1. Write a Python 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))
```

3. 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
```

5. Write a Python program using the recursion function to find the sum of n natural numbers.

```
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)
```

7. 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(","))
```

9. Write a Python program to demonstrate Python 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])
```

11. 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)
```

13. 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.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)
```

15. 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))
```

16. 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))
```

18. 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}")
```

20. 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}")
```

19. Write a program to implement 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)
```

17. Write a program to implement 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)
```

6. Write a Python program to draw a decision Tree using C4.5 algorithm for real time data.

```
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()
```

8. Write a Python program to draw a decision Tree using CART algorithm for real time data.

```
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()
```

12. Write a Python program to implement Support Vector Machine 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)
```

2.Write a Python 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)')
```

4. Write a Python program to implement K-Means 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()
```

10. Write a Python program to implement K-Nearest Neighbors.

```
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)
```

14. Write a Python program to implement Hierarchical 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()
```