DEPARTMENT OF INFORMATION TECHNOLOGY

SMT. PARMESHWARIDEVI DURGADUTT TIBREWALA

LIONS JUHU COLLEGE

OF ARTS, COMMERE AND SCIENCE

Affiliated to University of Mumbai

J.B. NAGAR, ANDHERI (E), MUMBAI-400059

Academic Year 2022-2023

MACHINE LEARNING

For

Semester III

Submitted By:

MR. SAURAV KANOJIA

Msc.IT (Sem III)

SMT. PARMESHWARIDEVI DURGADUTT TIBREWALA

LIONS JUHU COLLEGE

OF ARTS, COMMERE AND SCIENCE

Affiliated to University of Mumbai

J.B. NAGAR, ANDHERI (E), MUMBAI-400059

DEPARTMENT OF INFORMATION TECHNOLOGY

Certificate of Approval

This is to certify that practical entitled **"Machine Learning"** Undertaken at **SMT.PARMESHWARIDEVI DURGADUTT TIBREWALA LIONS JUHU COLLEGE OF ARTS, COMMERECE & SCIENCE.** By **MR. SAURAV KANOJIA Seat No.3269789** in partial fulfilment of **M.Sc. (IT) master degree (Semester III)** Examination had not been submitted for any other examination and does not form of any other course undergone by the candidate. It is further certified that she has completed all required phases of the practical.

___________ ____________

Internal Examiner External Examiner

__________________ _____________ HOD

/ In-Charge / Coordinator Signature/ Principal/Stamp

INDEX

Sr. No.	Practical	Date	Sign
1	A. Design a simple machine learning model to train the training instances and test the same.		
	B. Implementing and demonstrate the FIND-S Algorithm for finding the most specific hypothesis based on a given set of training data and samples. Read the training data from a .CSV file		
2	A. Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking		
	B. 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.		
3	A. Write a program to implement the naive Bayesian Classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.		
4.	A. For a given set of training data examples stored in a .CSV file implement Least Square Regression Algorithm.		
	B. For a given set of training data examples stored in a .CSV file implement Logistic Regression Algorithm.		
5	A. 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.		
	B. Write a program to implement K-Nearest Neighbour algorithm to classify the iris dataset.		
6.	A. Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.		
	B. Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.		

7.	A. Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix.		

	B. Implement the Rule based method and test the same		
8.	A. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets		
	B. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.		

Practical 1

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

The practical contains of a write, followed by code and output ofthe same, along with the observation i.e., the interpretation of the algorithm applied on the dataset.

This Practical consists of a write up with the following key points: -

1) What is Linear regression.
2) Algorithm of Linear regression.
3) Data set used for Linear regression.

The Practical is performed in Python. More about the dataset canbe seen ahead in this document.

What is Linear Regression?

Linear regression is used to predict the relationship between two variables by applying a linear equation to observed data. There are two types of variable, one variable is called an independent variable, and the other is a dependent variable. Linear regression is commonly used for predictive analysis. The main idea of regression is to examine two things. First, does a set of predictor variables do a good job in predicting an outcome (dependent) variable? The second thing is which variables are significant predictors of the outcome variable?

Linear regression can be further divided into two types of the algorithm:

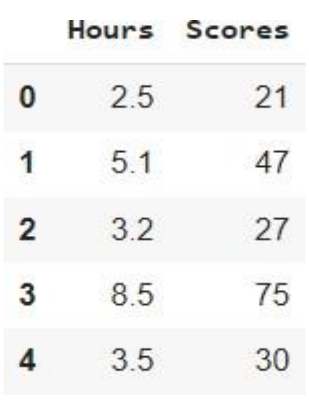
- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

To calculate best-fit line linear regression uses a traditional slope-intercept form.

$$y = mx+b \quad \Longrightarrow \quad y = a_0+a_1x$$

y= Dependent Variable. x= Independent Variable. a0= intercept of the line. a1 = Linear regression coefficient. **Simple Linear Regression**

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

#Reading Dataset
lineardataset = pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/student_scores.cs
v")

lineardataset.head()
```

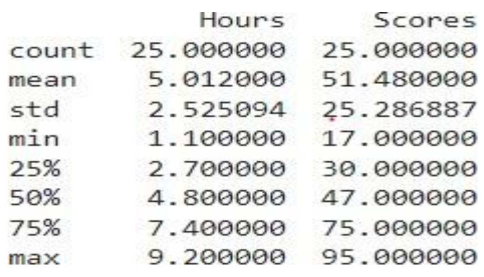


	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

```
lineardataset.shape
```

```
○ (25, 2)
```

```
datasetDescription = lineardataset.describe() print(datasetDescription)
```



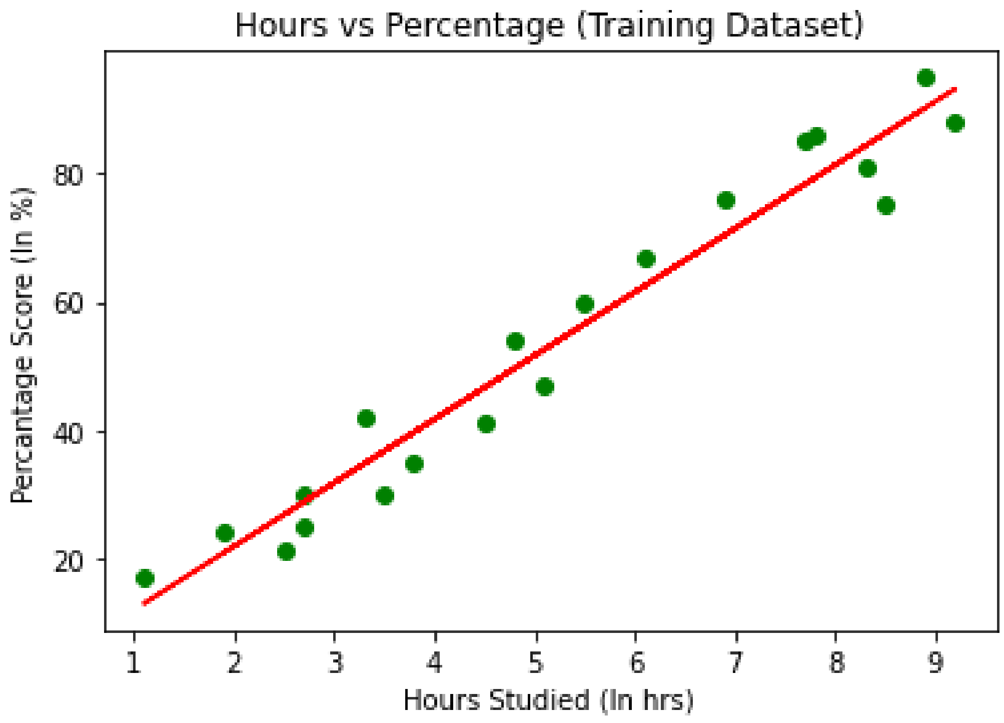
```
           Hours     Scores
count  25.000000  25.000000
mean    5.012000  51.480000
std     2.525094  25.286887
min     1.100000  17.000000
25%     2.700000  30.000000
50%     4.800000  47.000000
75%     7.400000  75.000000
max     9.200000  95.000000
```

```
x= lineardataset.iloc[:, :-1].values y=
lineardataset.iloc[:, 1].values

# Splitting the dataset into training 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= 0.2, random_state=0)

#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression regressor=
LinearRegression()
```

```
regressor.fit(x_train, y_train)
```

```
LinearRegression()
```

```
#Prediction of Test and Training set result y_pred=
regressor.predict(x_test)
x_pred= regressor.predict(x_train)

plt.scatter(x_train, y_train, color="green")
plt.plot(x_train, x_pred, color="red")
plt.title("Hours vs Percentage (Training Dataset)")
plt.xlabel("Hours Studied (In hrs)")
plt.ylabel("Percantage Score (In %)") plt.show()
```



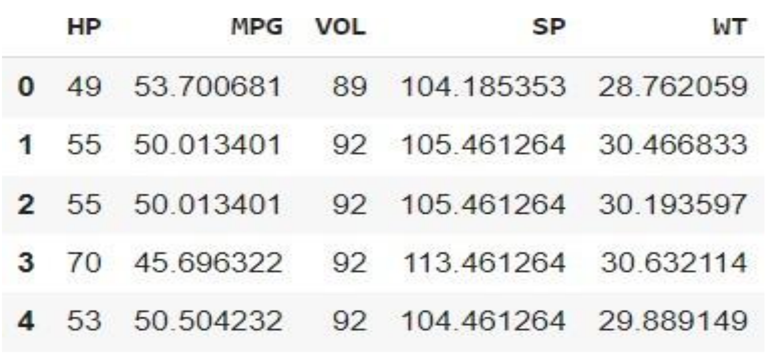
```
#Making Predictions from sklearn
import metrics y_pred =
regressor.predict(x_test)
linear_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}) print(linear_df)
```

```
   Actual  Predicted
0      20  16.884145
1      27  33.732261
2      69  75.357018
3      30  26.794801
4      62  60.491033
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred)) ○ Mean
Absolute Error: 4.183859899002982

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

    ○ Mean Squared Error: 21.598769307217456

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

    ○ Root Mean Squared Error: 4.647447612100373
```

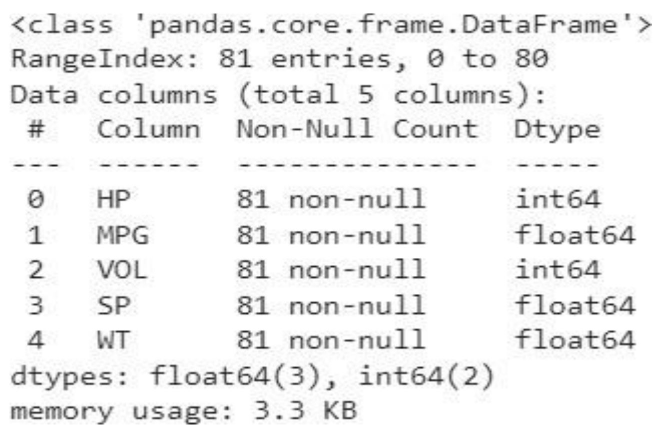
Multiple Linear Regression

```
import pandas as pd import
matplotlib.pyplot as plt import
seaborn as sns
from statsmodels.graphics.regressionplots import influence_plot
import statsmodels.formula.api as smf import numpy as np

#Read the data cars =
pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/Cars.csv")

cars.head()
```



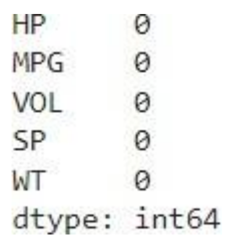
	HP	MPG	VOL	SP	WT
0	49	53.700681	89	104.185353	28.762059
1	55	50.013401	92	105.461264	30.466833
2	55	50.013401	92	105.461264	30.193597
3	70	45.696322	92	113.461264	30.632114
4	53	50.504232	92	104.461264	29.889149

```
cars.info()
```



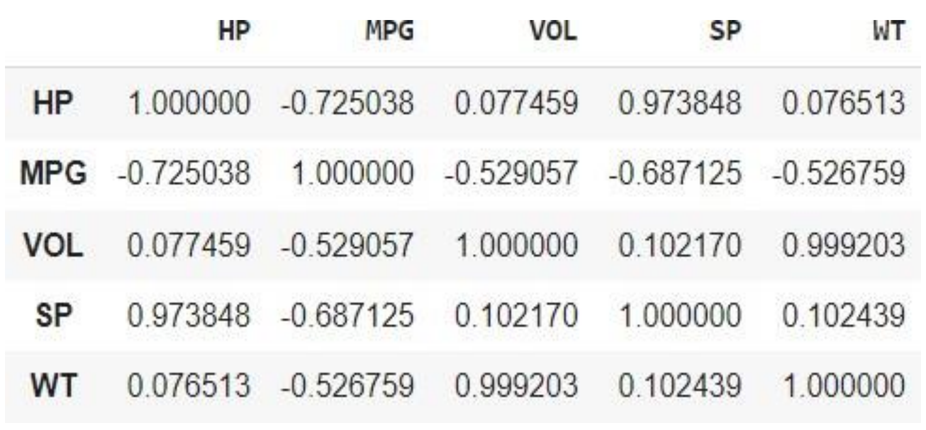
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   HP      81 non-null     int64
 1   MPG     81 non-null     float64
 2   VOL     81 non-null     int64
 3   SP      81 non-null     float64
 4   WT      81 non-null     float64
dtypes: float64(3), int64(2)
memory usage: 3.3 KB
```

#check for missing values cars.isna().sum()



```
HP     0
MPG    0
VOL    0
SP     0
WT     0
dtype: int64
```

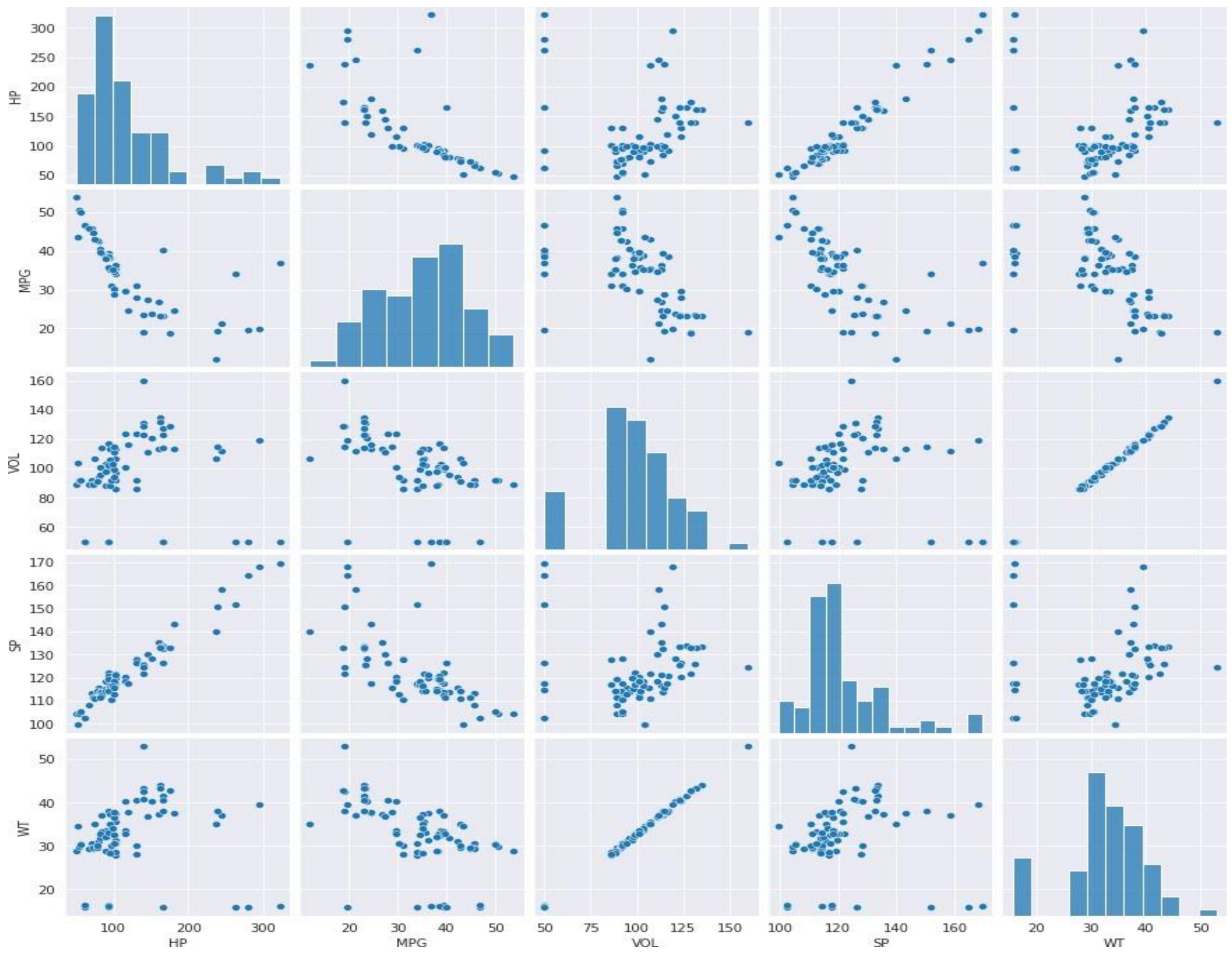
Correlation matrix cars.corr()



	HP	MPG	VOL	SP	WT
HP	1.000000	-0.725038	0.077459	0.973848	0.076513
MPG	-0.725038	1.000000	-0.529057	-0.687125	-0.526759
VOL	0.077459	-0.529057	1.000000	0.102170	0.999203
SP	0.973848	-0.687125	0.102170	1.000000	0.102439
WT	0.076513	-0.526759	0.999203	0.102439	1.000000

Scatterplot between variables along with histograms

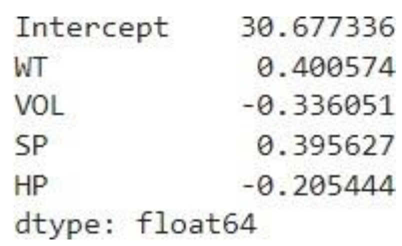
#Format the plot background and scatter plots for all the variables
sns.set_style(style='darkgrid') sns.pairplot(cars)



Preparing a model

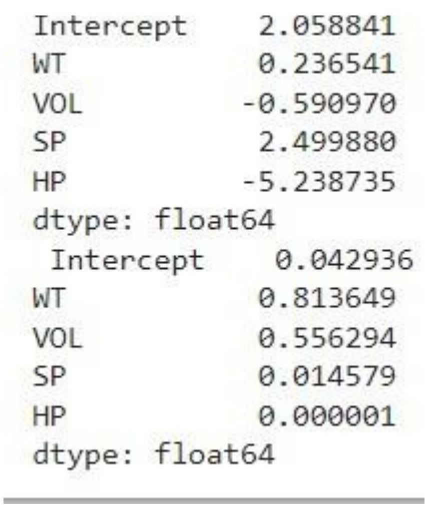
```
#Build model
import statsmodels.formula.api as smf model =
smf.ols('MPG~WT+VOL+SP+HP',data=cars).fit()
```

#Coefficients model.params



```
Intercept    30.677336
WT            0.400574
VOL          -0.336051
SP            0.395627
HP           -0.205444
dtype: float64
```

#t and p-Values print(model.tvalues,
'\n', model.pvalues)



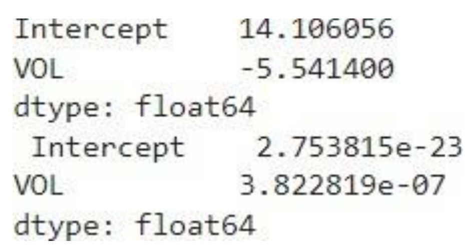
```
Intercept    2.058841
WT           0.236541
VOL         -0.590970
SP           2.499880
HP          -5.238735
dtype: float64
 Intercept    0.042936
WT           0.813649
VOL          0.556294
SP           0.014579
HP           0.000001
dtype: float64
```

#R squared values
(model.rsquared,model.rsquared_adj)

- (0.7705372737359842, 0.7584602881431413)

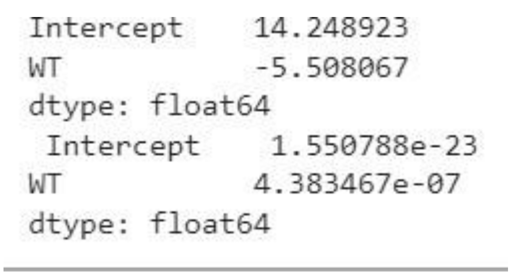
Simple Linear Regression Models

ml_v=smf.ols('MPG~VOL',data = cars).fit()
#t and p-Values print(ml_v.tvalues,
'\n', ml_v.pvalues)



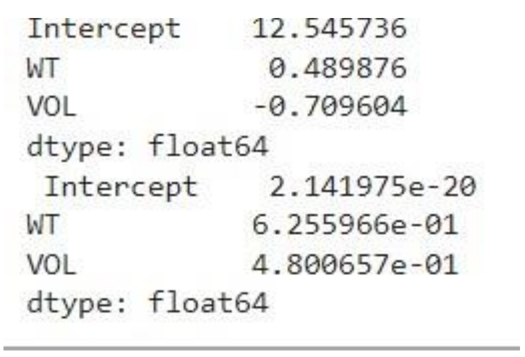
```
Intercept    14.106056
VOL          -5.541400
dtype: float64
 Intercept    2.753815e-23
VOL          3.822819e-07
dtype: float64
```

ml_w=smf.ols('MPG~WT',data = cars).fit() print(ml_w.tvalues,
'\n', ml_w.pvalues)



```
Intercept    14.248923
WT           -5.508067
dtype: float64
 Intercept    1.550788e-23
WT           4.383467e-07
dtype: float64
```

ml_wv=smf.ols('MPG~WT+VOL',data = cars).fit() print(ml_wv.tvalues,
'\n', ml_wv.pvalues)



```
Intercept    12.545736
WT            0.489876
VOL          -0.709604
dtype: float64
 Intercept    2.141975e-20
WT           6.255966e-01
VOL          4.800657e-01
dtype: float64
```

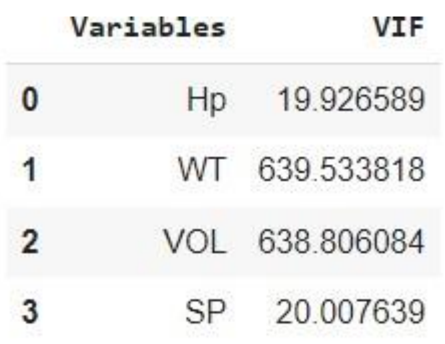
Calculating VIF

rsq_hp = smf.ols('HP~WT+VOL+SP',data=cars).fit().rsquared vif_hp
= 1/(1-rsq_hp)

rsq_wt = smf.ols('WT~HP+VOL+SP',data=cars).fit().rsquared vif_wt
= 1/(1-rsq_wt)

rsq_vol = smf.ols('VOL~WT+SP+HP',data=cars).fit().rsquared vif_vol
= 1/(1-rsq_vol)

```
rsq_sp = smf.ols('SP~WT+VOL+HP',data=cars).fit().rsquared vif_sp
= 1/(1-rsq_sp)

# Storing vif values in a data frame
d1 = {'Variables':['Hp','WT','VOL','SP'],'VIF':[vif_hp,vif_wt,vif_vol,vif_sp]}
Vif_frame = pd.DataFrame(d1)
Vif_frame
```

