Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Junaid A Girkar
60004190057
TE Comps A-4

DWM
EXPERIMENT 1

Aim: Perform data Pre-processing task using Weka data mining tool

1) What is Weka?

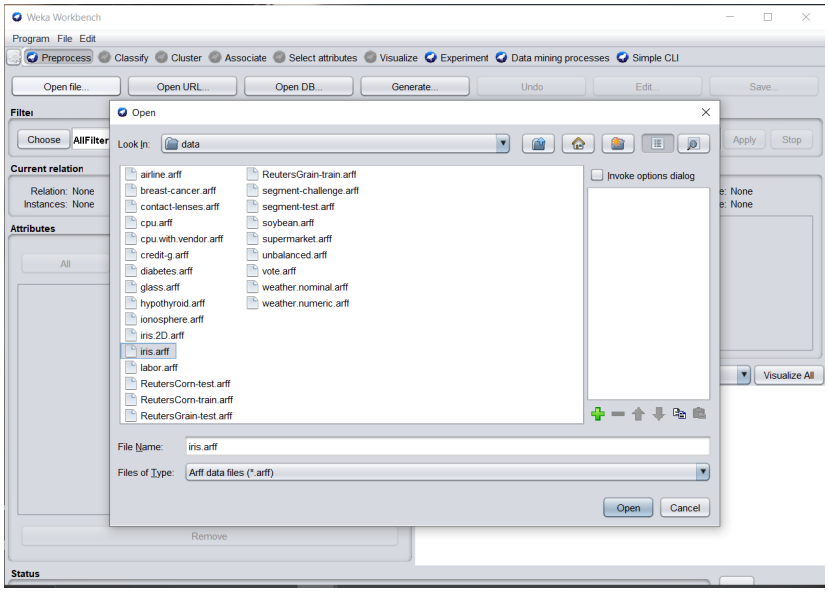
Waikato Environment for Knowledge Analysis (Weka), developed at the University of Waikato, New Zealand, is free software licensed under the GNU General Public License. Weka contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions in particular for educational purposes and research. Advantages of Weka include:

- Free availability under the GNU General Public License.
- Portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform.
- A comprehensive collection of data preprocessing and modeling techniques.
- Ease of use due to its graphical user interfaces.

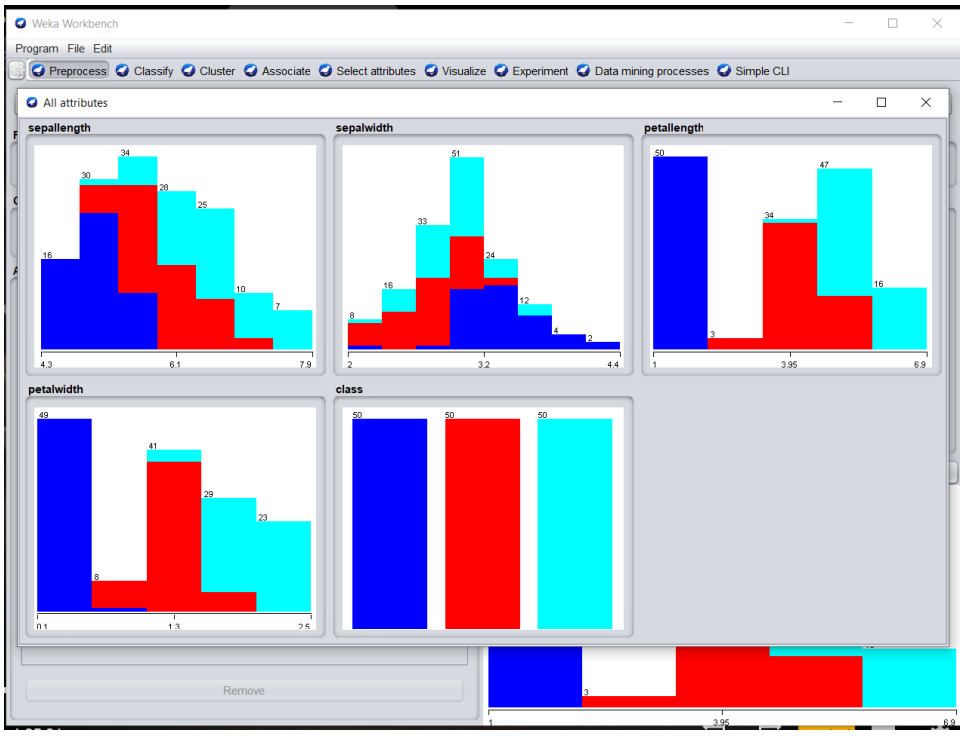
2) Using Weka

The interface is divided into 6 tabs, each with a specific function:

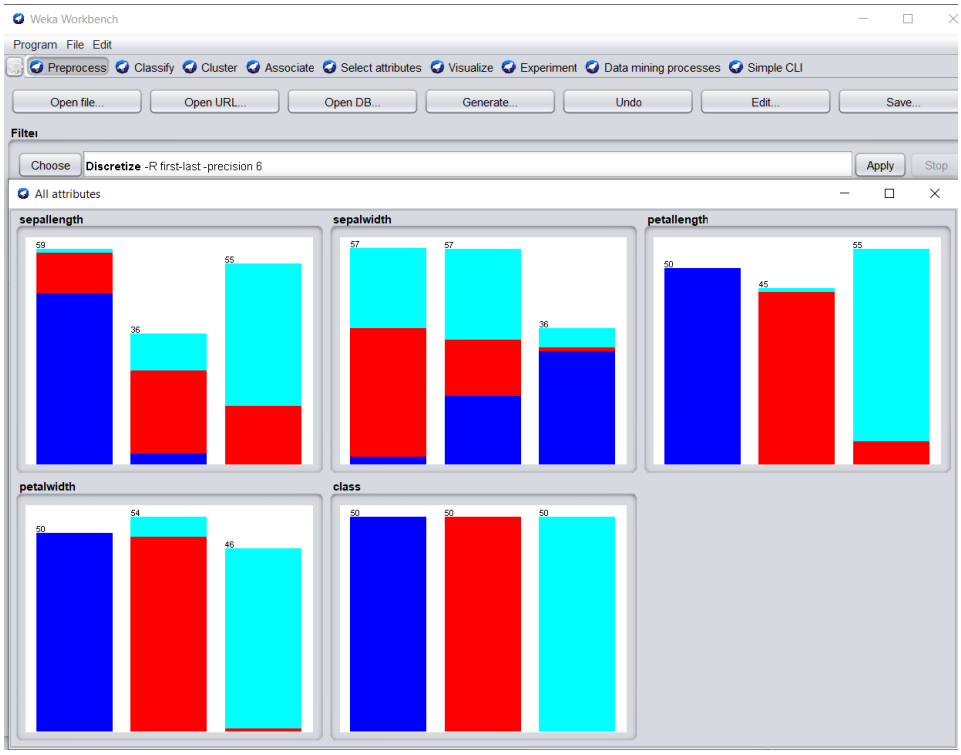
The preprocess tab is for loading your dataset and applying filters to transform the data into a form that better exposes the structure of the problem to the modeling processes. Also provides some summary statistics about loaded data.



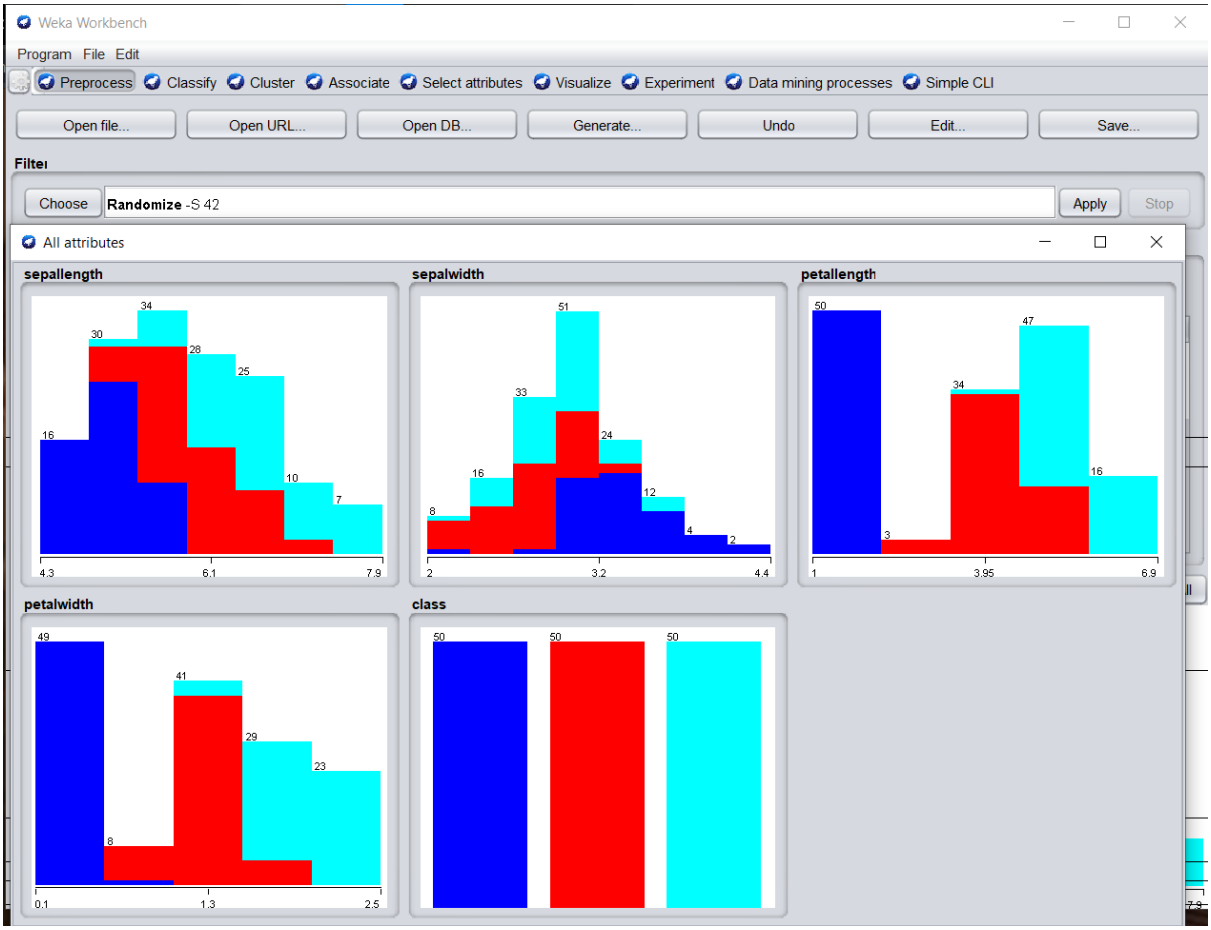
Visualise before filtering



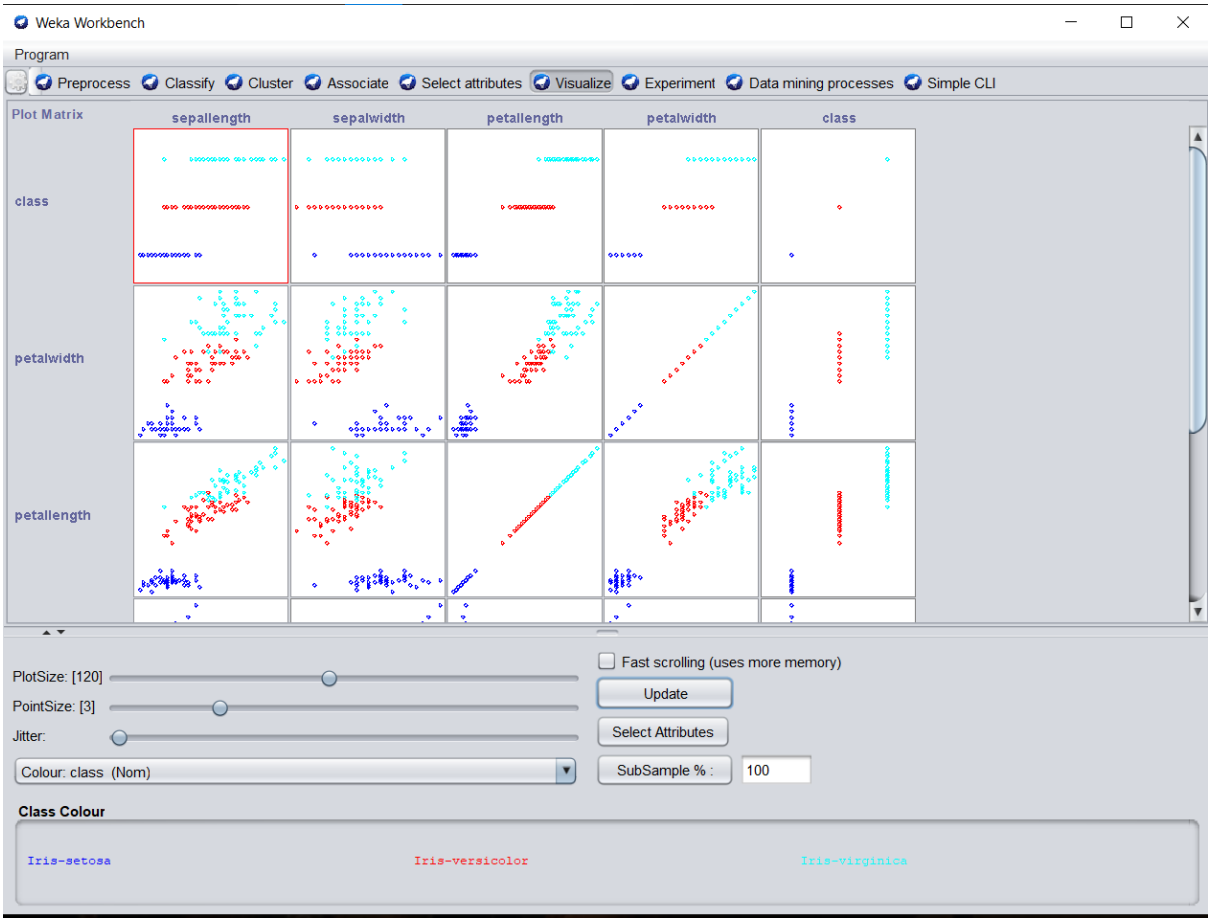
SUPERVISED DISCRETE FILTERING


Weka Workbench
Program File Edit
Preprocess Classify Cluster Associate Select attributes Visualize Experiment Data mining processes Simple CLI
Open file... Open URL... Open DB... Generate... Undo Edit... Save...
Filter
Choose Discretize -R first-last -precision 6 Apply Stop
All attributes
sepallength
59
36
55
sepalwidth
57
57
36
petallength
50
45
55
petalwidth
50
54
46
class
50
50
50

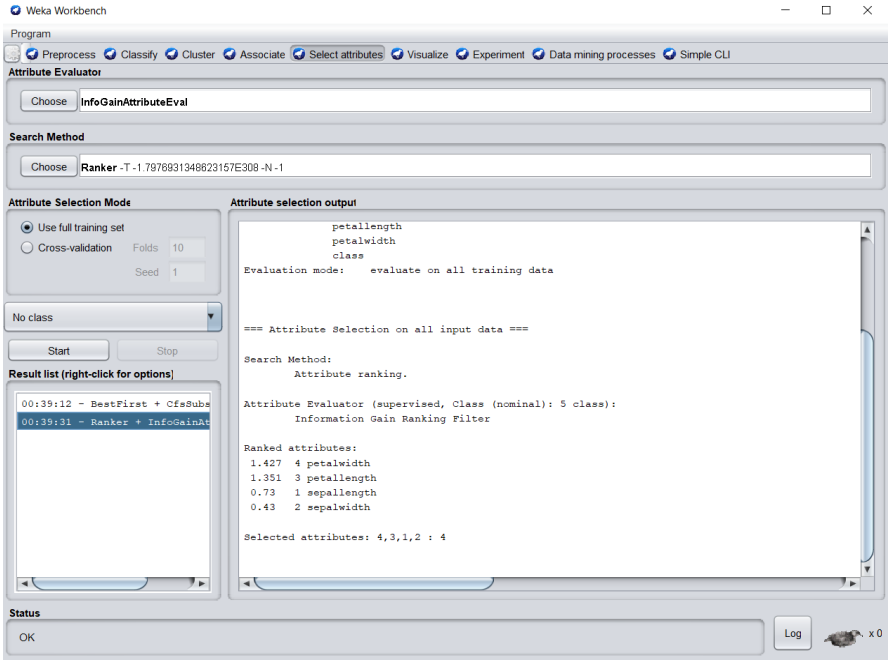
UNSUPERVISED RANDOMIZE FILTERING



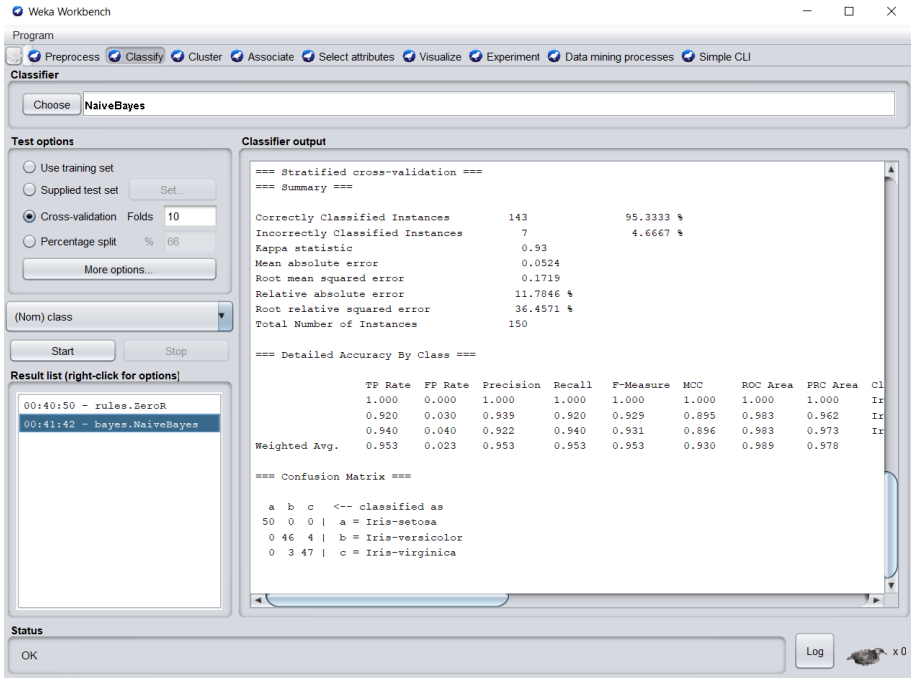
The visualize tab is for reviewing the pairwise scatterplot matrix of each attribute plotted against every other attribute in the loaded dataset. It is useful to get an idea of the shape and relationship of attributes that may aid in data filtering, transformation and modeling.
Increase the point size and the jitter and click the “Update” button to set an improved plot of the categorical attributes of the loaded dataset.



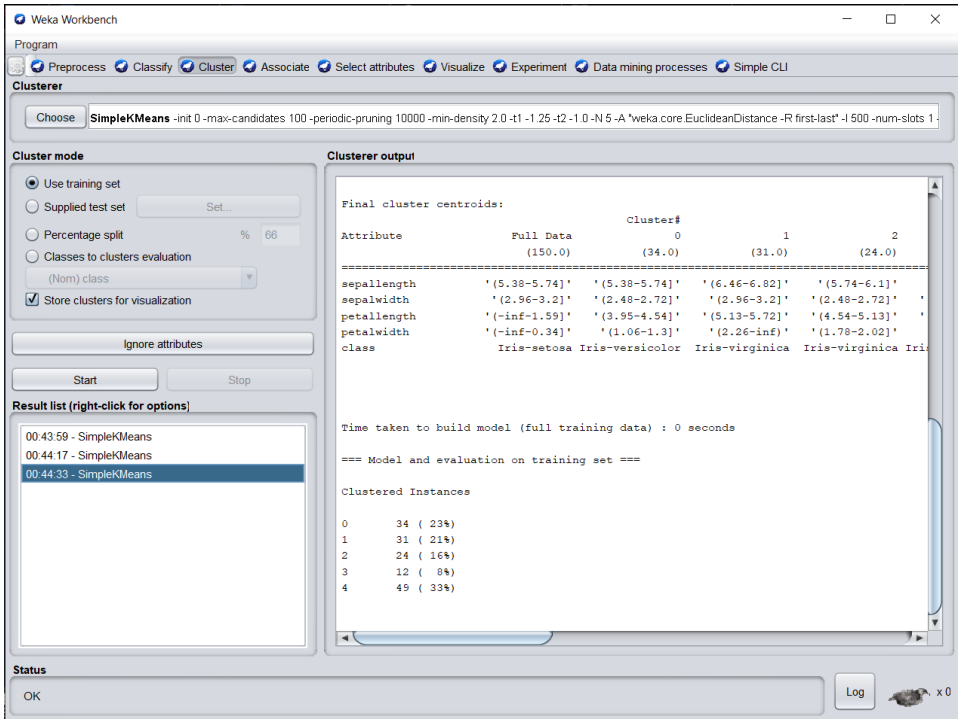
The **select attributes** tab is for performing feature selection on the loaded dataset and identifying those features that are most likely to be relevant in developing a predictive model.



