DATA MINING USING PYTHON LAB

EX:1

1. 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

a) Loading the dataset

```
#Importing the pandas
import pandas as pd
#Loading the dataset
dataset = pd.read_excel("age_salary.xls")
print(dataset)
```

output:

Index	Years	Experience	Age	Salary
0	1	1.1	21.0	39343.0
1	2	1.3	21.5	46205.0
2	3	1.5	21.7	37731.0
3	4	2.0	22.0	43525.0
4	5	2.2	22.2	39891.0
5	6	2.9	23.0	56642.0
6	7	3.0	23.0	60150.0
7	8	3.2	23.3	54445.0
8	9	3.2	23.3	64445.0
9	10	3.7	23.6	57189.0
10	11	3.9	23.9	63218.0
11	12	4.0	24.0	55794.0
12	13	4.9	NaN	56957.0
13	14	4.1	24.0	57081.0
14	15	4.5	25.0	61111.0
15	16	4.9	25.0	67938.0
16	17	5.1	26.0	NaN
17	18	5.3	27.0	83088.0
18	19	5.9	28.0	81363.0
19	20	6.0	29.0	93940.0
20	21	6.8	30.0	91738.0

21	22	7.1	30.0	98273.0
22	23	7.9	31.0	101302.0
23	24	8.2	32.0	NaN
24	25	8.7	33.0	109431.0
25	26	9.0	34.0	105582.0
26	27	NaN	35.0	116969.0
27	28	9.6	NaN	112635.0
28	29	10.3	37.0	122391.0
29	30	10.5	38.0	NaN

b)Identifying the dependent and Independent Variables:

X = dataset.iloc[:,:-1].values **#Takes all rows of all columns except the last column, independent variable set**
print(X)
output:

```
[[ 1.   1.1 21. ][ 2.   1.3 21.5][ 3.   1.5 21.7][ 4.   2.  22. ][ 5.   2.2 22.2][ 6.   2.9 23.]
[ 7.   3.  23. ][ 8.   3.2 23.3][ 9.   3.2 23.3][10.   3.7 23.6][11.   3.9 23.9][12.   4.  24. ]
[13.   4.9  nan][14.   4.1 24. ][15.   4.5 25. ][16.   4.9 25. ][17.   5.1 26. ][18.   5.3 27.
]
[19.   5.9 28. ][20.   6.  29. ][21.   6.8 30. ][22.   7.1 30. ][23.   7.9 31. ][24.   8.2 32. ]
[25.   8.7 33. ][26.   9.  34. ] [27.   nan 35. ] [28.   9.6  nan][29.  10.3 37. ]
[30.  10.5 38. ]]
```

Y = dataset.iloc[:,-1].values **# Takes all rows of the last column, dependent variable set**
print(Y)
output:

```
 [39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
  57189.  63218.  55794.  56957.  57081.  61111.  67938.     nan  83088.
  81363.  93940.  91738.  98273. 101302.     nan 109431. 105582. 116969.
 112635. 122391.     nan]
```

c)Dealing with Missing Data:

from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy="mean")
X = imp.fit_transform(X)
print(X)
output:

```
[[ 1.   1.1 21. ][ 2.   1.3 21.5][ 3.   1.5 21.7][ 4.   2.  22. ][ 5.   2.2 22.2][ 6.   2.9 23. ]
 [ 7.   3.  23. ][ 8.   3.2 23.3][ 9.   3.2 23.3][10.   3.7 23.6][11.   3.9 23.9][12.   4.  24. ]
 [13.   4.9  nan][14.   4.1 24. ][15.   4.5 25. ][16.   4.9 25. ][17.   5.1 26. ][18.   5.3 27.
]
 [19.   5.9 28. ][20.   6.  29. ][21.   6.8 30. ][22.   7.1 30. ][23.   7.9 31. ][24.   8.2 32. ]
 [25.   8.7 33. ][26.   9.  34. ][27.   nan 35. ][28.   9.6  nan][29.  10.3 37. ][30.  10.5
38.]]
Y = Y.reshape(-1,1)
Y = imp.fit_transform(Y)
Y = Y.reshape(-1)
print(Y)
```

output:

```
[ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
  57189.  63218.  55794.  56957.  57081.  61111.  67938.     nan  83088.
  81363.  93940.  91738.  98273. 101302.     nan 109431. 105582. 116969.
 112635. 122391.     nan]
```

EX:2

2.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

a) Dealing with categorical data

#Importing the pandas

import pandas as pd

#Loading the dataset

dataset = pd.read_excel("D:\dataset.csv.xlsx")

print(dataset)

output:

	Index	nation	purchased_item	age	salary
0	0	india	no	25.0	35000.0
1	1	russia	yes	27.0	40000.0
2	2	germany	no	50.0	54000.0
3	3	russia	no	35.0	55909.1
4	4	germany	yes	40.0	60000.0
5	5	india	yes	35.0	58000.0

6	6	russia	no	39.1	52000.0
7	7	india	yes	48.0	79000.0
8	8	germany	no	50.0	83000.0
9	9	india	yes	37.0	55909.1
10	10	germany	no	21.0	24000.0
11	11	india	yes	39.1	60000.0
12	12	russia	no	63.0	70000.0

b) Scaling the features

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
print(X_train)
```

Output:

```
[[2 'no' 50.0][8 'no' 50.0][1 'yes' 27.0][7 'yes' 48.0][9 'yes' 37.0][3 'no' 35.0]
 [0 'no' 25.0][5 'yes' 35.0][12 'no' 63.0]]
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(Y_train)
```

Output:

```
[-1.50755672 -1.50755672  1.20604538 -0.15075567 -0.15075567  1.20604538
 -0.15075567 -0.15075567  1.20604538]
```

c) Splitting dataset into Training and Testing Sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3,
random_state = 0)
print(X_train)
```

Output:

```
[[2 'no' 50.0][8 'no' 50.0][1 'yes' 27.0][7 'yes' 48.0][9 'yes' 37.0][3 'no' 35.0]
 [0 'no' 25.0][5 'yes' 35.0][12 'no' 63.0]]
print(Y_test)
```

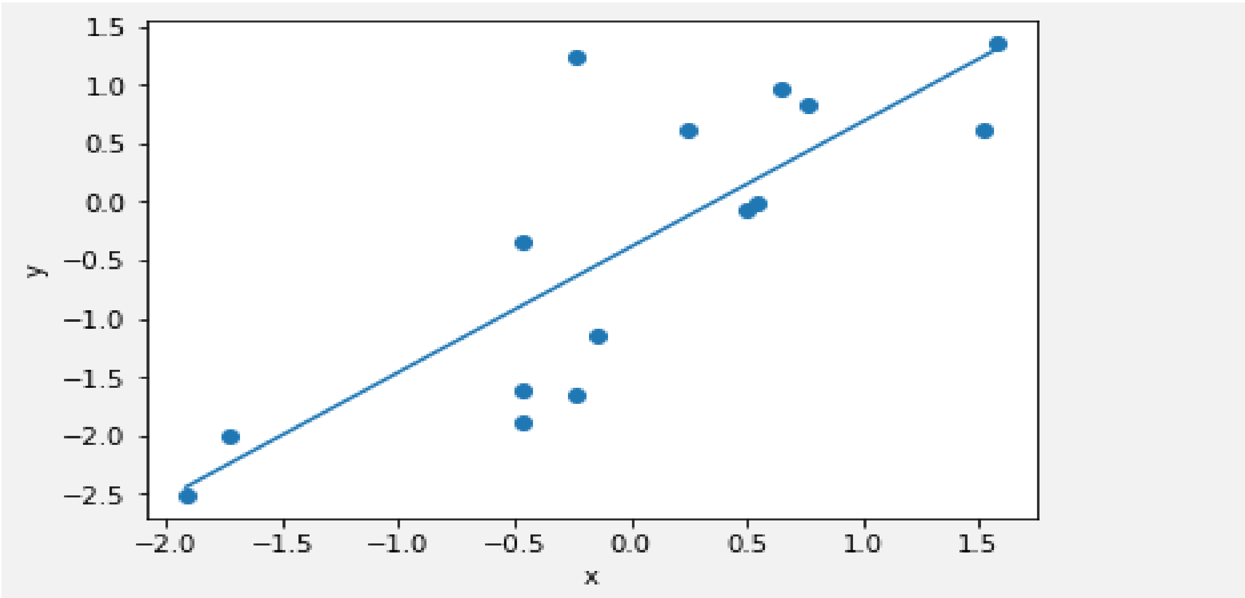
Output:
[2 1 0 0]
print(Y_train)
Output:
[0 0 2 1 1 2 1 1 2]