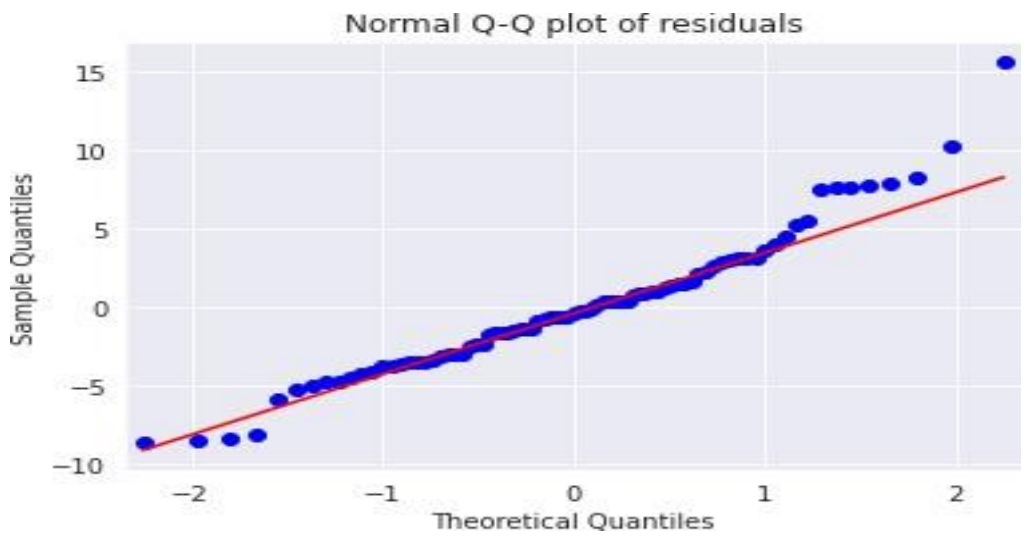


	Variables	VIF
0	Hp	19.926589
1	WT	639.533818
2	VOL	638.806084
3	SP	20.007639

Residual Analysis

Test for Normality of Residuals (Q-Q Plot)

```
import statsmodels.api as sm
qqplot=sm.qqplot(model.resid,line='q') # line = 45 to draw the diagnoal line
plt.title("Normal Q-Q plot of residuals")
plt.show()
```

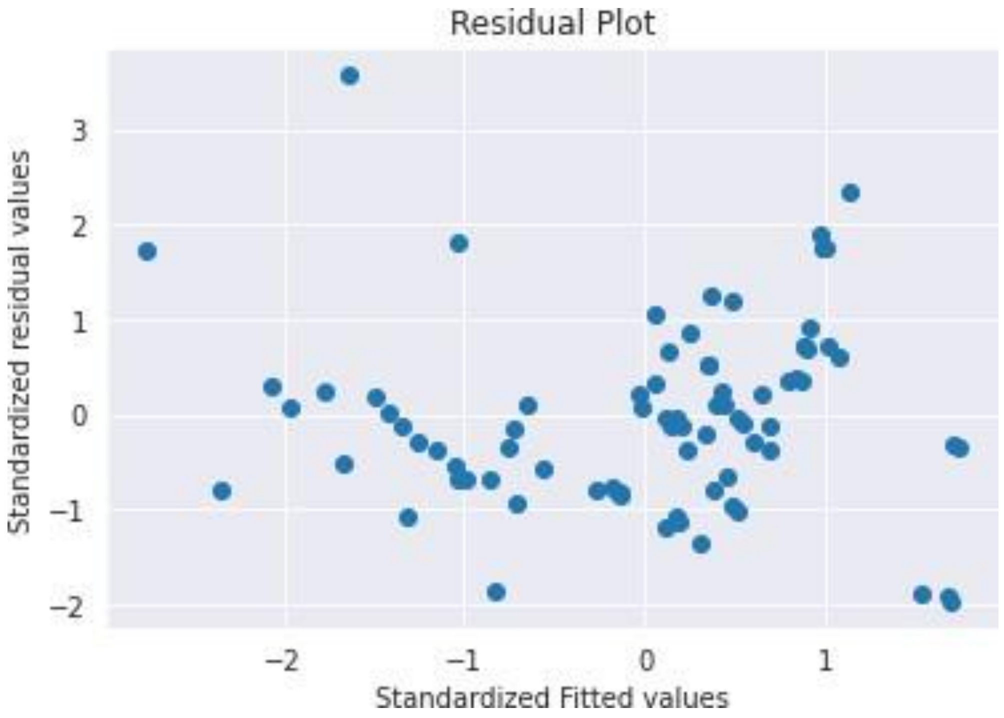


```
list(np.where(model.resid>10))
```

- [array([0, 76])]

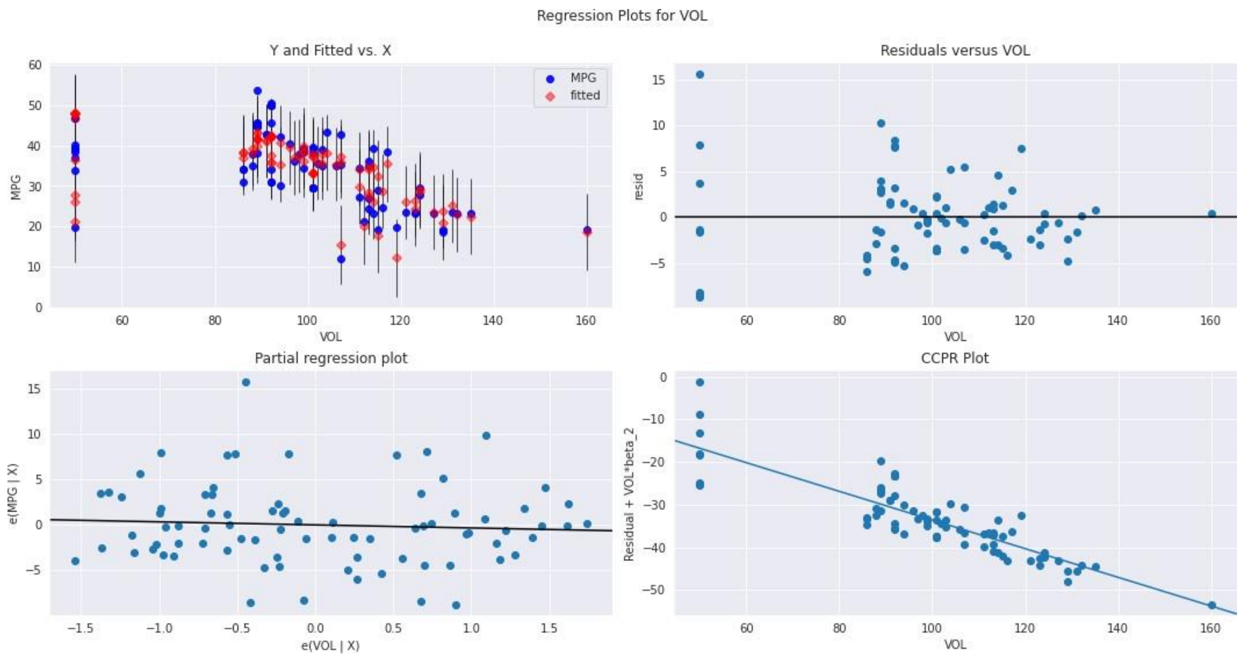
Residual Plot for Homoscedasticity def

```
get_standardized_values( vals ):
    return (vals - vals.mean())/vals.std()

plt.scatter(get_standardized_values(model.fittedvalues),
          get_standardized_values(model.resid))

plt.title('Residual                    Plot')
plt.xlabel('Standardized   Fitted   values')
plt.ylabel('Standardized residual values')
plt.show()
```

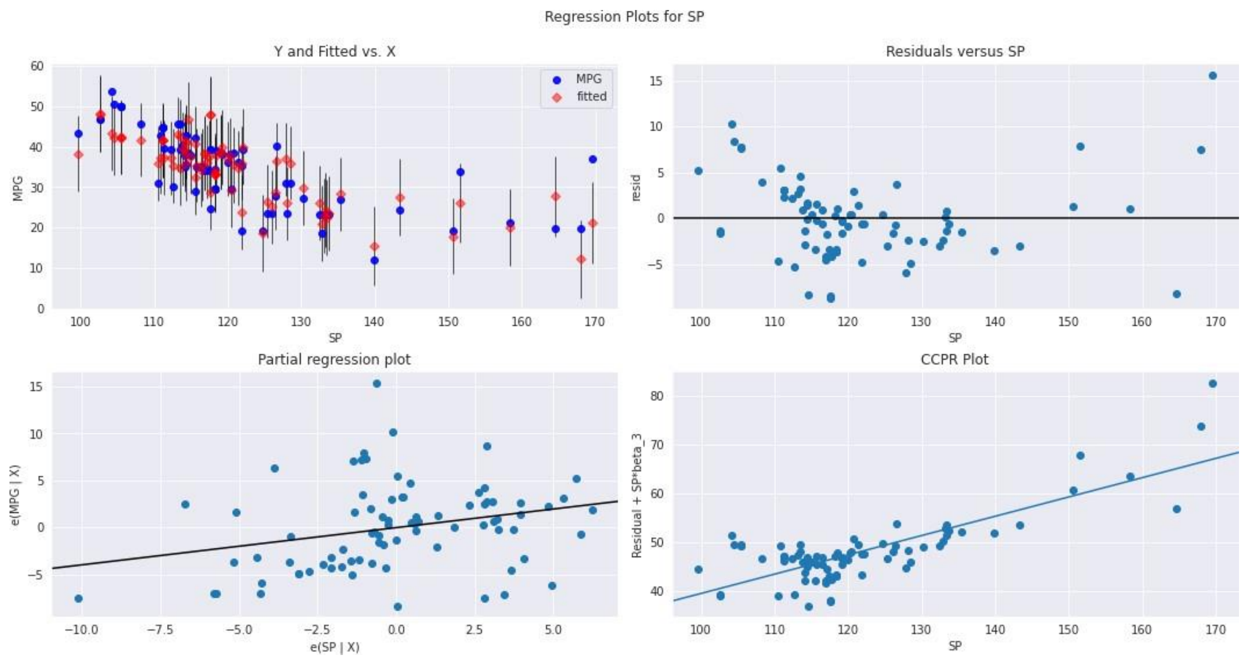


Residual Vs Regressors

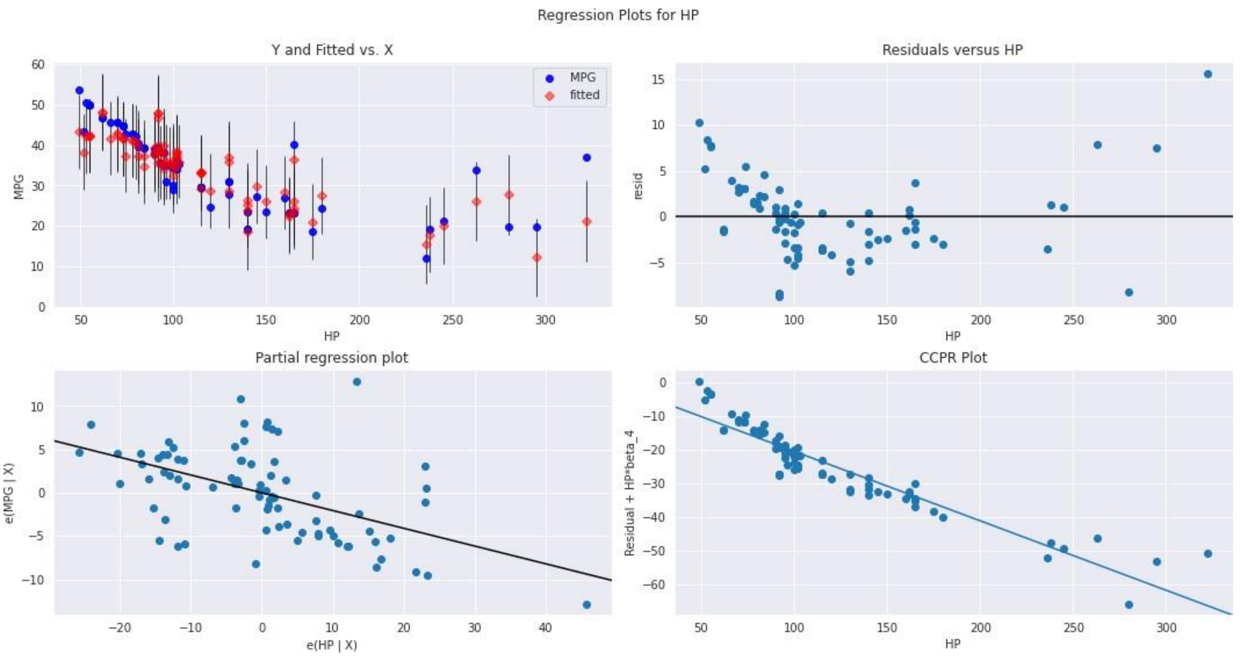
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "VOL", fig=fig) plt.show()
```



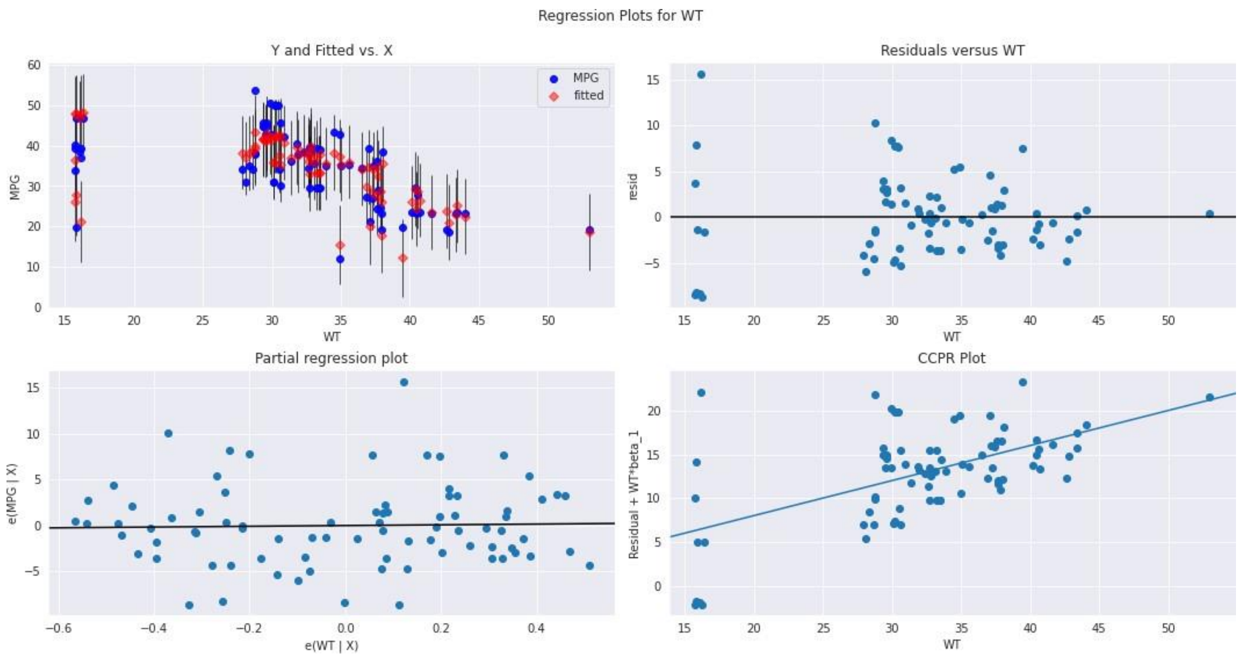
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "SP", fig=fig) plt.show()
```



```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "HP", fig=fig) plt.show()
```



```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "WT", fig=fig) plt.show()
```


Regression Plots for WT
Y and Fitted vs. X
MPG
fitted
WT
Residuals versus WT
resid
Partial regression plot
e(MPG | X)
e(WT | X)
CCPR Plot
Residual + WT*beta_1

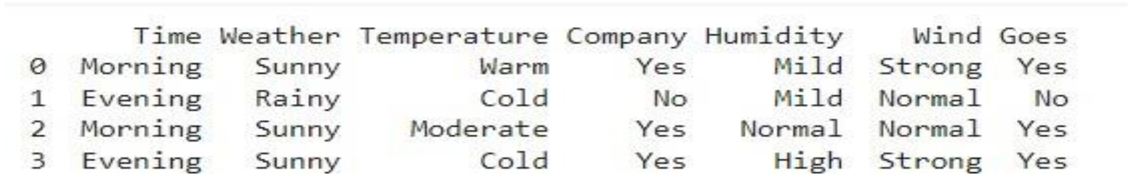
B. Implementing and demonstrate the FIND-S Algorithm for finding the most specific hypothesis based on a given set of training data and samples. Read the training data from a .CSV file

The practical contains of a write, followed by code and output of the same, along with the observation i.e., the interpretation of the algorithm applied on the dataset. This Practical consists of a write up with the following key points: - 1. What is Find-S Algorithm.
2. Algorithm of Find-S Algorithm.
3. Data set used for Find-S Algorithm.
The Practical is performed in Python. More about the dataset can be seen ahead in this document.

1. What is Find-S Algorithm.
The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We must note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.
Important Representation:

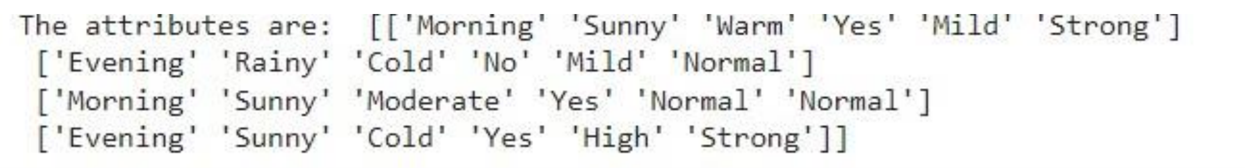
1. **?** indicates that any value is acceptable for the attribute.
2. specify a single required value (e.g., Cold) for the attribute.
3. **Φ** indicates that no value is acceptable.
4. The most **general hypothesis** is represented by: **{?, ?, ?, ?, ?, ?}**
5. The most **specific hypothesis** is represented by: **{ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}**

```
import pandas as pd import
numpy as np

#reading the dataset
data = pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/walkdata.csv") print(data)
```



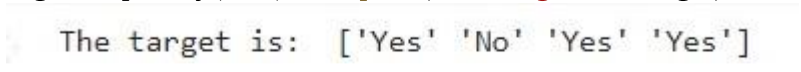
```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes
```

```
#making an array of all the attributes
d = np.array(data)[:,:-1] print("The
attributes are: ",d)
```



```
The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]
```

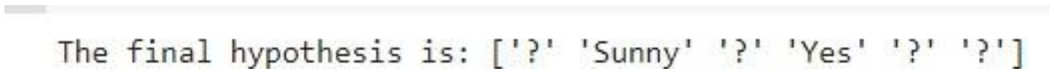
```
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1] print("The target is: ",target)
```



```
The target is:  ['Yes' 'No' 'Yes' 'Yes']
```

```
#training function to implement find-s algorithm def
train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
    return specific_hypothesis
```

```
#obtaining the final hypothesis print("The
final hypothesis is:",train(d,target))
```



```
The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

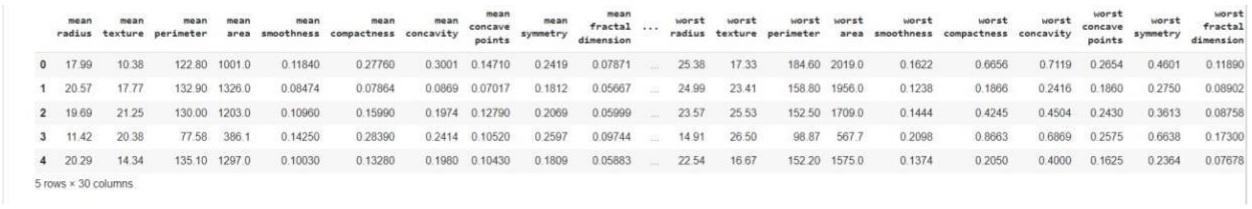
Practical 2

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

In machine learning, principal component analysis (PCA) is a pre-processing step that is often used to reduce the dimensionality of a data set before training a model. By identifying the directions in which the data varies the most, PCA can help to reduce the complexity of the data and identify patterns in the data that may be useful for making predictions.
PCA is often used in combination with other techniques, such as clustering or classification, to improve the performance of a machine learning model. For example, by reducing the dimensionality of the data, PCA can help to speed up the training of a model and reduce overfitting. It can also be used to visualize high-dimensional data, which can be helpful for understanding the structure of the data and identifying trends or patterns.
To perform PCA in machine learning, you follow the same steps as in standard PCA: standardize the data, compute the covariance matrix, compute the eigenvectors and eigenvalues, and project the data onto the lower-dimensional space. However, in machine learning, you may also need to consider how the choice of the number of dimensions affects the performance of your model, and you may need to tune the parameters of the model based on the reduced data.

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