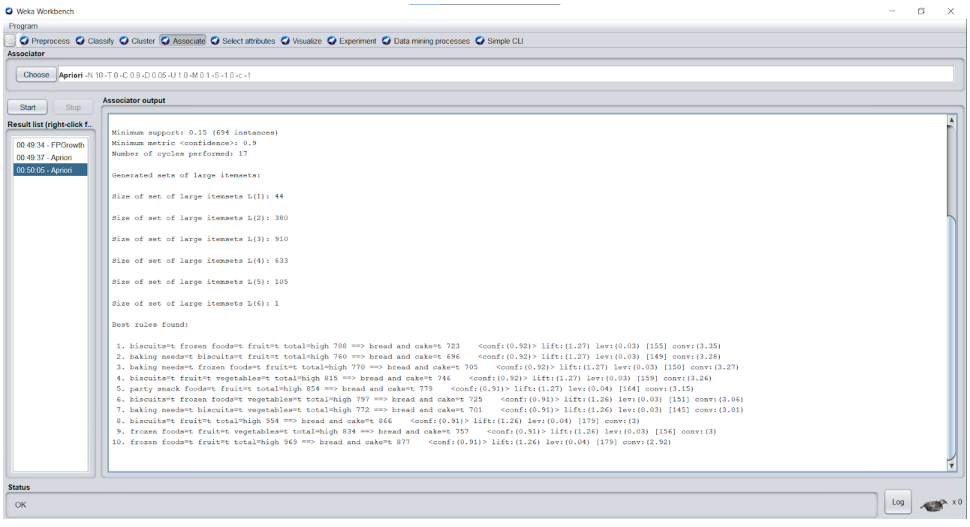
The **classify** tab is for training and evaluating the performance of different machine learning algorithms on your classification or regression problem. Algorithms are divided up into groups, results are kept in a result list and summarized in the main Classifier output.



The **cluster** tab is for training and evaluating the performance of different unsupervised clustering algorithms on your unlabeled dataset. Like the Classify tab, algorithms are divided into groups, results are kept in a result list and summarized in the main Clusterer output.



The **associate** tab is for automatically finding associations in a dataset. The techniques are often used for market basket analysis type data mining problems and require data where all attributes are categorical.



CONCLUSION: We learnt about the Weka tool and how to do data analysis with it. We used 2 different databases : Iris petals and Supermarket.
We tried both the supervised and unsupervised learning algorithms. We can easily visualize with charts how the data transforms when we filter it using different algorithms.
We also used the select attribute to find out which attribute is ranked best for classification. We implemented different clustering and classification algorithms.
In the second database i.e. the supermarket one, we implemented the associate function where we found the different associations in a dataset.

EXPERIMENT 2

THEORY

Naive Bayes:

Naive 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. It 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 on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for text classification problems.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

DECISION TREE:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Some **advantages** of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.

- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. However scikit-learn implementation does not support categorical variables for now. Other techniques are usually specialised in analysing datasets that have only one type of variable.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The **disadvantages** of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations as seen in the above figure. Therefore, they are not good at extrapolation.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.

- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

DATASET 1: Diabetes

CODE:

```
import pandas as pd
df = pd.read_csv('diabetes.csv')

df.head()

df.isnull().sum()

from sklearn.model_selection import train_test_split

X=df.drop(columns=['Outcome'])
y=df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

print("NAIVE BAYERS CLASSIFICATION")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)

nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report

print("Confusion Matrix")
confusion_matrix(y_test,y_pred)

print("Classification Report")
```

```
print(classification_report(y_test,y_pred))

X=df.drop(columns=['Outcome'])
y=df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

from sklearn import tree
dt = tree.DecisionTreeClassifier()

print("\nDECISION TREE CLASSIFICATION")
dt.fit(X_train,y_train)
print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print("Confusion Matrix")
confusion_matrix(y_test,y_pred_dt)

print("Classification Report")
print(classification_report(y_test,y_pred_dt))

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
```

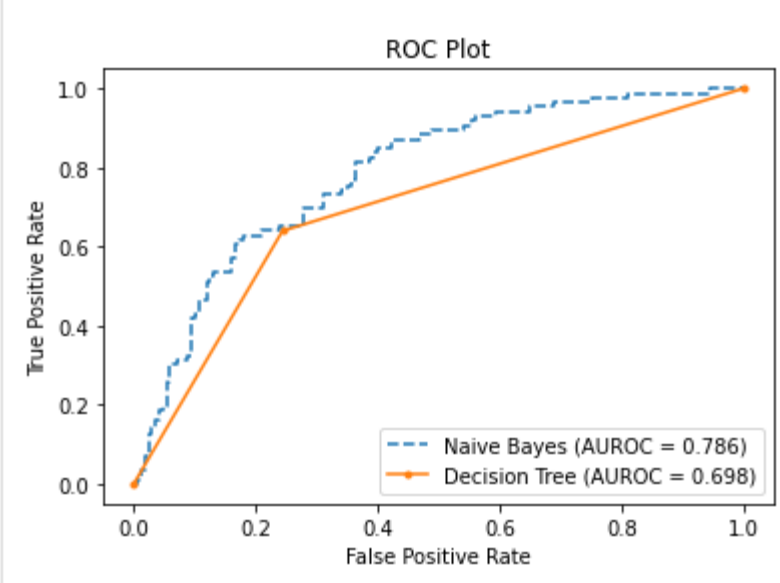
```
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC = %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC = %0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

OUTPUT:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
NAIVE BAYERS CLASSIFICATION
Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.81      0.79      0.80       168
           1       0.61      0.63      0.62        86

    accuracy                           0.74       254
   macro avg       0.71      0.71      0.71       254
weighted avg       0.74      0.74      0.74       254


DECISION TREE CLASSIFICATION
Testing Score
Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.80      0.76      0.78       168
           1       0.57      0.64      0.60        86

    accuracy                           0.72       254
   macro avg       0.69      0.70      0.69       254
weighted avg       0.73      0.72      0.72       254

Decision Tree AUROC = 0.6977436323366556
Naive Bayes AUROC = 0.7857834994462902
```



DATASET 2: Sonar

CODE:

```
import pandas as pd
df = pd.read_csv('sonar.csv',header=None)
print("Showing First 5 rows of the database")
df.head()

print("Checking null fields in the dataset")
df.isnull().sum()

from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()

print("Data before using LavelEncoder")
df[60]

df[60]=le.fit_transform(df[60])
print("Data after using LavelEncoder")
df[60]


from sklearn.model_selection import train_test_split

X=df.drop(columns=[60])
y=df[60]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)
print("Testing Score")
nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report
```

```
print("Confusion Matrix for Naive Bayers")
confusion_matrix(y_test,y_pred)

print("Classification Report")
print(classification_report(y_test,y_pred))

X=df.drop(columns=[60])
y=df[60]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

print("\n\n DECISION TREE CLASSIFIER")
from sklearn import tree
dt = tree.DecisionTreeClassifier()

dt.fit(X_train,y_train)
print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)
print("Classification Report")
print(classification_report(y_test,y_pred_dt))

print("Confusion Matrix for Decision Tree")
confusion_matrix(y_test,y_pred_dt)
nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs
from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))
```

```
nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC = %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC = %0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

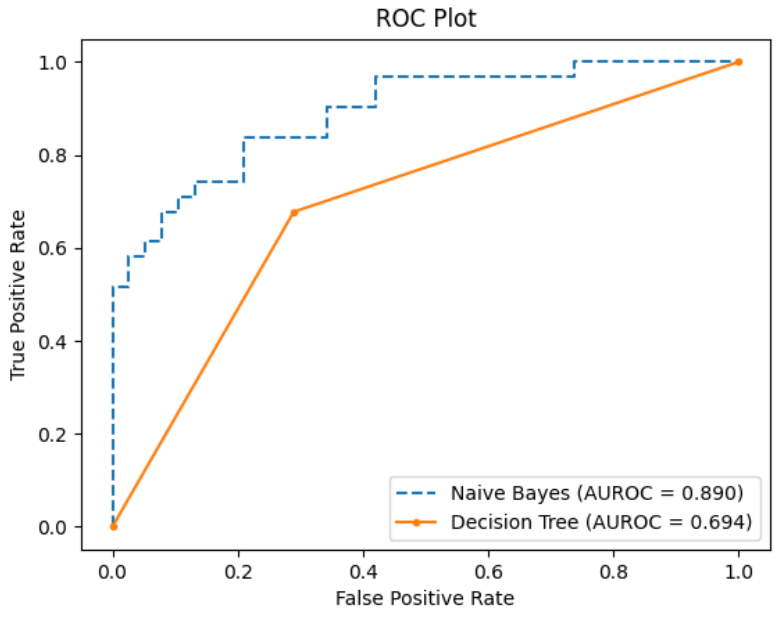
OUTPUT:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	56	57	58	59	60
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	0.0090	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	0.0052	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	0.0095	0.0078	R
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	0.0040	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	0.0107	0.0094	R

5 rows × 61 columns

```
Checking null fields in the dataset

0     0
1     0
2     0
3     0
4     0
     ..
56    0
57    0
58    0
59    0
60    0
Length: 61, dtype: int64
```

```
Showing First 5 rows of the database
Checking null fields in the dataset
Data before using LavelEncoder
Data after using LavelEncoder
Testing Score
Confusion Matrix for Naive Bayers
Classification Report
              precision    recall  f1-score   support

           0       0.86      0.66      0.75        38
           1       0.68      0.87      0.76        31

    accuracy                           0.75        69
   macro avg       0.77      0.76      0.75        69
weighted avg       0.78      0.75      0.75        69



 DECISION TREE CLASSIFIER
Testing Score
Classification Report
              precision    recall  f1-score   support

           0       0.76      0.82      0.78        38
           1       0.75      0.68      0.71        31

    accuracy                           0.75        69
   macro avg       0.75      0.75      0.75        69
weighted avg       0.75      0.75      0.75        69

Confusion Matrix for Decision Tree
Decision Tree AUROC = 0.74660441426146
Naive Bayes AUROC = 0.8904923599320883
```



DATASET 3: Haberman

CODE:

```
import pandas as pd

df = pd.read_csv('haberman.csv',header=None)

df.head()
df.isnull().sum()

from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
df[3]

df[3]=le.fit_transform(df[3])
df[3]

from sklearn.model_selection import train_test_split

X=df.drop(columns=[3])
y=df[3]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

print("NAIVE BAYERS CLASSIFICATION")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)
print("Testing Score")
nb.score(X_test,y_test)
y_pred = nb.predict(X_test)
from sklearn.metrics import confusion_matrix,classification_report

print("Naive Bayers Confusion Matrix")
confusion_matrix(y_test,y_pred)
print("Classification Report")
print(classification_report(y_test,y_pred))
```

```
X=df.drop(columns=[3])
y=df[3]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)
print("DECISION TREE CLASSIFIER")
from sklearn import tree
dt = tree.DecisionTreeClassifier()

dt.fit(X_train,y_train)

print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print(classification_report(y_test,y_pred_dt))
confusion_matrix(y_test,y_pred_dt)

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC = %0.3f)' % nb_auc)
```

```
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC = %0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

OUTPUT:

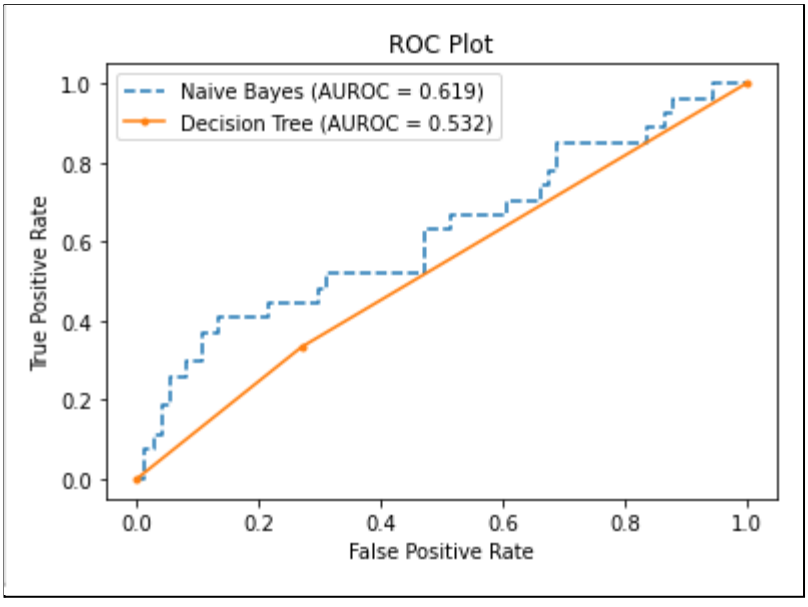
```
df.head()
```

	0	1	2	3
0	30	64	1	1
1	30	62	3	1
2	30	65	0	1
3	31	59	2	1
4	31	65	4	1

```
df.isnull().sum()
```

```
0    0
1    0
2    0
3    0
dtype: int64
```

```
NAIVE BAYERS CLASSIFICATION
Testing Score
Naive Bayers Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.78      0.91      0.84        74
           1       0.53      0.30      0.38        27

    accuracy                           0.74       101
   macro avg       0.66      0.60      0.61       101
weighted avg       0.71      0.74      0.72       101

DECISION TREE CLASSIFIER
Testing Score
              precision    recall  f1-score   support

           0       0.75      0.73      0.74        74
           1       0.31      0.33      0.32        27

    accuracy                           0.62       101
   macro avg       0.53      0.53      0.53       101
weighted avg       0.63      0.62      0.63       101

Decision Tree AUROC = 0.5315315315315315
Naive Bayes AUROC = 0.6191191191191191
```



DATASET 4: Ionosphere

CODE:

```
import pandas as pd

df = pd.read_csv('ionosphere_data.csv')

df.head()

df.isnull().sum()

from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
df['column_ai']

df['column_ai']=le.fit_transform(df['column_ai'])
df['column_ai']

from sklearn.model_selection import train_test_split

X=df.drop(columns=['column_ai'])
y=df['column_ai']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

print("NAIVE BAYERS CLASSIFICATION\n")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)

print("Naive bayers Score:")
nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report

print("Confusion Matrix")
```

```
confusion_matrix(y_test,y_pred)

print("Classification Report")
print(classification_report(y_test,y_pred))

X=df.drop(columns=['column_ai'])
y=df['column_ai']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

from sklearn import tree
dt = tree.DecisionTreeClassifier()

print("\n\nDECISION TREE CLASSIFIER")
dt.fit(X_train,y_train)

dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print("Classification Report")
print(classification_report(y_test,y_pred_dt))

print("Confusion Matrix")
confusion_matrix(y_test,y_pred_dt)

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
```

```
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC = %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC = %0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

OUTPUT:

	column_a	column_b	column_c	column_d	column_e	column_f	column_g	column_h	column_i	column_j	...	column_z	column_aa	column_ab	column_
0	True	False	0.99539	-0.05889	0.85243	0.02306	0.83398	-0.37708	1.00000	0.03760	...	-0.51171	0.41078	-0.46168	0.212
1	True	False	1.00000	-0.18829	0.93035	-0.36156	-0.10868	-0.93597	1.00000	-0.04549	...	-0.26569	-0.20468	-0.18401	-0.190
2	True	False	1.00000	-0.03365	1.00000	0.00485	1.00000	-0.12062	0.88965	0.01198	...	-0.40220	0.58984	-0.22145	0.431
3	True	False	1.00000	-0.45161	1.00000	1.00000	0.71216	-1.00000	0.00000	0.00000	...	0.90695	0.51613	1.00000	1.000
4	True	False	1.00000	-0.02401	0.94140	0.06531	0.92106	-0.23255	0.77152	-0.16399	...	-0.65158	0.13290	-0.53206	0.024

5 rows × 35 columns

```
column_a     0
column_b     0
column_c     0
column_d     0
column_e     0
column_f     0
column_g     0
column_h     0
column_i     0
column_j     0
column_k     0
column_l     0
column_m     0
column_n     0
column_o     0
column_p     0
column_q     0
column_r     0
column_s     0
column_t     0
column_u     0
column_v     0
column_w     0
column_x     0
column_y     0
column_z     0
column_aa    0
column_ab    0
column_ac    0
column_ad    0
column_ae    0
column_af    0
column_ag    0
column_ah    0
column_ai    0
dtype: int64
```

NAIVE BAYERS CLASSIFICATION

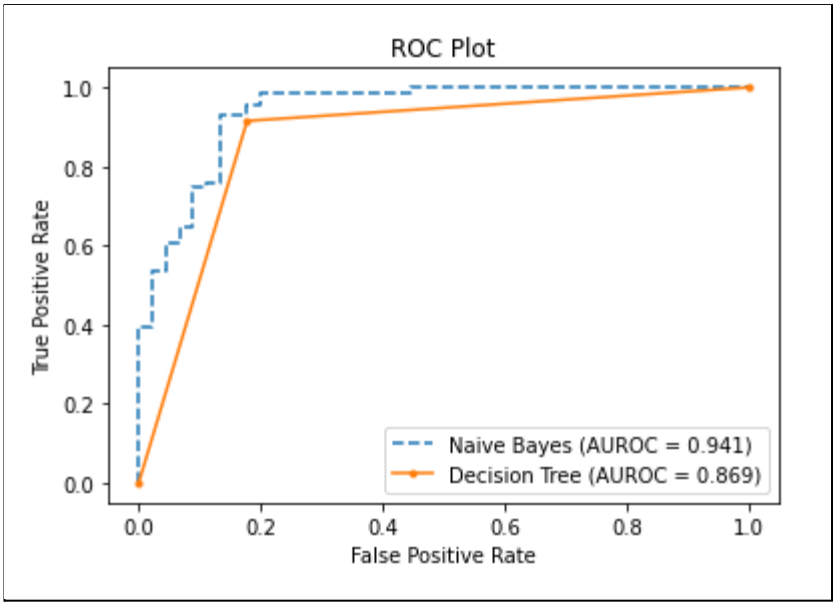
Naive bayers Score:
Confusion Matrix
Classification Report

	precision	recall	f1-score	support
0	0.97	0.78	0.86	45
1	0.88	0.99	0.93	71
accuracy			0.91	116
macro avg	0.92	0.88	0.90	116
weighted avg	0.91	0.91	0.90	116

DECISION TREE CLASSIFIER
Classification Report

	precision	recall	f1-score	support
0	0.86	0.82	0.84	45
1	0.89	0.92	0.90	71
accuracy			0.88	116
macro avg	0.88	0.87	0.87	116
weighted avg	0.88	0.88	0.88	116

Confusion Matrix
Decision Tree AUROC = 0.8688575899843505
Naive Bayes AUROC = 0.9411580594679188



DATASET 5: BankNote Authentication

CODE:

```
import pandas as pd

df = pd.read_csv('BankNoteAuthentication.csv')

df.head()

df.isnull().sum()

from sklearn.model_selection import train_test_split

X=df.drop(columns=['class'])
y=df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

print("NAIVE BAYERS CLASSFICATION")

from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

nb.fit(X_train,y_train)

print("TESTING SCORE")
nb.score(X_test,y_test)

y_pred = nb.predict(X_test)

from sklearn.metrics import confusion_matrix,classification_report

print("CONFUSION MATRIX")
confusion_matrix(y_test,y_pred)

print("CLASSIFICATION REPORT")
print(classification_report(y_test,y_pred))
```

```
X=df.drop(columns=['class'])
y=df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

print("\nDECISION TREE CLASSIFIER")
from sklearn import tree
dt = tree.DecisionTreeClassifier()

dt.fit(X_train,y_train)

print("Testing Score")
dt.score(X_test,y_test)

y_pred_dt = dt.predict(X_test)

print("Confusion Matrix")
confusion_matrix(y_test,y_pred_dt)

print("Classification Report")
print(classification_report(y_test,y_pred_dt))

nb_probs = nb.predict_proba(X_test)
dt_probs = dt.predict_proba(X_test)

dt_probs = dt_probs[:, 1]
nb_probs = nb_probs[:, 1]
nb_probs

from sklearn.metrics import roc_curve, roc_auc_score

nb_auc = roc_auc_score(y_test, nb_probs)
dt_auc = roc_auc_score(y_test, dt_probs)

print('Decision Tree AUROC = ' + str(dt_auc))
print('Naive Bayes AUROC = ' + str(nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(y_test, nb_probs)
```

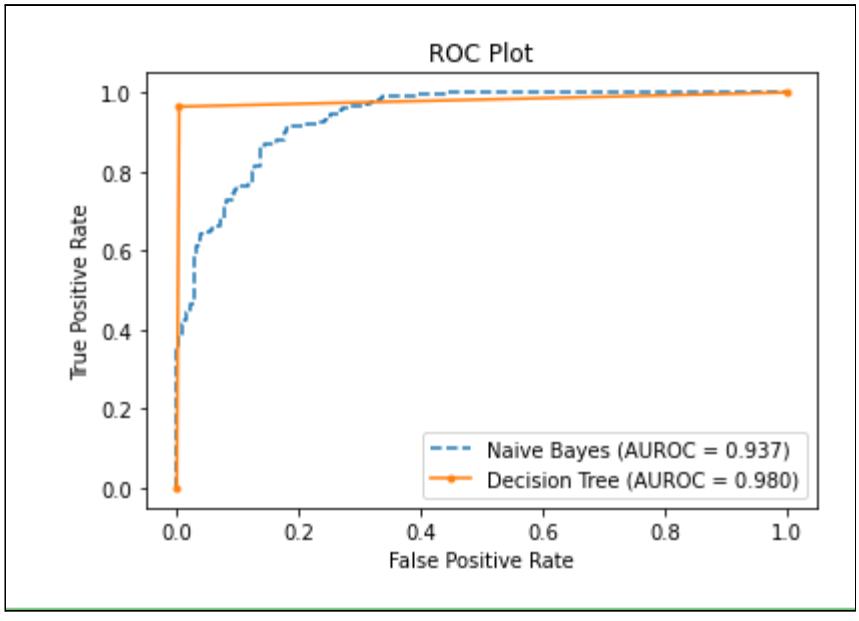
```
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)

import matplotlib.pyplot as plt

plt.plot(nb_fpr, nb_tpr, linestyle='--', label='Naive Bayes (AUROC = %0.3f)' % nb_auc)
plt.plot(dt_fpr, dt_tpr, marker='.', label='Decision Tree (AUROC = %0.3f)' % dt_auc)


# Title
plt.title('ROC Plot')
# Axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show legend
plt.legend() #
# Show plot
plt.show()
```

OUTPUT:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

```
variance    0
skewness    0
curtosis    0
entropy     0
class       0
dtype: int64
```

```
NAIVE BAYERS CLASSFICATION
TESTING SCORE
CONFUSION MATRIX
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.83      0.91      0.86       257
           1       0.86      0.75      0.80       196

    accuracy                           0.84       453
   macro avg       0.84      0.83      0.83       453
weighted avg       0.84      0.84      0.84       453


DECISION TREE CLASSIFIER
Testing Score
Confusion Matrix
Classification Report
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       257
           1       0.99      0.96      0.98       196

    accuracy                           0.98       453
   macro avg       0.98      0.98      0.98       453
weighted avg       0.98      0.98      0.98       453

Decision Tree AUROC = 0.9801973318510284
Naive Bayes AUROC = 0.9371476216945922
```

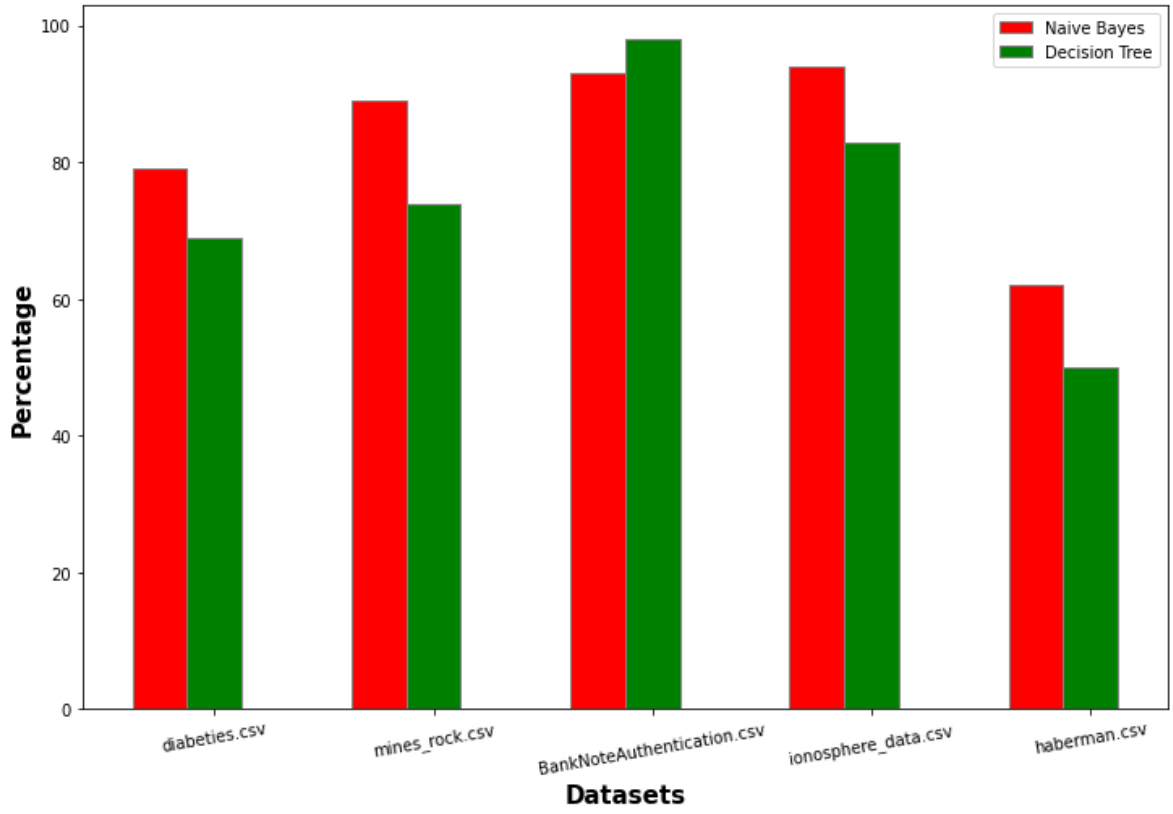


Comparison:

CODE:

```
import numpy as np
import matplotlib.pyplot as plt

barWidth = 0.25
fig = plt.subplots(figsize =(12, 8))

naive_bayes = [79, 89, 93, 94, 62]
decision_tree = [69,74, 98, 83, 50]

br1 = np.arange(len(naive_bayes))
br2 = [x + barWidth for x in br1]

plt.bar(br1, naive_bayes, color ='b', width = barWidth,
        edgecolor ='grey', label ='Naive Bayes')
plt.bar(br2, decision_tree, color ='y', width = barWidth,
        edgecolor ='grey', label ='Decision Tree')
plt.xlabel('Datasets', fontweight ='bold', fontsize = 15)
plt.ylabel('Percentage', fontweight ='bold', fontsize = 15)
plt.xticks([r+ barWidth for r in range(len(naive_bayes))],
        ['diabeties.csv', 'mines_rock.csv', 'BankNoteAuthentication.csv',
'ionosphere_data.csv', 'haberman.csv'],rotation=30)

plt.legend()
plt.show()
```

OUTPUT:


Naive Bayes
Decision Tree
Percentage
100
80
60
40
20
0
diabeties.csv
mines_rock.csv
BankNoteAuthentication.csv
ionosphere_data.csv
haberman.csv
Datasets

PART C

THEORY:

k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    1. Take the group as a hold out or test data set
    2. Take the remaining groups as a training data set
    3. Fit a model on the training set and evaluate it on the test set
    4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Configuration of k

The k value must be chosen carefully for your data sample.

A poorly chosen value for k may result in a mis-representative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the

data used to fit the model), or a high bias, (such as an overestimate of the skill of the model).
Three common tactics for choosing a value for k are as follows:

- Representative: The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.
- k=10: The value for k is fixed to 10, a value that has been found through experimentation to generally result in a model skill estimate with low bias a modest variance.
- k=n: The value for k is fixed to n, where n is the size of the dataset to give each test sample an opportunity to be used in the hold out dataset. This approach is called leave-one-out cross-validation.

Ensemble Learning

Ensemble learning is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models.
Although there are a seemingly unlimited number of ensembles that you can develop for your predictive modeling problem, there are three methods that dominate the field of ensemble learning. So much so, that rather than algorithms per se, each is a field of study that has spawned many more specialized methods.
The three main classes of ensemble learning methods are bagging, stacking, and boosting

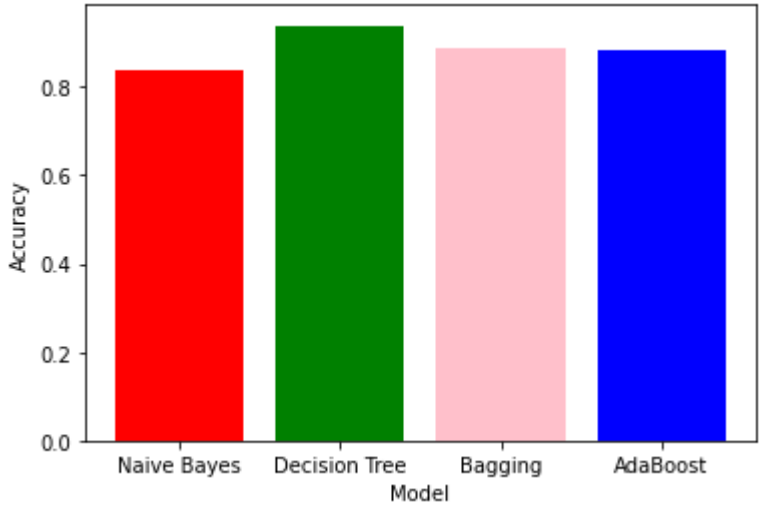
CODE:

```
from sklearn.model_selection import KFold, train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier, VotingClassifier
from numpy import mean
from sklearn.metrics import accuracy_score

cv = KFold(n_splits=10, shuffle=True, random_state=1)
model = AdaBoostClassifier()
def evaluate_model(cv, model):
   X, y = get_dataset()
   scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
   return np.mean(scores), scores.min(), scores.max()
```

```
def naive_bayes_classification(X_train, X_test, y_train, y_test) :
    #Training gaussian model
    gnb = GaussianNB()
    gnb.fit(X_train, y_train)
    #Getting predictions
    y_pred = gnb.predict(X_test)
    return accuracy_score(y_test, y_pred)

def decision_tree_classification(X_train, X_test, y_train, y_test) :
    #Training decision tree
    dtc = tree.DecisionTreeClassifier(
        criterion="entropy",
        max_depth=4,
        max_features=2,
        max_leaf_nodes=None,
        min_samples_leaf=1,
        min_samples_split=2,
        min_weight_fraction_leaf=0.0,
        random_state=None,
        splitter="best",
    )
    dtc.fit(X_train, y_train)
    #Getting predictions
    y_pred = dtc.predict(X_test)
    return accuracy_score(y_test, y_pred)

n_splits=10
#K-Fold Cross Validation
kf = KFold(n_splits=n_splits)
avg_score = [0, 0]
for trainIndex, testIndex in kf.split(df) :
    avg_score[0] += naive_bayes_classification(X_train, X_test, y_train, y_test)
    avg_score[1] += decision_tree_classification(X_train, X_test, y_train, y_test)
print(f"Naive Bayes Avg. Accuracy = {avg_score[0]*100/10} %")
print(f"Decision Tree Avg. Accuracy = {avg_score[1]*100/10} %")
```

```
#Bagging Ensemble model
estimators = [("naiveBayes", GaussianNB()), ("decisionTree",
tree.DecisionTreeClassifier())]
baggingEnsemble = VotingClassifier(estimators)
baggingEnsemble.fit(X_train, y_train)
y_pred = baggingEnsemble.predict(X_test)
baggingAccuracy = accuracy_score(y_test, y_pred)
print(f"Bagging Accuracy: {baggingAccuracy*100} %")


#Adaboost Ensemble model
adaboostEnsemble = AdaBoostClassifier(n_estimators=3)
adaboostEnsemble.fit(X_train, y_train)
y_pred = adaboostEnsemble.predict(X_test)
adaboostAccuracy = accuracy_score(y_test, y_pred)
print(f"Adaboost Accuracy: {adaboostAccuracy*100} %")


#Plotting
plt.bar([1,2,3,4],
[avg_score[0]/10,avg_score[1]/10,baggingAccuracy,adaboostAccuracy],
color=["red","green","pink","blue"])
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.xticks([1,2,3,4],["Naive Bayes", "Decision Tree", "Bagging", "AdaBoost"])
plt.show()
```

OUTPUT:


0.8
0.6
0.4
0.2
0.0
Accuracy
Naive Bayes
Decision Tree
Bagging
AdaBoost
Model

EXPERIMENT 3

LINEAR REGRESSION

AIM: Implementation of Linear Regression
1. Single Variate
2. Multi Variate

THEORY:
Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering and the number of independent variables being used.
Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2.x$$

While training the model we are given :
x: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)
When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.
θ1: intercept
θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Cost Function (J):

By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the θ1 and θ2 values, to reach the best value that minimize the error between predicted y value (pred) and true y value (y).

$$minimize \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

Cost function(J) of Linear Regression is the Root Mean Squared Error (RMSE) between predicted y value (pred) and true y value (y).

CODE:

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

from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plb
from sklearn.model_selection import train_test_split

sns.set()
warnings.simplefilter("ignore")

df = pd.read_csv("StudentsPerformance.csv")
df.head()

print(df.info())
```

```
df['final score'] = df.apply(lambda x : (x['math score'] + x['reading score'] + x['writing score'])
/ 3, axis=1)

df.head()
data2 = df.drop('final score', axis=1)
plt.figure(figsize=(16, 6))
sns.boxplot(data=data2)

df = df.apply(LabelEncoder().fit_transform)

# MULTIVARIATE

X = df.drop('final score', axis=1)
y = df['final score']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
lr = LinearRegression()
lr.fit(X_train, y_train)

pred = lr.predict(X_test)

lr.score(X_test, y_test)
accuracy = mean_squared_error(y_test, pred)
print('Mean Squared Error: ', accuracy)

# UNIVARIATE
sns.scatterplot(df["writing score"],df["final score"])
plt.savefig('scp-1', dpi=500)
m, b = np.polyfit(df["writing score"], df["final score"], 1)

plt.plot(df["writing score"], m*df["writing score"] + b)

X_uni = df['writing score']
y_uni = df['final score']
X_uni_train, X_uni_test, y_uni_train, y_uni_test = train_test_split(X_uni,y_uni,test_size = 0.2)

lr2 = LinearRegression()
X_uni_train = X_uni_train.reshape(-1,1)
X_uni_test = X_uni_test.values.reshape(-1,1)

lr2.fit(X_uni_train, y_uni_train)
pred_uni = lr2.predict(X_uni_test)
lr2.score(X_uni_test, y_uni_test)
```

```
accuracy_uni = mean_squared_error(y_uni_test, pred_uni)
print('Mean Squared Error: ', accuracy_uni)
```

OUTPUT:

head() of the database:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

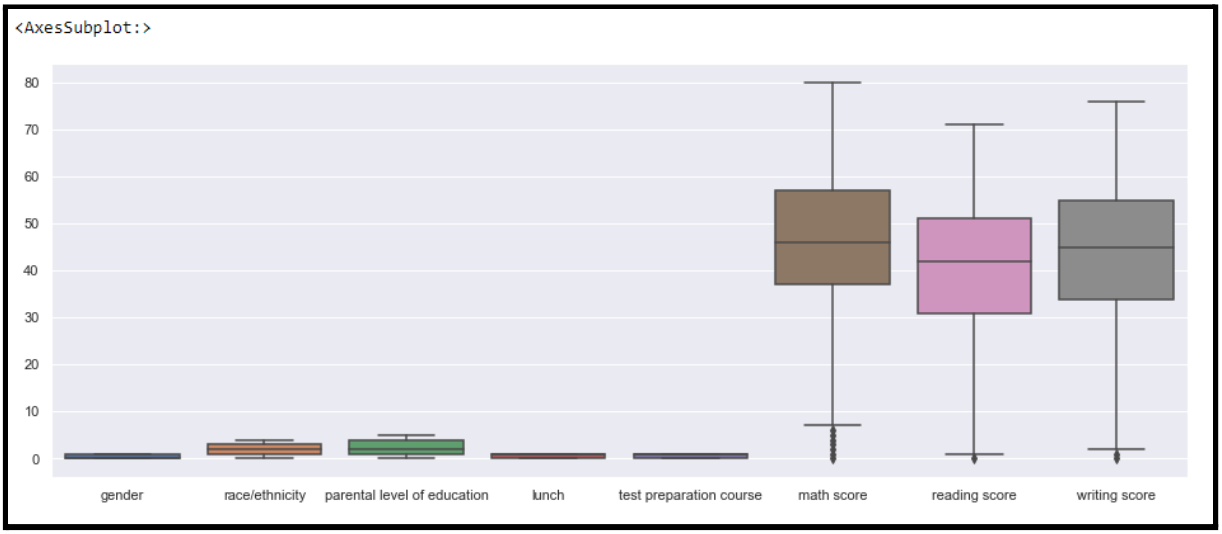
After running df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
None
```

df.head() after adding a final score column

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	final score
0	female	group B	bachelor's degree	standard	none	72	72	74	72.666667
1	female	group C	some college	standard	completed	69	90	88	82.333333
2	female	group B	master's degree	standard	none	90	95	93	92.666667
3	male	group A	associate's degree	free/reduced	none	47	57	44	49.333333
4	male	group C	some college	standard	none	76	78	75	76.333333

Boxplot of the features



df.head() after applying LabelEncoder to the dataset

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	final score
0	0	1	1	1	1	52	44	50	118
1	0	2	4	1	0	49	62	64	147
2	0	1	3	1	1	70	67	69	178
3	1	0	0	0	1	27	29	20	48
4	1	2	4	1	1	56	50	51	129

df.info() after applying LabelEncoder to the dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   int64
 1   race/ethnicity               1000 non-null   int64
 2   parental level of education  1000 non-null   int64
 3   lunch                        1000 non-null   int64
 4   test preparation course      1000 non-null   int64
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
 8   final score                  1000 non-null   int64
dtypes: int64(9)
memory usage: 70.4 KB
None
```

Considering Multivariate Linear Regression

Prediction Score of MultiVariate Linear Regression

```
lr.score(X_test, y_test)
```

```
0.9992194766540022
```

Mean Square Error of MultiVariate Linear Regression

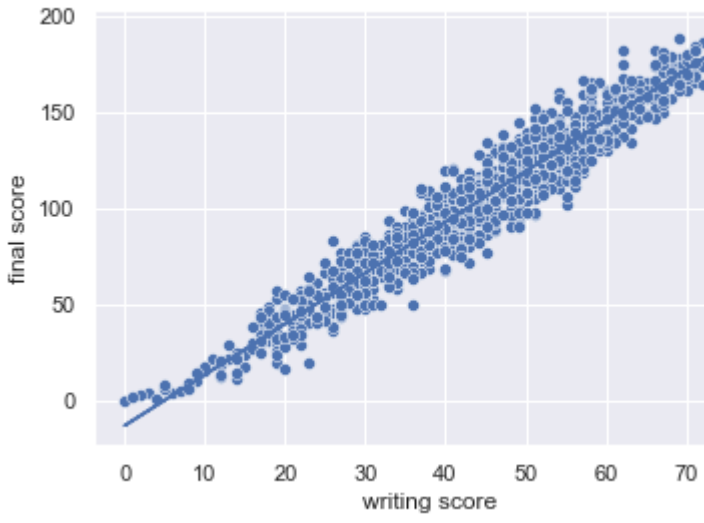
```
print('Mean Squared Error: ', accuracy)
```

```
Mean Squared Error:  1.692171227952071
```

Now considering Univariate Linear Regression with Writing Score as the feature

Scatter Plot of the dataset

```
[<matplotlib.lines.Line2D at 0x2119c6f7e20>]
```



Prediction Score of Univariate LR

```
lr2.score(X_uni_test, y_uni_test)
```

```
0.9421228773316737
```

Mean Square Error of Univariate LR

```
accuracy_uni = mean_squared_error(y_uni_test, pred_uni)
print('Mean Squared Error: ', accuracy_uni)
```

```
Mean Squared Error:  109.48409107917793
```

CONCLUSION: We have implemented Multivariate and Univariate Linear Regression on a dataset and have observed the differences in their Accuracy Score and Mean Squared Errors. We observe 99.92% accuracy in the case of Multivariate with a Mean Squared Error of 1.62 whereas in the case of Univariate, the accuracy score is 94.21% and the Mean Squared Error is 109.48. Therefore we can conclude that using Multivariate Linear Regression is better than using Univariate but nevertheless the efficiency of Univariate is still great.

