COCSC16 - DATA MINING PRACTICAL FILE

Amogh Garg – 2020UCO1688

Netaji Subhas University of Technology, New Delhi

INDEX

EXPERIMENT-1

AIM: Exploring and getting started with WEKA.

STEPS AND OUTPUT:

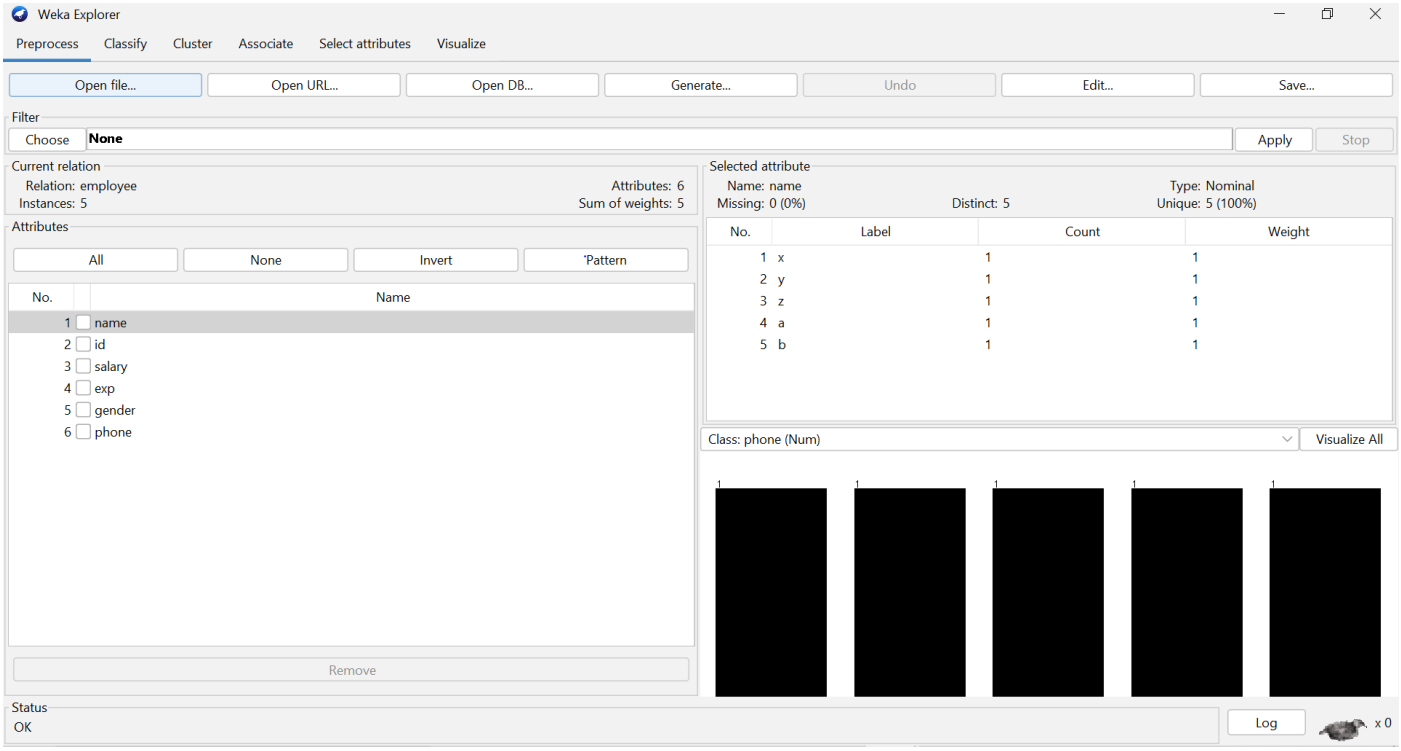
Steps:

1) Open Start → Programs → Accessories → Notepad
2) Type the following training data set with the help of Notepad for Employee Table.

```
@relation employee
@attribute name {x,y,z,a,b}
@attribute id numeric
@attribute salary {low,medium,high}
@attribute exp numeric
@attribute gender {male,female}
@attribute phone numeric

@data
x,101,low,2,male,250311
y,102,high,3,female,251665
z,103,medium,1,male,240238
a,104,low,5,female,200200
b,105,high,2,male,240240
```

3) After that the file is saved with **.arff** file format.
4) Minimize the arff file and then open Start → Programs → weka-3-4.
5) Click on **weka-3-4**, then Weka dialog box is displayed on the screen.
6) In that dialog box there are four modes, click on **explorer**.
7) Explorer shows many options. In that click on **'open file'** and select the arff file



EXPERIMENT-2

AIM: Implement the following on WEKA using various filters:

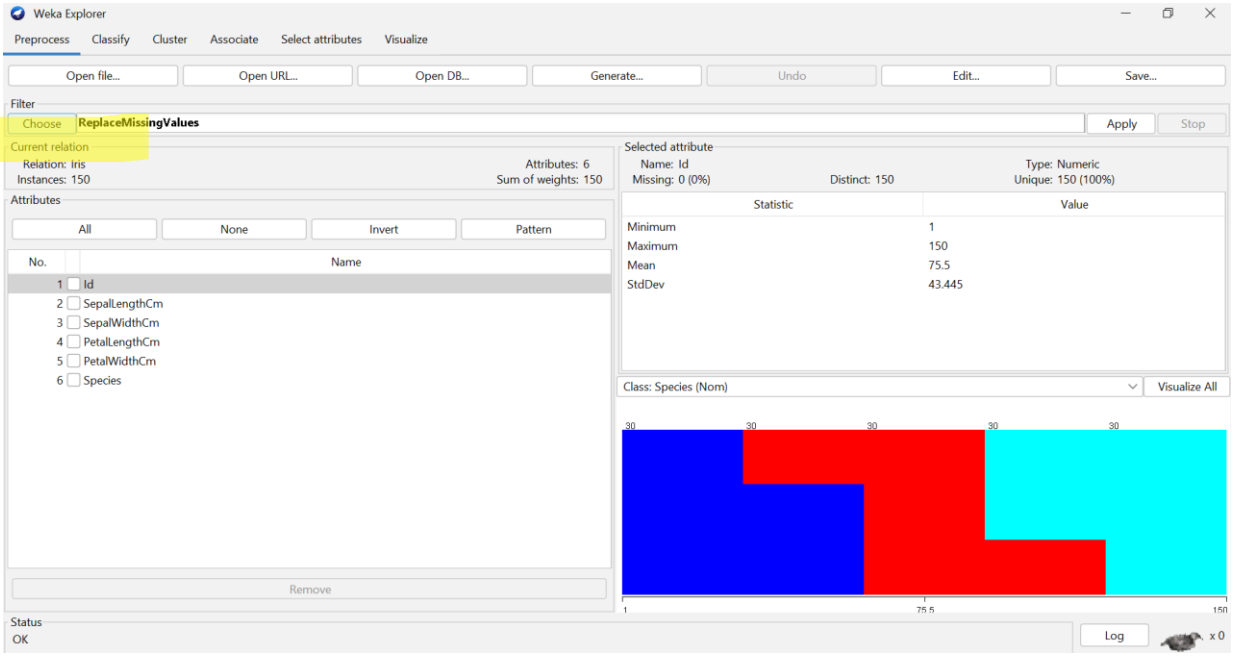
1. Data Cleaning
2. Missing Values Handling
3. Normalisation

STEPS AND OUTPUT:

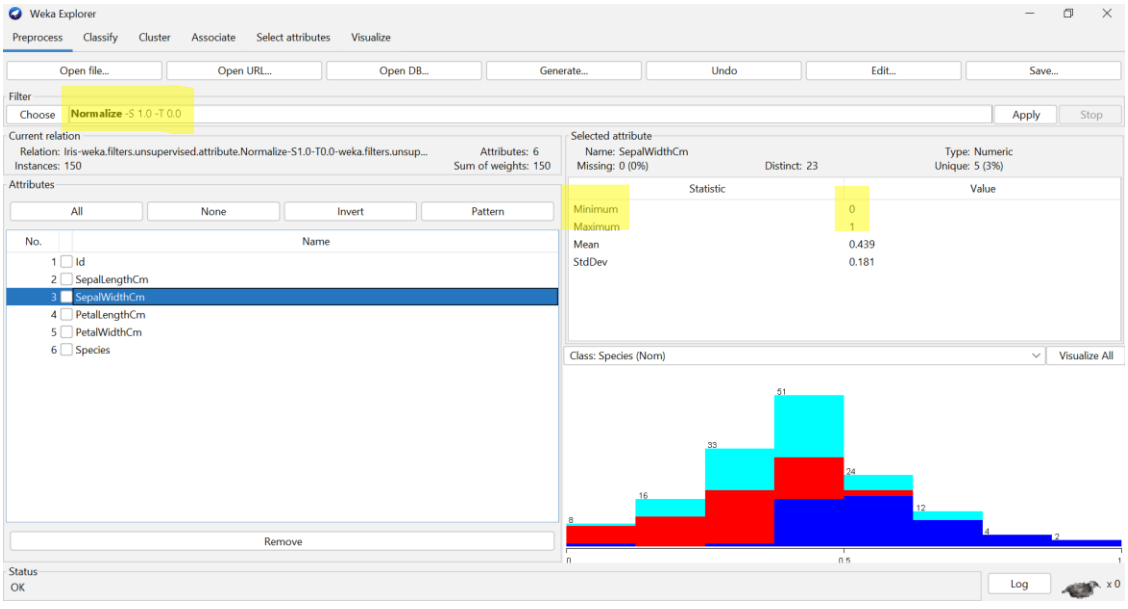
1. Load the IRIS dataset on WEKA and open the “Pre-process” tab.

