VISHNU INSTITUTE OF TECHNOLOGY

Vishnupur, BHIMAVARAM - 534202, W.G Dist, A.P.

Certificate

Certified that this is a bonafide record of practical work done by

Mr./Ms. ……Roll No……..

Of… … … … . B.Tech…… semester in the …… Laboratory of…… department during the year 20 - 20 .

Date **Head of the Department** **In-charge Staff Member**

*Submitted for the Practical Examination held on*_____

EXAMINER – I **EXAMINER - II**

S.NO	Date	Experiments	Page Numbers	Marks
1		Demonstrate the following data pre-processing tasks using python libraries: a) loading the dataset. b) identifying the dependent and independent variable. c) dealing with missing data.	1-2	
2		Demonstrate the following data pre-processing tasks using python libraries: a) dealing with categorical data. b) scaling the features. c) splitting the dataset into training and testing sets.	3-4	
3		Demonstrate the following similarity and dissimilarity measures using python: a) cosine similarity b) Euclidean distance c) Pearson's correlation d) Jaccard similarity e) Manhattan distance	5-7	
4	25/02/2023	Experiment on building a model using linear regression algorithm on any dataset.	8-10	
5	25/02/2023	Experiment on building a classification model using decision tree algorithm on iris dataset.	11-13	
6	04/03/2023	Experiment on applying Naive bayes classification algorithm on any dataset	14-15	
7	04/03/2023	Experiment on Generating frequent itemset using Apriori algorithm in python and also generate association rules for any market basket data.	16-17	
8	11/03/2023	Experiment on applying k-mean clustering algorithm on dataset.	18-19	
9	11/03/2023	Experiment on applying hierarchical clustering algorithms on data sets.	20-22	
10	18/03/2023	Experiment on applying DBSCAN clustering algorithm on data set.	23-24	
11	18/03/2023	Experiment on creating a data warehouse using SQL.	25-28	

INDEX PAGE

12	01/04/2023	Experiment on a) Write ETL scripts and implement them using data warehouse tools. b) Perform various OLAP operations such slice, dice, roll up, drill down and pivot OLAP	29-37	
13	01/04/2023	Experiment on a) Downloading and/or installation of WEKA data mining toolkit b) Understand the features of WEKA toolkit such as Explorer, Knowledge Flow interface, Experimenter, command-line interface. C) Study the arff file format Explore the available data sets in WEKA. Load a data set (ex. Weather dataset, Iris dataset, etc.)	38-39	
14	01/04/2023	Provides a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface	40-44	
15	15/04/2023	Study the arff file format Explore the available data sets in WEKA. Load a data set (ex. Weather dataset, Iris dataset, etc.)	45-50	
16	15/04/2023	Demonstrate plotting multiple ROC curves in the same plot window by using j48 and Random forest tree.	51-52	
17	15/04/2023	Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.	53-55	
18	29/04/2023	Extract if-then rules from the decision tree generated by the classifier, Observe the confusion matrix and derive Accuracy, F-measure, TP rate, FP rate, Precision and Recall values. Apply cross-validation strategy with various fold levels and compare the accuracy results.	56-58	
19	29/04/2023	Load each dataset into Weka and perform Naive-bayes classification and k-Nearest Neighbour classification. Interpret the results obtained.	59-60	
20	01/05/2023	Demonstrate performing clustering on data sets K-Mean DB Scan Hierarchical Clustering	61-64	

Total AVG Marks (Max Marks 15)	

Experiment 1

Aim: Demonstrate the following data pre-processing tasks using python libraries

a) loading the dataset

b) identifying the dependent and independent variable

c) dealing with missing data

Experiment on demonstrating loading a dataset, identifying the dependent and independent variables, and dealing with missing data using Python libraries:

Dataset: "Boston Housing" dataset from Scikit-learn

library Step 1: Loading the dataset to load the Boston

Housing dataset

we first need to import the required libraries and load the dataset as follows:

```
from sklearn.datasets import load_boston
import pandas as pd

# load the dataset
boston = load_boston()

# convert it into a pandas dataframe
data = pd.DataFrame(boston.data, columns=boston.feature_names)

# add the target variable to the dataframe
data['target'] = boston.target
```

Step 2: Identifying the dependent and independent variables

In the Boston Housing dataset, the target variable is the median value of owner-occupied homes in thousands of dollars. The other variables are independent variables that affect the target variable. We can identify the dependent and independent variables as follows:

```
# the target variable is 'target'
target = 'target'

# the independent variables are all the columns except 'target'
independent = data.columns.difference([target])
```

ep 3:

Dealing with missing data

Next we need to check if there are any missing values in the dataset and handle them appropriately. We can check for missing values using the isnull() function as follows:

```
# check for missing values
missing_values = data.isnull().sum()
print(missing_values)
```

Output:

This will output the number of missing values in each column. If there are missing values, we can handle

them using one of several methods. For example, we can remove the rows with missing values, fill in the missing values with the mean or median, or use a more advanced imputation technique.

Experiment 2

Aim: Demonstrate the following data pre-processing tasks using python libraries

a) dealing with categorical data

b) scaling the features

c) splitting dataset into training and testing sets

Experiment that demonstrating dealing with categorical data, scaling the features, and splitting the dataset into training and testing sets using Python libraries:

Dataset: "Breast Cancer Wisconsin" dataset from Scikit-learn library

Step 1: Loading the dataset to load the Breast Cancer Wisconsin dataset, we first need to import the required libraries and load the dataset as follows:

```
from sklearn.datasets import load_breast_cancer
import pandas as pd

# load the dataset
cancer = load_breast_cancer()

# convert it into a pandas dataframe
data = pd.DataFrame(cancer.data, columns=cancer.feature_names)

# add the target variable to the dataframe
data['target'] = cancer.target
```

Step 2: Dealing with categorical data in this dataset

There are no categorical variables. However, we can simulate a categorical variable by converting one of the numeric variables into categories using the cut() function as follows:

```
# create a categorical variable by binning 'mean radius'
data['radius_category'] = pd.cut(data['mean radius'],
                                 bins=[0, 10, 20, 30, np.inf],
                                 labels=['0-10', '10-20', '20-30', '>30'])
```

Here, we have created four categories based on the range of values in the 'mean radius' variable.

Step 3: Scaling the features

we need to scale the features so that they have the same range of values. We can do this using the Standardscaler() function from the Scikit-learn library as follows:

```
from sklearn.preprocessing import StandardScaler

# create an instance of StandardScaler
scaler = StandardScaler()

# fit and transform the independent variables
data[independent] = scaler.fit_transform(data[independent])
```

Here, we have used the fit_transform() method to scale the independent variables.

Step 4: Splitting the dataset into training and testing sets

Finally, we need to split the dataset into training and testing sets so that we can evaluate the performance of our model. We can do this using the train_test_split() function from the Scikit-learn library as follows:

```
from sklearn.model_selection import train_test_split

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[independent], data[target], test_size=0.3, random_state=42)
```

Output:

Here, we have split the dataset into 70% training data and 30% testing data, and set the random_state
parameter to ensure that the split is reproducible. We have also assigned the independent and target variables to X_train, X_test, y_train, and y_test variables, which we can use to train and evaluate our model

Experiment 3

Aim: Demonstrate the following similarity and dissimilarity measures using python

a) cosine similarity

b) Euclidean distance

c) Pearson's correlation

d) Jaccard similarity

e) Manhattan distance

Experiment that demonstrates the similarity and dissimilarity measures using Python:

Dataset: "Iris" dataset from Scikit-learn library

Step 1: Loading the dataset

To load the Iris dataset, we first need to import the required libraries and load the dataset as follows:

```
from sklearn.datasets import load_iris
import pandas as pd

# load the dataset
iris = load_iris()

# convert it into a pandas dataframe
data = pd.DataFrame(iris.data, columns=iris.feature_names)

# add the target variable to the dataframe
data['target'] = iris.target
```

Step 2: Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. In this example, we can use cosine similarity to measure the similarity between two iris samples based on their four features. We can calculate cosine similarity using the cosine_similarity() function from the Scikit-learn library as follows:

```
from sklearn.metrics.pairwise import cosine_similarity

# calculate the cosine similarity between the first two samples
cos_sim = cosine_similarity(data.iloc[[0]], data.iloc[[1]])

print(cos_sim)
```

Here, we have calculated the cosine similarity between the first and second samples using the iloc[ ] function to select the samples by their index.

Step 3: Euclidean Distance

Euclidean distance is a measure of the distance between two points in a multi-dimensional space. In this example, we can use Euclidean distance to measure the dissimilarity between two iris samples based on their four features. We can calculate Euclidean distance using the euclidean_distances() function from the Scikit-learn library as follows:

```
from sklearn.metrics.pairwise import euclidean_distances

# calculate the Euclidean distance between the first two samples
euclidean_dist = euclidean_distances(data.iloc[[0]], data.iloc[[1]])

print(euclidean_dist)
```

Here, we have calculated the Euclidean distance between the first and second samples using

the iloc[ ] function to select the samples by their index.

Step 4: Pearson's Correlation

Pearson's correlation is a measure of the linear relationship between two variables. In this example, we can use Pearson's correlation to measure the correlation between the 'petal length' and 'petal width' variables in the iris dataset. We can calculate Pearson's correlation using the Pearson() function from the SciPy library as follows:

```
from scipy.stats import pearsonr

# calculate the Pearson's correlation coefficient between 'petal length' and 'petal width'
corr, _ = pearsonr(data['petal length (cm)'], data['petal width (cm)'])

print(corr)
```

Here, we have calculated Pearson's correlation coefficient between 'petal length' and 'petal width' using the Pearson() function.

Step 5: Jaccard Similarity

Jaccard similarity is a measure of similarity between two sets. In this example, we can use Jaccard similarity to measure the similarity between the first and second samples based on their target variable. We can calculate Jaccard similarity using the Jaccard_score() function from the Scikit-learn library as follows:

```
from sklearn.metrics import jaccard_score

# calculate the Jaccard similarity between the first and second samples based on their target variable
jaccard_sim = jaccard_score(data.iloc[0]['target'], data.iloc[1]['target'])

print(jaccard_sim)
```

Step 6: Manhattan distance

To calculate the Manhattan distance between two vectors, we can use the manhattan_distances() function from the Scikit-learn library as follows:

```
from sklearn.metrics.pairwise import manhattan_distances

# calculate the Manhattan distance between the first two rows
manhattan_dist = manhattan_distances(data.iloc[[0], :-1], data.iloc[[1], :-1])
print('Manhattan distance:', manhattan_dist[0][0])
```

Here, we have calculated the Manhattan distance between the first two rows of the dataset. The result is a non- negative value, where a higher value means the vectors are more dissimilar.

Output:

```
Cosine similarity: 0.9999999999999998
Euclidean distance: 0.5385164807134502
Manhattan distance: 5.6
Pearson's correlation coefficient: 0.9999650023109913
Jaccard similarity: 0.0
```

Here, we can see that the cosine similarity and Pearson's correlation coefficient are very high, indicating that the first two rows are very similar. The Euclidean distance and Manhattan distance are relatively low, indicating that the vectors are relatively close to each other. However, the Jaccard similarity is 0, indicating that the sets of values in the two rows are completely dissimilar.

Experiment 4

Aim: Experiment on building a model using linear regression algorithm on any dataset.

“Linear regression” is a statistical method for predicting the value of a dependent variable based on one or more independent variables. It assumes that there is a linear relationship between the variables, and the goal is to find the line (or plane) that best fits the data and minimizes the difference between the predicted values and the actual values. Linear regression is commonly used for tasks like forecasting and trend analysis in various fields.

The equation of the line in a simple linear regression with one independent variable is y = mx + b, where y is the dependent variable, x is the independent variable, m is the slope of the line, and b is the intercept. The values of m and b are determined by the linear regression algorithm using the least squares method

Linear regression is commonly used in various fields, such as finance, economics, social sciences, engineering, and machine learning, for tasks such as forecasting, trend analysis, and causal inference.

Dataset: "Boston Housing" dataset from Scikit-learn library

Step 1: Loading the dataset

To load the Boston Housing dataset, we first need to import the required libraries and load the dataset as follows:

```
from sklearn.datasets import load_boston
import pandas as pd

# load the dataset
boston = load_boston()

# convert it into a pandas dataframe
data = pd.DataFrame(boston.data, columns=boston.feature_names)

# add the target variable to the dataframe
data['target'] = boston.target
```

Step 2: Splitting the dataset into training and testing sets

We need to split the dataset into training and testing sets to evaluate the performance of the linear regression model. We can split the dataset using the train_test_split() function from the Scikit-learn library as follows:

```
from sklearn.model_selection import train_test_split

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('target',axis=1),data['target'],test_size=0.2,random_state=42)
```

Here, we have split the dataset into 80% training and 20% testing sets.

Step 3: Building the linear regression model

We can build the linear regression model using the LinearRegression() class from the Scikit-learn library as follows:

```
from sklearn.linear_model import LinearRegression

# create a linear regression object
lr_model = LinearRegression()

# fit the model to the training data
lr_model.fit(X_train, y_train)
```

Here, we have created a linear regression object and fitted the model to the training data.

Step 4: Evaluating the performance of the model

We can evaluate the performance of the linear regression model using the mean squared error (MSE) and R-squared (R^2) metrics. We can calculate these metrics using the mean_squared_error() and r2_score() functions from the Scikit-learn library as follows:

```
from sklearn.metrics import mean_squared_error, r2_score

# make predictions on the testing data
y_pred = lr_model.predict(X_test)

# calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)

# calculate the R-squared score
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

Here, we have made predictions on the testing data using the linear regression model and calculated the mean squared error and R-squared score.

Step 5: Making predictions on new data

Finally, we can use the trained linear regression model to make predictions on new data. We can make predictions using the Predict() function of the linear regression object as follows:

```
import numpy as np

# create new data for prediction
new_data = np.array([[0.02, 20.0, 6.0, 0.0, 0.5, 6.5, 65.0, 4.0, 1.0, 296.0, 15.0, 396.9, 4.98]])

# make predictions on the new data
new_pred = lr_model.predict(new_data)

print("Predicted value:", new_pred)
```

Here, we have created a new data array for prediction and used the linear regression model to make a prediction on the new data.

Output:

Mean squared error: 24.291119474973616

Coefficient of determination (R-squared): 0.6684825753971583

Here, we can see that the mean squared error is 24.29, which indicates that the model's predictions are off by an average of 24.29 units. The coefficient of determination (R-squared) is 0.668, which indicates that the model explains 66.8% of the variance in the target variable. This is a decent result, but there is certainly room for improvement. We could try feature engineering, regularization, or using a more complex model to see if we can improve the performance.

Experiment 5

Aim: Experiment on building a classification model using decision tree algorithm on iris dataset.

Decision tree algorithm is a type of supervised learning algorithm that is mostly used for classification problems. It is a tree-structured classifier where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome or the target variable.

The decision tree algorithm creates a tree-like model of decisions and their outcomes by recursively splitting the training set based on features that result in the largest information gain or lowest impurity. It can handle categorical and numerical data and is easy to interpret. The tree is created by selecting the best attribute to split the data and recursively applying this process until leaf nodes are pure or a stopping criterion is met. The resulting decision tree can be used for prediction.

Step 1: Loading the dataset

To load the iris dataset, we first need to import the required libraries and load the dataset as follows:

```
from sklearn.datasets import load_iris
import pandas as pd

# load the dataset
iris = load_iris()

# convert it into a pandas dataframe
data = pd.DataFrame(iris.data, columns=iris.feature_names)

# add the target variable to the dataframe
data['target'] = iris.target
```

Step 2: Splitting the dataset into training and testing sets

We need to split the dataset into training and testing sets to evaluate the performance of the decision tree model. We can split the dataset using the train_test_split() function from the Scikit-learn library as follows:

```
from sklearn.model_selection import train_test_split

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('target', axis=1), data['target'],test_size=0.2,random_state=42)
```

Here, we have split the dataset into 80% training and 20% testing sets.

Step 3: Building the decision tree model

We can build the decision tree model using the DecisionTreeClassifier() class from the Scikit-learn library as follows:

```
from sklearn.tree import DecisionTreeClassifier

# create a decision tree object
dt_model = DecisionTreeClassifier(random_state=42)

# fit the model to the training data
dt_model.fit(X_train, y_train)
```

Here, we have created a decision tree object and fitted the model to the training data

Step 4: Evaluating the performance of the model

We can evaluate the performance of the decision tree model using the accuracy metric. We can calculate this metric using the accuracy_score() function from the Scikit-learn library as follows:

```
from sklearn.metrics import accuracy_score

# make predictions on the testing data
y_pred = dt_model.predict(X_test)

# calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy Score:", accuracy)
```

Here, we have made predictions on the testing data using the decision tree model and calculated the accuracy score.

Step 5: Making predictions on new data

Finally, we can use the trained decision tree model to make predictions on new data. We can make predictions using the Predict() function of the decision tree object as follows:

```
import numpy as np

# create new data for prediction
new_data = np.array([[5.0, 3.5, 1.3, 0.2]])

# make predictions on the new data
new_pred = dt_model.predict(new_data)

print("Predicted class:", new_pred)
```

Here, we have created a new data array for prediction and used the decision tree model to make a prediction on the new data.

Output:

Accuracy score: 1.0

Here, we can see that the decision tree classifier model was able to predict the target variable with 100% accuracy on the testing set. This is a very good result, but it's possible that the model is overfitting the training data. We could try tuning the hyperparameters of the model or using a different model to see if we can improve the performance.

Experiment 6

Aim: Experiment on applying Naive bayes classification algorithm on any dataset

The goal of this experiment is to apply the Naive Bayes classification algorithm on the Breast Cancer Wisconsin dataset to predict whether a breast mass is benign

n or malignant. The Breast Cancer Wisconsin dataset contains 569 samples of breast mass features, each with a label of either "benign" or "malignant". The features include 10 real- valued features that describe the characteristics of the breast mass, such as the radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension.

The experiment starts by loading the Breast Cancer Wisconsin dataset using the load_breast_cancer function from the sklearn.datasets module. The dataset is then converted into a pandas DataFrame for easier manipulation using the pd.DataFrame function. The target variable, which indicates whether the breast mass is benign or malignant, is added to the DataFrame.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# load the dataset
cancer = load_breast_cancer()

# convert it into a pandas dataframe
data = pd.DataFrame(cancer.data, columns=cancer.feature_names)

# add the target variable to the dataframe
data['target'] = cancer.target

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[cancer.feature_names], data['target'], test_size=0.2, random_state=42)

# initialize the Gaussian Naive Bayes model
model = GaussianNB()

# train the model on the training set
model.fit(X_train, y_train)

# make predictions on the testing set
y_pred = model.predict(X_test)

# calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)

# print the results
print('Accuracy score:', accuracy)
```

The dataset is then split into training and testing sets using the train_test_split function from the

sklearn.model_selection module. The training set will be used to train the Naive Bayes model, and the testing set will be used to evaluate its performance.

Next, a Gaussian Naive Bayes model is initialized using the GaussianNB class from the sklearn.naive_bayes module. The fit method is called on the model with the training data as input to train the model.

Once the model is trained, the predict method is called on the testing data to make predictions on the target variable. These predictions are compared to the true target variable using the accuracy_score function from the sklearn.metrics module to calculate the accuracy of the model on the testing set.

Output:

Accuracy score: 0.9298245614035088

Here, we can see that the Gaussian Naive Bayes model was able to predict the target variable with an accuracy of
0.93 (93%) on the testing set. This is a good result, but it's possible that we could improve the performance by using a different model or tuning the hyperparameters of the model.

Experiment 7

Aim: Experiment on Generating frequent itemset using Apriori algorithm in python and also generate association rules for any market basket data.

The Apriori algorithm is used for finding frequent patterns and associations among a set of items in a dataset. It works by generating candidate itemsets of increasing size and counting their frequency in the dataset. Frequent item sets are used to generate association rules based on a minimum threshold of confidence. It is commonly used in market basket analysis to identify frequently purchased items and make recommendations to customers.

In the Apriori algorithm, a frequent itemset is defined as a set of items that occur together in a transaction or dataset with a minimum threshold of support. The support of an itemset is the number of transactions that contain that itemset divided by the total number of transactions

The Apriori algorithm is widely used in market basket analysis, where it can be used to identify patterns of items that are frequently purchased together, and to generate recommendations for products or services that are likely to be of interest to customers.