EXPERIMENT 4

Aim

Implementation of K Means and Hierarchical Clustering algorithm

Theory

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.
Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering Methods :

- **Density-Based Methods**: These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.
- **Hierarchical Based Methods:** The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category
    - Agglomerative (bottom-up approach)
    - Divisive (top-down approach)

- **Partitioning Methods:** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.
- **Grid-based Method**s: In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects.

K Means

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

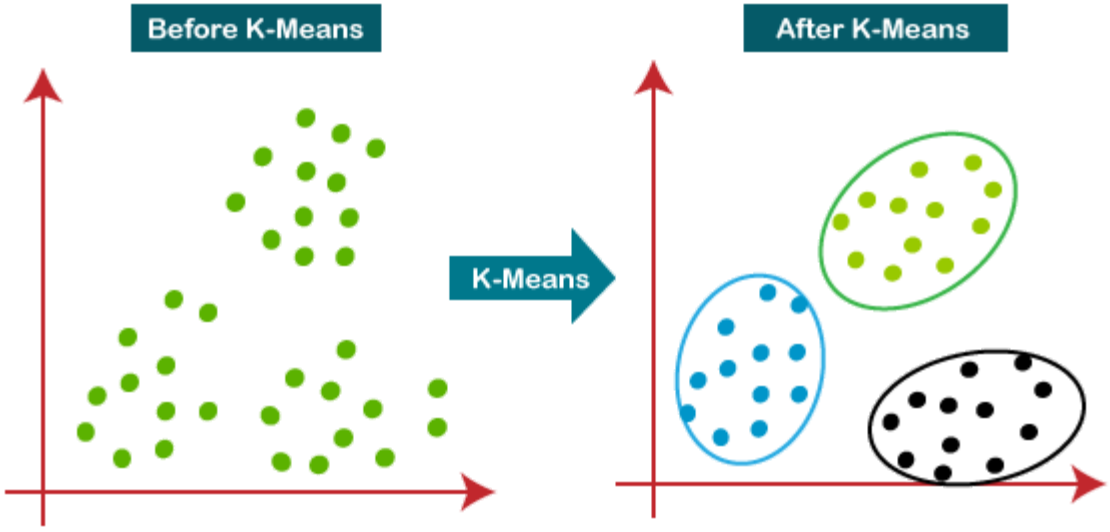
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

Determines the best value for K center points or centroids by an iterative process.
Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.
Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

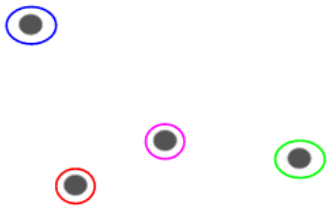
The below diagram explains the working of the K-means Clustering Algorithm:



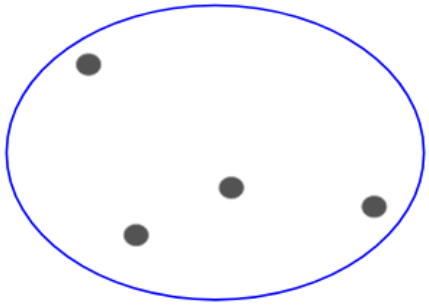
Hierarchical

Let's say we have the below points and we want to cluster them into groups:

We can assign each of these points to a separate cluster:



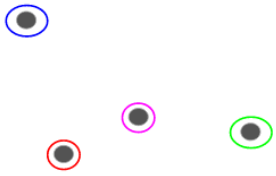
Now, based on the similarity of these clusters, we can combine the most similar clusters together and repeat this process until only a single cluster is left:



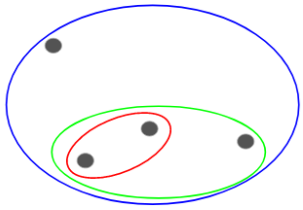
We are essentially building a hierarchy of clusters. That's why this algorithm is called hierarchical clustering. I will discuss how to decide the number of clusters in a later section.

Agglomerative Hierarchical Clustering

We assign each point to an individual cluster in this technique. Suppose there are 4 data points. We will assign each of these points to a cluster and hence will have 4 clusters in the beginning:



Then, at each iteration, we merge the closest pair of clusters and repeat this step until only a single cluster is left:

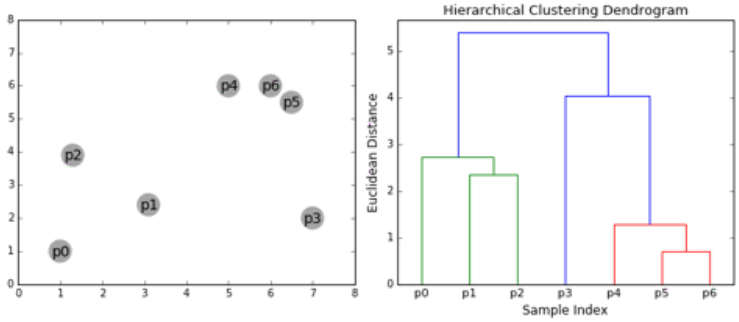


We are merging (or adding) the clusters at each step, right? Hence, this type of clustering is also known as additive hierarchical clustering.

DENDROGRAM

A Dendrogram is a type of tree diagram showing hierarchical relationships between different sets of data.

As already said, a Dendrogram contains the memory of a hierarchical clustering algorithm, so just by looking at the Dendrogram you can tell how the cluster is formed.


p0
p1
p2
p3
p4
p5
p6
Hierarchical Clustering Dendrogram
Euclidean Distance
Sample Index

PART A (Using Inbuilt function)

K Means:

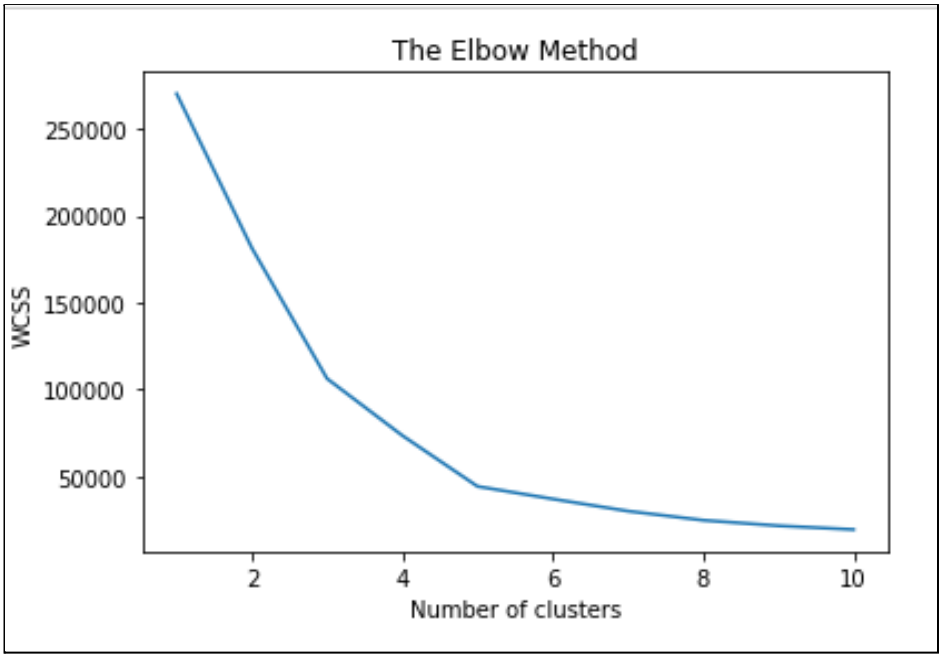
CODE:

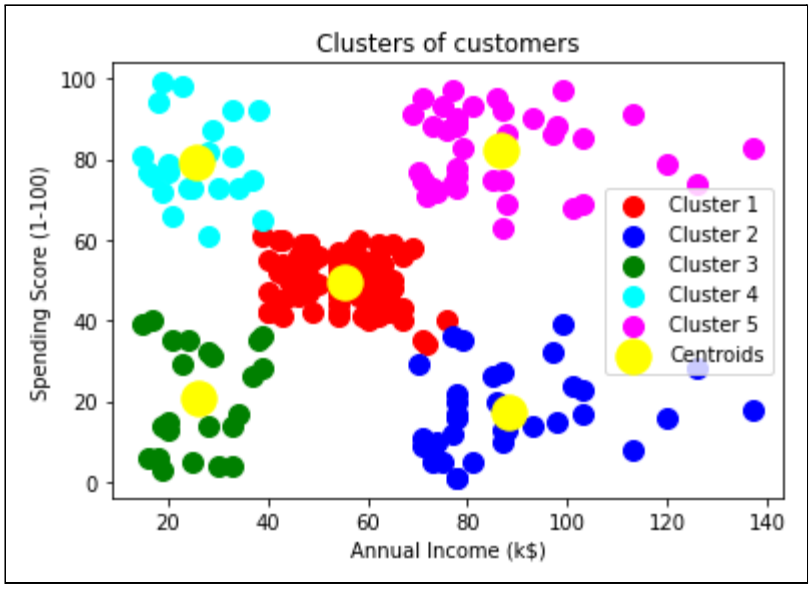
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('Mall_Customers.csv')
dataset.head()

X = dataset.iloc[:, [3, 4]].values

X

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Training the K-Means model on the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

print(y_kmeans)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label =
'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label =
```

```
'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label =
'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label =
'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label =
'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c =
'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40





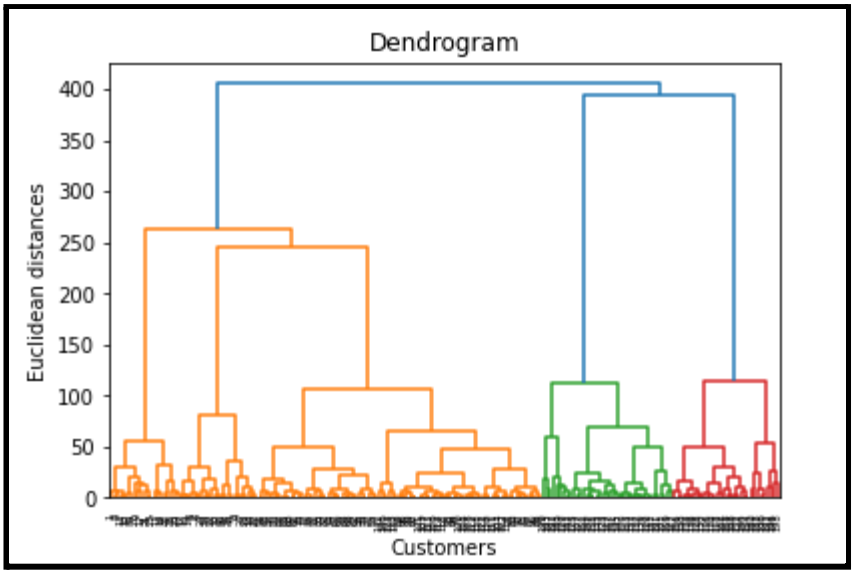
Hierarchical Clustering

CODE:

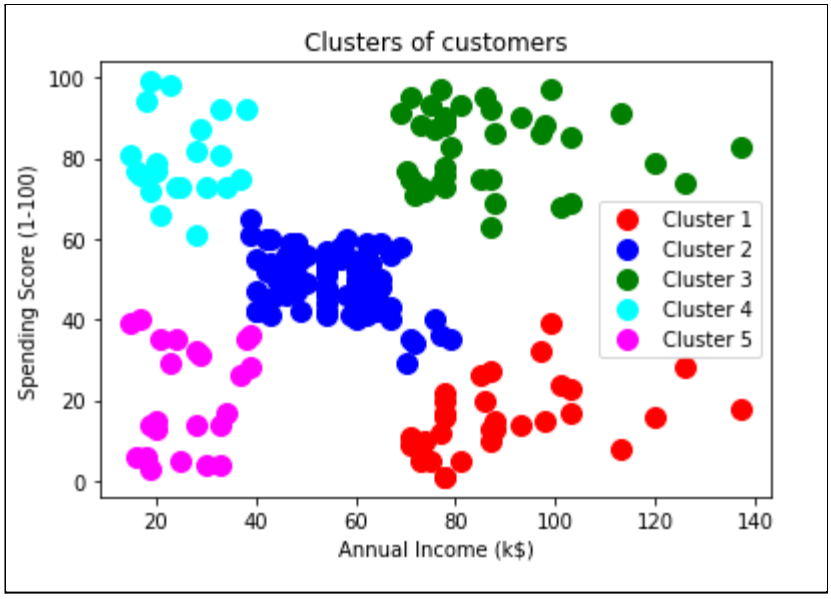
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

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

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

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

print(y_hc)

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

OUTPUT:




```
[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1 2 0 2 1 2
 0 2 0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```



PART B

K Means
CODE:

```
import pandas as pd

data = pd.read_csv("driver-data.csv", index_col="id")
data.head()

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4)

kmeans.fit(data)

kmeans.cluster_centers_

kmeans.labels_

import numpy as np

unique, counts = np.unique(kmeans.labels_, return_counts=True)

dict_data = dict(zip(unique, counts))
dict_data

import seaborn as sns

data["cluster"] = kmeans.labels_

sns.pairplot(data)

kmeans.inertia_

kmeans.score

data
```

```
from sklearn import metrics

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
import pandas as pd

style.use('ggplot')

class K_Means:
	def __init__(self, k =3, tolerance = 0.0001, max_iterations = 500):
		self.k = k
		self.tolerance = tolerance
		self.max_iterations = max_iterations

	def fit(self, data):

		self.centroids = {}

		#initialize the centroids, the first 'k' elements in the dataset will be our initial centroids
		for i in range(self.k):
			self.centroids[i] = data[i]

		#begin iterations
		for i in range(self.max_iterations):
			self.classes = {}
			for i in range(self.k):
				self.classes[i] = []

			#find the distance between the point and cluster; choose the nearest centroid
			for features in data:
				distances = [np.linalg.norm(features - self.centroids[centroid]) for centroid in self.centroids]
				classification = distances.index(min(distances))
				self.classes[classification].append(features)
```

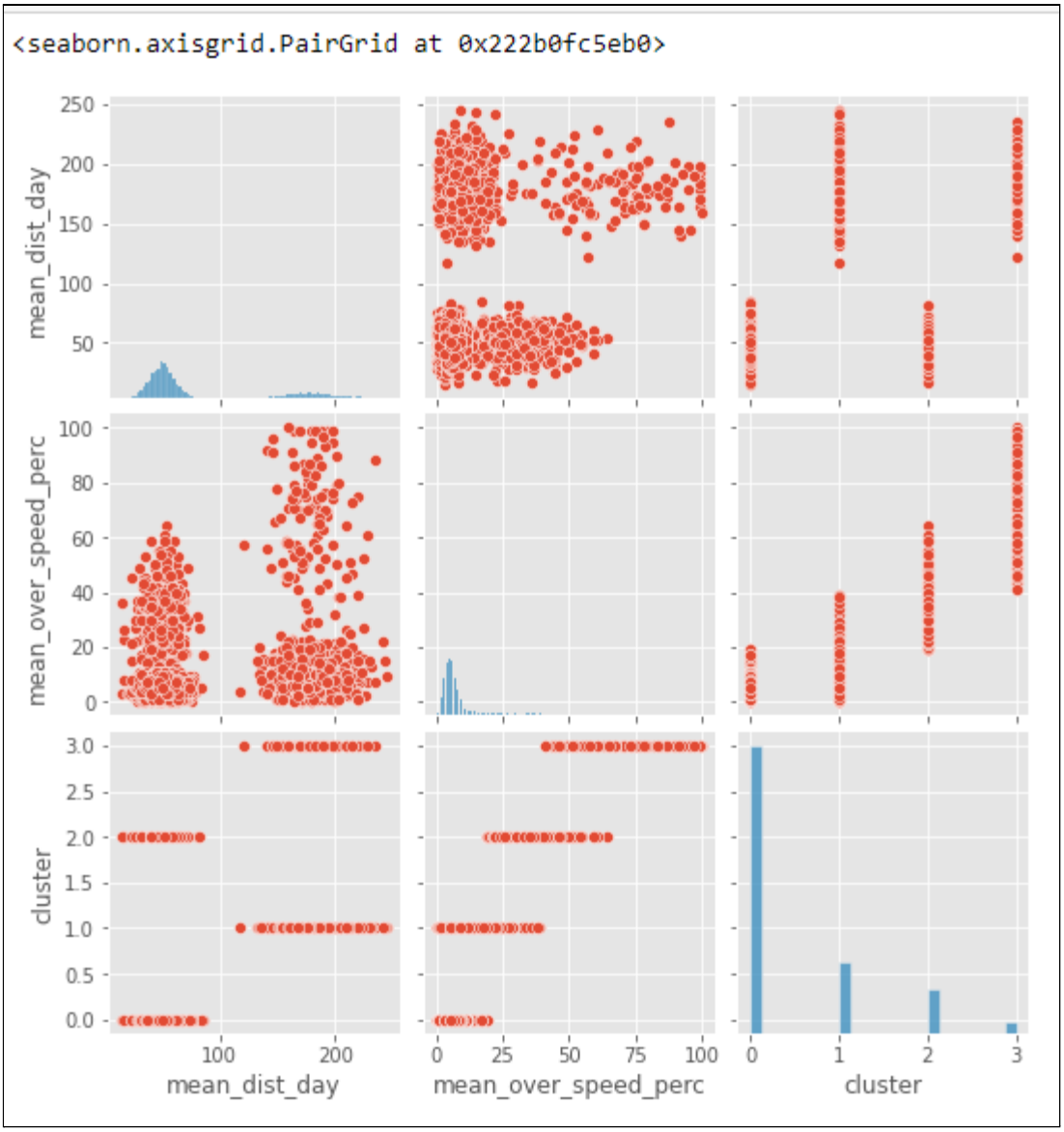
```
                previous = dict(self.centroids)

                #average the cluster datapoints to re-calculate the centroids
                for classification in self.classes:
                        self.centroids[classification] =
np.average(self.classes[classification], axis = 0)

                isOptimal = True

                for centroid in self.centroids:

                        original_centroid = previous[centroid]
                        curr = self.centroids[centroid]

                        if np.sum((curr - original_centroid)/original_centroid *
100.0) > self.tolerance:
                                isOptimal = False

