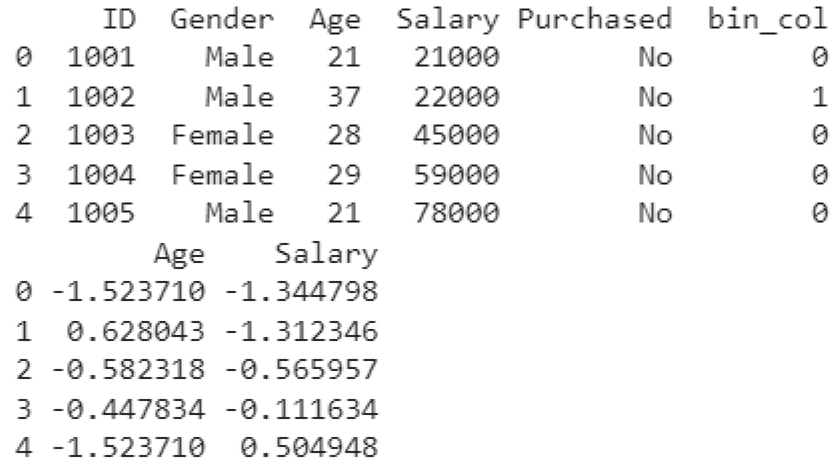


```
[1 9 8 3]
[1 9 8 3]
(3,)
int64
float64
(2, 3)
(2, 2, 3)
[[1 2 3]
 [4 5 6]]
Horizontal Append: [1 2 3 4 5 6]
Vertical Append: [[1 2 3]
 [4 5 6]]
[ 1  2  3  4  5  6  7  8  9 10]
[ 1  5  9 13]
First row: [1 2 3]
Second row: [4 5 6]
Second column: [2 5]
[6]
[4.87225989 5.04840606 4.82591072 5.431643   5.09566923 5.54517749
 4.81552547 4.86373049 5.45837792 4.54530558]
1
3
2.0
2.0
0.816496580927726
mean: 4.5
median: 4.5
median_low: 4
median_high: 5
Standard Deviation: 3.0276503540974917
Variance 9.166666666666666
[0.   2.25 4.5  6.75]
[0.   6.75]
stats mode ModeResult(mode=0, count=1)
ModeResult(mode=0, count=1)
ModeResult(mode=0, count=1)
```

9)Write a program to demonstrate Data Processing Techniques

```
#Binarizer with pandas
import pandas as pd
from sklearn.preprocessing import Binarizer
dataset = pd.read_csv('Age-salary.csv')
features = dataset.iloc[:, [2]].values  # represents age column
binarizer2 = Binarizer(threshold=33)
binarizer_scaled2 = binarizer2.fit_transform(features)
dataset['bin_col'] = binarizer_scaled2
print(dataset.head())

#StandardScaler with pandas
import pandas as pd
from sklearn.preprocessing import StandardScaler
dataset = pd.read_csv('Age-salary.csv')
features = dataset.iloc[:, [2, 3]].values
stscaler = StandardScaler()
scaled = stscaler.fit_transform(features)
print(pd.DataFrame(scaled, columns=['Age', 'Salary']).head())
```

OUTPUT



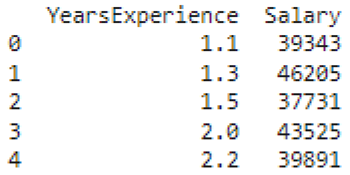
```
     ID  Gender  Age  Salary Purchased  bin_col
0  1001    Male   21   21000        No        0
1  1002    Male   37   22000        No        1
2  1003  Female   28   45000        No        0
3  1004  Female   29   59000        No        0
4  1005    Male   21   78000        No        0
        Age    Salary
0 -1.523710 -1.344798
1  0.628043 -1.312346
2 -0.582318 -0.565957
3 -0.447834 -0.111634
4 -1.523710  0.504948
```

10) Write a program for simple Linear Regression

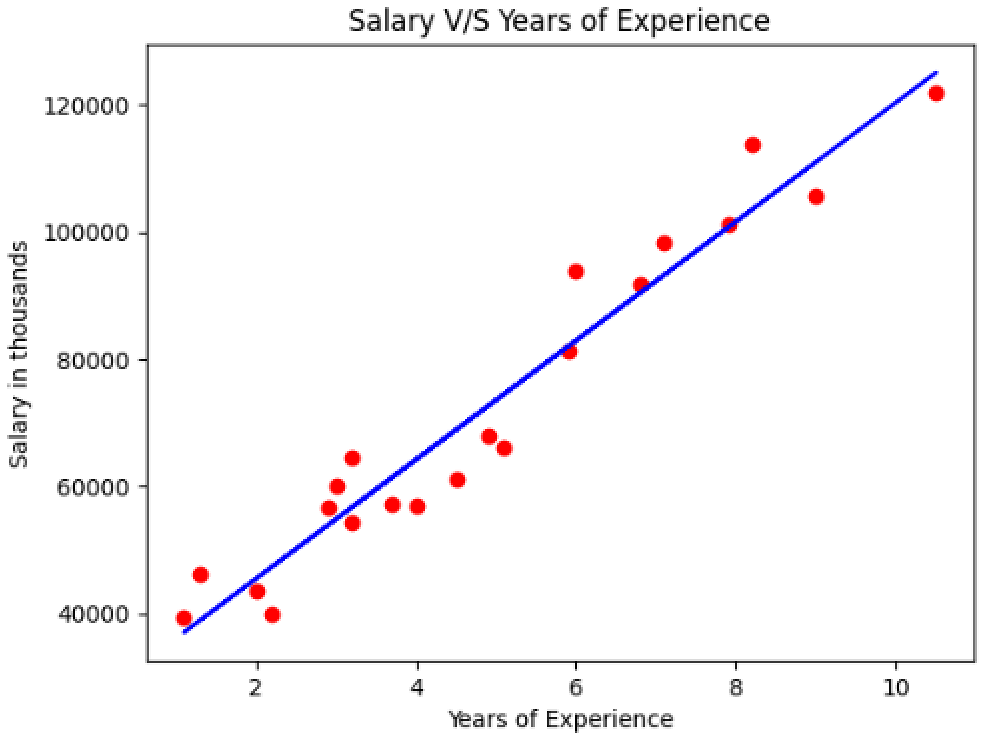
```
import pandas as pd
import matplotlib.pyplot as plt
```

