****

****

**SAKET GYANPEETH’S**

**SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE**

**(Permanently Affiliated to University of Mumbai)**

**NAAC Accredited B Grade**

Saket Vidyanagari Marg, Chinchpada Road, Katemanivali, Kalyan (East) – 421306, Dist. Thane (MAH)

Department of Information Technology

**CERTIFICATE**

**This is to certify that**

**SEJAL JAIPRAKASH SINGH** of **MSc Information Technology Part-II** Class has satisfactory carried out the required practical in the subject.

**MACHINE LEARNING**

For the Academic year 2023 – 2024

**Practical In-Charge Head of Department External Examiner**

**INDEX:**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Content** | **Page No.** |
| **1** | 1A. Design a simple machine learning model to train the training instances and test the same.  1B. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file. | 4-8 |
| **2** | 2A. Perform Data Loading, Feature selection (Principal Component Analysis) and Feature Scoring and Ranking  2B. For a given set of training data example stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples. | 9-13 |
| **3** | 3A. Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.  3B. Write a program to implement Decision Tree and Random Forest with Prediction, Test Score and Confusion Matrix. | 14-17 |
| **4** | 4A. For a given set of training data example stored in a .CSV file implement Least Square Regression Algorithm.  4B. For a given set of training data examples stored in a .CSV file implement Logistic Regression Algorithm. | 18-23 |
| **5** | Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. | 24-27 |
| **6** | Implement the classification model using clustering for the following techniques with K means clustering with prediction, Test Score and Confusion Matrix. | 28-34 |
| **7** | Implement the Rule Based method and test the same. | 35-39 |
| **8** | 8A. Write a program to construct a Bayesian Network considering medical data. Use this model to demonstrate the diagnosis of heart patient using standard Heart Disease Data Set. | 40-43 |

**PRACTICAL:1**

**1A. Design a simple machine learning model to train the training instances and test the same.**

import numpy

import matplotlib.pyplot as plt

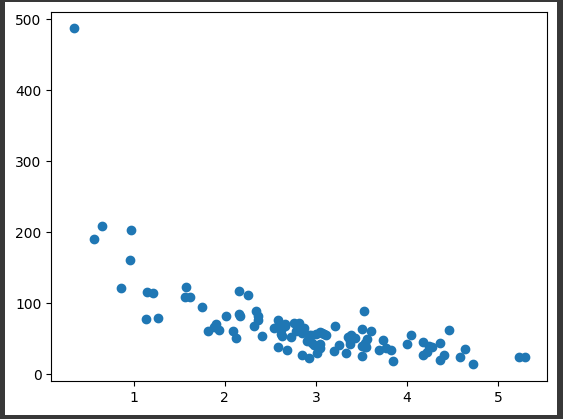
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)

y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)

plt.show()



train\_x = x[:80]

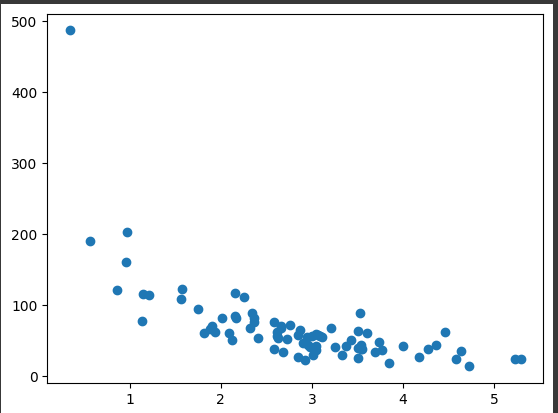
train\_y = y[:80]

test\_x = x[80:]

test\_y = y[80:]

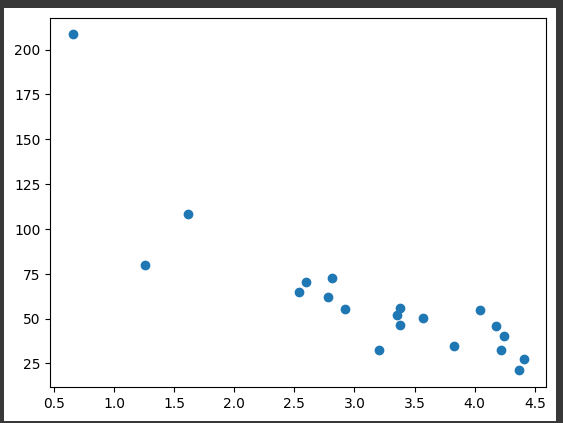
plt.scatter(train\_x, train\_y)

plt.show()



plt.scatter(test\_x, test\_y)

plt.show()



import numpy

import matplotlib.pyplot as plt

numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)

y = numpy.random.normal(150, 40, 100) / x

train\_x = x[:80]

train\_y = y[:80]

test\_x = x[80:]

test\_y = y[80:]

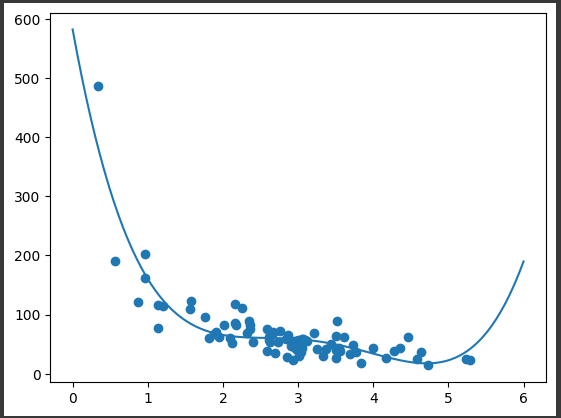
mymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))

myline = numpy.linspace(0, 6, 100)

plt.scatter(train\_x, train\_y)

plt.plot(myline, mymodel(myline))

plt.show()



#USING R2

import numpy

from sklearn.metrics import r2\_score

numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)

y = numpy.random.normal(150, 40, 100) / x

train\_x = x[:80]

train\_y = y[:80]

test\_x = x[80:]

test\_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))

r2 = r2\_score(train\_y, mymodel(train\_x))

print(r2)

**OUTPUT**: 0.79886455446298

import numpy

from sklearn.metrics import r2\_score

numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)

y = numpy.random.normal(150, 40, 100) / x

train\_x = x[:80]

train\_y = y[:80]

test\_x = x[80:]

test\_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train\_x, train\_y, 4))

r2 = r2\_score(test\_y, mymodel(test\_x))

print(r2)

**OUTPUT**: 0.8086921460343566

**1B. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from**

**a .CSV file**

import csv

hypo = ['%','%','%','%','%','%'];

with open('/trainingdata.csv') as csv\_file:

    readcsv = csv.reader(csv\_file, delimiter=',')

    print(readcsv)

    data = []

    print("\nThe given training examples are:")

    for row in readcsv:

        print(row)

        if row[len(row)-1].upper() == "YES":

            data.append(row)

**OUTPUT:**

<\_csv.reader object at 0x796a0f73b060>The given training examples are:

['sky','airTemp', 'humidity', 'wind', 'water', 'forecast', 'enjoySport']

