1. Calculate the mean and standard deviation.

```
# Mean deviation of Elements
# Using loop + mean() + abs()
from statistics import mean

# initializing list
test_list = [7, 5, 1, 2, 10, 3]

# printing original lists
print("The original list is : " + str(test_list))

res = []

# getting mean
mean_val = mean(test_list)

for ele in test_list:

    # getting deviation
    res.append(abs(ele - mean_val))

# printing result
print("Mean deviations : " + str(res))
```

OUTPUT:-

The original list is : [7, 5, 1, 2, 10, 3] Mean deviations : [2.333333333333333, 0.33333333333333304, 3.666666666666667, 2.666666666666667, 5.333333333333333, 1.666666666666667]

2. Read the CSV file.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

df=pd.read_csv('My_Income.csv')
df.head()
```

OUTPUT

id	Name	Salary	Age
0	1	ABC	1200	22
1	2	PQR	21000	26
2	3	PPP	877	54
3	4	AAA	9456	47
4	5	BBB	15232	14

3. Perform data filtering, and calculate aggregate statistics.

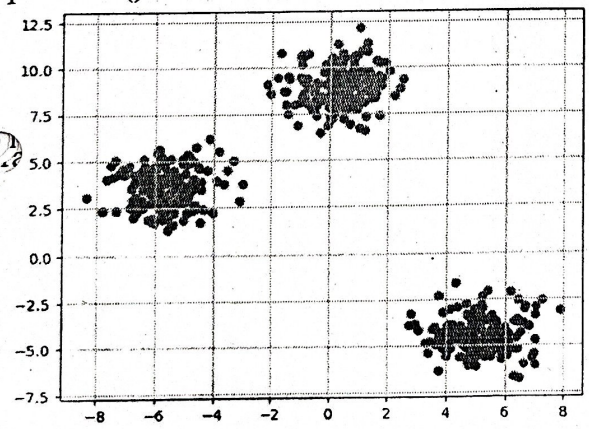
```
import pandas as pd
import numpy as np

# Creating DataFrame
data = {
    'A': [10, 20, 30, 40, 50],
    'B': [15.5, 20.5, np.nan, 35.0, 45.5],
    'C': ['a', 'b', 'a', 'b', 'c'],
    'D': [1, 2, 3, 4, 5],
    'E': [100, 150, 200, 250, 300]
}

df = pd.DataFrame(data)
numerical_df =
df.select_dtypes(include=['number'])
mean_values = numerical_df.mean()
print(mean_values)
# Converting column C to numeric values
df['C_numeric'] = df['C'].map({'a': 1, 'b': 2, 'c': 3})

# You can now use the new column in operations
# Calculate the mean of only numerical columns,
including the new 'C_numeric' column
mean_values =
df.select_dtypes(include=['number']).mean()
print(mean_values)

numerical_df =
df.select_dtypes(include=['number'])
median_values = numerical_df.median()
print(median_values)

sum_values = df.sum()
print(sum_values)

min_values = df.min()
print(min_values)

max_values = df.max()
print(max_values)

numerical_df =
df.select_dtypes(include=['number'])
std_values = numerical_df.std()
print(std_values)

numerical_df =
df.select_dtypes(include=['number'])
var_values = numerical_df.var()
print(var_values)

count_values = df.count()
print(count_values)
 mode_values = df.mode()
print(mode_values)
```

4. Calculate total sales by month.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df=pd.read_csv('sales_data.csv.zip')
df.head()
df.groupby('Month')['Revenue'].sum()
```

```
Revenue
Month
April    7602750
August5711193
December     9086931
February     6834583
January      7005895
July     5721459
June     9043008
March 7347164
May     8836763
November     6244298
October      5995079
September    5841885
```