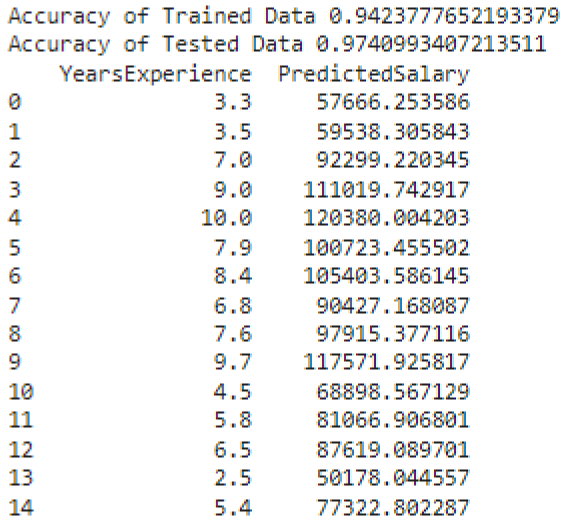
```
df_train = pd.read_csv('SalaryData_Train.csv')

print(df_train.head())

yoe = df_train.iloc[:,0].values
sal = df_train.iloc[:,1].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(yoe,sal,test_size = 0.3,random_state=0)

from sklearn.linear_model import LinearRegression

reg = LinearRegression()

reg.fit(X_train.reshape(-1,1),y_train.reshape(-1,1))

plt.scatter(X_train,y_train,color='r')
y_pred=reg.predict(X_train.reshape(-1,1))
plt.plot(X_train,reg.predict(X_train.reshape(-1,1)),color='b')
plt.xlabel('Years of Experience')
plt.ylabel('Salary in thousands')
plt.title('Salary V/S Years of Experience')
plt.show()

print('Accuracy of Trained Data',reg.score(X_train.reshape(-1,1),y_train.reshape(-1,1)))
print('Accuracy of Tested Data',reg.score(X_test.reshape(-1,1),y_test.reshape(-1,1)))

df_test=pd.read_csv('SalaryData_Test.csv')

feature_test=df_test.iloc[:,:].values
feature_test=feature_test.reshape(-1,1)
y_pred_featuretest=reg.predict(feature_test)
df_test['PredictedSalary']=y_pred_featuretest

print(df_test)
```

OUTPUT



```
   YearsExperience  Salary
0              1.1   39343
1              1.3   46205
2              1.5   37731
3              2.0   43525
4              2.2   39891
```





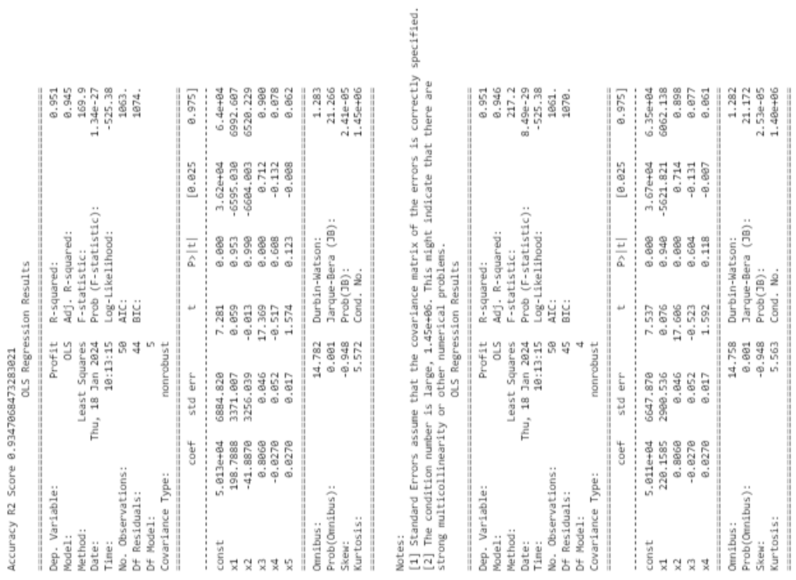
```
Accuracy of Trained Data 0.9423777652193379
Accuracy of Tested Data 0.9740993407213511
    YearsExperience  PredictedSalary
0               3.3     57666.253586
1               3.5     59538.305843
2               7.0     92299.220345
3               9.0    111019.742917
4              10.0    120380.004203
5               7.9    100723.455502
6               8.4    105403.586145
7               6.8     90427.168087
8               7.6     97915.377116
9               9.7    117571.925817
10              4.5     68898.567129
11              5.8     81066.906801
12              6.5     87619.089701
13              2.5     50178.044557
14              5.4     77322.802287
```

11) Write a program to demonstrate Multiple Linear Regression Backward Elimination

```
import numpy as np
import pandas as pd
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 4]
```

```
states=pd.get_dummies(X['State'],drop_first=True)
X=X.drop('State',axis=1)
X=pd.concat([states,X],axis=1)
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)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
from sklearn.metrics import r2_score
score=r2_score(y_test,y_pred)
print('Accuracy R2 Score',score)
import statsmodels.api as sm
X = np.append(arr = np.ones((50, 1)).astype(int), values = X, axis = 1)
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
regressor_OLS = sm.OLS( y, X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 1, 3, 4, 5]]
regressor_OLS = sm.OLS( y,  X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 3, 4, 5]]
regressor_OLS = sm.OLS( y,  X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 3, 5]]
regressor_OLS = sm.OLS(y,  X_opt).fit()
print(regressor_OLS.summary())
X_opt = X[:, [0, 3]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
print(regressor_OLS.summary())
```

OUTPUT

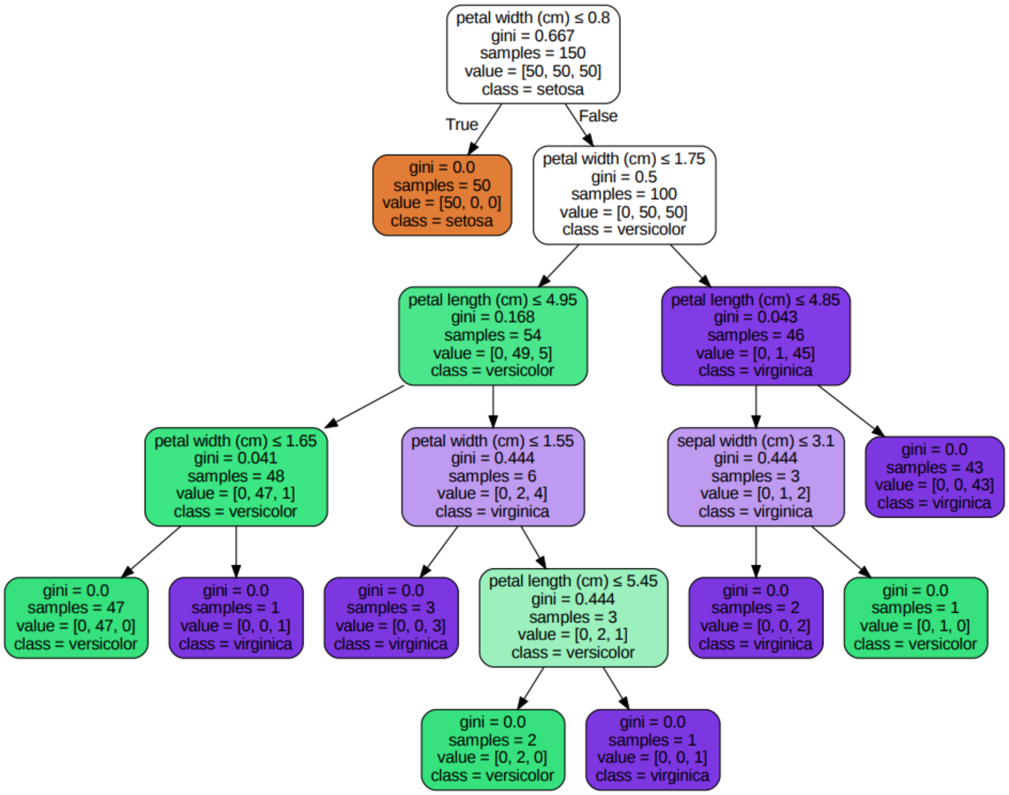


```
Accuracy R2 Score 0.9347068473283021
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Profit   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.945
Method:                 Least Squares   F-statistic:                     169.9
Date:                Thu, 18 Jan 2024   Prob (F-statistic):           1.34e-27
Time:                        10:13:15   Log-Likelihood:                -525.38
No. Observations:                  50   AIC:                             1063.
Df Residuals:                      44   BIC:                             1074.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.013e+04   6884.820      7.281      0.000    3.62e+04     6.4e+04
x1           198.7888   3371.007      0.059      0.953   -6595.030    6992.607
x2           -41.8870   3256.039     -0.013      0.990   -6604.003    6520.229
x3             0.8060      0.046     17.369      0.000       0.712       0.900
x4            -0.0270      0.052     -0.517      0.608      -0.132       0.078
x5             0.0270      0.017      1.574      0.123      -0.008       0.062
==============================================================================
Omnibus:                       14.782   Durbin-Watson:                   1.283
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               21.266
Skew:                          -0.948   Prob(JB):                     2.41e-05
Kurtosis:                       5.572   Cond. No.                     1.45e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.45e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Profit   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.946
Method:                 Least Squares   F-statistic:                     217.2
Date:                Thu, 18 Jan 2024   Prob (F-statistic):           8.49e-29
Time:                        10:13:15   Log-Likelihood:                -525.38
No. Observations:                  50   AIC:                             1061.
Df Residuals:                      45   BIC:                             1070.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5.011e+04   6647.870      7.537      0.000    3.67e+04    6.35e+04
x1           220.1585   2900.536      0.076      0.940   -5621.821    6062.138
x2             0.8060      0.046     17.606      0.000       0.714       0.898
x3            -0.0270      0.052     -0.523      0.604      -0.131       0.077
x4             0.0270      0.017      1.592      0.118      -0.007       0.061
==============================================================================
Omnibus:                       14.758   Durbin-Watson:                   1.282
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               21.172
Skew:                          -0.948   Prob(JB):                     2.53e-05
Kurtosis:                       5.563   Cond. No.                     1.40e+06
==============================================================================
```