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

print("\nThe positive examples are:");

for x in data:

    print(x);

print("\n");

**OUTPUT:**

The positive examples are:

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

TotalExamples = len(data);

i=0;

j=0;

k=0;

print("The steps of the Find-s algorithm are :\n",hypo);

list = [];

p=0;

d=len(data[p])-1;

for j in range(d):

    list.append(data[i][j]);

hypo=list;

i=1;

for i in range(TotalExamples):

    for k in range(d):

        if hypo[k]!=data[i][k]:

            hypo[k]='?';

            k=k+1;

        else:

            hypo[k];

    print(hypo);

i=i+1;

**OUTPUT:**

The steps of the Find-s algorithm are :

['%', '%', '%', '%', '%', '%']

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

['Sunny', 'Warm', '?', 'Strong', '?', '?']

print("\nThe maximally specific Find-s hypothesis for the given training examples is :");

list=[];

for i in range(d):

    list.append(hypo[i]);

print(list);

**OUTPUT:**

The maximally specific Find-s hypothesis for the given training examples is :

['Sunny', 'Warm', '?', 'Strong', '?', '?']

**PRACTICAL 2**

**2A. Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.**

import numpy as np

import pandas as pd

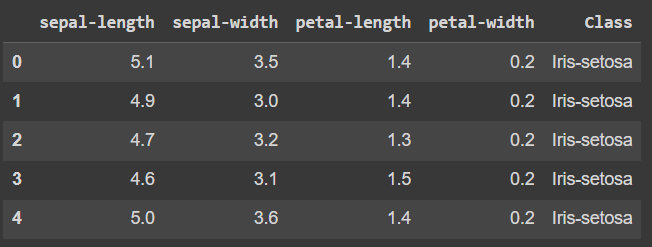
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

dataset = pd.read\_csv(url, names=names)

dataset.head()

**OUTPUT:**

****

X = dataset.drop('Class', 1)

y = dataset['Class']

**OUTPUT:**

****

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.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.decomposition import PCA

pca = PCA()

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

explained\_variance = pca.explained\_variance\_ratio

from sklearn.decomposition import PCA

pca = PCA(n\_components=1)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(max\_depth=2, random\_state=0)

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

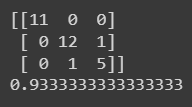
from sklearn.metrics import accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

print( accuracy\_score(y\_test, y\_pred))

**OUTPUT:**

****

from sklearn.decomposition import PCA

pca = PCA(n\_components=1)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

print()

**OUTPUT:**

<class 'sklearn.decomposition.\_pca.PCA'>

**2B. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

import numpy as np

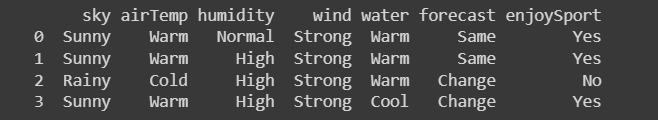
import pandas as pd

# Loading Data from a CSV File

data = pd.DataFrame(data=pd.read\_csv('/content/train (1).csv'))

print(data)

**OUTPUT:**

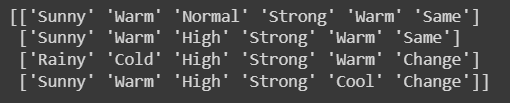


# Separating concept features from Target

concepts = np.array(data.iloc[:,0:-1])

print(concepts)

**OUTPUT:**



# Isolating target into a separate DataFrame

# copying last column to target array

target = np.array(data.iloc[:,-1])

print(target)

**OUTPUT:**



def learn(concepts, target):

    # Initialise S0 with the first instance from concepts

    # .copy() makes sure a new list is created instead of just pointing to the same memory location

    specific\_h = concepts[0].copy()

    print("\nInitialization of specific\_h and general\_h")

    print(specific\_h)

    #h=["#" for i in range(0,5)]

    #print(h)

    general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

    print(general\_h)

    # The learning iterations

    for i, h in enumerate(concepts):

        # Checking if the hypothesis has a positive target

        if target[i] == "Yes":

            for x in range(len(specific\_h)):

                # Change values in S & G only if values change

                if h[x] != specific\_h[x]:

                    specific\_h[x] = '?'

                    general\_h[x][x] = '?'

        # Checking if the hypothesis has a positive target

        if target[i] == "No":

            for x in range(len(specific\_h)):

                # For negative hyposthesis change values only  in G

                if h[x] != specific\_h[x]:

                    general\_h[x][x] = specific\_h[x]

                else:

                    general\_h[x][x] = '?'

        print("\nSteps of Candidate Elimination Algorithm",i+1)

        print(specific\_h)

        print(general\_h)

    # find indices where we have empty rows, meaning those that are unchanged

    indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

    for i in indices:

        # remove those rows from general\_h

        general\_h.remove(['?', '?', '?', '?', '?', '?'])

    # Return final values

    return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("\nFinal Specific\_h:", s\_final, sep="\n")

print("\nFinal General\_h:", g\_final, sep="\n")

**OUTPUT:**

Initialization of specific\_h and general\_h

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 1

['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3

['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Steps of Candidate Elimination Algorithm 4

['Sunny' 'Warm' '?' 'Strong' '?' '?']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General\_h:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

**PRACTICAL 3**

**3A. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

iris = datasets.load\_iris()

X = iris.data[:, :2]

y = iris.target

C = 1.0 # SVM regularization parameter

svc = svm.SVC(kernel='linear', C=1,gamma=0).fit(X, y)

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

h = (x\_max / x\_min)/100

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

 np.arange(y\_min, y\_max, h))

plt.subplot(1, 1, 1)

Z = svc.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)

plt.xlabel('Sepal length')

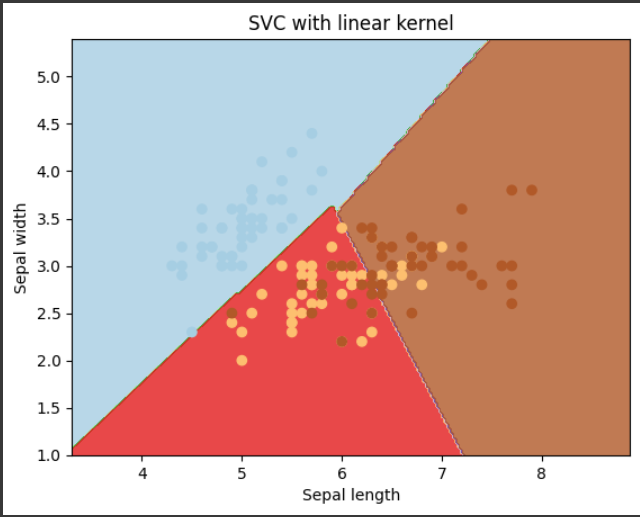
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

plt.title('SVC with linear kernel')

plt.show(

**OUTPUT:**

****

**3B. Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix**

import numpy as np

import math

import csv

def read\_data(filename):

    with open(filename, 'r') as csvfile:

        datareader = csv.reader(csvfile, delimiter=',')

        headers = next(datareader)

        metadata = []

        traindata = []

        for name in headers:

            metadata.append(name)

        for row in datareader:

            traindata.append(row)

    return (metadata, traindata)

class Node:

    def \_\_init\_\_(self, attribute):

        self.attribute = attribute

        self.children = []

        self.answer = ""

    def \_\_str\_\_(self):

        return self.attribute

def subtables(data, col, delete):

    dict = {}

    items = np.unique(data[:, col])

