# DATA MINING

# LABORATORY

## TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **Exercise Number** | **Program Name** | **Page No.** |
| 1 | Demonstrate the following data preprocessing tasks using python libraries.   1. Loading the dataset 2. Identifying the dependent and independent variables 3. Dealing with missing data | 6-9 |
| 2 | Demonstrate the following data preprocessing tasks using python libraries.   1. Dealing with categorical data 2. Scaling the features 3. Splitting dataset into Training and Testing Sets | 10-14 |
| 3 | Demonstrate the following Similarity and Dissimilarity Measures using python   1. Pearson’s Correlation 2. Cosine Similarity 3. Jaccard Similarity 4. Euclidean Distance 5. Manhattan Distance | 15-17 |
| 4 | Build a classification model using Decision Tree algorithm on iris dataset | 18-20 |
| 5 | Apply Naive Bayes Classification algorithm on any dataset | 21-22 |
| 6 | Generate frequent itemsets using Apriori Algorithm in python and also generate association rules for any market basket data. | 23-25 |
| 7 | Apply FP-Growth algorithm on any market basket data. | 26-27 |
| 8 | Apply K- Means clustering algorithm on any dataset. | 28-30 |
| 9 | Apply Hierarchical Clustering algorithm on any dataset. | 31 |
| 10 | Apply DBSCAN clustering algorithm on any dataset. | 32-34 |

**EXERCISE 1 :**

**AIM: Demonstrate the following data preprocessing tasks using         python libraries.**

**a) Loading the dataset**

**b) Identifying the dependent and independent variables**

**c) Dealing with missing data**

**PROGRAM :**

**a) Loading the dataset**

**PROGRAM:**

import pandas as pd

df = pd.read\_csv('pima-indians-diabetes.csv')

print(df)

print(df.count(0))

**OUTPUT:**

6 148 72 35 0 33.6 0.627 50 1

0 1 85 66 29 0 26.6 0.351 31 0

1 8 183 64 0 0 23.3 0.672 32 1

2 1 89 66 23 94 28.1 0.167 21 0

3 0 137 40 35 168 43.1 2.288 33 1

4 5 116 74 0 0 25.6 0.201 30 0

.. .. ... .. .. ... ... ... .. ..

762 10 101 76 48 180 32.9 0.171 63 0

763 2 122 70 27 0 36.8 0.340 27 0

764 5 121 72 23 112 26.2 0.245 30 0

765 1 126 60 0 0 30.1 0.349 47 1

766 1 93 70 31 0 30.4 0.315 23 0

[767 rows x 9 columns]

6 767

148 767

72 767

35 767

0 767

33.6 767

0.627 767

50 767

1 767

dtype: int64

**b) Identifying the dependent and independent variables**

**PROGRAM:**

X = df.iloc[:, :-1].values

print(X)

Y = df.iloc[:, -1].values

print(Y)

**OUTPUT:**

[[1.00e+00 8.50e+01 6.60e+01 ... 2.66e+01 3.51e-01 3.10e+01]

[8.00e+00 1.83e+02 6.40e+01 ... 2.33e+01 6.72e-01 3.20e+01]

[1.00e+00 8.90e+01 6.60e+01 ... 2.81e+01 1.67e-01 2.10e+01]

...

[5.00e+00 1.21e+02 7.20e+01 ... 2.62e+01 2.45e-01 3.00e+01]

[1.00e+00 1.26e+02 6.00e+01 ... 3.01e+01 3.49e-01 4.70e+01]

[1.00e+00 9.30e+01 7.00e+01 ... 3.04e+01 3.15e-01 2.30e+01]]

[0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 1

1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0

.

.

.

0 0 1 0 1 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1

0 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 0 1 0]

**c) Dealing with missing data**

**PROGRAM:**

# load and summarize the dataset

from pandas import read\_csv

# load the dataset

dataset = read\_csv('pima-indians-diabetes.csv', header=None)

# summarize the dataset

print(dataset.describe())

# load the dataset and review rows

from pandas import read\_csv

# load the dataset

dataset = read\_csv('pima-indians-diabetes.csv', header=None)

# print the first 20 rows of data

print(dataset.head(20))

# example of summarizing the number of missing values for each variable

from pandas import read\_csv

# load the dataset

dataset = read\_csv('pima-indians-diabetes.csv', header=None)

# count the number of missing values for each column

num\_missing = (dataset[[1,2,3,4,5]] == 0).sum()

# report the results

print(num\_missing)

# example of marking missing values with nan values

from numpy import nan

from pandas import read\_csv

# load the dataset

dataset = read\_csv('pima-indians-diabetes.csv', header=None)

# replace '0' values with 'nan'

dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)

# count the number of nan values in each column

print(dataset.isnull().sum())

# example of review rows from the dataset with missing values marked

from numpy import nan

from pandas import read\_csv

# load the dataset

dataset = read\_csv('pima-indians-diabetes.csv', header=None)

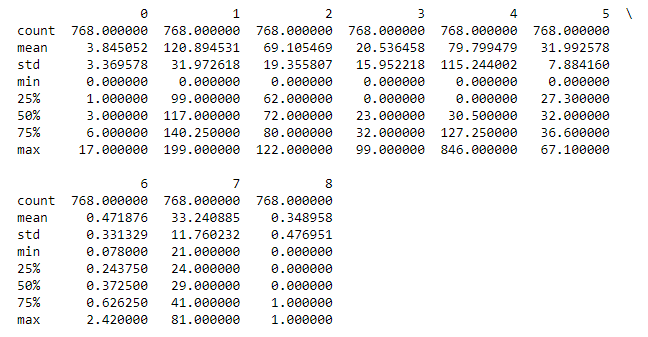
# replace '0' values with 'nan'