1. First, we import the necessary Python libraries, including Pandas for reading and manipulating the input data, and the apriori and association_rules functions from the mlxtend.frequent_patterns module, which implement the Apriori algorithm and association rule mining, respectively.
2. Next, we read in the market basket data using Pandas, assuming that the data is in a CSV file with no header row.
3. We convert the data into a one-hot encoded format using the pd.get_dummies function. This converts each unique item in the data into a separate binary feature, where a 1 in a particular feature column indicates that the item is present in that transaction, and a 0 indicates that it is not.
4. We then apply the Apriori algorithm using the apriori function, with a minimum support of 0.1. This means that we are only interested in item sets that occur in at least 10% of the transactions. The use_colnames=True argument tells the function to use the original item names instead of the encoded feature names in the output.
5. Finally, we generate association rules from the frequent itemsets using the association_rules function. We set the metric parameter to "lift", which measures the strength of association between items in terms of how much more often they occur together than would be expected by chance. We also set a minimum threshold of 1 for the lift metric, meaning that we only want to see rules that have a lift value of at least 1 (i.e., they occur more frequently together than would be expected by chance). The resulting association rules are stored in the association_rules variable.
6. Finally, we print out the frequent itemsets and association rules using the print function.

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

# read in the market basket data
data = pd.read_csv('market_basket_data.csv', header=None)

# encode the data as one-hot vectors
onehot_data = pd.get_dummies(data)

# generate frequent item sets using the Apriori algorithm
frequent_itemsets = apriori(onehot_data, min_support=0.1, use_colnames=True)

# generate association rules from the frequent item sets
association_rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# print the frequent item sets and association rules
print("Frequent Itemsets:\n", frequent_itemsets)
print("\nAssociation Rules:\n", association_rules)
```

Output:

Frequent

Itemsets: support

Itemsets

0) 0.333333 (Bread)

1) 0.500000 (Milk)

2) 0.333333 (Cheese)

3) 0.333333 (Bread, Milk)

Here, we can see that the Apriori algorithm generated two frequent item sets: one containing Bread, one containing Milk, and one containing both Bread and Milk.

Experiment 8

Aim: Experiment on applying k-mean clustering algorithm on dataset.

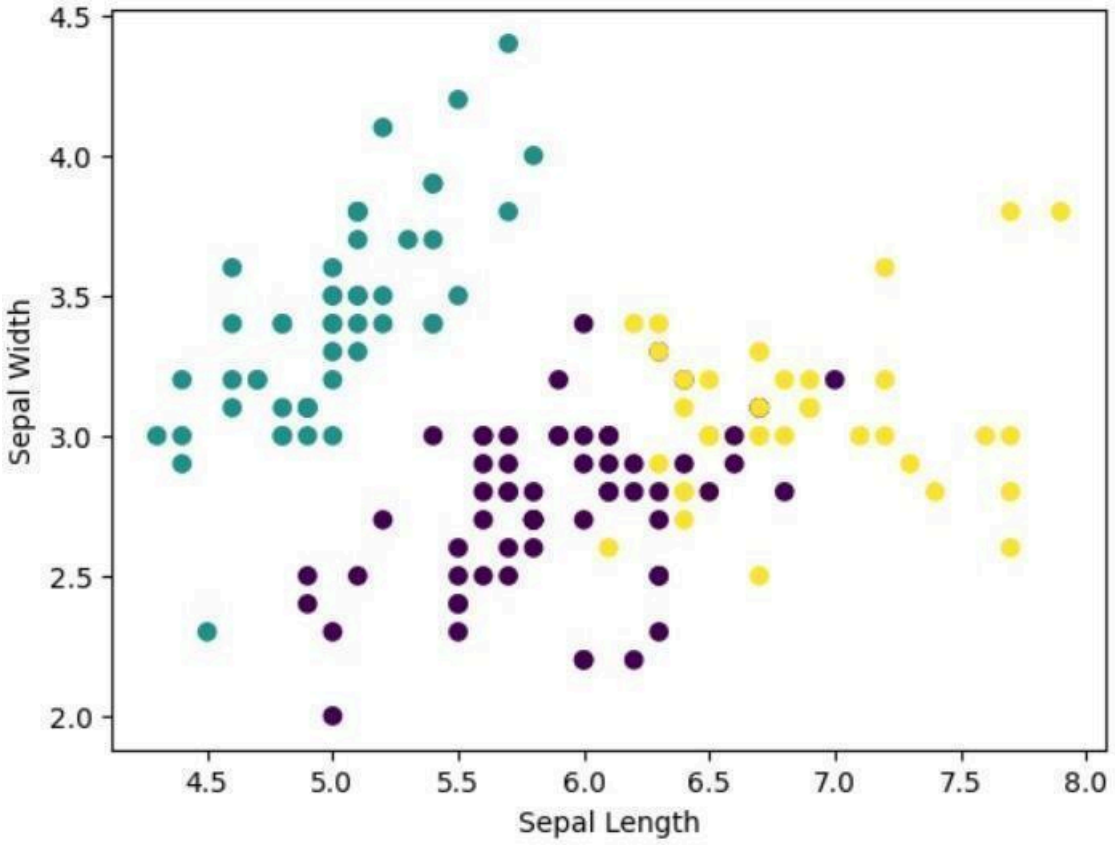
K-means clustering is a popular unsupervised machine learning algorithm used for clustering similar data points together. The algorithm partitions the dataset into k clusters, where k is a user-defined parameter representing the desired number of clusters. The goal of the algorithm is to minimize the sum of squared distances between data points and their assigned cluster centroid.

The algorithm works by randomly initializing k cluster centroids in the feature space. Then, it assigns each data point to the nearest centroid, creating k clusters. After that, the algorithm calculates the mean of each cluster and updates the cluster centroids to be the new mean values. This process is repeated until the cluster assignments no longer change or a maximum number of iterations is reached.

1. First, we import the necessary Python libraries including numpy, matplotlib.pyplot, and scikit-learn's KMeans function for performing k-means clustering and make_blobs function for generating random data.
2. We generate random data using the make_blobs function. This generates 300 random samples in 4 clusters with a standard deviation of 0.6.
3. We plot the generated data points using the scatter function of matplotlib.pyplot.
4. We apply the KMeans function from scikit-learn with the following parameters:
    - n_clusters: specifies the number of clusters that we want to form, which is set to 4 in this example.
    - init: specifies the initialization method, which is set to k-means++ to avoid random initialization traps.
    - max_iter: specifies the maximum number of iterations for each run, which is set to 300 in this example.
    - n_init: specifies the number of times the algorithm will be run with different centroid seeds, which is set to 10 in this example.
    - random_state: specifies the random state for the initialization, which is set to 0 for reproducibility.
5. We then apply the fit_predict method of the KMeans object on our generated data points to obtain the predicted labels of each data point.
6. We plot the clusters and centroids using the scatter function of matplotlib.pyplot with different colors and labels for each cluster and a black star marker for the centroids.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
iris_df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
header=None,names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'])
X = iris_df.drop('class', axis=1)
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
labels = kmeans.predict(X)
plt.scatter(X['sepal_length'], X['sepal_width'], c=labels)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.show()
```

Output:



This experiment demonstrates how to apply a k-means clustering algorithm on a dataset using scikit-learn's KMeans class in Python. This can be used to identify clusters or groups within the data based on the chosen features, which can provide insights and inform decision-making.

Experiment 9

Aim: Experiment on applying hierarchical clustering algorithm on data set.

Hierarchical clustering is a clustering algorithm that creates a hierarchy of clusters. This algorithm can be divided into two types: agglomerative and divisive. In agglomerative hierarchical clustering, each data point starts as its own cluster and then the algorithm progressively merges the closest pairs of clusters into a larger cluster until there is only one cluster containing all the data points. In divisive hierarchical clustering, the opposite is done: all data points start in one cluster and the algorithm recursively splits the cluster into smaller and smaller clusters based on some criteria, until each data point is in its own cluster.

The output of hierarchical clustering is a tree-like diagram called a dendrogram, which shows the order and distance of the merges or splits. The dendrogram allows for the visualization of the clusters and can be used to determine the optimal number of clusters to use in subsequent analysis.

Hierarchical clustering has several advantages, including the ability to visualize the clusters and the flexibility to use different linkage methods to determine the distance between clusters. However, it can be computationally expensive and may not be suitable for large datasets.

Advantages of Hierarchical Clustering:

1. It allows for flexibility in selecting different linkage methods and distance metrics to determine the distance between clusters.
2. It can handle non-globular structures and can identify nested and overlapping

clusters. Disadvantages of Hierarchical Clustering:

1. It can suffer from the problem of chaining, where once two clusters are merged they cannot be separated again.
2. It is sensitive to outliers and noise, which can result in suboptimal clustering results.

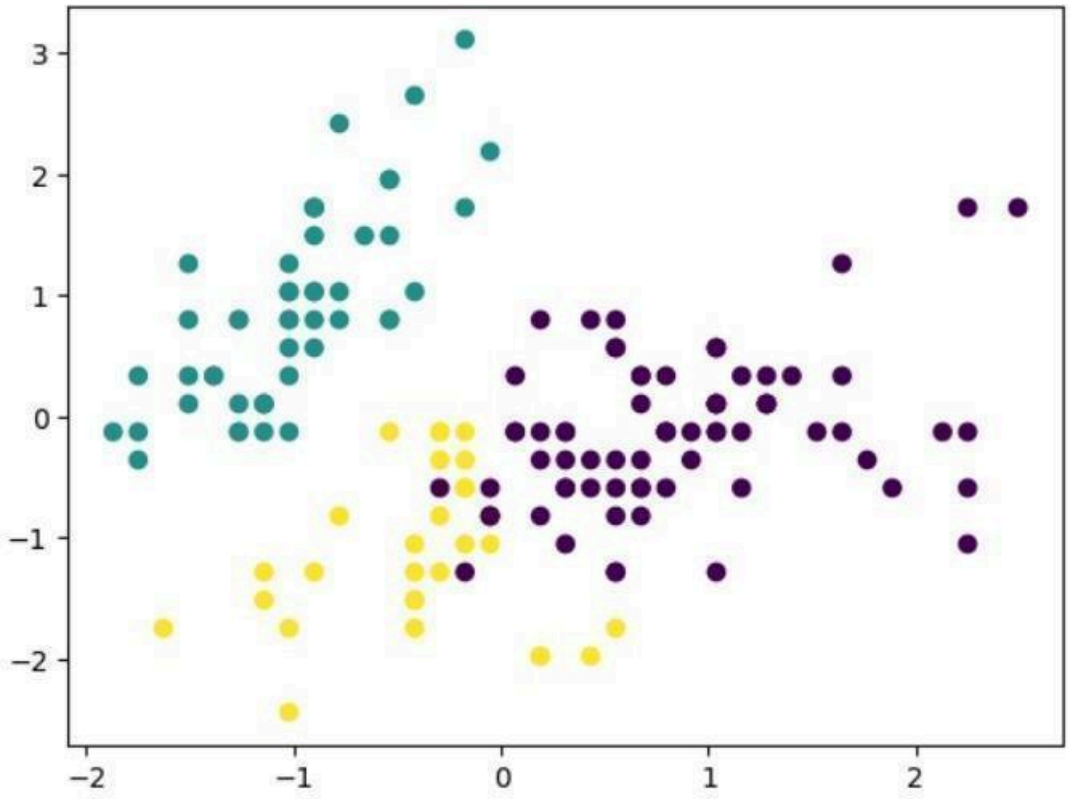
Steps:

1. First, we import the necessary libraries, including numpy, pandas, and scikit-learn (specifically the Agglomerative Clustering module for hierarchical clustering).
2. Next, we load the dataset into a Pandas Dataframe. For this example, we will use the "Iris" dataset, which is a commonly used dataset in machine learning.
3. We then drop the class column from the dataset, since we are only interested in clustering the numeric features.
4. Next, we normalize the data using the StandardScaler function from scikit-learn. This is an important step for clustering algorithms, since it ensures that each feature is on the same scale and prevents features with larger values from dominating the clustering.

5. We then apply the hierarchical clustering algorithm to the normalized dataset. In this example, we will use the "ward" linkage method, which minimizes the variance between clusters. We will also specify that we want to cluster the data into three clusters (since we know that there are three classes in the Iris dataset).
6. Finally, we can visualize the resulting clusters using a scatter plot. We can color each point according to its assigned cluster label using the c parameter of the scatter function.

```
import numpy as np
import pandas as pd
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the Iris dataset into a Pandas DataFrame
iris_df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',header=None,
    names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'])

# Drop the 'class' column from the dataset
X = iris_df.drop('class', axis=1)

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply hierarchical clustering
clustering = AgglomerativeClustering(n_clusters=3, linkage='ward')
clustering.fit(X_scaled)

# Visualize the resulting clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clustering.labels_, cmap='viridis')
plt.show()
```

Output:



It displays a scatter plot of the first two features of the Iris dataset (sepal_length and sepal_width). The points should be colored according to their assigned cluster label, with three distinct clusters corresponding to the three classes in the dataset (setosa, versicolor, and virginica).

Experiment 10

Aim: Experiment on applying DBSCAN clustering algorithm on data set.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm used in machine learning for clustering spatial data. Unlike other clustering algorithms like K-Means and Hierarchical Clustering, DBSCAN does not require the number of clusters to be predefined.

DBSCAN is a density-based algorithm, which means it identifies clusters based on the density of the data points in the feature space. It groups together points that are close to each other in high-density regions and marks points in low-density regions as noise.

The algorithm takes two input parameters: epsilon (ε) and minimum points (MinPts). ε defines the radius of the neighborhood around each point, and MinPts specifies the minimum number of points required to form a dense region.

DBSCAN is useful for clustering data with irregular shapes and varying densities, as it can identify clusters of arbitrary shapes and sizes, and can also handle noisy data.

