MACHINE LEARNING

CA2

NAME: WEQAR FATIMA SAYED ABEDI

UID: 18BIT046

CLASS: TYBSC-INFORMATION TECHNOLOGY

**Write a python code for following models on ‘Crimes’ dataset**

1) Perform data visualization:

|  |
| --- |
| **1.1) Display bar graph for the Total number crime for each day of week**  **# Importing Libraries**  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  **#Reading from the crimes.csv dataset**  boston\_crime=pd.read\_csv('crime.csv',encoding = 'unicode\_escape')  boston\_crime.shape  **#Displaying bar graph for the Total number crime for each day of week.**  robbery=boston\_crime[boston\_crime['OFFENSE\_CODE\_GROUP'].str.contains("Robbery|BURGLARY", na=False,case=False)].groupby(['DAY\_OF\_WEEK']).size().reset\_index(name='count')  sns.barplot(x = "DAY\_OF\_WEEK", y= "count", data=robbery)  plt.show() |
| **OUTPUT** |
| **1.2) Display bar graph for Total number crime for each hour**  **#Import Libraries**  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  **#Load the CSV file and Read the Data**  data = pd.read\_csv('crime.csv', encoding='latin-1')  print("DATA: \n",data.head())  print("Summing up the Values :\n",data.isnull().sum())  max\_hour\_crime = data['HOUR'].value\_counts().index[0]  #**Displays the Total Number of Crimes in Each Hour**  print('Hour with highest crime occurrence:', max\_hour\_crime)  **#Setting the Graph Features**  plt.subplots(figsize=(15,6))  sns.countplot('HOUR',  data=data,  palette='RdYlGn\_r',  edgecolor=sns.color\_palette('dark',7))  **#Setting the Title of the Graph**  plt.title('Number Of Crimes Each Hour')  **#Displaying bar graph for Total number crime for each hour**  plt.show() |
| **OUTPUT** |
| Grapgh For “**Number Of Crimes Each Hour**” |

**Perform k-mean clustering using Boston Housing Prices Dataset.**

2. Perform K means Clustering

|  |
| --- |
| **Using the Entire Boston Housing Dataset**  **#Importing Libraries**  import pandas as pd  from sklearn.datasets import load\_boston  from sklearn.preprocessing import MinMaxScaler  from sklearn.cluster import KMeans  import pylab as pl  **#Loading Dataset**  boston=load\_boston()  ds=pd.DataFrame(boston.data,columns=boston.feature\_names)  **#1-hot encoding of RAD variable; because its categorical variable**  ds["RAD"]=ds["RAD"].astype("category")  print("RAD:", ds)  **#automatically pick the categorical variable and apply 1-hot encoding**  ds=pd.get\_dummies(ds,prefix="RAD")  **#Scaling on entire Dataset**  scaler=MinMaxScaler()  scaler=scaler.fit(ds)  scaledData=scaler.transform(ds)  **#Creating Scaled dataframe**  dss=pd.DataFrame(scaledData,columns=ds.columns)  **#Performing the cluster configuration**  clust=KMeans(n\_clusters=4,max\_iter=500,random\_state=0).fit(dss)  clusterCenter=clust.cluster\_centers\_  ccd=pd.DataFrame(clusterCenter,columns=dss.columns)  labels=clust.labels\_  clusterIds=list(labels)  **#now perform the inverse Scaling**  originalDataAsNumpy=scaler.inverse\_transform(dss)  **#converting numpy to dataset**  originalDataset=pd.DataFrame(originalDataAsNumpy,columns=dss.columns)  **#adding the labelled column to the originalDataset**  originalDataset["Label"]=labels  originalDataset.Label[0]  **#Plotting the classification with different colors**  len=originalDataset.shape[0]  for i in range(0, len):  if originalDataset.Label[i] == 0:  c1 = pl.scatter(originalDataset.iloc[i,2],originalDataset.iloc[i,4],c='y', marker='+')  elif originalDataset.Label[i] == 1:  c2 = pl.scatter(originalDataset.iloc[i,2],originalDataset.iloc[i,4],c='r',marker='o')  elif originalDataset.Label[i] == 2:  c3 = pl.scatter(originalDataset.iloc[i,2],originalDataset.iloc[i,4],c='g',marker='\*')  elif originalDataset.Label[i] == 3:  c4 = pl.scatter(originalDataset.iloc[i,2],originalDataset.iloc[i,4],c='b',marker='^')  pl.legend([c1, c2, c3,c4], ['c1','c2','c3','c4'])  pl.title('Boston Data classification')  pl.show() |
| **OUTPUT** |
|  |

|  |
| --- |
| **Performing K means clustering on 1 Target and 1 Feature**  **#Import Libraries**  from sklearn.cluster import KMeans  from pandas import DataFrame  import matplotlib.pyplot as plt  **#Taking 2 columns one for Test data and one for Training data**  Data={  'x':[0.00632,0.02731,0.02729,0.03237,0.06905,0.02985,0.08829,0.14455,0.21124,0.17004,0.22489,0.11747,0.09378,0.62976,0.63796,0.62739,1.05393,0.7842,0.80271,0.7258,1.25179,0.85204,1.23247,0.98843,0.75026,0.84054,0.67191,0.95577,0.77299,1.00245  ],  'y':[24,21.6,34.7,33.4,36.2,28.7,22.9,27.1,16.5,18.9,15,18.9,21.7,20.4,18.2,19.9,23.1,17.5,20.2,18.2,13.6,19.6,15.2,14.5,15.6,13.9,16.6,14.8,18.4,21]  }  **#Giving the Columns values 'x' and 'y'**  df= DataFrame(Data, columns=['x','y'])  **#Performing K means and fitting the data with 3 Clustering**  k\_means = KMeans(n\_clusters=3).fit(df)  **#Printing the Centroid Values**  centroids= k\_means.cluster\_centers\_  print("CENTROIDS: \n", centroids)  #**Setting the features for the Graphs**  plt.scatter(df['x'],df['y'], c=k\_means.labels\_.astype(float), s=50,alpha=0.5)  plt.scatter(centroids[:,0], centroids[:,1], c='red', s=50)  plt.show() |
| **OUTPUT** |
|  |

3**. Perform decision tree regression.**

|  |
| --- |
| **CODE**  **#Importing Libraries**  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.tree import DecisionTreeRegressor  import matplotlib.pyplot as plt  from sklearn.tree import plot\_tree  from sklearn.tree import export\_graphviz  import pydotplus  **#Reading the Dataset**  dataset=pd.read\_csv('boston\_house\_dataset.csv')  print(dataset.shape)  **#Splitting the dataset**  x=pd.DataFrame(dataset.iloc[:,:-1])  y=pd.DataFrame(dataset.iloc[:,-1])  print(x)  print(y)  #**Splitting into training and test dataset**  X\_train,X\_test,y\_train, y\_test=train\_test\_split(x,y,random\_state=0, test\_size=0.20)  **#Apply the model on training data**  regression\_data=DecisionTreeRegressor(criterion='mse',  random\_state=100,  max\_depth=None,  min\_samples\_leaf=1)  regression\_data.fit(X\_train, y\_train)  **#Plotting the tree**  fig=plt.figure(figsize=(35,20))  a=plot\_tree(regression\_data,feature\_names=x.columns, class\_names=y,filled=True,rounded=True,fontsize=14)  fig.savefig("decision\_tree\_image.png") |
| OUTPUT |