DSML_midOne

December 10, 2023

[2]:
```
#1a. Write a python program to demonstrate Python Datatypes, 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))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'str'>
<class 'bool'>
Enter a number:  5Enter a number:  2Sum of x & y is:  7
Difference of x & y is:  3
Product of x & y is:  10
Division of x & y is:  2.5
Modulo Division of x & y is:  1
Exponentiation of x & y is:  25
Floor Division of x & y is:  2
```

[3]:
```
#1b. 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()

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

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Random Forest Classification to the Training set
classifier = RandomForestClassifier(n_estimators=10, criterion='entropy',␣
 ↪random_state=0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
```

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

# Visualising the Training set results
visualize_results(X_train, y_train, 'Random Forest Classification (Training␣
 ↪set)')

# Visualising the Test set results
visualize_results(X_test, y_test, 'Random Forest Classification (Test set)')
```

```
[[63  5]
 [ 4 28]]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_1041/54424465.py in <cell line: 51>()
     49
     50 # Visualising the Training set results
---> 51 visualize_results(X_train, y_train, 'Random Forest Classification␣
 ↪(Training set)')
     52
     53 # Visualising the Test set results

/tmp/ipykernel_1041/54424465.py in visualize_results(X_set, y_set, title)
     12                          np.arange(X_set[:, 1].min() - 1, X_set[:, 1].
 ↪max() + 1, step=0.01))
     13     plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.
 ↪ravel()]).T).reshape(X1.shape),
---> 14                  alpha=0.75, cmap=ListedColormap(('red', 'green')))
     15     plt.xlim(X1.min(), X1.max())
     16     plt.ylim(X2.min(), X2.max())

NameError: name 'ListedColormap' is not defined
```

1 New section

[0]:
```
#2a. 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):
```

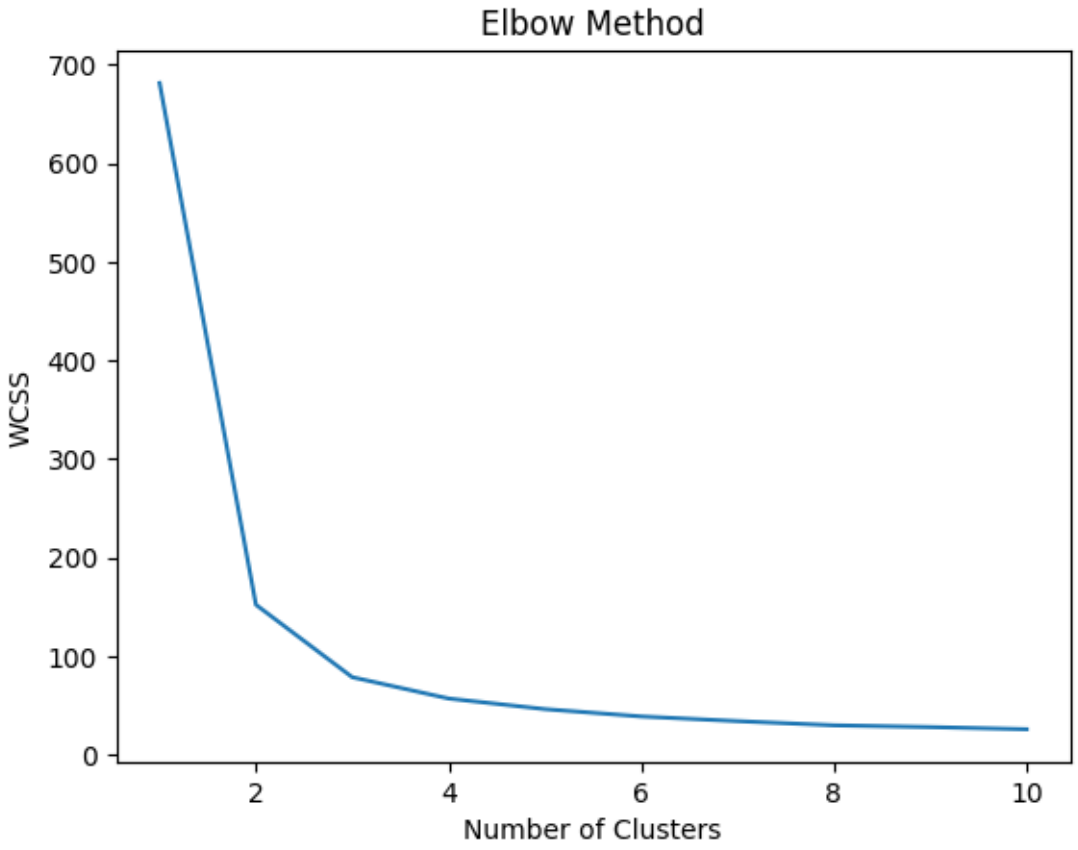
```
    print(i)
  i=i+1
```