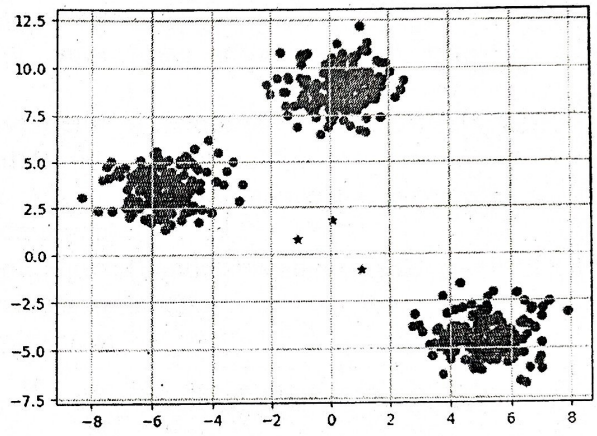
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X,y = make_blobs(n_samples = 500,n_features =
2,centers = 3,random_state = 23)

fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0],X[:,1])
plt.show()
```




```
k = 3

clusters = {}
np.random.seed(23)

for idx in range(k):
    center =
2*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }
    clusters[idx] = cluster

clusters
```

```
{0: {'center': array([0.06919154, 1.78785042]),
'points': []},
 1: {'center': array([ 1.06183904, -0.87041662]),
'points': []},
 2: {'center': array([-1.11581855,  0.74488834]),
'points': []}}
```

```
plt.scatter(X[:,0],X[:,1])
plt.grid(True)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '*',c =
'red')
plt.show()
```

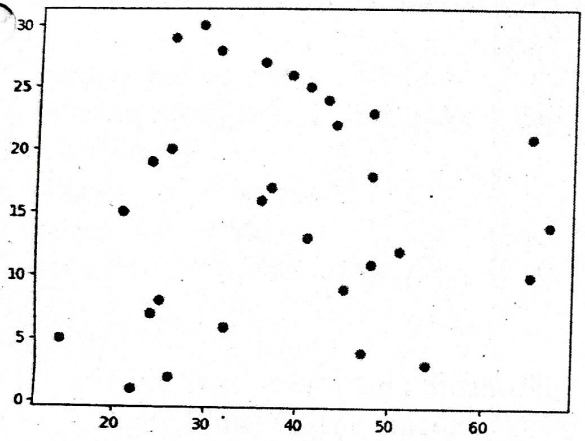


5. Implement the Clustering using K-means.

```
import numpy as nm
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
df= pd.read_csv('My_Income.csv')
print(df.columns)
df.head()
```

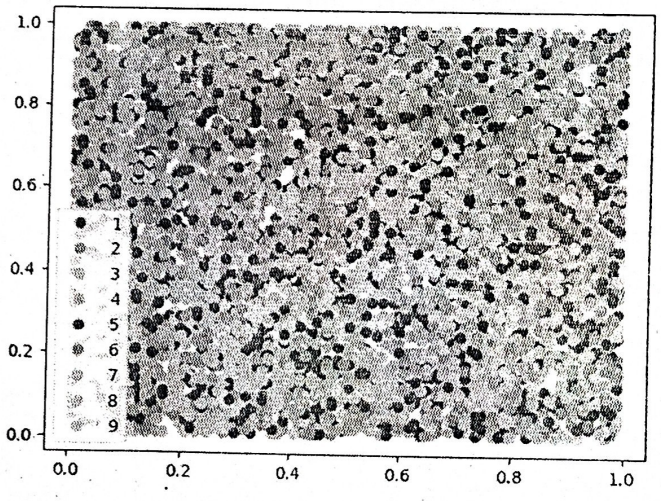
id	Name	Salary	Age
0	1	ABC	1200	22
1	2	PQR	21000	26
2	3	PPP	877	54
3	4	AAA	9456	47
4	5	BBB	15232	14