                #break out of the main loop if the results are optimal, ie. the
centroids don't change their positions much(more than our tolerance)
                if isOptimal:
                        break

        def pred(self, data):
                distances = [np.linalg.norm(data - self.centroids[centroid]) for
centroid in self.centroids]
                classification = distances.index(min(distances))
                return classification

def main():

        df = pd.read_csv("Mall_Customers.csv")
        df = X = df.iloc[:, [3, 4]]
        dataset = df.astype(float).values.tolist()

        X = df.values #returns a numpy array

        km = K_Means(5)
```

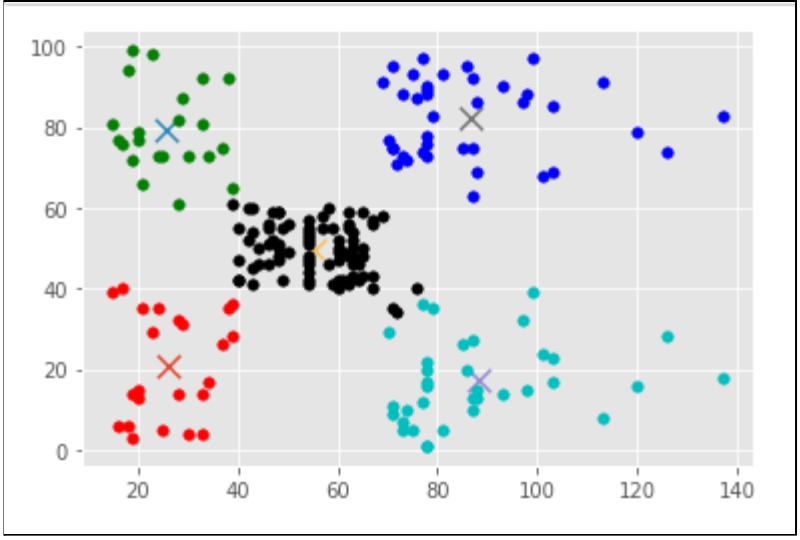
```
        km.fit(X)

        # Plotting starts here
        colors = 10*["r", "g", "c", "b", "k"]

        for centroid in km.centroids:
                plt.scatter(km.centroids[centroid][0], km.centroids[centroid][1], s =
130, marker = "x")

        for classification in km.classes:
                color = colors[classification]
                for features in km.classes[classification]:
                        plt.scatter(features[0], features[1], color = color,s = 30)

        plt.show()


if __name__ == "__main__":
        main()
```

OUTPUT:

id	mean_dist_day	mean_over_speed_perc
3423311935	71.24	28
3423313212	52.53	25
3423313724	64.54	27
3423311373	55.69	22
3423310999	54.58	25

```
<seaborn.axisgrid.PairGrid at 0x222b0fc5eb0>
```





Hierarchical Clustering

CODE:

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

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

new_data = dataset
new_data = new_data.drop('CustomerID', axis=1)
new_data

sns.pairplot(dataset)

from sklearn.preprocessing import LabelEncoder
new_data = new_data.apply(LabelEncoder().fit_transform)

X = new_data.to_numpy()

class Distance_computation_grid(object):
    '''
        class to enable the Computation of distance matrix
    '''
    def __init__(self):
        pass

    def compute_distance(self,samples):
        '''
            Creates a matrix of distances between individual samples and clusters
attained at a particular step
```

```
        '''
        Distance_mat = np.zeros((len(samples),len(samples)))
        for i in range(Distance_mat.shape[0]):
            for j in range(Distance_mat.shape[0]):
                if i!=j:
                    Distance_mat[i,j] = float(self.distance_calculate(samples[i],samples[j]))
                else:
                    Distance_mat[i,j] = 10**4
        return Distance_mat


    def distance_calculate(self,sample1,sample2):
        '''
            Distance calulated between two samples. The two samples can be both
samples, both clusters or
            one cluster and one sample. If both of them are samples/clusters, then
simple norm is used. In other
            cases, we refer it as an exception case and pass the samples as parameter
to some function that
            calculates the necessary distance between cluster and a sample
        '''
        dist = []
        for i in range(len(sample1)):
            for j in range(len(sample2)):
                try:
                    dist.append(np.linalg.norm(np.array(sample1[i])-np.array(sample2[j])))
                except:
                    dist.append(self.intersampledist(sample1[i],sample2[j]))
        return min(dist)


    def intersampledist(self,s1,s2):
        '''
            To be used in case we have one sample and one cluster . It takes the help of
one
            method 'interclusterdist' to compute the distances between elements of a
cluster(which are
            samples) and the actual sample given.
        '''
```

```
        if str(type(s2[0]))!='<class \'list\'>':
            s2=[s2]
        if str(type(s1[0]))!='<class \'list\'>':
            s1=[s1]
        m = len(s1)
        n = len(s2)
        dist = []
        if n>=m:
            for i in range(n):
                for j in range(m):
                    if (len(s2[i])>=len(s1[j])) and str(type(s2[i][0])!='<class \'list\'>'):
                        dist.append(self.interclusterdist(s2[i],s1[j]))
                    else:
                        dist.append(np.linalg.norm(np.array(s2[i])-np.array(s1[j])))
        else:
            for i in range(m):
                for j in range(n):
                    if (len(s1[i])>=len(s2[j])) and str(type(s1[i][0])!='<class \'list\'>'):
                        dist.append(self.interclusterdist(s1[i],s2[j]))
                    else:
                        dist.append(np.linalg.norm(np.array(s1[i])-np.array(s2[j])))
        return min(dist)

    def interclusterdist(self,cl,sample):
        if sample[0]!='<class \'list\'>':
            sample = [sample]
        dist   = []
        for i in range(len(cl)):
            for j in range(len(sample)):
                dist.append(np.linalg.norm(np.array(cl[i])-np.array(sample[j])))
        return min(dist)

progression = [[i] for i in range(X.shape[0])]
samples     = [[list(X[i])] for i in range(X.shape[0])][:10]
m = len(samples)
distcal  = Distance_computation_grid()

while m>2:
    print('Sample size before clustering    :- ',m)
```

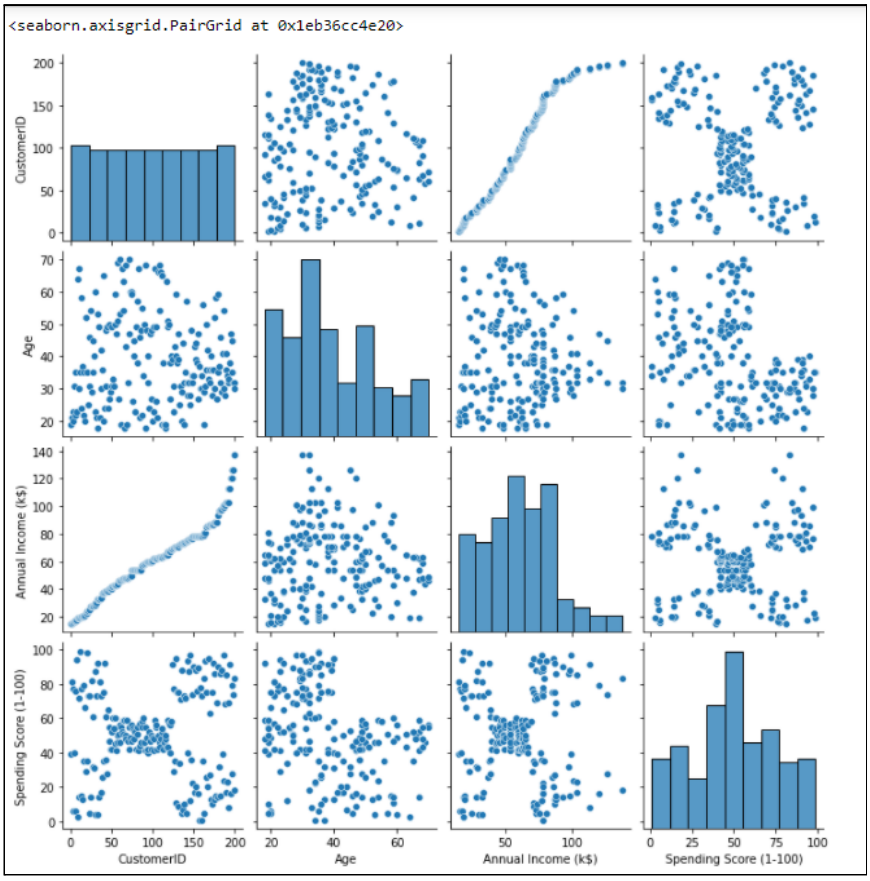
```
    Distance_mat     = distcal.compute_distance(samples)
    sample_ind_needed = np.where(Distance_mat==Distance_mat.min())[0]
    value_to_add     = samples.pop(sample_ind_needed[1])
    samples[sample_ind_needed[0]].append(value_to_add)

    print('Cluster Node 1              :-',progression[sample_ind_needed[0]])
    print('Cluster Node 2              :-',progression[sample_ind_needed[1]])

    progression[sample_ind_needed[0]].append(progression[sample_ind_needed[1]])
    progression[sample_ind_needed[0]] = [progression[sample_ind_needed[0]]]
    v = progression.pop(sample_ind_needed[1])
    m = len(samples)

    print('Progression(Current Sample)     :-',progression)
    print('Cluster attained            :-',progression[sample_ind_needed[0]])
    print('Sample size after clustering    :-',m)
    print('\n')

from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt
Z = linkage(X, 'single')
fig = plt.figure(figsize=(8, 8))
plt.title('Dendrogram')

dn = dendrogram(Z)

plt.scatter(X[:,2], X[:,3], cmap="rainbow")

from sklearn.cluster import AgglomerativeClustering
aggclus = AgglomerativeClustering().fit(X)
aggclus.labels_
```

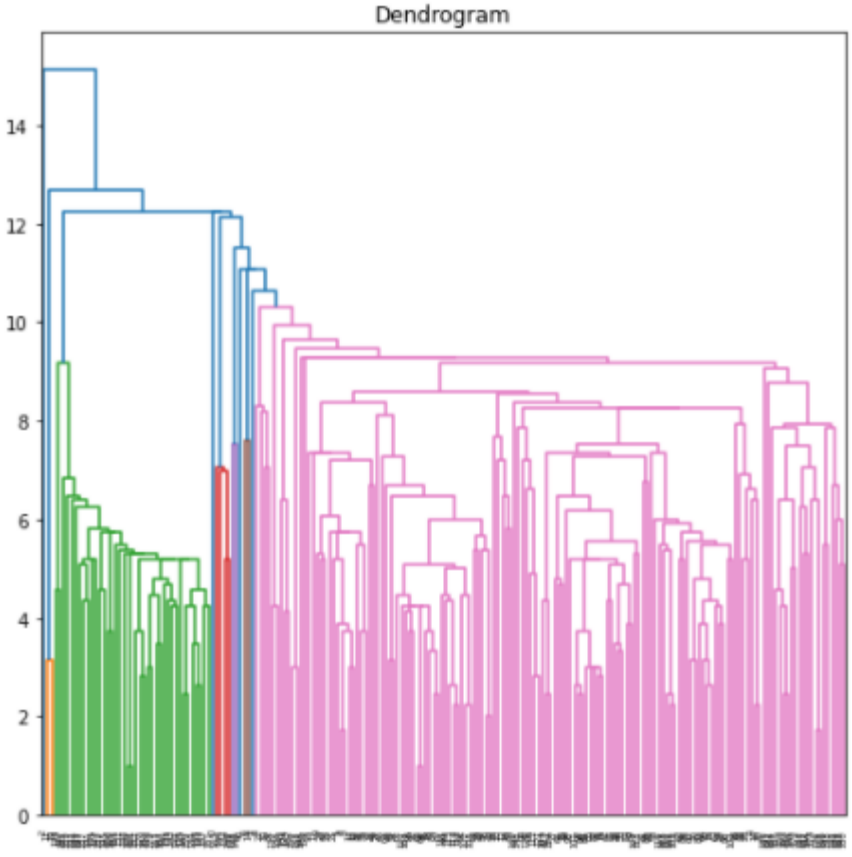
OUTPUT:

```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
```

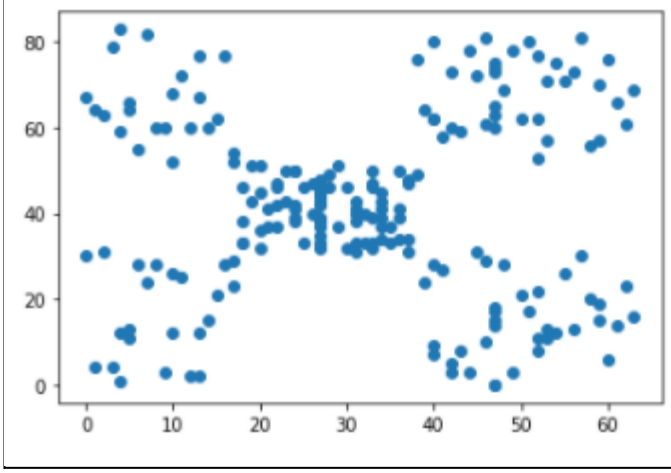
	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19	15	39
1	Male	21	15	81
2	Female	20	16	6
3	Female	23	16	77
4	Female	31	17	40
...	...	...	...	...
195	Female	35	120	79
196	Female	45	126	28
197	Male	32	126	74
198	Male	32	137	18
199	Male	30	137	83

200 rows × 4 columns


<seaborn.axisgrid.PairGrid at 0x1eb36cc4e20>
CustomerID
Age
Annual Income (k$)
Spending Score (1-100)




```
<matplotlib.collections.PathCollection at 0x1eb365528e0>
```



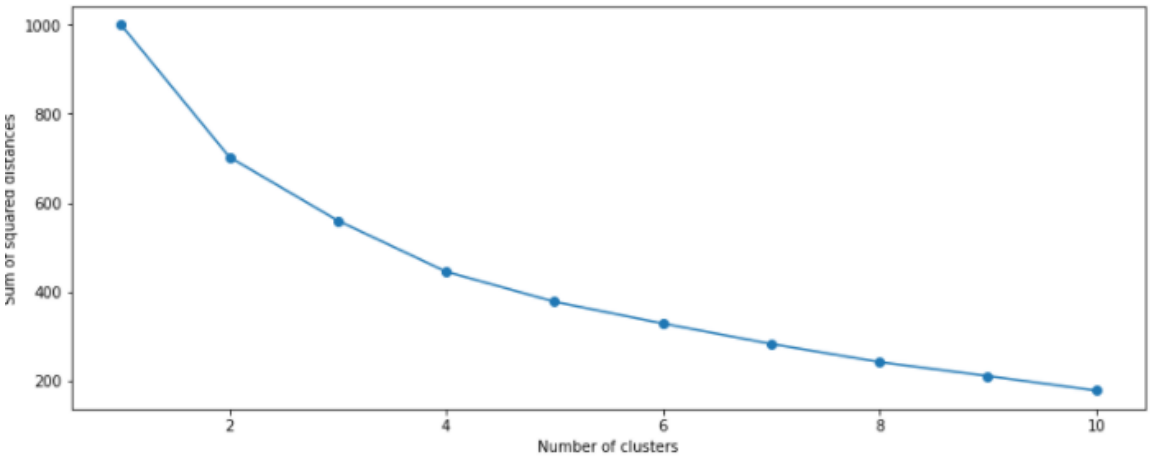
```
array([0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1], dtype=int64)
```

PART C

Code:

```
from sklearn import datasets, preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans

df=pd.read_csv('Mall_Customers.csv')
df = df.apply(LabelEncoder().fit_transform)

scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
pd.DataFrame(scaled_df).describe()
clusters = range(1, 11)
sse=[]
for cluster in clusters:
   model = KMeans(n_clusters=cluster, init='k-means++', max_iter=300, tol=0.0001,
verbose=0,random_state=0)
   model.fit(scaled_df)
   sse.append(model.inertia_)
sse_df = pd.DataFrame(np.column_stack((clusters, sse)),
columns=['cluster', 'SSE'])
fig, ax = plt.subplots(figsize=(13, 5))
ax.plot(sse_df['cluster'], sse_df['SSE'], marker='o')
ax.set_xlabel('Number of clusters')
```

Output:



Conclusion:

Clustering is a method of partitioning a set of data or objects into a set of significant subclasses called clusters. Elbow graph is used to find the optimal value of k, no of clusters.

K Means clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of k means is to group data points into distinct non-overlapping subgroups. It doesn't learn the number of clusters from the data and requires it to be pre-defined.

Hierarchical clustering is a powerful technique that allows you to build tree structures from data similarities. We can see how different sub-clusters relate to each other, and how far apart data points are. The advantage of not having to pre-define the number of clusters gives it quite an edge over kMeans. However, it doesn't work well when we have a huge amount of data.

EXPERIMENT 5

AIM: Implementation of Association rule mining Using

1. Apriori Algorithm

2. FPTree

THEORY:
Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of the dataset. It is based on different rules to discover the interesting relations between variables in the database.
The association rule learning is one of the very important concepts of machine learning, and it is employed in Market Basket analysis, Web usage mining, continuous production, etc. Here market basket analysis is a technique used by the various big retailers to discover the associations between items.
Association rules are created by thoroughly analyzing data and looking for frequent if/then patterns. Then, depending on the following two parameters, the important relationships are observed:

1. Support: Support indicates how frequently the if/then relationship appears in the database.
2. Confidence: Confidence tells about the number of times these relationships have been found to be true.