2. The attributes which are not useful can be removed by selecting the attribute and clicking on the “remove” option.

3. Click the “Choose” button for the filter and select “ReplaceMissingValues”, it us under unsupervised-> attribute-> ReplaceMissing-> Values and then select “Apply”.



4. Click the “Choose” button for the filter and select “Normalise”, it us under unsupervised-> attribute-> Normalise and then select “Apply”.

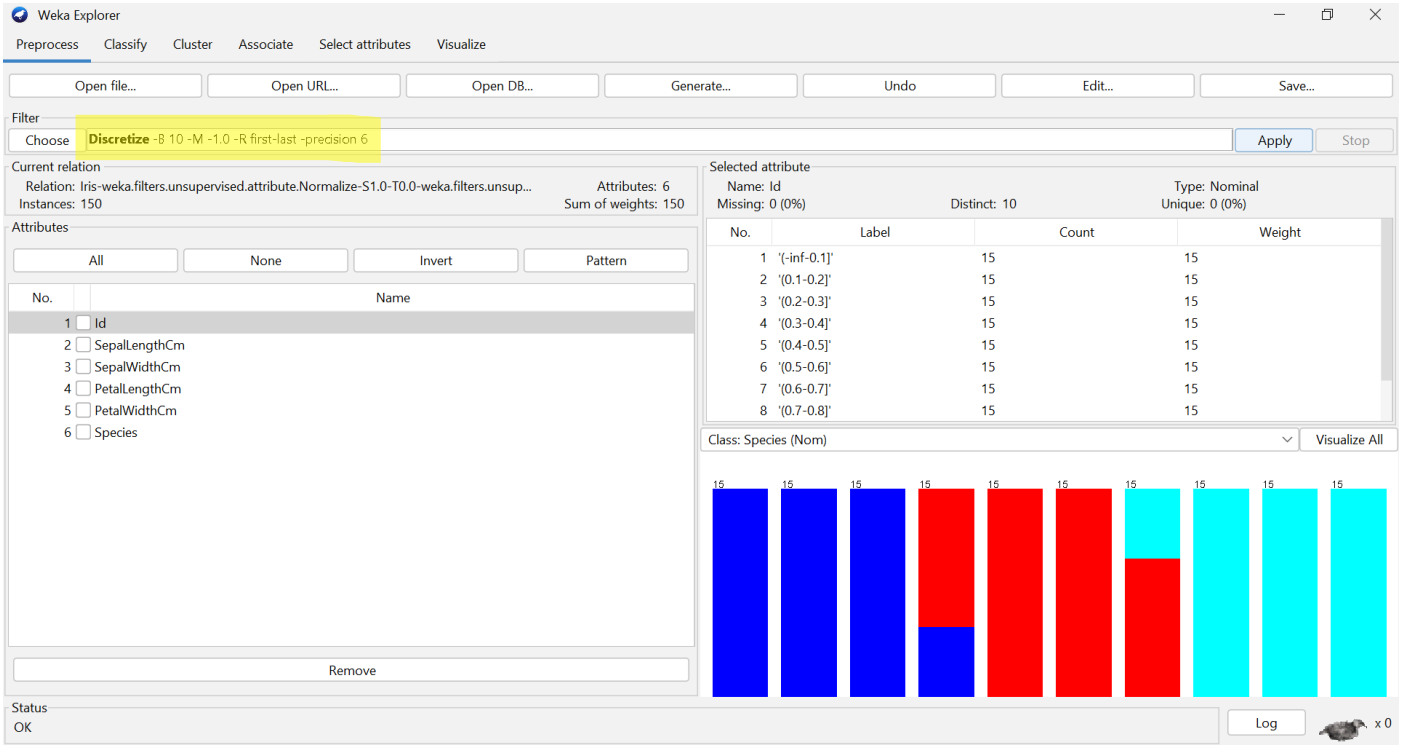


EXPERIMENT-3

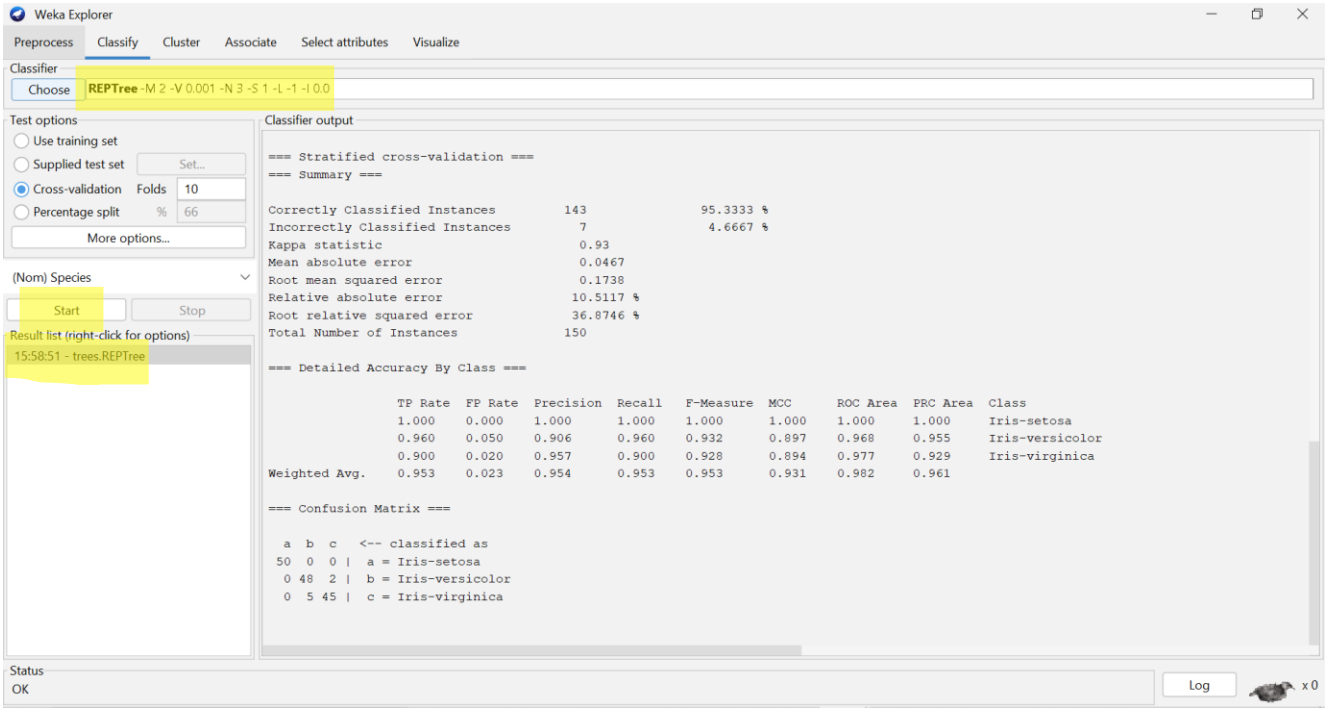
AIM: On a dataset first do normalisation and then discretisation, then fit a decision tree model for classification.