```
plt.scatter(df['Age'],df['id'])
#plt.xlable('Age')
#plt.ylabe('id')
plt.show()
```



9. Visualize the result of the clustering and compare.

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

# Defining the data_normalized DataFrame before using it
d={'col1': [i/100 for i in random.choices(range(1,100), k=7315)],
    'col2':[i/100 for i in random.choices(range(1,100), k=7315)],
    'y_kmeans':random.choices(range(1,10), k=7315)}
data_normalized = pd.DataFrame(d)

u_labels = np.unique(data_normalized['y_kmeans']).tolist()

scatter = plt.scatter(data_normalized['col1'], data_normalized['col2'],
        c=data_normalized['y_kmeans'], cmap='tab20')
plt.legend(handles=scatter.legend_elements()[0], labels=u_labels)
plt.show()
```



6. Classification using Random Forest.

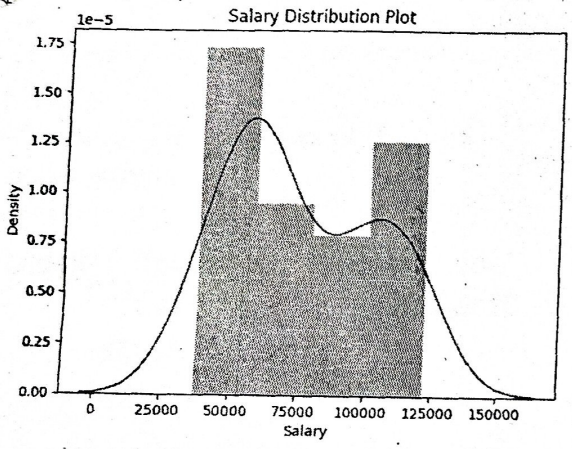
```
import pandas as pd
from sklearn.model_selection import
train_test_split
from sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import accuracy_score,
classification_report
import warnings
warnings.filterwarnings('ignore')

# Corrected URL for the dataset
url =
"https://raw.githubusercontent.com/datasciencedoj
o/datasets/master/titanic.csv"
titanic_data = pd.read_csv(url)

# Drop rows with missing 'Survived' values
titanic_data =
titanic_data.dropna(subset=['Survived'])

# Features and target variable
X = titanic_data[['Pclass', 'Sex', 'Age', 'SibSp',
'Parch', 'Fare']]
y = titanic_data['Survived']
# Encode 'Sex' column
X.loc[:, 'Sex'] = X['Sex'].map({'female': 0, 'male':
1})

# Fill missing 'Age' values with the median
X.loc[:, 'Age'].fillna(X['Age'].median(),
inplace=True)

# Split data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize RandomForestClassifier
rf_classifier =
RandomForestClassifier(n_estimators=100,
random_state=42)

# Fit the classifier to the training data
rf_classifier.fit(X_train, y_train)

# Make predictions
y_pred = rf_classifier.predict(X_test)

# Calculate accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test,
y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n",
classification_rep)

# Sample prediction
sample = X_test.iloc[0:1]  # Keep as DataFrame to
match model input format
prediction = rf_classifier.predict(sample)

# Retrieve and display the sample
sample_dict = sample.iloc[0].to_dict()
print(f"\nSample Passenger: {sample_dict}")
print(f"Predicted Survival: {'Survived' if
prediction[0] == 1 else 'Did Not Survive'}")
```

```
Accuracy: 0.80

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       105
           1       0.77      0.73      0.75        74

    accuracy                           0.80       179
   macro avg       0.79      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179


Sample Passenger: {'Pclass': 3, 'Sex': 1, 'Age': 28.0,
'SibSp': 1, 'Parch': 1, 'Fare': 15.2458}
Predicted Survival: Did Not Survive
```