    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):

        for y in range(data.shape[0]):

            if data[y, col] == items[x]:

                count[x] += 1

    for x in range(items.shape[0]):

        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

        pos = 0

        for y in range(data.shape[0]):

            if data[y, col] == items[x]:

                dict[items[x]][pos] = data[y]

                pos += 1

        if delete:

            dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict

def entropy(S):

    items = np.unique(S)

    if items.size == 1:

        return 0

    counts = np.zeros((items.shape[0], 1))

    sums = 0

    for x in range(items.shape[0]):

        counts[x] = sum(S == items[x]) / (S.size \* 1.0)

    for count in counts:

        sums += -1 \* count \* math.log(count, 2)

    return sums

def gain\_ratio(data, col):

    items, dict = subtables(data, col, delete=False)

    total\_size = data.shape[0]

    entropies = np.zeros((items.shape[0], 1))

    intrinsic = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):

        ratio = dict[items[x]].shape[0]/(total\_size \* 1.0)

        entropies[x] = ratio \* entropy(dict[items[x]][:, -1])

        intrinsic[x] = ratio \* math.log(ratio, 2)

    total\_entropy = entropy(data[:, -1])

    iv = -1 \* sum(intrinsic)

    for x in range(entropies.shape[0]):

        total\_entropy -= entropies[x]

    return total\_entropy / iv

def create\_node(data, metadata):

    if (np.unique(data[:, -1])).shape[0] == 1:

        node = Node("")

        node.answer = np.unique(data[:, -1])[0]

        return node

    gains = np.zeros((data.shape[1] - 1, 1))

    for col in range(data.shape[1] - 1):

        gains[col] = gain\_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])

    metadata = np.delete(metadata, split, 0)

    items, dict = subtables(data, split, delete=True)

    for x in range(items.shape[0]):

        child = create\_node(dict[items[x]], metadata)

        node.children.append((items[x], child))

    return node

def empty(size):

    s = ""

    for x in range(size):

        s += "   "

    return s

def print\_tree(node, level):

    if node.answer != "":

        print(empty(level), node.answer)

        return

    print(empty(level), node.attribute)

    for value, n in node.children:

        print(empty(level + 1), value)

        print\_tree(n, level + 2)

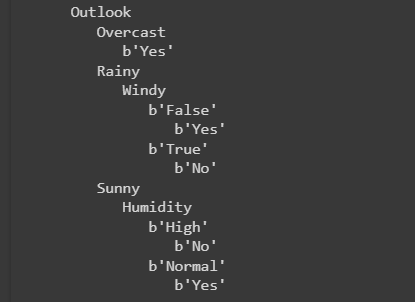
metadata, traindata = read\_data("/content/tennisdata.csv")

data = np.array(traindata)

node = create\_node(data, metadata)

print\_tree(node, 0)

**OUTPUT:**



**PRACTICAL 4**

**4A. For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.**

import datetime

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

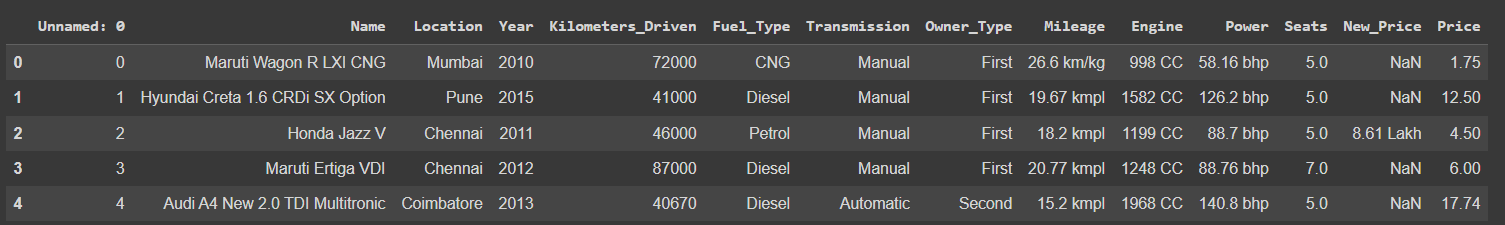
from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import r2\_score

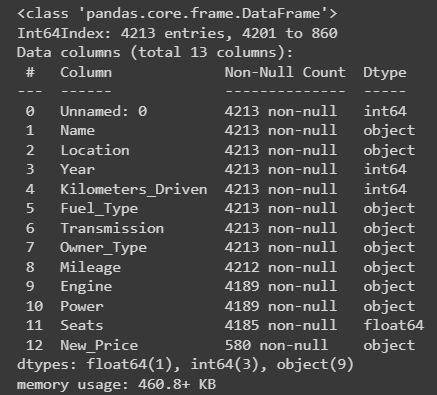
dataset = pd.read\_csv("/content/dataset.csv")

dataset.head(5)



X\_train, X\_test, y\_train, y\_test = train\_test\_split(dataset.iloc[:, :-1],dataset.iloc[:, -1],test\_size = 0.3,random\_state = 42)

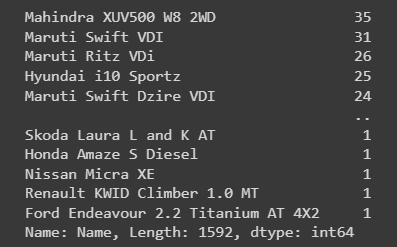
X\_train.info()



X\_train = X\_train.iloc[:, 1:]

X\_test = X\_test.iloc[:, 1:]

X\_train["Name"].value\_counts()



make\_train = X\_train["Name"].str.split(" ", expand = True)

make\_test = X\_test["Name"].str.split(" ", expand = True)

X\_train["Manufacturer"] = make\_train[0]