12) Write a program to demonstrate CART (Classification and Regression Test)

```
from sklearn import tree
from sklearn.datasets import load_iris

iris = load_iris()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)

import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iriscart")

dot_data = tree.export_graphviz(clf, out_file=None,
                    feature_names=iris.feature_names,
                    class_names=iris.target_names,
                    filled=True, rounded=True,
                    special_characters=True)
graph = graphviz.Source(dot_data)
graph.view()
```

OUTPUT



13) Write a program for Decision Tree

```
from sklearn.datasets import load_iris
from sklearn import tree
iris45 = load_iris()
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf.fit(iris45.data, iris45.target)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris45.data, iris45.target, test_size = 0.2,
random_state = 0)

clf.score(iris45.data, iris45.target)
predicted= clf.predict(X_test)

import graphviz

dot_data = tree.export_graphviz(clf, out_file=None, feature_names=iris45.feature_names,
class_names=iris45.target_names, filled=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph.view()
```

OUTPUT

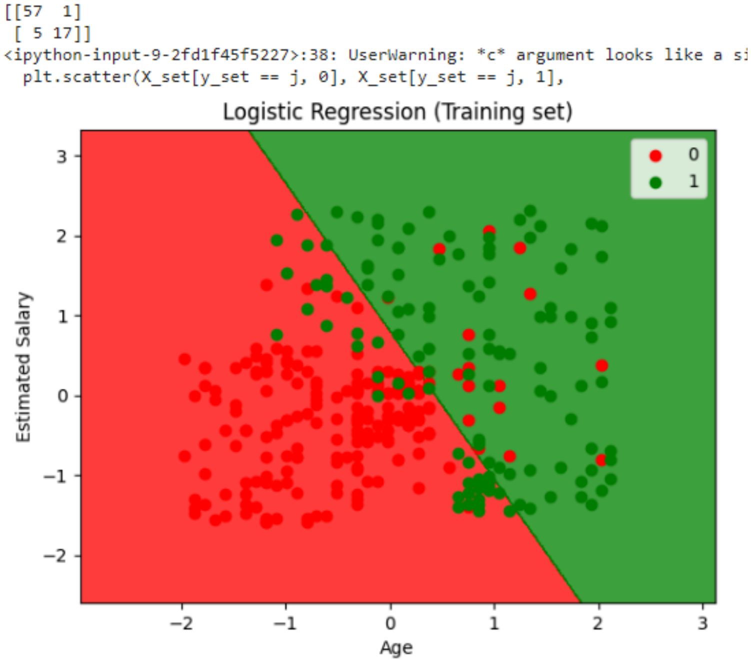
Same as 12th Output

14) Write a program for Logistic Regression

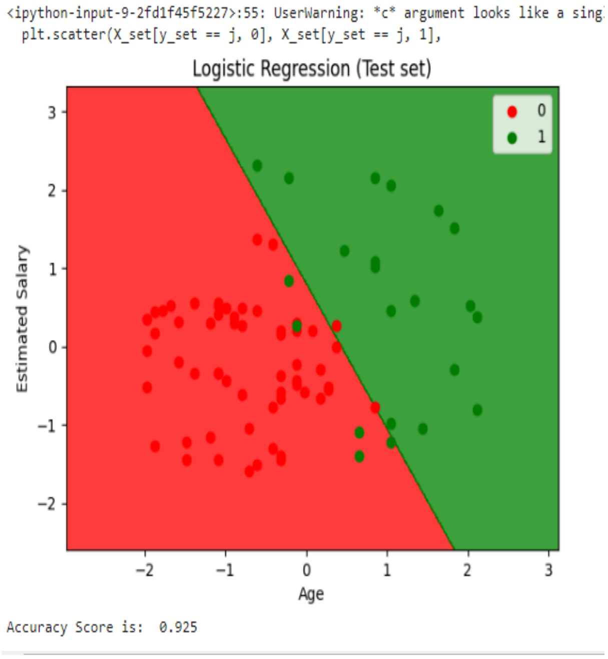
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
```

```
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

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(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
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(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
score = classifier.score(X_test, y_test)
print('Accuracy Score is: ',score)
```

OUTPUT