```
Enter n value: 5
2
3
5
```

[0]:
```
#2b.K-Means Clustering

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()
```

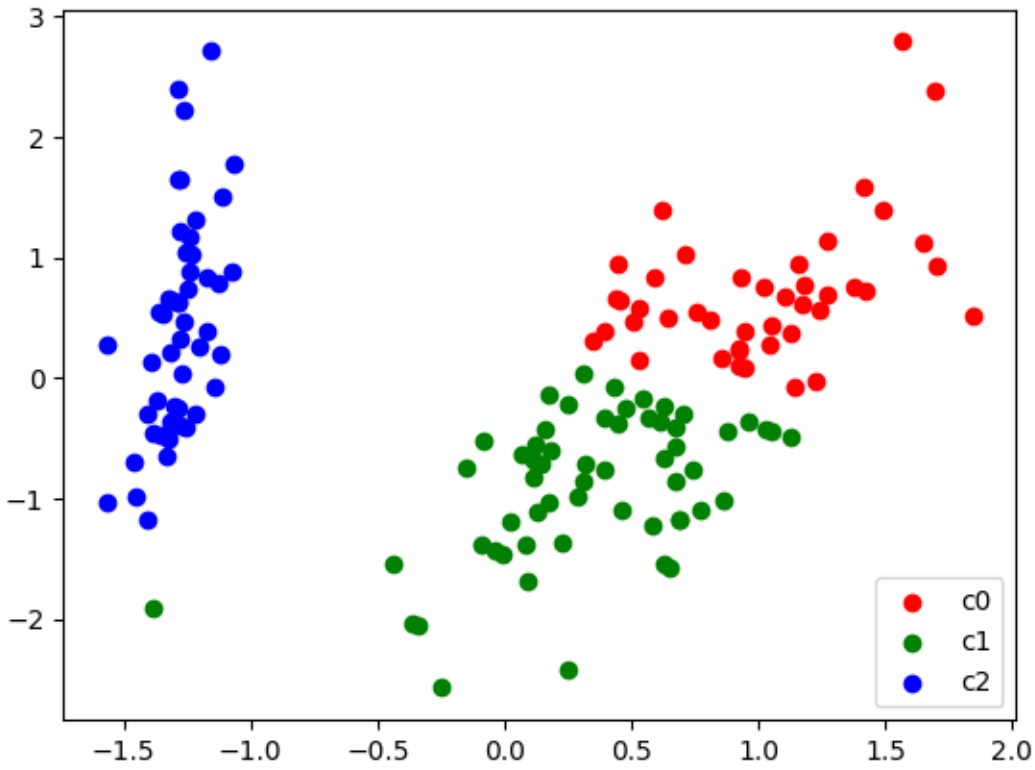
```
    pl.show()

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

[0]:




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[0]:



[0]:
```
#3a. Write a python program to find sum of n natural numbers using recursion␣
 ↪function
def sum1(n):
  if(n<0):
    print("Enter a valid number ")
  elif (n==0):
    return 0
  else:
    return n+sum1(n-1)

n=int(input("Enter a number "))
s=sum1(n)
print("Sum of first ",n," natural numbers is : ",s)
```

```
Enter a number 5
Sum of first  5  natural numbers is :  15
```

[0]:
```
#3b: C4.5  Decision Tree

# Note: Make sure that to install the GraphViz and set the path with C:\Program␣
 ↪Files (x86)\Graphviz2.38\bin
from sklearn.datasets import load_iris
```

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

# Splitting the dataset into the Training set and Test set
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)

#print((y_test!=predicted).sum(),'/',((y_test==predicted).sum()+(y_test!
 ↪=predicted).sum()))



import graphviz
'''dot_data = tree.export_graphviz(clf, out_file=None)

graph = graphviz.Source(dot_data)
graph.render("irisc45DT") #iris.pdf will be generated with decision tree
'''
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()
```

[0]: 'Source.gv.pdf'

[0]:
```
#4a. Write a python program to demonstrate Strings in Python
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(","))
```

```
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']
```

[0]:
```
#4b. CART (Classification and Regression Tree)

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()
```

[0]:
```
'Source.gv.pdf'
```

[0]:
```
#5a. Write a python program to demonstrate Python Lists
list=[]
print("Emplty 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(14)
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: ")
print(list2)
print("Accessing elements from the Multi-Dimensional list using index: ")
print(list2[0][0])
print(list2[1][0])
```

```
Emplty 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]
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, 4, 6, 27, 45,
73, 'Python']
list after poping an element(pop operation):  [0, 'MCA', 1, 4, 6, 27, 45, 73]
Print last element of the list:  73
Sliced list:  [1, 4, 6]
Multi-Dimensional list:
[['MCA', 'Mtech'], ['Python', 'C', 'Java']]
Accessing elements from the Multi-Dimensional list using index:
MCA
Python
```

[35]:
```
#5b. Write a Python program to implement K-Nearest Neighbors
```

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 22 22:22:51 2020

@author: Rams
"""

# K-Nearest Neighbors (K-NN)

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

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

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state = 0)

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

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

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

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
```