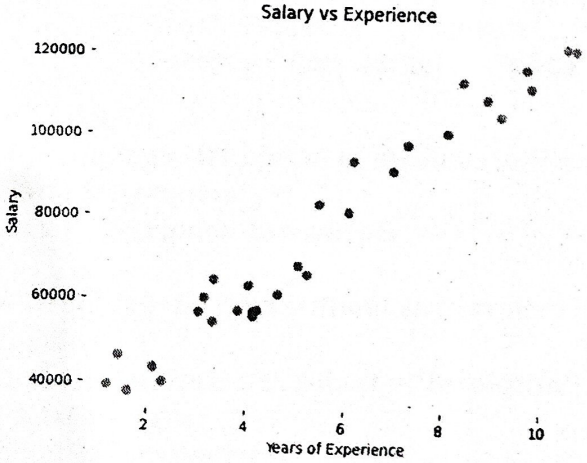
5

7. Regression Analysis using Linear Regression.

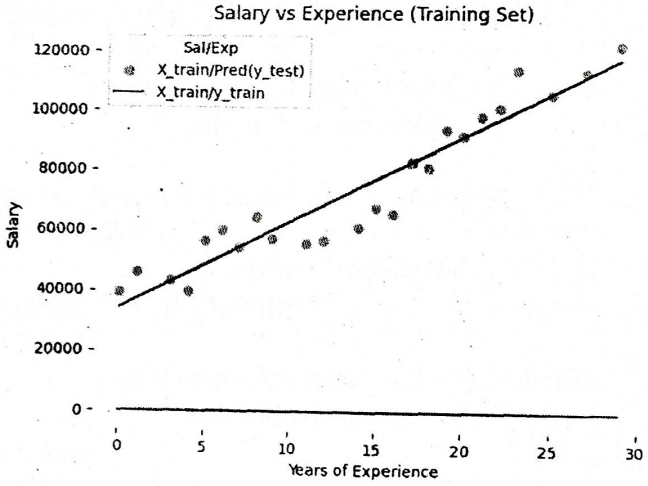
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import
train_test_split
from pandas.core.common import random_state
from sklearn.linear_model import
LinearRegression
# Get dataset
df_sal = pd.read_csv('Salary_dataset.csv')
df_sal.head()
# Describe data
df_sal.describe()
# Data distribution
plt.title('Salary Distribution Plot')
sns.distplot(df_sal['Salary'])
plt.show()
```




```
# Relationship between Salary and Experience
plt.scatter(df_sal['YearsExperience'],
df_sal['Salary'], color = 'lightcoral')
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()
```




```
# Splitting variables
X = df_sal.iloc[:, :1] # independent
y = df_sal.iloc[:, 1:] # dependent
# Splitting dataset into test/train
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size = 0.2, random_state
= 0)
# Regressor model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Prediction result
y_pred_test = regressor.predict(X_test)  #
predicted value of y_test
y_pred_train = regressor.predict(X_train)  #
predicted value of y_train
# Prediction on training set
plt.scatter(X_train, y_train['Salary'], color =
'lightcoral') # Changed this line
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)',
'X_train/y_train'], title = 'Sal/Exp', loc='best',
facecolor='white')
plt.box(False)
plt.show()
```




```
# Regressor coefficients and intercept
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept: {regressor.intercept_}')
Coefficient: [[3.06733871e-01]
 [2.84975857e+03]]
Intercept: [9.15181452e-01 3.44655481e+04]
```

8. Association Rule Mining using Apriori.

```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori,
association_rules
```

1 Code:

```
# Changing the working location to the location of
the file
cd C:\Users\Dev\Desktop\Kaggle\Apriori
Algorithm

# Loading the Data
data = pd.read_excel('Online_Retail.xlsx')
data.head()
```

Output:

	InvoiceNo	StockCode	Description	Quantity
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6
1	536365	71053	WHITE METAL LANTERN	6
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6

2 Code:

```
# Exploring the columns of the data
data.columns
```

Output:

```
Index(['InvoiceNo', 'StockCode', 'Descrip
       'UnitPrice', 'CustomerID', 'Countr
      dtype='object')
```

3 Code:

```
# Exploring the different regions of transactions
data.Country.unique()
```

Output:

```
array(['United Kingdom', 'France', 'Austral
       'Norway', 'EIRE', 'Switzerland', 'Sp
       'Italy', 'Belgium', 'Lithuania', 'Ja
       'Channel Islands', 'Denmark', 'Cypru
       'Israel', 'Finland', 'Bahrain', 'Gre
       'Lebanon', 'United Arab Emirates', '
       'Czech Republic', 'Canada', 'Unspeci
       'European Community', 'Malta', 'RSA'
```

4 Code:

```
# Stripping extra spaces in the description
data['Description'] =
data['Description'].str.strip()

# Dropping the rows without any invoice
number
data.dropna(axis = 0, subset =['InvoiceNo'],
inplace = True)
data['InvoiceNo'] =
data['InvoiceNo'].astype('str')

# Dropping all transactions which were done on
credit
data = data[~data['InvoiceNo'].str.contains('C')]
```

5Code:

```
# Transactions done in France
basket_France = (data[data['Country'] =="France"]
                 .groupby(['InvoiceNo',
'Description'])['Quantity']

        .sum().unstack().reset_index().fillna(0)
                 .set_index('InvoiceNo'))

# Transactions done in the United Kingdom
basket_UK = (data[data['Country'] =="United
Kingdom"]
                 .groupby(['InvoiceNo',
'Description'])['Quantity']

        .sum().unstack().reset_index().fillna(0)
                 .set_index('InvoiceNo'))

# Transactions done in Portugal
basket_Por = (data[data['Country'] =="Portugal"]
                 .groupby(['InvoiceNo',
'Description'])['Quantity']

        .sum().unstack().reset_index().fillna(0)
                 .set_index('InvoiceNo'))

basket_Sweden = (data[data['Country']
=="Sweden"]
                 .groupby(['InvoiceNo',
'Description'])['Quantity']

        .sum().unstack().reset_index().fillna(0)
                 .set_index('InvoiceNo'))
```

6 Code:

```
# Defining the hot encoding function to make
the data suitable
# for the concerned libraries
def hot_encode(x):
        if(x<= 0):
                return 0
        if(x>= 1):
                return 1

# Encoding the datasets
basket_encoded =
basket_France.applymap(hot_encode)
basket_France = basket_encoded

basket_encoded =
basket_UK.applymap(hot_encode)
basket_UK = basket_encoded

basket_encoded =
basket_Por.applymap(hot_encode)
basket_Por = basket_encoded
```

7

```
basket_encoded =
basket_Sweden.applymap(hot_encode)
basket_Sweden = basket_encoded
```

7 Code:

```
# Building the model
frq_items = apriori(basket_France, min_support =
0.05, use_colnames = True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric ="lift",
min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'],
ascending =[False, False])
print(rules.head())
```

Output:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
44	(JUMBO BAG WOODLAND ANIMALS)	(POSTAGE)	0.076531	0.765306	0.076531	1.000	1.306667	0.017661	inf
268	(PLASTERS IN TIN CIRCUS PARADE, RED TOADSTOOL...	(POSTAGE)	0.051020	0.765306	0.051020	1.000	1.306667	0.011974	inf
270	(PLASTERS IN TIN WOODLAND ANIMALS, RED TOADSTO...	(POSTAGE)	0.053571	0.765306	0.053571	1.000	1.306667	0.012573	inf
301	(SET 6 RED SPOTTY PAPER CUPS, SET 20 RED RETRO...	(SET 6 RED SPOTTY PAPER PLATES)	0.102041	0.127551	0.099490	0.975	7.644000	0.086474	34.897959
302	(SET 6 RED SPOTTY PAPER PLATES, SET 20 RED RET...	(SET 6 RED SPOTTY PAPER CUPS)	0.102041	0.137755	0.099490	0.975	7.077778	0.085433	34.489796

8 Code:

```
frq_items = apriori(basket_UK, min_support =
0.01, use_colnames = True)
rules = association_rules(frq_items, metric ="lift",
min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'],
ascending =[False, False])
print(rules.head())
```

Output:

	antecedents	consequents	antecedent support
116	(BEADED CRYSTAL HEART PINK ON STICK)	(DOTCOM POSTAGE)	0.011036
2019	(SUKI SHOULDER BAG, JAM MAKING SET PRINTED)	(DOTCOM POSTAGE)	0.011625
2296	(HERB MARKER THYME, HERB MARKER MINT)	(HERB MARKER ROSEMARY)	0.010714
2302	(HERB MARKER PARSLEY, HERB MARKER ROSEMARY)	(HERB MARKER THYME)	0.011089
2300	(HERB MARKER THYME, HERB MARKER PARSLEY)	(HERB MARKER ROSEMARY)	0.011089

9 Code:

```
frq_items = apriori(basket_Por, min_support =
0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift",
min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'],
ascending =[False, False])
print(rules.head())
```

Output:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1170	(SET 12 COLOUR PENCILS DOLLY GIRL)	(SET 12 COLOUR PENCILS SPACEBOY)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1171	(SET 12 COLOUR PENCILS SPACEBOY)	(SET 12 COLOUR PENCILS DOLLY GIRL)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1172	(SET 12 COLOUR PENCILS DOLLY GIRL)	(SET OF 4 KNICK KNACK TINS LONDON)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1173	(SET OF 4 KNICK KNACK TINS LONDON)	(SET 12 COLOUR PENCILS DOLLY GIRL)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf
1174	(SET 12 COLOUR PENCILS DOLLY GIRL)	(SET OF 4 KNICK KNACK TINS POPPIES)	0.051724	0.051724	0.051724	1.0	19.333333	0.049049	inf

9 Code:

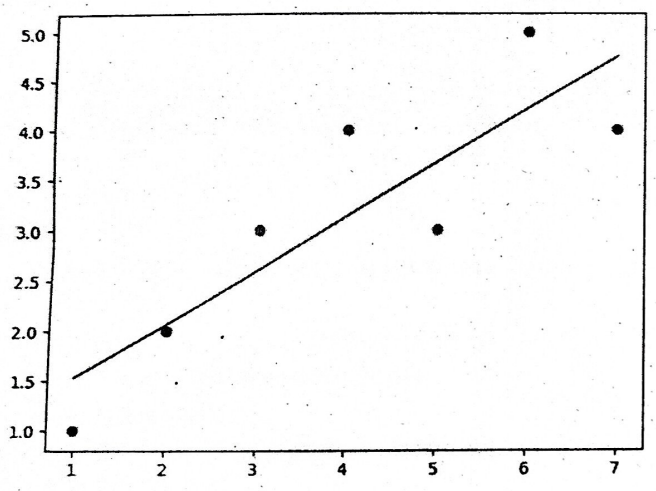
```
frq_items = apriori(basket_Sweden, min_support =
0.05, use_colnames = True)
rules = association_rules(frq_items, metric ="lift",
min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'],
ascending =[False, False])
print(rules.head())
```

Output:

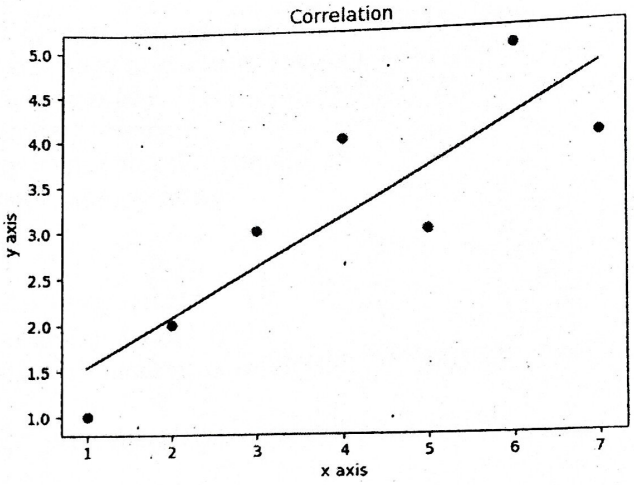
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(12 PENCILS SMALL TUBE SKULL)	(PACK OF 72 SKULL CAKE CASES)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
1	(PACK OF 72 SKULL CAKE CASES)	(12 PENCILS SMALL TUBE SKULL)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
4	(36 DOILIES DOLLY GIRL)	(ASSORTED BOTTLE TOP MAGNETS)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
5	(ASSORTED BOTTLE TOP MAGNETS)	(36 DOILIES DOLLY GIRL)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf
180	(CHILDRENS CUTLERY DOLLY GIRL)	(CHILDRENS CUTLERY CIRCUS PARADE)	0.055556	0.055556	0.055556	1.0	18.0	0.052469	inf

10. Visualize the correlation matrix using a pseudocolor plot.

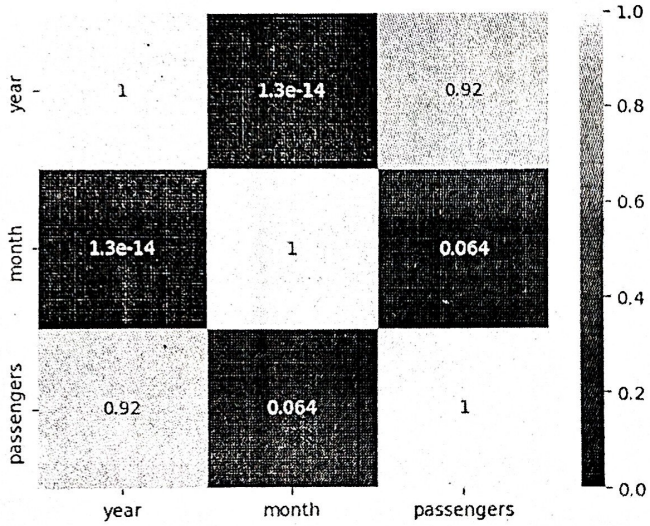
```
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
y = pd.Series([1, 2, 3, 4, 3, 5, 4])
x = pd.Series([1, 2, 3, 4, 5, 6, 7])
correlation = y.corr(x)
correlation
o/p= np.float64(0.8603090020146067)