```
[[57  1]
 [ 5 17]]
<ipython-input-9-2fd1f45f5227>:38: UserWarning: *c* argument looks like a si
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



```
<ipython-input-9-2fd1f45f5227>:55: UserWarning: *c* argument looks like a sing
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

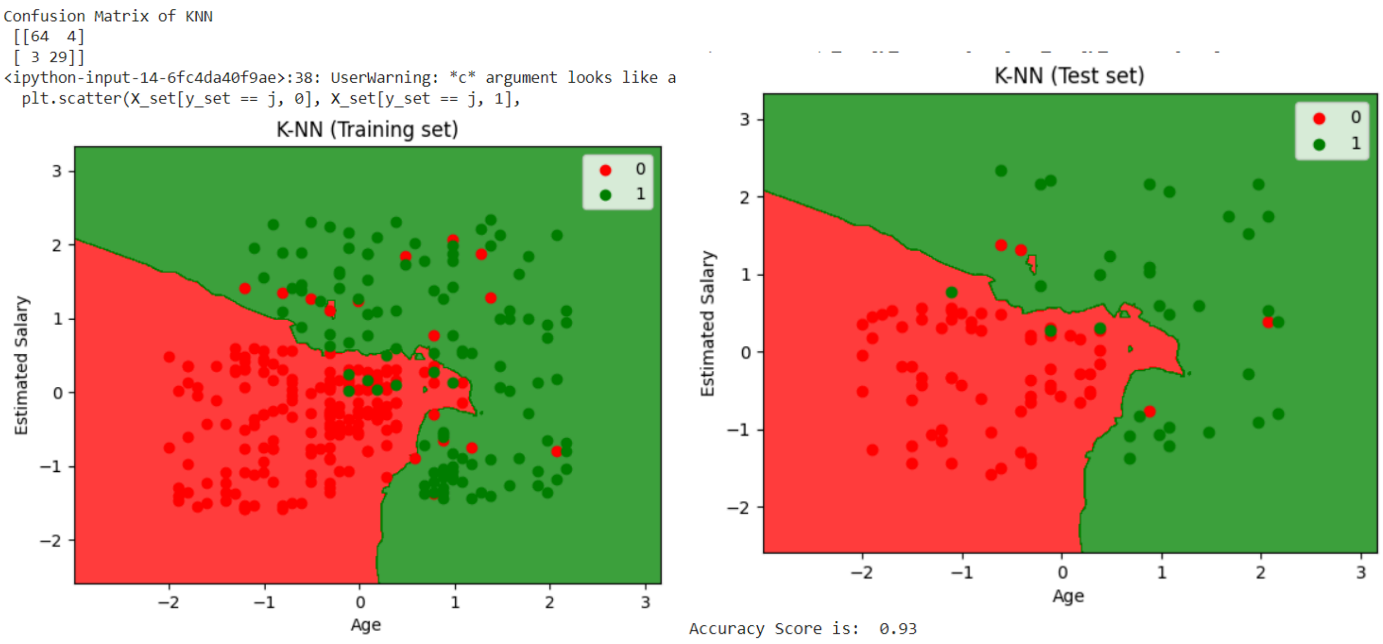
```
Accuracy Score is:  0.925
```

15) Write a program to implement the K Nearest Neighbour Algorithm

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

dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix of KNN \n",cm)

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(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
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(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

score = classifier.score(X_test, y_test)
print('Accuracy Score is: ',score)
```

OUTPUT



```
Confusion Matrix of KNN
 [[64  4]
 [ 3 29]]
<ipython-input-14-6fc4da40f9ae>:38: UserWarning: *c* argument looks like a
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

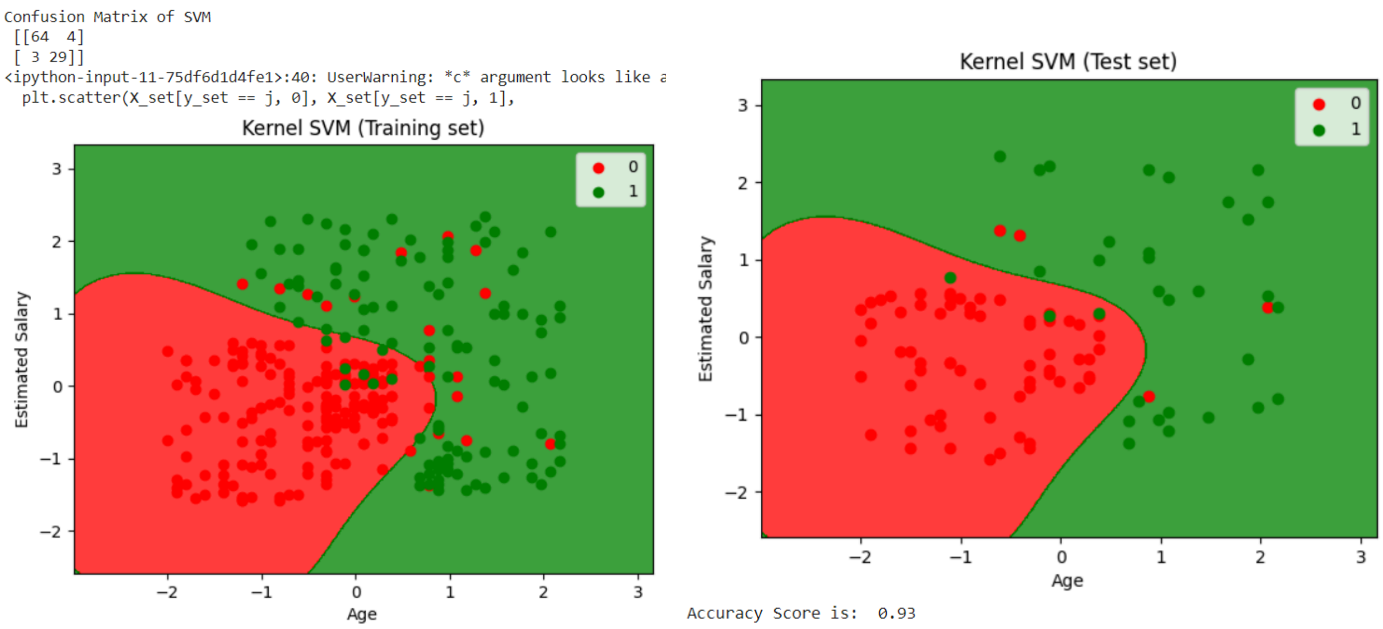
```
Accuracy Score is:  0.93
```

16) Write a program to implement a Support Vector Machine (SVM) with different kernels

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

dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print("Confusion Matrix of SVM \n",cm)

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(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
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(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

score = classifier.score(X_test, y_test)
print('Accuracy Score is: ',score)
```

OUTPUT



```
Confusion Matrix of SVM
 [[64  4]
 [ 3 29]]
<ipython-input-11-75df6d1d4fe1>:40: UserWarning: *c* argument looks like a
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

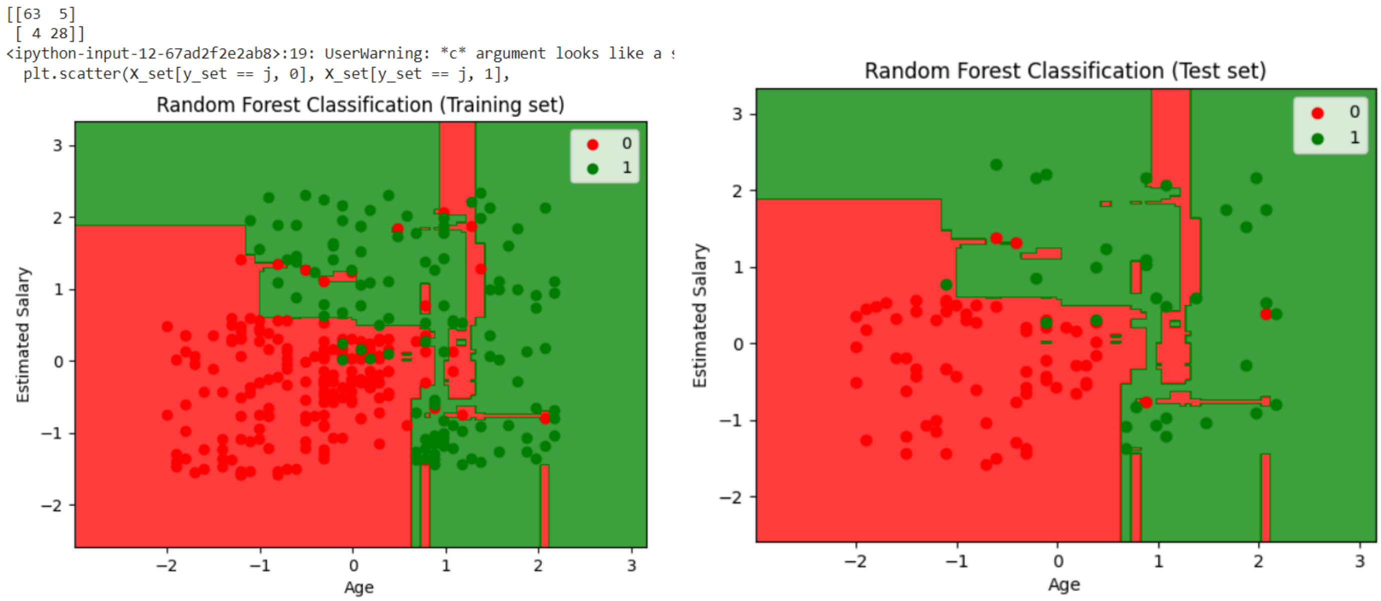
```
Accuracy Score is:  0.93
```

17) Write a program to implement Random Forest Classification

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

def visualize_results(X_set, y_set, title):
    X1, X2 = np.meshgrid(np.arange(X_set[:, 0].min() - 1, X_set[:, 0].max() + 1, step=0.01),
                         np.arange(X_set[:, 1].min() - 1, 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(('red', 'green')))
    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(('red', 'green'))(i), label=j)
    plt.title(title)
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.legend()
    plt.show()

dataset = pd.read_csv('Social_Network_Ads.csv')
```