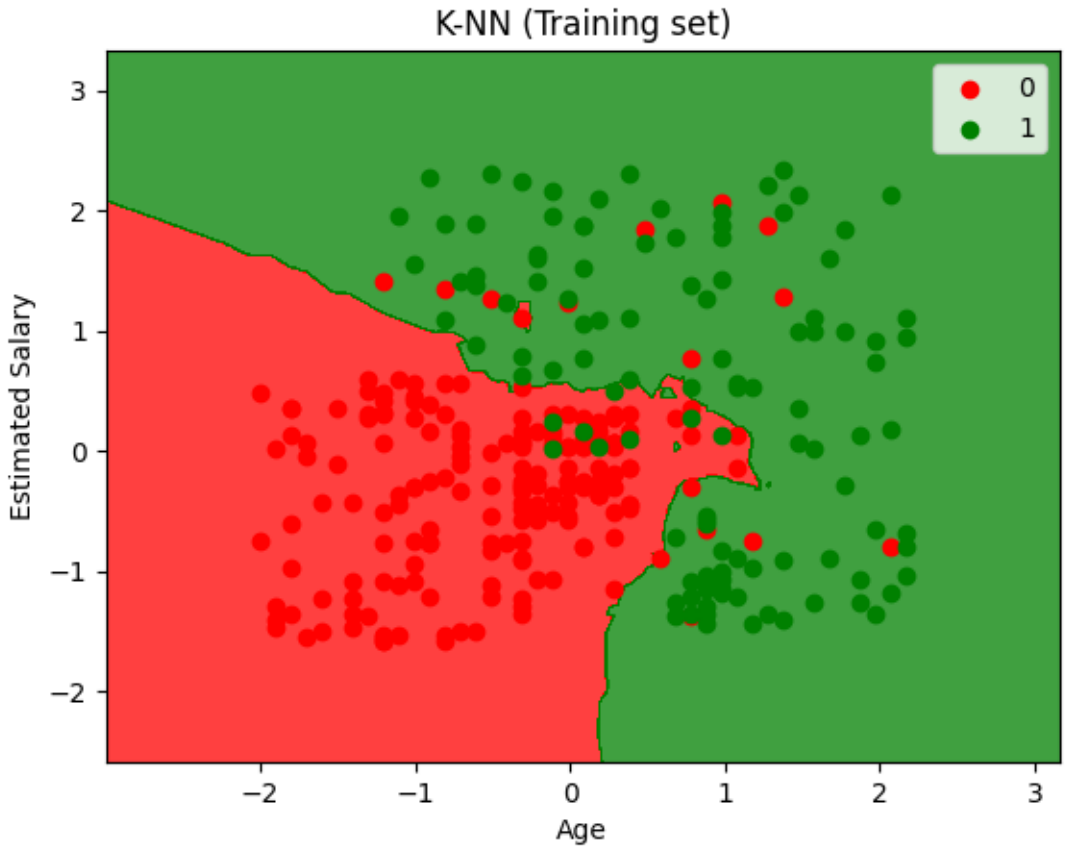
```
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
 ↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
 ↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
 ↪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()

# Visualising the Test set results
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)
```

```
Confusion Matrix of KNN
 [[64  4]
 [ 3 29]]
```

```
<ipython-input-35-c7d17738b659>:55: UserWarning: *c* argument looks like a
```

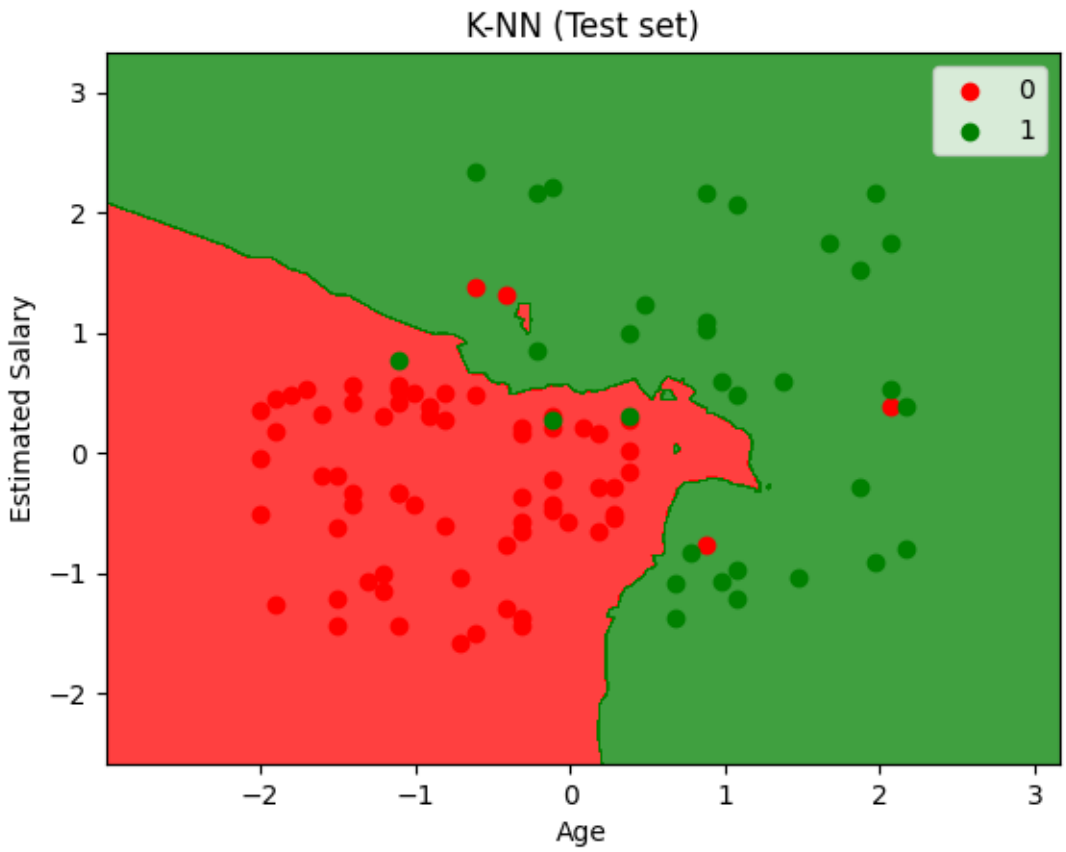
```
single numeric RGB or RGBA sequence, which should be avoided as value-mapping
will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend
to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

[35]:




```
<ipython-input-35-c7d17738b659>:73: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mapping
will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend
to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

[35]:




```
Accuracy Score is:  0.93
```

[0]:
```
#6a. Write a python demonstrate Python 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)
```

```
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')
```

[0]:
```
#6b. Support Vector Machines with Kernels
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 22 22:41:06 2020

@author: Rams
"""

# Kernel SVM

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

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

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state = 0)

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

# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
#classifier = SVC(kernel = 'linear', random_state = 0)
#from sklearn.svm import LinearSVC
#classifier = LinearSVC( random_state = 0)
```

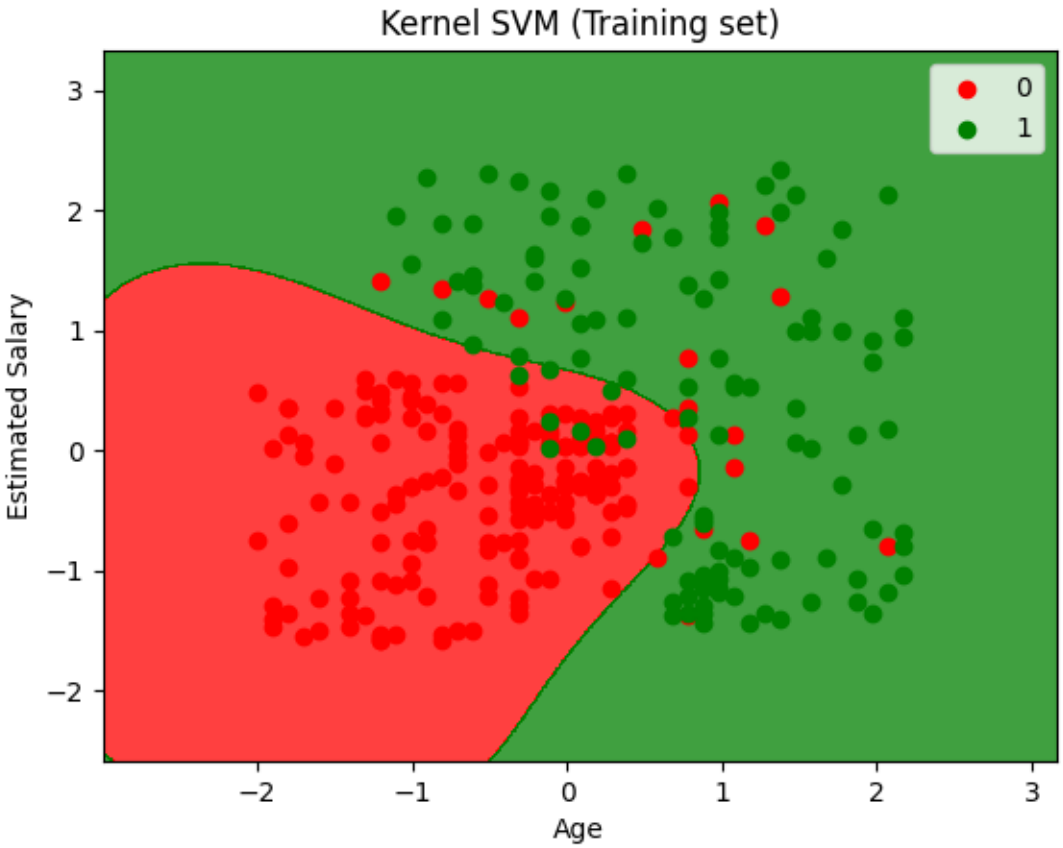
```
#classifier = SVC(kernel = 'poly', random_state = 0)

classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

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

# Visualising the Training set results
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()

# Visualising the Test set results
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)
```

```
Confusion Matrix of SVM
 [[64  4]
 [ 3 29]]
```