STEPS AND OUTPUT:

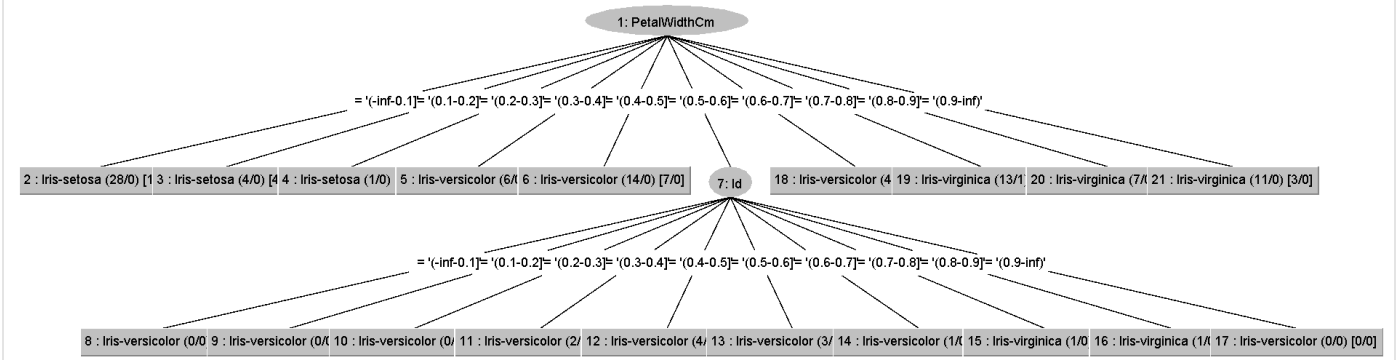
1. Follow the steps of experiment-2 till normalisation on the IRIS dataset.
2. Click the “Choose” button for the filter and select “Discretise”, it us under unsupervised-> attribute-> Discretise and then select “Apply”.



3. To plot a decision-tree go on to the “Classify tab”.
4. Click the “Choose” button and select “REPTree”, it us under tree -> REPTree and then select “Apply”. Click on the Start button and then visualise the tree by right clicking on the model.



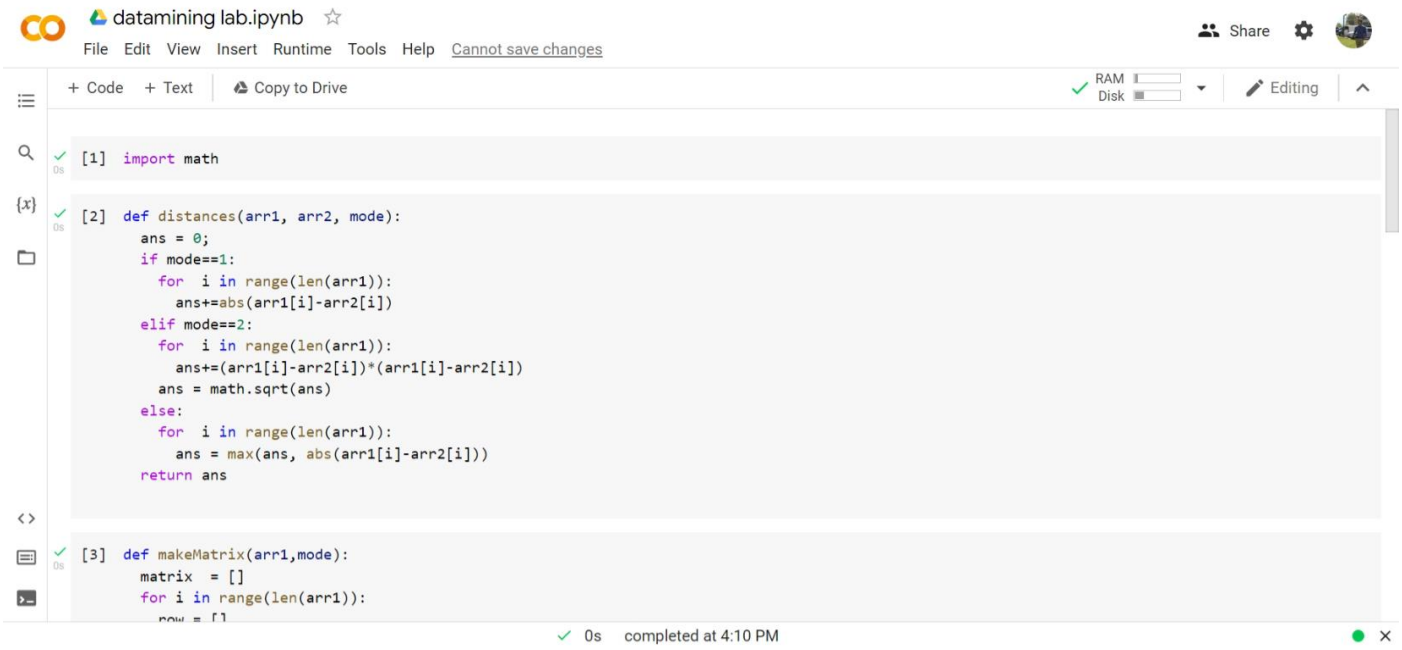
Tree View

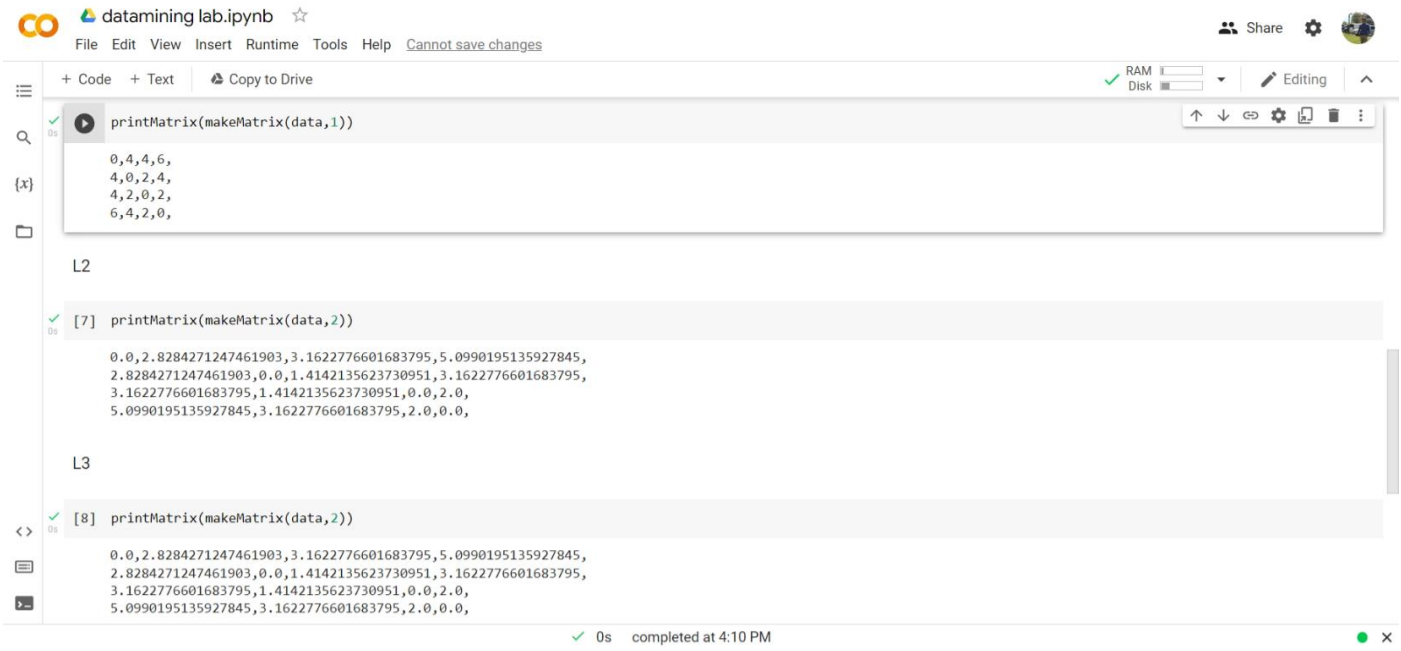

1: PetalWidthCm
= '(-inf-0.1]'= '(0.1-0.2]'= '(0.2-0.3]'= '(0.3-0.4]'= '(0.4-0.5]'= '(0.5-0.6]'= '(0.6-0.7]'= '(0.7-0.8]'= '(0.8-0.9]'= '(0.9-inf)'
2 : Iris-setosa (28/0) [1
3 : Iris-setosa (4/0) [4
4 : Iris-setosa (1/0)
5 : Iris-versicolor (6/
6 : Iris-versicolor (14/0) [7/0]
7: Id
18 : Iris-versicolor (4
19 : Iris-virginica (13/1
20 : Iris-virginica (7/
21 : Iris-virginica (11/0) [3/0]
= '(-inf-0.1]'= '(0.1-0.2]'= '(0.2-0.3]'= '(0.3-0.4]'= '(0.4-0.5]'= '(0.5-0.6]'= '(0.6-0.7]'= '(0.7-0.8]'= '(0.8-0.9]'= '(0.9-inf)'
8 : Iris-versicolor (0/0
9 : Iris-versicolor (0/
10 : Iris-versicolor (0/
11 : Iris-versicolor (2/
12 : Iris-versicolor (4/
13 : Iris-versicolor (3/
14 : Iris-versicolor (1/
15 : Iris-virginica (1/0
16 : Iris-virginica (1/
17 : Iris-versicolor (0/0) [0/0]

EXPERIMENT-4

AIM: Make a dataset of about 10 rows having various types of attributes (nominal, ordinal, interval, ratio), then calculate the dissimilarity matrix for it. You can implement this in any programming language of your choice.