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=0)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print(cm)

visualize_results(X_train, y_train, 'Random Forest Classification (Training set)')

visualize_results(X_test, y_test, 'Random Forest Classification (Test set)')
```

<u>OUTPUT</u>



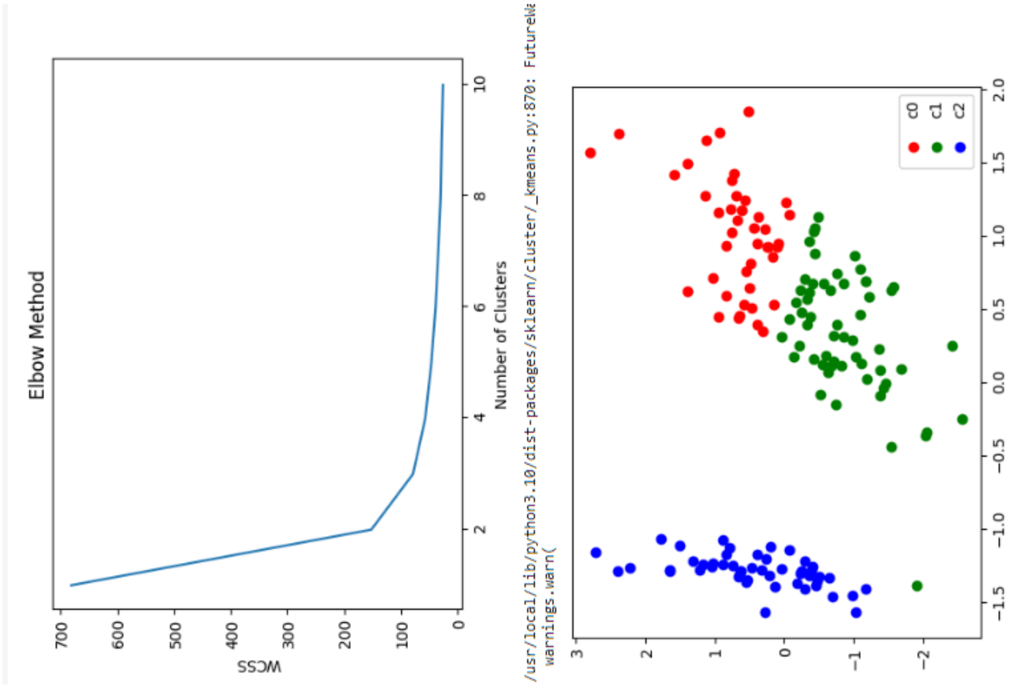
```
[[63  5]
 [ 4 28]]
<ipython-input-12-67ad2f2e2ab8>:19: UserWarning: *c* argument looks like a
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

Random Forest Classification (Training set)
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3
0
1
Random Forest Classification (Test set)
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3
0
1

18) Write a program to implement the K-Means Clustering Algorithm

```
from sklearn.datasets import load_iris
from itertools import cycle
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from numpy.random import RandomState
import pylab as pl
import matplotlib.pyplot as plt

class clustering:
    def __init__(self):
        self.plot(load_iris().data)

    def plot(self, X):
        wcss=[]
        for i in range(1,11):

kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
            kmeans.fit(X)
            wcss.append(kmeans.inertia_)
        plt.plot(range(1,11),wcss)
        plt.title('Elbow Method')
        plt.xlabel('Number of Clusters')
        plt.ylabel('WCSS')
        plt.show()

        pca = PCA(n_components=2, whiten=True).fit(X)
        X_pca = pca.transform(X)
        kmeans = KMeans(n_clusters=3, random_state=RandomState(42)).fit(X_pca)
        plot_2D(X_pca, kmeans.labels_, ["c0", "c1", "c2"])
def plot_2D(data, target, target_names):
    colors = cycle('rgbcmykw')
    target_ids = range(len(target_names))
    pl.figure()
    for i, c, label in zip(target_ids, colors, target_names):
        pl.scatter(data[target == i, 0], data[target == i, 1],
                                        c=c, label=label)
    pl.legend()
    pl.show()

if __name__ == '__main__':
          c = clustering()
```

OUTPUT

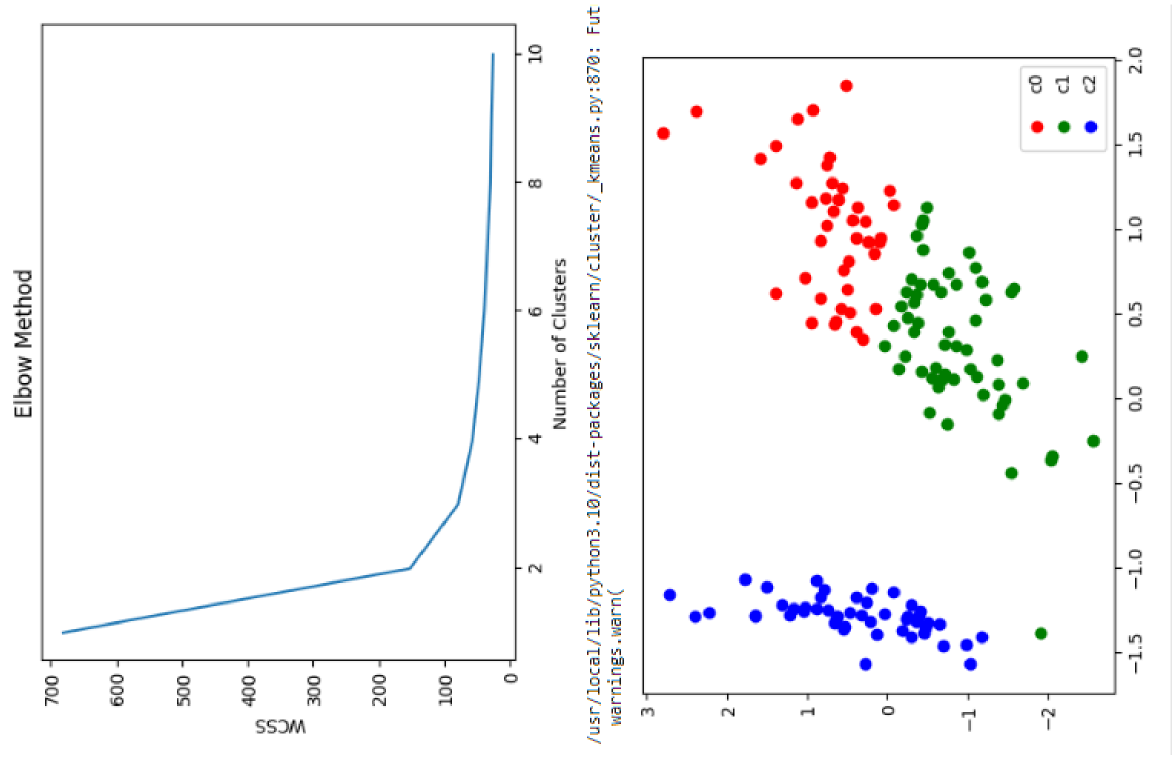


```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWa
  warnings.warn(
```