EX:3

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
a) Pearson’s Correlation :

```
import numpy as np
from scipy.stats import pearsonr
import matplotlib.pyplot as plt# seed random number generator
np.random.seed(42)
# prepare data
x = np.random.randn(15)
y = x + np.random.randn(15)# plot x and y
plt.scatter(x, y)
plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))(np.unique(x)))
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

OUTPUT:



b) Cosine Similarity :

```
from sklearn.metrics.pairwise import cosine_similarity
cos_sim = cosine_similarity(x.reshape(1,-1),y.reshape(1,-1))
print('Cosine similarity: %.3f' % cos_sim)
```

OUTPUT:

Cosine similarity: 0.773

c) Jaccard Similarity :

```
from sklearn.metrics import jaccard_score
A = [1, 1, 1, 0]
B = [1, 1, 0, 1]
jacc = jaccard_score(A,B)
print(‘Jaccard similarity: %.3f’ % jacc)
```

OUTPUT:

Jaccard similarity: 0.500

d) Euclidean Distance:

```
from scipy.spatial import distance
dst = distance.euclidean(x,y)
print(‘Euclidean distance: %.3f’ % dst)
```

Output:

Euclidean distance: 3.273

e) Manhattan Distance:

```
from scipy.spatial import distance
dst = distance.cityblock(x,y)
print('Manhattan distance: %.3f' % dst)
```

output: Manhattan distance: 10.468

EX:4

Build a model using linear regression algorithm on any dataset .

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import seaborn as sns
from matplotlib import rcParams

%matplotlib inline
%pylab inline
df = pd.read_csv('D:\kc_house_data.csv')
df.head()
```

Output:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080

 # Checking to see if any of our data has null values. If there were any, we'd drop or filter the null values out.

```
df.isnull().any()
```

Output:

id False
date False
price False
bedrooms False
bathrooms False
sqft_living False
sqft_lot False
dtype: bool

 # Checking out the data types for each of our variables. We want to get a sense of whether or not data is numerical (int64, float64) or not (object).

df.dtypes

output:

id int64
date object
price float64
bedrooms int64
bathrooms float64
sqft_living int64
sqft_lot int64
dtype: object

#Simple exploratory analysis and regression results, **use df.describe() to look at all the variables in your analysis, plot histograms of the variables that the analysis is targeting using plt.pyplot.hist()**

df.describe()

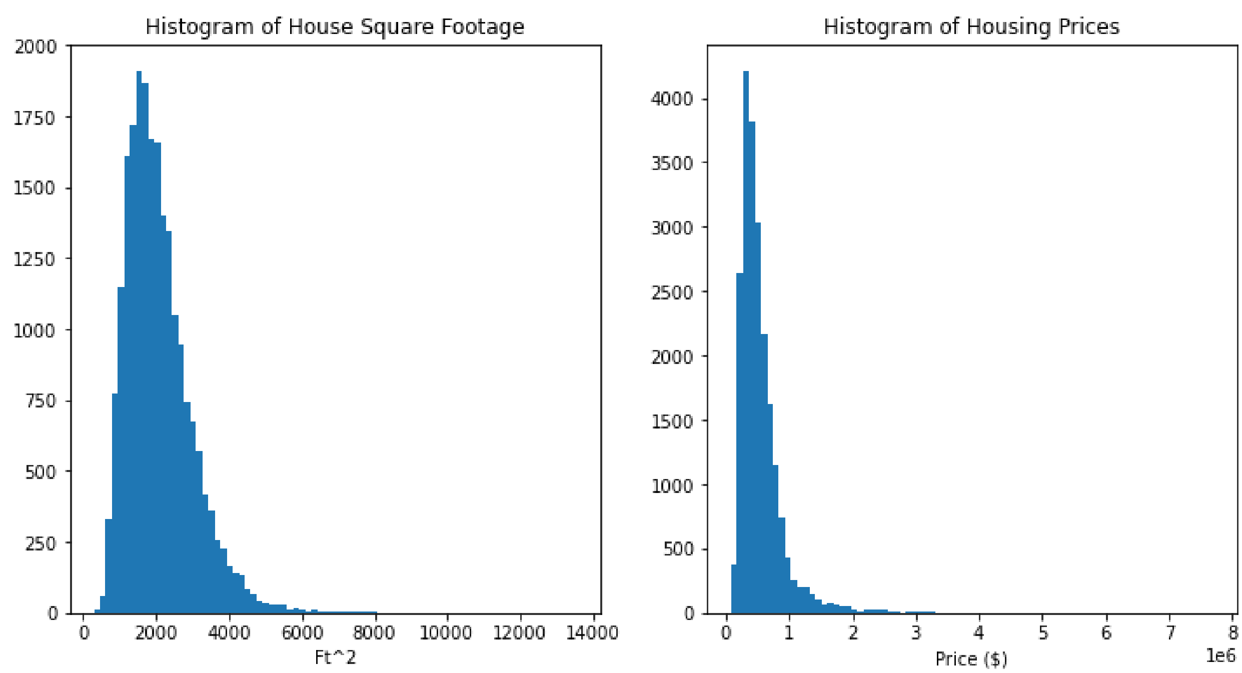
output:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04
mean	4.580302e+09	5.401822e+05	3.370842	2.114757	2079.899736	1.510697e+04
std	2.876566e+09	3.673622e+05	0.930062	0.770163	918.440897	4.142051e+04
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06

```
fig = plt.figure(figsize=(12, 6))
sqft = fig.add_subplot(121)
cost = fig.add_subplot(122)
sqft.hist(df.sqft_living, bins=80)
sqft.set_xlabel('Ft^2')
sqft.set_title("Histogram of House Square Footage")
cost.hist(df.price, bins=80)
cost.set_xlabel('Price ($)')
cost.set_title("Histogram of Housing Prices")
plt.show()
```

Output:



 # When you code to produce a linear regression summary with OLS with only two variables this will be the formula that you use:

Reg = ols('Dependent variable ~ independent variable(s), dataframe).fit()

print(Reg.summary())

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
m = ols('price ~ sqft_living',df).fit()
print (m.summary())
```

output:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:
0.493
Model:                            OLS   Adj. R-squared:
0.493
Method:                 Least Squares   F-statistic:
2.100e+04
Date:                Fri, 05 May 2023   Prob (F-statistic):
0.00
Time:                        11:05:31   Log-Likelihood:            -
3.0028e+05
No. Observations:               21613   AIC:
6.006e+05
Df Residuals:                   21611   BIC:
6.006e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                  coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
Intercept   -4.387e+04   4405.455     -9.958      0.000   -5.25e+04   -
3.52e+04
sqft_living   280.8067      1.938    144.924      0.000     277.009
284.605
==============================================================================
Omnibus:                    14815.593   Durbin-Watson:
1.983
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
543533.863
Skew:                           2.820   Prob(JB):
0.00
Kurtosis:                      26.911   Cond. No.
5.63e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 5.63e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

 # It is easy to adjust this formula to include more than one independent variable, simply follow the formula:
Reg = ols('Dependent variable ~ivar1 + ivar2 + ivar3… + ivarN, dataframe).fit()
print(Reg.summary())

```
m = ols('price ~ sqft_living + bedrooms + grade + condition',df).fit()
print (m.summary())
```

output:

```
                   OLS Regression Results
=================================================================
Dep. Variable:                  price   R-squared:
0.555
Model:                            OLS   Adj. R-squared:
0.555
Method:                 Least Squares   F-statistic:
6749.
Date:                Fri, 23 Sep 2016   Prob (F-statistic):
0.00
Time:                        15:11:41   Log-Likelihood:
-2.9884e+05
No. Observations:               21613   AIC:
5.977e+05
Df Residuals:                   21608   BIC:
5.977e+05
Df Model:                           4
Covariance Type:            nonrobust
=================================================================
                  coef    std err          t      P>|t|
[95.0% Conf. Int.]
--------------------------------------------------------------
Intercept   -7.398e+05   1.81e+04    -40.855      0.000     -
7.75e+05 -7.04e+05
sqft_living   212.3034      3.249     65.353      0.000
205.936   218.671
bedrooms    -4.568e+04   2222.205    -20.555      0.000        -
5e+04 -4.13e+04
grade        1.001e+05   2241.553     44.673      0.000
9.57e+04  1.05e+05
condition    6.615e+04   2598.352     25.457      0.000
6.11e+04  7.12e+04
=================================================================
Omnibus:                    16773.778   Durbin-Watson:
1.988
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
973426.793
Skew:                           3.249   Prob(JB):
0.00
```

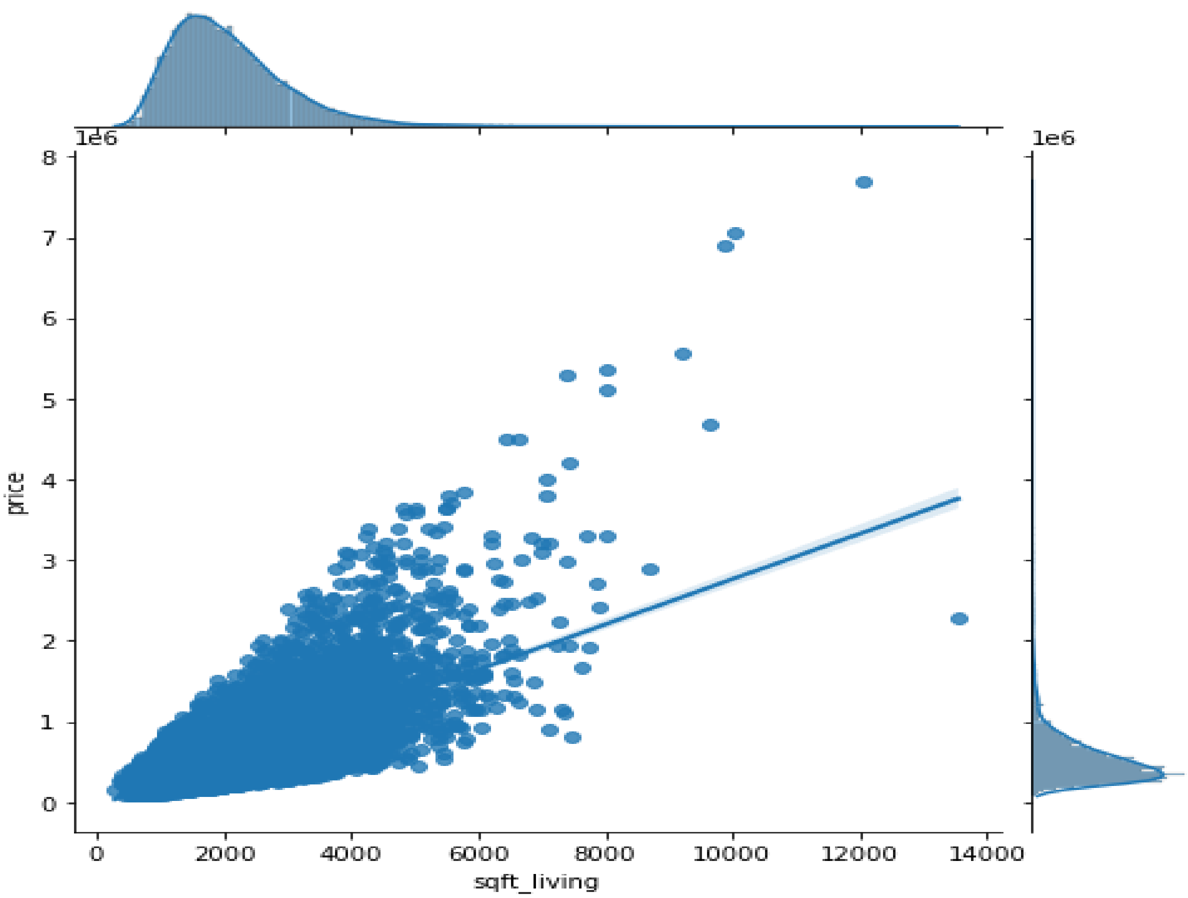
```
Kurtosis:                      35.229   Cond. No.
2.50e+04
================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the
errors is correctly specified.
[2] The condition number is large, 2.5e+04. This might indicate
that there are
strong multicollinearity or other numerical problems.
```

 # Visualizing the regression results:

sns.jointplot(x="sqft_living", y="price", data=df, kind = 'reg',fit_reg= True, size = 7)
plt.show()

output:



EX:5

Build a classification model using Decision Tree algorithm on iris dataset

import pandas as pd
import numpy as np

```
import matplotlib.pyplot as plt
import seaborn as sns

Iris_data = pd.read_csv("E:\Iris.csv")

Iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
Iris_data.head(10)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

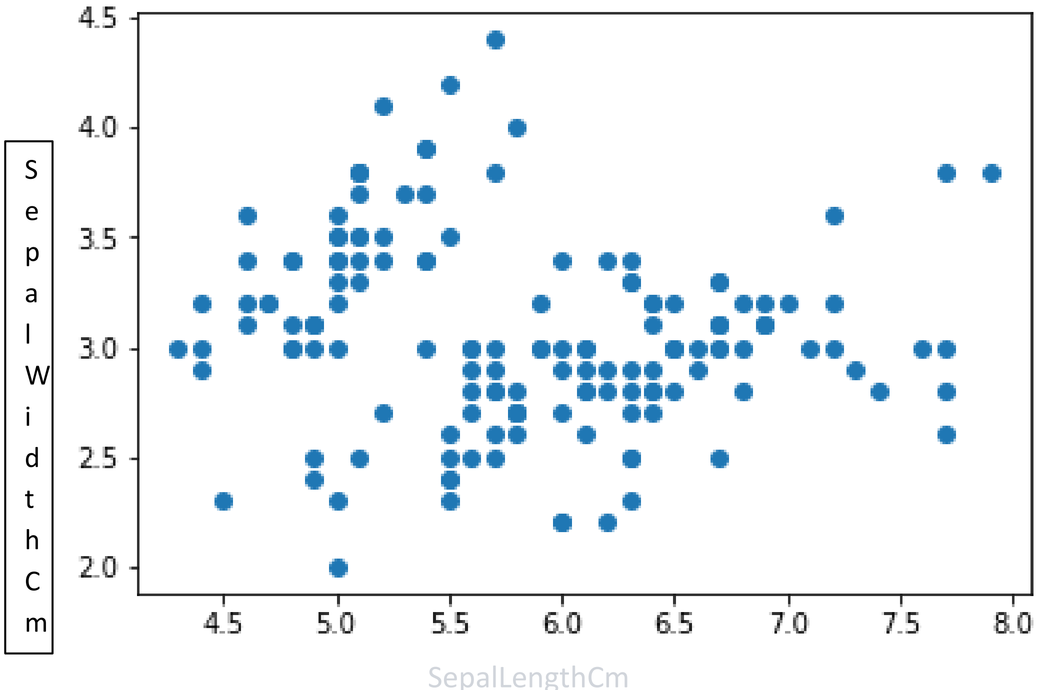
Iris_data.describe()

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

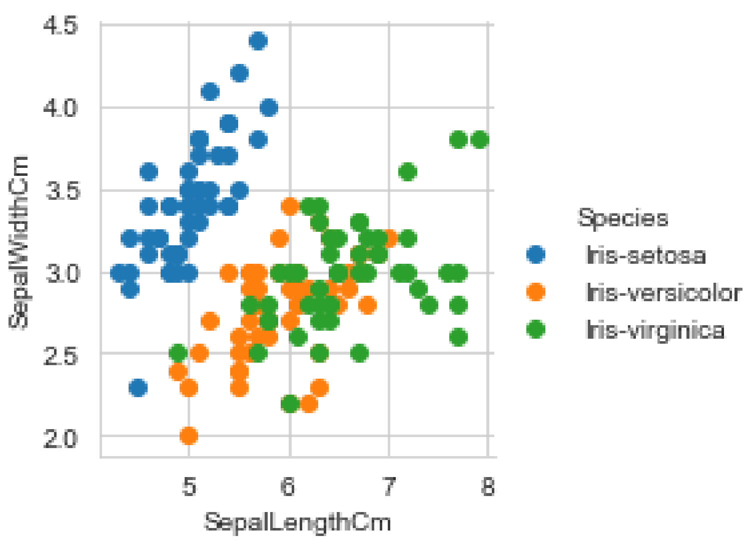
Iris_data.Species.value_counts()