CODE AND OUTPUT:




datamining lab.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM
Disk
Editing
printMatrix(makeMatrix(data,1))
0,4,4,6,
4,0,2,4,
4,2,0,2,
6,4,2,0,

L2

[7] printMatrix(makeMatrix(data,2))
0.0,2.8284271247461903,3.1622776601683795,5.0990195135927845,
2.8284271247461903,0.0,1.4142135623730951,3.1622776601683795,
3.1622776601683795,1.4142135623730951,0.0,2.0,
5.0990195135927845,3.1622776601683795,2.0,0.0,

L3

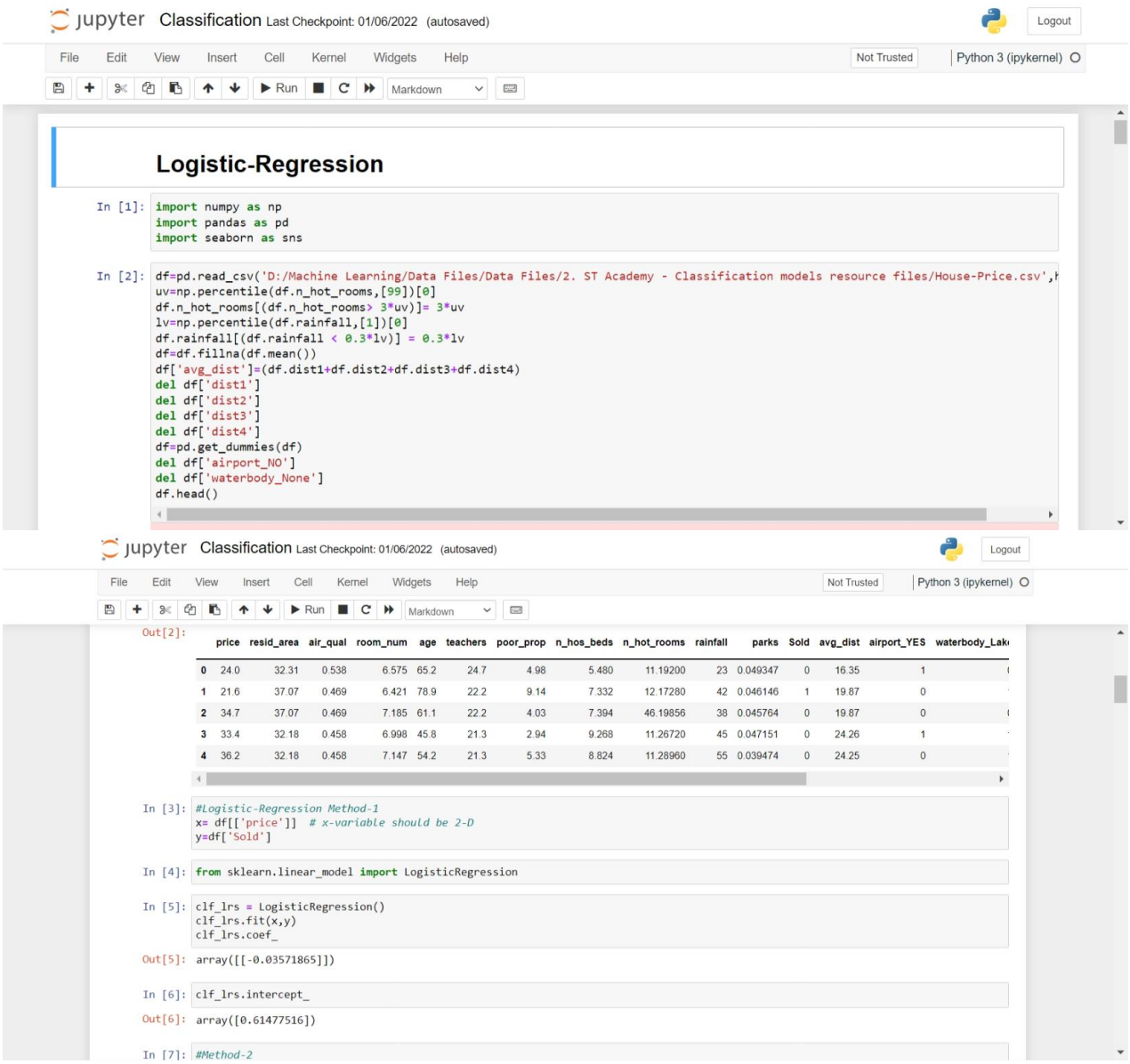
[8] printMatrix(makeMatrix(data,2))
0.0,2.8284271247461903,3.1622776601683795,5.0990195135927845,
2.8284271247461903,0.0,1.4142135623730951,3.1622776601683795,
3.1622776601683795,1.4142135623730951,0.0,2.0,
5.0990195135927845,3.1622776601683795,2.0,0.0,

0s completed at 4:10 PM

EXPERIMENT-5

AIM: Implement regression (linear, logistic, any one) in python on any dataset of your choice.

CODE AND OUTPUT:



jupyter Classification Last Checkpoint: 01/06/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run | Markdown

Logistic-Regression

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

In [2]:
```
df=pd.read_csv('D:/Machine Learning/Data Files/Data Files/2. ST Academy - Classification models resource files/House-Price.csv',
uv=np.percentile(df.n_hot_rooms,[99])[0]
df.n_hot_rooms[(df.n_hot_rooms> 3*uv)]= 3*uv
lv=np.percentile(df.rainfall,[1])[0]
df.rainfall[(df.rainfall < 0.3*lv)] = 0.3*lv
df=df.fillna(df.mean())
df['avg_dist']=(df.dist1+df.dist2+df.dist3+df.dist4)
del df['dist1']
del df['dist2']
del df['dist3']
del df['dist4']
df=pd.get_dummies(df)
del df['airport_NO']
del df['waterbody_None']
df.head()
```


jupyter Classification Last Checkpoint: 01/06/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run | Markdown

Out[2]:

	price	resid_area	air_qual	room_num	age	teachers	poor_prop	n_hos_beds	n_hot_rooms	rainfall	parks	Sold	avg_dist	airport_YES	waterbody_Lak
0	24.0	32.31	0.538	6.575	65.2	24.7	4.98	5.480	11.19200	23	0.049347	0	16.35	1	
1	21.6	37.07	0.469	6.421	78.9	22.2	9.14	7.332	12.17280	42	0.046146	1	19.87	0	
2	34.7	37.07	0.469	7.185	61.1	22.2	4.03	7.394	46.19856	38	0.045764	0	19.87	0	
3	33.4	32.18	0.458	6.998	45.8	21.3	2.94	9.268	11.26720	45	0.047151	0	24.26	1	
4	36.2	32.18	0.458	7.147	54.2	21.3	5.33	8.824	11.28960	55	0.039474	0	24.25	0	

In [3]:
```
#Logistic-Regression Method-1
x= df[['price']]  # x-variable should be 2-D
y=df['Sold']
```

In [4]:
```
from sklearn.linear_model import LogisticRegression
```

In [5]:
```
clf_lrs = LogisticRegression()
clf_lrs.fit(x,y)
clf_lrs.coef_
```

Out[5]:
```
array([[-0.03571865]])
```

In [6]:
```
clf_lrs.intercept_
```

Out[6]:
```
array([0.61477516])
```

In [7]:
```
#Method-2
```

jupyter **Classification** Last Checkpoint: 01/06/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown

In [7]:
```
#Method-2
import statsmodels.api as sn
```

In [8]:
```
x_cons=sn.add_constant(x)
x_cons.head()
```

Out[8]:

	const	price
0	1.0	24.0
1	1.0	21.6
2	1.0	34.7
3	1.0	33.4
4	1.0	36.2

In [9]:
```
import statsmodels.discrete.discrete_model as sm
```

In [10]:
```
logit = sm.Logit(y,x_cons).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.676690
         Iterations 5
```

In [11]:
```
logit.summary()
```

Out[11]:

Logit Regression Results

Dep. Variable:	Sold	**No. Observations:**	506

jupyter **Classification** Last Checkpoint: 01/06/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

Run | Markdown

In [9]:
```
import statsmodels.discrete.discrete_model as sm
```

In [10]:
```
logit = sm.Logit(y,x_cons).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.676690
         Iterations 5
```

In [11]:
```
logit.summary()
```

Out[11]:

Logit Regression Results

Dep. Variable:	Sold	**No. Observations:**	506
Model:	Logit	**Df Residuals:**	504
Method:	MLE	**Df Model:**	1
Date:	Fri, 28 May 2021	**Pseudo R-squ.:**	0.01788
Time:	17:22:31	**Log-Likelihood:**	-342.41
converged:	True	**LL-Null:**	-348.64
Covariance Type:	nonrobust	**LLR p-value:**	0.0004142

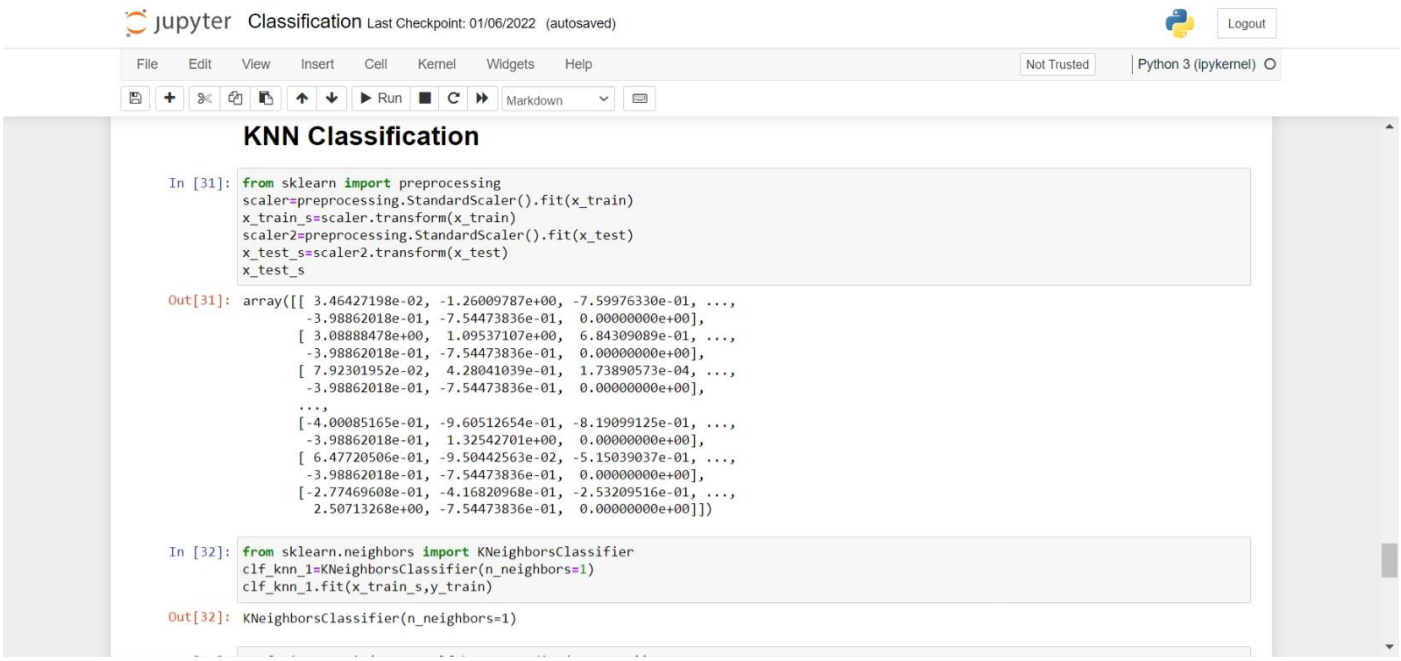
	coef	std err	z	P>|z|	[0.025	0.975]
const	0.6149	0.248	2.484	0.013	0.130	1.100
price	-0.0357	0.010	-3.417	0.001	-0.056	-0.015

EXPERIMENT-6

AIM: Implement KNN (K Nearest Neighbours) in python on any dataset of your choice.

CODE AND OUTPUT:

Dataset and pre-processing are same as in experiment-5.



jupyter Classification Last Checkpoint: 01/06/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)

Run Markdown

Out[35]: 0.5392156862745098

In [36]:
```
#Create single model for various values of k
from sklearn.model_selection import GridSearchCV
```

In [37]:
```
params={'n_neighbors': [1,2,3,4,5,6,7,8,9,10]}
grid_search_cv=GridSearchCV(KNeighborsClassifier(),params)
grid_search_cv.fit(x_train_s,y_train)
```

Out[37]:
```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]})
```

In [38]:
```
#Value of best parameter
grid_search_cv.best_params_
```

Out[38]:
```
{'n_neighbors': 9}
```

In [39]:
```
optimised_KNN=grid_search_cv.best_estimator_
```

In [40]:
```
y_test_pred=optimised_KNN.predict(x_test_s)
confusion_matrix(y_test,y_test_pred)
```

Out[40]:
```
array([[36, 22],
       [18, 26]], dtype=int64)
```

In [42]:
```
accuracy_score(y_test,y_test_pred)
```

Out[42]: 0.6078431372549019

EXPERIMENT-7

AIM: Implement OneR algorithm in python on any dataset of your choice.

CODE AND OUTPUT:

jupyter OneR Last Checkpoint: 10/11/2022 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
from mlxtend.data import iris_data
X, y = iris_data()
X[:15]
```

Out[1]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2]])
```

In [2]:
```
import numpy as np


def get_feature_quartiles(X):
    X_discretized = X.copy()
    for col in range(X.shape[1]):
        for q, class_label in zip([1.0, 0.75, 0.5, 0.25], [3, 2, 1, 0]):
            threshold = np.quantile(X[:, col], q=q)
            X_discretized[X[:, col] <= threshold, col] = class_label
    return X_discretized.astype(np.int)
```

jupyter OneR Last Checkpoint: 10/11/2022 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

Out[2]:
```
array([[0, 3, 0, 0],
       [0, 1, 0, 0],
       [0, 2, 0, 0],
       [0, 2, 0, 0],
       [0, 3, 0, 0],
       [1, 3, 1, 1],
       [0, 3, 0, 0],
       [0, 3, 0, 0],
       [0, 1, 0, 0],
       [0, 2, 0, 0],
       [1, 3, 0, 0],
       [0, 3, 0, 0],
       [0, 1, 0, 0],
       [0, 1, 0, 0],
       [1, 3, 0, 0]])
```

In [3]:
```
from sklearn.model_selection import train_test_split


Xd_train, Xd_test, y_train, y_test = train_test_split(Xd, y, random_state=0, stratify=y)
```

In [4]:
```
from mlxtend.classifier import OneRClassifier
oner = OneRClassifier()

oner.fit(Xd_train, y_train)
```

Out[4]:
```
OneRClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

jupyter OneR Last Checkpoint: 10/11/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Python 3 (ipykernel)

Run Code

In [4]:
```
from mlxtend.classifier import OneRClassifier
oner = OneRClassifier()

oner.fit(Xd_train, y_train)
```

Out[4]:
```
OneRClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [5]:
```
oner.feature_idx_
```

Out[5]:
```
2
```

In [6]:
```
y_pred = oner.predict(Xd_train)
train_acc = np.mean(y_pred == y_train)
print(f'Training accuracy {train_acc*100:.2f}%')
```

```
Training accuracy 85.71%
```

In [7]:
```
y_pred = oner.predict(Xd_test)
test_acc = np.mean(y_pred == y_test)
print(f'Test accuracy {test_acc*100:.2f}%')
```

```
Test accuracy 84.21%
```

In [8]:
```
test_acc = oner.score(Xd_test, y_test)
print(f'Test accuracy {test_acc*100:.2f}%')
```

```
Test accuracy 84.21%
```

EXPERIMENT-8

AIM: Implement decision-tree and visualise it in python.