19) Write a program to implement Hierarchical Clustering Algorithm

```
from sklearn.datasets import load_iris
from itertools import cycle
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from numpy.random import RandomState
import pylab as pl
import matplotlib.pyplot as plt

class clustering:
    def __init__(self):
        self.plot(load_iris().data)

    def plot(self, X):
        wcss=[]
        for i in range(1,11):

kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
            kmeans.fit(X)
            wcss.append(kmeans.inertia_)
        plt.plot(range(1,11),wcss)
        plt.title('Elbow Method')
        plt.xlabel('Number of Clusters')
        plt.ylabel('WCSS')
        plt.show()
        pca = PCA(n_components=2, whiten=True).fit(X)
        X_pca = pca.transform(X)
        kmeans = KMeans(n_clusters=3, random_state=RandomState(42)).fit(X_pca)
        plot_2D(X_pca, kmeans.labels_, ["c0", "c1", "c2"])
def plot_2D(data, target, target_names):
    colors = cycle('rgbcmykw')
    target_ids = range(len(target_names))
    pl.figure()
    for i, c, label in zip(target_ids, colors, target_names):
        pl.scatter(data[target == i, 0], data[target == i, 1],
                                                    c=c, label=label)
    pl.legend()
    pl.show()

if __name__ == '__main__':
        c = clustering()
```

<u>OUTPUT</u>



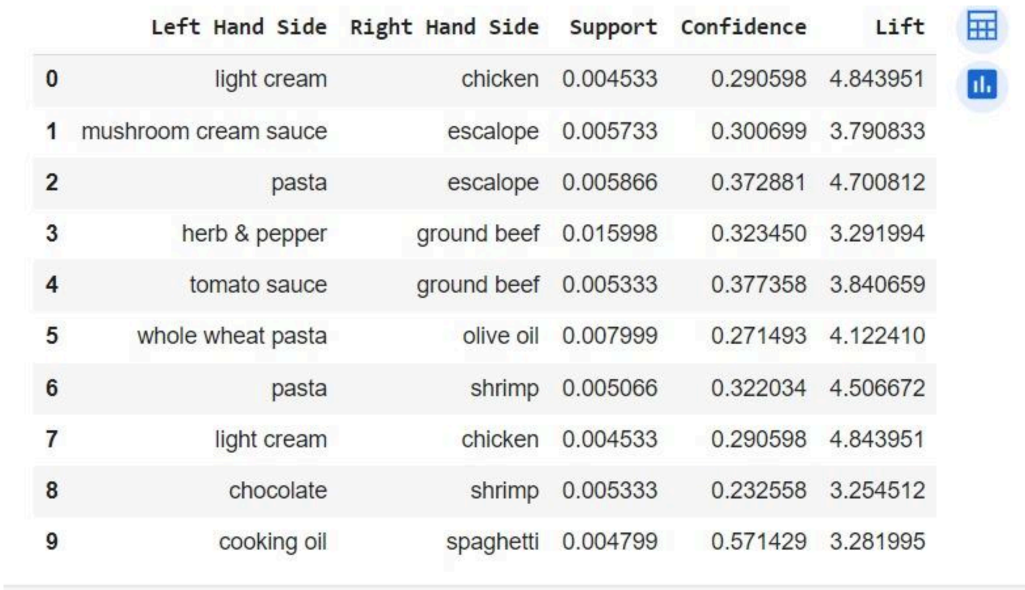
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Fut
  warnings.warn(
```

20) Write a program to implement the Apriori Algorithm

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

```
dataset = pd.read_csv('Market_Basket_Optimisation.csv', low_memory=False, header=None)
!pip install apyori
list_of_transactions = []
for i in range(0, 7501):
   list_of_transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])
list_of_transactions[0]
from apyori import apriori
rules = apriori(list_of_transactions, min_support = 0.004, min_confidence = 0.2, min_lift = 3,
min_length = 2)
results = list(rules)
def inspect(results):
   lhs  =  [tuple(result [2] [0] [0]) [0] for result in results]
   rhs  =  [tuple(result [2] [0] [1]) [0] for result in results]
   supports = [result [1] for result in results]
   confidences = [result [2] [0] [2]   for result in results]
   lifts = [result [2] [0] [3]   for result in results]
   return list(zip(lhs,rhs,supports,confidences, lifts))
resultsinDataFrame = pd.DataFrame(inspect(results),columns = ['Left Hand Side', 'Right Hand
Side', 'Support', 'Confidence', 'Lift'] )
resultsinDataFrame.head(10)
```

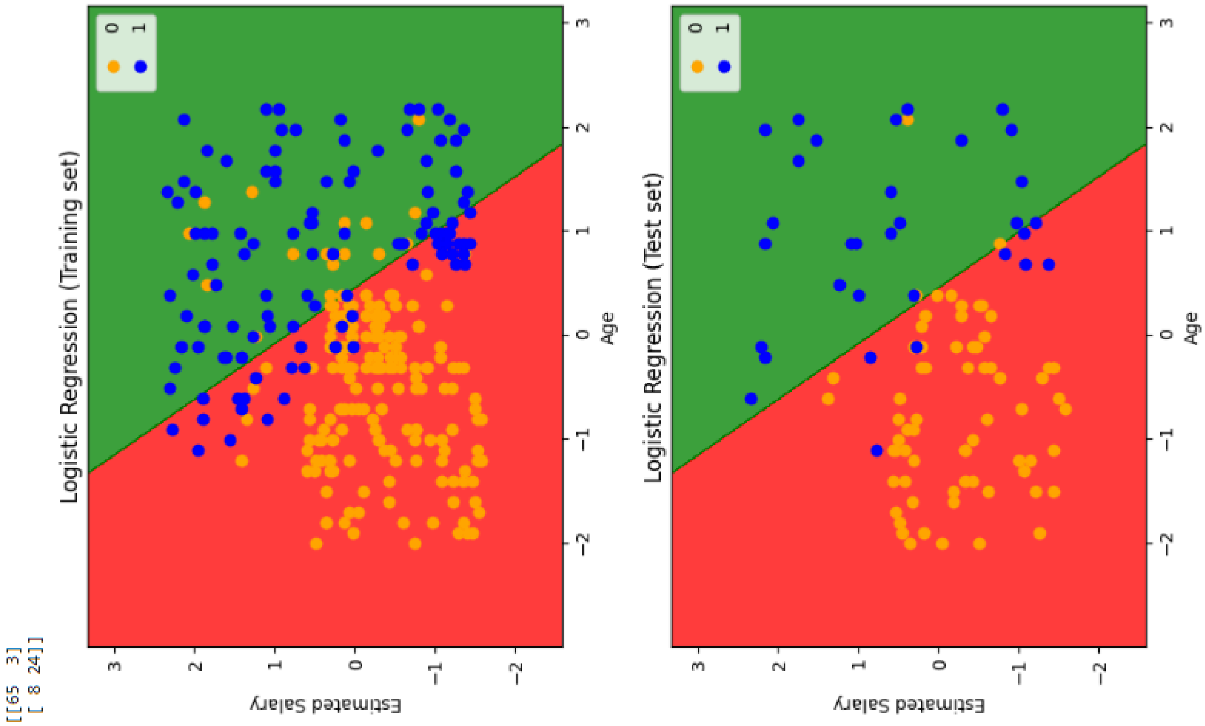
OUTPUT



	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
2	pasta	escalope	0.005866	0.372881	4.700812
3	herb & pepper	ground beef	0.015998	0.323450	3.291994
4	tomato sauce	ground beef	0.005333	0.377358	3.840659
5	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
6	pasta	shrimp	0.005066	0.322034	4.506672
7	light cream	chicken	0.004533	0.290598	4.843951
8	chocolate	shrimp	0.005333	0.232558	3.254512
9	cooking oil	spaghetti	0.004799	0.571429	3.281995

21) Write a program to demonstrate the logistic regression using the sigmoid function.

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
    dataset = pd.read_csv('Social_Network_Ads.csv')
    features = dataset.iloc[:, [2, 3]].values
    labels = dataset.iloc[:, 4].values
    x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.25,
random_state=0)
    scaler = preprocessing.StandardScaler()
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)
    classifier = LogisticRegression(random_state=0)
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)

    def visualize(x_label, y_label, title, features, labels):
        col_zero = features[:, 0]
        col_one = features[:, 1]
        start1 = col_zero.min() - 1
        stop1 = col_zero.max() + 1
        start2 = col_one.min() - 1
        stop2 = col_one.max() + 1
        xi = np.arange(start1, stop1, 0.01)
        yi = np.arange(start2, stop2, 0.01)
        x, y = np.meshgrid(xi, yi)
        predict_data = np.array([x.ravel(), y.ravel()]).transpose()
        y_pred = classifier.predict(predict_data).reshape(x.shape)
        plt.contourf(x, y, y_pred, alpha=0.75, cmap=ListedColormap(('red', 'green')))
        plt.xlim(x.min(), x.max())
        plt.ylim(y.min(), y.max())
        unique_labels = np.unique(labels)
        for index, value in enumerate(unique_labels):
            scatter_x = features[labels == value, 0]
            scatter_y = features[labels == value, 1]
            color = ListedColormap(('orange', 'blue'))(index)
            plt.scatter(scatter_x, scatter_y, color=color, label=value)
        plt.title(title)
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        plt.legend()
        plt.show()

    visualize('Age', 'Estimated Salary', 'Logistic Regression (Training set)', x_train.copy(),
y_train.copy())
    visualize('Age', 'Estimated Salary', 'Logistic Regression (Test set)', x_test.copy(),
y_test.copy())

except Exception as e:
    print(e)
```