X\_test["Manufacturer"] = make\_test[0]

plt.figure(figsize = (12, 8))

plot = sns.countplot(x = 'Manufacturer', data = X\_train)

plt.xticks(rotation = 90)

for p in plot.patches:

    plot.annotate(p.get\_height(),

                        (p.get\_x() + p.get\_width() / 2.0,

                         p.get\_height()),

                        ha = 'center',

                        va = 'center',

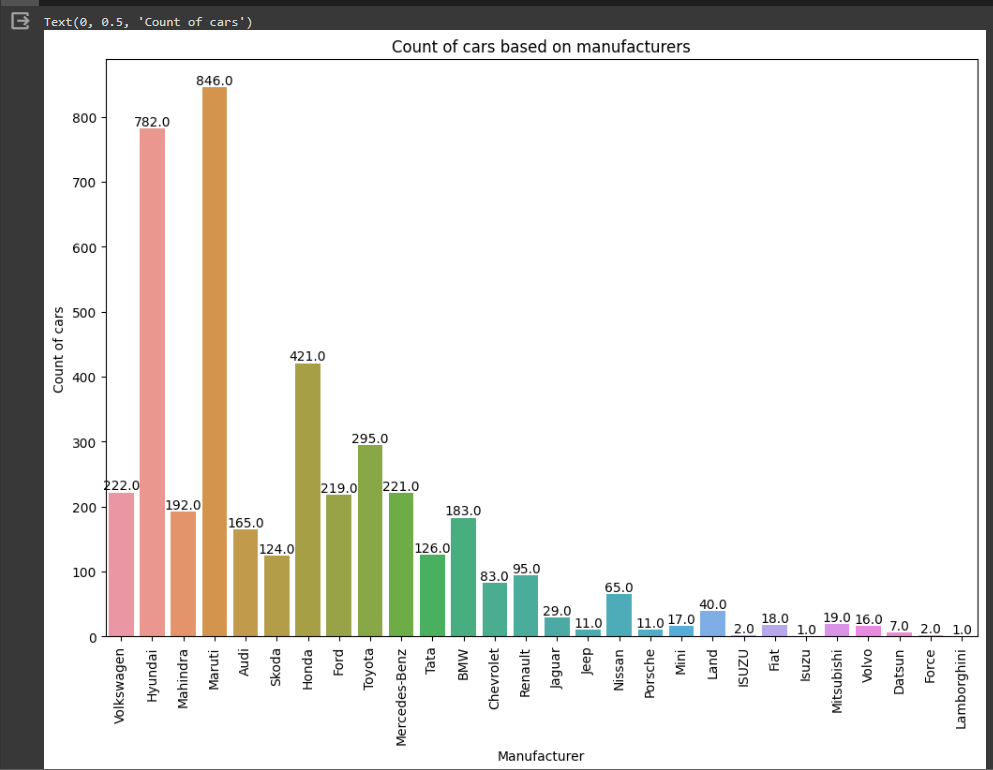
                        xytext = (0, 5),

                        textcoords = 'offset points')

plt.title("Count of cars based on manufacturers")

plt.xlabel("Manufacturer")

plt.ylabel("Count of cars")



X\_train.drop("Name", axis = 1, inplace = True)

X\_test.drop("Name", axis = 1, inplace = True)

X\_train.drop("Location", axis = 1, inplace = True)

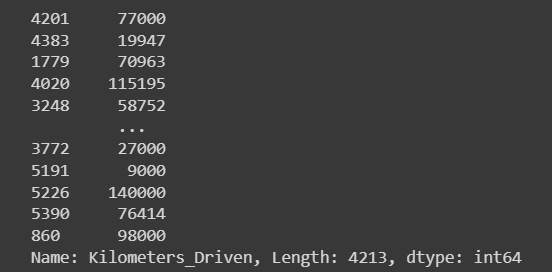
X\_test.drop("Location", axis = 1, inplace = True)

curr\_time = datetime.datetime.now()

X\_train['Year'] = X\_train['Year'].apply(lambda x : curr\_time.year - x)

X\_test['Year'] = X\_test['Year'].apply(lambda x : curr\_time.year - x)

X\_train["Kilometers\_Driven"]



mileage\_train = X\_train["Mileage"].str.split(" ", expand = True)

mileage\_test = X\_test["Mileage"].str.split(" ", expand = True)

X\_train["Mileage"] = pd.to\_numeric(mileage\_train[0], errors = 'coerce')

X\_test["Mileage"] = pd.to\_numeric(mileage\_test[0], errors = 'coerce')

print(sum(X\_train["Mileage"].isnull()))

print(sum(X\_test["Mileage"].isnull()))



X\_train["Mileage"].fillna(X\_train["Mileage"].astype("float64").mean(), inplace = True)

X\_test["Mileage"].fillna(X\_train["Mileage"].astype("float64").mean(), inplace = True)

cc\_train = X\_train["Engine"].str.split(" ", expand = True)

cc\_test = X\_test["Engine"].str.split(" ", expand = True)

X\_train["Engine"] = pd.to\_numeric(cc\_train[0], errors = 'coerce')

X\_test["Engine"] = pd.to\_numeric(cc\_test[0], errors = 'coerce')

bhp\_train = X\_train["Power"].str.split(" ", expand = True)

bhp\_test = X\_test["Power"].str.split(" ", expand = True)

X\_train["Power"] = pd.to\_numeric(bhp\_train[0], errors = 'coerce')

X\_test["Power"] = pd.to\_numeric(bhp\_test[0], errors = 'coerce')

X\_train["Engine"].fillna(X\_train["Engine"].astype("float64").mean(), inplace = True)

X\_test["Engine"].fillna(X\_train["Engine"].astype("float64").mean(), inplace = True)

X\_train["Power"].fillna(X\_train["Power"].astype("float64").mean(), inplace = True)

X\_test["Power"].fillna(X\_train["Power"].astype("float64").mean(), inplace = True)

X\_train["Seats"].fillna(X\_train["Seats"].astype("float64").mean(), inplace = True)

X\_test["Seats"].fillna(X\_train["Seats"].astype("float64").mean(), inplace = True)

X\_train.drop(["New\_Price"], axis = 1, inplace = True)

X\_test.drop(["New\_Price"], axis = 1, inplace = True)

X\_train = pd.get\_dummies(X\_train,columns = ["Manufacturer", "Fuel\_Type", "Transmission", "Owner\_Type"],drop\_first = True)

X\_test = pd.get\_dummies(X\_test, columns = ["Manufacturer", "Fuel\_Type", "Transmission", "Owner\_Type"],drop\_first = True)

missing\_cols = set(X\_train.columns) - set(X\_test.columns)

for col in missing\_cols:

    X\_test[col] = 0

X\_test = X\_test[X\_train.columns]

standardScaler = StandardScaler()

standardScaler.fit(X\_train)

X\_train = standardScaler.transform(X\_train)

X\_test = standardScaler.transform(X\_test)

linearRegression = LinearRegression()

linearRegression.fit(X\_train, y\_train)

y\_pred = linearRegression.predict(X\_test)

r2\_score(y\_test, y\_pred)



rf = RandomForestRegressor(n\_estimators = 100)

rf.fit(X\_train, y\_train)

y\_pred = rf.predict(X\_test)

r2\_score(y\_test, y\_pred)



**4B. For a given set of training data examples stored in a .CSV file implement Logistic Regression Algorithm.**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the datasets

datasets = pd.read\_csv('/content/Social\_Network\_Ads.csv')

X = datasets.iloc[:, [2,3]].values

Y = datasets.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\_X = StandardScaler()

X\_Train = sc\_X.fit\_transform(X\_Train)

X\_Test = sc\_X.transform(X\_Test)

# Fitting the Logistic Regression into 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)

# 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(), X2.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()

<ipython-input-1-3fd06d73ad23>:51: 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]