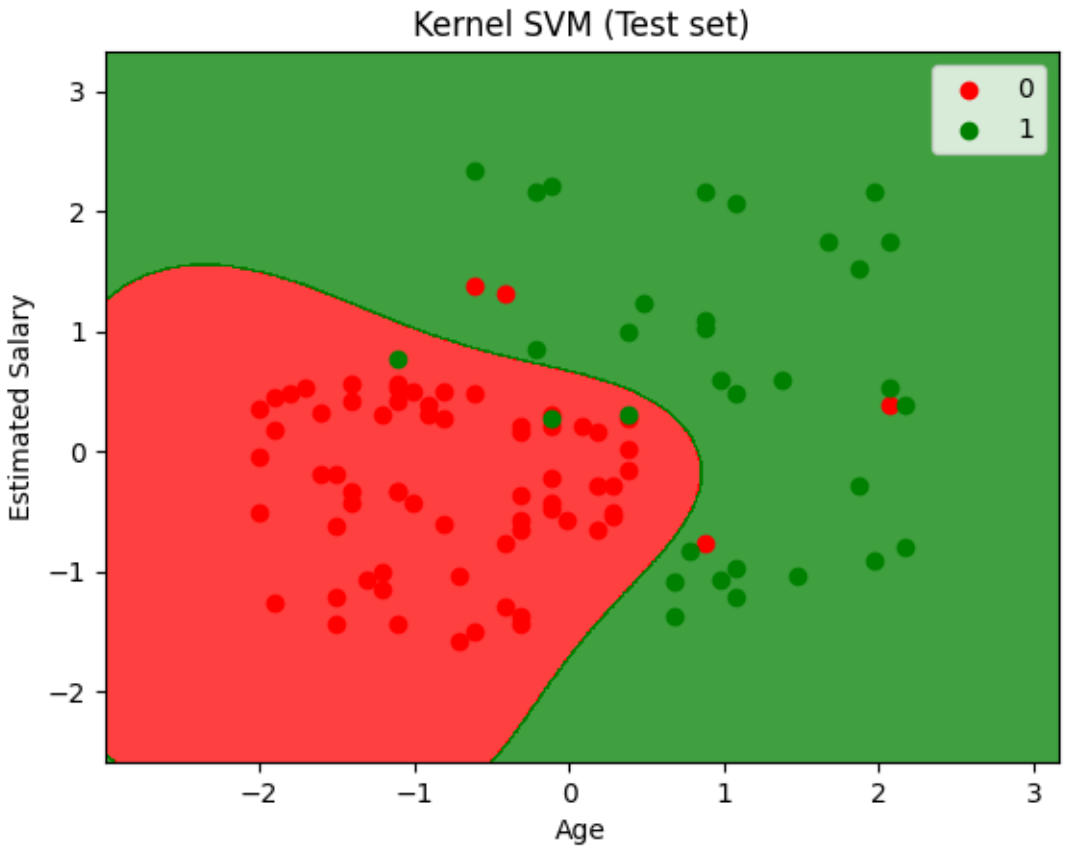
```
<ipython-input-26-dbd6acb266f7>:60: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mapping
will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend
to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

[0]:



```
<ipython-input-26-dbd6acb266f7>:78: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mapping
will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend
to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

[0]:




```
Accuracy Score is:  0.93
```

[0]:

```
#7a. Write a python demonstrate Python 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)
```