# plotting the data
plt.scatter(x, y)

# This will fit the best line into the graph
plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))
        (np.unique(x)), color='red')
# adds the title
plt.title('Correlation')
```




```
# plot the data
plt.scatter(x, y)

# fits the best fitting line to the data
plt.plot(np.unique(x),
        np.poly1d(np.polyfit(x, y, 1))
        (np.unique(x)), color='red')
```

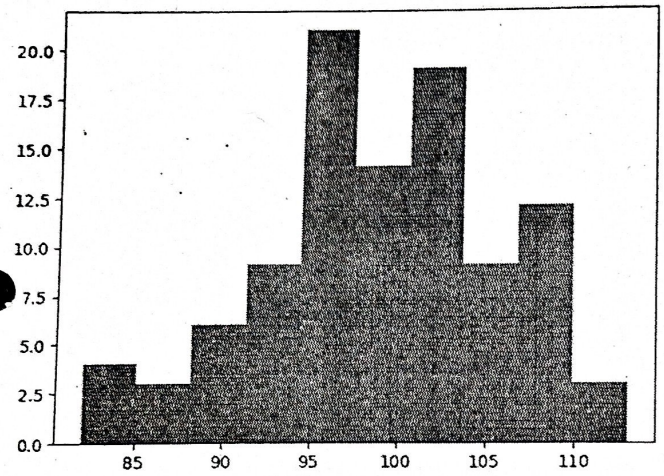



```
# Labelling axes
plt.xlabel('x axis')
plt.ylabel('y axis')
import seaborn as sns