CODE:
Apriori

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

transaction_df = pd.read_csv('GroceryStoreDataSet.csv')
transaction_df

transaction_df.index.rename('TID', inplace=True)
transaction_df.rename(columns={'MILK,BREAD,BISCUIT' : 'item_list'}, inplace=True)
```

```
trans_df = transaction_df.item_list.str.split(',')
trans_df

def prune(data,supp):

    df = data[data.supp_count >= supp]
    return df

def count_itemset(transaction_df, itemsets):

    count_item = {}
    for item_set in itemsets:
        set_A = set(item_set)
        for row in trans_df:
            set_B = set(row)

            if set_B.intersection(set_A) == set_A:
                if item_set in count_item.keys():
                    count_item[item_set] += 1

                else:
                    count_item[item_set] = 1

    data = pd.DataFrame()
    data['item_sets'] = count_item.keys()
    data['supp_count'] = count_item.values()

    return data

def count_item(trans_items):

    count_ind_item = {}
    for row in trans_items:
        for i in range(len(row)):
            if row[i] in count_ind_item.keys():
                count_ind_item[row[i]] += 1
            else:
                count_ind_item[row[i]] = 1
```

```
    data = pd.DataFrame()
    data['item_sets'] = count_ind_item.keys()
    data['supp_count'] = count_ind_item.values()
    data = data.sort_values('item_sets')
    return data


def join(list_of_items):
    itemsets = []
    i = 1
    for entry in list_of_items:
        proceding_items = list_of_items[i:]
        for item in proceding_items:
            if(type(item) is str):
                if entry != item:
                    tuples = (entry, item)
                    itemsets.append(tuples)
            else:
                if entry[0:-1] == item[0:-1]:
                    tuples = entry+item[1:]
                    itemsets.append(tuples)
        i = i+1
    if(len(itemsets) == 0):
        return None
    return itemsets

def apriori(trans_data,supp=3, con=0.5):
    freq = pd.DataFrame()

    df = count_item(trans_data)

    while(len(df) != 0):

        df = prune(df, supp)

        if len(df) > 1 or (len(df) == 1 and int(df.supp_count >= supp)):
            freq = df

        itemsets = join(df.item_sets)
```

```
        if(itemsets is None):
            return freq

        df = count_itemset(trans_data, itemsets)
    return df

freq_item_sets = apriori(trans_df, 5)
freq_item_sets

def calculate_conf(value1, value2):
    return round(int(value1)/int(value2) * 100, 2)

def strong_rules(freq_item_sets, threshold):

    confidences = {}
    for row in freq_item_sets.item_sets:
        for i in range(len(row)):
            for j in range(len(row)):
                if i != j:
                    tuples = (row[i], row[j])
                    conf = calculate_conf(freq_item_sets[freq_item_sets.item_sets ==
row].supp_count, count_item(trans_df)[count_item(trans_df).item_sets ==
row[i]].supp_count)
                    confidences[tuples] = conf


    conf_df = pd.DataFrame()
    conf_df['item_set'] = confidences.keys()
    conf_df['confidence'] = confidences.values()

    return conf_df[conf_df.confidence >= threshold]

confidence_threshold = int(input()) #50
strong_rules(freq_item_sets, threshold=confidence_threshold)


# ### Rules with confidence level >= 50.0%
```

```
from functools import reduce
import operator

def interesting_rules(freq_item_sets):

    lifts = {}
    prob_of_items = []

    for row in freq_item_sets.item_sets:
        num_of_items = len(row)

        prob_all = freq_item_sets[freq_item_sets.item_sets == row].supp_count / 18
        for i in range(num_of_items):
            prob_of_items.append(count_item(trans_df)[count_item(trans_df).item_sets
== row[i]].supp_count / 18)

        lifts[row] = round(float(prob_all / reduce(operator.mul,
(np.array(prob_of_items)), 1)), 2)

        prob_of_items = []


    lifts_df = pd.DataFrame()
    lifts_df['Rules'] = lifts.keys()
    lifts_df['lift'] = lifts.values()

    return lifts_df

int_rules = interesting_rules(freq_item_sets)
int_rules
```

OUTPUT:

	MILK,BREAD,BISCUIT
0	BREAD,MILK,BISCUIT,CORNFLAKES
1	BREAD,TEA,BOURNVITA
2	JAM,MAGGI,BREAD,MILK
3	MAGGI,TEA,BISCUIT
4	BREAD,TEA,BOURNVITA
5	MAGGI,TEA,CORNFLAKES
6	MAGGI,BREAD,TEA,BISCUIT
7	JAM,MAGGI,BREAD,TEA
8	BREAD,MILK
9	COFFEE,COKE,BISCUIT,CORNFLAKES
10	COFFEE,COKE,BISCUIT,CORNFLAKES
11	COFFEE,SUGER,BOURNVITA
12	BREAD,COFFEE,COKE
13	BREAD,SUGER,BISCUIT
14	COFFEE,SUGER,CORNFLAKES
15	BREAD,SUGER,BOURNVITA
16	BREAD,COFFEE,SUGER
17	BREAD,COFFEE,SUGER
18	TEA,MILK,COFFEE,CORNFLAKES

	item_set	confidence
0	(BISCUIT, BREAD)	50.00
2	(BISCUIT, CORNFLAKES)	50.00
3	(CORNFLAKES, BISCUIT)	50.00
4	(BOURNVITA, BREAD)	75.00
9	(MAGGI, BREAD)	60.00
11	(MILK, BREAD)	75.00
13	(SUGER, BREAD)	66.67
15	(TEA, BREAD)	57.14
17	(COKE, COFFEE)	100.00
18	(COFFEE, CORNFLAKES)	50.00
19	(CORNFLAKES, COFFEE)	66.67
20	(COFFEE, SUGER)	50.00
21	(SUGER, COFFEE)	66.67
22	(MAGGI, TEA)	80.00
23	(TEA, MAGGI)	57.14

	Rules	lift
0	(BISCUIT, BREAD)	0.75
1	(BISCUIT, CORNFLAKES)	1.50
2	(BOURNVITA, BREAD)	1.12
3	(BREAD, COFFEE)	0.56
4	(BREAD, MAGGI)	0.90
5	(BREAD, MILK)	1.12
6	(BREAD, SUGER)	1.00
7	(BREAD, TEA)	0.86
8	(COFFEE, COKE)	2.25
9	(COFFEE, CORNFLAKES)	1.50
10	(COFFEE, SUGER)	1.50
11	(MAGGI, TEA)	2.06

FP TREE

CODE:

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth
dataset = [['f', 'a', 'c', 'd', 'g', 'i', 'm', 'p'],
           ['a', 'b', 'c', 'f', 'l', 'm', 'o'],
           ['b', 'f', 'h', 'j', 'o', 'w'],
           ['b', 'c', 'k', 's', 'p'],
           ['a', 'f', 'c', 'e', 'l', 'p', 'm', 'n']]


te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
fpgrowth(df, min_support=0.6, use_colnames=True, verbose=2) # 3/5 = 60%
```

OUTPUT:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	s	w
0	True	False	True	True	False	True	True	False	True	False	False	False	True	False	False	True	False	False
1	True	True	True	False	False	True	False	False	False	False	False	True	True	False	True	False	False	False
2	False	True	False	False	False	True	False	True	False	True	False	False	False	False	True	False	False	True
3	False	True	True	False	False	False	False	False	False	False	True	False	False	False	False	True	True	False
4	True	False	True	False	True	True	False	False	False	False	False	True	True	True	False	True	False	False

	support	itemsets
0	0.8	(f)
1	0.8	(c)
2	0.6	(p)
3	0.6	(m)
4	0.6	(a)
5	0.6	(b)
6	0.6	(c, f)
7	0.6	(c, p)
8	0.6	(c, m)
9	0.6	(m, f)
10	0.6	(c, m, f)
11	0.6	(m, a)
12	0.6	(c, a)
13	0.6	(f, a)
14	0.6	(c, m, a)
15	0.6	(m, f, a)
16	0.6	(c, f, a)
17	0.6	(c, m, f, a)

CONCLUSION: We learnt about association rule mining and the two different algorithms that can be used - Apriori and FP Tree. We then learn about the uses of this algorithm and implemented the algorithm in a python program.

EXPERIMENT 6

Subject: DMW
Group Members:
Aryan Parekh - 60004190013
Junaid Girkar – 60004190057
Kanaad Deshpande – 60004190058
Manish Jha - 60004190066

Aim
1. Making information package diagram
2. Design dimensional data model i.e. Star schema, Snowflake schema and Fact Constellation schema (if applicable)

Theory
Information Package Diagram
The information package diagram is a novel idea for determining and recording information requirements for a data warehouse. This concept helps us to give a concrete form to the various insights, nebulous thoughts, and opinions expressed during the process of collecting requirements. The information packages, put together while collecting requirements, are very useful for taking the development of the data warehouse to the next phases.
An information package diagram defines the relationships between subject matter and key performance measures. The information package diagram has a highly targeted purpose, providing a focused scope for user requirements. Because information package diagrams target what the users want, they are effective in facilitating communication between the technical staff and the users, indicating any inconsistencies between the requirements and what the data warehouse will deliver.

Facts
Facts and dimensions are data warehousing terms. A fact is a quantitative piece of information -such as a sale or a download. Facts are stored in fact tables, and have a foreign key relationship with a number of dimension tables.
Where multiple fact tables are used, these are arranged as a fact constellation schema. A fact table typically has two types of columns: those that contain facts and those that are a foreign key to dimension tables. The primary key of a fact table is usually a composite key that is made up of all of its foreign keys. Fact tables contain the content of the data warehouse and store different types of measures like additive, non-additive, and semi additive measures.
Fact tables provide the (usually) additive values that act as independent variables by which dimensional attributes are analyzed. Fact tables are often defined by their grain. The grain of a fact table represents the most atomic level by which the facts may be defined. The grain of a sales fact table might be stated as "sales volume by day by product by store". Each record in this fact table is therefore uniquely defined by a day, product and store. Other dimensions might be members of this fact table

(such as location/region) but these add nothing to the uniqueness of the fact records. These "affiliate dimensions" allow for additional slices of the independent facts but generally provide insights at a higher level of aggregation (a region contains many stores).

Dimensions

Dimensions are companions to facts, and describe the objects in a fact table. A dimension is thus, a structure that categorizes facts and measures in order to enable users to answer business questions. Commonly used dimensions are people, products, place and time.

In a data warehouse, dimensions provide structured labelling information to otherwise unordered numeric measures. The dimension is a data set composed of individual, non overlapping data elements. The primary functions of dimensions are threefold: to provide filtering, grouping and labelling.

Star Schema

Star Schema in a data warehouse, in which the center of the star can have one fact table and a number of associated dimension tables. It is known as star schema as its structure resembles a star. The Star Schema data model is the simplest type of Data Warehouse schema. It is also known as Star Join Schema and is optimized for querying large data sets.

Characteristics of Star Schema:

• Every dimension in a star schema is represented with the only one-dimension table.
• The dimension table should contain the set of attributes.
• The dimension table is joined to the fact table using a foreign key
• The dimension table are not joined to each other
• Fact table would contain key and measure
• The Star schema is easy to understand and provides optimal disk usage. • The dimension tables are not normalized. For instance, in the above figure, Country_ID does not have a Country lookup table as an OLTP design would have.
• The schema is widely supported by BI Tools

Advantages of Star Schema

• Join logic of star schema is quite cinch in comparison to other join logic which are needed to fetch data from a transactional schema that is highly normalized. • In comparison to a transactional schema that is highly normalized, the star schema makes simpler common business reporting logic, such as as-of reporting and period over-period.
• Star schema is widely used by all OLAP systems to design OLAP cubes efficiently. In fact, major OLAP systems deliver a ROLAP mode of operation which can use a star schema as a source without designing a cube structure.

Disadvantages of Star Schema –

• Data integrity is not enforced well since in a highly denormalized schema state. • Not flexible in terms of analytical needs as a normalized data model. • Star schemas

don't reinforce many-to-many relationships within business entities – at least not frequently.

Snowflake Schema

Snowflake Schema in a data warehouse is a logical arrangement of tables in a multidimensional database such that the ER diagram resembles a snowflake shape. A Snowflake Schema is an extension of a Star Schema, and it adds additional dimensions. The dimension tables are normalized which splits data into additional tables.

Characteristics of Snowflake Schema

• The main benefit of the snowflake schema is that it uses smaller disk space. • Easier to implement a dimension is added to the Schema

• Due to multiple tables query performance is reduced

• The primary challenge that you will face while using the snowflake Schema is that you need to perform more maintenance efforts because of the more lookup tables.

Advantages of snowflake schema

• It provides structured data which reduces the problem of data integrity. • It uses small disk space because data is highly structured.

Disadvantages of snowflake schema

• Snowflaking reduces space consumed by dimension tables but compared with the entire data warehouse the saving is usually insignificant.

• Avoid snowflaking or normalization of a dimension table, unless required and appropriate.

• Do not snowflake hierarchies of one dimension table into separate tables. Hierarchies should belong to the dimension table only and should never be snowflakes. • Multiple hierarchies that can belong to the same dimension have been designed at the lowest possible detail.

Fact Constellation Schema/Galaxy Schema

Fact Constellation Schema is a schema that represents a multidimensional model of tables. This schema is a group of different fact tables that have few similar dimensional tables. It can be represented as a group of multiple star schemas and thus, it is also called a Galaxy schema. Fact schema is the most frequently used schema to design a Data warehouse and also, it is a little more complicated than the star and snowflake schema model.

Fact constellation schema is a tool of analytical processing via online, which has a huge group of the number of fact tables that share dimensional tables, also aggregated as a group of stars. We can also call it an extension of the star constellation model.

Characteristics of Fact Constellation Schema

• A fact constellation schema can have multiple fact tables associated with it.

• It is commonly used as a schema for designing data warehouses and it is much complicated than any other schema such as star and snowflake schema.

• We can create a constellation schema from a star schema by splitting them into one or more, star schemas.
• It is also said that a fact constellation schema can have many fact tables and a shared dimensional table.

Advantages of Fact Constellation Schema

• Tables are subdivided into fact and dimensional to understand the relationship between them.
• It is a flexible schema that makes users use it.
• Here dimensional tables are shared by the number of fact tables.
• It is a normalized form of snowflake and star schema.
• We can access the data in the database using complex queries.

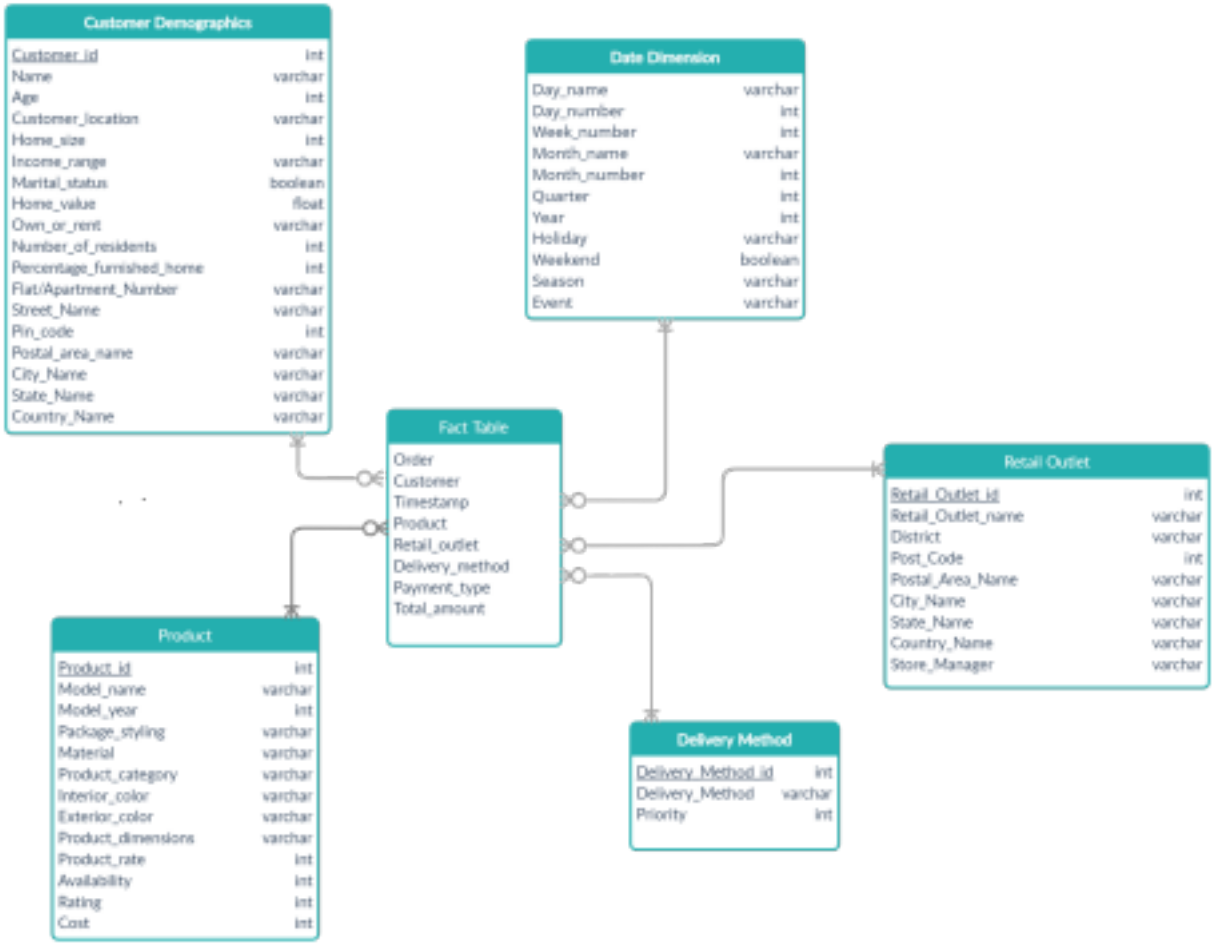
Disadvantages of Fact Constellation Schema

• It is difficult to understand as it is a very complex schema to implement. • It uses more space in the database compared to the star schema.
• It has many joins between dimensional and fact tables and thus it is difficult to understand.
• This is difficult to maintain and operate.

1. INFORMATION PACKAGE DIAGRAM

Time	Product	Customer Demographics	Retail Outlet	Delivery Method
Day_name	Model_name	Customer ID	Retail_Outlet_id	delivery_method_id
Day_number	Model_year	Name	Retail_Outlet_name	delivery_method
Week_number	Package_styling	Age	District	priority
Month_name	Material	Customer_Location	Post_Code	
Month_number	Product_category	Home_Size	Postel_Area_Name	
Quarter	Interior_color	Income_Range	City_Name	
Year	Exterior_color	Maritial_status	State_Name	
Holiday	Product_dimension	Home_value	Country_Name	
Weekend	Product_weight	Own_or_Rent	Store Manager	
Season	Availability	Number_of_residents		
Event	Rating	Percentage_furnished_home		
	Cost	Flat/Apartment_Number		
		Street_Name		
		Pin_Code		
		Postal_area_name		
		City_Name		
		State_Name		
		Country_Name		
Facts - Order, customer, timestamp, product, retail outlet, delivery method, payment type, total amount				

2. STAR SCHEMA



Conclusion

In this experiment, we constructed an information package diagram and the star schema for an IKEA warehouse. The information package diagram helped create a system to record information of everything necessary to the IKEA warehouse and the star schema helped visualize all the different relationships in the information package diagram. We learned about various data warehousing terms such as facts. Facts help us analyze dimensional attributes. We also learned about the various schema structures available such as star schema, snowflake schema and constellation schema

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

JUNAID GIRKAR
60004190057
TE COMPS A4

DWM
LAB EXPERIMENT NO. 07

AIM: Perform OLAP operations such as Roll up, Drill down, Slice and Dice, Pivot on Data Warehouse.

Theory:

OLAP is an acronym for On Line Analytical Processing. Online Analytical Processing: An OLAP system manages large amounts of historical data, provides Facilities for summarization and aggregation, and stores and manages information at different levels of granularity.
OLAP operations:

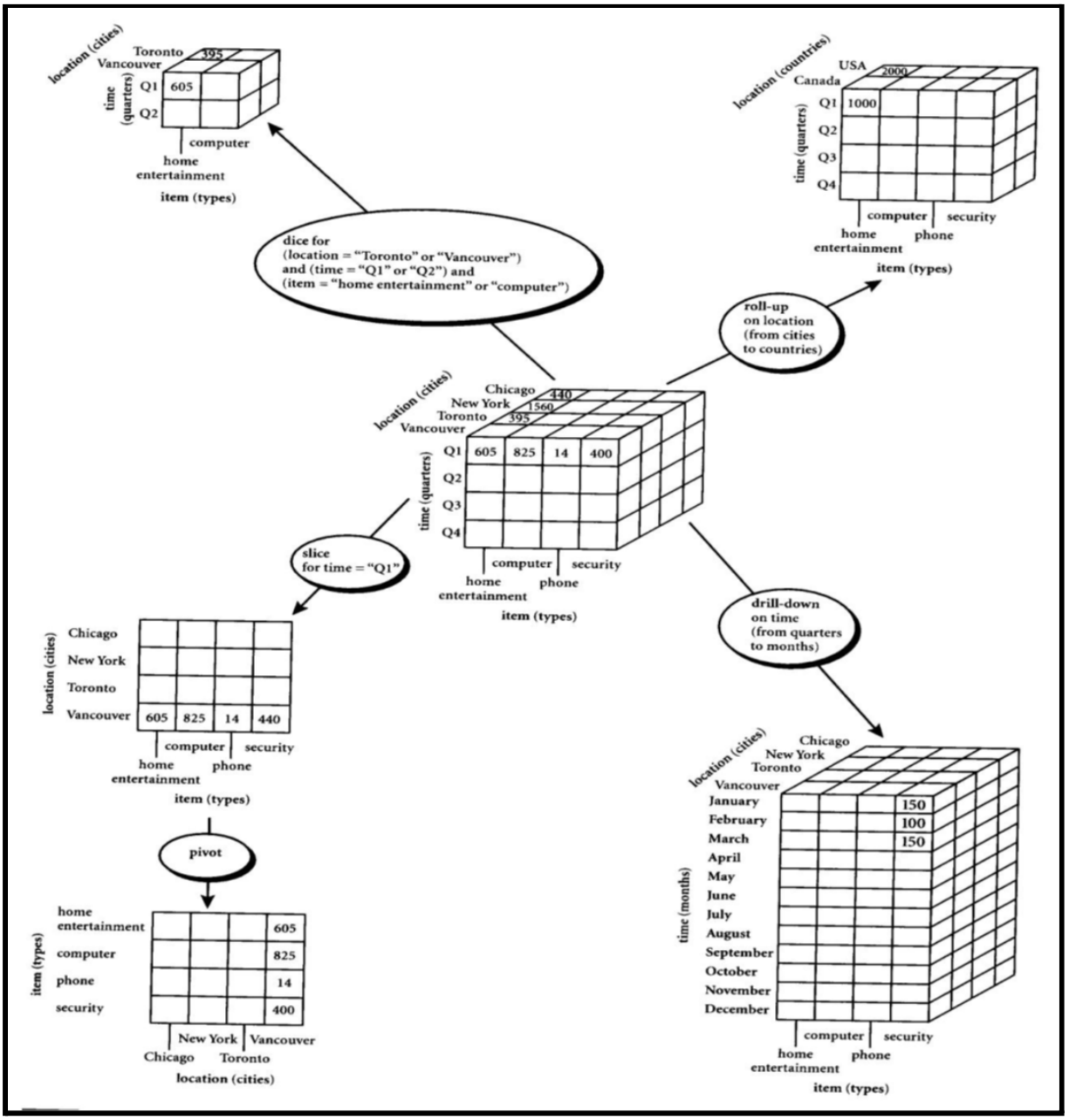
Slice: A slice is a subset of a multidimensional array corresponding to a single value for one or more members of the dimensions not in the subset.