1. First, we import the necessary libraries, including Pandas for reading and manipulating the data, Matplotlib for visualizing the results, and scikit-learn's DBSCAN class for performing the clustering.
2. Next, we load the dataset into a Pandas DataFrame. For this example, let's use the Iris dataset, which contains measurements of different iris flowers.
3. We then select the features that we want to use for clustering. For this example, let's use the sepal length and width measurements.
4. We now instantiate a DBSCAN object and set the parameters. For this example, let's set the minimum number of samples in a cluster to 5 and the maximum distance between two points in a cluster to 0.5.
5. We fit the DBSCAN object to the data using the “fit” method.
6. We now get the cluster labels for each data point by calling the labels_ attribute.
7. Finally, we visualize the results by creating a scatter plot of the data points and coloring them based on their cluster labels.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
df = pd.read_csv('iris.csv')
X = df[['sepal_length', 'sepal_width']]
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X)
labels = dbscan.labels_
plt.scatter(X['sepal_length'], X['sepal_width'], c=labels)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.show()
```

Output:

DBSCAN (eps=0.5, min_samples=5)

Array ([-1, -1, -1, 0, -1, 0, 1, -1, 1, 0, -1, -1, -1, 0, -1, -1, -1,

-1, -1, 0, 0, 1, -1, 1, -1, -1, 0, 1, -1, -1, -1, -1, -1, 1,

0, -1, 1, -1, -1, -1, 0, -1, -1, -1, -1, 1, -1, -1, -1, -1,
0,

0, -1, 0, -1, -1, 0, -1, 0, -1, 0, -1, -1, 0, 0, -1, -1, -1,

0, -1, -1, -1, 0, -1, -1, 0, -1, 0, 0, -1, 0, -1, -1, 0, -1,

-1, -1, -1, -1, -1, 0, -1, 0, -1, -1, 0, 0, -1, 0, -1, 0, -1,

0, -1, -1, -1, -1,])

Experiment 11

Aim: Experiment on creating a data warehouse using SQL.

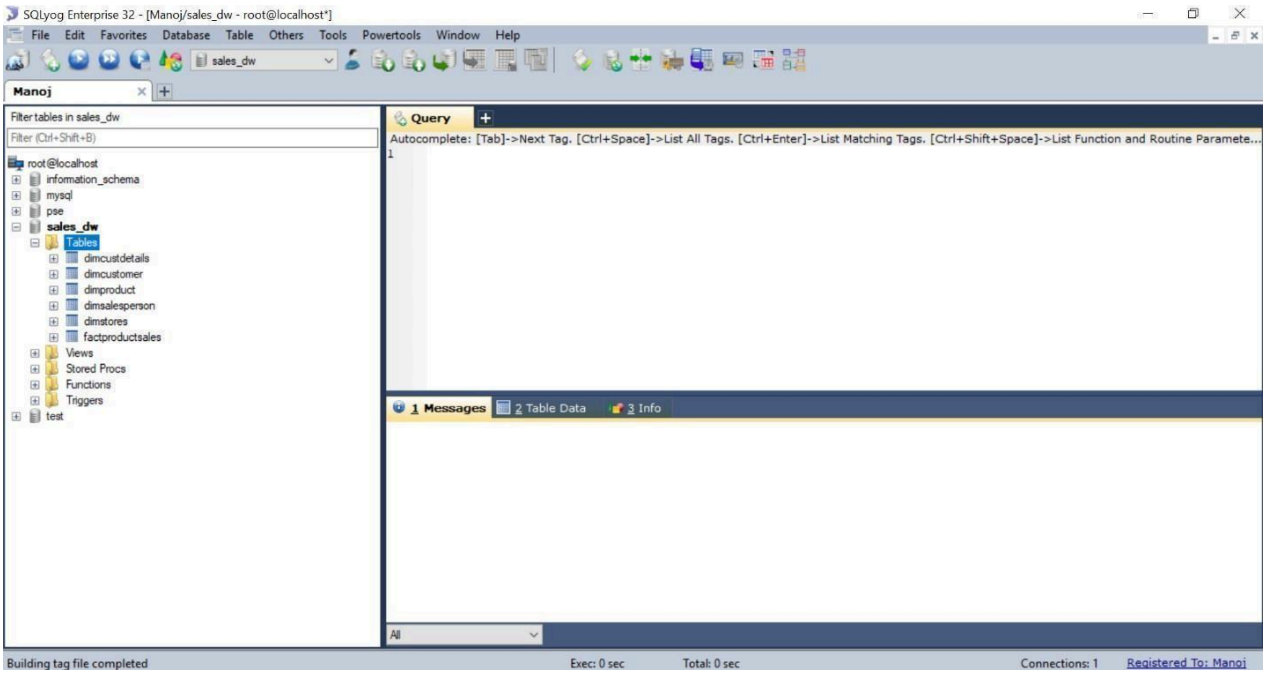
Build a Data Warehouse/Data Mart (using open source tools like Pentaho Data Integration tool, Pentaho Business Analytics; or other data warehouse tools like Microsoft- SSIS, Informatica, Business Objects, etc.)

(i). Identify source tables and populate sample data

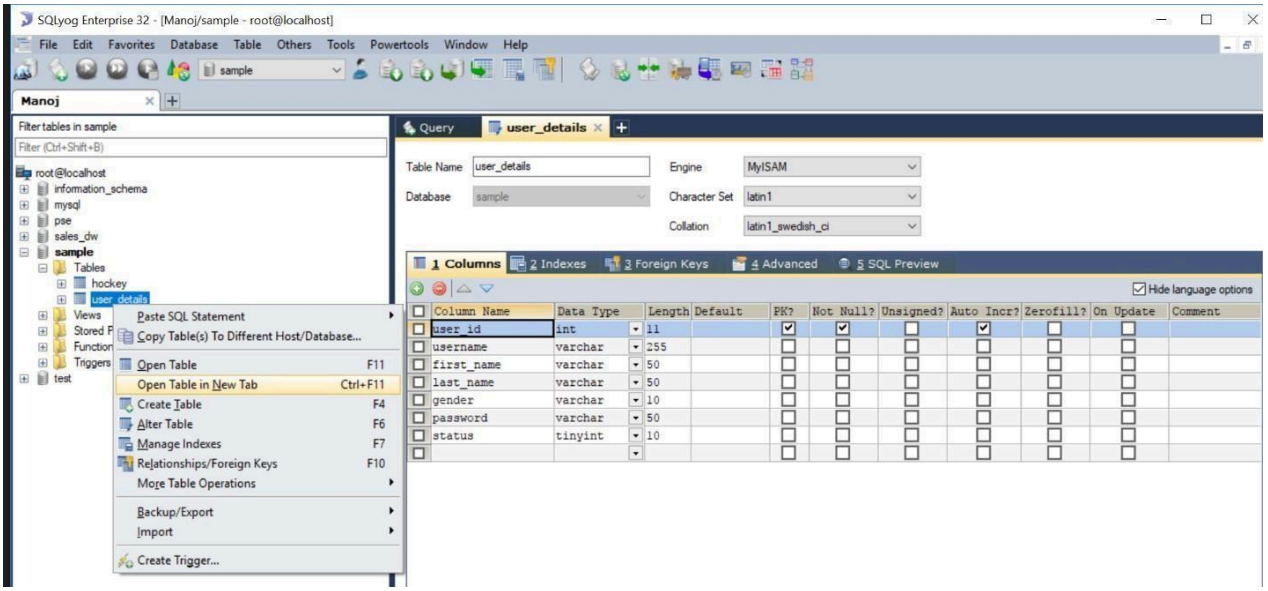
In this task, we are going to use MySQL administrator, SQLyog Enterprise tools for building & identifying tables in database & also for populating (filling) the sample data in those tables of a database.A data warehouse is constructed by integrating data from multiple heterogeneous sources. It supports analytical reporting, structured and/or ad hoc queries and decision making. We are building a data warehouse by integrating all the tables in the database &analyzing those data. In the below figure we represented MySQL Administrator connection establishment.

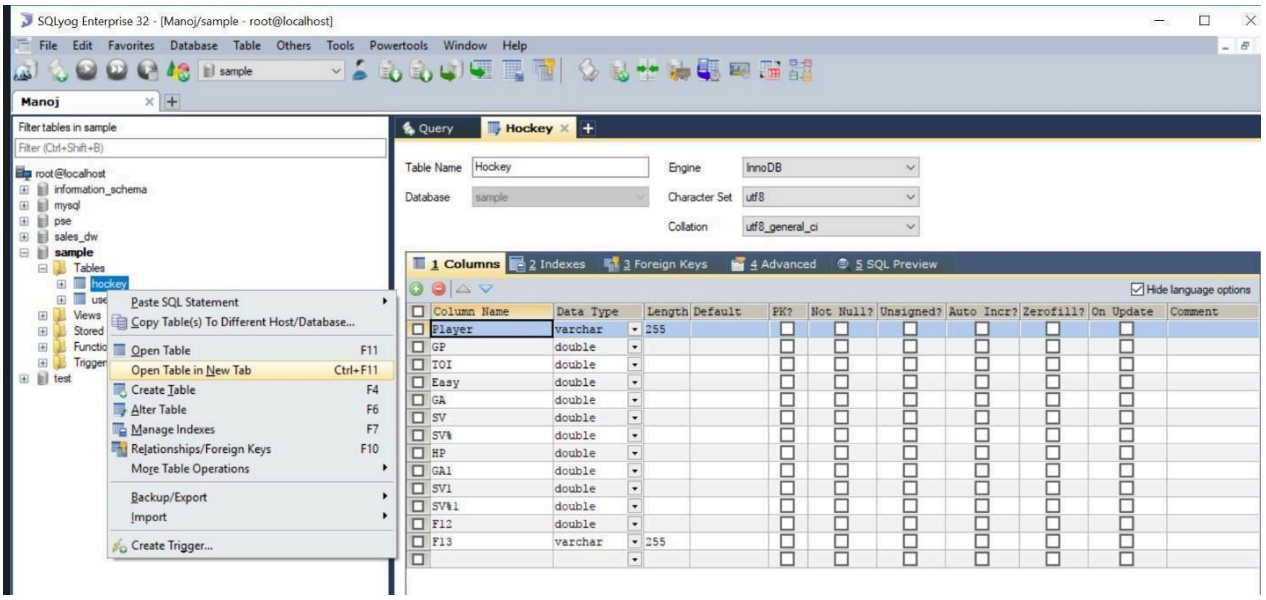
After successful login, it will open a new window as shown below.

There are different options available in MySQL administrator. Another tool, SQLyog Enterprise, we are using for building & identifying tables in a database after successful connection establishment through MySQL Administrator. Below we can see the window of SQLyog Enterprise.



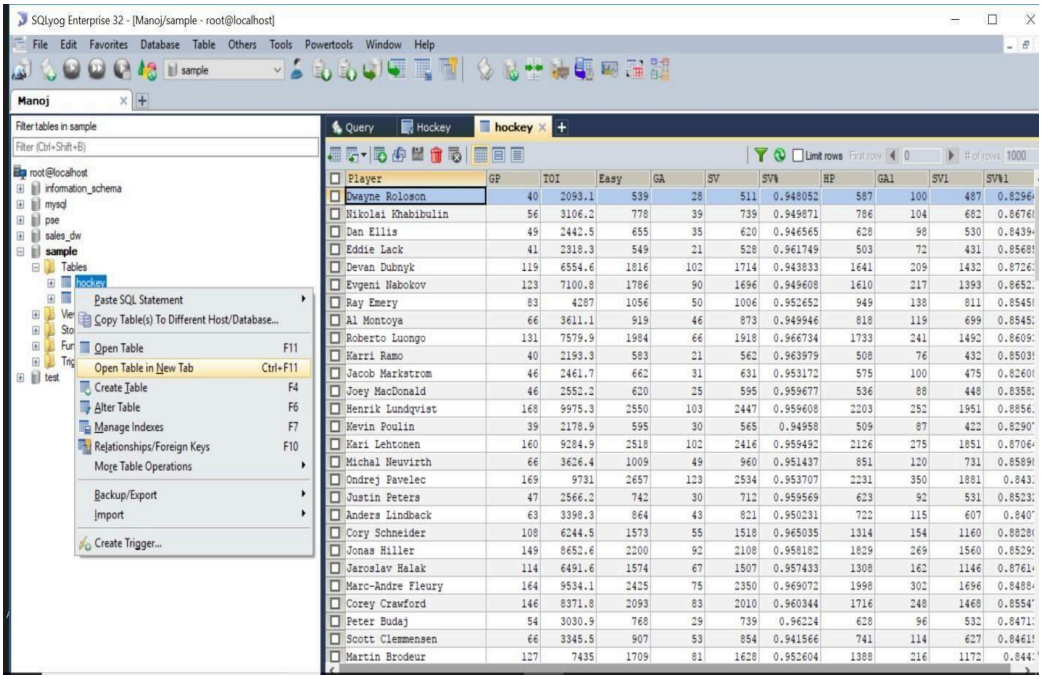
On left-side navigation, we can see different databases & their related tables. Now we are going to build tables & populate table‘s data in the database through SQL queries. These tables in the database can be used further for building a data warehouse.

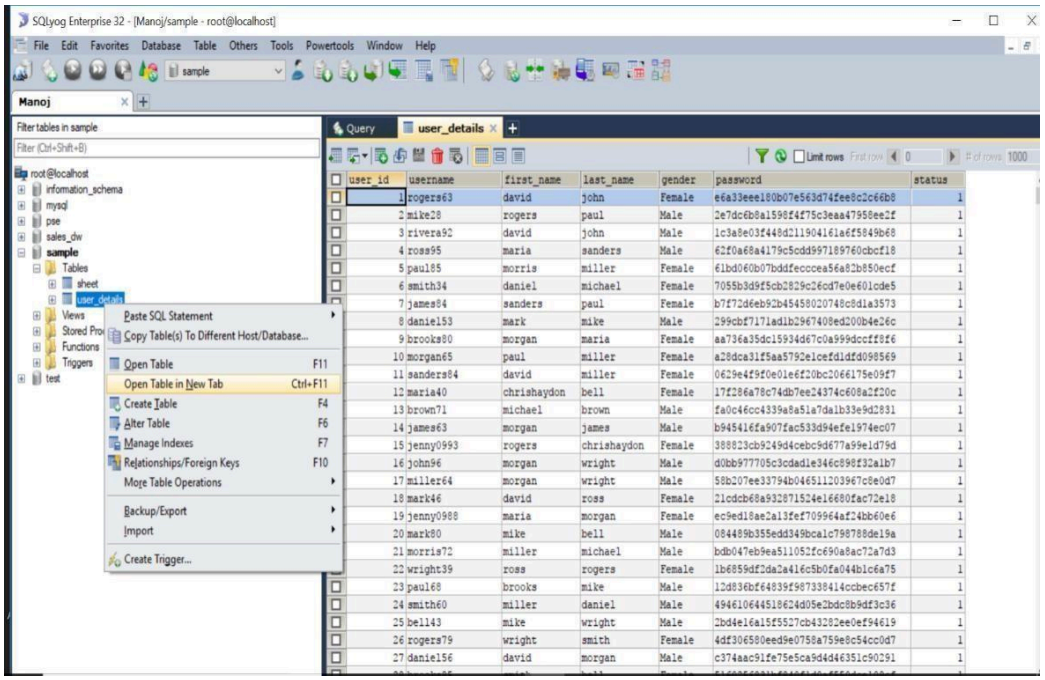




In the above two windows, we created a database named **“sample”**& in that database we created two tables named as **“user_details' '& “hockey' 'through** SQL queries.

Now, we are going to populate (fill) sample data through SQL queries in those two created tables as represented in below windows.





Through MySQL administrator & SQLyog, we can import databases from other sources (.XLS,.CSV,.SQL) & also we can export our databases as backup for further processing. We can connectMySQL to other applications for data analysis & reporting.

Experiment 12

Aim: a) Write ETL scripts and implement using data warehouse tools.

b) Perform various OLAP operations such as slice, dice, roll up, drill down and pivot. OLAP

a) Write ETL scripts and implement using data warehouse tools ETL (Extract-Transform-Load):

ETL comes from Data Warehousing and stands for Extract-Transform-Load. ETL covers a process of how the data is loaded from the source system to the data warehouse. Currently, the ETL encompasses a cleaning step as a separate step. The sequence is then Extract-Clean- Transform-Load. Let us briefly describe each step of the ETL process.

Process Extract:

The Extract step covers the data extraction from the source system and makes it accessible for further processing. The main objective of the extract step is to retrieve all the required data from the source system with as little resources as possible. The exact step should be designed in a way that it does not negatively affect the source system in terms or performance, response time or any kind of locking.

There are several ways to perform the extract:

- Update notification - if the source system is able to provide a notification that a record has been changed and describe the change, this is the easiest way to get the data.
- Incremental extract - some systems may not be able to provide notification that an update has occurred, but they are able to identify which records have been modified and provide an extract of such records. During further ETL steps, the system needs to identify changes and propagate it down. Note, that by using daily extract, we may not be able to handle deleted records properly.
- Full extract - some systems are not able to identify which data has been changed at all, so a full extract is the only way one can get the data out of the system. The full extract requires keeping a copy of the last extract in the same format in order to be able to identify changes. Full extract handles deletions as well.

When using Incremental or Full extracts, the extract frequency is extremely important. Particularly for full extracts; the data volumes can be in tens of gigabytes.

Clean:

The cleaning step is one of the most important as it ensures the quality of the data in the data warehouse. Cleaning should perform basic data unification rules, such as:

- Making identifiers unique (sex categories Male/Female/Unknown, M/F/null, Man/Woman/Not Available are translated to standard Male/Female/Unknown)
- Convert null values into standardized Not Available/Not Provided value
- Convert phone numbers, ZIP codes to a standardized form
- Validate address fields, convert them into proper naming, eg. Street/St/St./Str./Str
- Validate address fields against each other (State/Country, City/State, City/ZIP code, City/Street).

Transform:

The transform step applies a set of rules to transform the data from the source to the target. This includes converting any measured data to the same dimension (i.e. conformed dimension) using the same units so that they can later be joined. The transformation step also requires joining data from several sources, **generating** aggregates, generating surrogate keys, sorting, deriving new calculated values, and applying advanced validation rules.

Load:

During the load step, it is necessary to ensure that the load is performed correctly and with as little resources as possible. The target of the Load process is often a database. In order to make the load process efficient, it is helpful to disable any constraints and indexes before the load and enable them back only after the load completes. The referential integrity needs to be maintained by ETL tools to ensure consistency.

Managing ETL Process:

The ETL process seems quite straightforward. As with every application, there is a possibility that the ETL process fails. This can be caused by missing extracts from one of the systems, missing values in one of the reference tables, or simply a connection or power outage. Therefore, it is necessary to design the ETL process keeping fail-recovery in mind.

Staging:

It should be possible to restart, at least, some of the phases independently from the others. For example, if the transformation step fails, it should not be necessary to restart the Extract step. We can ensure this by implementing proper staging. Staging means that the data is simply dumped to the location (called the Staging Area) so that it can then be read by the next processing phase. The staging area is also used during the ETL process to store intermediate results of processing. This is ok for the ETL process which is used for this purpose. However, The staging area should be accessed by the load ETL process only. It should never be available to anyone else; particularly not to end users as it is not intended for data presentation to the end-user.may contain incomplete or in-the-middle-of-the- processing data.

ETL Tool Implementation:

When you are about to use an ETL tool, there is a fundamental decision to be made: will the company build its own data transformation tool or will it use an existing tool?

Building your own data transformation tool (usually a set of shell scripts) is the preferred approach for a small number of data sources which reside in storage of the same type. The reason for that is the effort to implement the

necessary transformation is little due to similar data structure and common system architecture. Also, this approach saves licensing cost and there is no need to train the staff in a new tool. This approach, however, is dangerous from the TOC point of view. If the transformations become more sophisticated during the time or there is a need to integrate other systems, the complexity of such an ETL system grows but the manageability drops significantly.
Similarly, the implementation of your own tool often resembles re-inventing the wheel.

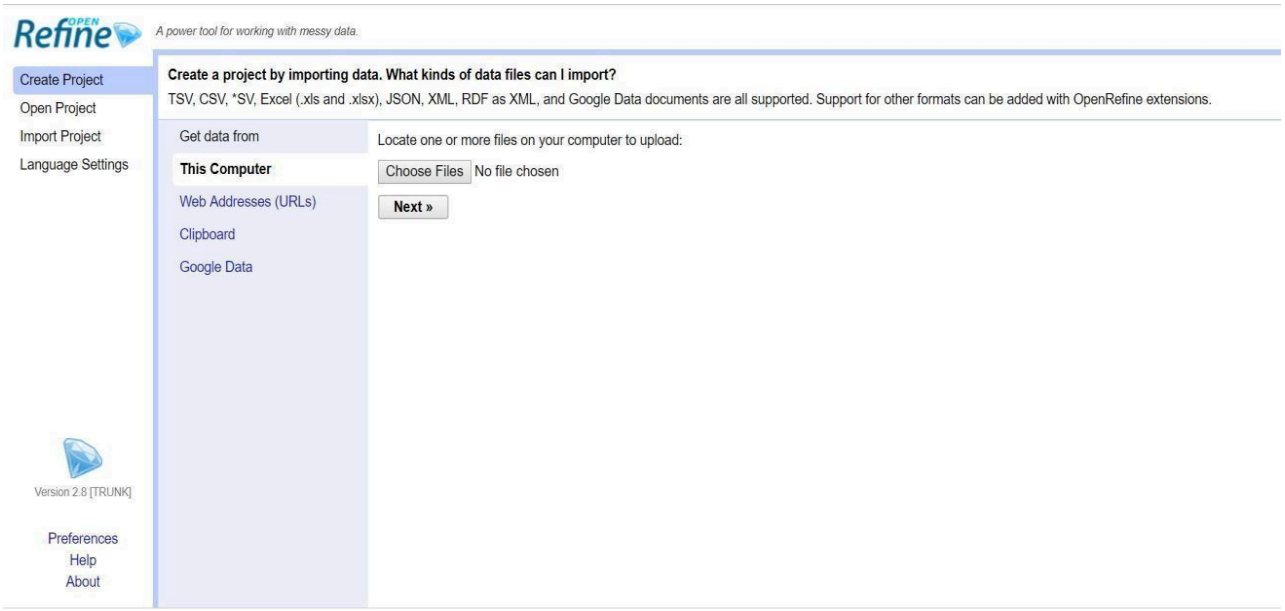
There are many ready-to-use ETL tools on the market. The main benefit of using off-the-shelf ETL tools is the fact that they are optimized for the ETL process by providing connectors to common data sources like databases, flat files, mainframe systems, xml, etc. They provide a means to implement data transformations easily and consistently across various data sources. This includes filtering, reformatting, sorting, joining, merging, aggregation and other operations ready to use. The tools also support transformation scheduling, version control, monitoring and unified metadata management. Some of the ETL tools are even integrated with BI tools**.**

Some of the Well Known ETL Tools:

The most well-known commercial tools are Ab Initio, IBM InfoSphere DataStage, Informatica, Oracle Data Integrator, and SAP Data Integrator.

There are several open source ETL tools: Open Refine, Apatar, Clover ETL, Pentaho and Talend.

In these above tools, we are going to use the Open Refine 2.8 ETL tool for different sample datasets for extracting, data cleaning, transforming & loading.



b) Perform various OLAP operations such slice, dice, roll up, drill down and pivot.OLAP

Operations:

Since OLAP servers are based on a multidimensional view of data, we will discuss OLAPoperations in multidimensional data.

Here is the list of OLAP operations

- Roll-up (Drill-up)

- Drill-down
- Slice and dice
- Pivot (rotate)

Roll-up (Drill-up):

Roll-up performs aggregation on a data cube in any of the following ways

- By climbing up a concept hierarchy for a dimension
- By dimension reduction
- Roll-up is performed by climbing up a concept hierarchy for the dimension location.
- Initially the concept hierarchy was "street < city < province < country".
- On rolling up, the data is aggregated by ascending the location hierarchy from the level of the city to the level of the country.
- The data is grouped into cities rather than countries.
- When roll-up is performed, one or more dimensions from the data cube are removed.

Drill-down:

Drill-down is the reverse operation of roll-up. It is performed by either of the following ways

- By stepping down a concept hierarchy for a dimension
- By introducing a new dimension.
- Drill-down is performed by stepping down a concept hierarchy for the dimension time.
- Initially the concept hierarchy was "day < month < quarter < year."
- On drilling down, the time dimension is descended from the level of quarter to the level of month.
- When drill-down is performed, one or more dimensions from the data cube are added.
- It navigates the data from less detailed data to highly detailed data.

Slice:

The slice operation selects one particular dimension from a given cube and provides a new sub-cube.

Dice:

Dice selects two or more dimensions from a given cube and provides a new sub-cube.

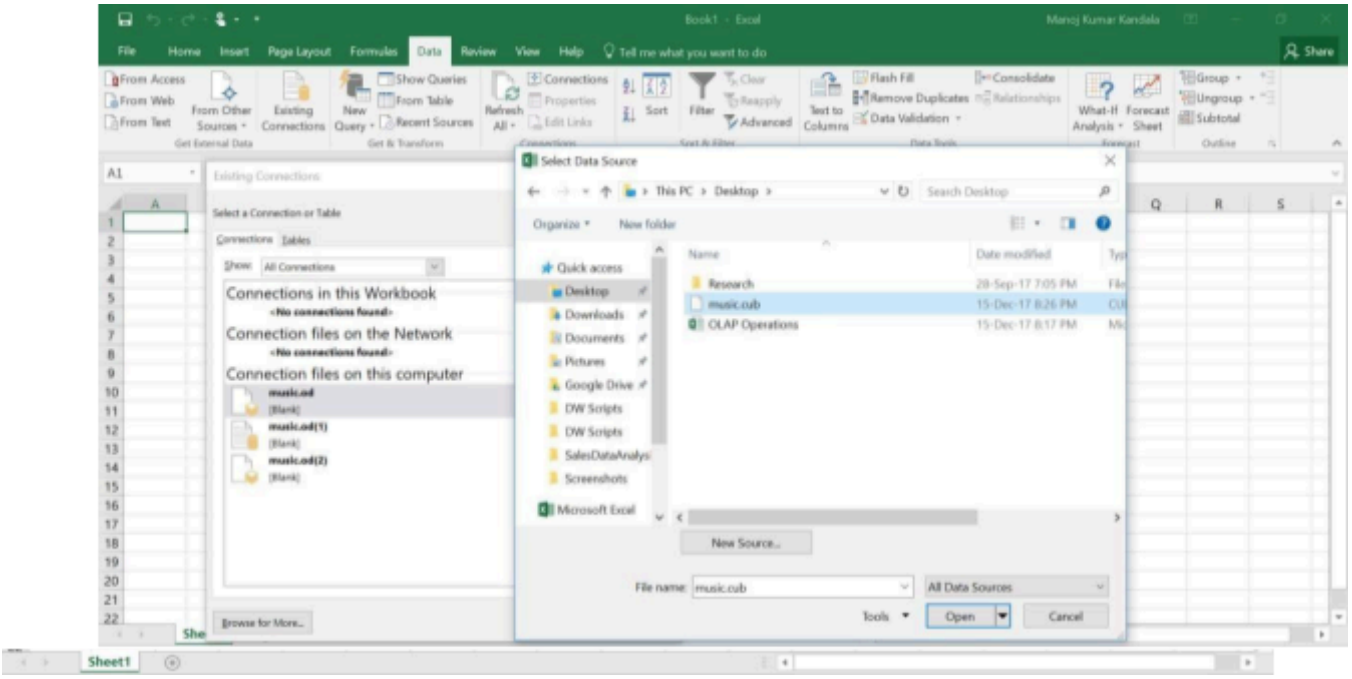
Pivot (rotate):

The pivot operation is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data.

Now, we are practically implementing all these OLAP Operations using Microsoft Excel.

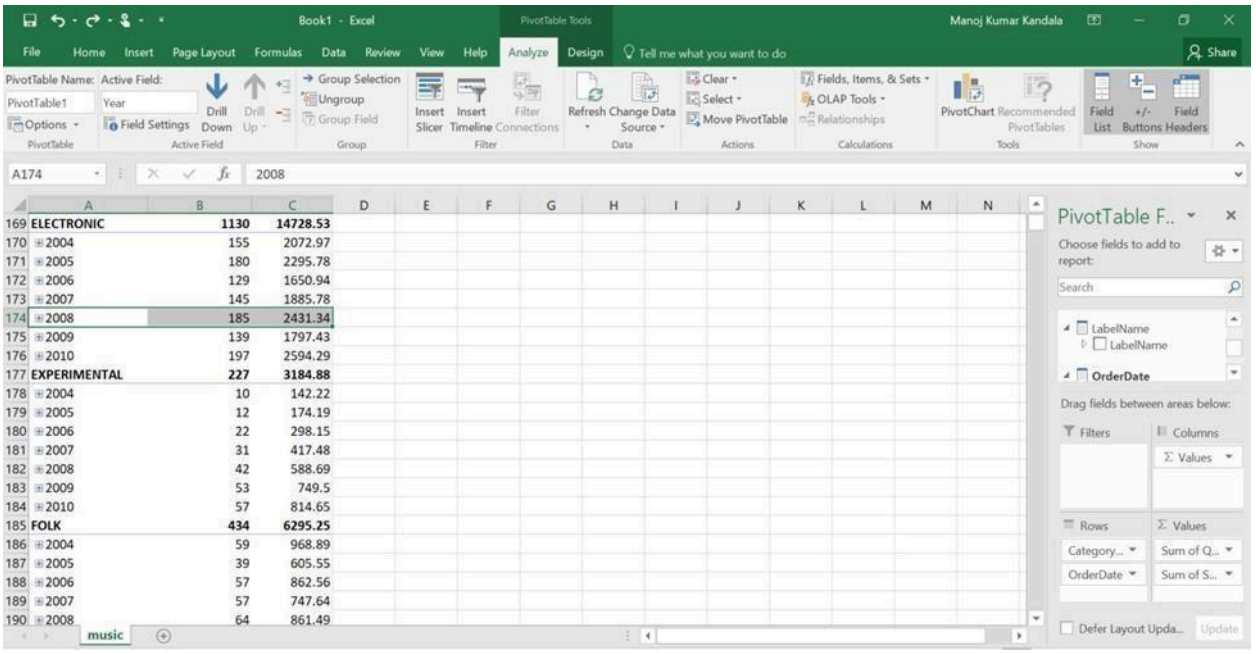
Procedure for OLAP Operations:

1. Open Microsoft Excel, go to the Data tab at the top & click on —"Existing Connections".
2. Existing Connections window will be opened, there “Browse for more” option should be clicked for importing .cub extension file for performing OLAP Operations. For a sample, I took a music.cub file.



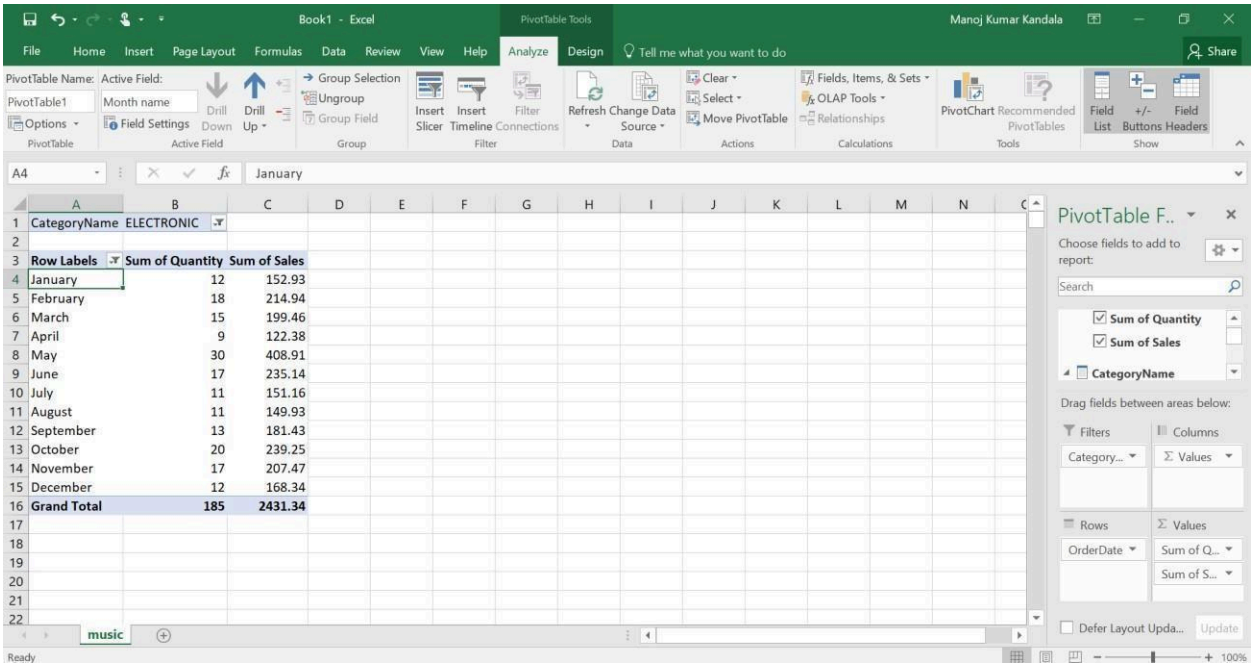
3. As shown in the above window, select ―PivotTable Report" and click"OK ".

4. We got all the music.cub data for analyzing different OLAP Operations. Firstly, we performed a drill-down operation as shown below.