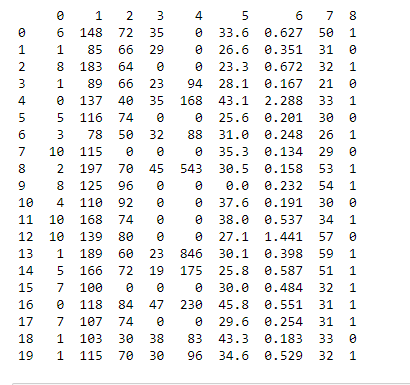
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)

# print the first 20 rows of data

print(dataset.head(20))

**OUTPUT:**

****

****

1 5

2 35

3 227

4 374

5 11

dtype: int64

0 0

1 5

2 35

3 227

4 374

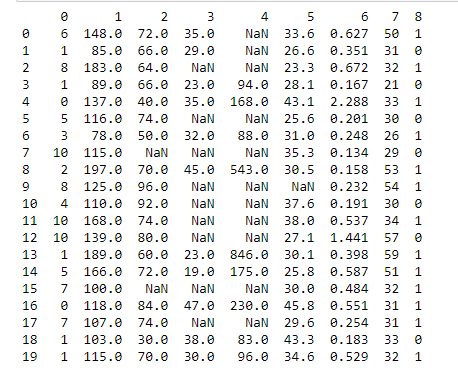
5 11

6 0

7 0

8 0

dtype: int64

****

**EXERCISE 2 :**

**AIM: Demonstrate the following data preprocessing tasks using         python libraries.**

**a) Dealing with categorical data**

**b) Scaling the features**

**c) Splitting dataset into Training and Testing Sets**

**PROGRAM :**

**a) Dealing with categorical data**

**PROGRAM:**

import pandas as pd

import sklearn

pip install category\_encoders

import category\_encoders as ce

data = pd.DataFrame({ 'gender' : ['Male', 'Female', 'Male', 'Female', 'Female'],

'class' : ['A','B','C','D','A'],

'city' : ['Delhi','Gurugram','Delhi','Delhi','Gurugram'] })

data.head()

ce\_OHE = ce.OneHotEncoder(cols=['gender','city'])

data1 = ce\_OHE.fit\_transform(data)

data1.head()

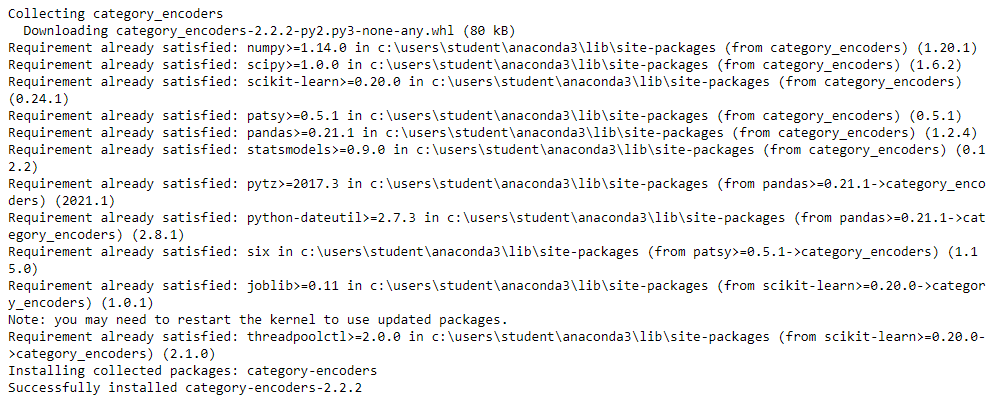
ce\_be = ce.BinaryEncoder(cols=['class']);

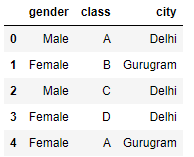
# transform the data

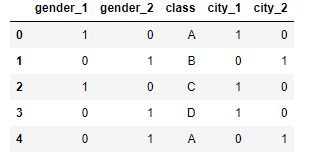
data\_binary = ce\_be.fit\_transform(data["class"]);

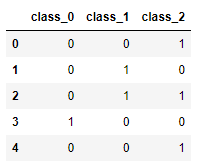
data\_binary

**OUTPUT:**

****

****

****

****

**b) Scaling the features**

**PROGRAM:**

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

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

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

sc\_y = StandardScaler()

Y\_train = Y\_train.reshape((len(Y\_train), 1))

Y\_train = sc\_y.fit\_transform(Y\_train)

Y\_train = Y\_train.ravel()

print(X\_train)

print(X\_test)

print(Y\_train)

print(Y\_test)

**OUTPUT:**

[[ 0.65130111 -0.47233997 0.39928583]

[-0.22656325 -0.26633979 0.52663913]

[-0.51918471 0.04266049 0.39928583]

...

[ 0.94392257 -0.26633979 -1.3199836 ]

[ 0.65130111 0.35166076 -1.3199836 ]

[-0.51918471 0.24866067 0.33560919]]

[[ 1.23654402 1.89666211 1.6091421 ]

[ 0.35867966 0.55766094 -1.3199836 ]

[ 0.94392257 -0.78134024 0.71766906]

[ 0.94392257 -3.56234267 -1.3199836 ]

.

.

.

.

[ 0.65130111 1.17566148 -1.3199836 ]

[ 0.0660582 -0.47233997 0.78134571]]

[-0.39053468 0.22886638 0.01207601 0.78632734 -0.48344484 -0.85508548

-0.70023522 1.37475836 -1.00993575 2.30385996 -0.5144149 0.50759686

-0.39053468 1.00311772 0.19789633 0.10498617 2.08706958 -0.23568442

0.13595622 -1.10284591 -0.32859458 -1.10284591 1.21990809 0.50759686

.