Dice: The dice operation is a slice on more than two dimensions of a data cube (or more than two consecutive slices).

Drill Down/Up: Drilling down or up is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down).

Roll-up: A roll-up involves computing all of the data relationships for one or more dimensions. To do this, a computational relationship or formula might be defined.

Pivot: To change the dimensional orientation of a report or page display.


location (cities)
Toronto
Vancouver
395
time (quarters)
Q1
605
Q2
computer
home entertainment
item (types)
location (countries)
USA
2000
Canada
Q1
1000
Q2
Q3
Q4
time (quarters)
computer
security
home entertainment
phone
item (types)
dice for
(location = "Toronto" or "Vancouver")
and (time = "Q1" or "Q2") and
(item = "home entertainment" or "computer")
roll-up
on location
(from cities
to countries)
location (cities)
Chicago
440
New York
1560
Toronto
395
Vancouver
Q1
605
825
14
400
Q2
Q3
Q4
time (quarters)
computer
security
home entertainment
phone
item (types)
slice
for time = "Q1"
drill-down
on time
(from quarters
to months)
location (cities)
Chicago
New York
Toronto
Vancouver
605
825
14
440
computer
security
home entertainment
phone
item (types)
pivot
item (types)
home entertainment
computer
phone
security
605
825
14
400
New York
Vancouver
Chicago
Toronto
location (cities)
location (cities)
Chicago
New York
Toronto
Vancouver
January
150
February
100
March
150
April
May
June
July
August
September
October
November
December
time (months)
computer
security
home entertainment
phone
item (types)

10/12/21

DWM

EXPERIMENT - 7

OLAP OPERATIONS

JUNAID. GIRKAR
6000419005 7
TE COMPS A4

Exercise 1

Consider a data warehouse for a hospital, where there are three dimensions :

(i) Doctor
(ii) Patient
(iii) Time

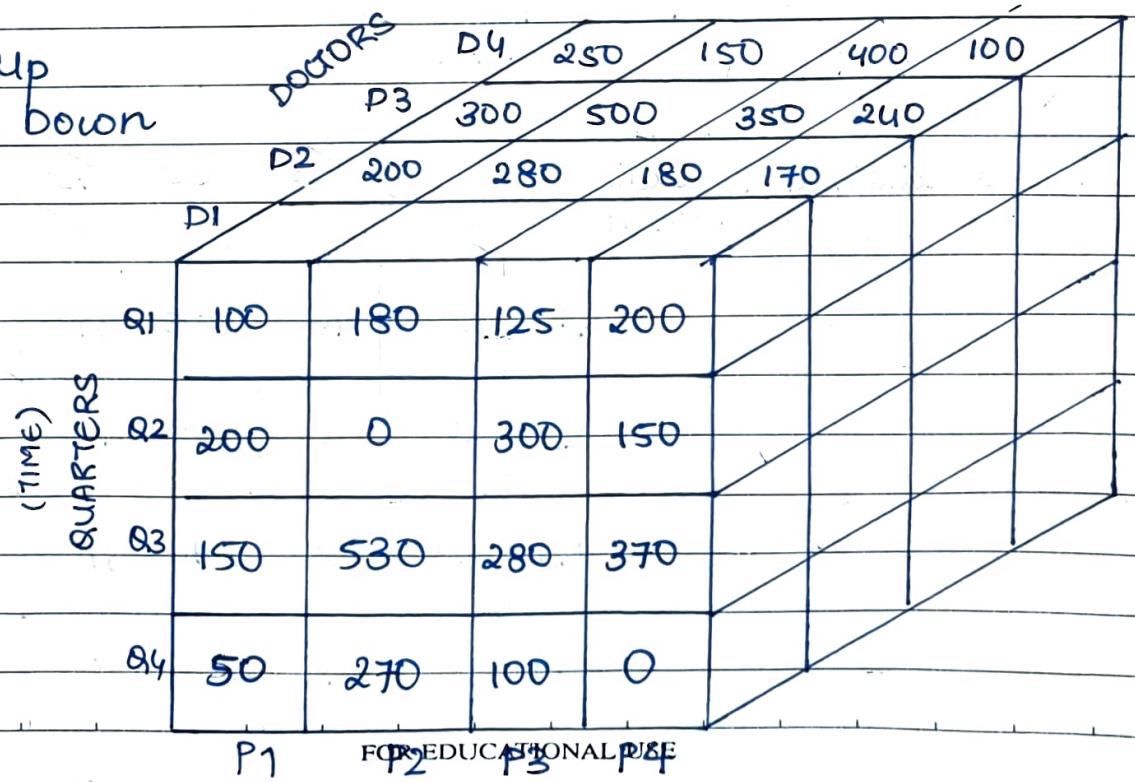
with two measures

a) Count
b) Charge

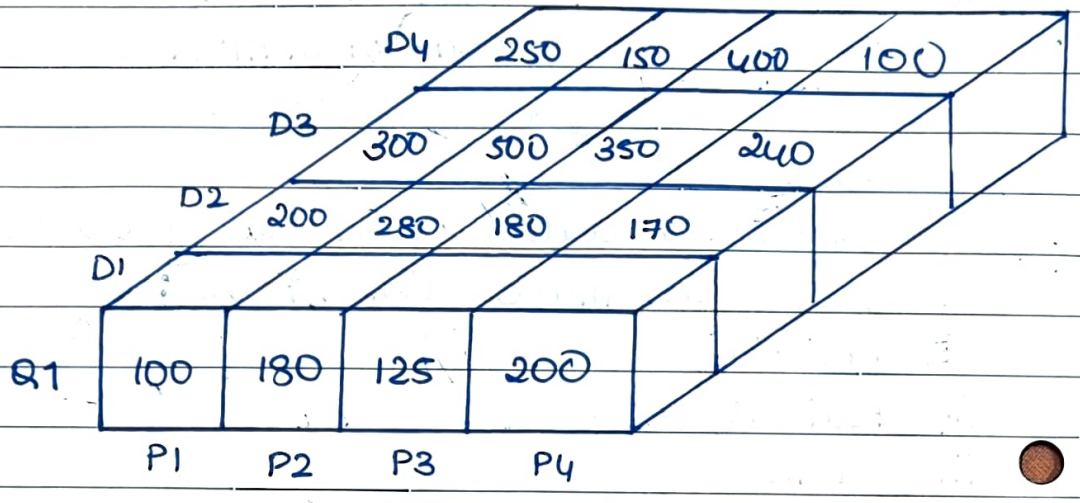
where charge is the fee that the doctor charges a patient for a visit. Using the above example describe the following operations :

i) slice
ii) Dice
iii) Roll up
iv) Drill down
v) Pivot

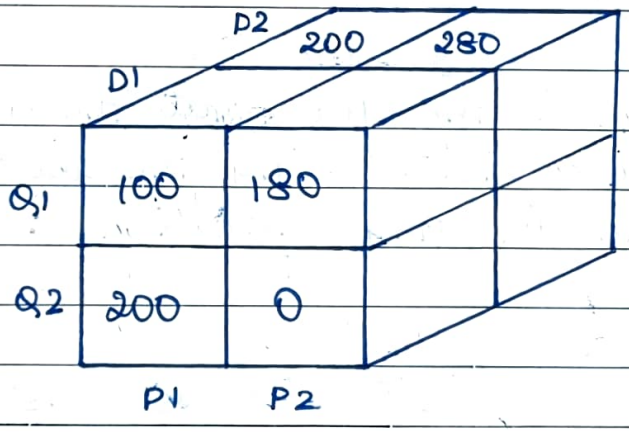
ANS.



SLICE

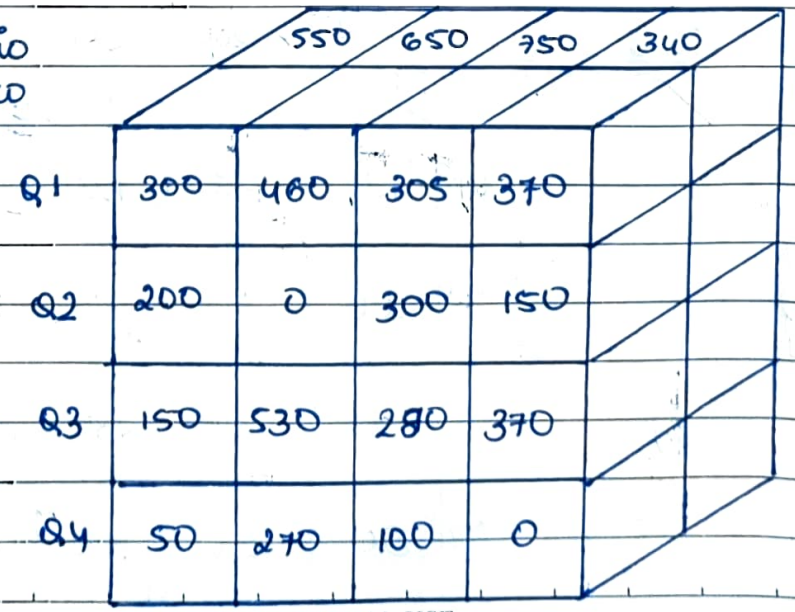


DICE



ROLL UP

Specialization { Cardio, Neuro



2

DRILL DOWN

	D1	D2	D3	D4
	200	280	180	170
	300	500	350	240
	250	150	400	100
January	70	90	75	100
February	15	45	25	50
March	15	45	25	50
April	50	0	100	75
May	100	0	150	25
June	50	0	50	50
July	50	200	70	100
August	50	100	140	200
September	50	230	70	70
October	10	90	50	0
November	20	100	30	0
December	20	80	20	0
	P1	P2	P3	P4

PIVOT

	P1	P2	P3	P4
	200	170	240	100
	125	180	350	400
	180	280	500	150
	100	200	300	250
	D1	D2	D3	D4

EXERCISE 2

To create Pivot of Table using MS Excel

Follow these steps …

1. Start with M.S Excel.
2. In excel sheet create 4 columns PRODUCT, ORIGIN, DAY OF SALE, SOLD UNITS (FACT COLUMN).
3. Insert around fifty rows of data.
4. Save the table data.
5. Go to Insert Tab-> click on Pivot Table-> New work sheet-> Ok.
6. Right side you will find pivot table fields.

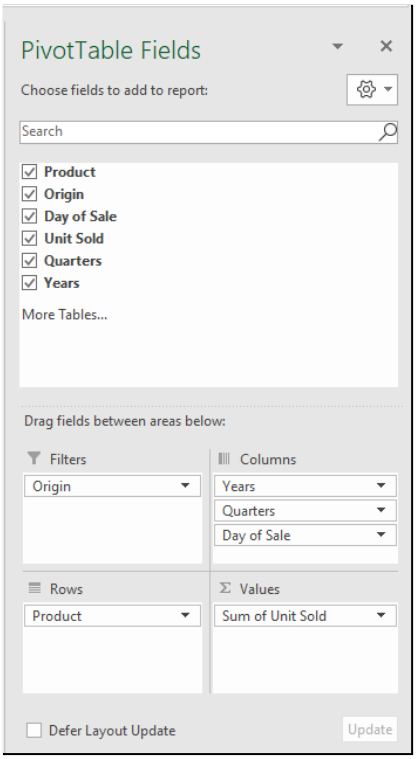
It contains all columns of our table that we created.
Select product in rows,
Days in column,
Unit sold in ∑ values.
Later apply filter using Origin.

Also we can flip the rows & columns or combine together as rows only to see different views of same data.

Dataset:-

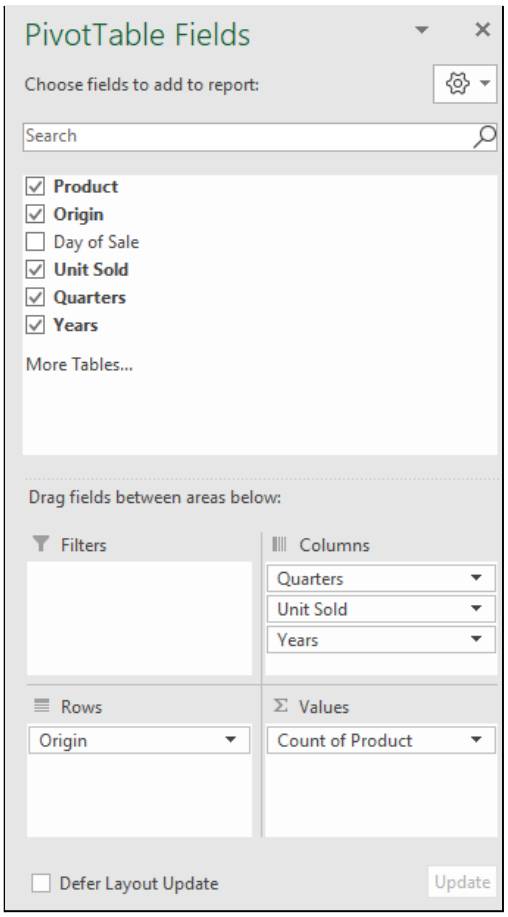
	A	B	C	D
1	**Product**	**Origin**	**Day of Sale**	**Unit Sold**
2	WHITE HANGING HEART T-LIGHT HOLDER	United Kingdom	01-12-2010 8.26	6
3	WHITE METAL LANTERN	United Kingdom	01-12-2010 8.26	6
4	CREAM CUPID HEARTS COAT HANGER	United Kingdom	01-12-2010 8.26	8
5	KNITTED UNION FLAG HOT WATER BOTTLE	United Kingdom	01-12-2010 8.26	6
6	RED WOOLLY HOTTIE WHITE HEART.	United Kingdom	01-12-2010 8.26	6
7	SET 7 BABUSHKA NESTING BOXES	United Kingdom	01-12-2010 8.26	2
8	GLASS STAR FROSTED T-LIGHT HOLDER	United Kingdom	01-12-2010 8.26	6
9	HAND WARMER UNION JACK	United Kingdom	01-12-2010 8.28	6
10	HAND WARMER RED POLKA DOT	United Kingdom	01-12-2010 8.28	6
11	ASSORTED COLOUR BIRD ORNAMENT	United Kingdom	01-12-2010 8.34	32
12	POPPY'S PLAYHOUSE BEDROOM	United Kingdom	01-12-2010 8.34	6
13	POPPY'S PLAYHOUSE KITCHEN	United Kingdom	01-12-2010 8.34	6
14	FELTCRAFT PRINCESS CHARLOTTE DOLL	United Kingdom	01-12-2010 8.34	8
15	IVORY KNITTED MUG COSY	United Kingdom	01-12-2010 8.34	6
16	BOX OF 6 ASSORTED COLOUR TEASPOONS	United Kingdom	01-12-2010 8.34	6
17	BOX OF VINTAGE JIGSAW BLOCKS	United Kingdom	01-12-2010 8.34	3
18	BOX OF VINTAGE ALPHABET BLOCKS	United Kingdom	01-12-2010 8.34	2
19	HOME BUILDING BLOCK WORD	United Kingdom	01-12-2010 8.34	3
20	LOVE BUILDING BLOCK WORD	United Kingdom	01-12-2010 8.34	3
21	RECIPE BOX WITH METAL HEART	United Kingdom	01-12-2010 8.34	4
22	DOORMAT NEW ENGLAND	United Kingdom	01-12-2010 8.34	4
23	JAM MAKING SET WITH JARS	United Kingdom	01-12-2010 8.34	6
24	RED COAT RACK PARIS FASHION	United Kingdom	01-12-2010 8.34	3
25	YELLOW COAT RACK PARIS FASHION	United Kingdom	01-12-2010 8.34	3
26	BLUE COAT RACK PARIS FASHION	United Kingdom	01-12-2010 8.34	3
27	BATH BUILDING BLOCK WORD	United Kingdom	01-12-2010 8.35	3
28	ALARM CLOCK BAKELIKE PINK	France	01-12-2010 8.45	24
29	ALARM CLOCK BAKELIKE RED	France	01-12-2010 8.45	24

CASE 1:



Origin	(All)														
Sum of Unit Sold	Column La														
	Jan		Feb	Feb Total	Mar Total	Apr Total	May Total	Jun Total	Jul Total	Aug Total	Sep Total	Oct Total	Nov Total	Dec Total	Grand Total
	2011	2011 Total	2011												
20 DOLLY PEGS RETROSPOT	77	77	256	256	179	735		-2	77	203	125	210	127	457	2444
200 BENDY SKULL STRAWS	14	14	2	2	16	194	218	23	14	102	2	2	28	45	660
200 RED + WHITE BENDY STRA	78	78	46	46	30	115	35	48	130	121	136	31	70	78	918
20713 wrongly marked												-200			-200
3 BIRDS CANVAS SCREEN						1			1						2
3 BLACK CATS W HEARTS BLANK CARD										6			18		24
3 DRAWER ANTIQUE WHITE WOOD CABINET										25	163	160	264	89	701
3 GARDENIA MORRIS BOXED C	21	21	3	3	6		9	2	1	4	12	330	96	13	497
3 HEARTS HANGING DECORAT	10	10	-5	-5	-1	10	19	20	23	416	301	703	1318	624	3438
3 HOOK HANGER MAGIC GARI	264	264	175	175	152	221	195	45	58	342	139	-2		100	1689
3 HOOK PHOTO SHELF ANTIQU	80	80	98	98	150	67	173	391	388	106	27	9	103	107	1699
3 PIECE SPACEBOY COOKIE CU	323	323	427	427	406	77	73	50	193	131	251	154	276	253	2614
3 PINK HEN+CHICKS IN BASKE	5	5	3	3	3	1									12
3 RAFFIA RIBBONS 50'S CHRISTMAS											478	576	887	154	2095
3 RAFFIA RIBBONS VINTAGE CHRISTMAS											361	212	189	65	827
3 ROSE MORRIS BOXED CANDL	100	100	20	20		4		5	1		1	271	573	138	1113
3 STRIPEY MICE FELTCRAFT	49	49	223	223	225	86	184	321	140	310	272	302	421	486	3019
3 TIER CAKE TIN GREEN AND C	24	24	39	39	58	15	52	29	27	21	71	20	40	37	433
3 TIER CAKE TIN RED AND CRE	125	125	-47	-47	128		88	23	7	15	70	27	95	30	561
3 TIER SWEETHEART GARDEN S	3	3	4	4	11	6	8	7	2	5	5	0	1	2	54
3 TRADITIONAI BISCUIT CUTTE	95	95	79	79	86	78	70	78	152	47	125	95	178	90	1173
3 TRADITIONAL COOKIE CUTTE	144	144												18	162
3 WHITE CHOC MORRIS BOXEI	113	113	4	4			8	5	1		15	198	465	141	950
3 WICK CHRISTMAS BRIAR CAI	-2	-2												1	-1

CASE 2:



Count of Product	Column Labels				
	Qtr1	Qtr2	Qtr3	Qtr4	Grand Total
Row Labels					
Australia	328	309	428	194	1259
Austria	39	79	143	140	401
Bahrain	1	17		1	19
Belgium	346	503	511	709	2069
Brazil		32			32
Canada	10	58	83		151
Channel Islands	224	127	207	200	758
Cyprus	221	49	1	351	622
Czech Republic	15	2		13	30
Denmark	39	125	82	143	389
EIRE	1194	1593	2644	2765	8196
European Community		32	29		61
Finland	239	56	178	222	695
France	1651	1571	2079	3256	8557
Germany	1780	1926	2414	3375	9495
Greece	54	31	25	36	146
Hong Kong	59	141	51	37	288
Iceland	29	42	22	89	182
Israel	30	6	225	36	297
Italy	213	58	122	410	803
Japan	113	73	51	121	358
Lebanon	45				45
Lithuania				35	35

Conclusion:.