```
%matplotlib inline

# Here we are using inbuilt dataset of scikit learn from
sklearn.datasets import load_breast_cancer

# instantiating cancer =
load_breast_cancer()

# creating dataframe df = pd.DataFrame(cancer['data'], columns =
cancer['feature_names'])

# checking head of dataframe df.head()
```



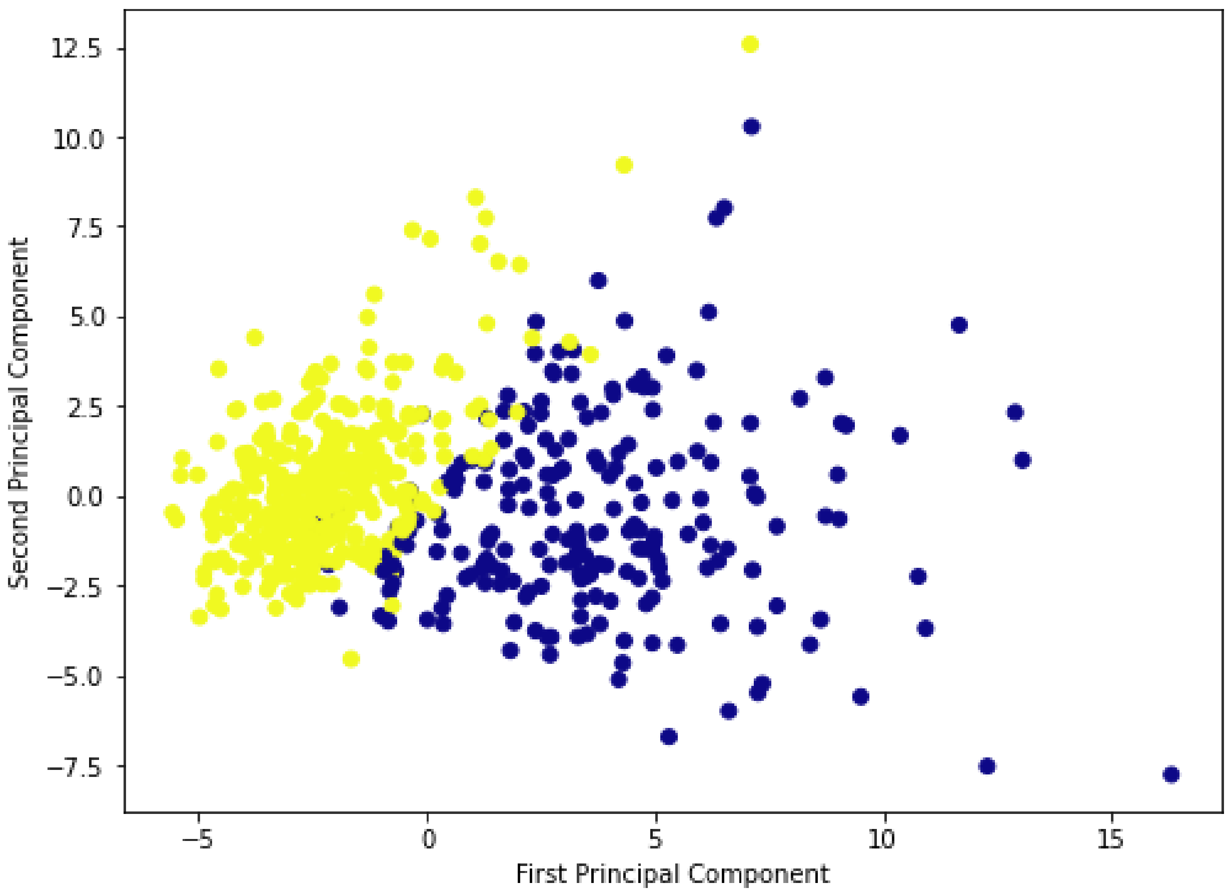
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 30 columns

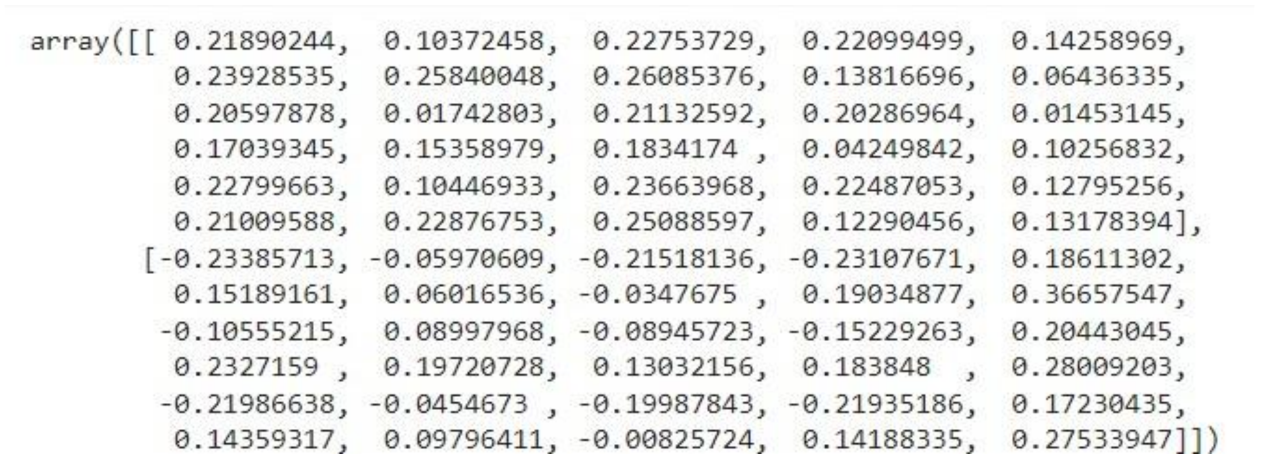
```
# Importing standardscalar module from
sklearn.preprocessing import StandardScaler
scalar = StandardScaler()

# fitting scalar.fit(df) scaled_data
= scalar.transform(df)

# Importing PCA from
sklearn.decomposition import PCA

# Let's say, components = 2 pca =
PCA(n_components = 2)
pca.fit(scaled_data) x_pca =
pca.transform(scaled_data)
x_pca.shape
```

```
(569, 2)
```

```
# giving a larger plot plt.figure(figsize =(8, 6)) plt.scatter(x_pca[:, 0],
x_pca[:, 1], c = cancer['target'], cmap ='plasma')

# labeling x and y axes plt.xlabel('First
Principal Component')
plt.ylabel('Second Principal
Component')
```



# components pca.components_

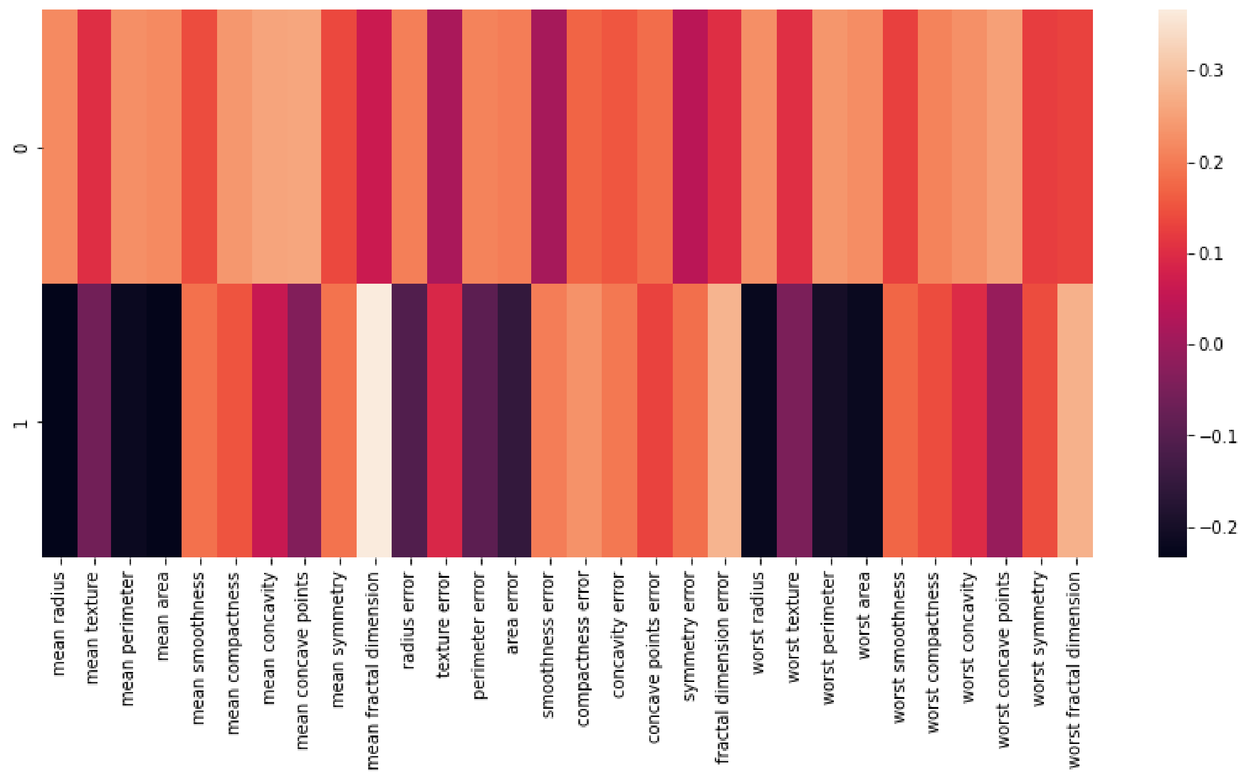


```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
         0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
         0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
         0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
       [-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611302,
         0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
        -0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
         0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
        -0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
         0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.27533947]])
```

df_comp = pd.DataFrame(pca.components_, columns = cancer['feature_names'])

plt.figure(figsize =(14, 6))

# plotting heatmap sns.heatmap(df_comp)


0
1
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
0.3
0.2
0.1
0.0
−0.1
−0.2

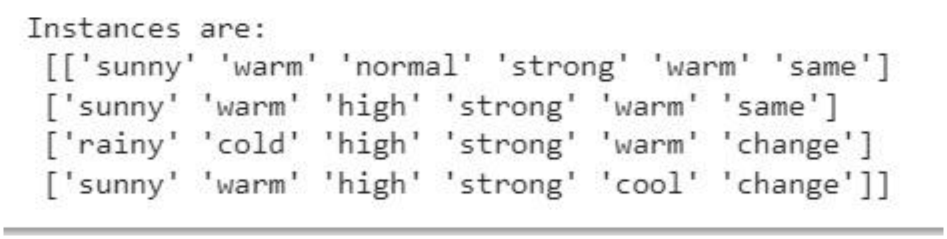
B. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

The practical contains of a write, followed by code and output of the same, along with the observation i.e., the interpretation of the algorithm applied on the dataset. This Practical consists of a write up with the following key points: - 1. What is Candidate Elimination Algorithm.
2. Algorithm of Candidate Elimination Algorithm.
3. Data set used for Candidate Elimination Algorithm.
The Practical is performed in Python. More about the dataset can be seen ahead in this document.

1. What is Candidate Elimination Algorithm.
The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

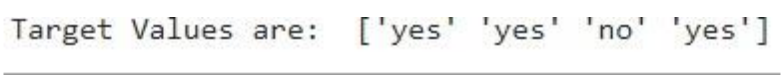
- You can consider this as an extended form of Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as Find-S algorithm (Basically they are generalizing from the specification).
- While the negative example is specified from generalize form.

```
import numpy as np import
pandas as pd
data = pd.read_csv('/content/drive/MyDrive/Data_Science_Demo/Candidate_Elimination.csv'
)
concepts = np.array(data.iloc[:,0:-1]) print("\nInstances
are:\n",concepts)
```



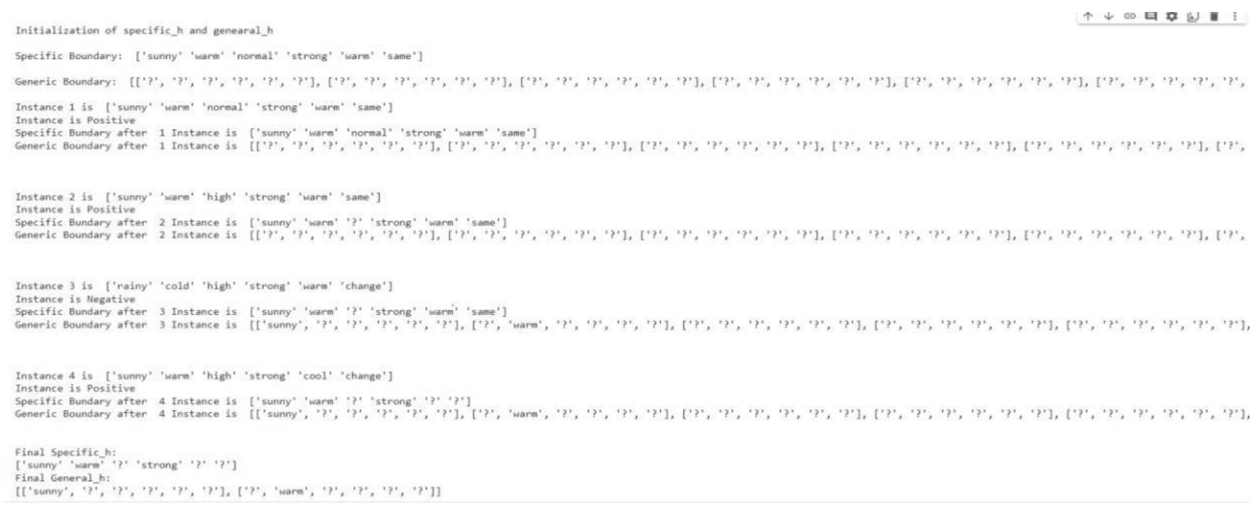
```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

```
target = np.array(data.iloc[:,-1]) print("\nTarget
Values are: ",target)
```



```
Target Values are:  ['yes' 'yes' 'no' 'yes']
```

```
def learn(concepts, target): specific_h
    = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
    range(len(specific_h))] print("\nGeneric Boundary: ",general_h) for i, h in
    enumerate(concepts): print("\nInstance", i+1 , "is ", h) if target[i] == "yes":
    print("Instance is Positive ") for x in range(len(specific_h)):
            if h[x]!= specific_h[x]:
                specific_h[x] ='?'
                general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]

    for i in indices: general_h.remove(['?', '?',
        '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n") print("Final
General_h: ", g_final, sep="\n")
```



```
Initialization of specific_h and genearal_h

Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',


Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',


Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],


Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Practical 3

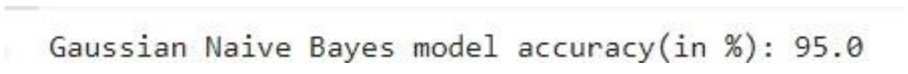
A. Write a program to implement the naive Bayesian Classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

A Naive Bayes classifier is a simple probabilistic classifier that is based on the application of Bayes' theorem with strong (naive) independence assumptions. It is a popular method for classifying text documents, such as spam and non-spam emails.

In a Naive Bayes classifier, the probability of a particular class (e.g., spam or non-spam) is calculated based on the probability of each feature (e.g., a particular word in the email) given that class. The class with the highest probability is then chosen as the predicted class. To build a Naive Bayes classifier, you need to first determine the classes you want to predict (e.g., spam and non-spam) and the features you will use to make the prediction (e.g., the words in the email). You then need to collect a training dataset of labelled examples (i.e., emails that have been manually labelled as spam or non-spam).

Next, you estimate the probability of each class and the probability of each feature given each class using the training data. You can then use these probabilities to classify new examples (e.g., emails that have not been labelled) by calculating the probability of each class given the features of the new example and choosing the class with the highest probability.

```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y into training and testing sets from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)

# training the model on training set from
sklearn.naive_bayes import GaussianNB gnb
= GaussianNB() gnb.fit(X_train, y_train)
# making predictions on the testing set y_pred
= gnb.predict(X_test)
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)
*100)
```



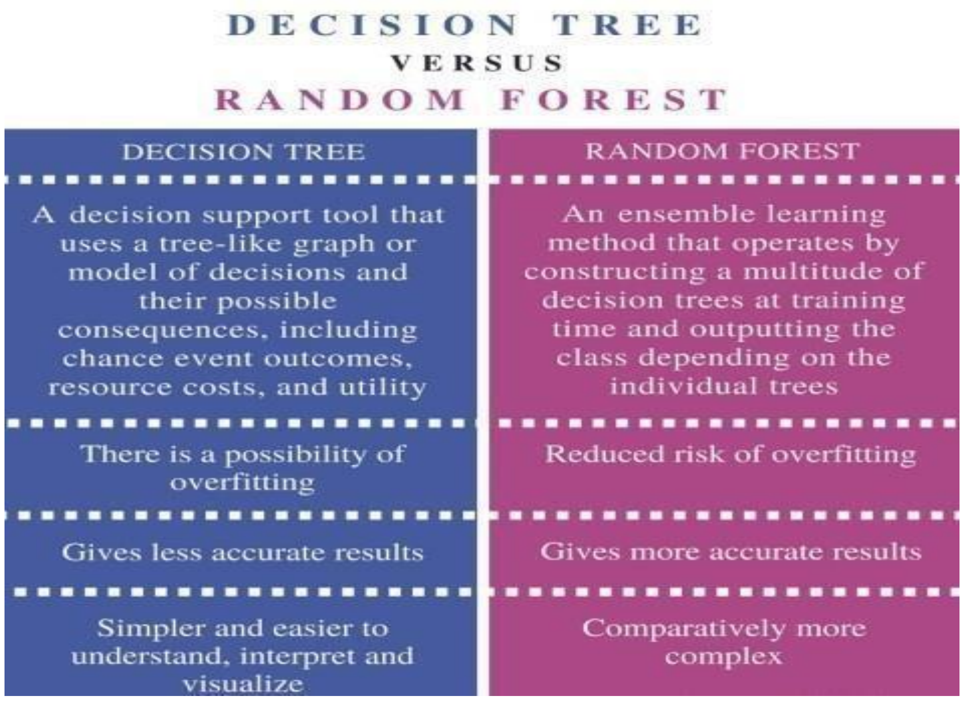
```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

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

The practical contains of a write, followed by code and output of the same, along with the observation i.e., the interpretation of the algorithm applied on the dataset.

This Practical consists of a write up with the following key points: - 1.
What is Decision Tree and Random Forest.
2. Difference between decision tree and random forest.
3. Algorithm of decision tree and random forest
4. Data sets used for decision tree and Random Forest

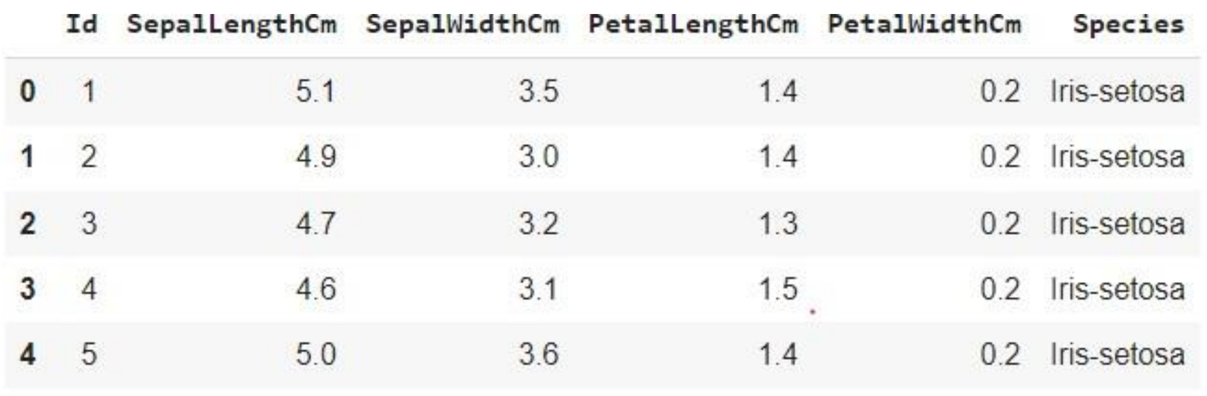
The Practical is performed in Python. The practical uses two different datasets for the sake of executing a similar algorithm for a classification problem with decision tree and regression problem with random forest.



DECISION TREE VERSUS RANDOM FOREST

DECISION TREE	RANDOM FOREST
A decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility	An ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class depending on the individual trees
There is a possibility of overfitting	Reduced risk of overfitting
Gives less accurate results	Gives more accurate results
Simpler and easier to understand, interpret and visualize	Comparatively more complex

```
Decision Tree import pandas as pd import
matplotlib.pyplot as plt # from sklearn import
datasets import numpy as np from
sklearn.model_selection import train_test_split from
sklearn.tree import DecisionTreeClassifier from
sklearn import tree
from sklearn.metrics import classification_report from
sklearn import preprocessing


# import some data to play with iris =
pd.read_csv('/content/drive/MyDrive/Data_Science_Demo/Iris.csv')
iris.head()
```



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris['Species'].value_counts()
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

```
#Complete Iris dataset
label_encoder = preprocessing.LabelEncoder() iris['Species']=
label_encoder.fit_transform(iris['Species'])

iris['Species'].value_counts()
```

```
0    50
1    50
2    50
Name: Species, dtype: int64
```

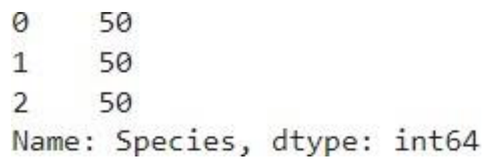
```
x=iris.iloc[:,0:4]
y=iris['Species'] y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

iris['Species'].unique()

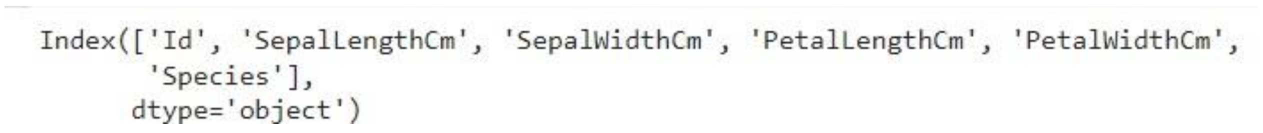
- array([0, 1, 2])

iris.Species.value_counts()



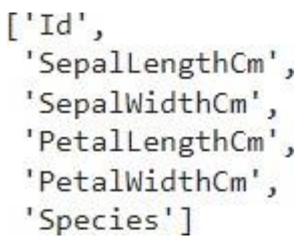
```
0    50
1    50
2    50
Name: Species, dtype: int64
```

iris.columns



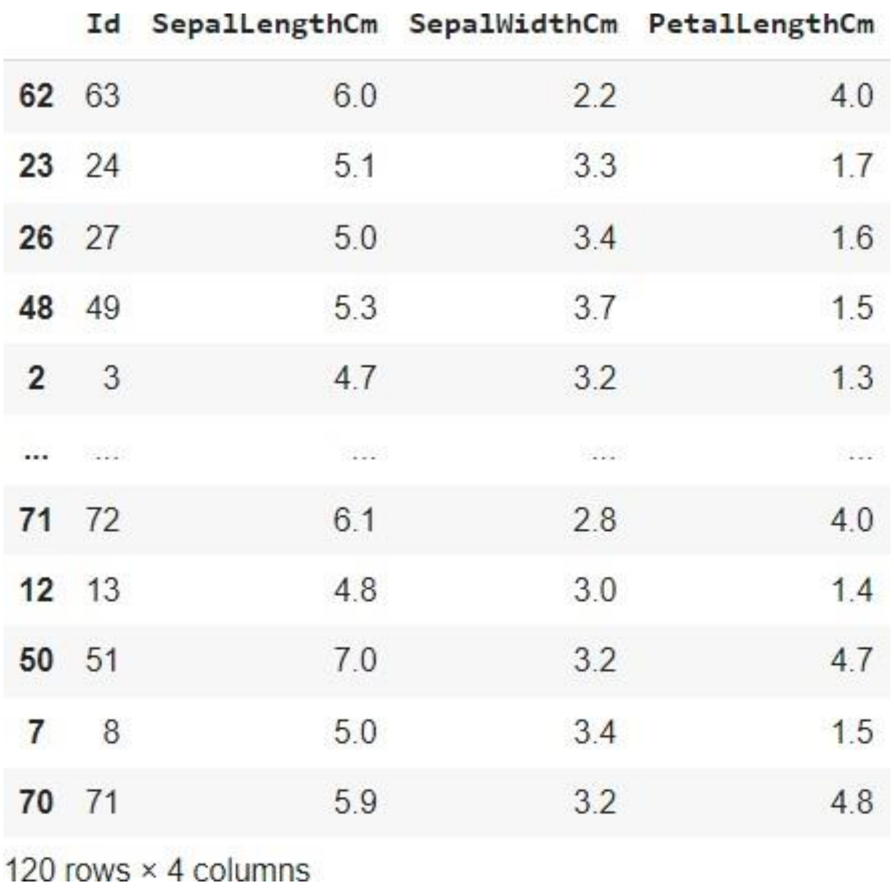
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

colnames = list(iris.columns) colnames