**PRACTICAL:5**

**Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

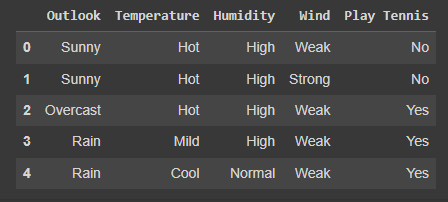
import pandas as pd

import numpy as np

train\_data\_m = pd.read\_csv("PlayTennis.csv")

test\_data\_m = pd.read\_csv("PlayTennis.csv")

train\_data\_m.head()



def calc\_total\_entropy(train\_data, label, class\_list):

    total\_row = train\_data.shape[0]

    total\_entr = 0

    for c in class\_list:

        total\_class\_count = train\_data[train\_data[label] == c].shape[0]

        total\_class\_entr = - (total\_class\_count/total\_row)\*np.log2(total\_class\_count/total\_row)

        total\_entr += total\_class\_entr

    return total\_entr

def calc\_entropy(feature\_value\_data, label, class\_list):

    class\_count = feature\_value\_data.shape[0]

    entropy = 0

    for c in class\_list:

        label\_class\_count = feature\_value\_data[feature\_value\_data[label] == c].shape[0]

        entropy\_class = 0

        if label\_class\_count != 0:

            probability\_class = label\_class\_count/class\_count

            entropy\_class = - probability\_class \* np.log2(probability\_class)

        entropy += entropy\_class

    return entropy

def calc\_info\_gain(feature\_name, train\_data, label, class\_list):

    feature\_value\_list = train\_data[feature\_name].unique()

    total\_row = train\_data.shape[0]

    feature\_info = 0.0

    for feature\_value in feature\_value\_list:

        feature\_value\_data = train\_data[train\_data[feature\_name] == feature\_value]

        feature\_value\_count = feature\_value\_data.shape[0]

        feature\_value\_entropy = calc\_entropy(feature\_value\_data, label, class\_list)

        feature\_value\_probability = feature\_value\_count/total\_row

        feature\_info += feature\_value\_probability \* feature\_value\_entropy

    return calc\_total\_entropy(train\_data, label, class\_list) - feature\_info

def find\_most\_informative\_feature(train\_data, label, class\_list):

    feature\_list = train\_data.columns.drop(label)

    max\_info\_gain = -1

    max\_info\_feature = None

    for feature in feature\_list:

        feature\_info\_gain = calc\_info\_gain(feature, train\_data, label, class\_list)

        if max\_info\_gain < feature\_info\_gain:

            max\_info\_gain = feature\_info\_gain

            max\_info\_feature = feature

    return max\_info\_feature

def generate\_sub\_tree(feature\_name, train\_data, label, class\_list):

    feature\_value\_count\_dict = train\_data[feature\_name].value\_counts(sort=False)

    tree = {}

    for feature\_value, count in feature\_value\_count\_dict.iteritems():

        feature\_value\_data = train\_data[train\_data[feature\_name] == feature\_value]

        assigned\_to\_node = False

        for c in class\_list:

            class\_count = feature\_value\_data[feature\_value\_data[label] == c].shape[0]

            if class\_count == count:

                tree[feature\_value] = c

                train\_data = train\_data[train\_data[feature\_name] != feature\_value]

                assigned\_to\_node = True

        if not assigned\_to\_node:

            tree[feature\_value] = "?"

    return tree, train\_data

def make\_tree(root, prev\_feature\_value, train\_data, label, class\_list):

    if train\_data.shape[0] != 0:

        max\_info\_feature = find\_most\_informative\_feature(train\_data, label, class\_list)

        tree, train\_data = generate\_sub\_tree(max\_info\_feature, train\_data, label, class\_list)

        next\_root = None

        if prev\_feature\_value != None:

            root[prev\_feature\_value] = dict()

            root[prev\_feature\_value][max\_info\_feature] = tree

            next\_root = root[prev\_feature\_value][max\_info\_feature]

        else:

            root[max\_info\_feature] = tree

            next\_root = root[max\_info\_feature]

        for node, branch in list(next\_root.items()):

            if branch == "?":

                feature\_value\_data = train\_data[train\_data[max\_info\_feature] == node]

                make\_tree(next\_root, node, feature\_value\_data, label, class\_list)

def id3(train\_data\_m, label):

    train\_data = train\_data\_m.copy()

    tree = {}

    class\_list = train\_data[label].unique()

    make\_tree(tree, None, train\_data, label, class\_list)

    return tree

def predict(tree, instance):

    if not isinstance(tree, dict):

        return tree

    else:

        root\_node = next(iter(tree))

        feature\_value = instance[root\_node]

        if feature\_value in tree[root\_node]:

            return predict(tree[root\_node][feature\_value], instance)

        else:

            return None

def evaluate(tree, test\_data\_m, label):

    correct\_preditct = 0

    wrong\_preditct = 0

    for index, row in test\_data\_m.iterrows():

        result = predict(tree, test\_data\_m.iloc[index])

        if result == test\_data\_m[label].iloc[index]:

            correct\_preditct += 1

        else:

            wrong\_preditct += 1

    accuracy = correct\_preditct / (correct\_preditct + wrong\_preditct)

    return accuracy

tree = id3(train\_data\_m, 'Play Tennis')

tree

<ipython-input-17-87bd3102be49>:5: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.

for feature\_value, count in feature\_value\_count\_dict.iteritems():

<ipython-input-17-87bd3102be49>:5: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.

for feature\_value, count in feature\_value\_count\_dict.iteritems():

<ipython-input-17-87bd3102be49>:5: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.

for feature\_value, count in feature\_value\_count\_dict.iteritems():

{'Outlook': {'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}},

'Overcast': 'Yes',

'Rain': {'Wind': {'Weak': 'Yes', 'Strong': 'No'}}}}

accuracy = evaluate(tree, test\_data\_m, 'Play Tennis')

print("accuracy:", accuracy)

accuracy: 1.0

**PRACTICAL :6**

**Implement the classification model using clustering for the following techniques with K means clustering with prediction, Test Score and Confusion Matrix.**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

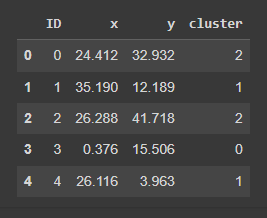
from matplotlib.colors import ListedColormap

%matplotlib inline

blobs = pd.read\_csv('kmeans\_blobs.csv')

colnames = list(blobs.columns[1:-1])

blobs.head()



customcmap = ListedColormap(["crimson", "mediumblue", "darkmagenta"])

fig, ax = plt.subplots(figsize=(8, 6))

plt.scatter(x=blobs['x'], y=blobs['y'], s=150,

            c=blobs['cluster'].astype('category'),

            cmap = customcmap)