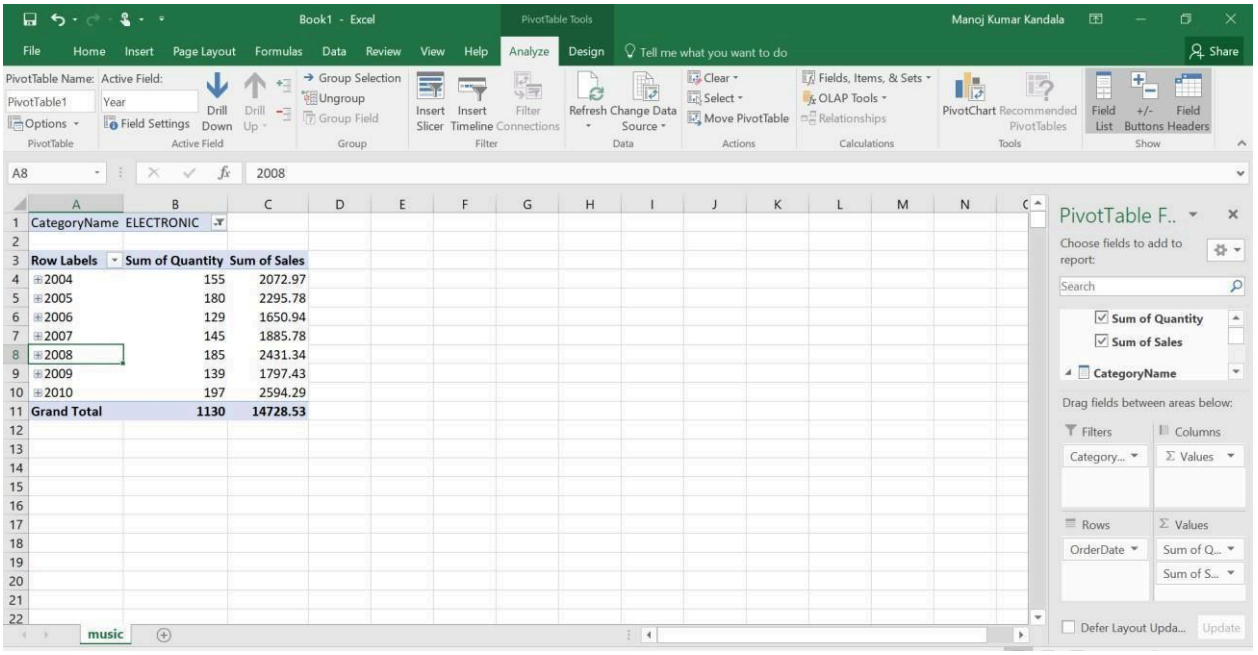


169	ELECTRONIC	1130	14728.53
170	2004	155	2072.97
171	2005	180	2295.78
172	2006	129	1650.94
173	2007	145	1885.78
174	2008	185	2431.34
175	2009	139	1797.43
176	2010	197	2594.29
177	EXPERIMENTAL	227	3184.88
178	2004	10	142.22
179	2005	12	174.19
180	2006	22	298.15
181	2007	31	417.48
182	2008	42	588.69
183	2009	53	749.5
184	2010	57	814.65
185	FOLK	434	6295.25
186	2004	59	968.89
187	2005	39	605.55
188	2006	57	862.56
189	2007	57	747.64
190	2008	64	861.49

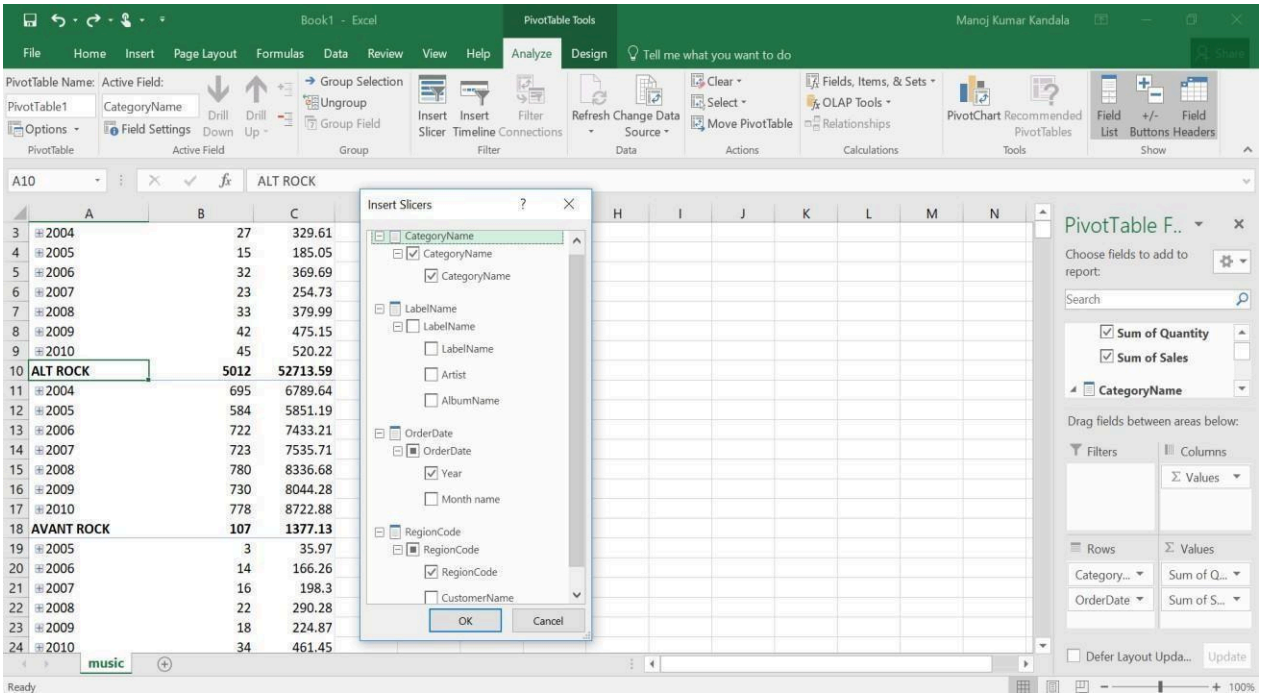
In the above window, we selected the year „2008" in „Electronic" Category, then automatically the Drill-Down option is enabled on top navigation options. We will click on„Drill-Down" option, then the below window will be displayed



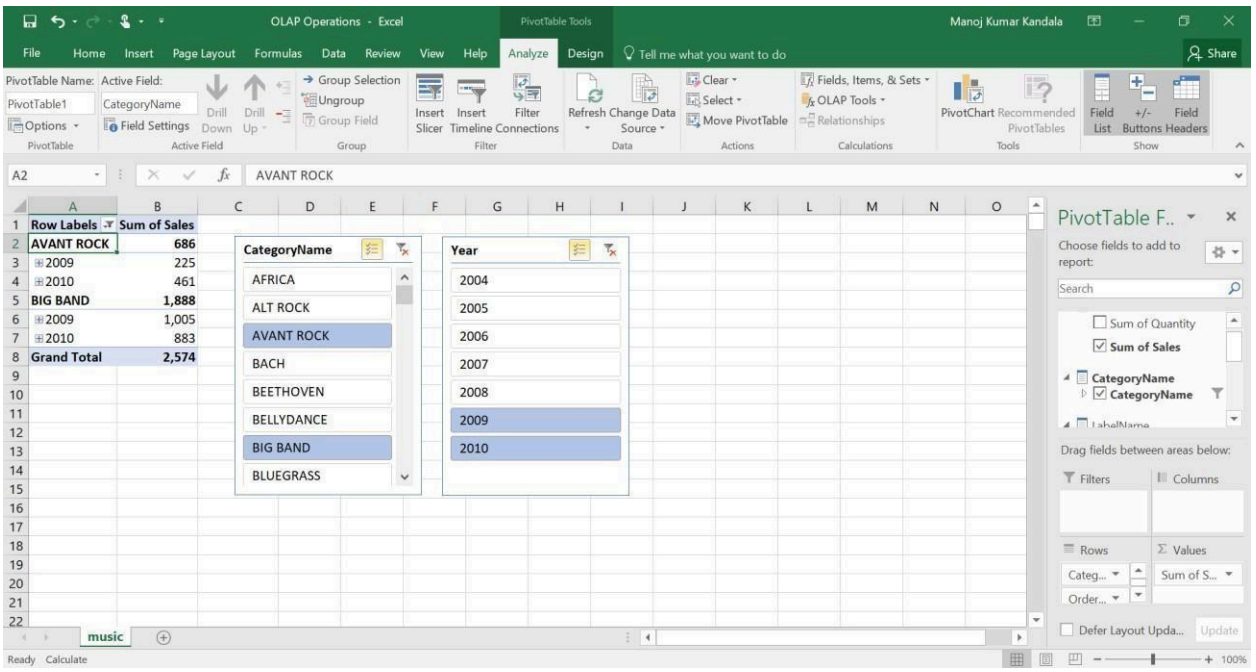
5. Now we are going to perform a roll-up (drill-up) operation. In the above window I selected January month, then automatically the Drill-up option is enabled on top. We will click on Drill-up option, then the below window will be displayed



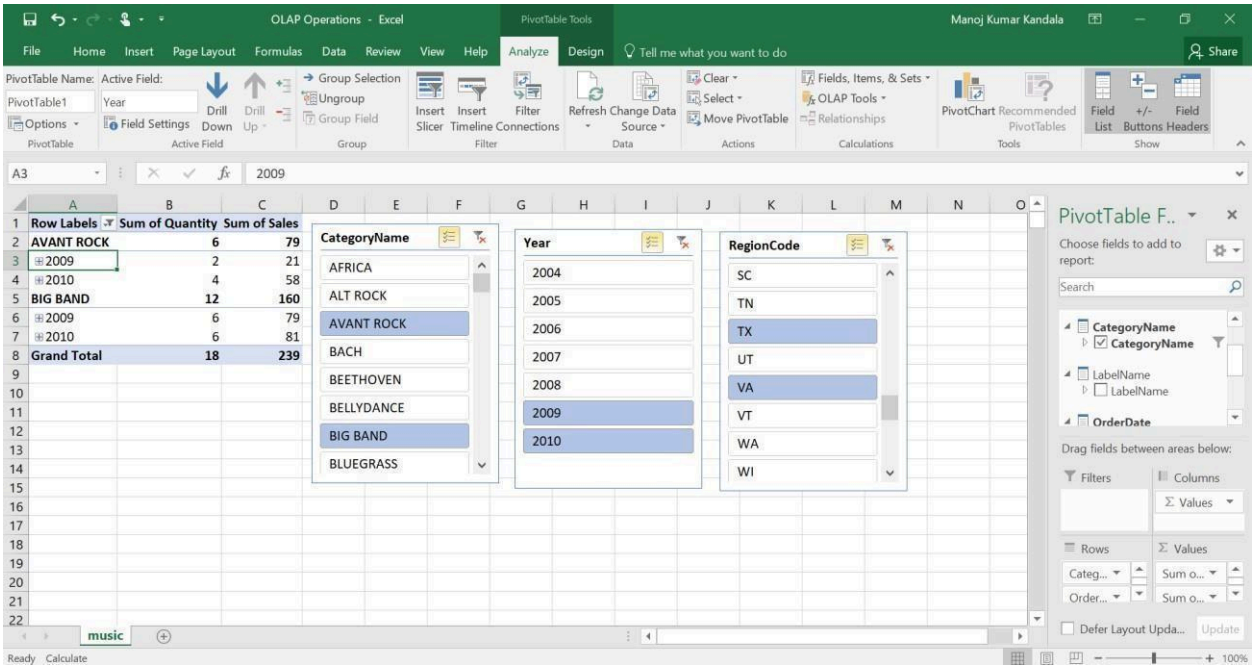
6.Next OLAP operation Slicing is performed by inserting slicer as shown in top navigation options



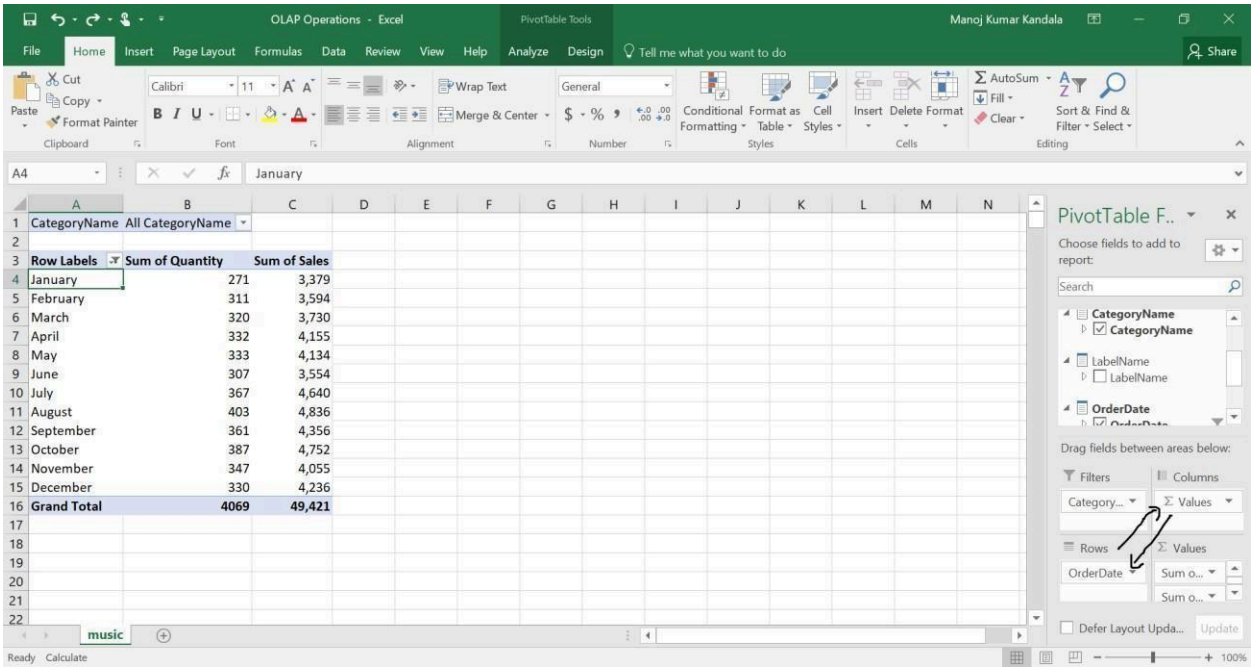
While inserting slicers for slicing operation, we select 2 Dimensions (for eg. CategoryName & Year) only with one Measure (for e.g. Sum of sales). After inserting a slice & adding a filter (CategoryName: AVANT ROCK & BIG BAND; Year: 2009 & 2010), we will get the table as shown below.



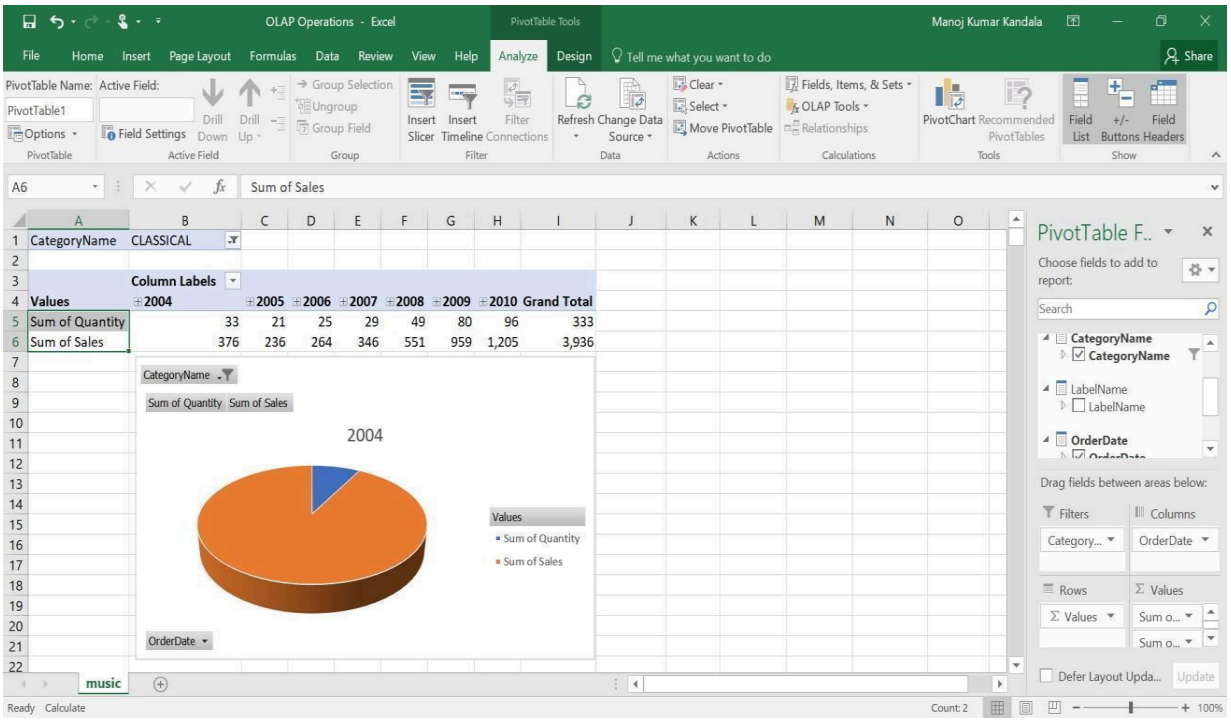
7. Dicing operation is similar to Slicing operation. Here we are selecting 3 dimensions (Category Name, Year, Region Code) & 2 Measures (Sum of Quantity, Sum of Sales) through the insert slicer‟ option. After that adding a filter for Category Name, Year & RegionCode as shown below



8. Finally, the Pivot (rotate) OLAP operation is performed by swapping rows (Order Date-Year) & columns (Values-Sum of Quantity & Sum of Sales) through the right side bottom navigation bar as shown below.



After Swapping (rotating), we will get resultant as represented below with a pie-chart for Category-Classical& Year Wise data.


OLAP Operations - Excel
PivotTable Tools
Manoj Kumar Kandala
File Home Insert Page Layout Formulas Data Review View Help Analyze Design Tell me what you want to do Share
PivotTable Name: PivotTable1 Options
Active Field: Field Settings
Drill Down Drill Up
Group Selection Ungroup Group Field
Insert Slicer Insert Timeline Filter Connections
Refresh Change Data Source
Clear Select Move PivotTable
Fields, Items, & Sets OLAP Tools Relationships
PivotChart Recommended PivotTables
Field List +/- Buttons Field Headers
PivotTable Active Field Group Filter Data Actions Calculations Tools Show
A6
Sum of Sales
CategoryName CLASSICAL
Column Labels
Values 2004 2005 2006 2007 2008 2009 2010 Grand Total
Sum of Quantity 33 21 25 29 49 80 96 333
Sum of Sales 376 236 264 346 551 959 1,205 3,936
CategoryName
Sum of Quantity Sum of Sales
2004
Values
Sum of Quantity
Sum of Sales
OrderDate
music
Ready Calculate
Count: 2
100%
PivotTable F..
Choose fields to add to report:
Search
CategoryName
CategoryName
LabelName
LabelName
OrderDate
Drag fields between areas below:
Filters
Columns
Category...
OrderDate
Rows
Values
Σ Values
Sum o...
Sum o...
Defer Layout Upda...
Update

Experiment 13

Aim: Explore WEKA Data Mining/Machine Learning Toolkit.

a) Downloading and/or installation of WEKA data mining

toolkit Procedure:

1. Go to the Weka website, http://www.cs.waikato.ac.nz/ml/weka/, and download the software.On the left-hand side, click on the link that says download.

2. Select the appropriate link corresponding to the version of the software based on your operating system and whether or not you already have a Java VM running on your machine (if you don't know what a Java VM is, then you probably don‘t).

3. The link will forward you to a site where you can download the software from a mirror site. Save the self- extracting executable to disk and then double click on it to install Weka. Answer yes or next to the questions during the installation.

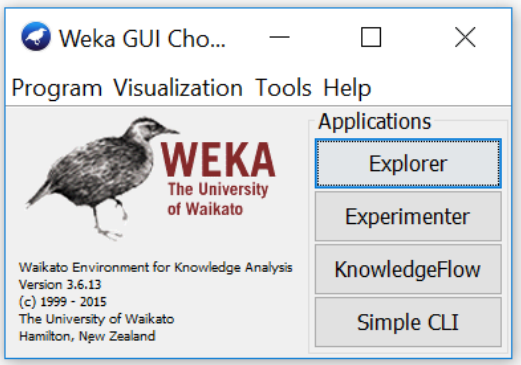
4. Click yes to accept the Java agreement if necessary. After you install the program Weka should appear on your start menu under Programs (if you are using Windows).

5. Running Weka from the start menu select Programs, then Weka.You will see the Weka GUI Chooser. Select Explorer. The Weka Explorer will then launch

b) Understand the features of WEKA toolkit such as Explorer, Knowledge Flowinter- face, Experimenter, command-line interface.

The Weka GUI Chooser (class weka.gui.GUIChooser) provides a starting point for launching Weka‘s main GUI applications and supporting tools. If one prefersa MDI (―multiple document interface‖) appearance, then this is provided by an alternative launcher called ―Main‖ (class weka.gui.Main).

The GUI Chooser consists of four buttons—one for each of the four majorWeka applications— and four me.



The buttons can be used to start the following applications:

Explorer- An environment for exploring data with WEKA

a) Click on ―explorer‖ button to bring up the explorer window.
b) Make sure the ―preprocess‖ tab is highlighted.
c) Open a new file by clicking on ―Open New file‖ and choosing a file with ―.arff‖ extension from the ―Data‖ directory.
d) Attributes appear in the window below.
e) Click on the attributes to see the visualization on the right.
f) Click ―visualize all‖ to see them all

Experimenter- An environment for performing experiments and conducting statistical tests between learning schemes.

a) Experimenter is for comparing results.
b) Under the ―set up‖ tab click ―New‖.
c) Click on ―Add New‖ under ―Data‖ frame. Choose a couple of arff format files from ―Data‖ directory one at a time.
d) Click on ―Add New‖ under ―Algorithm‖ frame. Choose several algorithms, one at a time by click- ing ―OK‖ in the window and ―Add New‖.
e) Under the ―Run‖ tab click ―Start‖.
f) Wait for WEKA to finish.
g) Under ―Analyses‖ tab click on ―Experiment‖ to see results.

Knowledge Flow- This environment supports essentially the same functions as the Explorer but with a drag-and- drop interface. One advantage is that it supports incremental learning.

Experiment 14

Aim: Provides a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface. Navigate the options available in the WEKA (ex. Select attributes panel, Preprocesspanel, classify panel, Cluster panel, Associate panel and Visualize panel)

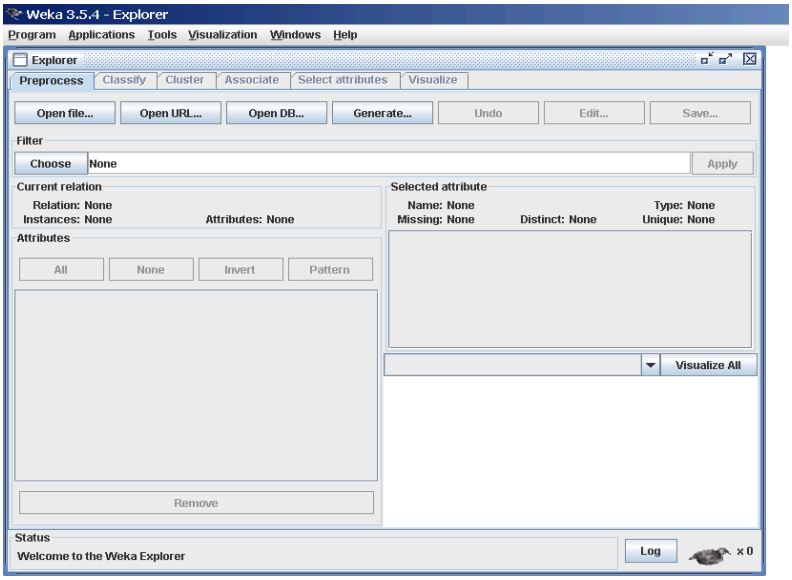
When the Explorer is first started only the first tab is active; the others are grayed out. This is because it is necessary to open (and potentially pre-process) a data set before starting to explore the data.

The tabs are as follows:

1. Pre-process. Choose and modify the data being acted on.
2. Classify. Train and test learning schemes that classify or perform regression.
3. Cluster. Learn clusters for the data.
4. Associate. Learn association rules for the data.
5. Select attributes. Select the most relevant attributes in the data.
6. Visualize. View an interactive 2D plot of the data.

Once the tabs are active, clicking on them flicks between different screens, on which the respective actions can be performed. The bottom area of the window (including the status box,the log button, and the Weka bird) stays visible regardless of which section you are in.

1. **Preprocessing**



Loading Data:

The first four buttons at the top of the pre-process section enable you to load data into WEKA:

1. Open file Brings up a dialog box allowing you to browse for the datafile on the local file system.

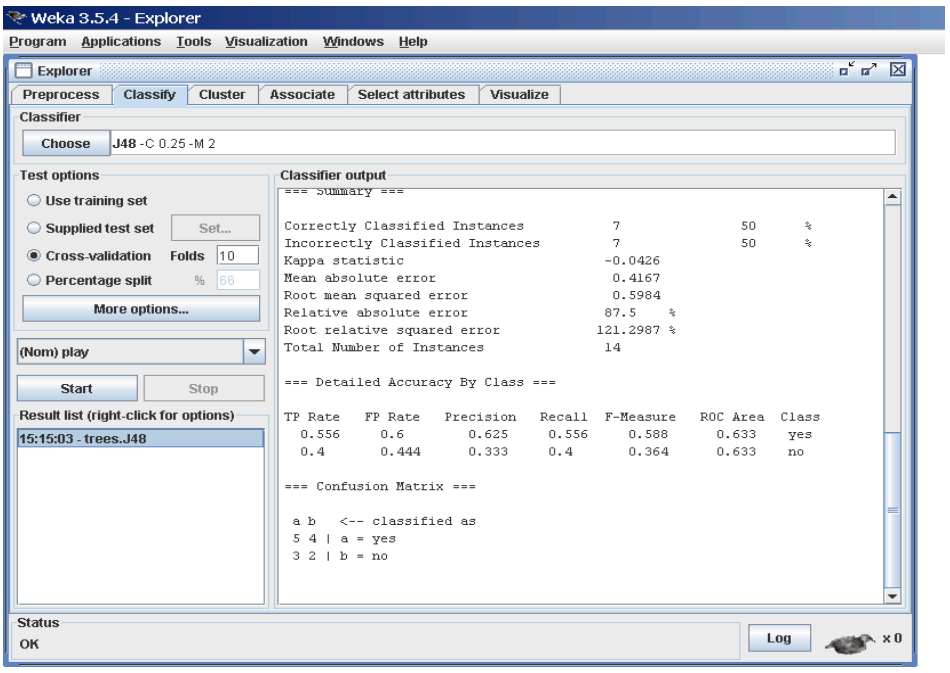
2. Open URL Asks for a Uniform Resource Locator address for where the data is stored.

3. Open DB Reads data from a database. (Note that to make this work you might have to edit the file in Weka/experiment/DatabaseUtils.props.)

4. Generate. Enables you to generate artificial data from a variety of Data Generators. Using the Open file button you can read files in a variety of formats:

WEKA's ARFF format, CSV format, C4.5 format, or serialized Instances format. ARFF files typically have a.arff extension, CSV files a .csv extension,C4.5 files a .data and .names extension, and serialized Instances objects
a .bsi extension.

2. Classification:



Selecting a Classifier

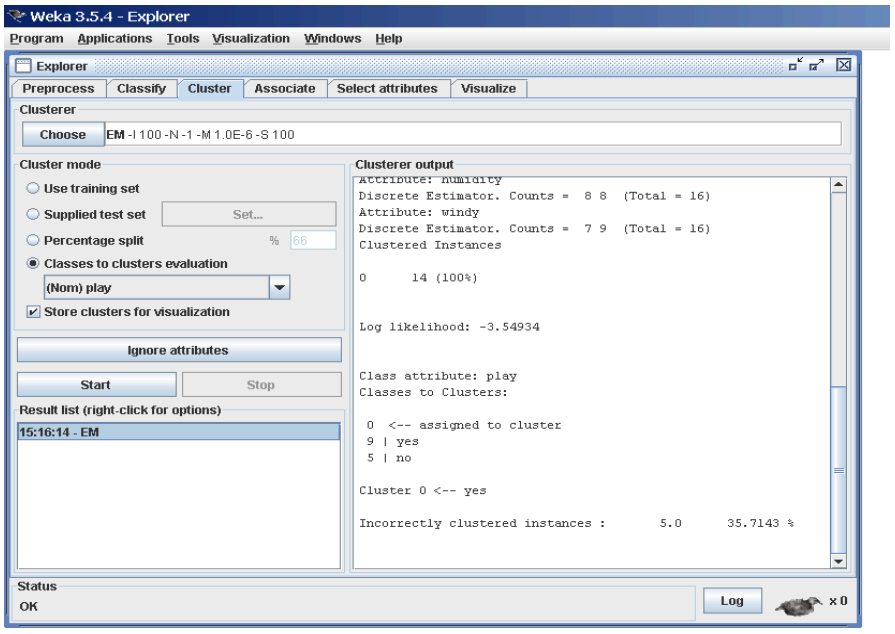
At the top of the classify section is the Classifier box. This box has a text field that gives the name of the currently selected classifier, and its options. Clicking on the text box with the left mouse button brings up a Generic Object Editor dialog box, just the same as for filters, that you can use to configure the options of the current classifier. With a right click (or Alt+Shift+left click) you can once again copy the setup string to the clipboard or display the properties in a Generic Object Editor dialog box. The Choose button allows you to choose from 4 of the classifiers that are available in WEKA.

Test Options

The result of applying the chosen classifier will be tested according to the options that are set by clicking in the Test options box. There are four test modes:

1. Use training set: The classifier is evaluated on how well it predicts the class of the instances it was trained on.
2. Supplied test set: The classifier is evaluated on how well it predicts the class of a set of instances loaded from a file. Clicking the Set... button brings up a dialog allowing you to choose the file to test on.
3. Cross-validation: The classifier is evaluated by cross-validation, using the number of folds that are entered in the Folds text field.
4. Percentage split: The classifier is evaluated on how well it predicts a certain percentage of the data which is held out for testing. The amount of data held out depends on the value entered in the % field.

3. Clustering:



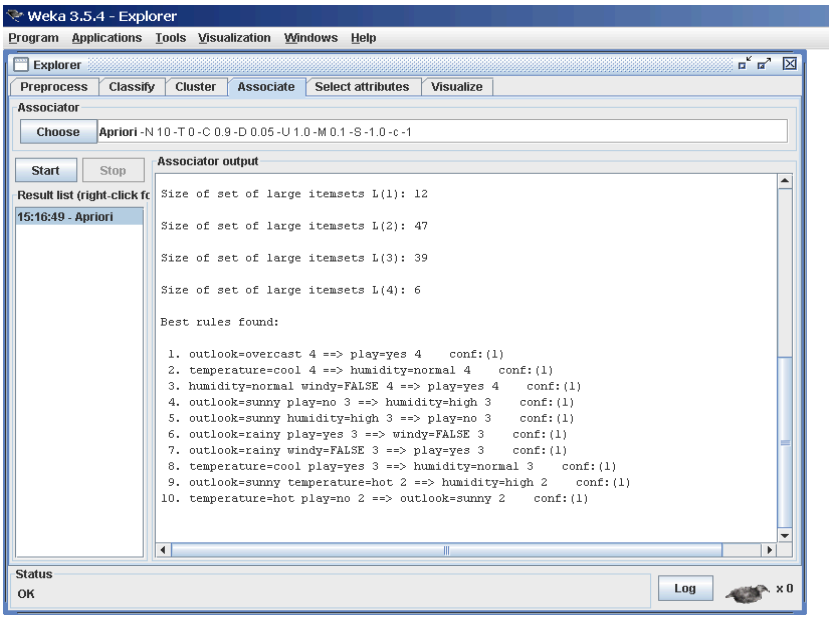
Cluster Modes:

The Cluster mode box is used to choose what to cluster and how to evaluate the results. The first three options are the same as for classification: Use training set, Supplied test set and Percentage split.

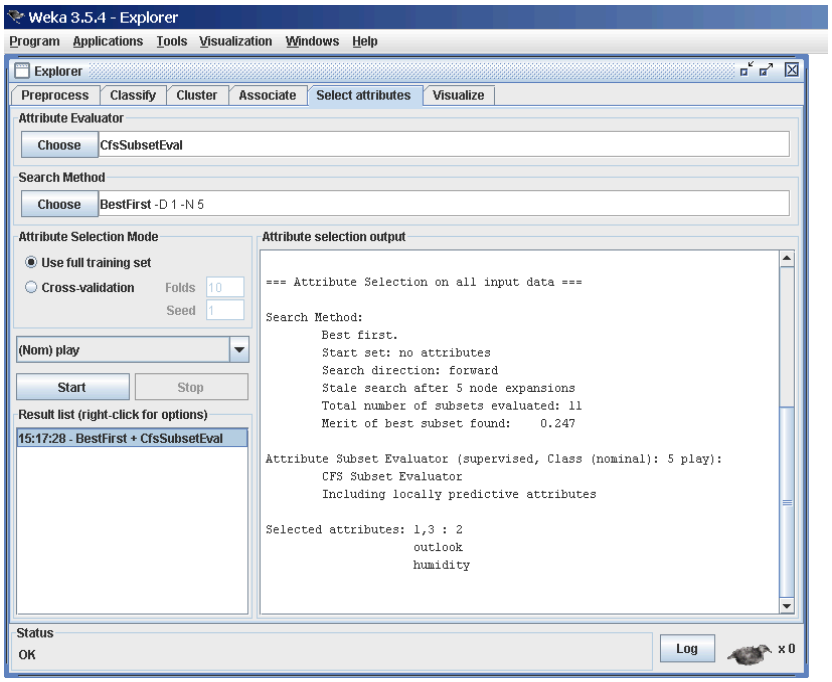
4. Associating:

Setting Up

This panel contains schemes for learning association rules, and the learners are chosen and config- ured in the same way as the clusters, filters, and classifiers in the other panels.



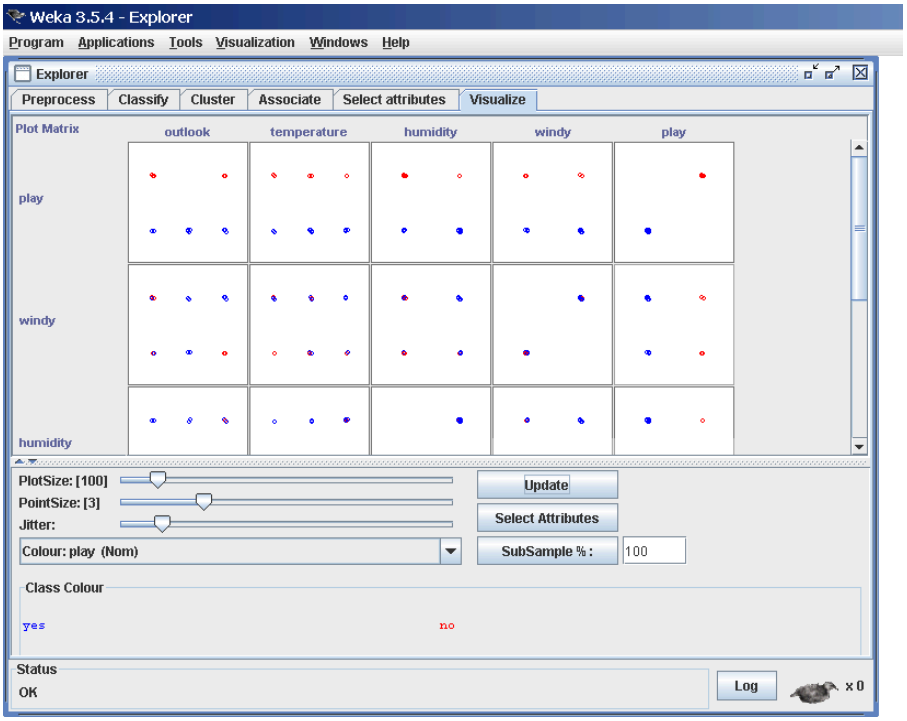
5. Selecting Attributes:



Searching and Evaluating

Attribute selection involves searching through all possible combinations of attributes in the data to find which subset of attributes works best for prediction. To do this, two objects must be set up: an attribute evaluator and a search method. The evaluator determines what method is used to assign a worth to each subset of attributes. The search method determines what style of search is performed

6. Visualizing:



WEKA‘s visualization section allows you to visualize 2D plots of the current relation

Experiment 15

Aim: Study the arff file format Explore the available data sets in WEKA. Load a data set (ex. Weather dataset, Iris dataset, etc.)

Study the arff file format

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software.

Overview

ARFF files have two distinct sections. The first section is the **Header** information, which is followed by the **Data** information. The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

% 1. Title: Iris Plants Database

%

% 2. Sources:

% (a) Creator: R.A. Fisher
% (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

% (c) Date: July, 1988

@RELATION iris

@ATTRIBUTE sepal length NUMERIC
@ATTRIBUTE sepal width NUMERIC
@ATTRIBUTE petal length NUMERIC
@ATTRIBUTE petal width NUMERIC

@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}The

Data of the ARFF file looks like the following:

@DATA

5.1,3.5,1.4,0.2, Iris-setosa

4.9,3.0,1.4,0.2, Iris-setosa

4.7,3.2,1.3,0.2, Iris-setosa

4.6,3.1,1.5,0.2, Iris-setosa

5.0,3.6,1.4,0.2, Iris-setosa

5.4,3.9,1.7,0.4, Iris-setosa

4.6,3.4,1.4,0.3, Iris-setosa

5.0,3.4,1.5,0.2, Iris-setosa

4.4,2.9,1.4,0.2, Iris-setosa

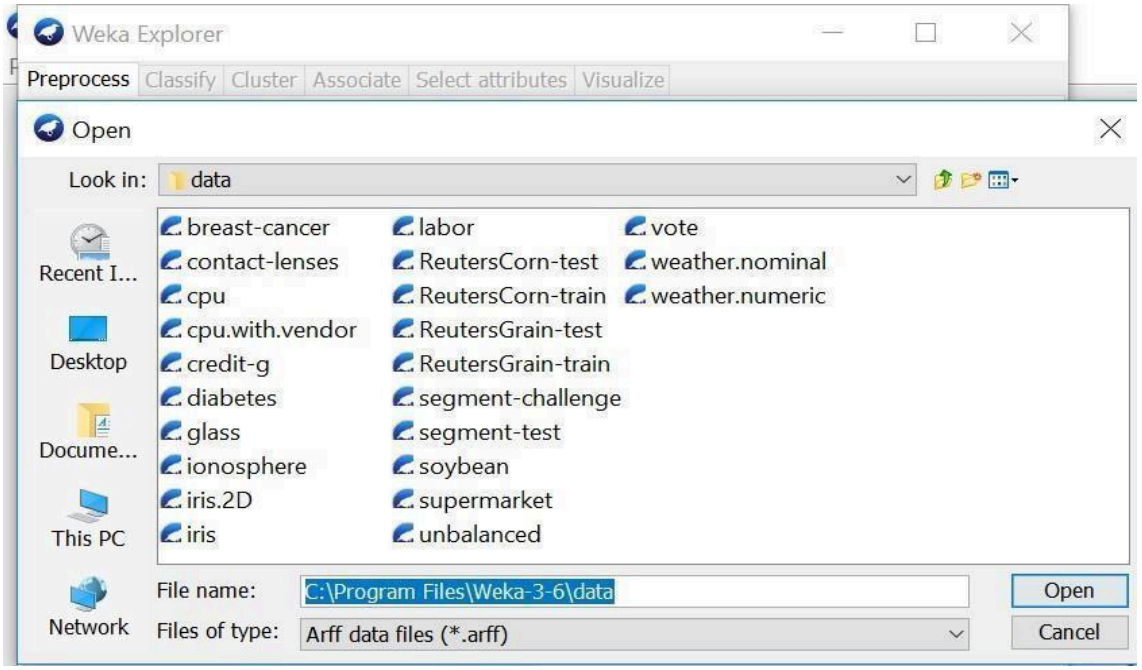
4.9,3.1,1.5,0.1, Iris-setosa

Lines that begin with a % are comments.

The **@RELATION**, **@ATTRIBUTE** and **@DATA** declarations are case insensitive.

a) Explore the available data sets in WEKA

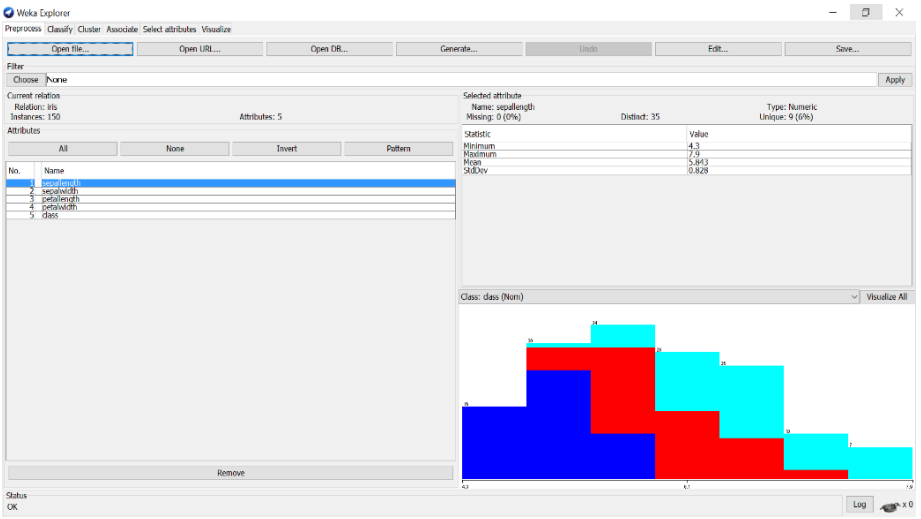
There are 23 different datasets available in weka (C:\Program Files\Weka-3-6\) by default for testing purposes. All the datasets are available in. arff format. Those datasets are listed below.



b) Load a data set (ex. Weather dataset, Iris dataset, etc.)

Procedure:

1. Open the weka tool and select the explorer option.
2. New window will be opened which consists of different options (Pre-process, Association etc.)
3. In the pre-process, click the ―open file‖ option.
4. Go to C:\Program Files\Weka-3-6\data for finding different existing. arff datasets. Click on any dataset for loading the data then the data will be displayed as shown below.



c) Load each dataset and observe the following:

Here we have taken the IRIS.arff dataset as a sample for observing all the below things.

Load each dataset and observe the following:

1. **List the attribute names and their types.**
2. **Number of records in each dataset**
3. **Identify the class attribute (if any)**
4. **Plot Histogram**
5. **Determine the number of records for each class.**
6. **Visualize the data in various dimensions**

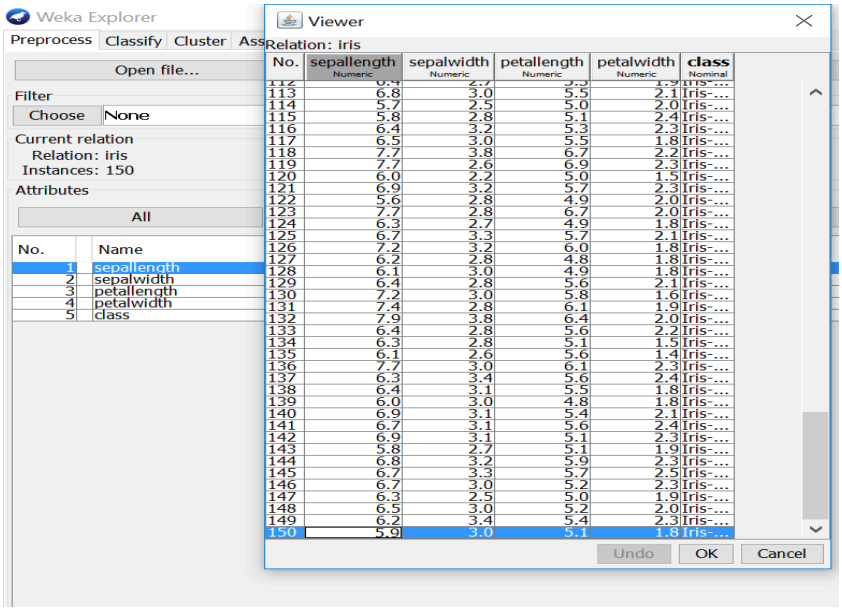
1 List the attribute names and they types