```
['Id',
 'SepalLengthCm',
 'SepalWidthCm',
 'PetalLengthCm',
 'PetalWidthCm',
 'Species']
```

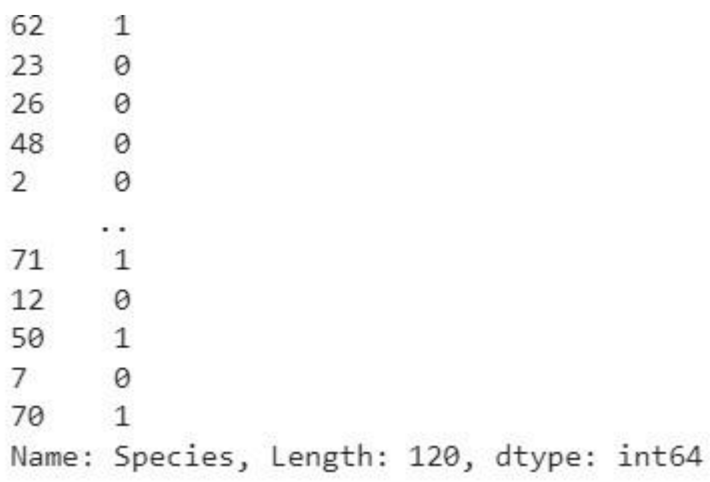
# Splitting data into training and testing data set # from sklearn.model_selection import train_test_split x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=40) x_train



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm
62	63	6.0	2.2	4.0
23	24	5.1	3.3	1.7
26	27	5.0	3.4	1.6
48	49	5.3	3.7	1.5
2	3	4.7	3.2	1.3
...	...	...	...	...
71	72	6.1	2.8	4.0
12	13	4.8	3.0	1.4
50	51	7.0	3.2	4.7
7	8	5.0	3.4	1.5
70	71	5.9	3.2	4.8

120 rows × 4 columns

y_train



```
62    1
23    0
26    0
48    0
2     0
     ..
71    1
12    0
50    1
7     0
70    1
Name: Species, Length: 120, dtype: int64
```

Building Decision Tree Classifier using Entropy Criteria

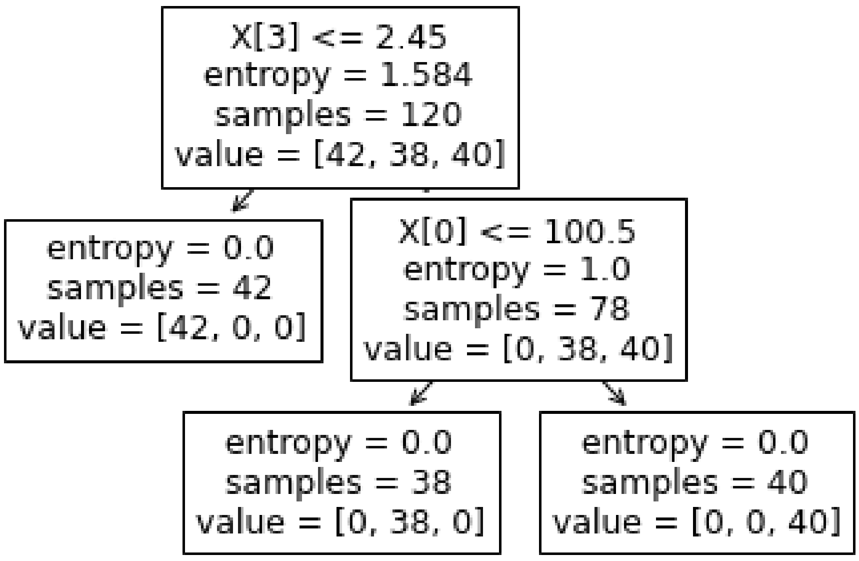
```
model = DecisionTreeClassifier(criterion = 'entropy', max_depth=3)
model.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

model

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

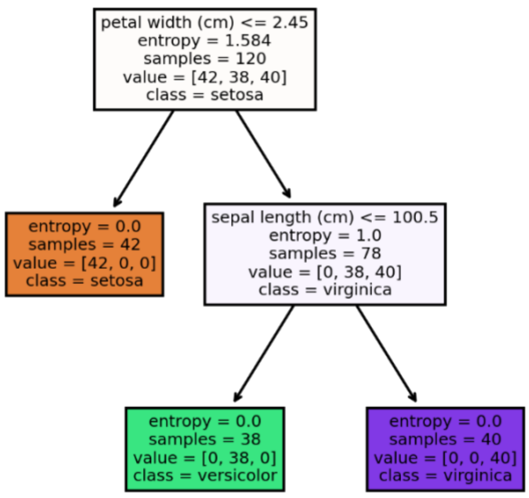
```
#PLot the decision tree from
sklearn import tree
tree.plot_tree(model);
```



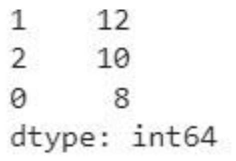
```
y_train.value_counts().keys()
```

```
Int64Index([0, 2, 1], dtype='int64')
```

```
fn=['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)'] #.columns
cn=['setosa', 'versicolor', 'virginica']            #.value_count.keys fig, axes =
plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300) tree.plot_tree(model,
feature_names = fn, class_names=cn, filled = True);
```

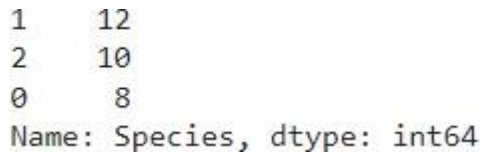


```
#Predicting on test data
preds = model.predict(x_test) # predicting on test data set pd.Series(preds).value_counts()
# getting the count of each category
```



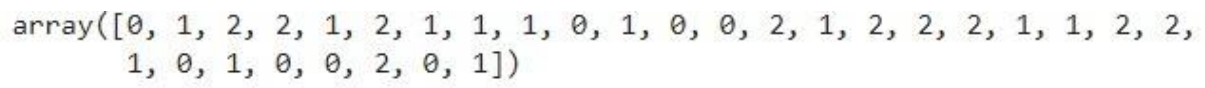
```
1    12
2    10
0     8
dtype: int64
```

```
y_test.value_counts()
```



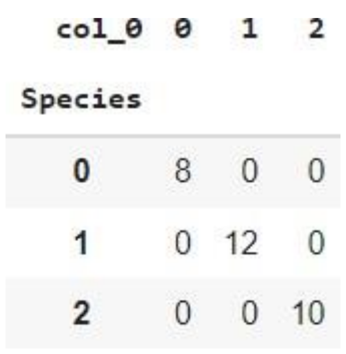
```
1    12
2    10
0     8
Name: Species, dtype: int64
```

```
preds
```



```
array([0, 1, 2, 2, 1, 2, 1, 1, 1, 0, 1, 0, 0, 2, 1, 2, 2, 2, 1, 1, 2, 2,
       1, 0, 1, 0, 0, 2, 0, 1])
```

```
pd.crosstab(y_test,preds) # getting the 2 way table to understand the correct and wrong predic
tions
```



col_0	0	1	2
Species			
0	8	0	0
1	0	12	0
2	0	0	10

```
# Accuracy np.mean(preds==y_test)
```

- 1.0

```
y_test[127:]
```

- Series([], Name: Species, dtype: int64)

Building Decision Tree Classifier (CART) using Gini Criteria

```
from sklearn.tree import DecisionTreeClassifier
model_gini = DecisionTreeClassifier(criterion='gini', max_depth=3)

model_gini.fit(x_train, y_train)
```

- DecisionTreeClassifier(max_depth=3)

```
#Prediction and computing the accuracy
pred=model.predict(x_test)
np.mean(preds==y_test)
```

- 1.0

Decision Tree Regression Example

```
# Decision Tree Regression from sklearn.tree
import DecisionTreeRegressor


array = iris.values
X = array[:,0:3] y
= array[:,3]
X_train, X_test,
y_train, y_test =
train_test_split(X
, y,
test_size=0.33,
random_state=1)


model = DecisionTreeRegressor() model.fit(X_train,
y_train)
```

- DecisionTreeRegressor()

```
#Find the accuracy model.score(X_test,y_test)
```

- 0.956559180939623

Random Forest

```
# Random Forest Classification from pandas import
read_csv from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

filename = 'https://raw.githubusercontent.com/slmsshk/pima-indians-
diabetes.data.csv/main/pima-indians-diabetes.csv'

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class'] dataframe
= read_csv(filename, names=names)
array = dataframe.values
X  =  array[:,0:8]
Y   =   array[:,8]
num_trees  =  100
max_features = 3
kfold = KFold(n_splits=10, random_state=7, shuffle=True) #Bootstrap aggregating (Bagging
)
```

```
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = cross_val_score(model, X, Y, cv=kfold) print(results.mean())
```

- 0.7630211893369788

Practical 4

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

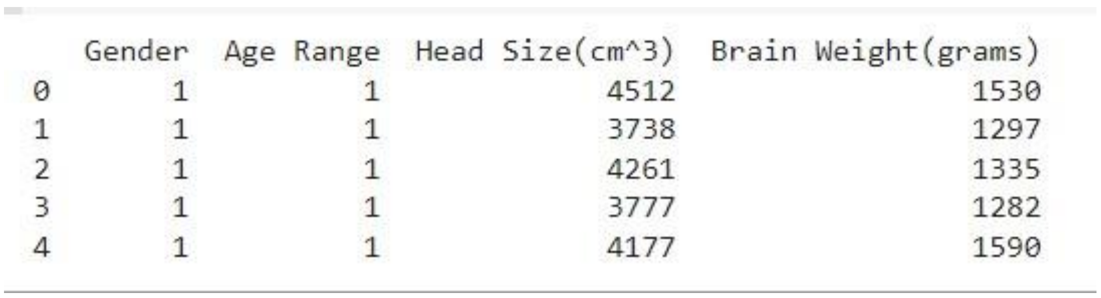
1. What is Least square regression.

The least-squares regression method is a technique commonly used in Regression Analysis. It is a mathematical method used to find the best fit line that represents the relationship between an independent and dependent variable.

Line of best fit is drawn to represent the relationship between two or more variables. To be more specific, the best fit line is drawn across a scatter plot of data points in order to represent a relationship between those data points.

Regression analysis makes use of mathematical methods such as least squares to obtain a definite relationship between the predictor variable (s) and the target variable. The least-squares method is one of the most effective ways used to draw the line of best fit. It is based on the idea that the square of the errors obtained must be minimized to the most possible extent and hence the name least squares method.

```
#Import the required libraries
import numpy as np import
pandas as pd
import matplotlib.pyplot as plt

# Reading Data data =
pd.read_csv('/content/drive/MyDrive/Data_Science_Demo/headbrain.csv')
print(data.head())
```



```
   Gender  Age Range  Head Size(cm^3)  Brain Weight(grams)
0       1          1             4512                 1530
1       1          1             3738                 1297
2       1          1             4261                 1335
3       1          1             3777                 1282
4       1          1             4177                 1590
```

print(data.shape)

```
(237, 4)
```

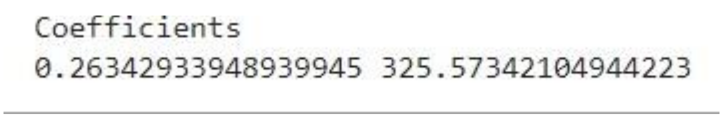
# Coomputing X and Y
X = data['Head Size(cm^3)'].values
Y = data['Brain Weight(grams)'].values """Next, in order to calculate
   the slope and y-
intercept we first need to compute the means of ‘x’ and ‘y’. This can be done as shown below
:"""

# Mean X and Y
mean_x = np.mean(X)

mean_y = np.mean(Y)

# Total number of values n
= len(X)
# Using the formula to calculate 'm' and 'c'
numer = 0
denom = 0 for i
in range(n):
  numer += (X[i] - mean_x) * (Y[i] -
  mean_y) denom += (X[i] - mean_x) ** 2 m
  = numer / denom
  c = mean_y - (m * mean_x)

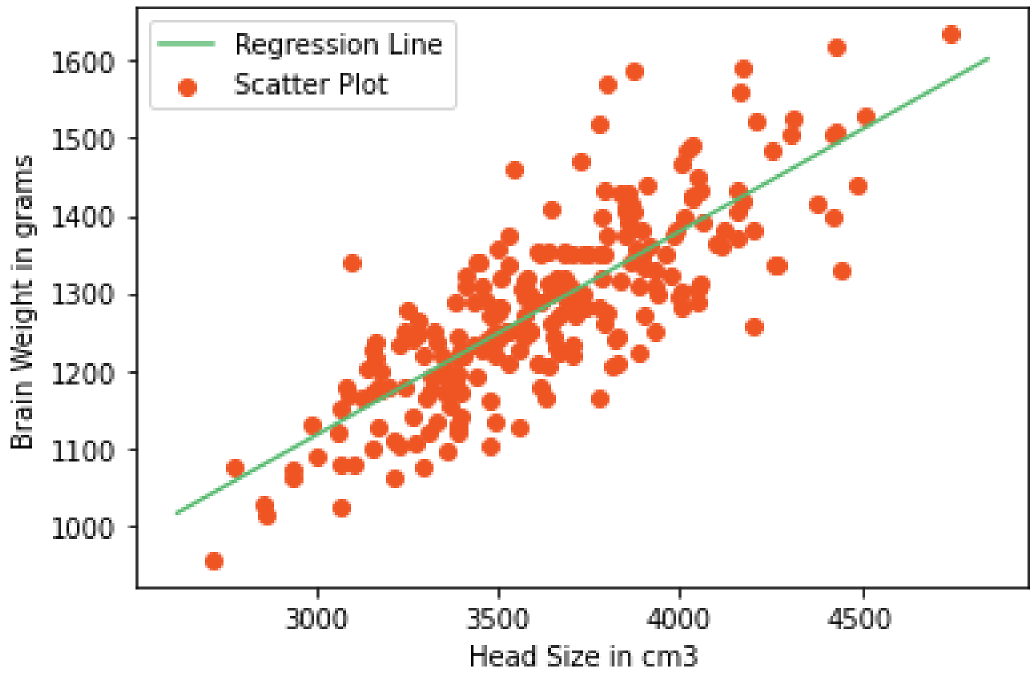
# Printing coefficients
print("Coefficients")
print(m, c)



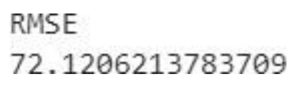
```
Coefficients
0.26342933948939945 325.57342104944223
```

# Plotting Values and Regression Line max_x
= np.max(X) + 100
min_x = np.min(X) - 100

```
# Calculating line values x and y x =
np.linspace(min_x, max_x, 1000)
y = c + m * x

# Ploting Line
plt.plot(x, y, color='#58b970', label='Regression Line')
# Ploting Scatter Points plt.scatter(X, Y,
c='#ef5423', label='Scatter Plot')

plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend() plt.show()
```

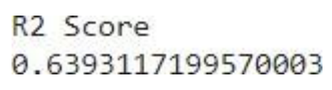


```
# Calculating Root Mean Squares Error
rmse = 0 for i in range(n):
   y_pred = c + m * X[i] rmse
   += (Y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/n)
print("RMSE") print(rmse)
```



```
RMSE
72.1206213783709
```

```
# Calculating R2 Score
ss_tot = 0 ss_res
=  0  for  i  in
range(n):
   y_pred = c + m * X[i] ss_tot
   += (Y[i] - mean_y) ** 2
   ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score") print(r2)
```



```
R2 Score
0.6393117199570003
```

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

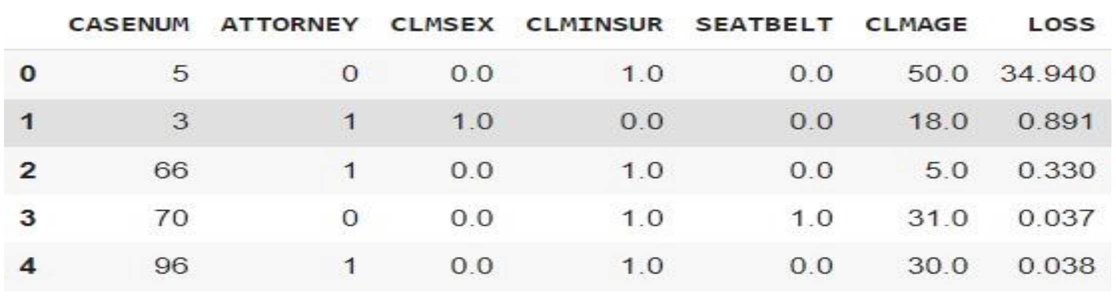
The practical contains of a write, followed by code and output of the same, along with the observation i.e., the interpretation of the algorithm applied on the dataset.

This Practical consists of a write up with the following key points: - 1.
What is Logistic regression?
2. Difference between Logistic and Linear regression.
3. Algorithm of Logistic regression.
4. Data set used for Logistic regression.
The Practical is performed in Python. More about the dataset can be seen ahead in this document.

What is Logistic regression?

○ Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. ○ Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**. ○ Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**. ○ In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

```
import pandas as pd
from sklearn.linear_model import LogisticRegression import
pickle

#Load the data set
claimants = pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/claimants.csv")
claimants.head()
```



	CASENUM	ATTORNEY	CLMSEX	CLMINSUR	SEATBELT	CLMAGE	LOSS
0	5	0	0.0	1.0	0.0	50.0	34.940
1	3	1	1.0	0.0	0.0	18.0	0.891
2	66	1	0.0	1.0	0.0	5.0	0.330
3	70	0	0.0	1.0	1.0	31.0	0.037
4	96	1	0.0	1.0	0.0	30.0	0.038

```
claimants.shape
```

○ (1340, 7)

```
len(claimants['CASENUM'].unique())
```

○ 1283

```
# dropping the case number columns as it is not required
claimants.drop(["CASENUM"],inplace=True,axis = 1)

#Shape of the data set
claimants.shape
```

○ (1340, 6)

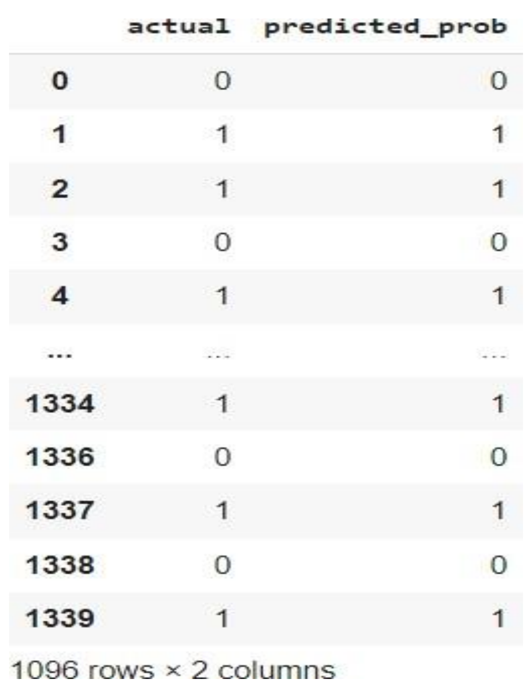
```
# Removing NA values in data set claimants
= claimants.dropna() claimants.shape
```

○ (1096, 6)

```
# Dividing our data into input and output variables
X = claimants.iloc[:,1:]
Y = claimants.iloc[:,0]

#Logistic regression and fit the model classifier
= LogisticRegression()
classifier.fit(X,Y)

# classifier.write_to_pickle('path of file.pkl')
# # classifier.save('Model.hd5')
```

○ LogisticRegression()

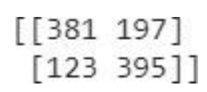
```
# save the model to disk filename =
'finalized_model.sav' pickle.dump(classifier,
open(filename, 'wb'))
#Predict for X dataset pickle.load(open(filename,
'rb'))
# classifier.read_pickle_file('/content/finalized_model.sav') y_pred
= classifier.predict(X)


y_pred_df= pd.DataFrame({'actual': Y,
                        'predicted_prob': classifier.predict(X)})

y_pred_df
```



	actual	predicted_prob
0	0	0
1	1	1
2	1	1
3	0	0
4	1	1
...	...	...
1334	1	1
1336	0	0
1337	1	1
1338	0	0
1339	1	1

1096 rows × 2 columns

```
# Confusion Matrix for the model accuracy from
sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(Y,y_pred)
print (confusion_matrix)
```

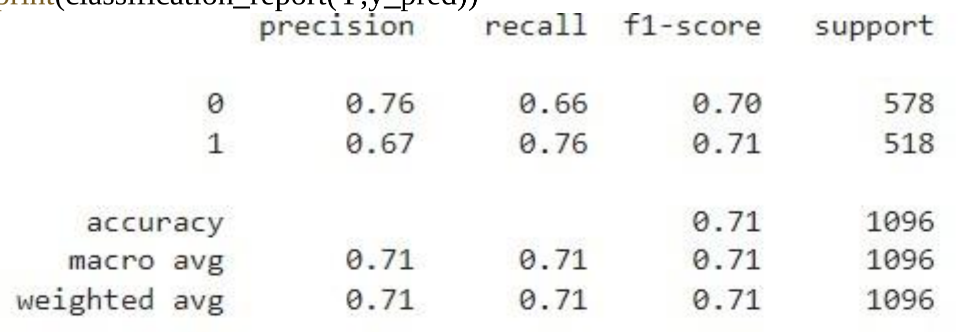


```
[[381 197]
 [123 395]]
```

```
((381+395)/(381+197+123+395))*100
```