.

.

.

-0.32859458 -0.35956463 2.08706958 1.31281825 0.4146867 -0.23568442

0.10498617 0.22886638]

[167 132 160 100 122 85 178 73 96 111 88 131 108 123 99 99 85 187

158 124 83 133 79 90 145 181 91 100 113 81 155 115 128 140 116 123

107 84 108 133 0 165 91 95 125 57 122 119 148 136 197 108 109 123

.

.

.

146 82 151 75 137 81 137 79 179 147 148 139 165 190 103]

**c) Splitting dataset into Training and Testing Sets**

**PROGRAM:**

import pandas as pd

df = pd.read\_csv('pima-indians-diabetes.csv')

X = df.iloc[:,[0,2,3]].values

Y = df.iloc[:,1].values

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

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.3, random\_state = 0)

print(X\_train)

print(X\_test)

print(Y\_train)

print(Y\_test)

**OUTPUT:**

[[ 6 60 27]

[ 3 64 29]

[ 2 70 27]

...

[ 7 64 0]

[ 6 76 0]

[ 2 74 26]]

[[ 8 106 46]

[ 5 80 0]

[ 7 54 32]

[ 7 0 0]

[ 2 60 18]

[ 2 65 0]

[ 7 84 0]

.

.

.

.

[ 0 76 43]

[ 6 92 0]

[ 4 60 33]]

[109 129 122 147 106 94 99 166 89 196 105 138 109 154 128 125 189 114

126 86 111 86 161 138 151 129 147 104 142 77 143 120 125 112 162 118

122 152 115 130 124 126 100 72 168 123 101 136 180 112 99 120 196 100

123 166 145 158 135 84 74 75 130 109 151 162 117 71 127 95 136 80

197 87 85 71 87 120 139 83 96 183 100 126 81 95 112 83 129 165

132 97 120 175 116 143 92 87 97 158 106 153 116 137 125 122 141 128

125 146 147 95 124 107 101 94 87 94 106 173 169 114 174 195 125 139

.

.

.

.

162 143 163 120 80 158 124 129 127 121 154 179 136 114 173 115 112 131

96 136 139 108 128 114 111 110 189 164 135 114 125 129]

[167 132 160 100 122 85 178 73 96 111 88 131 108 123 99 99 85 187

158 124 83 133 79 90 145 181 91 100 113 81 155 115 128 140 116 123

107 84 108 133 0 165 91 95 125 57 122 119 148 136 197 108 109 123

119 62 183 150 91 127 145 99 132 128 115 172 125 71 129 168 90 142

92 94 181 61 123 117 114 110 124 114 106 116 126 137 119 115 179 108

150 90 109 101 189 98 100 111 180 101 100 109 168 146 102 117 65 119

182 89 170 151 103 109 97 108 146 155 106 102 116 146 194 100 138 142

118 105 86 92 126 103 147 102 96 90 140 105 95 106 180 121 107 95

99 68 176 106 144 114 84 134 117 105 183 99 124 88 154 85 137 129

93 107 102 112 129 128 87 57 173 88 103 120 123 139 126 115 80 91

134 121 89 146 121 141 112 101 77 90 106 115 101 117 95 98 115 102

146 196 106 91 138 114 111 132 131 92 113 74 171 84 167 125 99 134

146 82 151 75 137 81 137 79 179 147 148 139 165 190 103]

**EXERCISE 3 :**

**AIM: Demonstrate the following Similarity and Dissimilarity         Measures using python**

**a) Pearson’s Correlation b) Cosine Similarity**

**c) Jaccard Similarity d) Euclidean Distance**

**e) Manhattan Distance**

**PROGRAM :**

**a) Pearson’s Correlation**

**PROGRAM:**

# calculate the Pearson's correlation between two variables

from numpy.random import randn

from numpy.random import seed

from scipy.stats import pearsonr

# seed random number generator

seed(1)

# prepare data

data1 = 20 \* randn(1000) + 100

data2 = data1 + (10 \* randn(1000) + 50)

# calculate Pearson's correlation

corr, \_ = pearsonr(data1, data2)

print('Pearsons correlation: %.3f' % corr)

**OUTPUT:**

Pearsons correlation: 0.888

**b) Cosine Similarity**

**PROGRAM:**

from sklearn.metrics.pairwise import cosine\_similarity

# Vectors

vec\_a = [1, 2, 3, 4, 5]

vec\_b = [1, 3, 5, 7, 9]

# Dot and norm

dot = sum(a\*b for a, b in zip(vec\_a, vec\_b))

norm\_a = sum(a\*a for a in vec\_a) \*\* 0.5

norm\_b = sum(b\*b for b in vec\_b) \*\* 0.5

# Cosine similarity

cos\_sim = dot / (norm\_a\*norm\_b)

# Results

print('My version:', cos\_sim)

print('Scikit-Learn:', cosine\_similarity([vec\_a], [vec\_b]))

**OUTPUT:**

My version: 0.9972413740548081

Scikit-Learn: [[0.99724137]]

**c) Jaccard Similarity**

**PROGRAM:**

import numpy as np

a = [0, 5, 2, 8, 6, 8, 9]

b = [0, 7, 3, 4, 5, 7, 9]

#define Jaccard Similarity function

def jaccard(list1, list2):

intersection = len(list(set(list1).intersection(list2)))

union = (len(list1) + len(list2)) - intersection

return float(intersection) / union

#find Jaccard Similarity between the two sets

jaccard(a, b)

c = [0, 1, 2, 3, 4, 5]

d = [6, 7, 8, 9, 10]

jaccard(c, d)

e = [0, 1, 2, 3, 4, 5]

f = [0, 1, 2, 3, 4, 5]

jaccard(e, f)