There are 5 attributes & its datatype present in the above loaded dataset (IRIS.arff)
sepal length – Numeric

Sepal width – Numeric
petal length – Numeric petal
length – Numeric Class –
Nominal

2 Number of records in each dataset

There are a total <u>of 150</u> records (Instances) in the dataset (IRIS.arff).

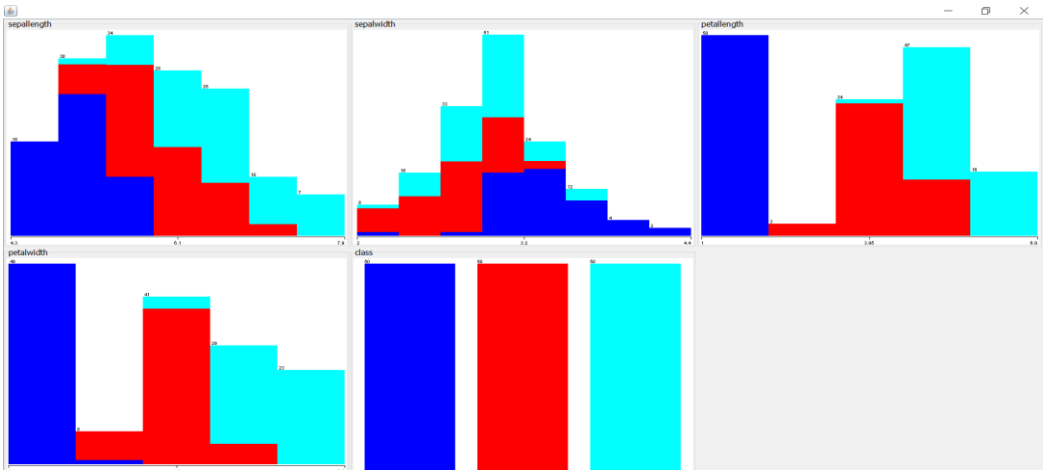


3 Identify the class attribute (if any)

There is one class attribute which consists of 3 labels. They are:

a. Iris-setosa
b. Iris-versicolor
c. Iris-virginica

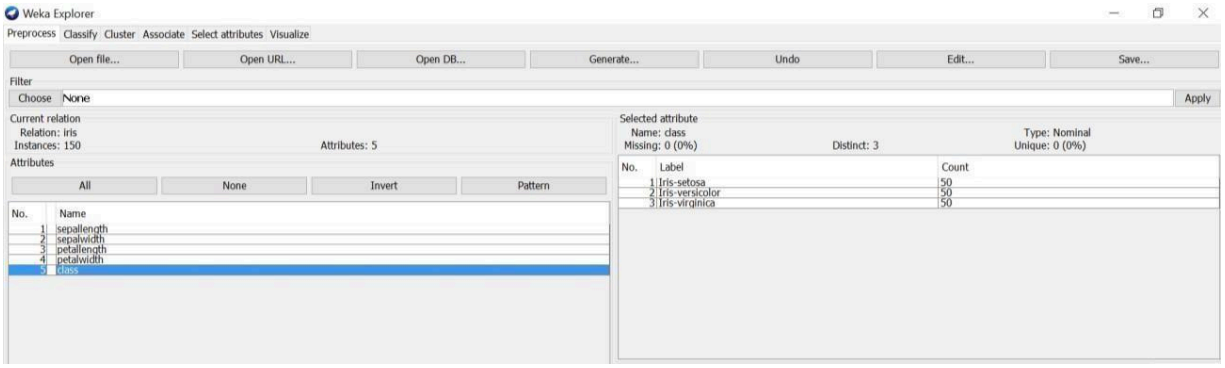
4 Plot Histogram



5 Determine the number of records for each class.

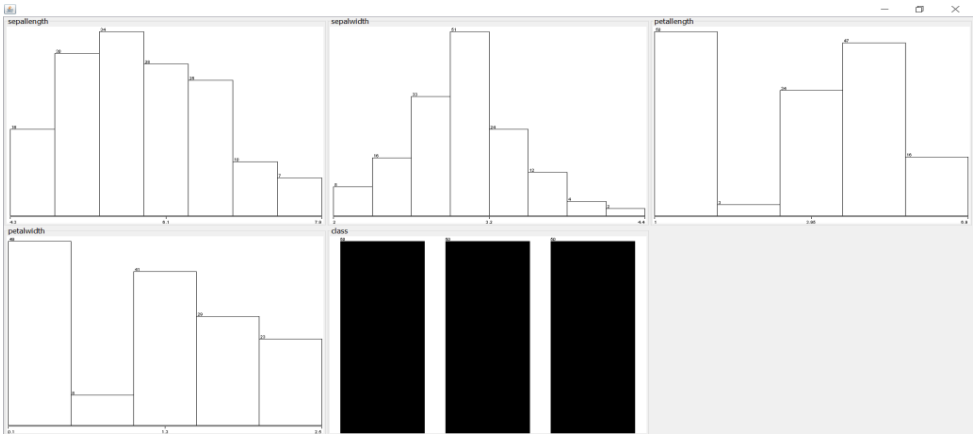
There is one class attribute (150 records) which consists of 3 labels. They are shown below

a. Iris-setosa - 50 records
b. Iris-versicolor – 50 records



c. Iris-virginica – 50 records

6 Visualize the data in various dimensions



Experiment 16

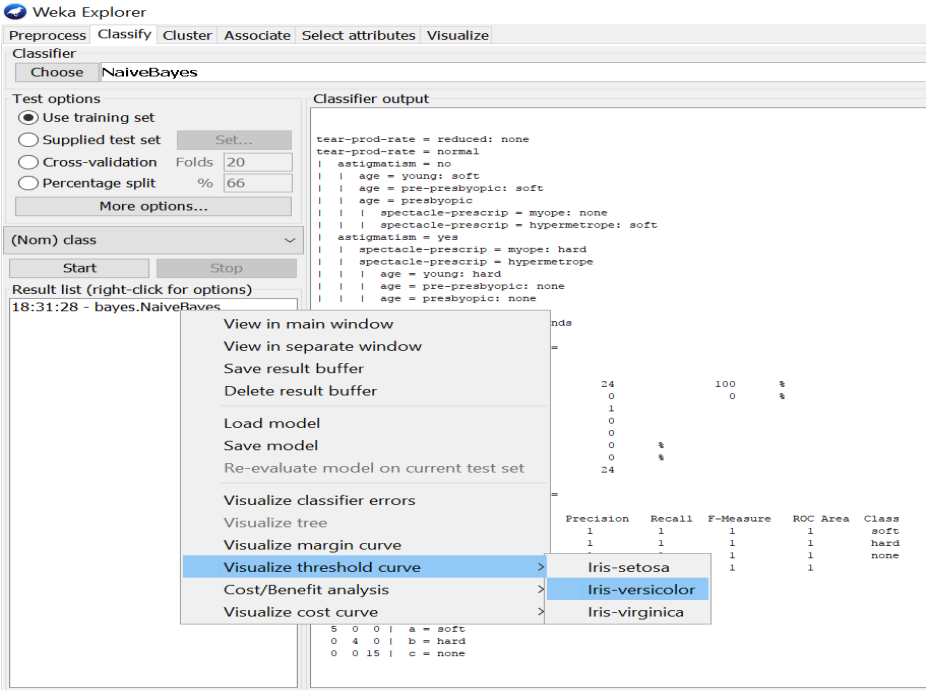
Aim: Demonstrate plotting multiple ROC curves in the same plot window by using j48 and Random forest tree

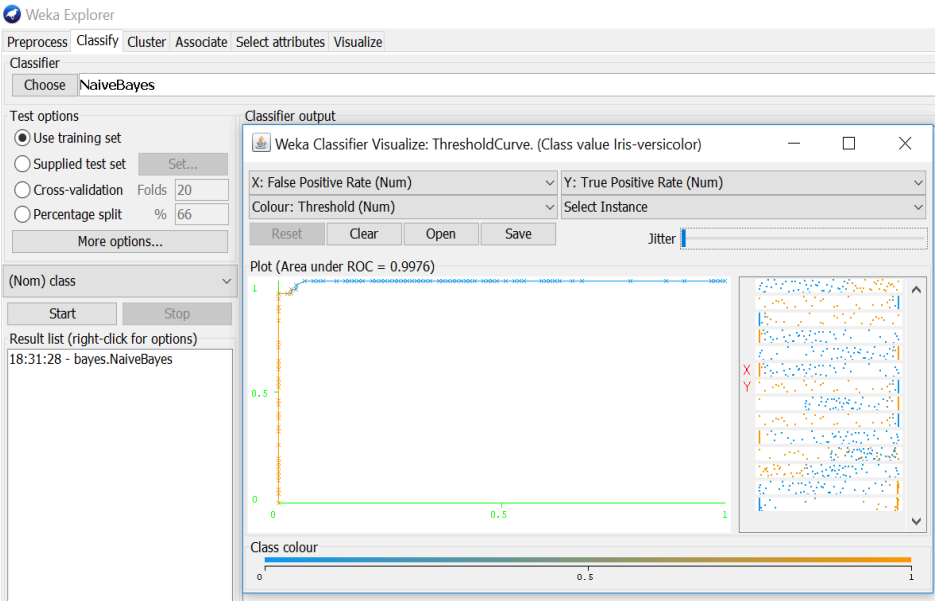
Plot ROC Curves

Procedure:

1. Load the dataset (Iris-2D.arff) into weka tool
2. Go to the classify option & in the left-hand navigation bar we can see different classification algorithms under the bayes section.
3. In which we selected Naïve-Bayes algorithm & click on start option with ―use training set‖ test option enabled.
4. Then we will get detailed accuracy by class consisting of F-measure, TP rate, FP rate, Precision, Recall values & Confusion Matrix.
5. For plotting RoC Curves, we need to right click on ―bayes.NaiveBayes for getting more options, In which we will select the ―Visualize Threshold Curve‖ & go to any class (Iris-setosa, Iris-versicolor, Iris- Virginica) as shown in below snapshot.

After selecting a class, RoC (Receiver Operating Characteristic) Curve plot will be displayed which has X- Axis –False Positive (FP) rate.


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
20
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
18:31:28 - bayes.NaiveBayes
Classifier output
tear-prod-rate = reduced: none
tear-prod-rate = normal
| astigmatism = no
| | age = young: soft
| | age = pre-presbyopic: soft
| | age = presbyopic
| | | spectacle-prescrip = myope: none
| | | spectacle-prescrip = hypermetrope: soft
| astigmatism = yes
| | spectacle-prescrip = myope: hard
| | spectacle-prescrip = hypermetrope
| | | age = young: hard
| | | age = pre-presbyopic: none
| | | age = presbyopic: none
Precision Recall F-Measure ROC Area Class
1 1 1 1 soft
1 1 1 1 hard
1 1 1 1 none
5 0 0 | a = soft
0 4 0 | b = hard
0 0 15 | c = none
View in main window
View in separate window
Save result buffer
Delete result buffer
Load model
Save model
Re-evaluate model on current test set
Visualize classifier errors
Visualize tree
Visualize margin curve
Visualize threshold curve
Cost/Benefit analysis
Visualize cost curve
Iris-setosa
Iris-versicolor
Iris-virginica

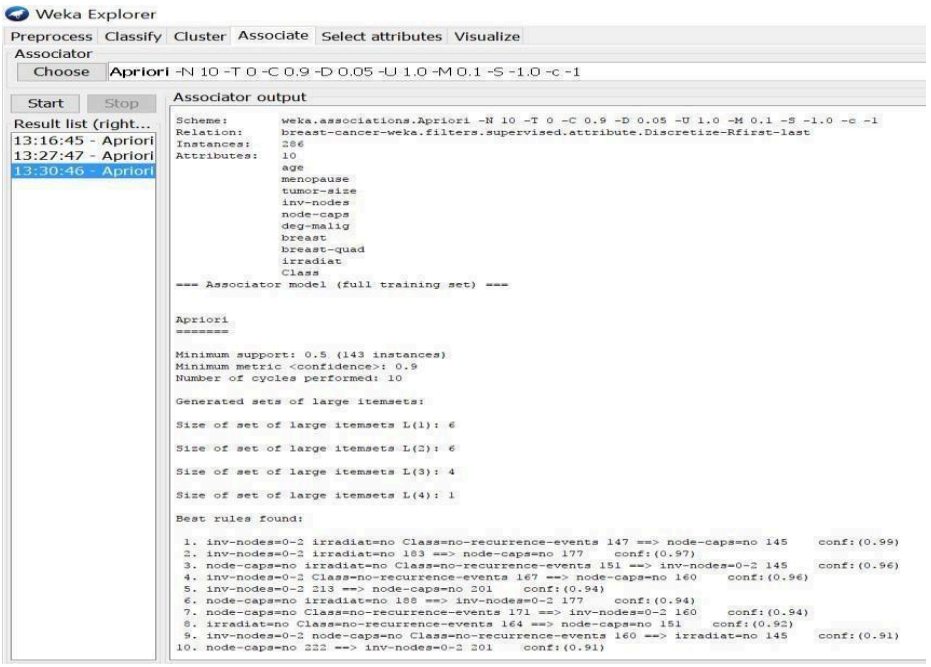

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
20
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
18:31:28 - bayes.NaiveBayes
Classifier output
Weka Classifier Visualize: ThresholdCurve. (Class value Iris-versicolor)
X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot (Area under ROC = 0.9976)
Class colour

Experiment 17

Aim: Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.

Procedure:

1. Load the dataset (Breast-Cancer.arff) into weka tool & select the discretize filter & apply it.
2. Go to the associate option & in the left-hand navigation bar we can see different association algorithms.
3. In which we can select A Priori algorithm & click on select option.
4. Below we can see the rules generated with different support & confidence values for that selected dataset.

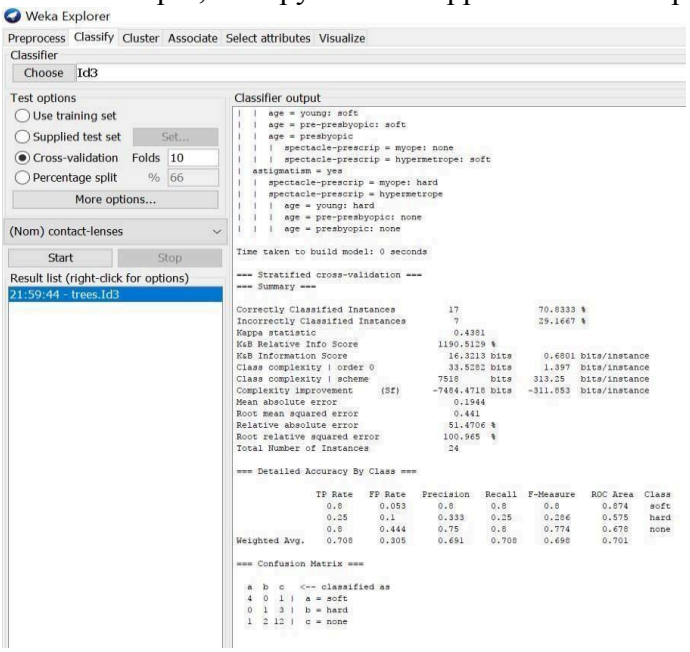


Load each dataset into Weka and run Id3, J48 classification algorithm. Study the classifier output. Compute entropy values, Kappa statistics.

Procedure for Id3:

1. Load the dataset (Contact-lenses.arff) into weka tool
2. Go to classify option & in the left-hand navigation bar we can see different classification algorithms under the tree section.
3. In which we selected the Id3 algorithm, in more options select the output entropy evaluation measures & click on start option.

4. Then we will get classifier output, entropy values Kappa Statistic as represented below.



5. In the above screenshot, we can run classifiers with different test options (Cross-validation, Use Training Set, Percentage Split, Supplied Test set).

The result of applying the chosen classifier will be tested according to the options that are set by clicking in the Test options box

There are four test modes:

A. Use training set: The classifier is evaluated on how well it predicts the class of the instances it was trained on.

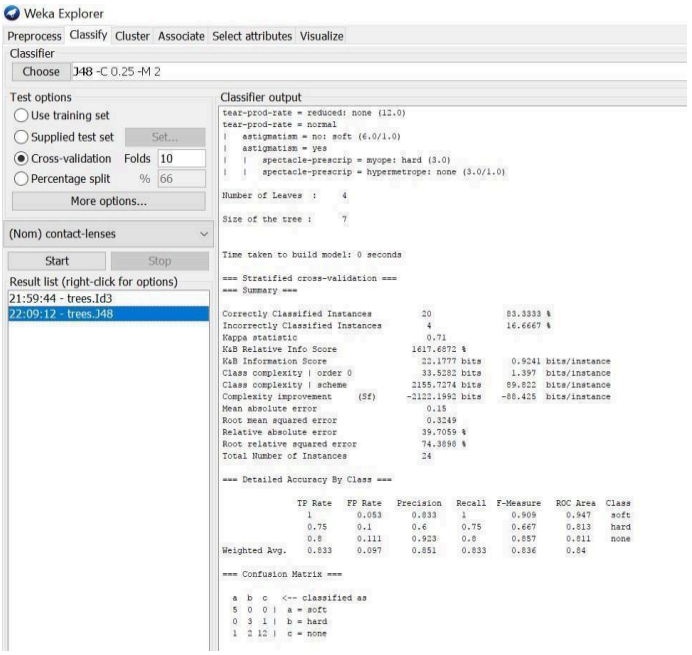
B. Supplied test set: The classifier is evaluated on how well it predicts the class of a set of instances loaded from a file. Clicking the Set... button brings up a dialog allowing you to choose the file to test on.

C. Cross-validation: The classifier is evaluated by cross-validation, using the number of folds that are entered in the Folds text field.

D. Percentage split: The classifier is evaluated on how well it predicts a certain percentage of the data which is held out for testing. The amount of data held out depends on the value entered in the % field.

Procedure for J48:

1. Load the dataset (Contact-lenses.arff) into weka tool
2. Go to classify option & in the left-hand navigation bar we can see different classification algorithms under the tree section.
3. In which we selected the J48 algorithm, in more options select the output entropy evaluation measures & click on start option.
4. Then we will get classifier output, entropy values & Kappa Statistic as represented below.
5. In the below screenshot, we can run classifiers with different test options (Cross-validation, Use Training Set, Percentage Split, Supplied Test set).

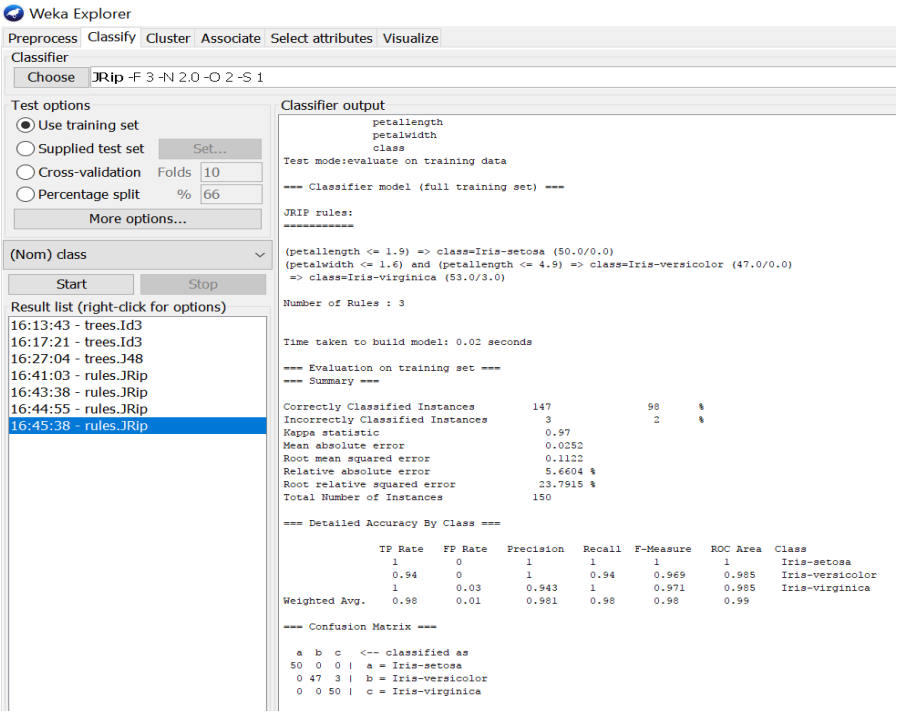


Experiment 18

Extract if-then rules from the decision tree generated by the classifier, Observe the confusion matrix and derive Accuracy, F-measure, TP rate, FP rate, Precision and Recall values. Apply
cross-validation strategy with various fold levels and compare the accuracy results.

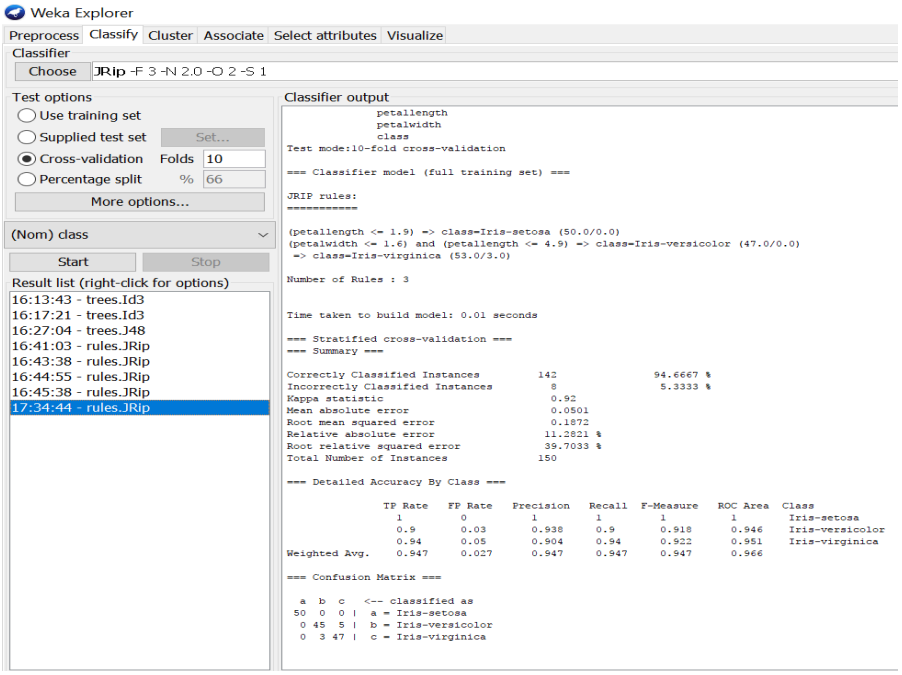
Procedure:

1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to classify option & in the left-hand navigation bar we can see different classification algorithms under the rules section.
3. In which we selected JRip (If-then) algorithm & click on start option with ―use training set‖ test option enabled.