- 70.8029197080292

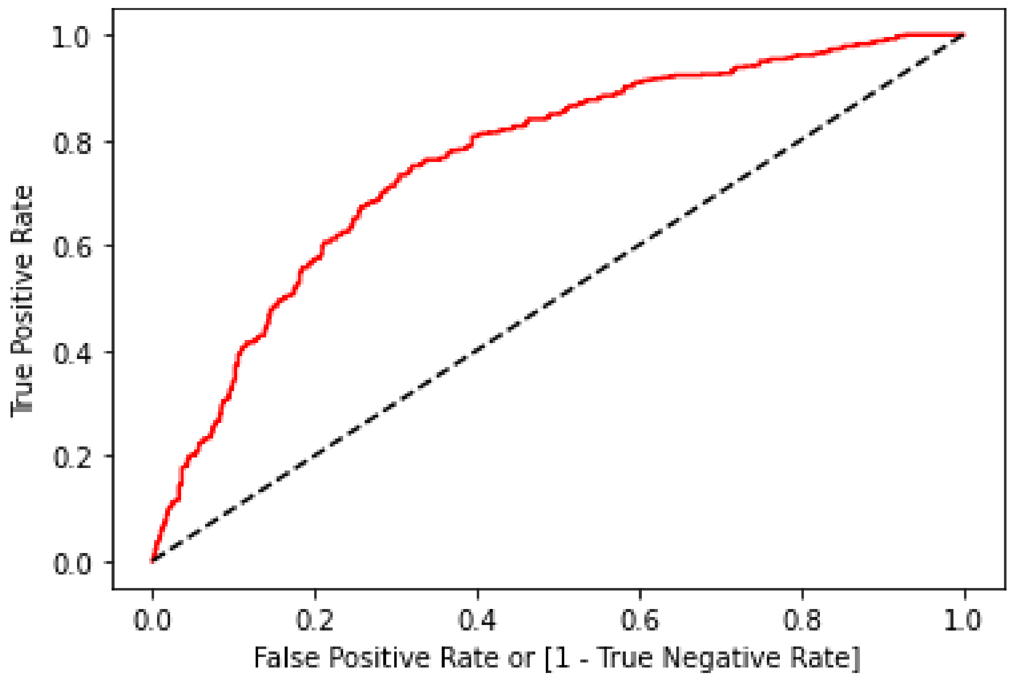
```
#Classification report from sklearn.metrics
import classification_report
print(classification_report(Y,y_pred))
```



```
              precision    recall  f1-score   support

           0       0.76      0.66      0.70       578
           1       0.67      0.76      0.71       518

    accuracy                           0.71      1096
   macro avg       0.71      0.71      0.71      1096
weighted avg       0.71      0.71      0.71      1096
```

```
# ROC Curve
```

```
from sklearn.metrics import roc_curve from
sklearn.metrics import roc_auc_score

fpr, tpr, thresholds = roc_curve(Y, classifier.predict_proba (X)[:,1]) auc

= roc_auc_score(Y, y_pred)

import matplotlib.pyplot as plt
plt.plot(fpr, tpr, color='red', label='logit model ( area = %0.2f)'%auc) plt.plot([0,
1], [0, 1], 'k--')
plt.xlabel('False Positive Rate or [1 - True Negative Rate]') plt.ylabel('True
Positive Rate')
```



auc

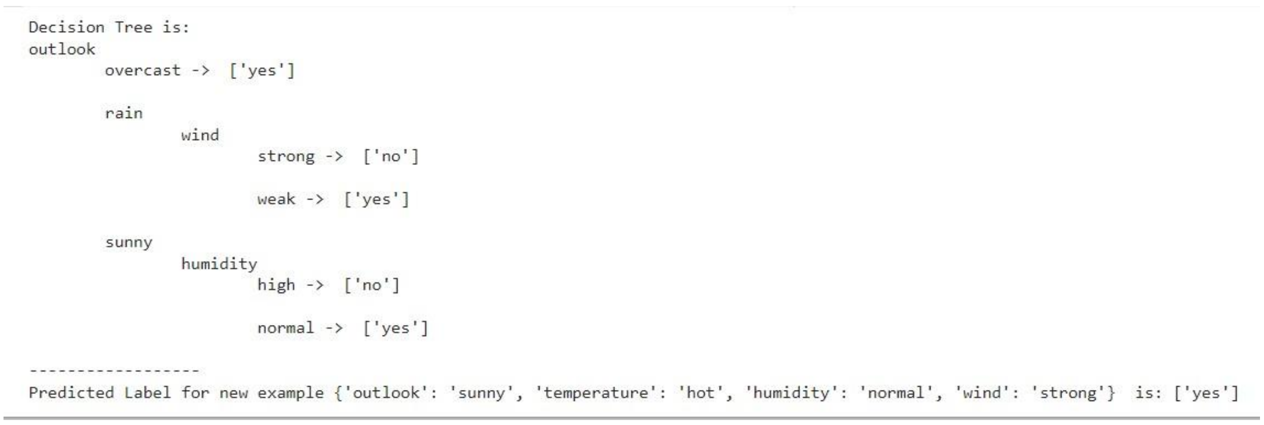
- 0.7108589063606365

Practical 5

A. 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.

What is ID3 Algorithm?
ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node.

```
import pandas as pd import
math
import numpy as np

data = pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/3-dataset.csv") features
= [feat for feat in data]
features.remove("answer")

#Create a class named Node with four members children, value, isLeaf and pred.
class Node:
    def_init_(self): self.children
        = [] self.value = ""
        self.isLeaf = False
        self.pred = ""

#Define a function called entropy to find the entropy of the dataset.
def entropy(examples): pos = 0.0
    neg = 0.0 for _, row in
    examples.iterrows():
        if row["answer"] == "yes": pos
            += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

#Define a function named info_gain to find the gain of the attribute
def info_gain(examples, attr): uniq = np.unique(examples[attr])

    #print ("\n",uniq) gain =
    entropy(examples) #print
    ("\n",gain) for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata) sub_e
        = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

#Define a function named ID3 to get the decision tree for the given dataset def
ID3(examples, attrs):
    root = Node()

    max_gain = 0 max_feat = "" for
    feature in attrs: #print ("\n",examples)
    gain = info_gain(examples, feature) if
    gain > max_gain: max_gain = gain
    max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
    #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata) if
        entropy(subdata) == 0.0:
        newNode = Node()
        newNode.isLeaf = True
        newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
        root.children.append(newNode) else:
            dummyNode = Node()
            dummyNode.value = u new_attrs =
            attrs.copy()
            new_attrs.remove(max_feat) child
            = ID3(subdata, new_attrs)
            dummyNode.children.append(child
            )
            root.children.append(dummyNode)

    return root


#Define a function named printTree to draw the decision tree def
printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value,
    end="") if root.isLeaf:
    print(" -> ", root.pred)
    print() for child in
    root.children:
        printTree(child, depth + 1)

#Define a function named classify to classify the new example def
classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
                exit
            else:
                classify (child.children[0], new)

#Finally, call the ID3, printTree and classify functions
root = ID3(data, features) print("Decision Tree is:")
printTree(root)
print (" ----------------- ")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"} classify
(root, new)
```



```
Decision Tree is:
outlook
        overcast ->  ['yes']

        rain
                wind
                        strong ->  ['no']

                        weak ->  ['yes']

        sunny
                humidity
                        high ->  ['no']

                        normal ->  ['yes']

------------------
Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong'}  is: ['yes']
```

B. Write a program to implement K-Nearest Neighbour algorithm to classify the iris dataset.

The practical contains of a write-up, followed by code and output of the same, along with the observation i.e., the interpretation of the algorithm applied on the dataset.

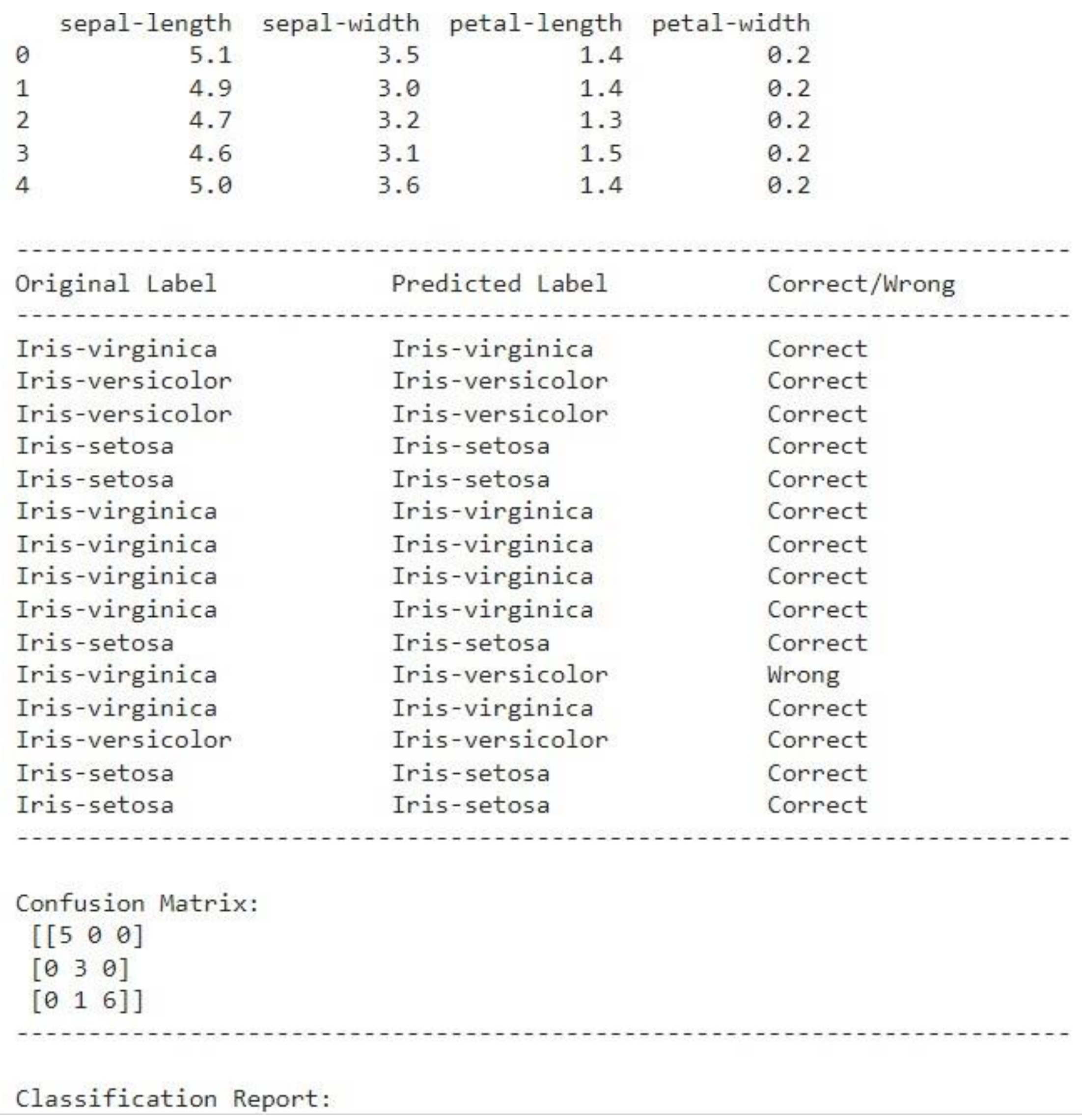
This Practical consists of a write up with the following key points: - 1.
What is K-Nearest Neighbour?
2. Algorithm of K-Nearest Neighbour.
3. Data set used for K-Nearest Neighbour.
The Practical is performed in Python. More about the dataset can be seen ahead in this document.

What is K-Nearest Neighbour?

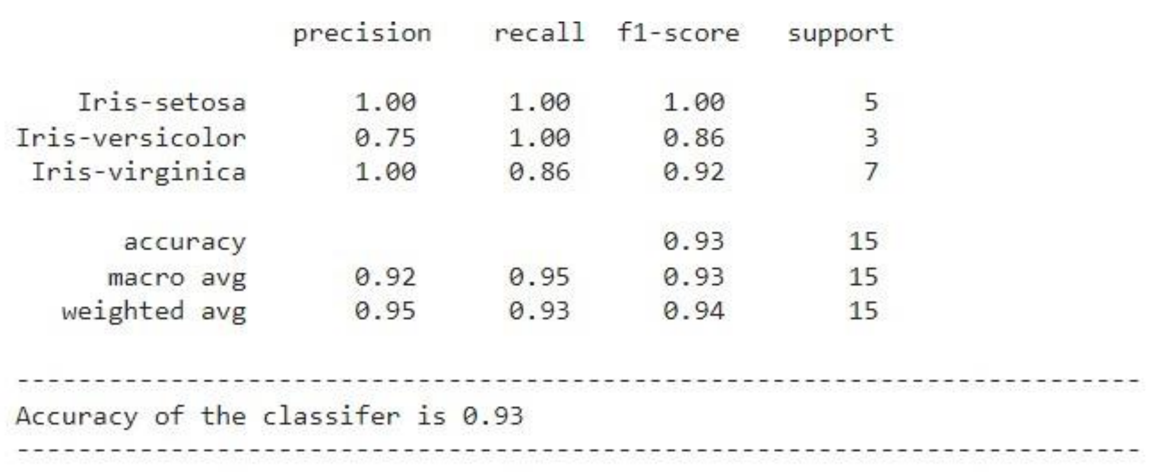
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. ○ K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

```
import numpy as np import pandas as pd from sklearn.neighbors import
KNeighborsClassifier from sklearn.model_selection import
train_test_split from sklearn import metrics names = ['sepal-length',
'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/8-
dataset.csv", names=names) X = dataset.iloc[:, :-1] y = dataset.iloc[:, -
1] print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)
```

```
  classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain) ypred
  = classifier.predict(Xtest)

  i = 0
print ("\n_______________________________________________________")
  print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
  print ("_____") for label in ytest:
      print ('%-25s %-25s' % (label, ypred[i]), end="")
      if (label == ypred[i]): print (' %-25s' %
      ('Correct'))
      else:
          print (' %-25s' % ('Wrong'))
      i = i + 1
print ("_______________________________________________________")
  print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("_______________________________________________________")
  print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("_______________________________________________________")
  print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred)) print
  ("___")
```



```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2

-----------------------------------------------------------------------
Original Label             Predicted Label            Correct/Wrong
-----------------------------------------------------------------------
Iris-virginica             Iris-virginica             Correct
Iris-versicolor            Iris-versicolor            Correct
Iris-versicolor            Iris-versicolor            Correct
Iris-setosa                Iris-setosa                Correct
Iris-setosa                Iris-setosa                Correct
Iris-virginica             Iris-virginica             Correct
Iris-virginica             Iris-virginica             Correct
Iris-virginica             Iris-virginica             Correct
Iris-virginica             Iris-virginica             Correct
Iris-setosa                Iris-setosa                Correct
Iris-virginica             Iris-versicolor            Wrong
Iris-virginica             Iris-virginica             Correct
Iris-versicolor            Iris-versicolor            Correct
Iris-setosa                Iris-setosa                Correct
Iris-setosa                Iris-setosa                Correct
-----------------------------------------------------------------------

Confusion Matrix:
 [[5 0 0]
 [0 3 0]
 [0 1 6]]
-----------------------------------------------------------------------

Classification Report:
```



```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         5
Iris-versicolor       0.75      1.00      0.86         3
 Iris-virginica       1.00      0.86      0.92         7

       accuracy                           0.93        15
      macro avg       0.92      0.95      0.93        15
   weighted avg       0.95      0.93      0.94        15

-----------------------------------------------------------------------
Accuracy of the classifer is 0.93
-----------------------------------------------------------------------
```

Practical 6

A. Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.

What is Euclidean Distance Method, Test Score and Confusion Matrix?

Euclidean distance method is a method for measuring the distance between two points in Euclidean space. Euclidean space is a mathematical space in which the distance between two points is the length of the shortest path between them. The Euclidean distance between two points with coordinates (x1, y1) and (x2, y2) is calculated as follows:

distance = sqrt((x2-x1)^2 + (y2-y1)^2)

The Euclidean distance method is often used in machine learning as a way to measure the similarity between two points. For example, it can be used to measure the distance between two points in a feature space, where each point represents a different object and the features of the object are the coordinates of the point in the space.

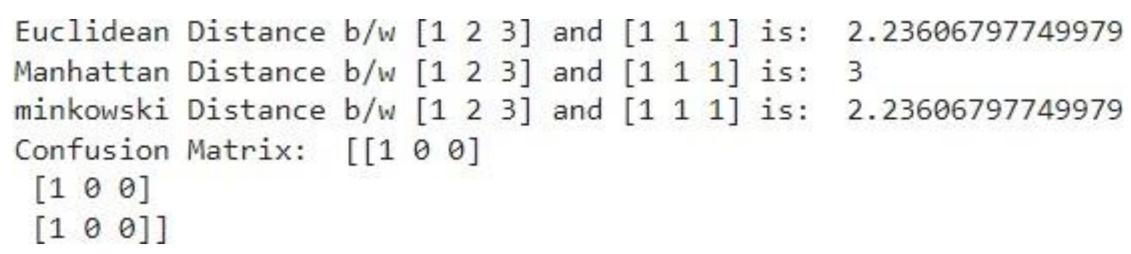
A test score is a measure of how well a machine learning model is able to make predictions on unseen data. In order to evaluate a model's performance, it is common to split the available data into a training set and a test set. The model is trained on the training set, and then its performance is evaluated on the test set. The test score is a metric that summarizes the model's performance on the test set. Common test scores include accuracy (the proportion of correct predictions), precision (the proportion of positive predictions that are actually positive), and recall (the proportion of actual positive cases that were correctly predicted as positive).

A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data. It is a table of counts, where the rows represent the true classes of the examples, and the columns represent the predicted classes of the examples. Each cell in the table contains the count of examples that have a particular true and predicted class. For example, in a binary classification problem with classes "positive" and "negative", a confusion matrix might look like this:

	Predicted Positive	**Predicted Negative**
True Positive	TP	FN
True Negative	FP	TN

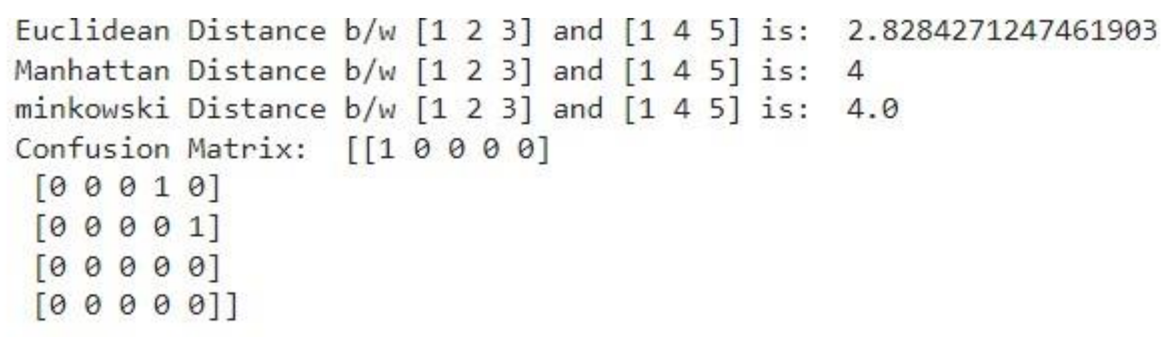
Here, TP stands for true positive, TN stands for true negative, FP stands for false positive, and FN stands for false negative. The rows of the matrix correspond to the true classes of the examples, and the columns correspond to the predicted classes. The diagonal elements of the matrix (TP and TN) represent the number of correctly classified examples, while the off-diagonal elements (FP and FN) represent the number of misclassified examples. The confusion matrix is a useful tool for understanding the strengths and weaknesses of a classification model, and for comparing the performance of different models. import numpy as np
from sklearn.metrics import confusion_matrix from
scipy.spatial import distance

```
point1 = np.array((1, 2, 3))

point2 = np.array((1, 1, 1)) point3
= np.array((1, 4, 5))

euclidean_distance = distance.euclidean(point1,point2) print('Euclidean
Distance b/w', point1, 'and', point2, 'is: ', euclidean_distance)

manhattan_distance = distance.cityblock(point1,point2) print('Manhattan
Distance b/w', point1, 'and', point2, 'is: ', manhattan_distance)

minkowski_distance = distance.minkowski(point1,point2, p=2) print('minkowski
Distance b/w', point1, 'and', point2, 'is: ', minkowski_distance)

print("Confusion Matrix: ",confusion_matrix(point1, point2))
```



```
Euclidean Distance b/w [1 2 3] and [1 1 1] is:  2.23606797749979
Manhattan Distance b/w [1 2 3] and [1 1 1] is:  3
minkowski Distance b/w [1 2 3] and [1 1 1] is:  2.23606797749979
Confusion Matrix:  [[1 0 0]
 [1 0 0]
 [1 0 0]]
```

```
euclidean_distance = distance.euclidean(point1,point3) print('Euclidean
Distance b/w', point1, 'and', point3, 'is: ', euclidean_distance)

manhattan_distance = distance.cityblock(point1,point3) print('Manhattan
Distance b/w', point1, 'and', point3, 'is: ', manhattan_distance)

minkowski_distance = distance.minkowski(point1,point3, p=1) print('minkowski
Distance b/w', point1, 'and', point3, 'is: ', minkowski_distance)

print("Confusion Matrix: ",confusion_matrix(point1, point3))
```



```
Euclidean Distance b/w [1 2 3] and [1 4 5] is:  2.8284271247461903
Manhattan Distance b/w [1 2 3] and [1 4 5] is:  4
minkowski Distance b/w [1 2 3] and [1 4 5] is:  4.0
Confusion Matrix:  [[1 0 0 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

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

In machine learning, k-means is a clustering algorithm that is used to partition a dataset into k clusters, where k is a user-specified number. The goal of the algorithm is to minimize the sum of the distances between each data point and the centroid (mean) of the cluster to which it belongs.

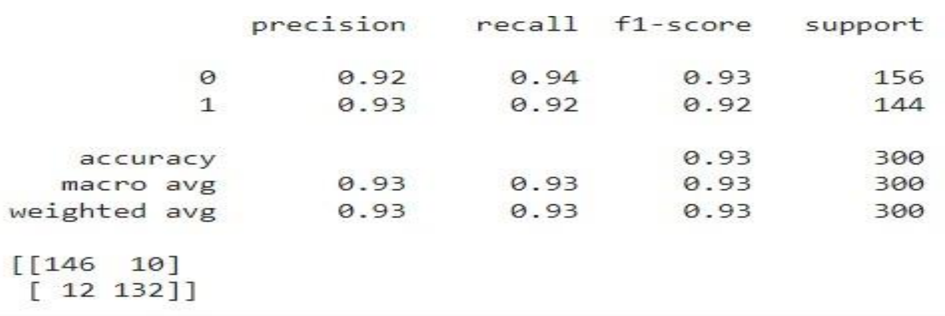
To perform k-means clustering, you first need to specify the number of clusters you want to find (k) and initialize the centroids of the clusters randomly. Then, you iterate over the following two steps until convergence:

1. Assign each data point to the cluster whose centroid it is closest to (according to a distance measure such as Euclidean distance).
2. Recompute the centroids of the clusters as the mean of the data points assigned to each cluster.

The algorithm converges when the centroids of the clusters do not change between iterations. K-means is a popular and widely used clustering algorithm because it is simple to implement and efficient for large datasets. However, it can be sensitive to the choice of the initial centroids and can sometimes produce suboptimal results.

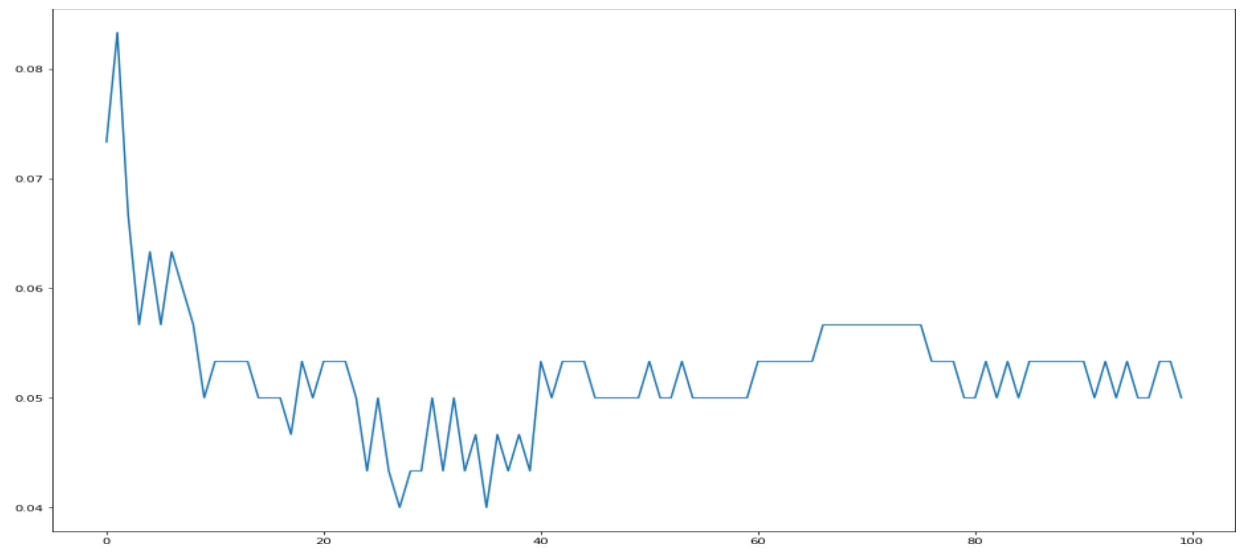
```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt import
seaborn as sns
%matplotlib inline