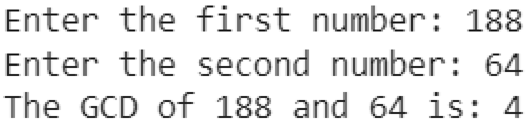


```
[[65  3]
 [ 8 24]]
```

22) Write a python program to find the GCD of two numbers using recursive functions.

```
def gcd_recursive(a, b):
    if b == 0:
        return a
    else:
        return gcd_recursive(b, a % b)

num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
if(num1>num2):
    result = gcd_recursive(num1, num2)
else:
      result = gcd_recursive(num2, num1)
```

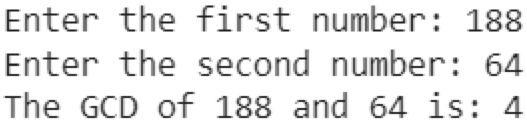
```
print(f"The GCD of {num1} and {num2} is: {result}")
```



```
Enter the first number: 188
Enter the second number: 64
The GCD of 188 and 64 is: 4
```

23) Write a python program to find the gcd of a given number using functions.

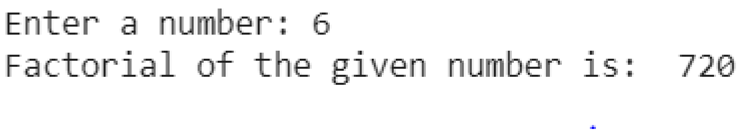
```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a


num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

if(num1>num2):
  result = gcd(num1, num2)
else:
  result = gcd(num2, num1)
print(f"The GCD of {num1} and {num2} is: {result}")
```



```
Enter the first number: 188
Enter the second number: 64
The GCD of 188 and 64 is: 4
```

24) Write a python program to find the factorial of a given number using functions

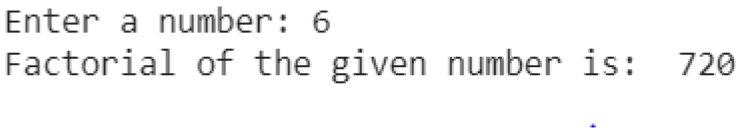
```
def fact(n):
  f=1
  while n>0:
    f=f*n
    n=n-1
  return f

n=int(input("Enter a number: "))
if(n<0):
  print("Invalid input")
else:
  print("Factorial of the given number is: ",fact(n))
```



```
Enter a number: 6
Factorial of the given number is:  720
```

25) Write a python program to find the factorial of a given number using recursive Functions

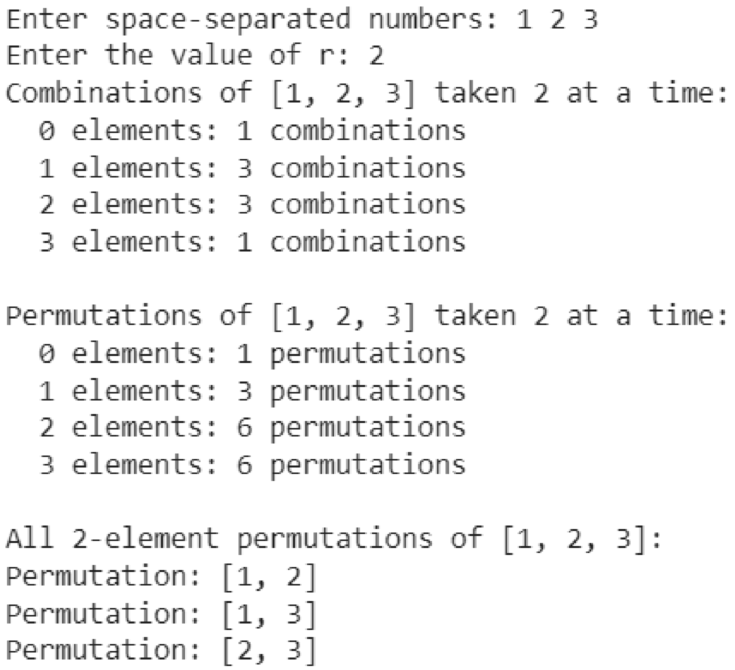
```
def recfact(n):
  if n==0 or n==1:
    return 1
  else:
    return n*recfact(n-1)
n=int(input("Enter a number: "))
if(n<0):
  print("Invalid input")
else:
  print("Factorial of the given number is: ",recfact(n))
```



```
Enter a number: 6
Factorial of the given number is:  720
```

26) A recursive program to find the Permutations and combinations of the given numbers.