**OUTPUT:**

0.2727272727272727

0.0

1.0

**d) Euclidean Distance**

**PROGRAM:**

from scipy.spatial import distance

a = (1, 2, 3)

b = (4, 5, 6)

print(distance.euclidean(a, b))

**OUTPUT:**

5.196152422706632

**e) Manhattan Distance**

**PROGRAM:**

import numpy as np

a = np.array((1, 2, 3))

b = np.array((4, 5, 6))

dist = np.linalg.norm(a-b)

print(dist)

from math import sqrt

#create function to calculate Manhattan distance

def manhattan(a, b):

return sum(abs(val1-val2) for val1, val2 in zip(a,b))

#define vectors

A = [2, 4, 4, 6]

B = [5, 5, 7, 8]

#calculate Manhattan distance between vectors

manhattan(A, B)

**OUTPUT:**

5.196152422706632

9

**EXERCISE 4 :**

**AIM: Build a classification model using Decision Tree algorithm on         iris dataset**

**PROGRAM :**

%matplotlib inline

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier

df = pd.read\_csv('Iris.csv')

df.isnull().any()

df.dtypes

df.describe()

df['PetalWidthCm'].plot.hist()

plt.show()

sns.pairplot(df, hue='Species')

all\_inputs = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values

all\_classes = df['Species'].values

(train\_inputs, test\_inputs, train\_classes, test\_classes) = train\_test\_split(all\_inputs, all\_classes, train\_size=0.7, random\_state=1)

dtc = DecisionTreeClassifier()

dtc.fit(train\_inputs,train\_classes)

dtc.score(test\_inputs,test\_classes)

**OUTPUT:**

Id False

SepalLengthCm False

SepalWidthCm False

PetalLengthCm False

PetalWidthCm False

Species False

dtype: bool

Id int64

SepalLengthCm float64

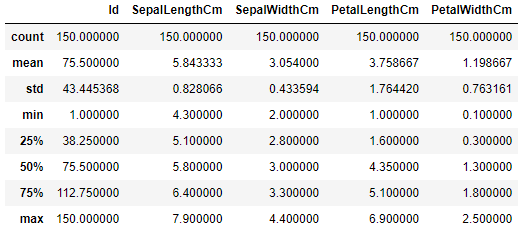
SepalWidthCm float64

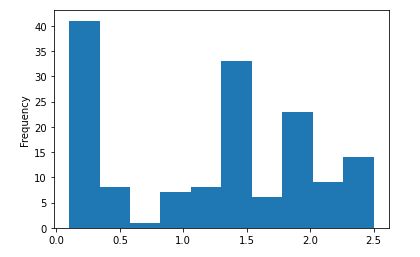
PetalLengthCm float64

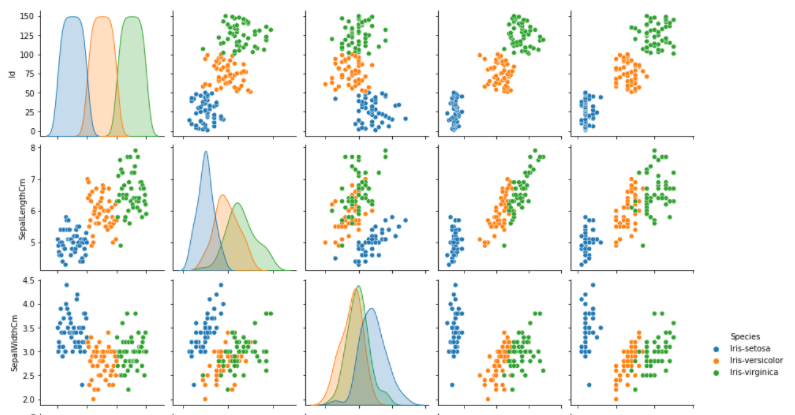
PetalWidthCm float64

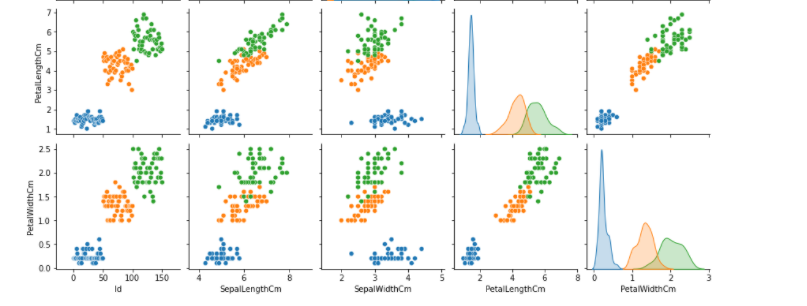
Species object

dtype: object

****

****

****

****

0.9555555555555556

**EXERCISE 5 :**

**AIM: Apply Naive Bayes Classification algorithm on any dataset**

**PROGRAM :**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import sklearn

dataset = pd.read\_csv('Iris.csv')

X = dataset.iloc[:, [1, 2, 3]].values

y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X[:,0] = le.fit\_transform(X[:,0])

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

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

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

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

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

y\_pred

y\_test

from sklearn.metrics import confusion\_matrix,accuracy\_score

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

ac = accuracy\_score(y\_test,y\_pred)

print(cm)

print(ac)

**OUTPUT:**

GaussianNB()

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',

'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',

'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',

'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',

'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',

'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',

'Iris-versicolor', 'Iris-setosa'], dtype='<U15')

array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',

'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',

'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',

'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',

'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',

'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',

'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',

'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',

'Iris-versicolor', 'Iris-setosa'], dtype=object)

[[11 0 0]

[ 0 11 2]

[ 0 2 4]]

0.8666666666666667

**EXERCISE 6 :**

**AIM: Generate frequent itemsets using Apriori Algorithm in python and         also generate association rules for any market basket data**.