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

```
plt.scatter(Iris_data['SepalLengthCm'],Iris_data['SepalWidthCm'])
plt.show()
```

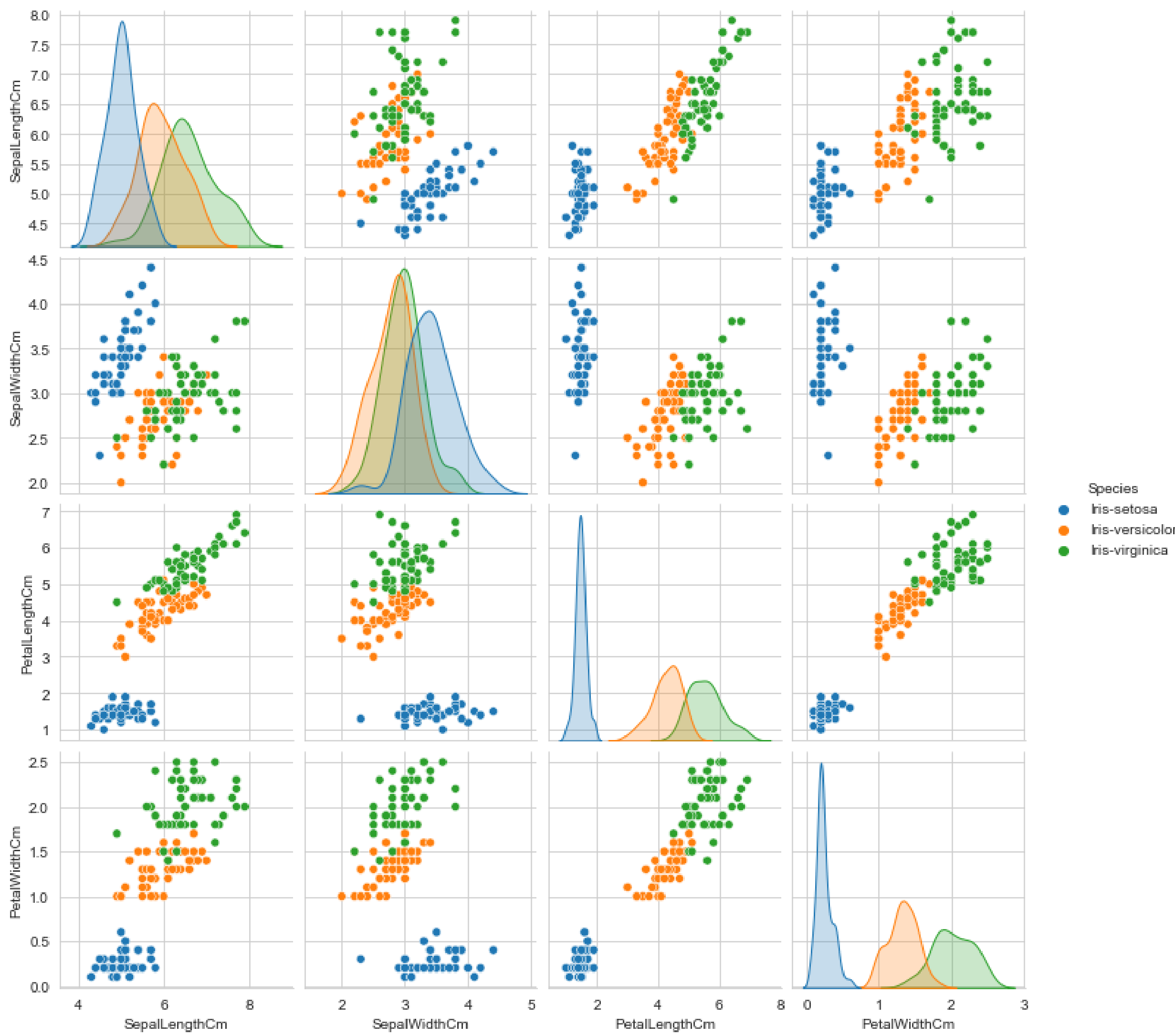



```
sns.set_style('whitegrid')
sns.FacetGrid(Iris_data,hue ='Species') \
  .map(plt.scatter,'SepalLengthCm','SepalWidthCm') \
  .add_legend() plt.show()
```




```
sns.pairplot(Iris_data.drop(['Id'],axis=1),hue='Species')
```

```
plt.show()
```



EX:6

Apply Naïve Bayes Classification algorithm on any dataset

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
import matplotlib.pyplot as plt
df = pd.read_csv('D:\loan_data.csv')
df.head(2)
```

OUTPUT:

credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	credit_card	0.1071	228.22	11.082143	14.29	707	5639.958333	33623	76.7	0	0	0	0

df.info()

OUTPUT:

RangeIndex: 9578 entries, 0 to 9577

Data columns (total 14 columns):

 # Column Non-Null Count Dtype

--- ------ -------------- -----

0 credit.policy 9578 non-null int64

1 purpose 9578 non-null object

2 int.rate 9578 non-null float64

3 installment 9578 non-null float64

4 log.annual.inc 9578 non-null float64

5 dti 9578 non-null float64

6 fico 9578 non-null int64

7 days.with.cr.line 9578 non-null float64

8 revol.bal 9578 non-null int64

9 revol.util 9578 non-null float64

10 inq.last.6mths 9578 non-null int64

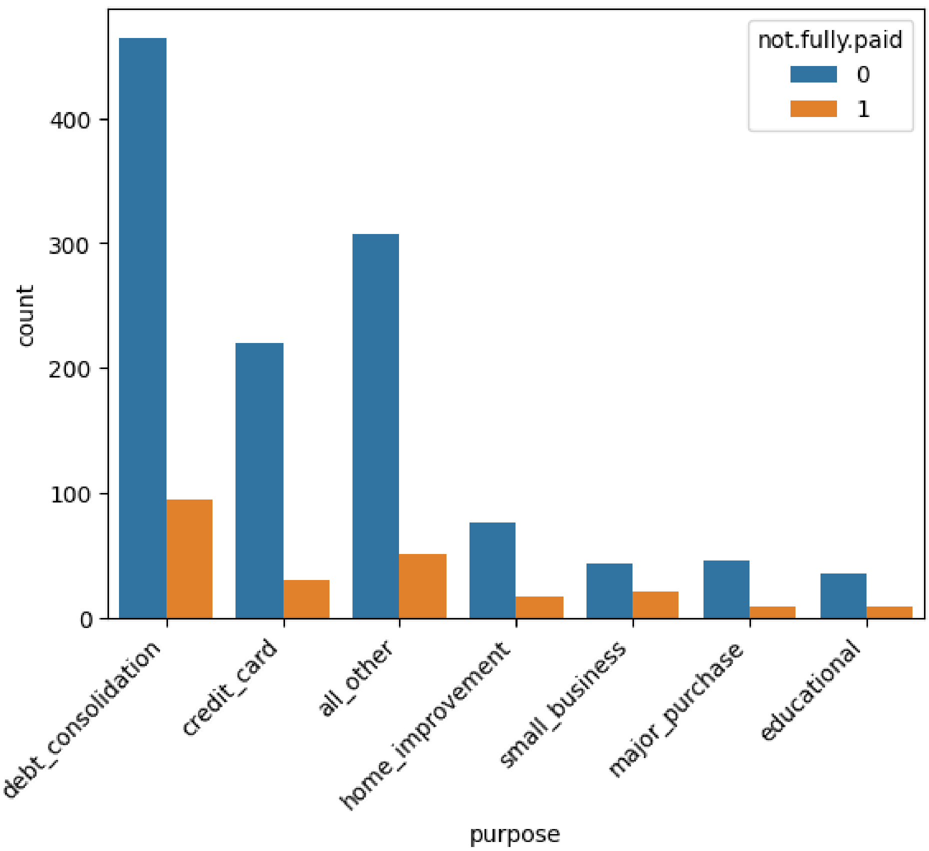
11 delinq.2yrs 9578 non-null int64

12 pub.rec   9578 non-null   int64
13 not.fully.paid   9578 non-null   int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(data=df,x='purpose',hue='not.fully.paid')
plt.xticks(rotation=45, ha='right');
```