CODE AND OUTPUT:

jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv("D:/Machine Learning/Data Files/Data Files/3. ST Academy - Decision Trees resource files/Movie_classification.csv"
df.head()
```

Out[2]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Critic_rating	Trailer_views	3I
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	7.94	527367	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	7.44	494055	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	7.44	547051	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	8.26	516279	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	8.26	531448	

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Marketing expense    506 non-null    float64
 1   Production expense   506 non-null    float64
```

jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [4]:
```
#Missing Value Imputation
df['Time_taken'].fillna(value=df['Time_taken'].mean(),inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Marketing expense    506 non-null    float64
 1   Production expense   506 non-null    float64
 2   Multiplex coverage   506 non-null    float64
 3   Budget               506 non-null    float64
 4   Movie_length         506 non-null    float64
 5   Lead_ Actor_Rating   506 non-null    float64
 6   Lead_Actress_rating  506 non-null    float64
 7   Director_rating      506 non-null    float64
 8   Producer_rating      506 non-null    float64
 9   Critic_rating        506 non-null    float64
 10  Trailer_views        506 non-null    int64
 11  3D_available         506 non-null    object
 12  Time_taken           506 non-null    float64
 13  Twitter_hastags      506 non-null    float64
 14  Genre                506 non-null    object
 15  Avg_age_actors       506 non-null    int64
 16  Num_multiplex        506 non-null    int64
 17  Collection           506 non-null    int64
 18  Start_Tech_Oscar     506 non-null    int64
dtypes: float64(12), int64(5), object(2)
memory usage: 75.2+ KB
```

jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [5]:
```
#Dummy Variable Creation
df=pd.get_dummies(df,columns=["3D_available","Genre"],drop_first=True)
df.head()
```

Out[5]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Critic_rating	...	Time_taken
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	7.94	...	109.60
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	7.44	...	146.64
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	7.44	...	147.88
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	8.26	...	185.36
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	8.26	...	176.48

5 rows × 21 columns

In [6]:
```
#X-Y Split
x=df.loc[:,df.columns != "Start_Tech_Oscar"]
x.head()
```

Out[6]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Critic_rating	Trailer_views	Ti
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	7.94	527367	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	7.44	494055	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	7.44	547051	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	8.26	516279	

jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [7]:
```
y=df["Start_Tech_Oscar"]
y.head()
```

Out[7]:
```
0    1
1    0
2    1
3    1
4    1
Name: Start_Tech_Oscar, dtype: int64
```

In [8]:
```
#Test-Train Split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [9]:
```
#Training Classification Tree
from sklearn import tree
clftree = tree.DecisionTreeClassifier(max_depth=3)
clftree.fit(x_train,y_train)
```

Out[9]:
```
DecisionTreeClassifier(max_depth=3)
```

In [10]:
```
#Predict Value using trained model
y_train_pred=clftree.predict(x_train)
y_test_pred=clftree.predict(x_test)
y_test_pred
```

Out[10]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
```

jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

Out[10]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0], dtype=int64)
```

In [11]:
```
#Model Performance
from sklearn.metrics import accuracy_score,confusion_matrix
confusion_matrix(y_train,y_train_pred)
```

Out[11]:
```
array([[172,  14],
       [126,  92]], dtype=int64)
```

In [12]:
```
confusion_matrix(y_test,y_test_pred)
```

Out[12]:
```
array([[39,  5],
       [41, 17]], dtype=int64)
```

In [13]:
```
accuracy_score(y_test,y_test_pred)
```

Out[13]:
```
0.5490196078431373
```

In [14]:
```
#Plotting Decision Tree
from IPython.display import Image
import pydotplus
dot_data=tree.export_graphviz(clftree,out_file=None,feature_names=x_train.columns,filled=True)
graph=pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

Logout

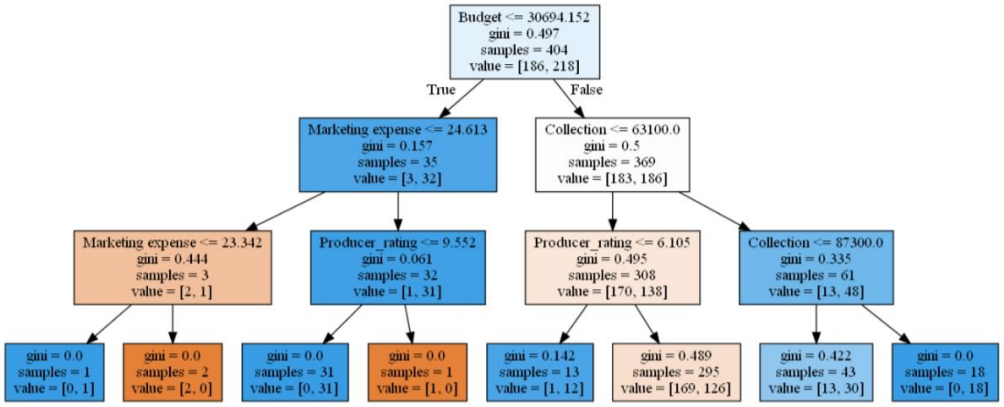
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [14]:
```
#Plotting Decision Tree
from IPython.display import Image
import pydotplus
dot_data=tree.export_graphviz(clftree,out_file=None,feature_names=x_train.columns,filled=True)
graph=pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Out[14]:



jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

Logout

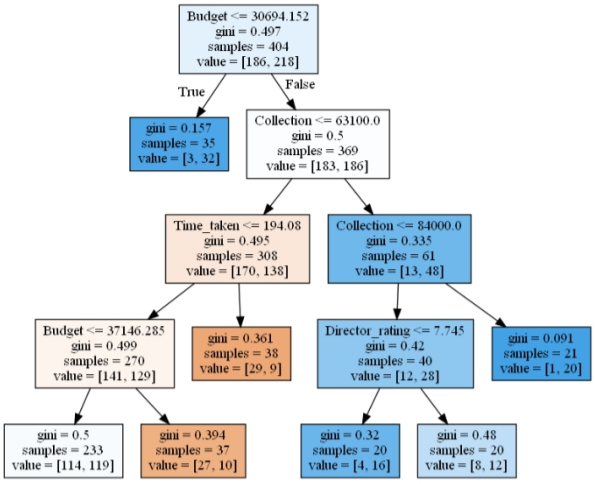
File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [15]:
```
#Controlling Tree Growth
clftree2=tree.DecisionTreeClassifier(min_samples_leaf=20,max_depth=4)
clftree2.fit(x_train,y_train)
dot_data=tree.export_graphviz(clftree2,out_file=None,feature_names=x_train.columns,filled=True)
graph2=pydotplus.graph_from_dot_data(dot_data)
Image(graph2.create_png())
```

Out[15]:



jupyter Classification-Tree Last Checkpoint: 06/03/2021 (autosaved)

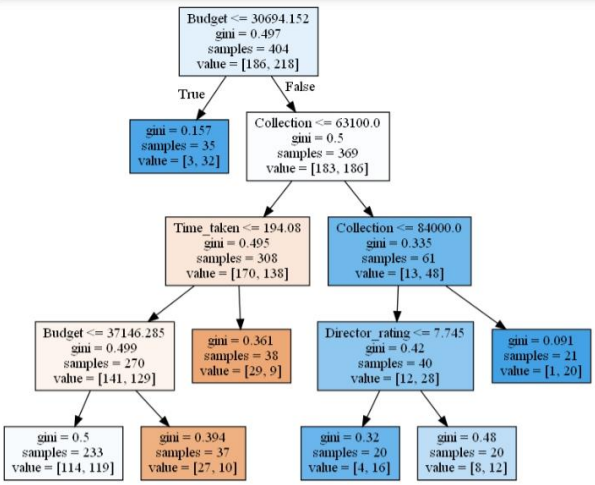
Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

Out[15]:



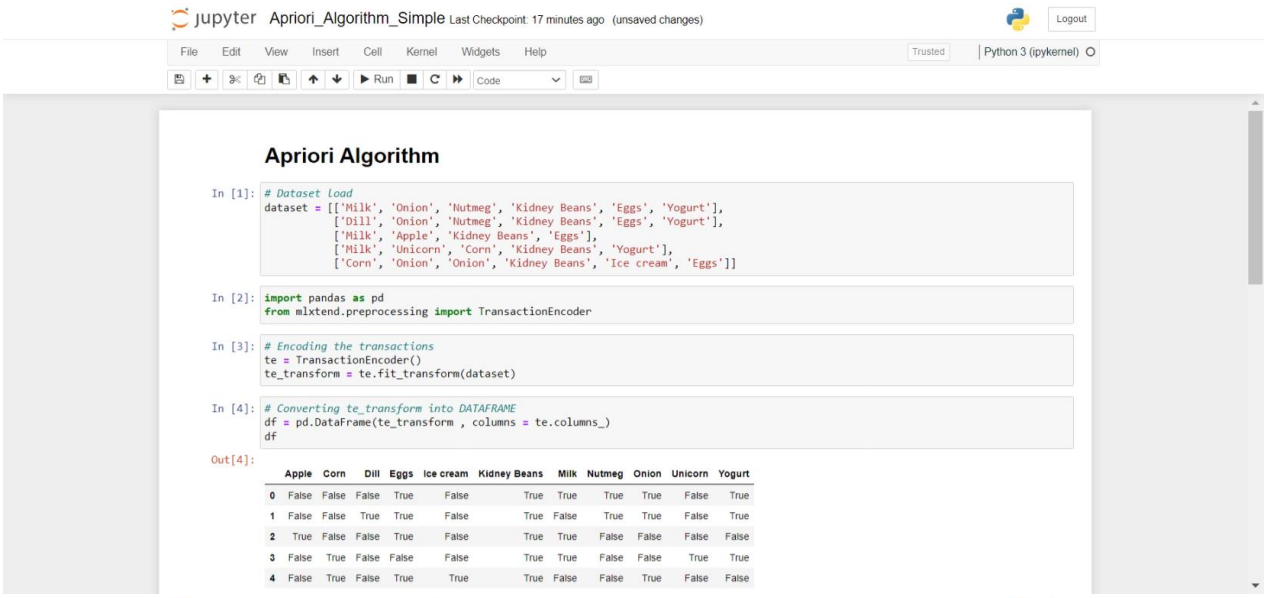
In [16]:
```
accuracy_score(y_test,clftree2.predict(x_test))
```

Out[16]: 0.5588235294117647

EXPERIMENT-9

AIM: Implement Apriori algorithm in python.

OUTPUT AND CODE:


jupyter Apriori_Algorithm_Simple Last Checkpoint: 17 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

Apriori Algorithm

In [1]:
```
# Dataset load
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