#Import the data set
raw_data = pd.read_csv('/content/drive/MyDrive/Data_Science_Demo/Classified Data.csv', i
ndex_col = 0)

#Import standardization functions from scikit-learn from
sklearn.preprocessing import StandardScaler
```

```
#Standardize the data set scaler
= StandardScaler()
scaler.fit(raw_data.drop('TARGET CLASS', axis=1))
scaled_features = scaler.transform(raw_data.drop('TARGET CLASS', axis=1)) scaled_data =
pd.DataFrame(scaled_features, columns = raw_data.drop('TARGET CLASS', a
xis=1).columns)

#Split the data set into training data and test data from
sklearn.model_selection import train_test_split x =
scaled_data

y = raw_data['TARGET CLASS']
x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(x, y, test_size = 0
.3)

#Train the model and make predictions from
sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 1)
model.fit(x_training_data, y_training_data)
predictions = model.predict(x_test_data)

#Performance measurement
from sklearn.metrics import classification_report from
sklearn.metrics import confusion_matrix
print(classification_report(y_test_data, predictions))
print(confusion_matrix(y_test_data, predictions))
```



```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       156
           1       0.93      0.92      0.92       144

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300

[[146  10]
 [ 12 132]]
```

```
#Selecting an optimal K value
error_rates = [] for i in
np.arange(1, 101):
    new_model = KNeighborsClassifier(n_neighbors = i)
    new_model.fit(x_training_data, y_training_data)
    new_predictions = new_model.predict(x_test_data)
    error_rates.append(np.mean(new_predictions != y_test_data))
plt.figure(figsize=(16,12)) plt.plot(error_rates)
```



Practical 7

A. Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix

In machine learning, hierarchical clustering is a method of clustering that creates a hierarchy of clusters by building a tree-like structure. There are two main types of hierarchical clustering: agglomerative and divisive.

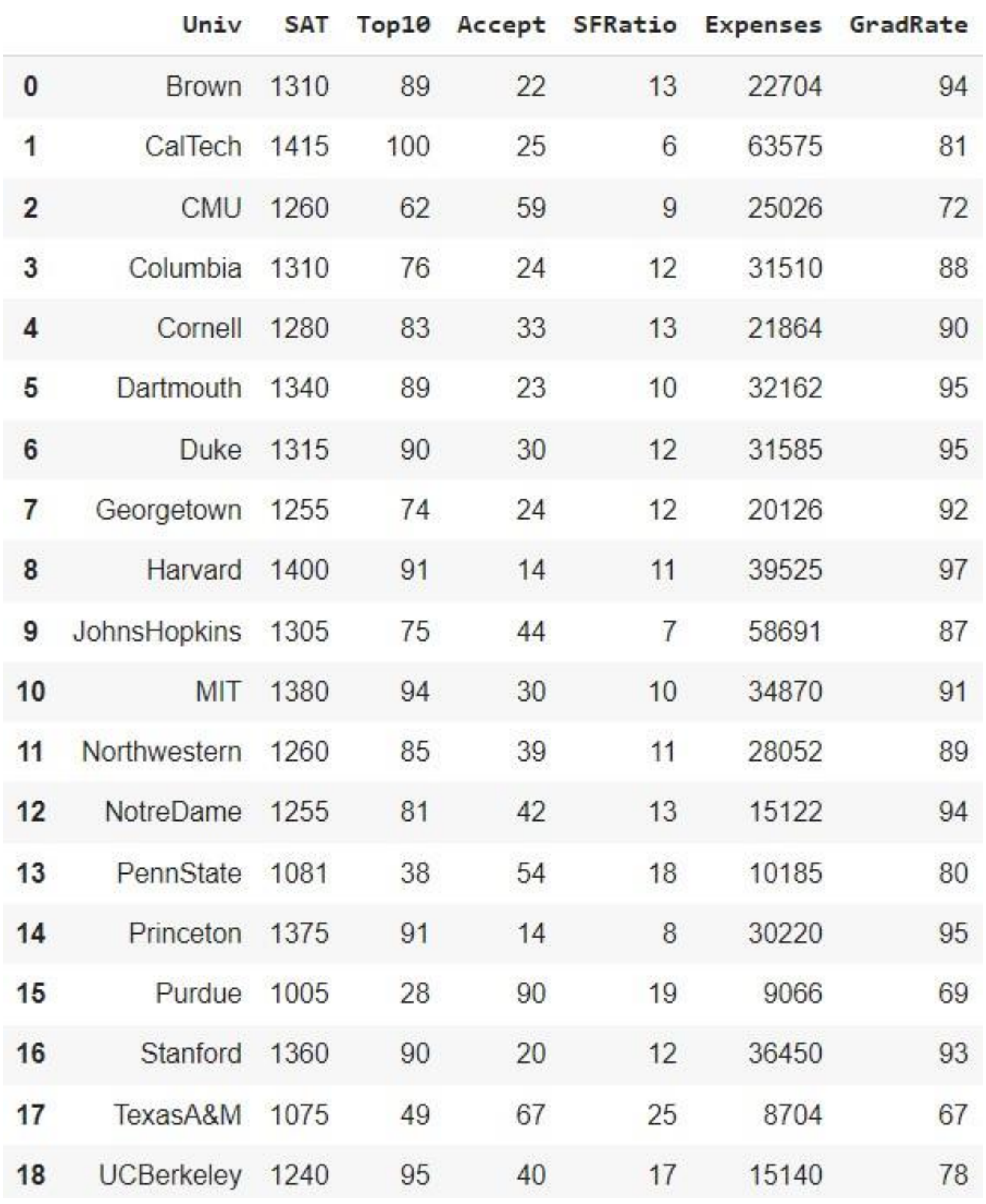
Agglomerative hierarchical clustering starts by treating each data point as a separate cluster and then iteratively merges the closest clusters until all the data points are in the same cluster. This process is controlled by a linkage criterion, which specifies the distance between clusters that should be minimized when merging them.

Divisive hierarchical clustering starts by treating all the data points as a single cluster and then iteratively splits the clusters until each data point is in its own cluster.

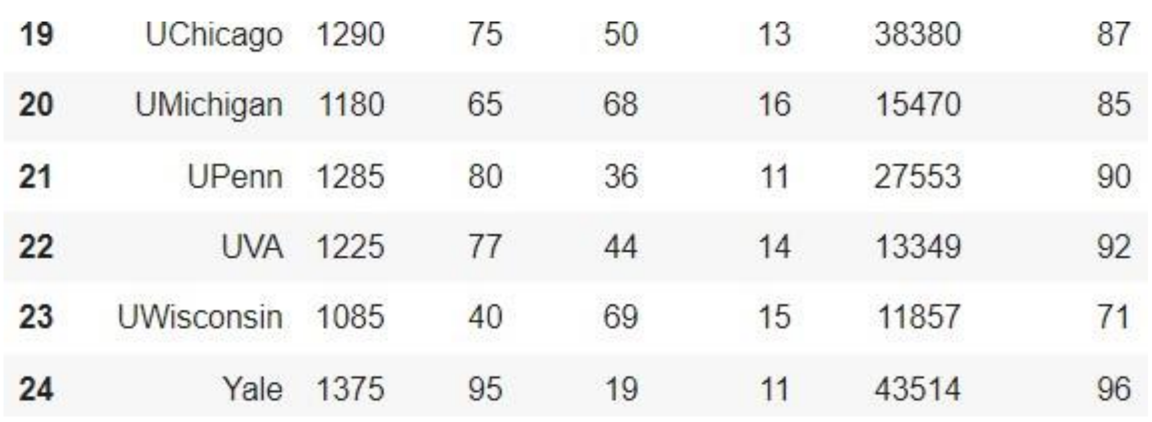
Hierarchical clustering is a useful technique for exploring the structure of a dataset and for visualizing the relationships between the data points. It is also useful for identifying clusters of different sizes and shapes, as it does not require the user to specify the number of clusters in advance. However, it can be slower and more memory-intensive than other clustering algorithms, such as k-means.

```
# import hierarchical clustering libraries import
scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
import numpy as np import pandas as pd
```

```
from matplotlib import pyplot as plt import
seaborn as sn

Univ = pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/Universities.csv") Univ
```

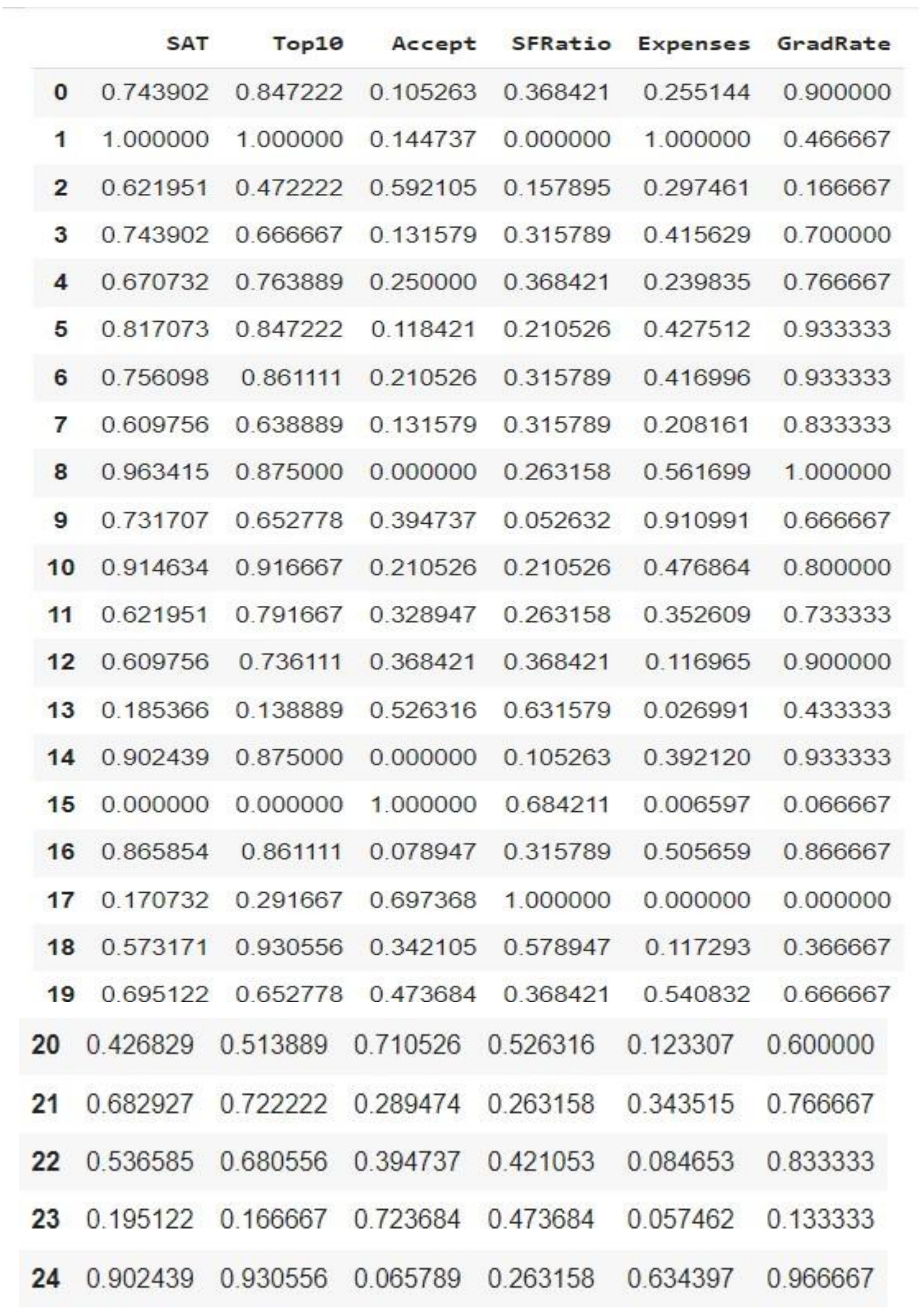


	Univ	SAT	Top10	Accept	SFRatio	Expenses	GradRate
0	Brown	1310	89	22	13	22704	94
1	CalTech	1415	100	25	6	63575	81
2	CMU	1260	62	59	9	25026	72
3	Columbia	1310	76	24	12	31510	88
4	Cornell	1280	83	33	13	21864	90
5	Dartmouth	1340	89	23	10	32162	95
6	Duke	1315	90	30	12	31585	95
7	Georgetown	1255	74	24	12	20126	92
8	Harvard	1400	91	14	11	39525	97
9	JohnsHopkins	1305	75	44	7	58691	87
10	MIT	1380	94	30	10	34870	91
11	Northwestern	1260	85	39	11	28052	89
12	NotreDame	1255	81	42	13	15122	94
13	PennState	1081	38	54	18	10185	80
14	Princeton	1375	91	14	8	30220	95
15	Purdue	1005	28	90	19	9066	69
16	Stanford	1360	90	20	12	36450	93
17	TexasA&M	1075	49	67	25	8704	67
18	UCBerkeley	1240	95	40	17	15140	78



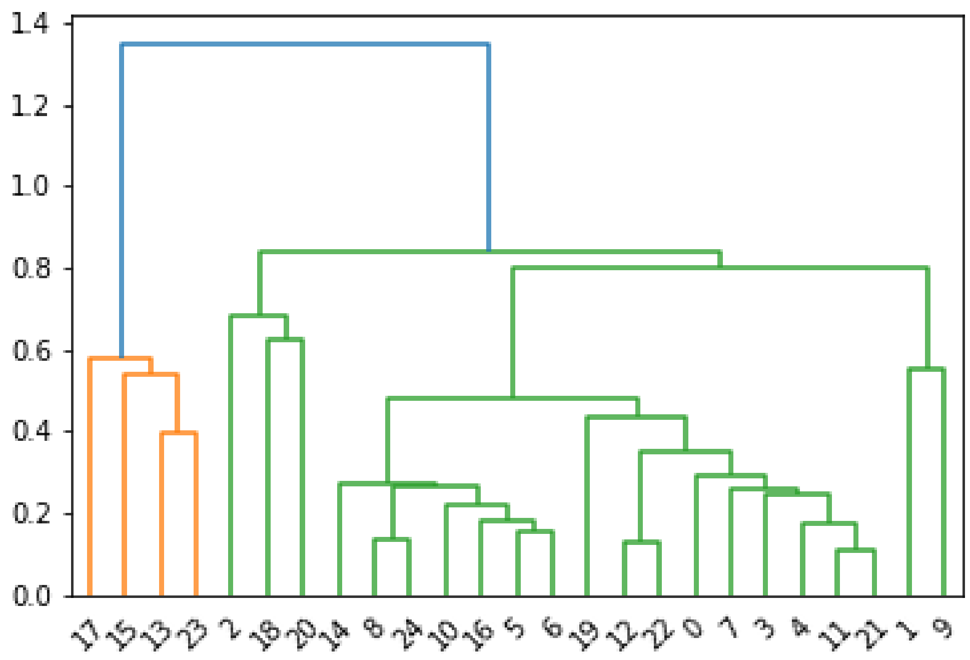
19	UChicago	1290	75	50	13	38380	87
20	UMichigan	1180	65	68	16	15470	85
21	UPenn	1285	80	36	11	27553	90
22	UVA	1225	77	44	14	13349	92
23	UWisconsin	1085	40	69	15	11857	71
24	Yale	1375	95	19	11	43514	96

```
# Normalization function def
norm_func(i):
    x = (i-i.min())/(i.max()-i.min())
    return (x)

# Normalized data frame (considering the numerical part of data) df_norm
= norm_func(Univ.iloc[:,1:])

df_norm
```

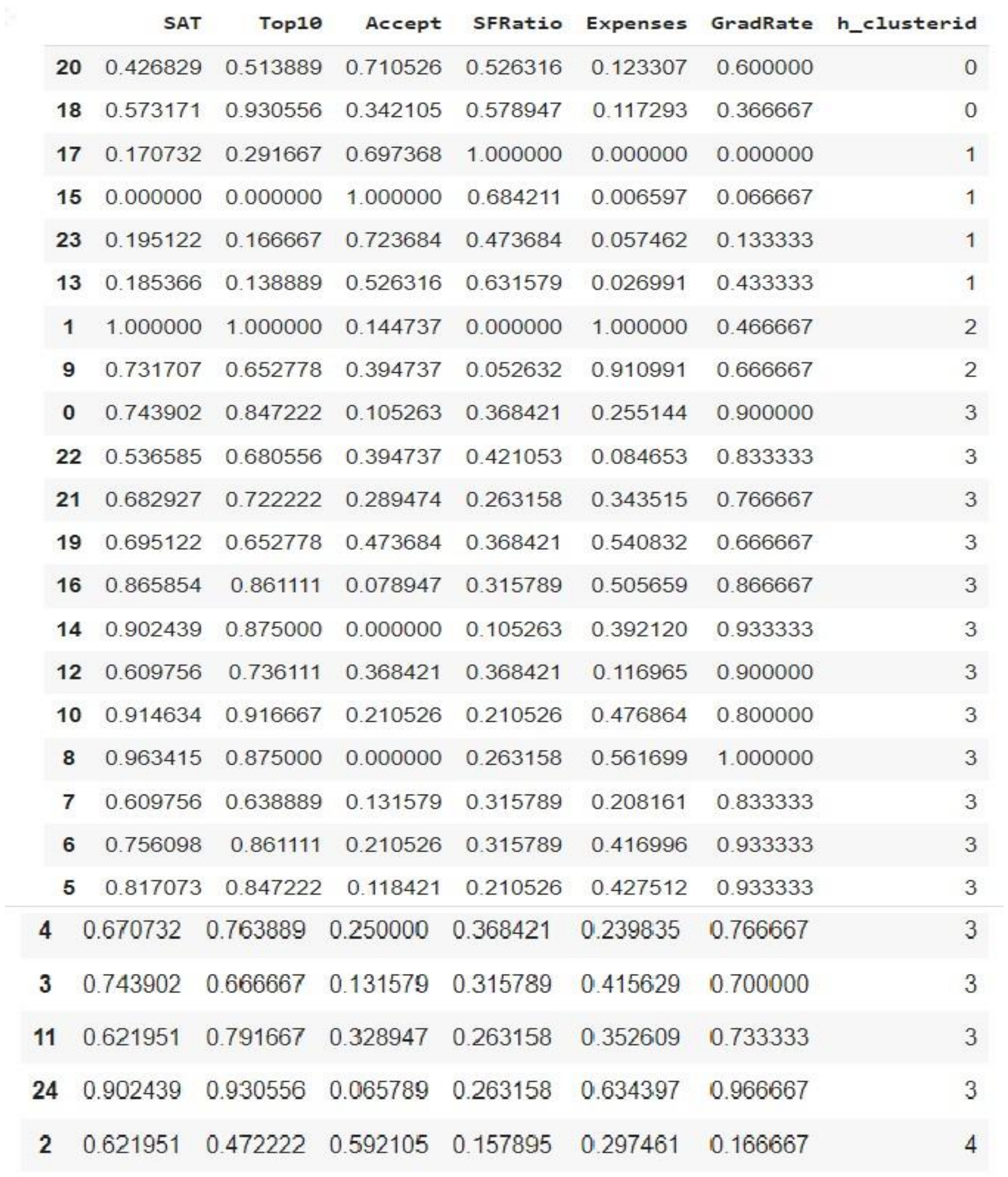


	SAT	Top10	Accept	SFRatio	Expenses	GradRate
0	0.743902	0.847222	0.105263	0.368421	0.255144	0.900000
1	1.000000	1.000000	0.144737	0.000000	1.000000	0.466667
2	0.621951	0.472222	0.592105	0.157895	0.297461	0.166667
3	0.743902	0.666667	0.131579	0.315789	0.415629	0.700000
4	0.670732	0.763889	0.250000	0.368421	0.239835	0.766667
5	0.817073	0.847222	0.118421	0.210526	0.427512	0.933333
6	0.756098	0.861111	0.210526	0.315789	0.416996	0.933333
7	0.609756	0.638889	0.131579	0.315789	0.208161	0.833333
8	0.963415	0.875000	0.000000	0.263158	0.561699	1.000000
9	0.731707	0.652778	0.394737	0.052632	0.910991	0.666667
10	0.914634	0.916667	0.210526	0.210526	0.476864	0.800000
11	0.621951	0.791667	0.328947	0.263158	0.352609	0.733333
12	0.609756	0.736111	0.368421	0.368421	0.116965	0.900000
13	0.185366	0.138889	0.526316	0.631579	0.026991	0.433333
14	0.902439	0.875000	0.000000	0.105263	0.392120	0.933333
15	0.000000	0.000000	1.000000	0.684211	0.006597	0.066667
16	0.865854	0.861111	0.078947	0.315789	0.505659	0.866667
17	0.170732	0.291667	0.697368	1.000000	0.000000	0.000000
18	0.573171	0.930556	0.342105	0.578947	0.117293	0.366667
19	0.695122	0.652778	0.473684	0.368421	0.540832	0.666667
20	0.426829	0.513889	0.710526	0.526316	0.123307	0.600000
21	0.682927	0.722222	0.289474	0.263158	0.343515	0.766667
22	0.536585	0.680556	0.394737	0.421053	0.084653	0.833333
23	0.195122	0.166667	0.723684	0.473684	0.057462	0.133333
24	0.902439	0.930556	0.065789	0.263158	0.634397	0.966667