O**UTPUT:**




```
pre_df = pd.get_dummies(df,columns=['purpose'],drop_first=True)
pre_df.head(1)
```

OUTPUT:

credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid	purpose_credit_card	purpose_debt_consolidation	purpose_educational	purpose_home_improvement	purpose_major_purchase
1	0.1189	829.1	11.350407	19.48	737	5639.958333	28854	**52.1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**

```
from sklearn.model_selection import train_test_split
X = pre_df.drop('not.fully.paid', axis=1)
y = pre_df['not.fully.paid']
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.33, random_state=125)
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train);
from sklearn.metrics import (
   accuracy_score,
   confusion_matrix,
   ConfusionMatrixDisplay,
   f1_score,
   classification_report,)
y_pred = model.predict(X_test)
accuray = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")
print("Accuracy:", accuray)
print("F1 Score:", f1)
```

OUTPUT:

```
Accuracy: 0.7923728813559322
F1 Score: 0.8251441989705616
labels = ["Fully Paid", "Not fully Paid"]
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=labels)
disp.plot();
```

OUTPUT:

EX:7

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

Use this to read data from the csv file on local system.

df = pd.read_csv('retail_data.csv') ## Print first 10 rows

df.head(10)

OUTPUT:

Column1	0	1	2	3	4	5	6	7
0	NaN	Bread	Wine	Eggs	Meat	Cheese	Pencil	Diaper
1	NaN	Bread	Cheese	Meat	Diaper	Wine	Milk	Pencil
2	NaN	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
3	NaN	Cheese	Meat	Eggs	Milk	Wine	NaN	NaN
4	NaN	Meat	Pencil	Wine	NaN	NaN	NaN	NaN
5	NaN	Eggs	Bread	Wine	Pencil	Milk	Diaper	Bagel

6	NaN	Wine	Pencil	Eggs	Cheese	NaN	NaN	NaN
7	NaN	Bagel	Bread	Milk	Pencil	Diaper	NaN	NaN
8	NaN	Bread	Diaper	Cheese	Milk	Wine	Eggs	NaN

```
items = set()
for col in df:
    items.update(df[col].unique())
print(items)
```

OUTPUT:

{'Bread', 'Cheese', 'Meat', 'Eggs', 'Wine', 'Bagel', 'Pencil', 'Diaper', 'Milk']}

```
itemset = set(items)
encoded_vals = []
for index, row in df.iterrows():
    rowset = set(row)
   labels = {}
    uncommons = list(itemset - rowset)
    commons = list(itemset.intersection(rowset))
    for uc in uncommons:
        labels[uc] = 0
    for com in commons:
        labels[com] = 1
    encoded_vals.append(labels)
encoded_vals[0]ohe_df = pd.DataFrame(encoded_vals)
```

Applying Apriori

apriori module from mlxtend library provides fast and efficient apriori implementation.

apriori(df, min_support=0.5, use_colnames=False, max_len=None, verbose=0,low_memory=False)

Parameters

- df : One-Hot-Encoded DataFrame or DataFrame that has 0 and 1 or True and False as values

- min_support : Floating point value between 0 and 1 that indicates the minimum support required for an itemset to be selected.
  # of observation with item / total observation# of observation with item / total observation
- use_colnames : This allows to preserve column names for itemset making it more readable.
- max_len : Max length of itemset generated. If not set, all possible lengths are evaluated.
- verbose : Shows the number of iterations if >= 1 and low_memory is True. If =1 and low_memory is False , shows the number of combinations.
- low_memory :
- If True, uses an iterator to search for combinations above min_support. Note that while low_memory=True should only be used for large dataset if memory resources are limited, because this implementation is approx. 3–6x slower than the default.

```
from mlxtend.frequent_patterns import apriori
freq_items = apriori(ohe_df, min_support=0.2, use_colnames=True, verbose=1)
freq_items.head(7)
```

OUTPUT:

support	itemsets	
0	0.869841	(nan)
1	0.425397	(Bagel)
2	0.501587	(Milk)
3	0.47619	(Meat)
4	0.501587	(Cheese)
5	0.438095	(Wine)
6	0.504762	(Bread)

```
from mlxtend.frequent_patterns import association_rules
rules = association_rules(freq_items, metric="confidence", min_threshold=0.6)
rules.head()
```

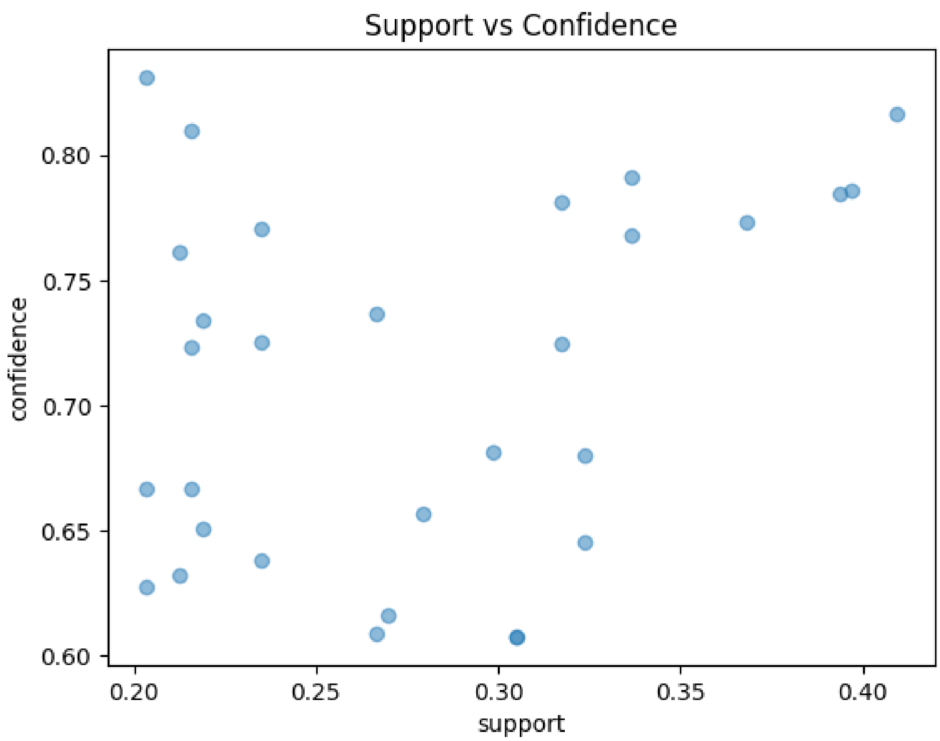
OUTPUT:

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(Bagel)	(nan)	0.425397	0.869841	0.336508	0.791045	0.909413	-0.03352	0.622902	-0.147743
1	(Milk)	(nan)	0.501587	0.869841	0.409524	0.816456	0.938626	-0.026778	0.709141	-0.115976
2	(Meat)	(nan)	0.47619	0.869841	0.368254	0.773333	0.889051	-0.045956	0.57423	-0.192405
3	(Cheese)	(nan)	0.501587	0.869841	0.393651	0.78481	0.902245	-0.042651	0.604855	-0.178565
4	(Wine)	(nan)	0.438095	0.869841	0.31746	0.724638	0.833069	-0.063613	0.472682	-0.262869

1. Support vs Confidence

```
import matplotlib.pyplot as plt
plt.scatter(rules['support'], rules['confidence'], alpha=0.5)
plt.xlabel('support')
plt.ylabel('confidence')
plt.title('Support vs Confidence')
plt.show()
```