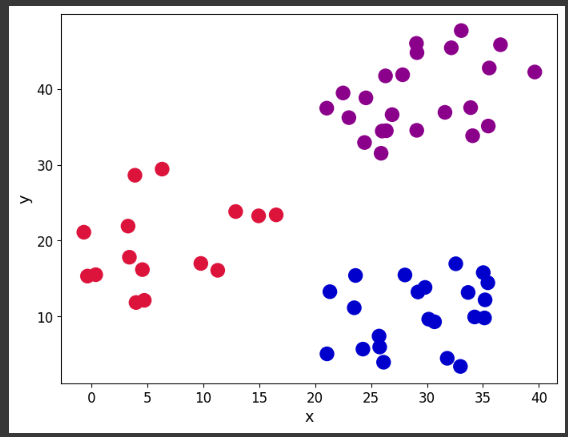
ax.set\_xlabel(r'x', fontsize=14)

ax.set\_ylabel(r'y', fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.show()



def initiate\_centroids(k, dset):

    '''

    Select k data points as centroids

    k: number of centroids

    dset: pandas dataframe

    '''

    centroids = dset.sample(k)

    return centroids

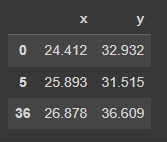
np.random.seed(42)

k=3

df = blobs[['x','y']]

centroids = initiate\_centroids(k, df)

centroids



def rsserr(a,b):

    '''

    Calculate the root of sum of squared errors.

    a and b are numpy arrays

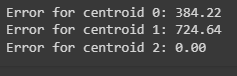
    '''

    return np.square(np.sum((a-b)\*\*2))

for i, centroid in enumerate(range(centroids.shape[0])):

    err = rsserr(centroids.iloc[centroid,:], df.iloc[36,:])

    print('Error for centroid {0}: {1:.2f}'.format(i, err))



def centroid\_assignation(dset, centroids):

    '''

    Given a dataframe `dset` and a set of `centroids`, we assign each

    data point in `dset` to a centroid.

    - dset - pandas dataframe with observations

    - centroids - pa das dataframe with centroids

    '''

    k = centroids.shape[0]

    n = dset.shape[0]

    assignation = []

    assign\_errors = []

    for obs in range(n):

        # Estimate error

        all\_errors = np.array([])

        for centroid in range(k):

            err = rsserr(centroids.iloc[centroid, :], dset.iloc[obs,:])

            all\_errors = np.append(all\_errors, err)

        # Get the nearest centroid and the error

        nearest\_centroid =  np.where(all\_errors==np.amin(all\_errors))[0].tolist()[0]

        nearest\_centroid\_error = np.amin(all\_errors)

        # Add values to corresponding lists

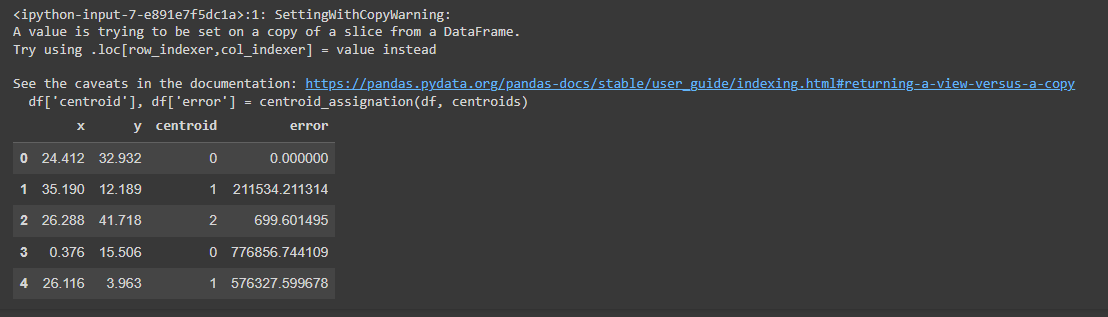
        assignation.append(nearest\_centroid)

        assign\_errors.append(nearest\_centroid\_error)

    return assignation, assign\_errors

df['centroid'], df['error'] = centroid\_assignation(df, centroids)

df.head()



fig, ax = plt.subplots(figsize=(8, 6))

plt.scatter(df.iloc[:,0], df.iloc[:,1],  marker = 'o',

            c=df['centroid'].astype('category'),

            cmap = customcmap, s=80, alpha=0.5)

plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],

            marker = 's', s=200, c=[0, 1, 2],

            cmap = customcmap)

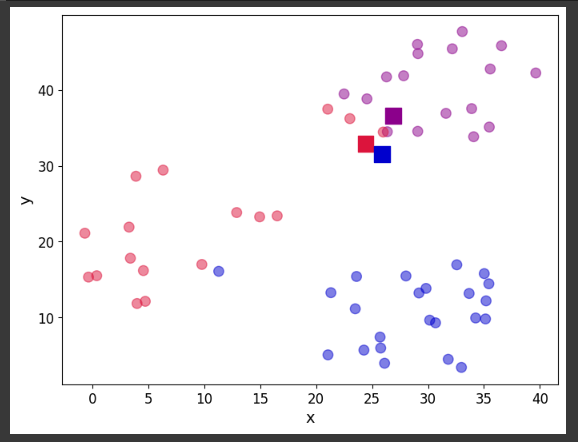
ax.set\_xlabel(r'x', fontsize=14)

ax.set\_ylabel(r'y', fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.show()

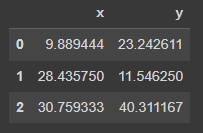


print("The total error is {0:.2f}".format(df['error'].sum()))

The total error is 11927659.01

centroids = df.groupby('centroid').agg('mean').loc[:, colnames].reset\_index(drop = True)

centroids



fig, ax = plt.subplots(figsize=(8, 6))

plt.scatter(df.iloc[:,0], df.iloc[:,1],  marker = 'o',

            c=df['centroid'].astype('category'),

            cmap = customcmap, s=80, alpha=0.5)

plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],

            marker = 's', s=200,

            c=[0, 1, 2], cmap = customcmap)

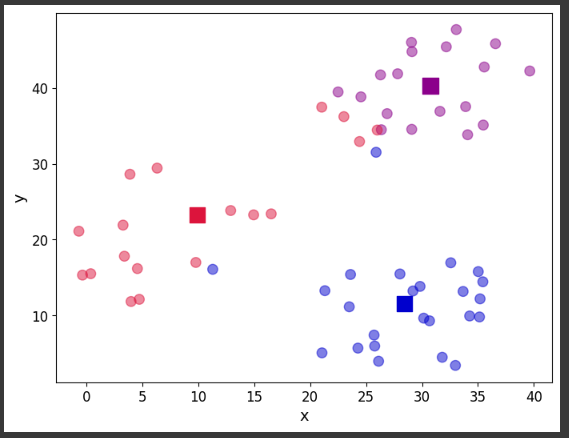
ax.set\_xlabel(r'x', fontsize=14)

ax.set\_ylabel(r'y', fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.show()



def kmeans(dset, k=2, tol=1e-4):

    '''

    K-means implementationd for a

    `dset`:  DataFrame with observations

    `k`: number of clusters, default k=2

    `tol`: tolerance=1E-4

    '''

    # Let us work in a copy, so we don't mess the original

    working\_dset = dset.copy()

    # We define some variables to hold the error, the

    # stopping signal and a counter for the iterations

    err = []

    goahead = True

    j = 0

    # Step 2: Initiate clusters by defining centroids

    centroids = initiate\_centroids(k, dset)

    while(goahead):