**PROGRAM :**

pip install mlxtend

import pandas as pd

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

df = pd.read\_excel('Online\_Retail.xlsx')

df.head()

df['Description'] = df['Description'].str.strip()

df.dropna(axis=0, subset=['InvoiceNo'], inplace=True)

df['InvoiceNo'] = df['InvoiceNo'].astype('str')

df = df[~df['InvoiceNo'].str.contains('C')]

df

basket = (df[df['Country'] =="France"]

.groupby(['InvoiceNo', 'Description'])['Quantity']

.sum().unstack().reset\_index().fillna(0)

.set\_index('InvoiceNo'))

Basket

def encode\_units(x):

if x <= 0:

return 0

if x >= 1:

return 1

basket\_sets = basket.applymap(encode\_units)

basket\_sets.drop('POSTAGE', inplace=True, axis=1)

basket\_sets

frequent\_itemsets = apriori(basket\_sets, min\_support=0.07, use\_colnames=True)

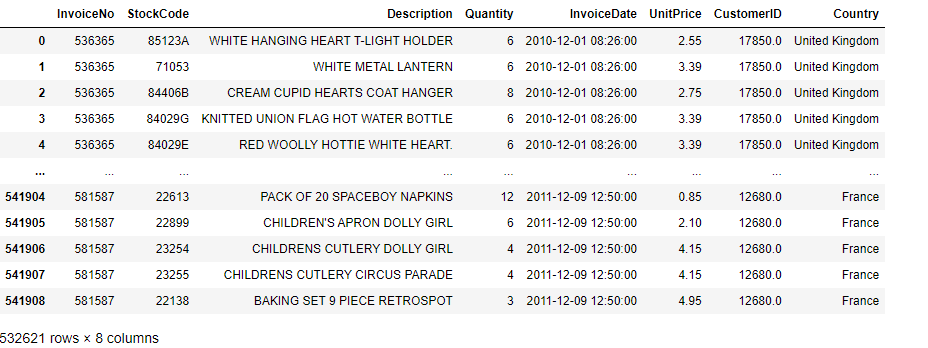
rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1)

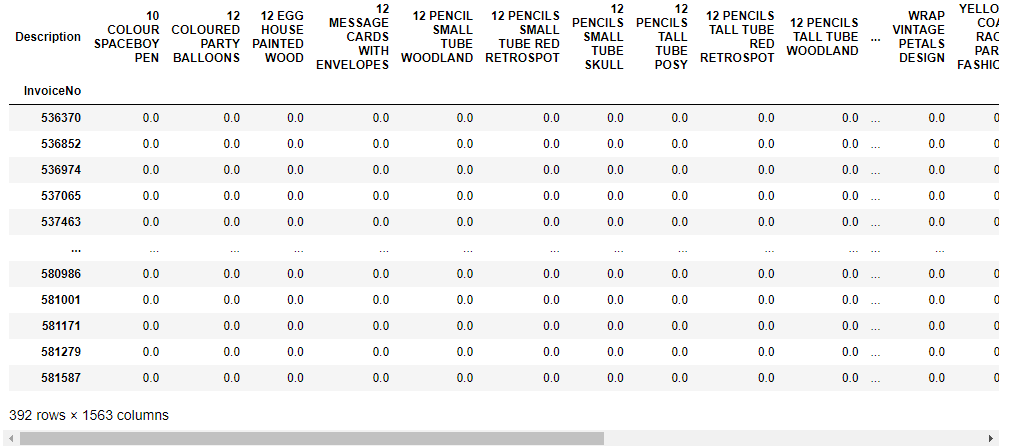
rules.head()

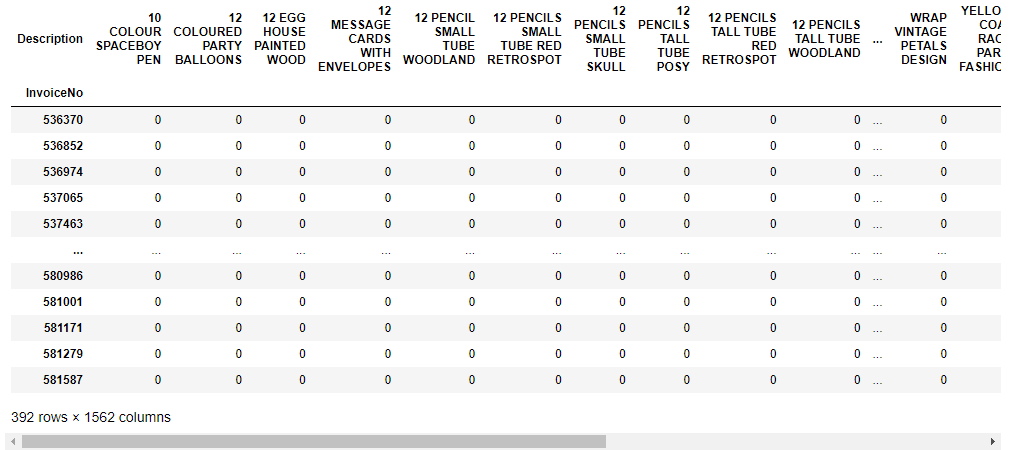
rules[ (rules['lift'] >= 6) &

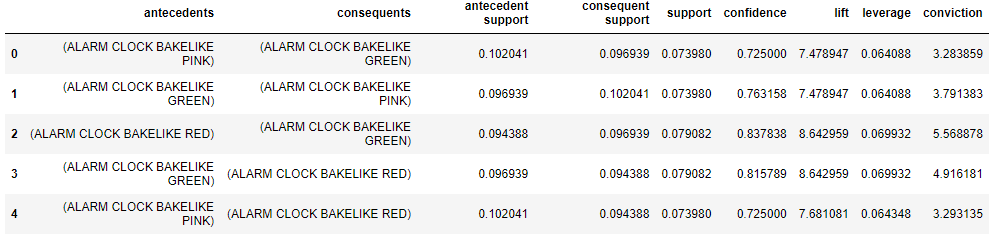
(rules['confidence'] >= 0.8) ]