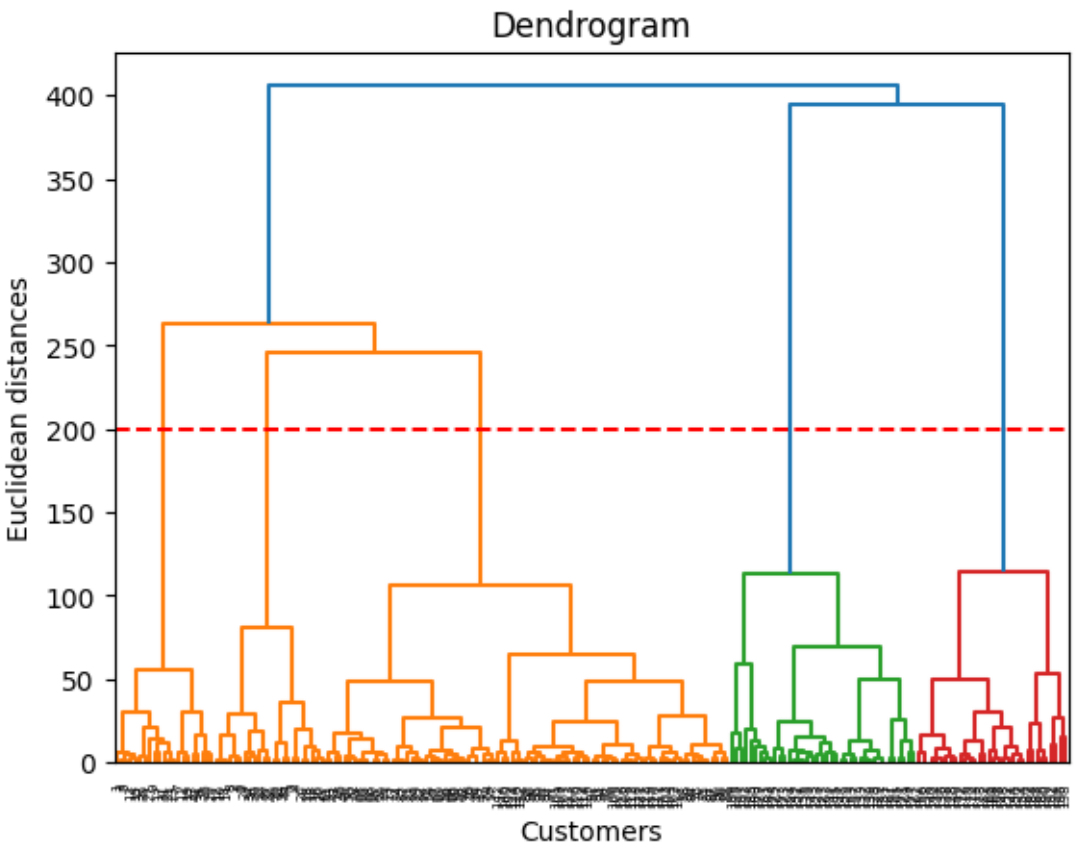
```
print("Acessing an element using key: ",dict[2])
print("Acessinga element using get method: ",dict.get(3))
del dict[4]
print("Dictionary after deleting a specific key(del operation): ",dict)
dict.pop(3)
print("Dictionary after deleting a specific element(pop operation): ",dict)
dict.clear()
print("Deleting 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)
```

```
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'}
Dictionary after deleting a specific element(pop operation):  {1: 'Machine
Learning', 2: 'Artificial Neural Network'}
Deleting the entire Dictionary:  {}
Nested Dictionary:  {1: 'CBIT', 2: 'MCA', 3: {'A': 'Machine Learning', 'B':
'Artificial Neural Network', 'C': 'Cloud Computing', 'D': 'IOT'}}
```

[0]:
```
#7b. Hierarchical Clustering

# Importing the libraries

import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values

# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.axhline(y=200, color='r', linestyle='--')
plt.show()
```

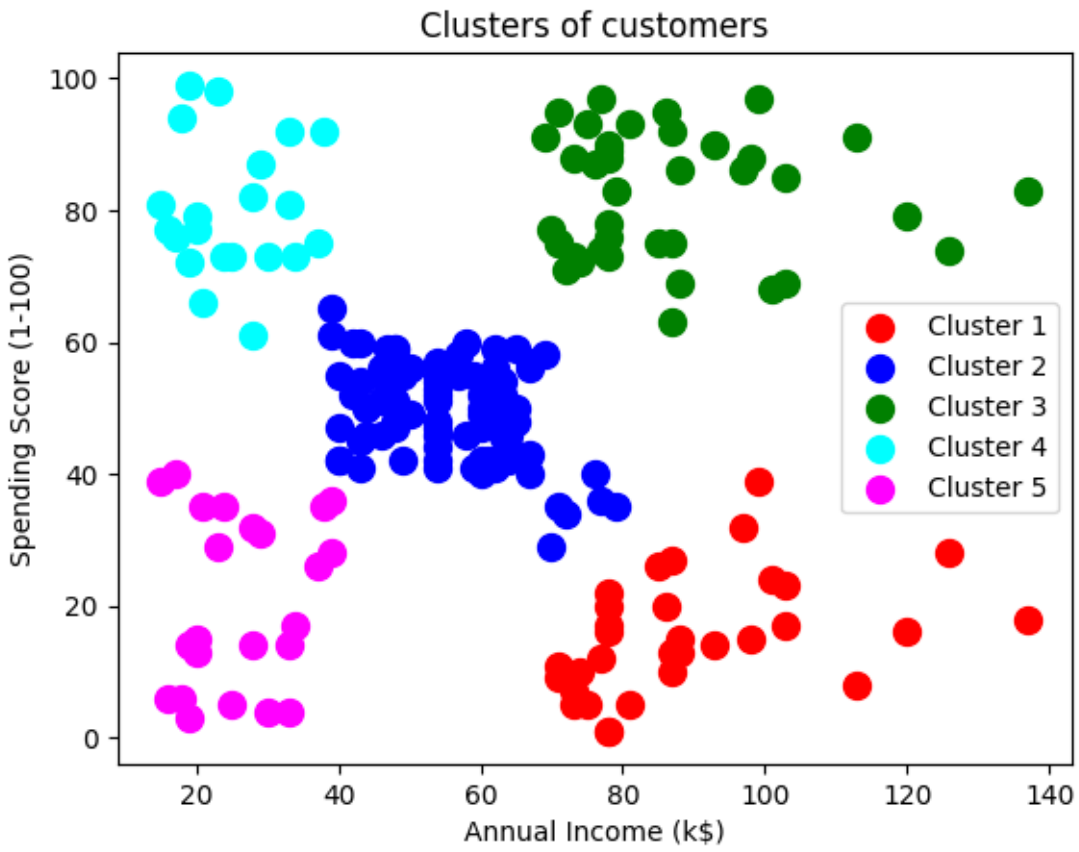
```
# Training the Hierarchical Clustering model on the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage =␣
 ↪'ward')
y_hc = hc.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label =␣
 ↪'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label =␣
 ↪'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label =␣
 ↪'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label =␣
 ↪'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label =␣
 ↪'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

[0]:




```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983:
FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be
removed in 1.4. Use `metric` instead
  warnings.warn(
```

[0]:



[0]:
```
#8a. 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: "))
print("Factorial of the given number is: ",fact(n))
```

```
Enter a number: 5
Factorial of the given number is:  120
```

[0]:
```
#9a. Write a python program to find the factorial of a given number using␣
 ↪recursive functions
def recfact(n):
  if n==0:
    return 1
```

```
  elif n==1:
    return 1
  else:
    return n*recfact(n-1)
n=int(input("Enter a number: "))
print("Factorial of the given number is: ",recfact(n))
```

```
Enter a number: 5
Factorial of the given number is:  120
```

[0]:
```
#9b. Logistic Regression
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 22 19:22:51 2020

@author: Rams
"""

# Logistic Regression

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

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

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,␣
 ↪random_state = 0)

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

# 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 results
```

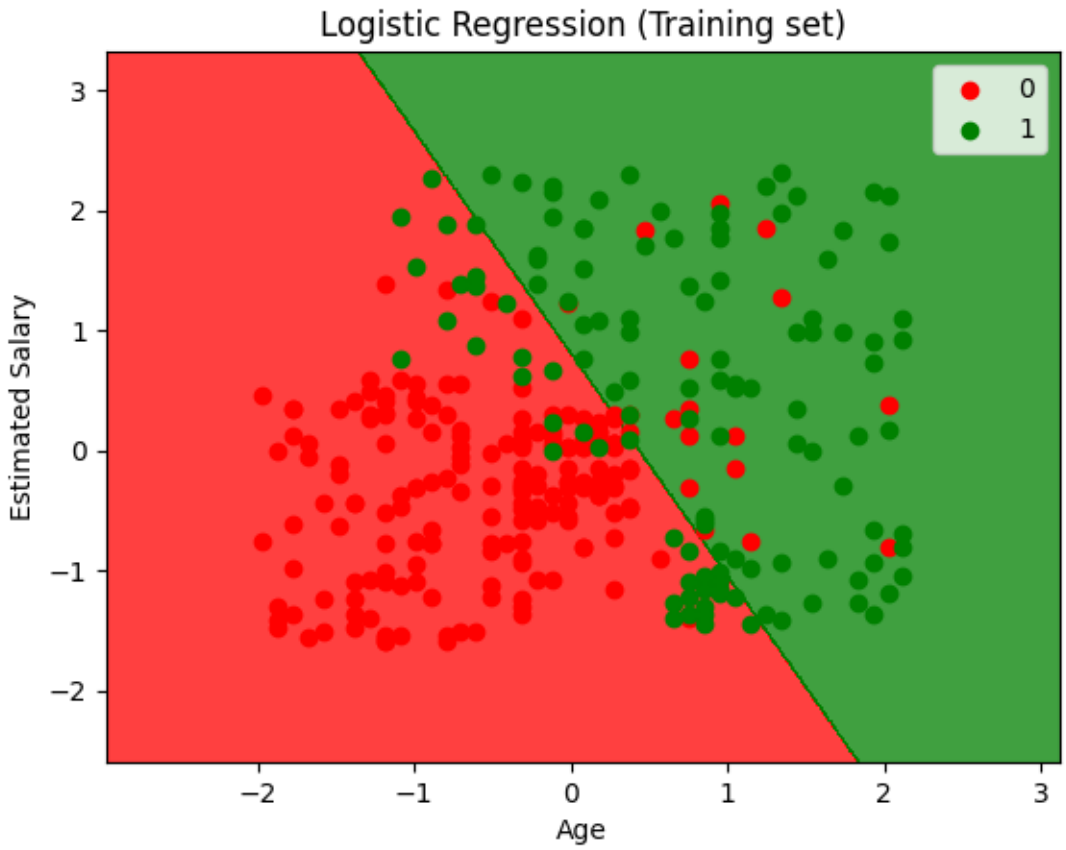
```
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Visualising the Training set results
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()