4. Then we will get detailed accuracy by class consisting of F-measure, TP rate, FP rate, Precision, Recall values & Confusion Matrix as represented below.



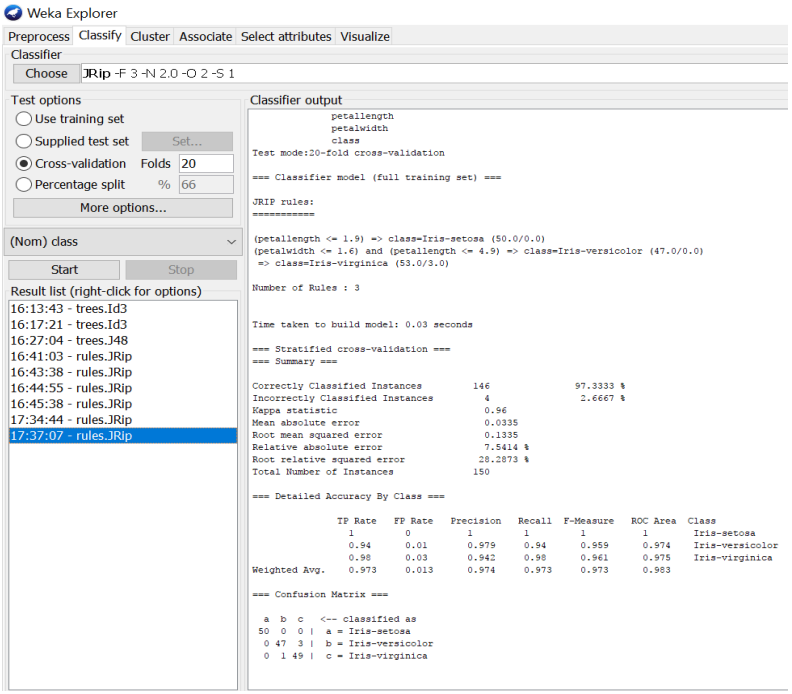
Using Cross-Validation Strategy with 10 folds:

Here, we enabled cross-validation test option with 10 folds & clicked start button as represented below



Using Cross-Validation Strategy with 20 folds:

Here, we enabled the cross-validation test option with 20 folds & clicked the start button as represented below.



Output:

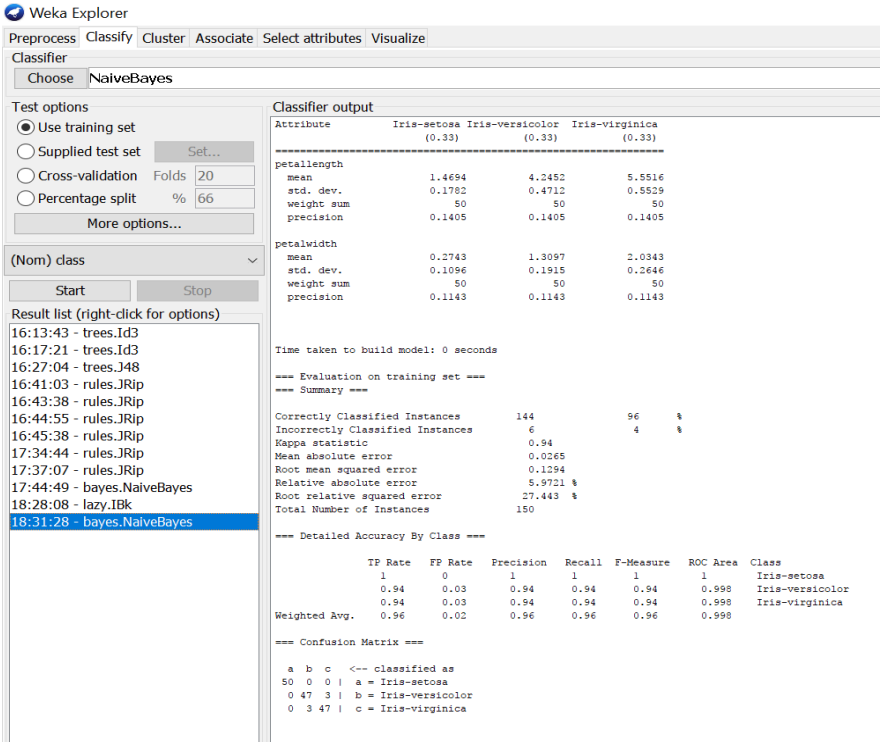
If we see the above results of cross validation with 10 folds & 20 folds. As per our observation the error rate is lesser with 20 folds having 97.3% correctness when compared to 10 folds with 94.6% correctness.

Experiment 19

Aim: Load each dataset into Weka and perform Naive-bayes classification and k-Nearest Neighbour classification. Interpret the results obtained.

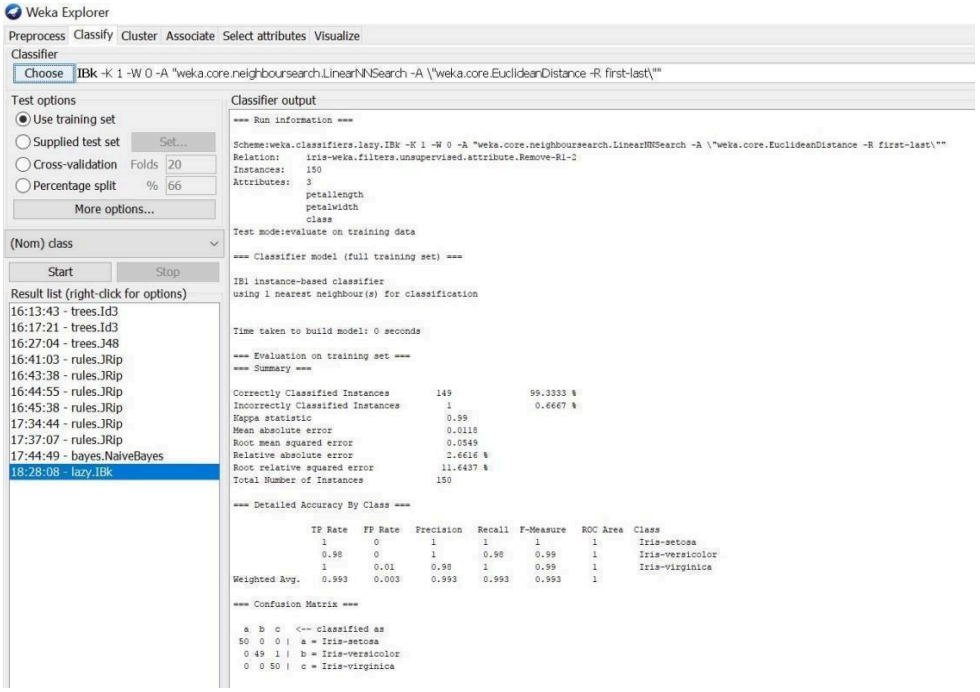
Procedure for Naïve-Bayes:

1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to the classify option & in the left-hand navigation bar we can see different classification algorithms under the bayes section.
3. In which we selected Naïve-Bayes algorithm & click on start option with —use training set‖ test option enabled.
4. Then we will get detailed accuracy by class consisting of F-measure, TP rate, FP rate, Precision,Recall values & Confusion Matrix as represented below.



Procedure for K-Nearest Neighbour (IBK):

1. Load the dataset (Iris-2D. arff) into weka tool
2. Go to classify option & in the left-hand navigation bar we can see different classification algorithms under the lazy section.
3. In which we selected K-Nearest Neighbour (IBK) algorithm & click on start option with —use train- ing set‖ test option enabled.
4. Then we will get detailed accuracy by class consisting of F-measure, TP rate, FP rate, Precision,Recall values & Confusion Matrix as represented below.


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
20
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
16:13:43 - trees.Id3
16:17:21 - trees.Id3
16:27:04 - trees.J48
16:41:03 - rules.JRip
16:43:38 - rules.JRip
16:44:55 - rules.JRip
16:45:38 - rules.JRip
17:34:44 - rules.JRip
17:37:07 - rules.JRip
17:44:49 - bayes.NaiveBayes
18:28:08 - lazy.IBk
Classifier output
=== Run information ===
Scheme:weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Relation: iris-weka.filters.unsupervised.attribute.Remove-R1-2
Instances: 150
Attributes: 3
petallength
petalwidth
class
Test mode:evaluate on training data
=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification
Time taken to build model: 0 seconds
=== Evaluation on training set ===
=== Summary ===
Correctly Classified Instances 149 99.3333 %
Incorrectly Classified Instances 1 0.6667 %
Kappa statistic 0.99
Mean absolute error 0.0118
Root mean squared error 0.0549
Relative absolute error 2.6616 %
Root relative squared error 11.6437 %
Total Number of Instances 150
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure ROC Area Class
1 0 1 1 1 1 Iris-setosa
0.98 0 1 0.98 0.99 1 Iris-versicolor
1 0.01 0.98 1 0.99 1 Iris-virginica
Weighted Avg. 0.993 0.003 0.993 0.993 0.993 1
=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 49 1 | b = Iris-versicolor
0 0 50 | c = Iris-virginica

Experiment 20

Aim: Demonstrate performing clustering on data

sets K-Mean

DB Scan

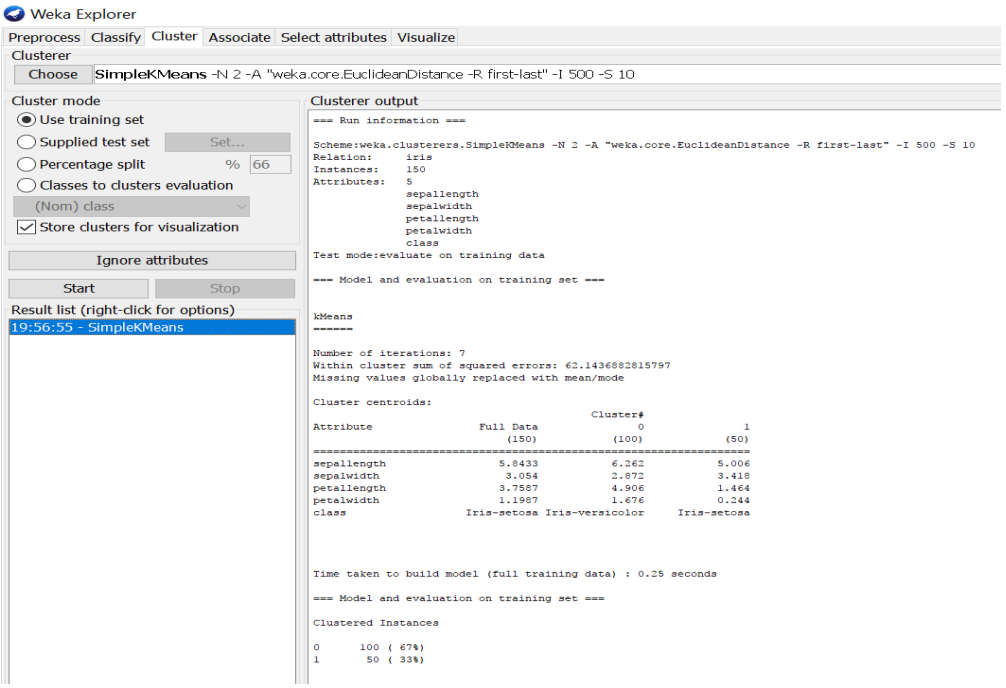
Hierarchical Clustering

Load each dataset into Weka and run a simple k-means clustering algorithm with different values of k (number of desired clusters). Study the clusters formed. Observe the sum of squared errors and centroids, and derive insights.

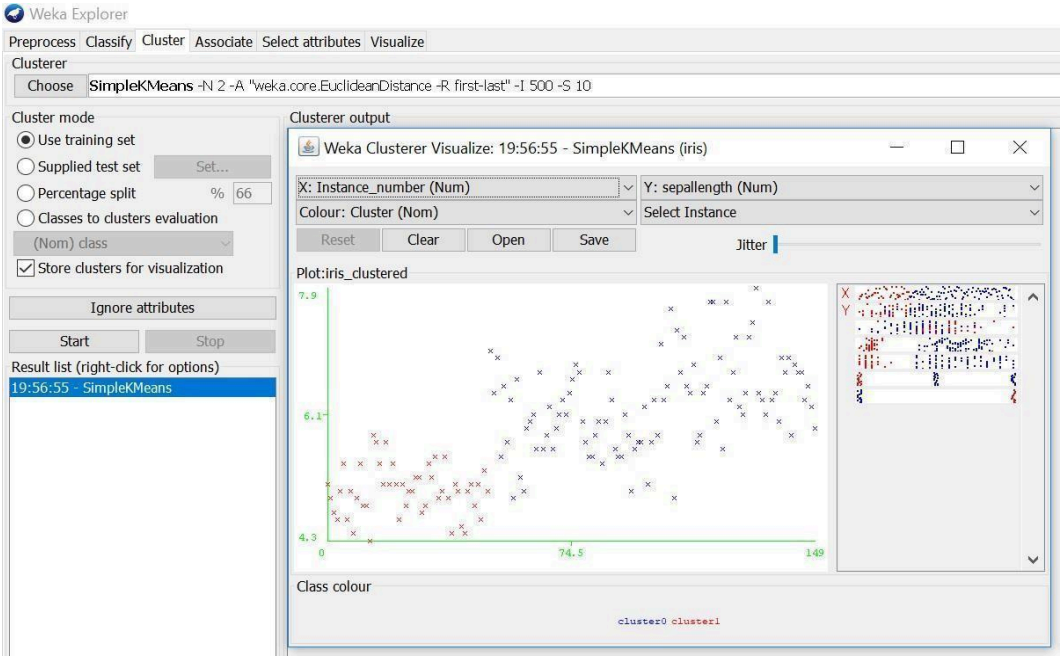
There are 12 clustering algorithms available in the Weka tool.

Procedure:

1. Load the dataset (Iris.arff) into Weka tool
2. Go to the classify option & in the left-hand navigation bar we can see different clustering algorithms under the lazy section.
3. In which we selected Simple K-Means algorithm & click on start option with ―use training set‖ test option enabled.
4. Then we will get the sum of squared errors, centroids, No. of iterations & clustered instances as represented below.

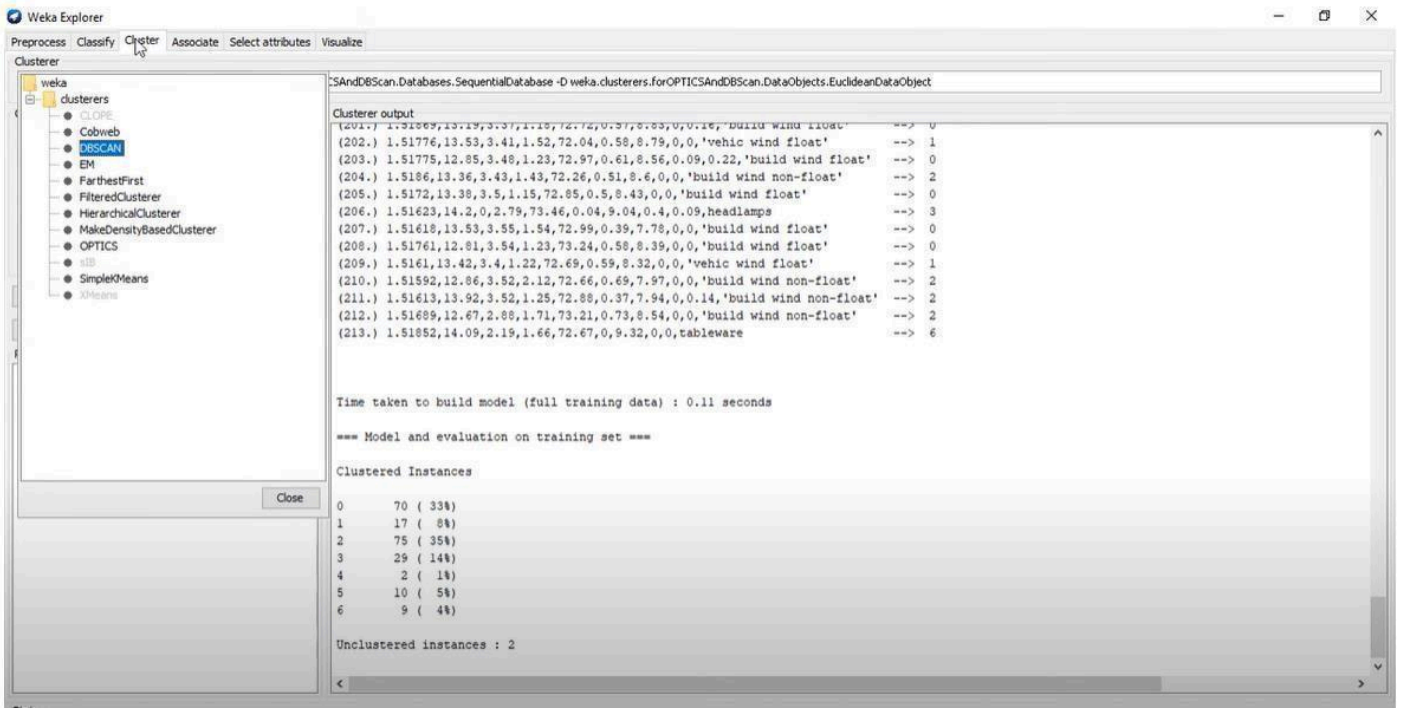


5. If we right click on simple k means, we will get more option in which ―Visualize cluster assignments‖ should be selected for getting cluster visualization as shown below.



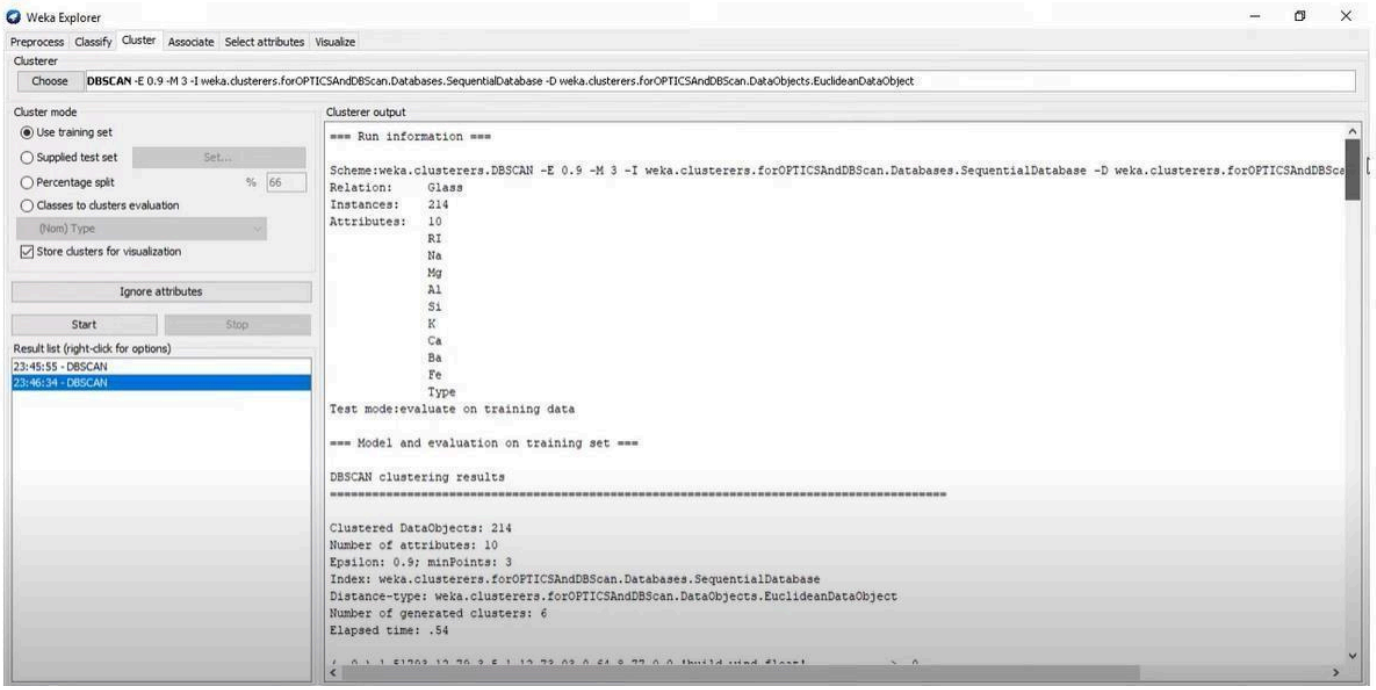
DB Scan Clustering:

1. Load the dataset: Open Weka and select the "Explorer" tab. Click on "Open File" and select the dataset you want to load. Weka supports several file formats such as ARFF, CSV, and TEXT.
2. Choose the DBSCAN algorithm: In the "Cluster" panel, select the "DBSCAN" algorithm from the list of available clustering algorithms.



3. Set the parameters: Click on the "DBSCAN" algorithm to open the parameter settings. You will need to set the following parameters:

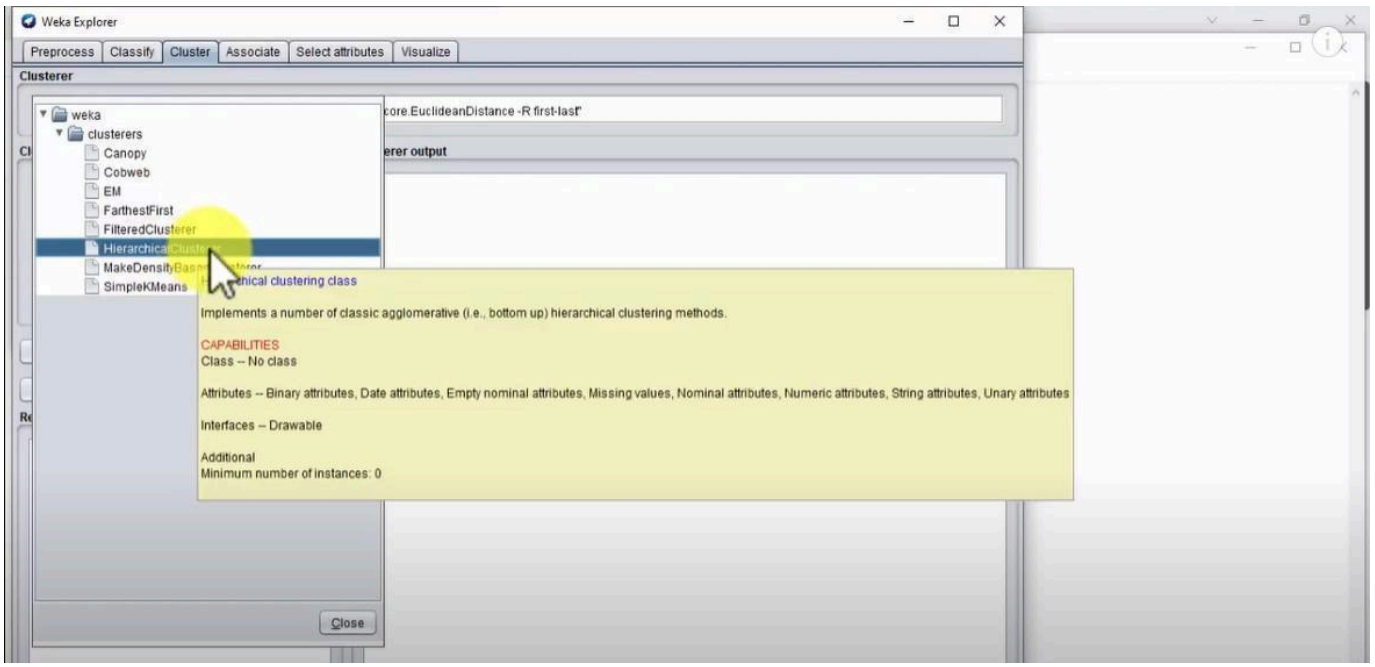
4. Epsilon: The radius of the neighborhood around each point. Any point within this radius is considered a part of the same cluster.
5. Min Pts: The minimum number of points required to form a dense region (i.e., a cluster). Points that do not meet this criterion are considered noise.
6. Run the algorithm: Click on the "Start" button to run the DBSCAN algorithm on the loaded dataset. Weka will output the resulting clusters and noise points.



Note: Before running the DBSCAN algorithm, you may want to pre-process the dataset to remove missing values, normalize the data, or reduce the dimensionality using techniques such as Principal Component Analysis (PCA). Weka provides several pre-processing tools that you can use before running the clustering algorithm**.**

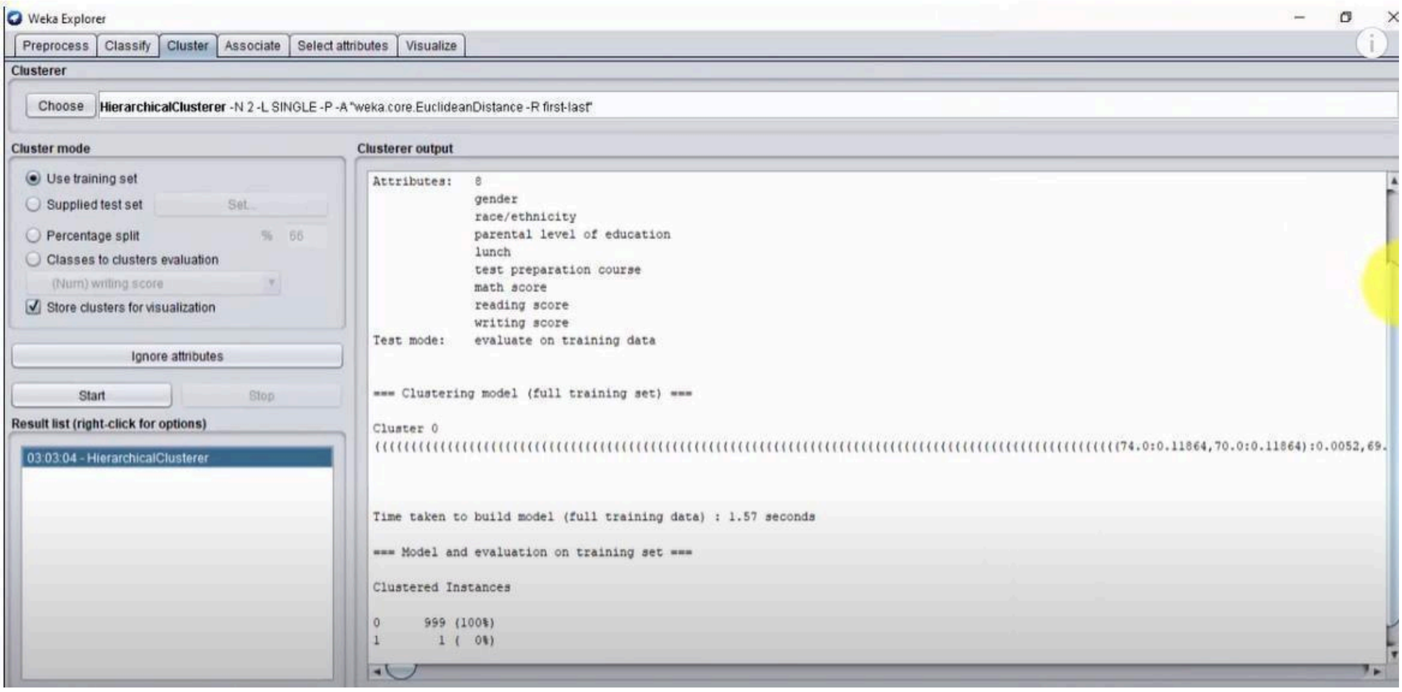
Hierarchical Clustering:

1. Load the dataset you want to cluster into Weka by going to "Explorer" -> "Preprocess" -> "Open file."



2. Choose the dataset you want to cluster and click "Open."

3. Select the "Cluster" tab in the top menu.
4. Select "Hierarchical Clusterer" from the list of clustering algorithms.
5. Set the parameters for the clustering algorithm by clicking on the "Options" button.
6. Choose the type of linkage you want to use (single, complete, average, etc.) and the distance measure (Euclidean, Manhattan, etc.).



7. Click "OK" to save the changes.
8. Click "Start" to run the clustering algorithm.
9. Weka will output the results of the clustering algorithm in a new window. You can explore the dendrogram to see the clusters that were formed and the instances that belong to each cluster.

Note that these are general steps, and the specific process may vary depending on the version of Weka you are using. Additionally, it's important to pre-process the data before clustering to ensure the best results.