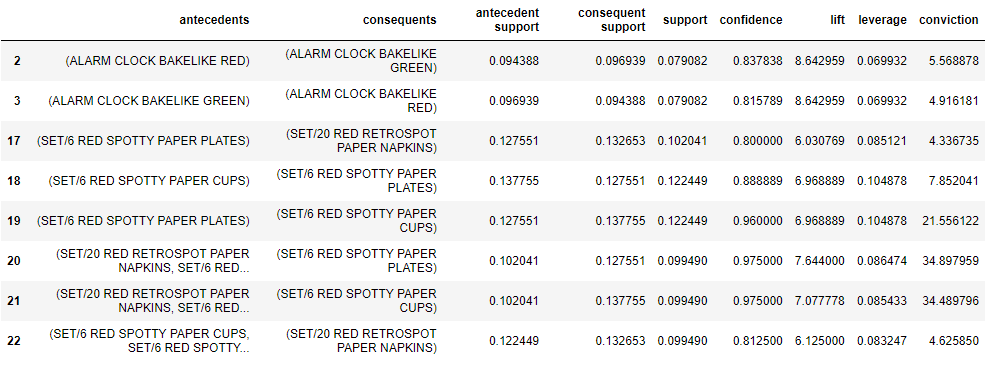
**OUTPUT:**

****

****

****

****

****

**EXERCISE 7 :**

**AIM: Apply FP-Growth algorithm on any market basket data**.

**PROGRAM :**

pip install apyori

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from apyori import apriori

# Data Preprocessing

dataset = pd.read\_csv('Market\_Basket\_Optimisation.csv', header = None)

#Getting the list of transactions from the dataset

transactions = []

for i in range(0, 7501):

transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

# Training Apriori algorithm on the dataset

rule\_list = apriori(transactions, min\_support = 0.003, min\_confidence = 0.3, min\_lift = 3, min\_length = 2)

# Visualizing the list of rules

results = list(rule\_list)

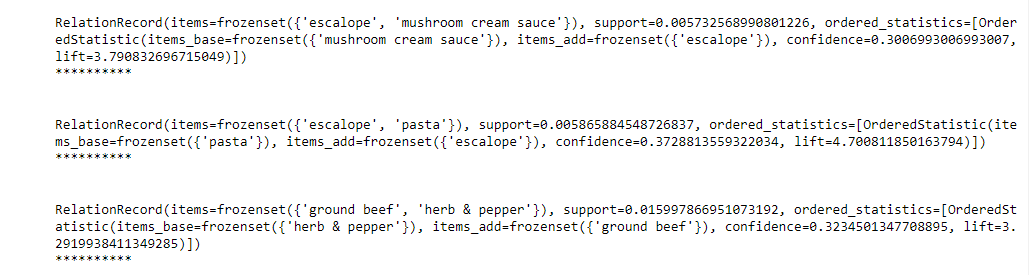
for i in results:

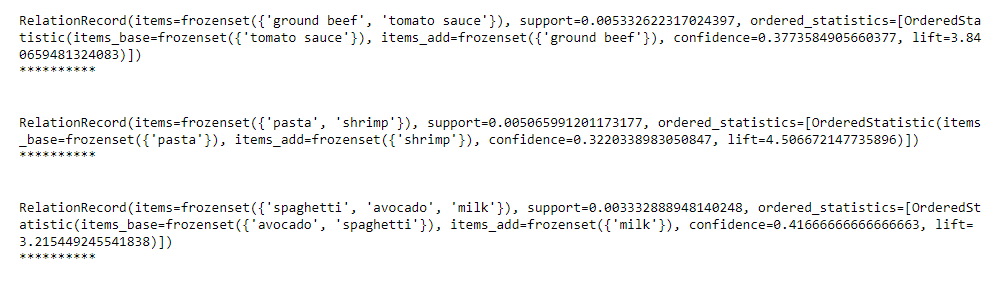
print('\n')

print(i)

print('\*\*\*\*\*\*\*\*\*\*')

**OUTPUT:**

****

****

**EXERCISE 8 :**

**AIM: Apply K- Means clustering algorithm on any dataset.**

**PROGRAM :**

#import libraries

import pandas as pd

import numpy as np

import random as rd

import matplotlib.pyplot as plt

data = pd.read\_csv('clustering.csv')

data.head()

X = data[["LoanAmount","ApplicantIncome"]]

#Visualise data points

plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')

plt.xlabel('AnnualIncome')

plt.ylabel('Loan Amount (In Thousands)')

plt.show()

# Step 1 and 2 - Choose the number of clusters (k) and select random centroid for each cluster

#number of clusters

K=3

# Select random observation as centroids

Centroids = (X.sample(n=K))

plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')

plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')

plt.xlabel('AnnualIncome')

plt.ylabel('Loan Amount (In Thousands)')

plt.show()