```
# create dendrogram dendrogram =
sch.dendrogram(sch.linkage(df_norm, method='average'))
```



```
# create clusters hc = AgglomerativeClustering(n_clusters=5, affinity = 'euclidean',
linkage = 'average')

# save clusters for chart y_hc =
hc.fit_predict(df_norm)
Clusters=pd.DataFrame(y_hc,columns=['Clusters'])

df_norm['h_clusterid'] = Clusters df_norm.sort_values("h_clusterid")
```



	SAT	Top10	Accept	SFRatio	Expenses	GradRate	h_clusterid
20	0.426829	0.513889	0.710526	0.526316	0.123307	0.600000	0
18	0.573171	0.930556	0.342105	0.578947	0.117293	0.366667	0
17	0.170732	0.291667	0.697368	1.000000	0.000000	0.000000	1
15	0.000000	0.000000	1.000000	0.684211	0.006597	0.066667	1
23	0.195122	0.166667	0.723684	0.473684	0.057462	0.133333	1
13	0.185366	0.138889	0.526316	0.631579	0.026991	0.433333	1
1	1.000000	1.000000	0.144737	0.000000	1.000000	0.466667	2
9	0.731707	0.652778	0.394737	0.052632	0.910991	0.666667	2
0	0.743902	0.847222	0.105263	0.368421	0.255144	0.900000	3
22	0.536585	0.680556	0.394737	0.421053	0.084653	0.833333	3
21	0.682927	0.722222	0.289474	0.263158	0.343515	0.766667	3
19	0.695122	0.652778	0.473684	0.368421	0.540832	0.666667	3
16	0.865854	0.861111	0.078947	0.315789	0.505659	0.866667	3
14	0.902439	0.875000	0.000000	0.105263	0.392120	0.933333	3
12	0.609756	0.736111	0.368421	0.368421	0.116965	0.900000	3
10	0.914634	0.916667	0.210526	0.210526	0.476864	0.800000	3
8	0.963415	0.875000	0.000000	0.263158	0.561699	1.000000	3
7	0.609756	0.638889	0.131579	0.315789	0.208161	0.833333	3
6	0.756098	0.861111	0.210526	0.315789	0.416996	0.933333	3
5	0.817073	0.847222	0.118421	0.210526	0.427512	0.933333	3
4	0.670732	0.763889	0.250000	0.368421	0.239835	0.766667	3
3	0.743902	0.666667	0.131579	0.315789	0.415629	0.700000	3
11	0.621951	0.791667	0.328947	0.263158	0.352609	0.733333	3
24	0.902439	0.930556	0.065789	0.263158	0.634397	0.966667	3
2	0.621951	0.472222	0.592105	0.157895	0.297461	0.166667	4

B. Implement the Rule based method and test the same

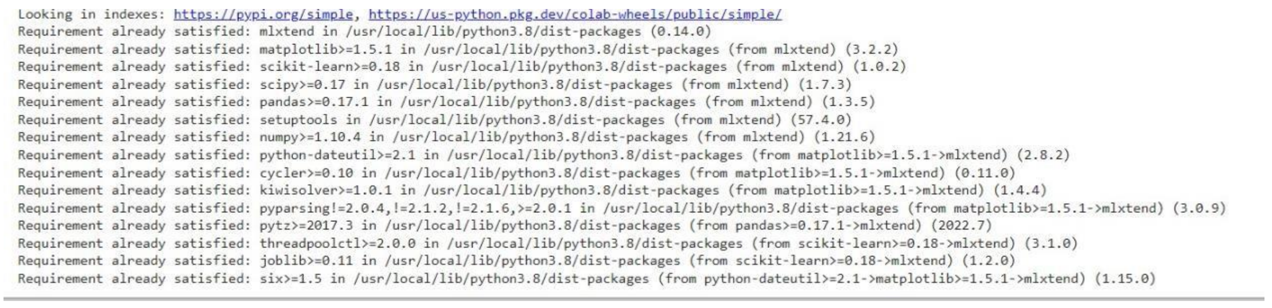
What is Rule Based Method?
Rule-based methods in machine learning involve using a set of pre-defined rules to make decisions or predictions. These rules are typically defined by humans, and the rule-based system makes predictions by applying the rules to the data it is given.

There are several different ways that rule-based systems can be used in machine learning. For example:

- Decision trees: A decision tree is a rule-based system that uses a tree-like structure to make decisions. At each node in the tree, the system considers a different feature of the data and makes a decision based on the value of that feature. The tree structure allows the system to make complex decisions by breaking them down into a series of simple decisions.
- Association rules: Association rule learning is a rule-based method for discovering relationships between variables in large datasets. It is often used in market basket analysis, where the goal is to identify products that are frequently purchased together. For example, an association rule might be "if a customer buys bread, they are also likely to buy butter."
- Expert systems: An expert system is a type of rule-based system that is designed to mimic the decision-making ability of a human expert. Expert systems are often used in domains where there is a lot of expert knowledge and it is not practical to encode this knowledge in a traditional machine learning model.

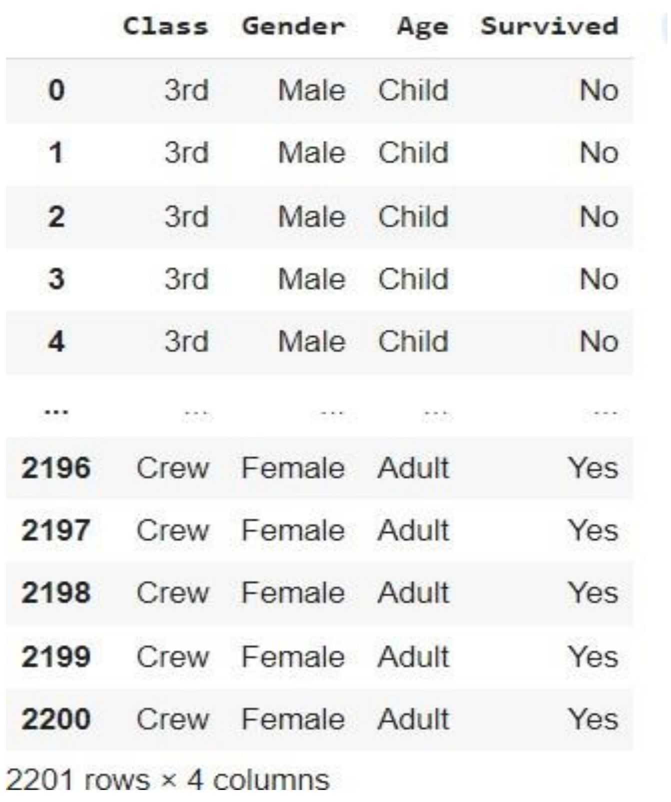
Overall, rule-based methods can be useful in situations where the relationships between variables are well understood and can be explicitly defined in the form of rules. However, they can be limited in their ability to learn from data and adapt to changing circumstances.

```
# Uncomment the below line and Install 'mlxtend' Library if not installed already !pip
install mlxtend
```



```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: mlxtend in /usr/local/lib/python3.8/dist-packages (0.14.0)
Requirement already satisfied: matplotlib>=1.5.1 in /usr/local/lib/python3.8/dist-packages (from mlxtend) (3.2.2)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.8/dist-packages (from mlxtend) (1.0.2)
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.8/dist-packages (from mlxtend) (1.7.3)
Requirement already satisfied: pandas>=0.17.1 in /usr/local/lib/python3.8/dist-packages (from mlxtend) (1.3.5)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from mlxtend) (57.4.0)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.8/dist-packages (from mlxtend) (1.21.6)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib>=1.5.1->mlxtend) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib>=1.5.1->mlxtend) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib>=1.5.1->mlxtend) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib>=1.5.1->mlxtend) (3.0.9)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=0.17.1->mlxtend) (2022.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.18->mlxtend) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.18->mlxtend) (1.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib>=1.5.1->mlxtend) (1.15.0)
```

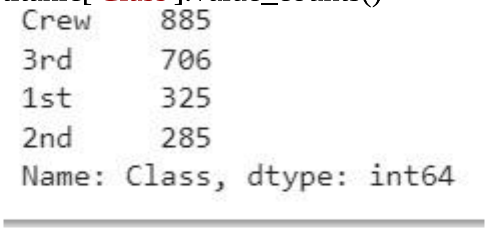
```
import mlxtend import pandas as pd from
mlxtend.frequent_patterns import apriori,association_rules from
mlxtend.preprocessing import TransactionEncoder

titanic = pd.read_csv("/content/drive/MyDrive/Data_Science_Demo/Titanic.csv") titanic
```



	Class	Gender	Age	Survived
0	3rd	Male	Child	No
1	3rd	Male	Child	No
2	3rd	Male	Child	No
3	3rd	Male	Child	No
4	3rd	Male	Child	No
...	...	...	...	...
2196	Crew	Female	Adult	Yes
2197	Crew	Female	Adult	Yes
2198	Crew	Female	Adult	Yes
2199	Crew	Female	Adult	Yes
2200	Crew	Female	Adult	Yes

2201 rows × 4 columns

```
titanic['Class'].value_counts()
```

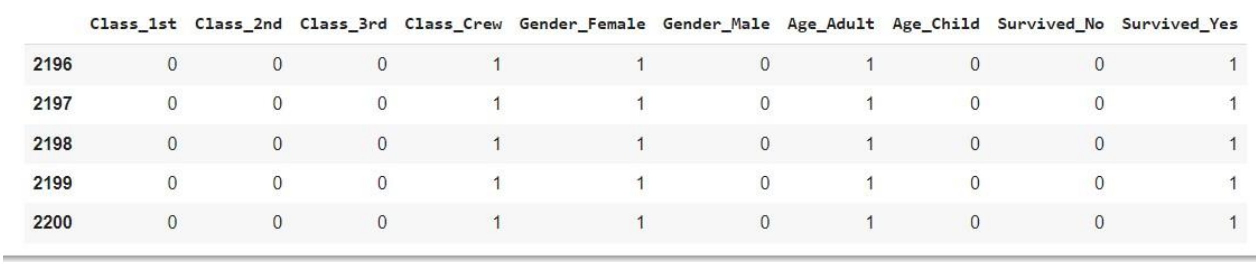