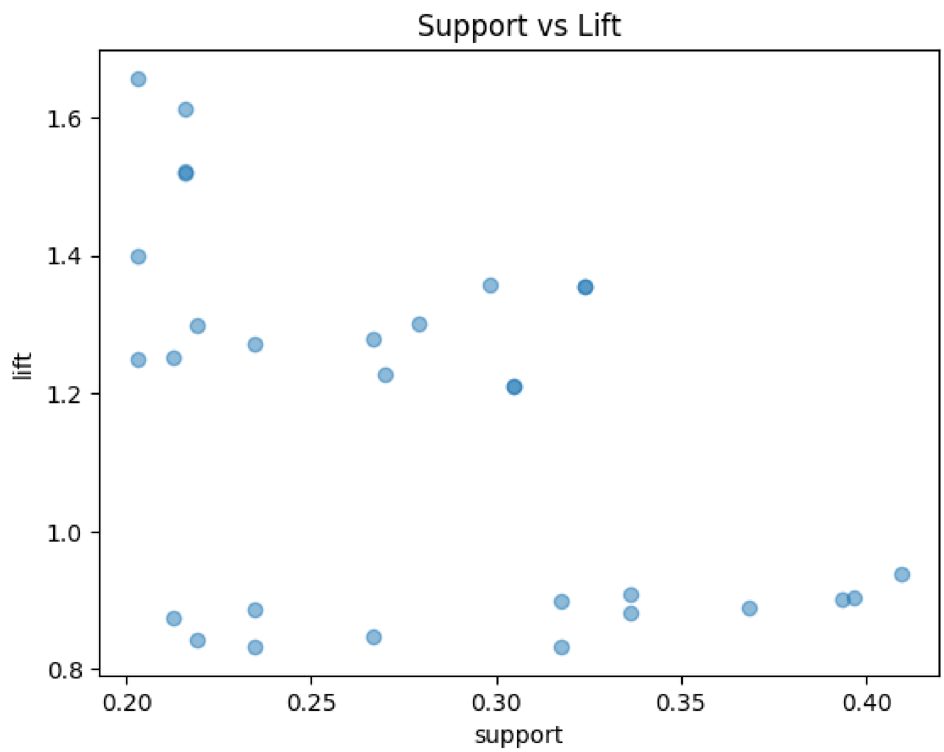
OUTPUT:



Support vs Lift

```
plt.scatter(rules['support'], rules['lift'], alpha=0.5)
plt.xlabel('support')
plt.ylabel('lift')
plt.title('Support vs Lift')
plt.show()
```

OUTPUT:



Lift vs Confidence

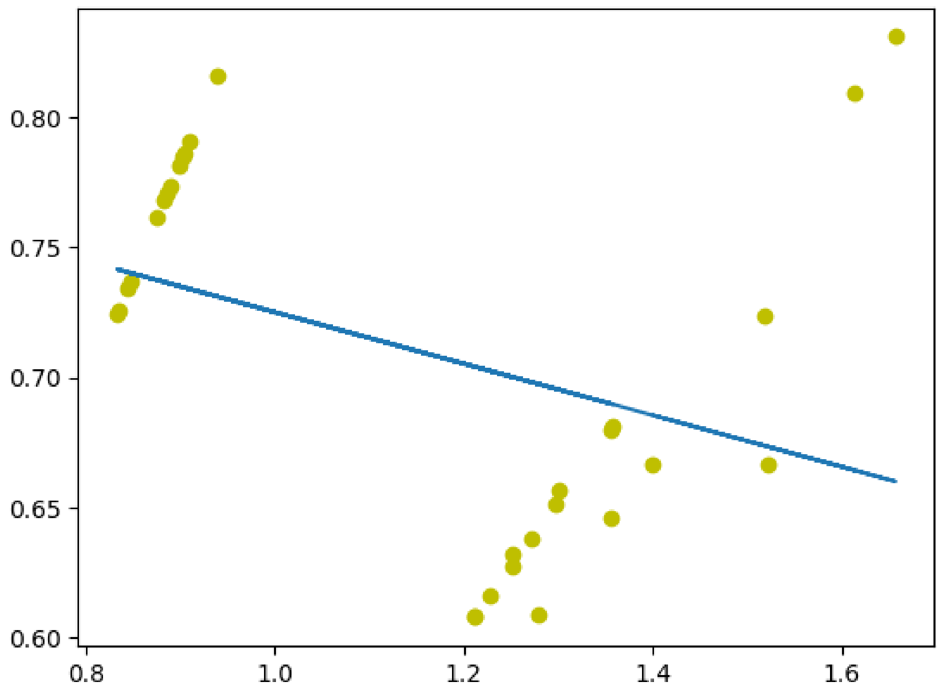
import numpy as np

fit = np.polyfit(rules['lift'], rules['confidence'], 1)

fit_fn = np.poly1d(fit)

plt.plot(rules['lift'], rules['confidence'], 'yo', rules['lift'], fit_fn(rules['lift']))

OUTPUT:



EX:8

Apply K- Means clustering algorithm on any dataset.

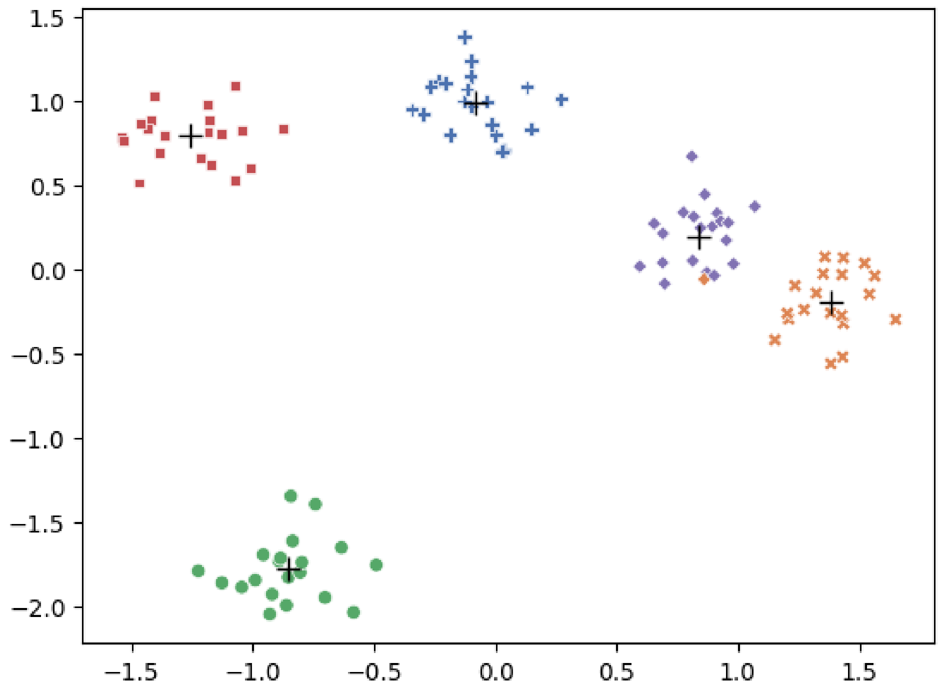
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from numpy.random import uniform
from sklearn.datasets import make_blobs
import seaborn as sns
import random
def euclidean(point, data):
 """
 Euclidean distance between point & data.
 Point has dimensions (m,), data has dimensions (n,m), and output will
be of size (n,).
 """
return np.sqrt(np.sum((point - data)**2, axis=1))
class KMeans:
```

```
def __init__(self, n_clusters=8, max_iter=300):
                  self.n_clusters = n_clusters
                  self.max_iter = max_iter
def fit(self, X_train):
# Initialize the centroids, using the "k-means++" method, where a
random datapoint is selected as the first,
 # then the rest are initialized w/ probabilities proportional to their
distances to the first
 # Pick a random point from train data for first centroid
 self.centroids = [random.choice(X_train)]
for _ in range(self.n_clusters-1):
 # Calculate distances from points to the centroids
 dists = np.sum([euclidean(centroid, X_train) for centroid in
self.centroids], axis=0)
 # Normalize the distances
 dists /= np.sum(dists)
 # Choose remaining points based on their distances
 new_centroid_idx, = np.random.choice(range(len(X_train)), size=1,
p=dists)
 self.centroids += [X_train[new_centroid_idx]]
# This initial method of randomly selecting centroid starts is less
effective
 # min_, max_ = np.min(X_train, axis=0), np.max(X_train, axis=0)
 # self.centroids = [uniform(min_, max_) for _ in range(self.n_clusters)]
# Iterate, adjusting centroids until converged or until passed max_iter
 iteration = 0
 prev_centroids = None
 while np.not_equal(self.centroids, prev_centroids).any() and iteration
```