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

def combinations(n, r):
    return factorial(n) // (factorial(r) * factorial(n - r))

def permutations(n, r):
    return factorial(n) // factorial(n - r)

def generate_permutations_combinations(nums, selected_nums, index, r):
    if r == 0:
        print("Permutation:", selected_nums)
        return

    if index == len(nums):
        return

    selected_nums.append(nums[index])
    generate_permutations_combinations(nums, selected_nums, index + 1, r - 1)
    selected_nums.pop()  # Backtrack

    generate_permutations_combinations(nums, selected_nums, index + 1, r)

if __name__ == "__main__":
    numbers = list(map(int, input("Enter space-separated numbers: ").split()))
    r = int(input("Enter the value of r: "))

    print(f"Combinations of {numbers} taken {r} at a time:")
    for i in range(len(numbers) + 1):
        print(f"  {i} elements: {combinations(len(numbers), i)} combinations")

    print(f"\nPermutations of {numbers} taken {r} at a time:")
    for i in range(len(numbers) + 1):
        print(f"  {i} elements: {permutations(len(numbers), i)} permutations")

    print(f"\nAll {r}-element permutations of {numbers}:")
    generate_permutations_combinations(numbers, [], 0, r)
```



```
Enter space-separated numbers: 1 2 3
Enter the value of r: 2
Combinations of [1, 2, 3] taken 2 at a time:
  0 elements: 1 combinations
  1 elements: 3 combinations
  2 elements: 3 combinations
  3 elements: 1 combinations

Permutations of [1, 2, 3] taken 2 at a time:
  0 elements: 1 permutations
  1 elements: 3 permutations
  2 elements: 6 permutations
  3 elements: 6 permutations

All 2-element permutations of [1, 2, 3]:
Permutation: [1, 2]
Permutation: [1, 3]
Permutation: [2, 3]
```

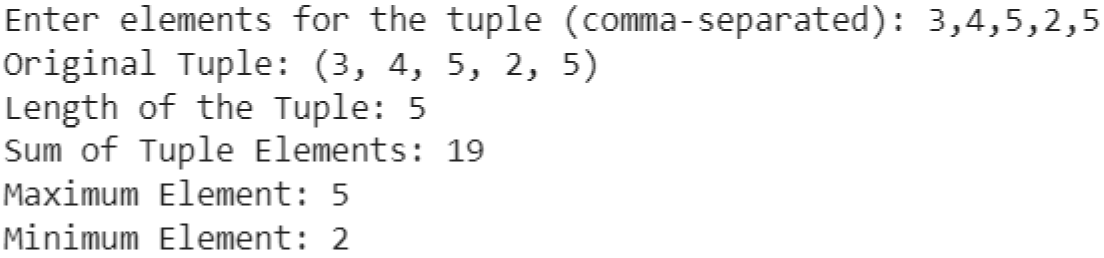
```
import pandas as pd
import numpy as np
from sklearn import datasets
import seaborn as sns
```

27) A Program for loading different datasets in Python

```
# 1. Load CSV file using pandas
df_csv = pd.read_csv("data/my_data.csv")
print(df_csv.head())  # Preview the first few rows

# 2. Load text file using NumPy
data_txt = np.loadtxt("data/my_data.txt")
print(data_txt)

# 3. Load dataset from scikit-learn
iris = datasets.load_iris()
print(iris.data)  # Features
print(iris.target)  # Labels

# 4. Load dataset from Seaborn
penguins = sns.load_dataset("penguins")
print(penguins.head())

# 5. Load Excel file using pandas
df_excel = pd.read_excel("data/my_data.xlsx")
print(df_excel.head())

# 6. Load JSON file using pandas
df_json = pd.read_json("data/my_data.json")
print(df_json.head())
```

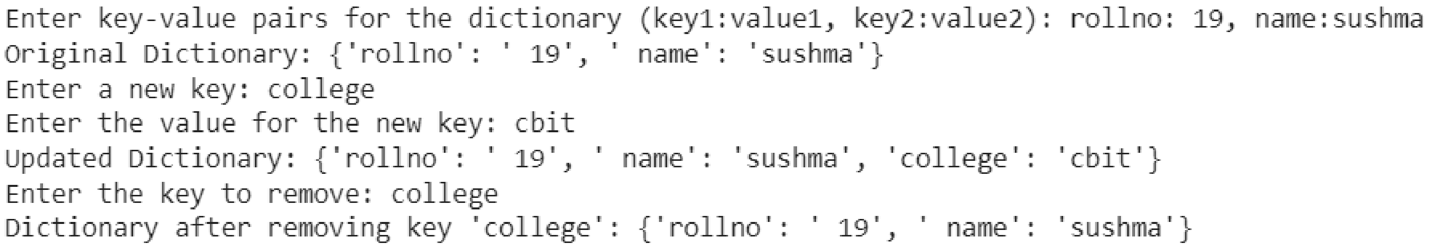
28) A program on tuples and dictionaries using user defined functions

```
# User-defined function to create a tuple
def create_tuple():
    elements = input("Enter elements for the tuple (comma-separated): ")
    user_tuple = tuple(map(int, elements.split(',')))
    return user_tuple

# User-defined function to manipulate the tuple
def manipulate_tuple(user_tuple):
    print(f"Original Tuple: {user_tuple}")
    print(f"Length of the Tuple: {len(user_tuple)}")
    print(f"Sum of Tuple Elements: {sum(user_tuple)}")
    print(f"Maximum Element: {max(user_tuple)}")
    print(f"Minimum Element: {min(user_tuple)}")

if __name__ == "__main__":
    # Create a tuple and manipulate it
    user_tuple = create_tuple()
    manipulate_tuple(user_tuple)
```



```
Enter elements for the tuple (comma-separated): 3,4,5,2,5
Original Tuple: (3, 4, 5, 2, 5)
Length of the Tuple: 5
Sum of Tuple Elements: 19
Maximum Element: 5
Minimum Element: 2
```

```
# User-defined function to create a dictionary
def create_dictionary():
    key_value_pairs = input("Enter key-value pairs for the dictionary (key1:value1, key2:value2): ")
    key_value_list = key_value_pairs.split(',')
    user_dict = dict(item.split(':') for item in key_value_list)
    return user_dict

# User-defined function to manipulate the dictionary
def manipulate_dictionary(user_dict):
    print(f"Original Dictionary: {user_dict}")

    # Add a new key-value pair to the dictionary
    new_key = input("Enter a new key: ")
    new_value = input("Enter the value for the new key: ")
    user_dict[new_key] = new_value
    print(f"Updated Dictionary: {user_dict}")

    # Remove a key-value pair from the dictionary
    key_to_remove = input("Enter the key to remove: ")
    if key_to_remove in user_dict:
        del user_dict[key_to_remove]
        print(f"Dictionary after removing key '{key_to_remove}': {user_dict}")
    else:
        print(f"Key '{key_to_remove}' not found in the dictionary.")

if __name__ == "__main__":
    # Create a dictionary and manipulate it
    user_dict = create_dictionary()
    manipulate_dictionary(user_dict)
```



```
Enter key-value pairs for the dictionary (key1:value1, key2:value2): rollno: 19, name:sushma
Original Dictionary: {'rollno': ' 19', ' name': 'sushma'}
Enter a new key: college
Enter the value for the new key: cbit
Updated Dictionary: {'rollno': ' 19', ' name': 'sushma', 'college': 'cbit'}
Enter the key to remove: college
Dictionary after removing key 'college': {'rollno': ' 19', ' name': 'sushma'}
```