# Step 3 - Assign all the points to the closest cluster centroid

# Step 4 - Recompute centroids of newly formed clusters

# Step 5 - Repeat step 3 and 4

diff = 1

j=0

while(diff!=0):

XD=X

i=1

for index1,row\_c in Centroids.iterrows():

ED=[]

for index2,row\_d in XD.iterrows():

d1=(row\_c["ApplicantIncome"]-row\_d["ApplicantIncome"])\*\*2

d2=(row\_c["LoanAmount"]-row\_d["LoanAmount"])\*\*2

d=np.sqrt(d1+d2)

ED.append(d)

X[i]=ED

i=i+1

C=[]

for index,row in X.iterrows():

min\_dist=row[1]

pos=1

for i in range(K):

if row[i+1] < min\_dist:

min\_dist = row[i+1]

pos=i+1

C.append(pos)

X["Cluster"]=C

Centroids\_new = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]

if j == 0:

diff=1

j=j+1

else:

diff = (Centroids\_new['LoanAmount'] - Centroids['LoanAmount']).sum() + (Centroids\_new['ApplicantIncome'] - Centroids['ApplicantIncome']).sum()

print(diff.sum())

Centroids = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]

color=['blue','green','cyan']

for k in range(K):

data=X[X["Cluster"]==k+1]

plt.scatter(data["ApplicantIncome"],data["LoanAmount"],c=color[k])

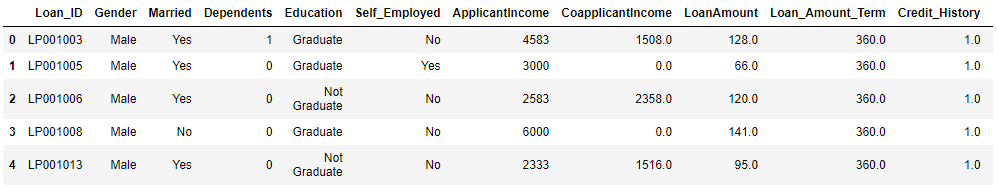
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')

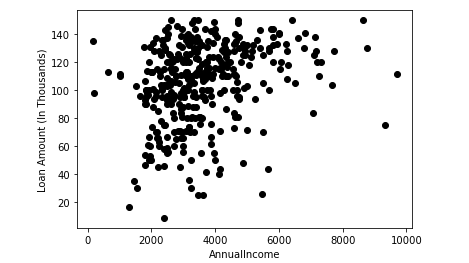
plt.xlabel('Income')

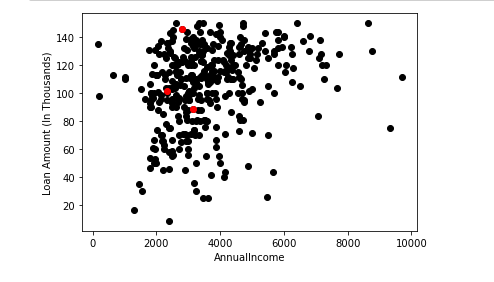
plt.ylabel('Loan Amount (In Thousands)')

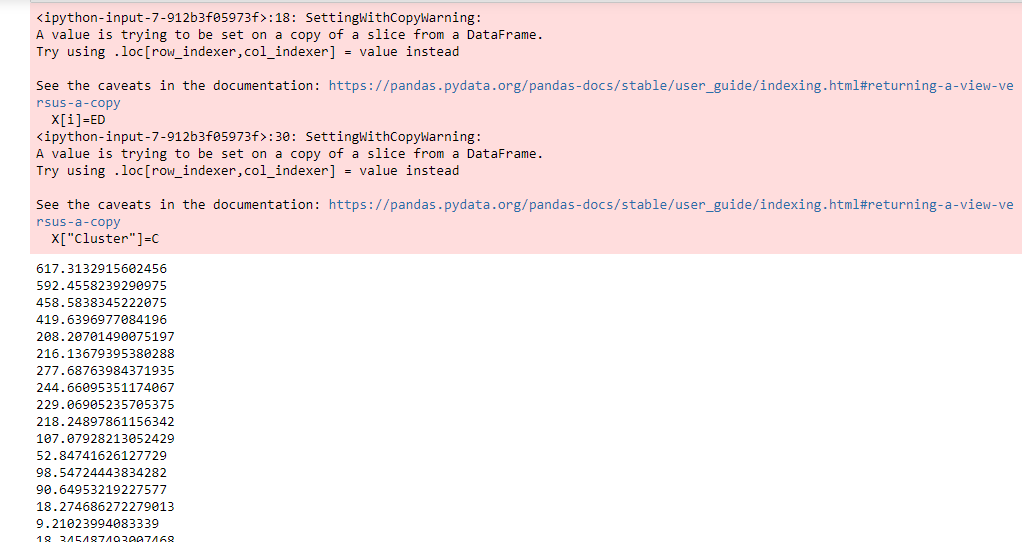
plt.show()

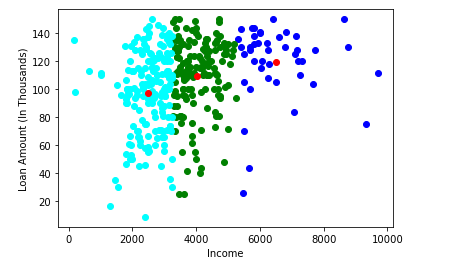
**OUTPUT:**

****

****

****

****

****

**EXERCISE 9 :**

**AIM: Apply Hierarchical Clustering algorithm on any dataset.**

**PROGRAM :**

#Importing required libraries

from sklearn.datasets import load\_iris

from sklearn.cluster import AgglomerativeClustering

import numpy as np

import matplotlib.pyplot as plt

#Getting the data ready

data = load\_iris()

df = data.data

#Selecting certain features based on which clustering is done

df = df[:,1:3]