```
<self.max_iter:
 # Sort each datapoint, assigning to nearest centroid
 sorted_points = [[] for _ in range(self.n_clusters)]
 for x in X_train:
 dists = euclidean(x, self.centroids)
 centroid_idx = np.argmin(dists)
 sorted_points[centroid_idx].append(x)
# Push current centroids to previous, reassign centroids as mean of the
points belonging to them
 prev_centroids = self.centroids
 self.centroids = [np.mean(cluster, axis=0) for cluster in sorted_points]
 for i, centroid in enumerate(self.centroids):
 if np.isnan(centroid).any(): # Catch any np.nans, resulting from a
centroid having no points
 self.centroids[i] = prev_centroids[i]
 iteration += 1
def evaluate(self, X):
 centroids = []
 centroid_idxs = []
 for x in X:
 dists = euclidean(x, self.centroids)
 centroid_idx = np.argmin(dists)
 centroids.append(self.centroids[centroid_idx])
 centroid_idxs.append(centroid_idx)
return centroids, centroid_idxs
# Create a dataset of 2D distributions
centers = 5
X_train, true_labels = make_blobs(n_samples=100, centers=centers,
```

```
random_state=42)
X_train = StandardScaler().fit_transform(X_train)
# Fit centroids to dataset
kmeans = KMeans(n_clusters=centers)
kmeans.fit(X_train)
# View results
class_centers, classification = kmeans.evaluate(X_train)
sns.scatterplot(x=[X[0] for X in X_train],
 y=[X[1] for X in X_train],
 hue=true_labels,
 style=classification,
 palette="deep",
 legend=None
 )
plt.plot([x for x, _ in kmeans.centroids],
 [y for _, y in kmeans.centroids],
 'k+',
 markersize=10,
 )
plt.show()
```

OUTPUT:



On the x-axis, there are the values of the first dimension of the dataset (the first feature), and on the y-axis, there are the values of the second dimension of the dataset (the second feature).In this , the x-axis and y-axis represent the values of the two-dimensional dataset used for the clustering algorithm. The scatter plot shows the clusters created by the KMeans algorithm, with each point colored according to its true label (centers) and classified label (classification). The centroids of each cluster are also plotted as black plus signs (+) on the plot.

EX:9

Apply Hierarchical Clustering algorithm on any dataset.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
data = pd.read_csv('Wholesale customers data.csv')
data.head()

OUTPUT:

Channel Regio Fresh Milk Grocery Froze Detergents Delicass

		n				n	_Paper	en
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	**2**	**3**	**22615**	**5410**	**7198**	**3915**	**1777**	**5185**

```
from sklearn.preprocessing import normalize
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
data_scaled.head()
```

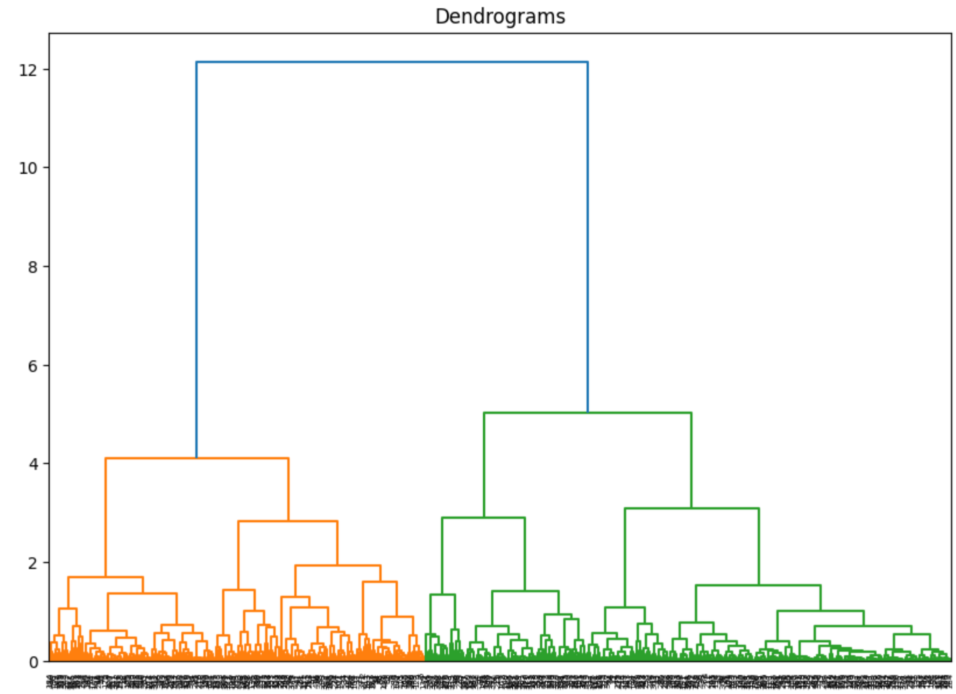
OUTPUT:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	0.000112	0.000168	0.708333	0.539874	0.422741	0.011965	0.149505	0.074809
1	0.000125	0.000188	0.442198	0.614704	0.59954	0.110409	0.206342	0.111286
2	0.000125	0.000187	0.396552	0.549792	0.479632	0.150119	0.219467	0.489619
3	0.000065	0.000194	0.856837	0.077254	0.27265	0.413659	0.032749	0.115494
4	0.000079	0.000119	0.895416	0.214203	0.284997	0.15501	0.070358	0.205294

```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
```

OUTPUT:

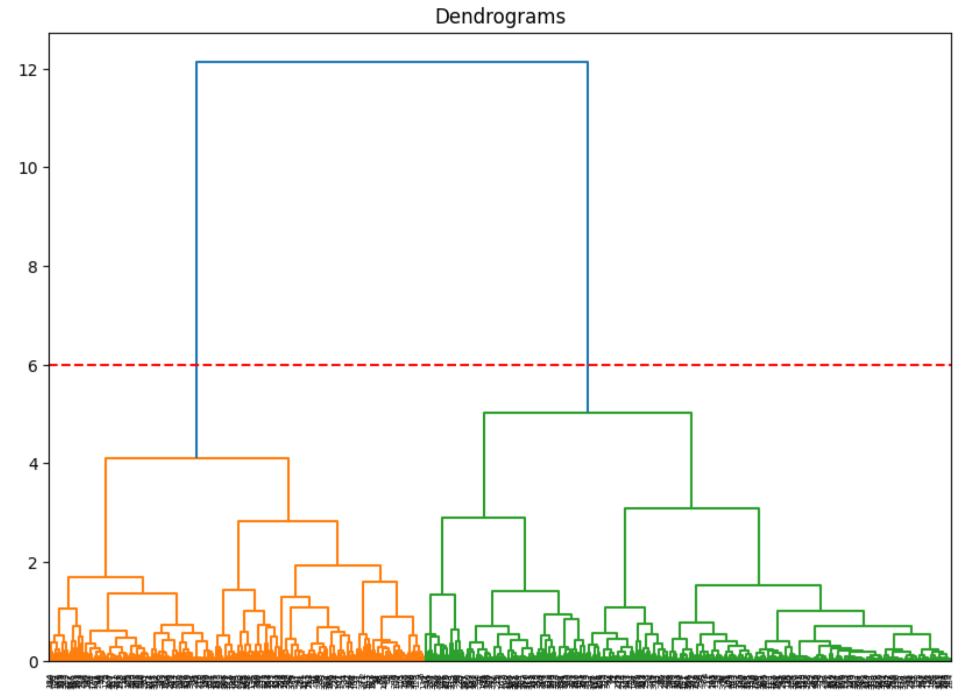
On the x-axis, the individual data points are represented by their index numbers. On the y-axis, the height of the vertical lines shows the distance between the data points or clusters being merged. The longer the vertical line, the greater the distance between the data points or clusters. By looking at the dendrogram, we can determine the optimal number of clusters by finding the longest vertical line that does not cross any horizontal line. This gives us the optimal number of clusters for our dataset.



plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
plt.axhline(y=6, color='r', linestyle='--')

OUTPUT:

On the x-axis of the plot, we have the individual data points, while on the y-axis, we have the distance between the clusters being merged. The dendrogram shows how the data points are clustered together at different distances. The horizontal line at y=6 is a threshold line that can be used to determine the number of clusters to form. The number of clusters is determined by counting the number of vertical lines that are crossed by the threshold line. In this case, we would have 2 clusters since the threshold line crosses two vertical lines.



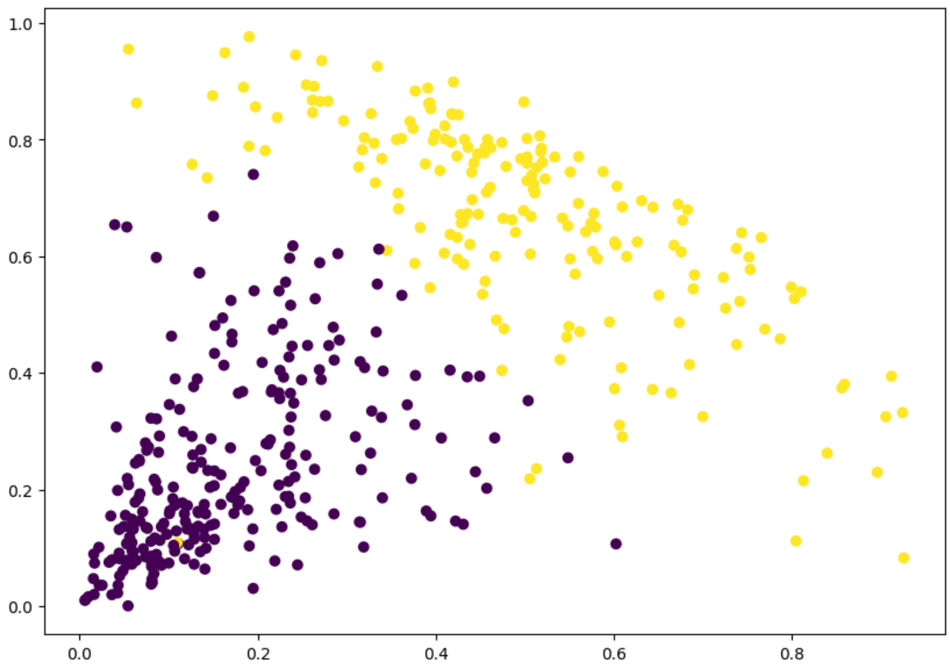

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
cluster.fit_predict(data_scaled)
```