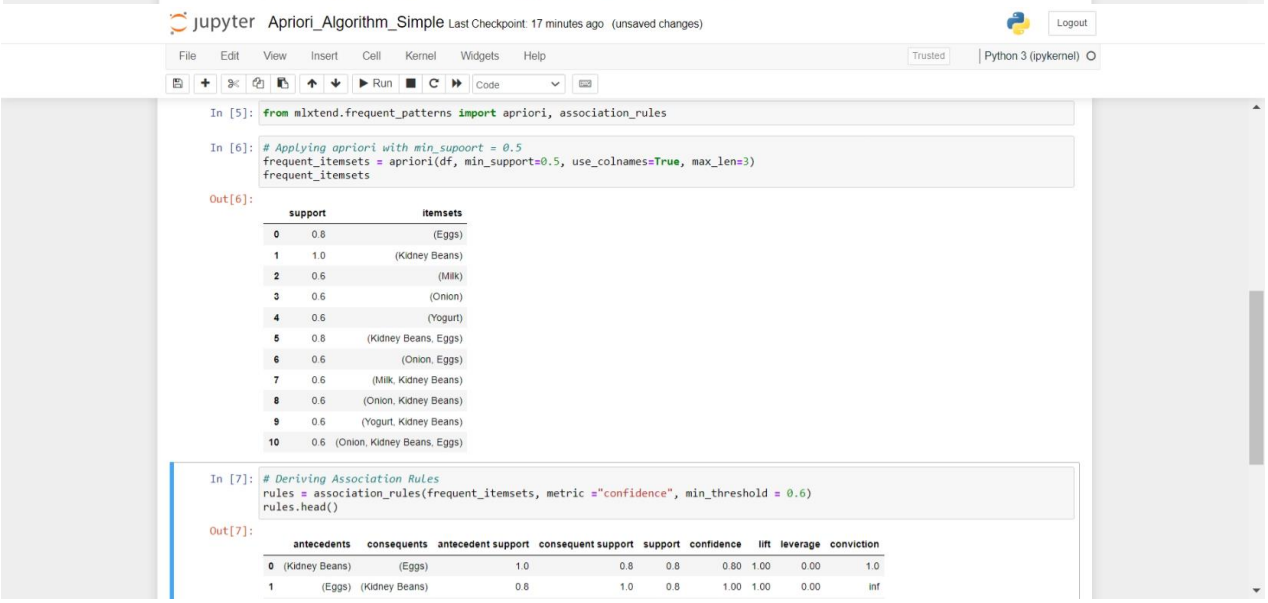
In [2]:
```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
```

In [3]:
```
# Encoding the transactions
te = TransactionEncoder()
te_transform = te.fit_transform(dataset)
```

In [4]:
```
# Converting te_transform into DATAFRAME
df = pd.DataFrame(te_transform , columns = te.columns_)
df
```

Out[4]:

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False


jupyter Apriori_Algorithm_Simple Last Checkpoint: 17 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [5]:
```
from mlxtend.frequent_patterns import apriori, association_rules
```

In [6]:
```
# Applying apriori with min_supoort = 0.5
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True, max_len=3)
frequent_itemsets
```

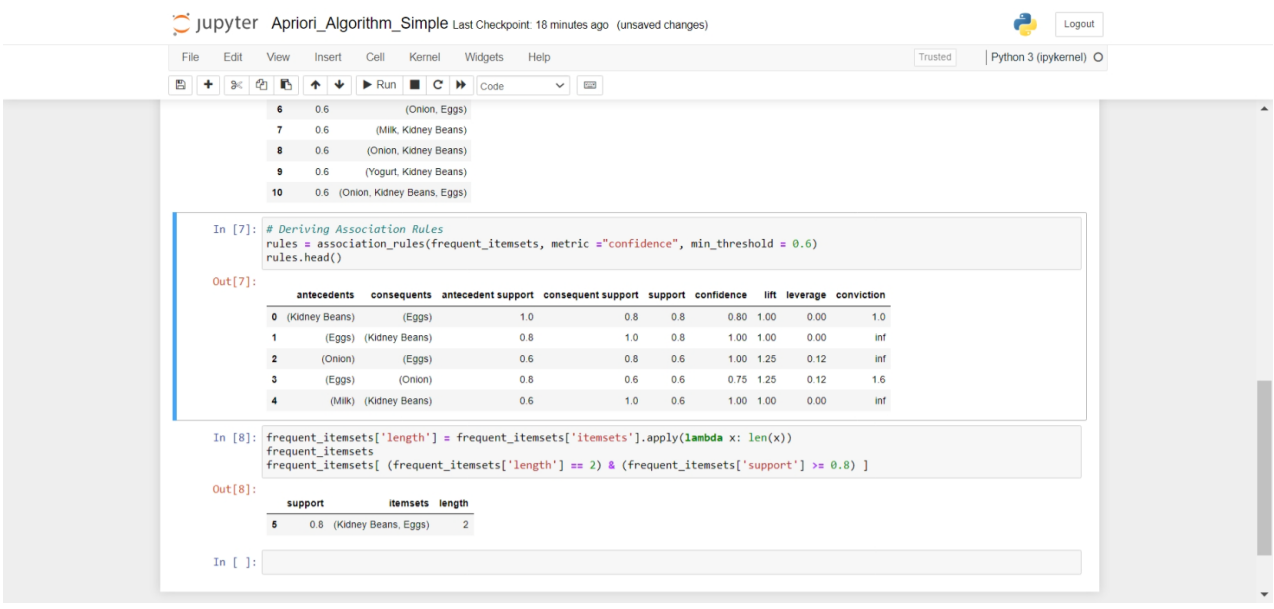
Out[6]:

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)
5	0.8	(Kidney Beans, Eggs)
6	0.6	(Onion, Eggs)
7	0.6	(Milk, Kidney Beans)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Yogurt, Kidney Beans)
10	0.6	(Onion, Kidney Beans, Eggs)

In [7]:
```
# Deriving Association Rules
rules = association_rules(frequent_itemsets, metric ="confidence", min_threshold = 0.6)
rules.head()
```

Out[7]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.0
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf


jupyter Apriori_Algorithm_Simple Last Checkpoint: 18 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

6	0.6	(Onion, Eggs)
7	0.6	(Milk, Kidney Beans)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Yogurt, Kidney Beans)
10	0.6	(Onion, Kidney Beans, Eggs)