```
Crew     885
3rd      706
1st      325
2nd      285
Name: Class, dtype: int64
```

Pre-Processing As the data is not in transaction formation, We are using transaction Encoder

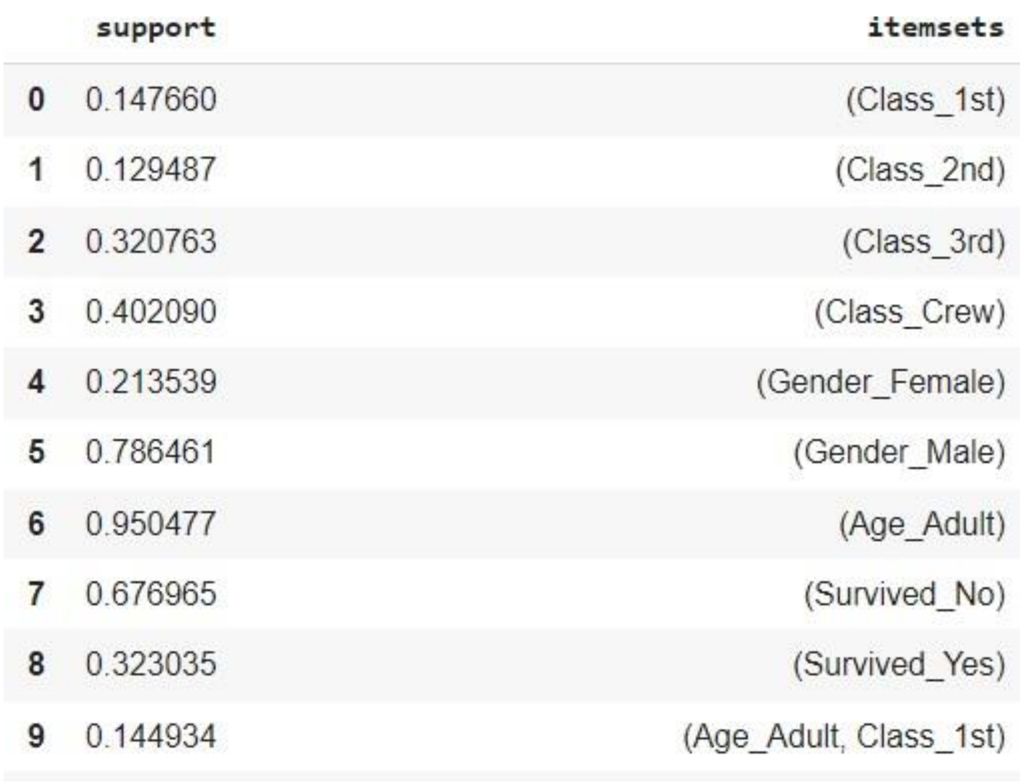
```
df=pd.get_dummies(titanic)
```

df.head() df.tail()



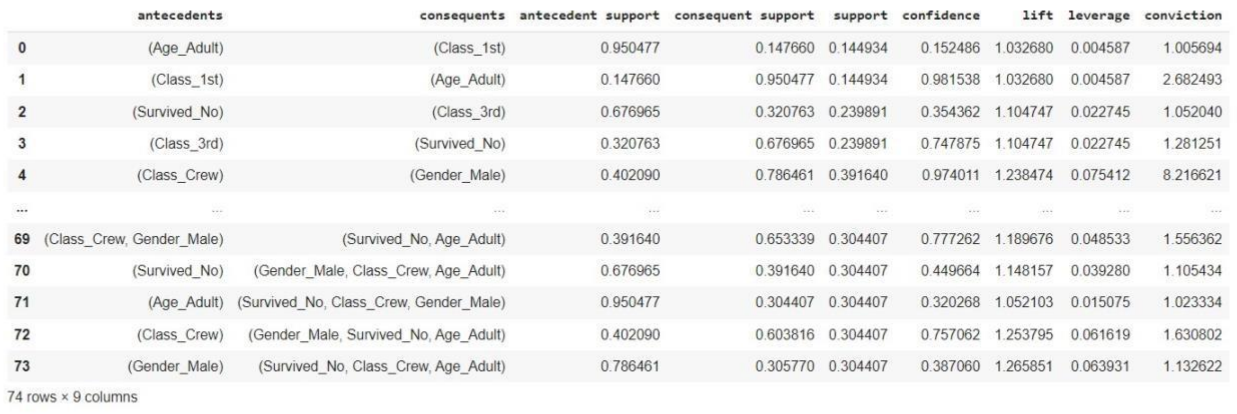
	Class_1st	Class_2nd	Class_3rd	Class_Crew	Gender_Female	Gender_Male	Age_Adult	Age_Child	Survived_No	Survived_Yes
2196	0	0	0	1	1	0	1	0	0	1
2197	0	0	0	1	1	0	1	0	0	1
2198	0	0	0	1	1	0	1	0	0	1
2199	0	0	0	1	1	0	1	0	0	1
2200	0	0	0	1	1	0	1	0	0	1

Apriori Algorithm
frequent_itemsets = apriori(df, min_support=0.1, use_colnames=True) frequent_itemsets



	support	itemsets
0	0.147660	(Class_1st)
1	0.129487	(Class_2nd)
2	0.320763	(Class_3rd)
3	0.402090	(Class_Crew)
4	0.213539	(Gender_Female)
5	0.786461	(Gender_Male)
6	0.950477	(Age_Adult)
7	0.676965	(Survived_No)
8	0.323035	(Survived_Yes)
9	0.144934	(Age_Adult, Class_1st)

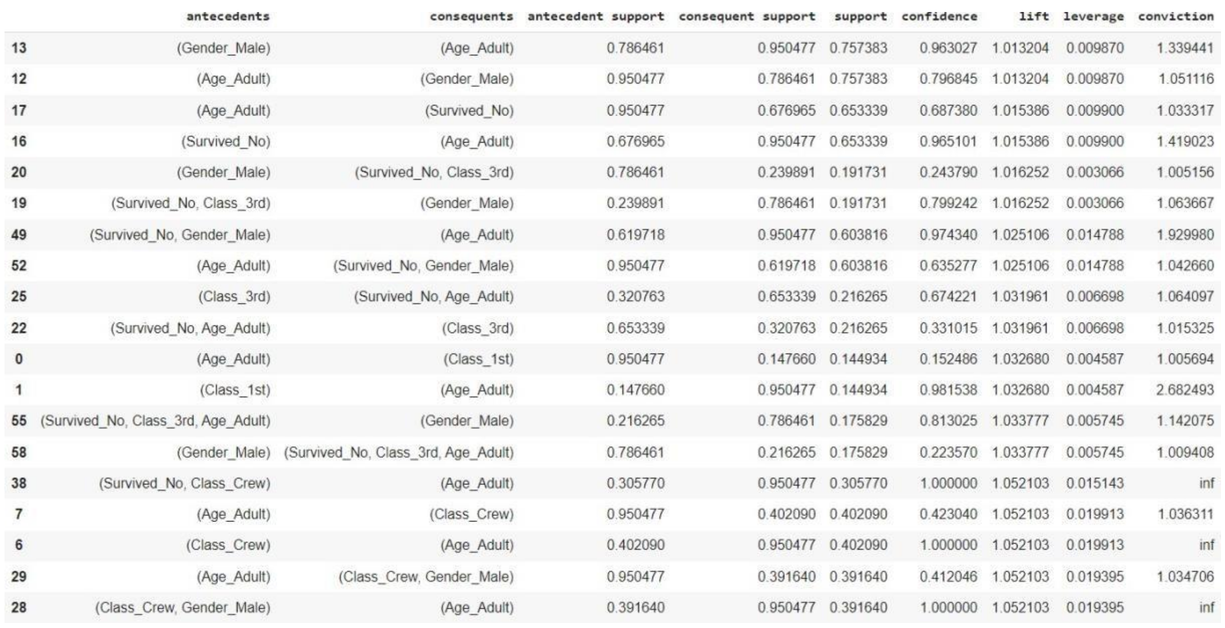
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0) rules
# rules.sort_values('lift',ascending = False)



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Age_Adult)	(Class_1st)	0.950477	0.147660	0.144934	0.152486	1.032680	0.004587	1.005694
1	(Class_1st)	(Age_Adult)	0.147660	0.950477	0.144934	0.981538	1.032680	0.004587	2.682493
2	(Survived_No)	(Class_3rd)	0.676965	0.320763	0.239891	0.354362	1.104747	0.022745	1.052040
3	(Class_3rd)	(Survived_No)	0.320763	0.676965	0.239891	0.747875	1.104747	0.022745	1.281251
4	(Class_Crew)	(Gender_Male)	0.402090	0.786461	0.391640	0.974011	1.238474	0.075412	8.216621
...	...	...	...	...	...	...	...	...	...
69	(Class_Crew, Gender_Male)	(Survived_No, Age_Adult)	0.391640	0.653339	0.304407	0.777262	1.189676	0.048533	1.556362
70	(Survived_No)	(Gender_Male, Class_Crew, Age_Adult)	0.676965	0.391640	0.304407	0.449664	1.148157	0.039280	1.105434
71	(Age_Adult)	(Survived_No, Class_Crew, Gender_Male)	0.950477	0.304407	0.304407	0.320268	1.052103	0.015075	1.023334
72	(Class_Crew)	(Gender_Male, Survived_No, Age_Adult)	0.402090	0.603816	0.304407	0.757062	1.253795	0.061619	1.630802
73	(Gender_Male)	(Survived_No, Class_Crew, Age_Adult)	0.786461	0.305770	0.304407	0.387060	1.265851	0.063931	1.132622

74 rows × 9 columns

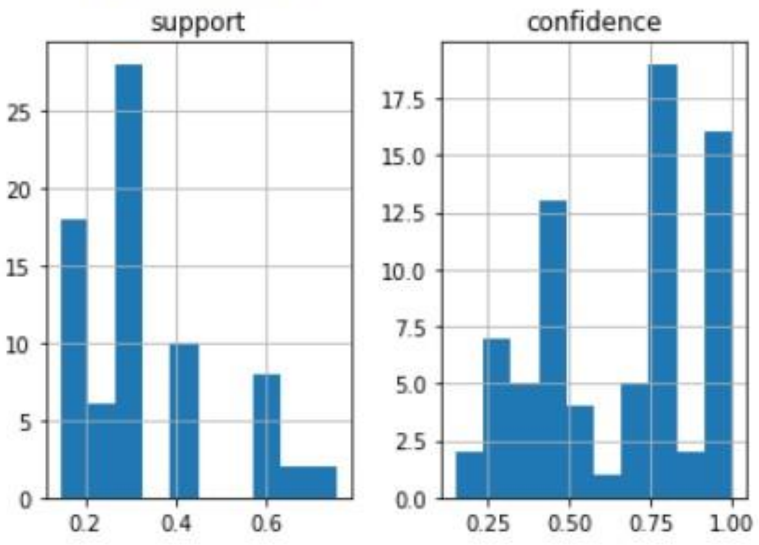
```
rules.sort_values('lift',ascending = True)[0:20]
```



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
13	(Gender_Male)	(Age_Adult)	0.786461	0.950477	0.757383	0.963027	1.013204	0.009870	1.339441
12	(Age_Adult)	(Gender_Male)	0.950477	0.786461	0.757383	0.796845	1.013204	0.009870	1.051116
17	(Age_Adult)	(Survived_No)	0.950477	0.676965	0.653339	0.687380	1.015386	0.009900	1.033317
16	(Survived_No)	(Age_Adult)	0.676965	0.950477	0.653339	0.965101	1.015386	0.009900	1.419023
20	(Gender_Male)	(Survived_No, Class_3rd)	0.786461	0.239891	0.191731	0.243790	1.016252	0.003066	1.005156
19	(Survived_No, Class_3rd)	(Gender_Male)	0.239891	0.786461	0.191731	0.799242	1.016252	0.003066	1.063667
49	(Survived_No, Gender_Male)	(Age_Adult)	0.619718	0.950477	0.603816	0.974340	1.025106	0.014788	1.929980
52	(Age_Adult)	(Survived_No, Gender_Male)	0.950477	0.619718	0.603816	0.635277	1.025106	0.014788	1.042660
25	(Class_3rd)	(Survived_No, Age_Adult)	0.320763	0.653339	0.216265	0.674221	1.031961	0.006698	1.064097
22	(Survived_No, Age_Adult)	(Class_3rd)	0.653339	0.320763	0.216265	0.331015	1.031961	0.006698	1.015325
0	(Age_Adult)	(Class_1st)	0.950477	0.147660	0.144934	0.152486	1.032680	0.004587	1.005694
1	(Class_1st)	(Age_Adult)	0.147660	0.950477	0.144934	0.981538	1.032680	0.004587	2.682493
55	(Survived_No, Class_3rd, Age_Adult)	(Gender_Male)	0.216265	0.786461	0.175829	0.813025	1.033777	0.005745	1.142075
58	(Gender_Male)	(Survived_No, Class_3rd, Age_Adult)	0.786461	0.216265	0.175829	0.223570	1.033777	0.005745	1.009408
38	(Survived_No, Class_Crew)	(Age_Adult)	0.305770	0.950477	0.305770	1.000000	1.052103	0.015143	inf
7	(Age_Adult)	(Class_Crew)	0.950477	0.402090	0.402090	0.423040	1.052103	0.019913	1.036311
6	(Class_Crew)	(Age_Adult)	0.402090	0.950477	0.402090	1.000000	1.052103	0.019913	inf
29	(Age_Adult)	(Class_Crew, Gender_Male)	0.950477	0.391640	0.391640	0.412046	1.052103	0.019395	1.034706
28	(Class_Crew, Gender_Male)	(Age_Adult)	0.391640	0.950477	0.391640	1.000000	1.052103	0.019395	inf

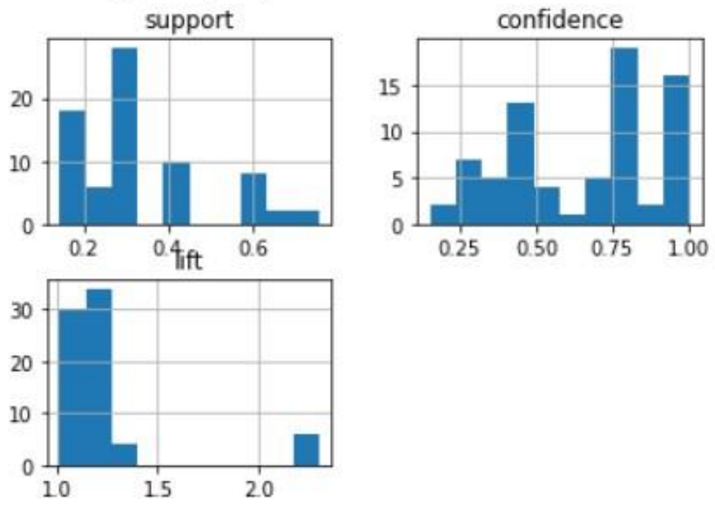
```
rules[['support','confidence']].hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fb9f1488f40>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb9f14683d0>]],
      dtype=object)
```



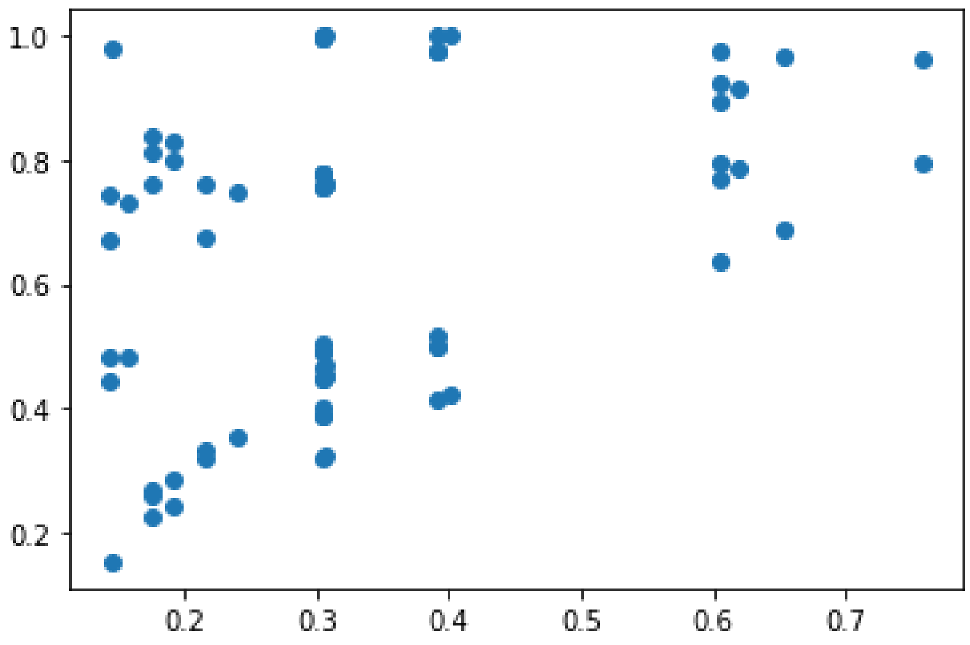
```
rules[['support','confidence','lift']].hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fb9f1394190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb9f0ec12b0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fb9f0e70730>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fb9f0e9db50>]],
      dtype=object)
```

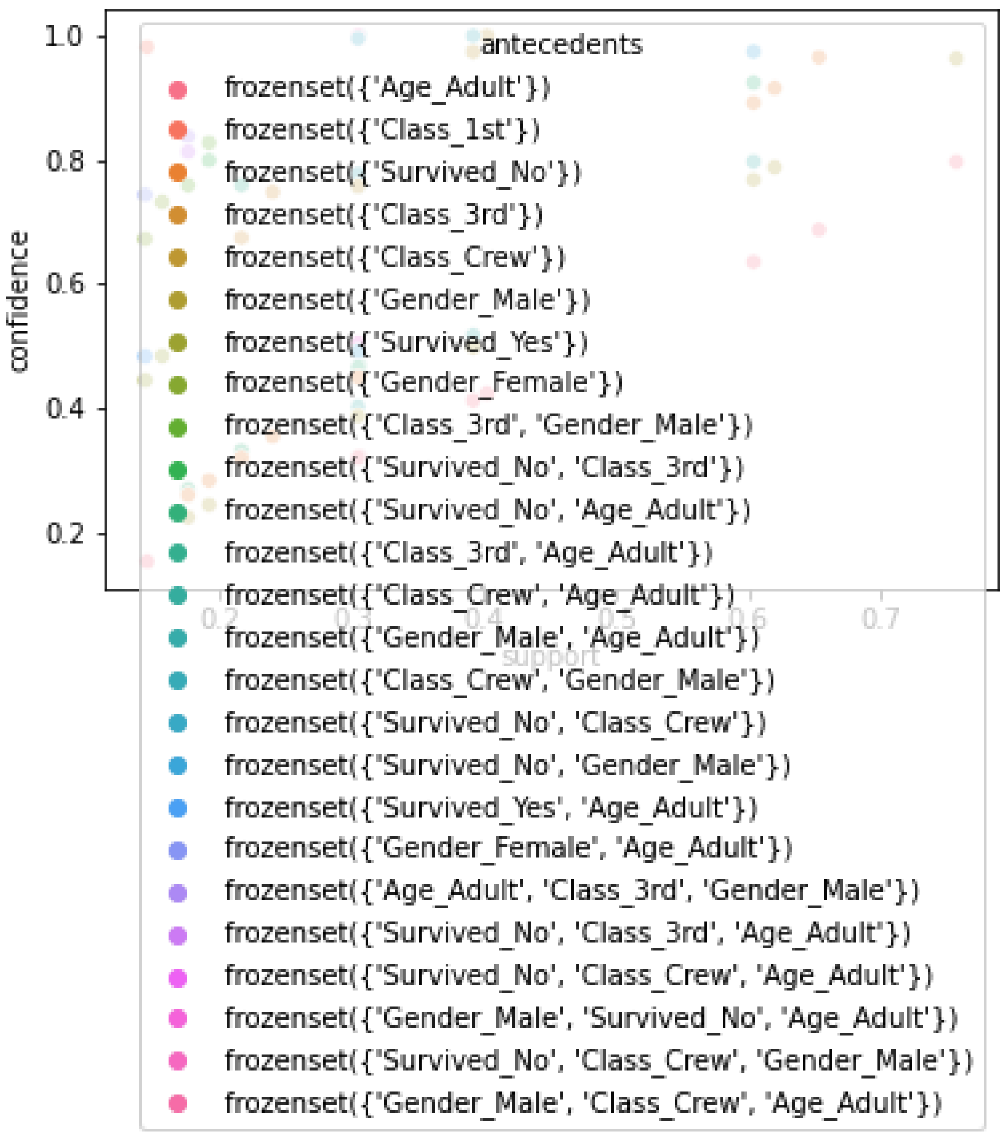


import matplotlib.pyplot as plt x = [5,7,8,7,2,17,2,9,4,11,12,9,6] y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(rules['support'], rules['confidence']) plt.show()



import seaborn as sns sns.scatterplot('support', 'confidence', data=rules, hue='antecedents') plt.show()



1.0
0.8
0.6
0.4
0.2
confidence
0.2
0.3
0.4
0.5
0.6
0.7
support
antecedents
frozenset({'Age_Adult'})
frozenset({'Class_1st'})
frozenset({'Survived_No'})
frozenset({'Class_3rd'})
frozenset({'Class_Crew'})
frozenset({'Gender_Male'})
frozenset({'Survived_Yes'})
frozenset({'Gender_Female'})
frozenset({'Class_3rd', 'Gender_Male'})
frozenset({'Survived_No', 'Class_3rd'})
frozenset({'Survived_No', 'Age_Adult'})
frozenset({'Class_3rd', 'Age_Adult'})
frozenset({'Class_Crew', 'Age_Adult'})
frozenset({'Gender_Male', 'Age_Adult'})
frozenset({'Class_Crew', 'Gender_Male'})
frozenset({'Survived_No', 'Class_Crew'})
frozenset({'Survived_No', 'Gender_Male'})
frozenset({'Survived_Yes', 'Age_Adult'})
frozenset({'Gender_Female', 'Age_Adult'})
frozenset({'Age_Adult', 'Class_3rd', 'Gender_Male'})
frozenset({'Survived_No', 'Class_3rd', 'Age_Adult'})
frozenset({'Survived_No', 'Class_Crew', 'Age_Adult'})
frozenset({'Gender_Male', 'Survived_No', 'Age_Adult'})
frozenset({'Survived_No', 'Class_Crew', 'Gender_Male'})
frozenset({'Gender_Male', 'Class_Crew', 'Age_Adult'})

Practical 8

A. Build an Artificial Neural Network by implementing the Back-propagation algorithm and test the same using appropriate data sets

What is Back Propagation Algorithm?

Backpropagation is an algorithm used to train artificial neural networks. It is a supervised learning algorithm, which means it requires a labelled training dataset in order to learn the weights and biases of the network's connections.

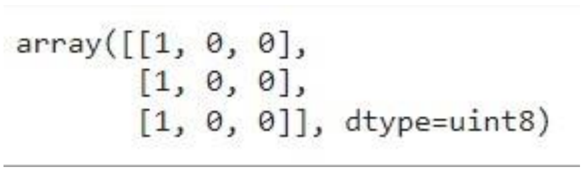
The goal of backpropagation is to adjust the weights and biases of the network in a way that minimizes the error between the network's predictions and the true labels of the training examples. The algorithm does this by propagating the error backwards through the network, using the chain rule of differentiation to calculate the gradient of the error with respect to the weights and biases.

The backpropagation algorithm consists of two phases: forward propagation and backward propagation. In the forward propagation phase, the input data is passed through the network, and the output of the network is calculated. In the backward propagation phase, the error is calculated between the network's output and the true labels, and the error is propagated backwards through the network, adjusting the weights and biases as it goes. This process is repeated for multiple epochs (iterations over the entire training dataset) until the error is minimized to an acceptable level.

Backpropagation is a widely used and effective algorithm for training artificial neural networks, and it is an essential component of many machine learning applications. However, it can be computationally intensive, and it can be sensitive to the choice of hyperparameters (e.g., learning rate, regularization strength).

```
#  Import  Libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split import
matplotlib.pyplot as plt

# Load dataset data
= load_iris()

# Get features and target
X=data.data
```

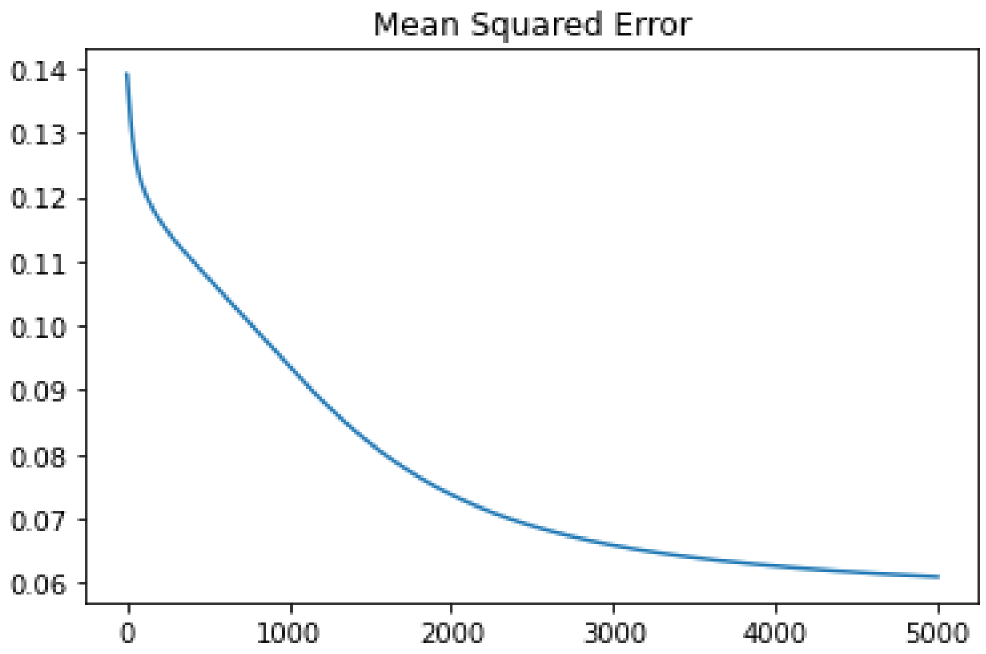
```
y=data.target

# Get dummy variable y =
pd.get_dummies(y).values
y[:3]
```



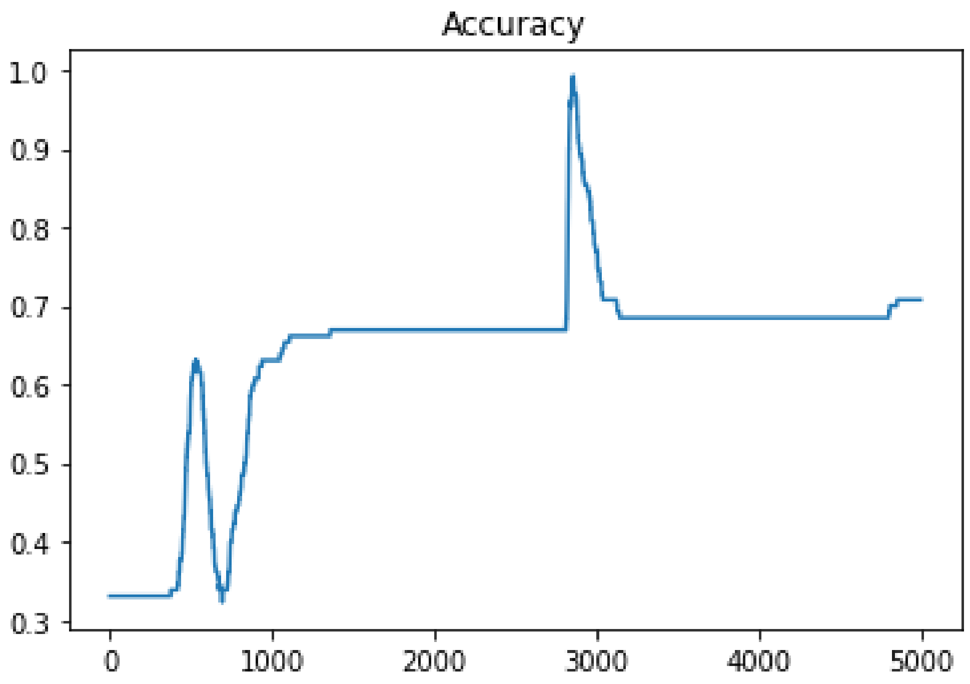
```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]], dtype=uint8)
```

```
#Split data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)

# Initialize variables
learning_rate = 0.1
iterations = 5000 N
= y_train.size

# number of input features input_size
= 4

# number of hidden layers neurons hidden_size
= 2

# number of neurons at the output layer output_size =

3 results = pd.DataFrame(columns=["mse",

"accuracy"])

# Initialize weights
np.random.seed(10)

# initializing weight for the hidden layer
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))

# initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)
```

```
def accuracy(y_pred, y_true): acc =
    y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()

for itr in range(iterations):

    # feedforward propagation


    # on hidden layer
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )

    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update

results.mse.plot(title="Mean Squared Error")
```



```
results.accuracy.plot(title="Accuracy")
```



```
# feedforward
Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)
```

```
acc = accuracy(A2, y_test) print("Accuracy:
{}".format(acc))
```

- Accuracy: 0.8

B. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts based on the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
The formula for Bayes' theorem is given as:
Naïve Bayes Classifier Algorithm
Were,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

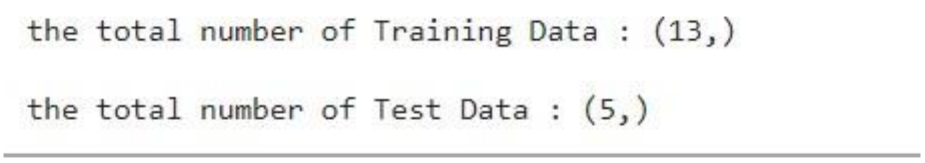
P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

```
import pandas as pd from sklearn.model_selection import
train_test_split from sklearn.feature_extraction.text import
CountVectorizer from sklearn.naive_bayes import
MultinomialNB
from sklearn import metrics

msg=pd.read_csv('/content/drive/MyDrive/Data_Science_Demo/naivetext.csv',names=['mess
age','label']) print('The dimensions of the dataset',msg.shape)
```

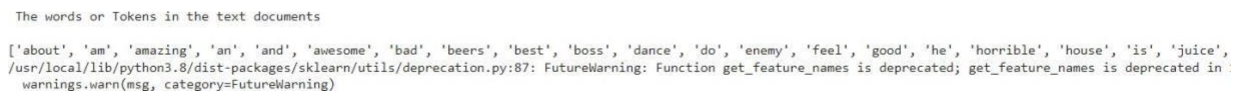
```
The dimensions of the dataset (18, 2)
```

```
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message y=msg.labelnum

#splitting the dataset into train and test data
xtrain,xtest,ytrain,ytest=train_test_split(X,y) print ('\n the
total number of Training Data :',ytrain.shape) print ('\n the
total number of Test Data :',ytest.shape)
```



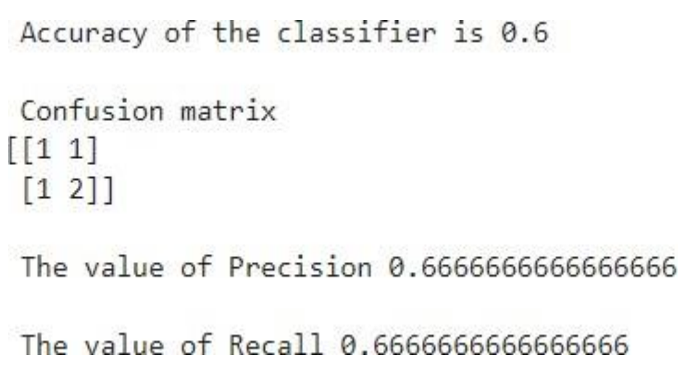
```
the total number of Training Data : (13,)

the total number of Test Data : (5,)
```

```
#output the words or Tokens in the text documents
cv = CountVectorizer() xtrain_dtm =
cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('\n The words or Tokens in the text documents \n') print(cv.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names())
```



```
 The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'bad', 'beers', 'best', 'boss', 'dance', 'do', 'enemy', 'feel', 'good', 'he', 'horrible', 'house', 'is', 'juice',
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in
  warnings.warn(msg, category=FutureWarning)
```

```
# Training Naive Bayes (NB) classifier on training data.
clf = MultinomialNB().fit(xtrain_dtm,ytrain) predicted
= clf.predict(xtest_dtm)

#printing accuracy, Confusion matrix, Precision and Recall
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted)) print('\n
Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted)) print('\n The value of
Precision', metrics.precision_score(ytest,predicted)) print('\n The value
of Recall', metrics.recall_score(ytest,predicted))
```



```
 Accuracy of the classifier is 0.6

 Confusion matrix
[[1 1]
 [1 2]]

 The value of Precision 0.6666666666666666

 The value of Recall 0.6666666666666666
```