# checking correlation using heatmap
#Loading dataset
flights = sns.load_dataset("flights")

# Convert 'month' column to numerical
representation
flights['month'] = flights['month'].cat.codes  #
Assigns numerical codes to months

#plotting the heatmap for correlation
ax = sns.heatmap(flights.corr(), annot=True0029
```

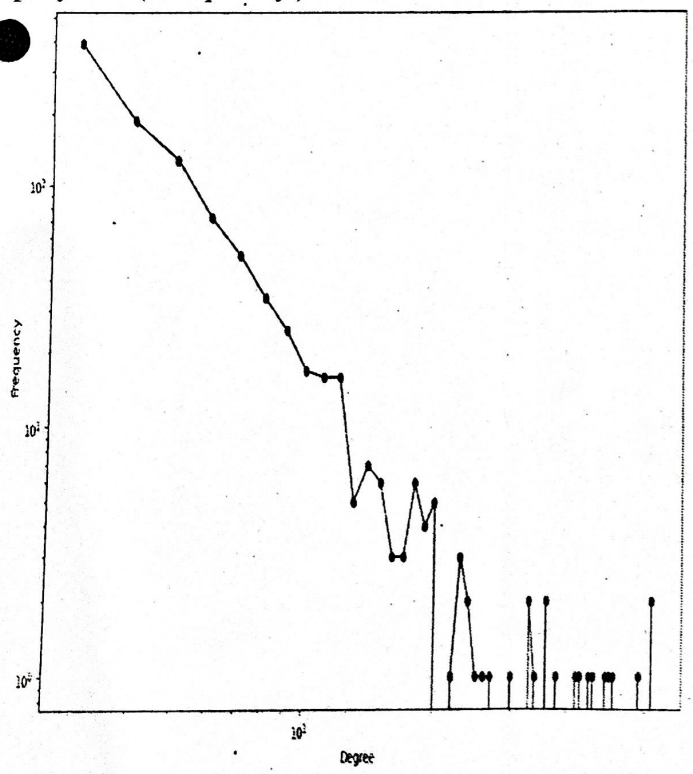


11. Use of degrees distribution of a network:

```
import matplotlib.pyplot as plt
import networkx as nx

def plot_degree_dist(G):
    degrees = [G.degree(n) for n in G.nodes()]
    plt.hist(degrees)
    plt.show()

plot_degree_dist(nx.gnp_random_graph(100, 0.5,
directed=True))
```

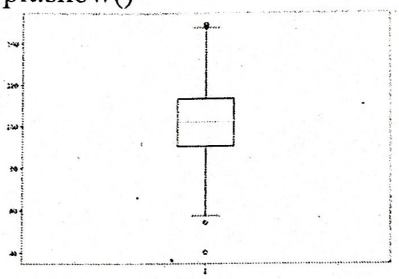



```
m=3
G = nx.barabasi_albert_graph(1000, m)

degree_freq = nx.degree_histogram(G)
degrees = range(len(degree_freq))
plt.figure(figsize=(12, 8))
plt.loglog(degrees[m:], degree_freq[m:],'go-')
plt.xlabel('Degree')
plt.ylabel('Frequency')
```



12. Graph visualization of a network using maximum, minimum, median, first quartile and third quartile.

```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np


# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)

fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(data)

# show plot
plt.show()
```



```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np

# Creating dataset
np.random.seed(10)
data_1 = np.random.normal(100, 10, 200)
data_2 = np.random.normal(90, 20, 200)
data_3 = np.random.normal(80, 30, 200)
data_4 = np.random.normal(70, 40, 200)
data = [data_1, data_2, data_3, data_4]
fig = plt.figure(figsize =(10, 7))
# Creating axes instance
ax = fig.add_axes([0, 0, 1, 1])
# Creating plot
bp = ax.boxplot(data)
# show plot
plt.show()
```