In [7]:
```
# Deriving Association Rules
rules = association_rules(frequent_itemsets, metric ="confidence", min_threshold = 0.6)
rules.head()
```

Out[7]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.0
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf
2	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
3	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf

In [8]:
```
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
frequent_itemsets[ (frequent_itemsets['length'] == 2) & (frequent_itemsets['support'] >= 0.8) ]
```

Out[8]:

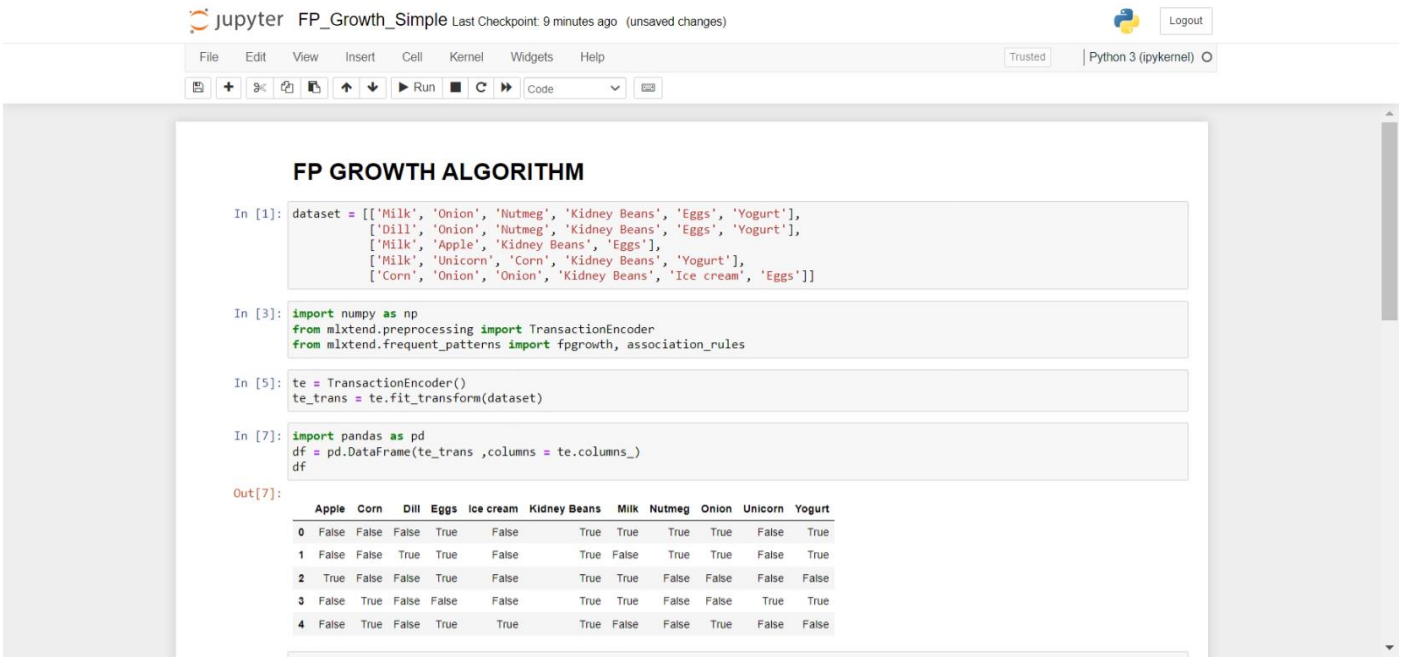
	support	itemsets	length
5	0.8	(Kidney Beans, Eggs)	2

In [ ]:

EXPERIMENT-10

AIM: Implement FP Growth algorithm in python.

CODE AND OUTPUT:



FP GROWTH ALGORITHM

In [1]:
```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

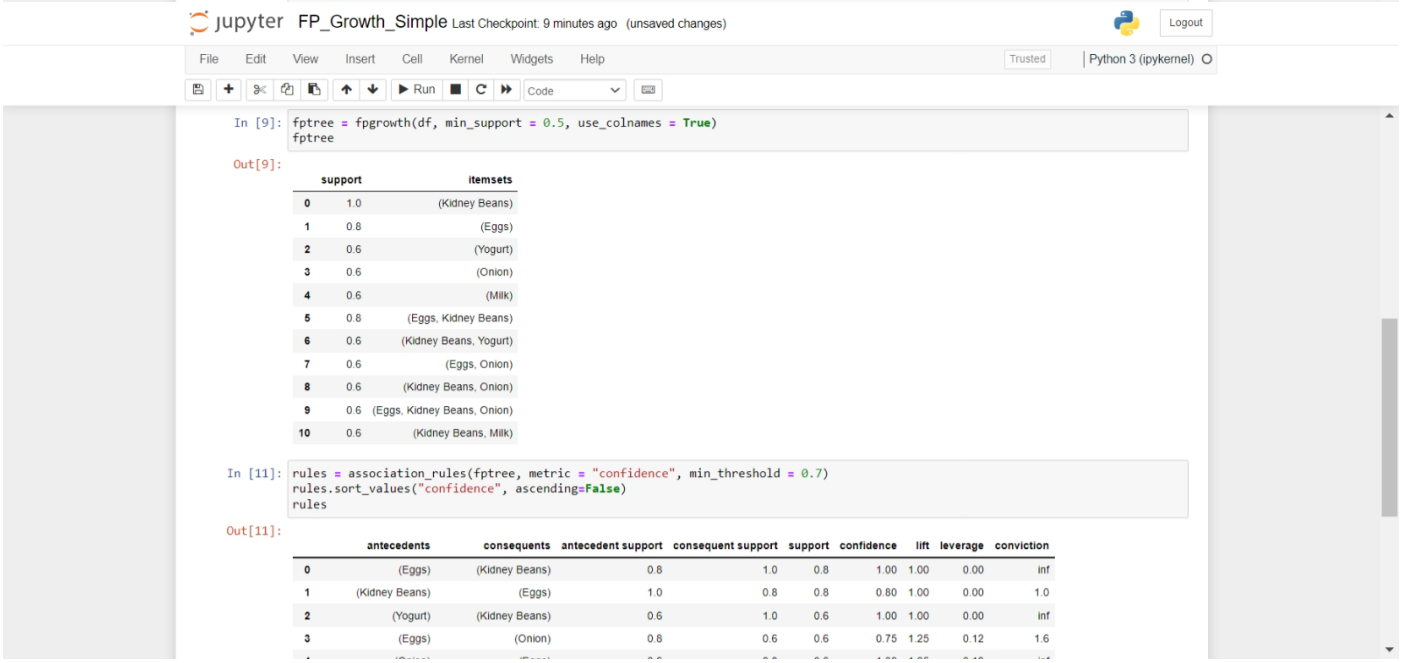
In [3]:
```
import numpy as np
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules
```

In [5]:
```
te = TransactionEncoder()
te_trans = te.fit_transform(dataset)
```

In [7]:
```
import pandas as pd
df = pd.DataFrame(te_trans ,columns = te.columns_)
df
```

Out[7]:

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogurt
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False



In [9]:
```
fptree = fpgrowth(df, min_support = 0.5, use_colnames = True)
fptree
```

Out[9]:

	support	itemsets
0	1.0	(Kidney Beans)
1	0.8	(Eggs)
2	0.6	(Yogurt)
3	0.6	(Onion)
4	0.6	(Milk)
5	0.8	(Eggs, Kidney Beans)
6	0.6	(Kidney Beans, Yogurt)
7	0.6	(Eggs, Onion)
8	0.6	(Kidney Beans, Onion)
9	0.6	(Eggs, Kidney Beans, Onion)
10	0.6	(Kidney Beans, Milk)

In [11]:
```
rules = association_rules(fptree, metric = "confidence", min_threshold = 0.7)
rules.sort_values("confidence", ascending=False)
rules
```

Out[11]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf
1	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.0
2	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
3	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6

EXPERIMENT-11

AIM: Implement K Means Clustering in python on any dataset.

CODE AND OUTPUT:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

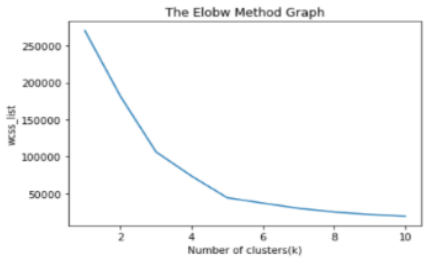
```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
```

```
x = dataset.iloc[:, [3, 4]].values
x
```

```
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],
       [ 23,  98]
```

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```




```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

```
#visualizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```