OUTPUT:

```
array([1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
     0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
     1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
     1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
     0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
     0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
     0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
     0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
     0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
     0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0,
     0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
     0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
     1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
     0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
     0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1,
```

```
    1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
    0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
    1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1,
    1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1],
   dtype=int64)
plt.figure(figsize=(10, 7))
plt.scatter(data_scaled['Milk'], data_scaled['Grocery'], c=cluster.labels_)
```

OUTPUT:



On the x-axis, we have the 'Milk' variable and on the y-axis, we have the 'Grocery' variable. The scatter plot represents the distribution of data points with their respective cluster labels.

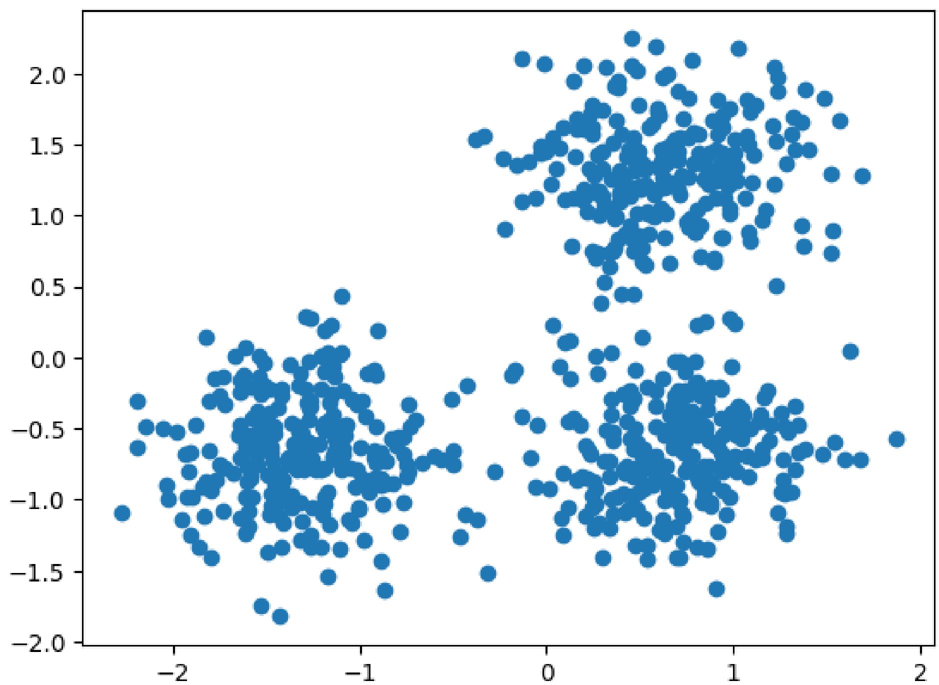
EX:10

Apply DBSCAN clustering algorithm on any dataset.

```
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
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)
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1])
plt.show()
```

OUTPUT:




```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
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_)
```

OUTPUT:

Estimated number of clusters: 3

Estimated number of noise points: 18

```
print(f"Homogeneity: {metrics.homogeneity_score(labels_true, labels):.3f}")
print(f"Completeness: {metrics.completeness_score(labels_true, labels):.3f}")
print(f"V-measure: {metrics.v_measure_score(labels_true, labels):.3f}")
print(f"Adjusted Rand Index: {metrics.adjusted_rand_score(labels_true,
labels):.3f}")
print(
   "Adjusted Mutual Information:"
   f" {metrics.adjusted_mutual_info_score(labels_true, labels):.3f}"
)
print(f"Silhouette Coefficient: {metrics.silhouette_score(X, labels):.3f}")
```

OUTPUT:

Homogeneity: 0.953

Completeness: 0.883

V-measure: 0.917

Adjusted Rand Index: 0.952

Adjusted Mutual Information: 0.916

Silhouette Coefficient: 0.626

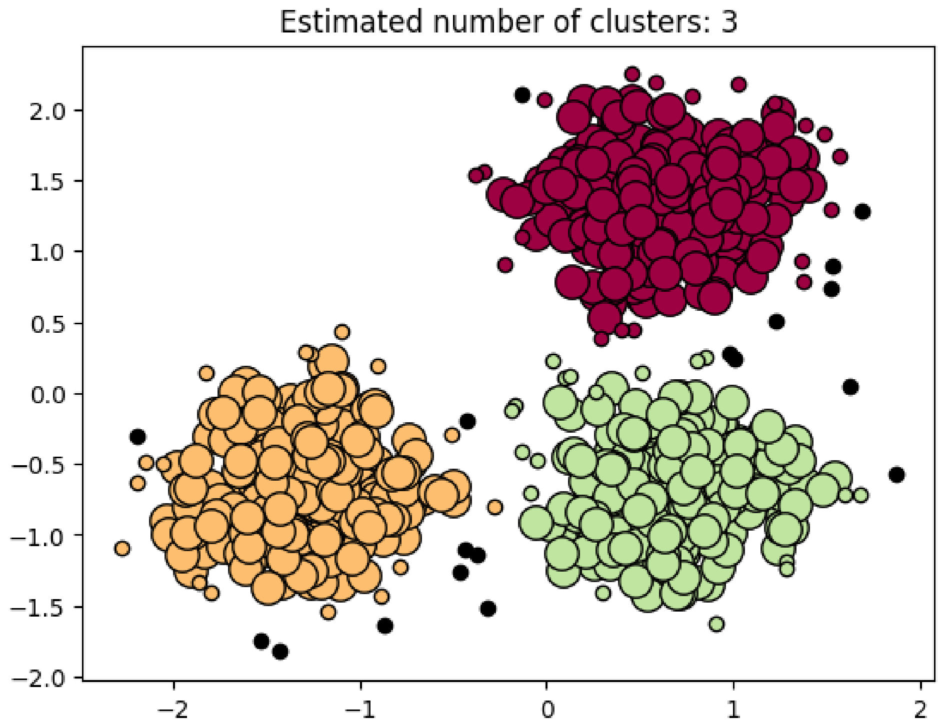
```
unique_labels = set(labels)
core_samples_mask = np.zeros_like(labels, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True

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(f"Estimated number of clusters: {n_clusters_}")
plt.show()
```

OUTPUT:



Estimated number of clusters: 3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2