From this experiment, we learn about the OLAP Operations. We also understand how the OLAP system manages a large amount of historical data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity. We also learn about the PivotTable function in MS Excel and how it helps us in making meaningful conclusions.

EXPERIMENT 8

Aim: Implementation of Page Rank Algorithm

Theory:
PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

Working:
Each link from one page (A) to another (B) casts a so-called vote, the weight of which depends on the collective weight of all the pages that link to page A. And we can't know their weight till we calculate it, so the process goes in cycles.

The mathematical formula of the original PageRank is the following:

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right)$$

Where A, B, C, and D are some pages, L is the number of links going out from each of them, and N is the total number of pages in the collection (i.e. on the Internet).

As for d, d is the so-called damping factor. Considering that PageRank is calculated simulating the behavior of a user who randomly gets to a page and clicks links, we apply this damping d factor as the probability of the user getting bored and leaving a page.

As you can see from the formula, if there are no pages pointing to the page, its PR will be not zero

$$PR(A) = \frac{1-d}{N}$$

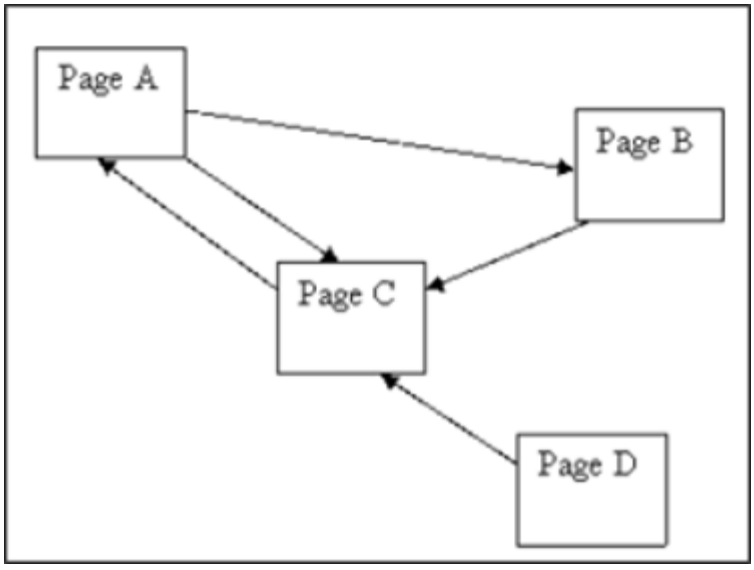
As there's a probability that the user could get to this page not from some other pages but, say, from bookmarks.

Code:

```
import numpy as np

def page_rank_algorithm(graph,damping_factor):
    outgoing = dict()
    incoming_nodes = dict()
    coefficients = dict()
    # Outgoing Nodes
    for i in range(len(graph)):
        outgoing[i]=0

    for i,node in enumerate(graph):
        for edge in node:
            if edge:
                outgoing[i] += 1

    # Incoming Nodes
    for i in range(len(graph)):
        temp=[]
        for node in graph:
            if node[i]:
                temp.append(node)
        incoming_nodes[i] = temp

    # Coefficient Matrix
    for i,node in enumerate(graph):
        temp = []
        for j,other_node in enumerate(graph):
            if other_node in incoming_nodes[i]:
                temp.append(damping_factor*(1.0/outgoing[j]))
```

```
            elif i == j:
                temp.append(-1)
            else:
                temp.append(0)
        coefficients[i] = temp

    coefficients_list = []
    for key,value in coefficients.items():
        coefficients_list.append(value)

    constant_matrix = []
    for i in range(len(graph)):
        constant_matrix.append(damping_factor-1)

    pageranks = np.linalg.solve(np.array(coefficients_list),np.array(constant_matrix))

    print()
    for i,rank in enumerate(pageranks):
        print('Page Rank of {} is {:.4f}'.format(chr(65+i), rank))

def main():
    n = int(input('Enter the number of nodes : '))

    d= float(input('Enter the damping factor : '))


    graph = []
    print('Enter Adjacency Matrix with terms separated by a space : ')
    for i in range(n):
        temp_list = input().split(' ')
        graph.append(list(map(int,temp_list)))

    page_rank_algorithm(graph,d)

main()
```

Graph:



Output:

```
Enter the number of nodes : 4
Enter the damping factor : 0.85
Enter Adjacency Matrix with terms separated by a space :
0 1 1 0
0 0 1 0
1 0 0 0
0 0 1 0

Page Rank of A is 1.4901
Page Rank of B is 0.7833
Page Rank of C is 1.5766
Page Rank of D is 0.1500
```

Conclusion:

Page Rank algorithm is one of the first algorithms in the history of Google search engine and is used to rank web pagesIt is a Web Structure Mining algorithm. Page Rank calculated is based on the incoming links (Backlinks). A dampening factor is used to avoid the rank sink problem. It's only drawback is that it favours old pages rather than the new ones but is still a widely used algorithm because of its efficiency.

EXPERIMENT 9

Aim: Implementation of HITS Algorithm

Theory:
Hyperlink Induced Topic Search (HITS) Algorithm is a Link Analysis Algorithm that rates webpages, developed by Jon Kleinberg. This algorithm is used to the web link-structures to discover and rank the webpages relevant for a particular search.
HITS uses hubs and authorities to define a recursive relationship between webpages. Before understanding the HITS Algorithm, we first need to know about Hubs and Authorities.

Given a query to a Search Engine, the set of highly relevant web pages are called Roots. They are potential Authorities.
Pages that are not very relevant but point to pages in the Root are called Hubs. Thus, an Authority is a page that many hubs link to whereas a Hub is a page that links to many authorities.

Working:
In the HITS algorithm, the first step is to retrieve the most relevant pages to the search query. This set is called the root set and can be obtained by taking the top pages returned by a text-based search algorithm. A base set is generated by augmenting the root set with all the web pages that are linked from it and some of the pages that link to it. The web pages in the base set and all hyperlinks among those pages form a focused subgraph. The HITS computation is performed only on this focused subgraph. According to Kleinberg the reason for constructing a base set is to ensure that most (or many) of the strongest authorities are included.

Authority and hub values are defined in terms of one another in a mutual recursion. An authority value is computed as the sum of the scaled hub values that point to that page. A hub value is the sum of the scaled authority values of the pages it points to. Some implementations also consider the relevance of the linked pages.

The algorithm performs a series of iterations, each consisting of two basic steps:

- **Authority update**: Update each node's authority score to be equal to the sum of the hub scores of each node that points to it. That is, a node is given a high authority score by being linked from pages that are recognized as Hubs for information.
- **Hub update**: Update each node's hub score to be equal to the sum of the authority scores of each node that it points to. That is, a node is given a high hub score by linking to nodes that are considered to be authorities on the subject.

The Hub score and Authority score for a node is calculated with the following algorithm:

- ➔ Start with each node having a hub score and authority score of 1.
- ➔ Run the authority update rule
- ➔ Run the hub update rule
- ➔ Normalize the values by dividing each Hub score by the square root of the sum of the squares of all Hub scores, and dividing each Authority score by the square root of the sum of the squares of all Authority scores.
- ➔ Repeat from the second step as necessary.

Comparison:
HITS, like Page and Brin's PageRank, is an iterative algorithm based on the linkage of the documents on the web. However it does have some major differences:

- It is query dependent, that is, the (Hubs and Authority) scores resulting from the link analysis are influenced by the search terms;
- As a corollary, it is executed at query time, not at indexing time, with the associated hit on performance that accompanies query-time processing.
- It is not commonly used by search engines. (Though a similar algorithm was said to be used by Teoma, which was acquired by Ask Jeeves/Ask.com.)
- It computes two scores per document, hub and authority, as opposed to a single score;
- It is processed on a small subset of ‘relevant’ documents (a 'focused subgraph' or base set), not all documents as was the case with PageRank.

Advantages:
1. HITS scores due to its ability to rank pages according to the query string, resulting in relevant authority and hub pages.
2. HITS is sensitive to user queries (as compared to PageRank).
3. Important pages are obtained on the basis of calculated authority and hub value.

Disadvantages:

1. Since HITS is a query dependent algorithm the query time evaluation is expensive.
2. The rating or scores of authorities and hubs could rise due to flaws done by the web page designer. HITS assumes that when a user creates a web page he links a hyperlink from his page to another authority page, as he honestly believes that the authority page is in some way related to his page (hub).
3. Topic drift occurs when there are irrelevant pages in the root set and they are strongly connected. Since the root set itself contains non-relevant pages, this will reflect on the pages in the base set.

Algorithm:

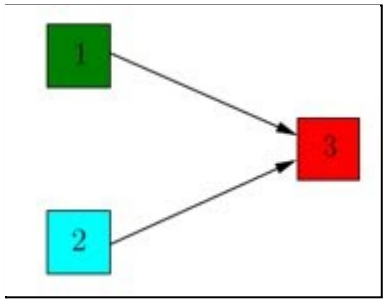
```
G := set of pages
for each page p in G do
   p.auth = 1 // p.auth is the authority score of the page p
   p.hub = 1 // p.hub is the hub score of the page p
for step from 1 to k do // run the algorithm for k steps
   norm = 0
   for each page p in G do  // update all authority values first
      p.auth = 0
      for each page q in p.incomingNeighbors do // p.incomingNeighbors is the set of pages that
link to p
         p.auth += q.hub
      norm += square(p.auth) // calculate the sum of the squared auth values to normalise
   norm = sqrt(norm)
   for each page p in G do  // update the auth scores
      p.auth = p.auth / norm  // normalise the auth values
   norm = 0
   for each page p in G do  // then update all hub values
      p.hub = 0
      for each page r in p.outgoingNeighbors do // p.outgoingNeighbors is the set of pages that p
links to
         p.hub += r.auth
      norm += square(p.hub) // calculate the sum of the squared hub values to normalise
   norm = sqrt(norm)
   for each page p in G do  // then update all hub values
      p.hub = p.hub / norm   // normalise the hub values
```

Code:

```
from math import sqrt

def hits_algorithm(num_nodes, graph, iterations):
    authority_scores = dict()
    hub_scores = dict()
    for i in range(len(graph)):
        authority_scores[i] = 1
        hub_scores[i] = 1
    incoming_nodes = dict()
    for i in range(len(graph)):
        temp=[]
        for node in graph:
            if node[i]:
                temp.append(node)
        incoming_nodes[i] = temp
    outgoing_nodes = dict()
    for i,node in enumerate(graph):
        temp = []
        for j,edge in enumerate(node):
            if edge:
                temp.append(graph[j])
        outgoing_nodes[i] = temp
    print()
    for k in range(iterations):
        print('Iteration : ',k+1)
        print('Authority Score')
        normalization_value = 0
        for i,node in enumerate(graph):
            authority_scores[i]=0
            for j,other_node in enumerate(graph):
                if other_node in incoming_nodes[i]:
                    authority_scores[i] += hub_scores[j]
            normalization_value += (authority_scores[i]**2)
        normalization_value = sqrt(normalization_value)
        for i in range(num_nodes):
            authority_scores[i] /= normalization_value
            print('{} :{:.2f}'.format(chr(65+i),authority_scores[i]),end=' | ')
        print()
        print('Hub Score')
        normalization_value = 0
        for i,node in enumerate(graph):
            hub_scores[i]=0
```

```
            for j,other_node in enumerate(graph):
                if other_node in outgoing_nodes[i]:
                    hub_scores[i] += authority_scores[j]
            normalization_value += (hub_scores[i]**2)
        normalization_value = sqrt(normalization_value)
        for i in range(num_nodes):
            hub_scores[i] /= normalization_value
            print('{} :{:.2f}'.format(chr(65+i),hub_scores[i]),end=' | ')
        print("\n\n")

def main():
    n = int(input('Enter the no of nodes : '))
    graph = []
    print('Enter Adjacency Matrix : ')
    for i in range(n):
        temp = input()
        temp_list = temp.split(' ')
        graph.append(list(map(int,temp_list)))
    k = int(input('Enter No of Iterations to be performed : '))
    hits_algorithm(n, graph, k)

main()
```

Graph:



Output:

```
Enter the no of nodes : 3
Enter Adjacency Matrix :
0 0 1
0 0 1
0 0 0
Enter No of Iterations to be performed : 3
```

Iteration : 1
Authority Score
A :0.00 | B :0.00 | C :1.00 |
Hub Score
A :0.71 | B :0.71 | C :0.00 |

Iteration : 2
Authority Score
A :0.00 | B :0.00 | C :1.00 |
Hub Score
A :0.71 | B :0.71 | C :0.00 |

Iteration : 3
Authority Score
A :0.00 | B :0.00 | C :1.00 |
Hub Score
A :0.71 | B :0.71 | C :0.00 |

Conclusion:
Hyperlink-Induced Topic Search (HITS) is a link analysis algorithm that rates Web pages. HITS, like Page and Brin's PageRank, is an iterative algorithm based on the linkage of the documents on the web. However its disadvantages outweigh its advantages and thus it is not commonly used in search engines as compared to the PageRank algorithm which proves to be more efficient on large datasets.

EXPERIMENT 10

AIM: Write and Explain one algorithm each on

1.Spatial Association Rules
2.Spatial Classification
3.Spatial Clustering - DBScan

Theory:
Spatial data means data related to space which can be the two-dimensional abstraction of the surface of the earth or a man-made space like the layout of a VLSI design, a volume containing a model of the human brain, or another 3d-space representing the arrangement of chains of protein molecules. The data consists of geometric information and can be either discrete or continuous. The explicit location and extension of spatial objects define implicit relations of spatial neighborhood (such as topological, distance and direction relations) which are used by spatial data mining algorithms. Therefore, spatial data mining algorithms are required for spatial characterization and spatial trend analysis.
Spatial data mining or knowledge discovery in spatial databases differs from regular data mining in analogous with the differences between non-spatial data and spatial data. The attributes of a spatial object stored in a database may be affected by the attributes of the spatial neighbors of that object. In addition, spatial location, and implicit information about the location of an object, may be exactly the information that can be extracted through spatial data mining

---

1.Spatial Association Rules

Spatial association means connectedness or relationship between and among variables over space. A single variable may be spatially autocorrelated; that is, values of the variable are somehow connected or related spatially. Many variables may be associated one with another at one or more sites. If there is spatial interaction there is also spatial association. Maps can depict spatial association. A mathematical shorthand technique can be used to represent, in general, measures of spatial association. Scientists test or theorize about variables to determine whether spatial association, either observed or expected, actually can be confirmed. Statistical procedures that have been developed for identifying and measuring the existence of spatial association are outlined.

Algorithm : Apriori Algorithm
The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rules, it determines how strongly or how weakly two objects are connected. This

algorithm uses a breadth-first search and Hash Tree to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset. This algorithm is mainly used for market basket analysis and helps to find those products that can be bought together. It can also be used in the healthcare field to Frequent itemsets which are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset.

Steps for Apriori Algorithm

Below are the steps for the apriori algorithm:

Step-1: Determine the support of itemsets in the transactional database, and select the minimum support and confidence.

Step-2: Take all supports in the transaction with higher support value than the minimum or selected support value.

Step-3: Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.

Step-4: Sort the rules as the decreasing order of lift.

Advantages of Apriori Algorithm

- This is easy to understand algorithm
- The join and prune steps of the algorithm can be easily implemented on large datasets.

Disadvantages of Apriori Algorithm

- The apriori algorithm works slow compared to other algorithms.
- The overall performance can be reduced as it scans the database for multiple times.
- The time complexity and space complexity of the apriori algorithm is $O(2^D)$, which is very high. Here D represents the horizontal width present in the database.

2.Spatial Classification

Spatial classification assigns an object to a class from a given set of classes based on the attribute values of the object. It mainly considers the distance, direction, or connectivity relationships among spatial objects.

Algorithm: KNN Algorithm

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- The K-NN algorithm assumes the similarity between the new case/data and available cases and puts 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 suited category by using K- NN algorithm.
- The K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- The KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know whether it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Working:
The K-NN working can be explained on the basis of the below algorithm:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready.

Selecting the value of K in the K-NN Algorithm
Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

3.Spatial Clustering - DBScan

Spatial Clustering Clustering is a descriptive task that seeks to identify homogeneous groups of objects based on the values of their attributes. In spatial data sets, clustering permits a generalization of the spatial component like explicit location and extension of spatial objects which define implicit relations of spatial neighborhood. Current spatial clustering techniques can be broadly classified into three categories;

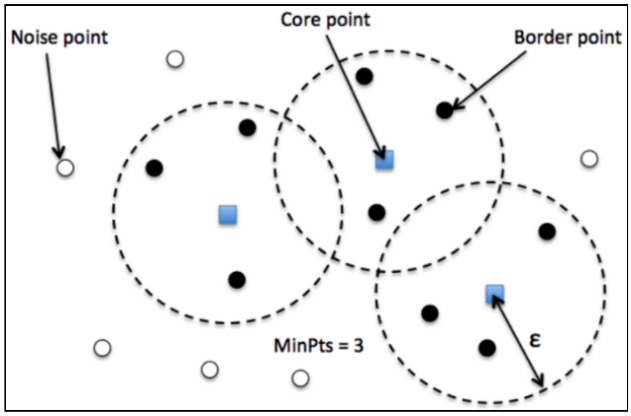
- partitional
- hierarchical
- locality-based.

Algorithm: DBSCAN Algorithm

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.
Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.
The DBSCAN algorithm uses two parameters:

- minPts: The minimum number of points (a threshold) clustered together for a region to be considered dense.
- eps (ε): A distance measure that will be used to locate the points in the neighborhood of any point.

These parameters can be understood if we explore two concepts called Density Reachability and Density Connectivity.
Reachability in terms of density establishes a point to be reachable from another if it lies within a particular distance (eps) from it.
Connectivity, on the other hand, involves a transitivity based chaining-approach to determine whether points are located in a particular cluster. For example, p and q points could be connected if p->r->s->t->q, where a->b means b is in the neighborhood of a.
There are three types of points after the DBSCAN clustering is complete:



- **Core** — This is a point that has at least m points within distance n from itself.
- **Border** — This is a point that has at least one Core point at a distance n.
- **Noise** — This is a point that is neither a Core nor a Border. And it has less than m points within distance n from itself.

Steps for DBSCAN clustering

- The algorithm proceeds by arbitrarily picking up a point in the dataset (until all points have been visited).
- If there are at least 'minPoint' points within a radius of 'ε' to the point then we consider all these points to be part of the same cluster.
- The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point

Parameter Estimation

Every data mining task has the problem of parameters. Every parameter influences the algorithm in specific ways. For DBSCAN, the parameters ε and minPts are needed.

- **minPts**: As a rule of thumb, a minimum minPts can be derived from the number of dimensions D in the data set, as minPts ≥ D + 1. The low value minPts = 1 does not make sense, as then every point on its own will already be a cluster. With minPts ≤ 2, the result will be the same as of hierarchical clustering with the single link metric, with the dendrogram cut at height ε. Therefore, minPts must be chosen at least 3. However, larger values are usually better for data sets with noise and will yield more significant clusters. As a rule of thumb, minPts = 2·dim can be used, but it may be necessary to choose larger values for very large data, for noisy data or for data that contains many duplicates.
- **ε**: The value for ε can then be chosen by using a k-distance graph, plotting the distance to the k = minPts-1 nearest neighbor ordered from the largest to the smallest value. Good values of ε are where this plot shows an "elbow": if ε is chosen much too small, a large part of the data will not be clustered; whereas for a too high value of ε, clusters will merge and the majority of objects will be in the same cluster.

In general, small values of ε are preferable, and as a rule of thumb, only a small fraction of points should be within this distance of each other.

- **Distance function**: The choice of distance function is tightly linked to the choice of ε, and has a major impact on the outcomes. In general, it will be necessary to first identify a reasonable measure of similarity for the data set, before the parameter ε can be chosen. There is no estimation for this parameter, but the distance functions need to be chosen appropriately for the data set.

CONCLUSION:

We learnt about spatial data, spatial association rules, classification and clustering. We also studied an algorithm of each type and learnt about the working, advantages and the disadvantages of each one of them.