        # Step 3 and 4 - Assign centroids and calculate error

        working\_dset['centroid'], j\_err = centroid\_assignation(working\_dset, centroids)

        err.append(sum(j\_err))

        # Step 5 - Update centroid position

        centroids = working\_dset.groupby('centroid').agg('mean').reset\_index(drop = True)

        # Step 6 - Restart the iteration

        if j>0:

            # Is the error less than a tolerance (1E-4)

            if err[j-1]-err[j]<=tol:

                goahead = False

        j+=1

    working\_dset['centroid'], j\_err = centroid\_assignation(working\_dset, centroids)

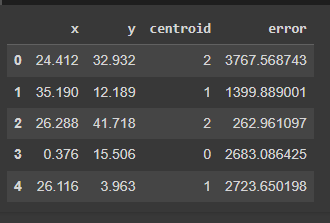
    centroids = working\_dset.groupby('centroid').agg('mean').reset\_index(drop = True)

    return working\_dset['centroid'], j\_err, centroids

np.random.seed(42)

df['centroid'], df['error'], centroids =  kmeans(df[['x','y']], 3)

df.head()



fig, ax = plt.subplots(figsize=(8, 6))

plt.scatter(df.iloc[:,0], df.iloc[:,1],  marker = 'o',

            c=df['centroid'].astype('category'),

            cmap = customcmap, s=80, alpha=0.5)

plt.scatter(centroids.iloc[:,0], centroids.iloc[:,1],

            marker = 's', s=200, c=[0, 1, 2],

            cmap = customcmap)

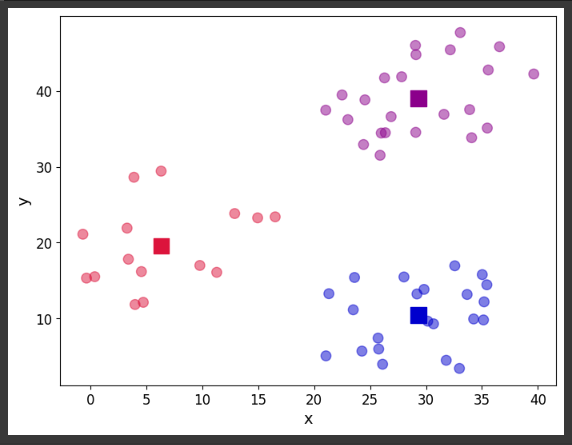
ax.set\_xlabel(r'x', fontsize=14)

ax.set\_ylabel(r'y', fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.show()



err\_total = []

n = 10

df\_elbow = blobs[['x','y']]

for i in range(n):

    \_, my\_errs, \_ = kmeans(df\_elbow, i+1)

    err\_total.append(sum(my\_errs))

fig, ax = plt.subplots(figsize=(8, 6))

plt.plot(range(1,n+1), err\_total, linewidth=3, marker='o')

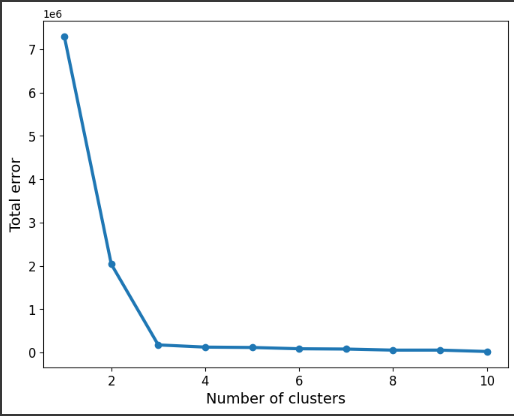
ax.set\_xlabel(r'Number of clusters', fontsize=14)

ax.set\_ylabel(r'Total error', fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

plt.show()



**PRACTICAL 7**

**Implement the Rule Based method and test the same.**

import pandas as pd

df = pd.read\_csv("persona.csv")

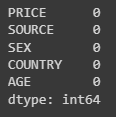
df.head()

df.shape

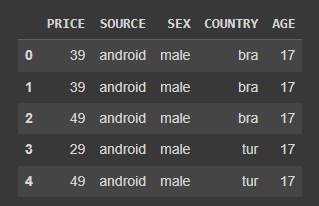
df.describe().T

df.isnull().values.any()

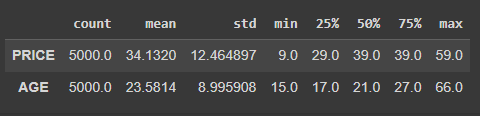
df.isnull().sum()



df.head()



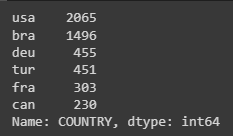
df.describe().T



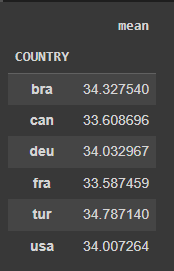
df["SOURCE"].nunique()

df["SOURCE"].value\_counts()

df["COUNTRY"].value\_counts()



df.groupby("COUNTRY")["PRICE"].agg({"mean"})

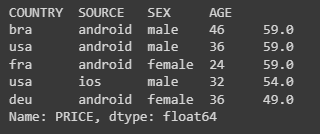


df.groupby(["COUNTRY", 'SOURCE'])["PRICE"].mean()



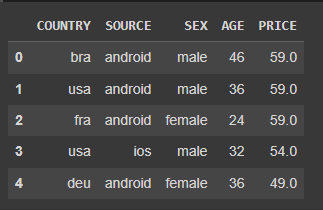
agg\_df = df.groupby(["COUNTRY", 'SOURCE', "SEX", "AGE"])["PRICE"].mean().sort\_values(ascending=False)

agg\_df.head()



agg\_df = agg\_df.reset\_index()

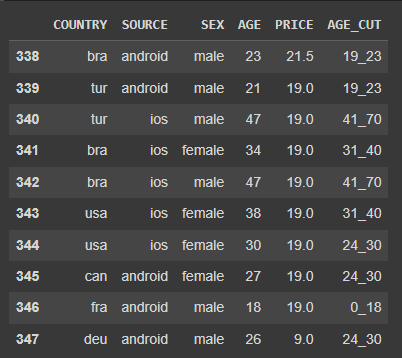
agg\_df.head()



my\_labels = ['0\_18', '19\_23', '24\_30', '31\_40', '41\_70']

agg\_df["AGE\_CUT"] = pd.cut(x=agg\_df["AGE"], bins=[0, 18, 23, 30, 40, 70], labels=my\_labels)