# Visualising the Test set results
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)
```

```
[[57  1]
 [ 5 17]]
```

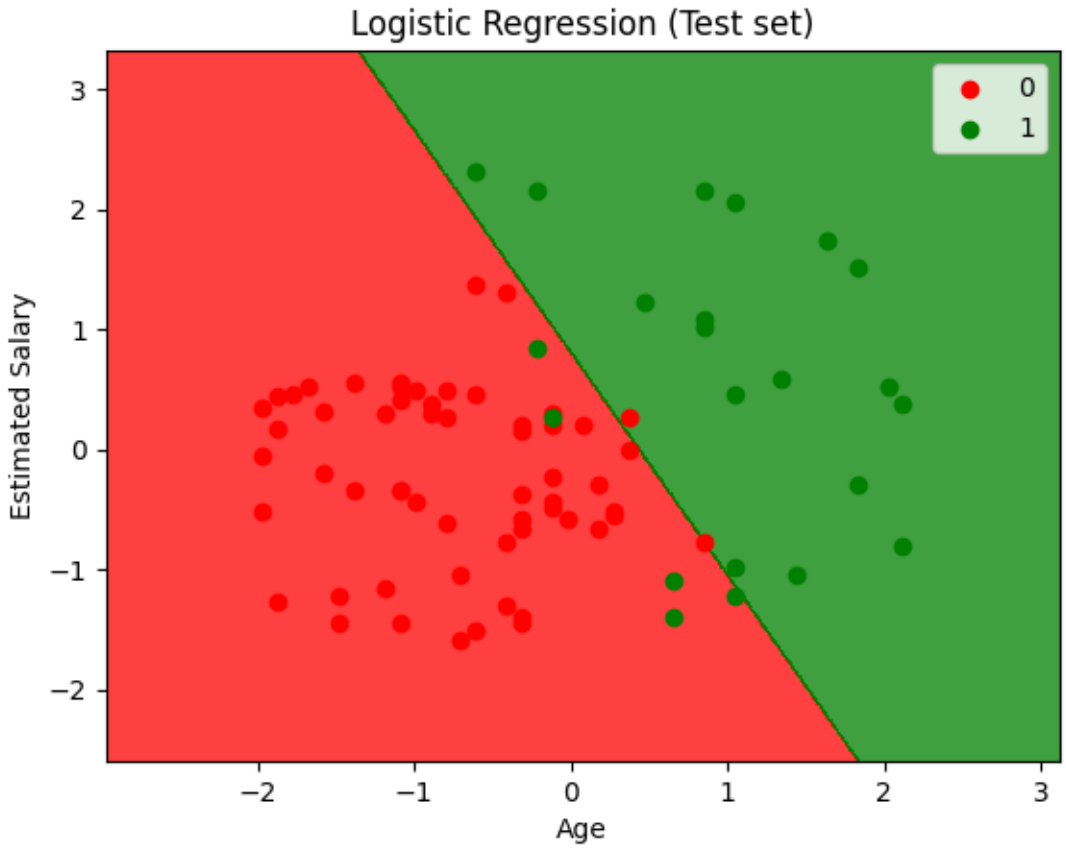
```
<ipython-input-30-d350b70b20fe>:54: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mapping
will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend
to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

[0]:




```
<ipython-input-30-d350b70b20fe>:72: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mapping
will have precedence in case its length matches with *x* & *y*.  Please use the
*color* keyword-argument or provide a 2D array with a single row if you intend
to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

[0]:




```
Accuracy Score is:  0.925
```

[0]:
```
#10a. Write a python program to find the gcd of a given number using recursive␣
 ↪functions
def gcd(a,b):
  if(a==b):
    return a
  elif(a>b):
    return gcd(a-b,b)
  else:
    return gcd(a,b-a)
a=int(input("Enter a number : "))
b=int(input("Enter a number : "))
gcd=gcd(a,b)
print("GCD of given two numbers is : ",gcd)
```

```
Enter a number : 5
Enter a number : 5
GCD of given two numbers is :  5
```

[0]:
```
#10b. Write a simple python program on Simple Linear Regression
# Step 1
import pandas as pd
import matplotlib.pyplot as plt

# Step 2 - Load the dataset
df_train = pd.read_csv('SalaryData_Train.csv')

# Step 3 Displays top 5 rows of Data Frame
print(df_train.head())

# Step 4 - Feature Extraction

yoe = df_train.iloc[:,0].values
sal = df_train.iloc[:,1].values
#print(feature)
#print(labels)

# Step 5 - Sample

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)
#random_state = n will be useful to get a fixed generated train and test for
# multiple runs otherwise it generates diff data as train and test

# Step 6 - Create the Linear Regression Model

from sklearn.linear_model import LinearRegression

reg = LinearRegression()

# Step 7 - fit the regression (reg) model that we have prepared with train and␣
 ↪test dataset in the sampling step

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

#Step 8 - Plot the trained data along with prediction
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()
```

```
#Step 9 - Find the accuracy of train and test data

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)))


# Step 10 - Testing the trained data on another data set to predict salary for␣
 ↪given years of exp

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
#Final Result with predicted Salaries
print(df_test)
```

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

[0]:


```
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
```