#Creating the model

agg\_clustering = AgglomerativeClustering(n\_clusters = 3, affinity = 'euclidean', linkage = 'ward')

#predicting the labels

labels = agg\_clustering.fit\_predict(df)

#Plotting the results

plt.figure(figsize = (8,5))

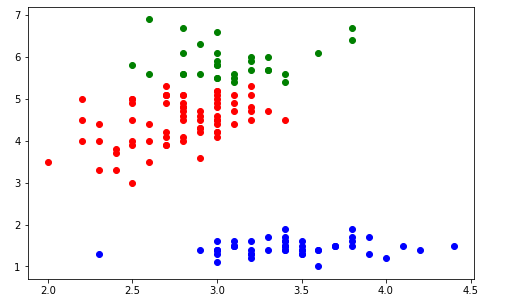
plt.scatter(df[labels == 0 , 0] , df[labels == 0 , 1] , c = 'red')

plt.scatter(df[labels == 1 , 0] , df[labels == 1 , 1] , c = 'blue')

plt.scatter(df[labels == 2 , 0] , df[labels == 2 , 1] , c = 'green')

plt.show()

**OUTPUT:**

****

**EXERCISE 10 :**

**AIM: Apply DBSCAN clustering algorithm on any dataset.**

**PROGRAM :**

import numpy as np

from sklearn.cluster import DBSCAN

from sklearn import metrics

from sklearn.datasets import make\_blobs

from sklearn.preprocessing import StandardScaler

# Generate sample data

centers = [[1, 1], [-1, -1], [1, -1]]

X, labels\_true = make\_blobs(n\_samples=750, centers=centers, cluster\_std=0.4,

random\_state=0)

X = StandardScaler().fit\_transform(X)

# Compute DBSCAN

db = DBSCAN(eps=0.3, min\_samples=10).fit(X)

core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = True

labels = db.labels\_

# Number of clusters in labels, ignoring noise if present.

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)

n\_noise\_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n\_clusters\_)

print('Estimated number of noise points: %d' % n\_noise\_)

print("Homogeneity: %0.3f" % metrics.homogeneity\_score(labels\_true, labels))

print("Completeness: %0.3f" % metrics.completeness\_score(labels\_true, labels))

print("V-measure: %0.3f" % metrics.v\_measure\_score(labels\_true, labels))

print("Adjusted Rand Index: %0.3f"

% metrics.adjusted\_rand\_score(labels\_true, labels))

print("Adjusted Mutual Information: %0.3f"

% metrics.adjusted\_mutual\_info\_score(labels\_true, labels))

print("Silhouette Coefficient: %0.3f"

% metrics.silhouette\_score(X, labels))

# Plot result

import matplotlib.pyplot as plt

%matplotlib inline

# Black removed and is used for noise instead.

unique\_labels = set(labels)

colors = [plt.cm.Spectral(each)

for each in np.linspace(0, 1, len(unique\_labels))]

for k, col in zip(unique\_labels, colors):

if k == -1:

# Black used for noise.

col = [0, 0, 0, 1]

class\_member\_mask = (labels == k)

xy = X[class\_member\_mask & core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),

markeredgecolor='k', markersize=14)

xy = X[class\_member\_mask & ~core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),

markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n\_clusters\_)

plt.show()

import numpy as np

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.datasets import make\_circles

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import DBSCAN

X, y = make\_circles(n\_samples=750, factor=0.3, noise=0.1)

X = StandardScaler().fit\_transform(X)

y\_pred = DBSCAN(eps=0.3, min\_samples=10).fit\_predict(X)

plt.scatter(X[:,0], X[:,1], c=y\_pred)

print('Number of clusters: {}'.format(len(set(y\_pred[np.where(y\_pred != -1)]))))

print('Homogeneity: {}'.format(metrics.homogeneity\_score(y, y\_pred)))

print('Completeness: {}'.format(metrics.completeness\_score(y, y\_pred)))

print("V-measure: %0.3f" % metrics.v\_measure\_score(labels\_true, labels))

print("Adjusted Rand Index: %0.3f"

% metrics.adjusted\_rand\_score(labels\_true, labels))

print("Adjusted Mutual Information: %0.3f"

% metrics.adjusted\_mutual\_info\_score(labels\_true, labels))

print("Silhouette Coefficient: %0.3f"

% metrics.silhouette\_score(X, labels))

**OUTPUT:**

Estimated number of clusters: 3

Estimated number of noise points: 18

Homogeneity: 0.953

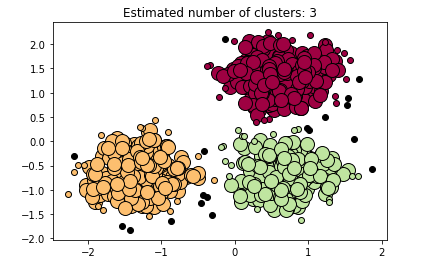
Completeness: 0.883

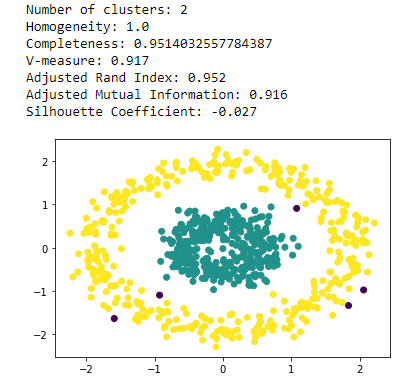
V-measure: 0.917

Adjusted Rand Index: 0.952

Adjusted Mutual Information: 0.916

Silhouette Coefficient: 0.626

****

****