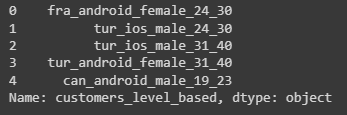
agg\_df.tail(10) # Just checking data



agg\_df["customers\_level\_based"] = [f"{i[0]}\_{i[1]}\_{i[2]}\_{i[-1]}" for i in agg\_df.values]

agg\_df = agg\_df.loc[:, ["customers\_level\_based", "PRICE"]].groupby("customers\_level\_based").agg({"PRICE": "mean"}).sort\_values(by="PRICE", ascending=False).reset\_index()

agg\_df["customers\_level\_based"].head() # Just checking data again



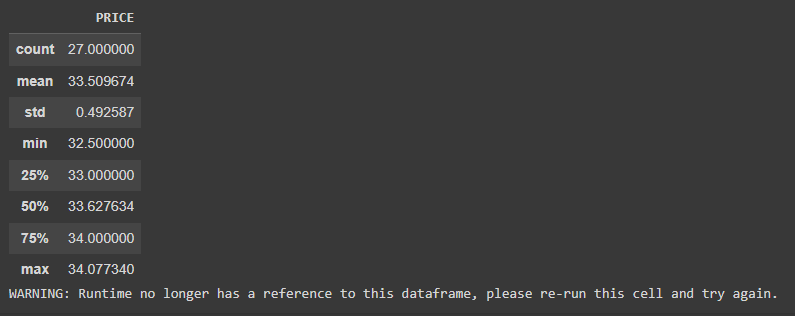
agg\_df["SEGMENT"] = pd.qcut(agg\_df["PRICE"], 4, labels=["D", "C", "B", "A"])

agg\_df.head()

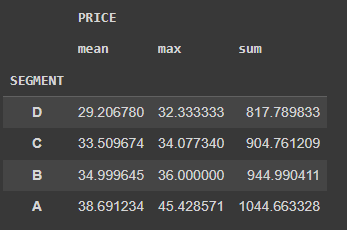


agg\_df.groupby(["SEGMENT"]).agg({"PRICE": ["mean", "max", "sum"]})

agg\_df[agg\_df["SEGMENT"] == "C"].describe()



agg\_df.groupby(["SEGMENT"]).agg({"PRICE": ["mean", "max", "sum"]})



new\_user = "fra\_android\_male\_24\_30"

print(agg\_df[agg\_df["customers\_level\_based"] == new\_user])



**PRACTICAL 8.**

**Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patient using standard Heart Disease Data Set.**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# visualization tools

import matplotlib.pyplot as plt

import seaborn as sns

from plotly.offline import init\_notebook\_mode, iplot

init\_notebook\_mode(connected=True)

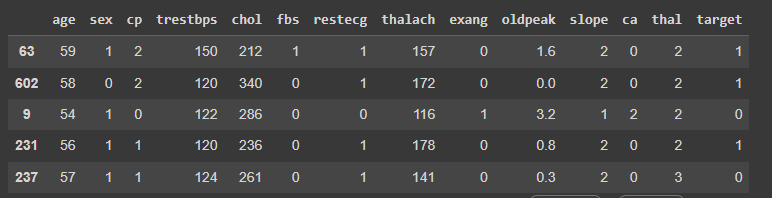
import os

for dirname, \_, filenames in os.walk('heart.csv'):

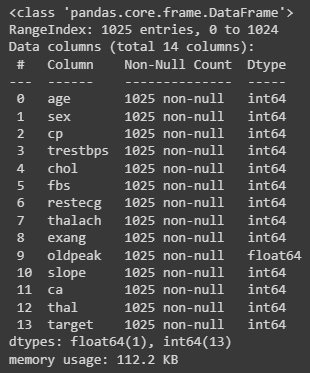
    for filename in filenames:

        print(os.path.join(dirname, filename))

df = pd.read\_csv("heart.csv")

****

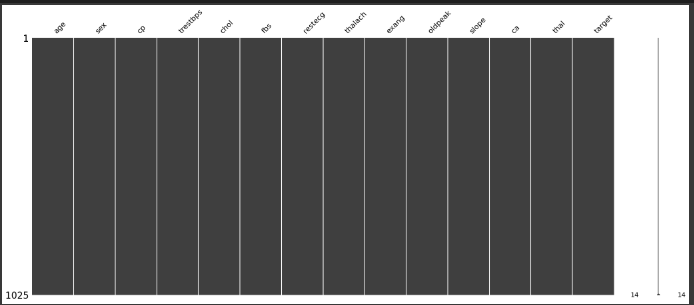
df.info()

****

import missingno as msno

msno.matrix(df)

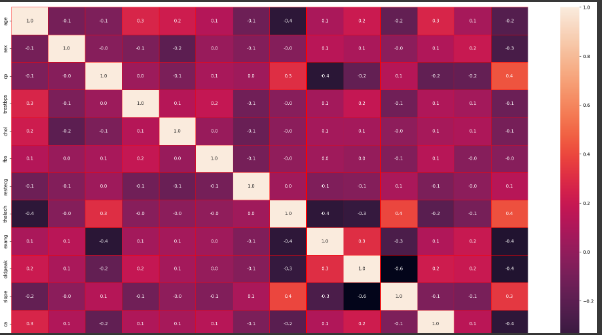
plt.show()

****

f,ax = plt.subplots(figsize=(25, 15))

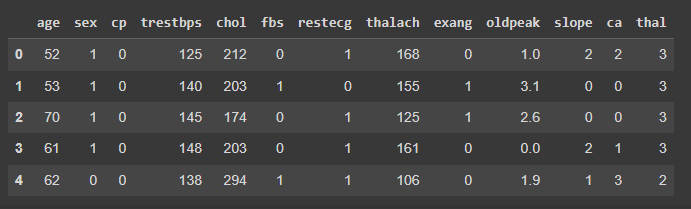
sns.heatmap(df.corr(), annot=True, linewidths=0.5,linecolor="red", fmt= '.1f',ax=ax)

plt.show()

****

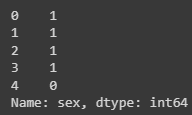
X=df.iloc[:, 0:13]

X.head()

****

y=df.iloc[:,1]

y.head()

****

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X)

X = scaler.transform(X)

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=42)

from sklearn.naive\_bayes import GaussianNB

from sklearn import metrics

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

print('Accuracy Score:')

print(metrics.accuracy\_score(y\_test,y\_pred))

**Accuracy Score: 1.0**

from sklearn.metrics import confusion\_matrix

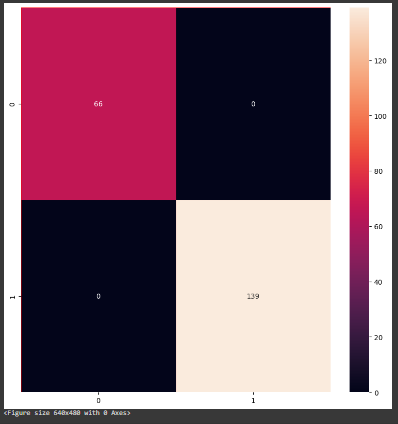
cm = confusion\_matrix(y\_test, y\_pred)

f,ax = plt.subplots(figsize=(10, 10))

sns.heatmap(cm, annot=True, linewidths=0.5,linecolor="red", fmt= '.0f',ax=ax)

plt.show()

plt.savefig('ConfusionMatrix.png')

****

from sklearn.metrics import  f1\_score

f1\_score = f1\_score(y\_test, y\_pred)

print("F1 Score:")

print(f1\_score)

F1 Score:1.0