**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

August 5711193

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

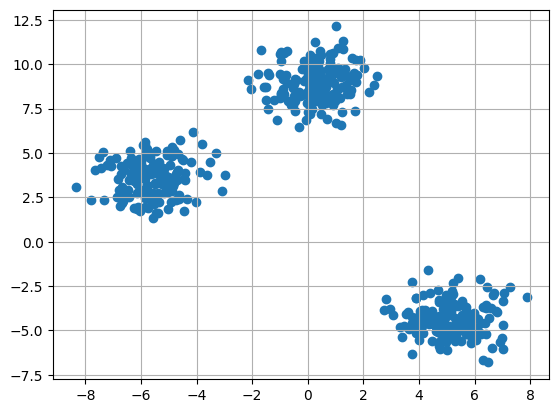
X,y = make\_blobs(n\_samples = 500,n\_features = 2,centers = 3,random\_state = 23)

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])

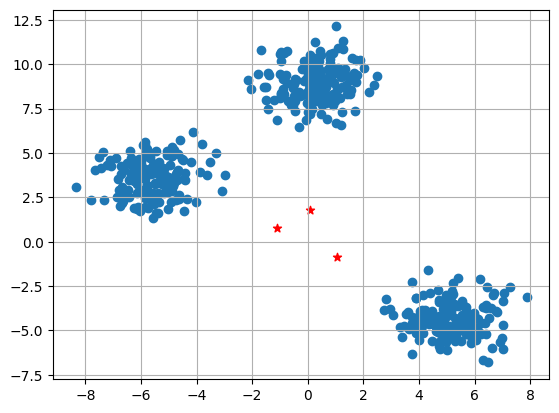
plt.grid(True)

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

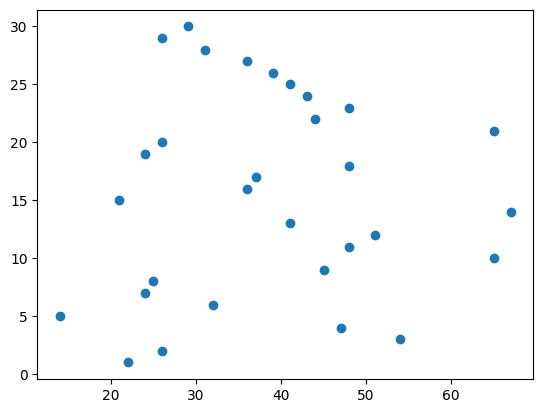
4 5 BBB 15232 14

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)

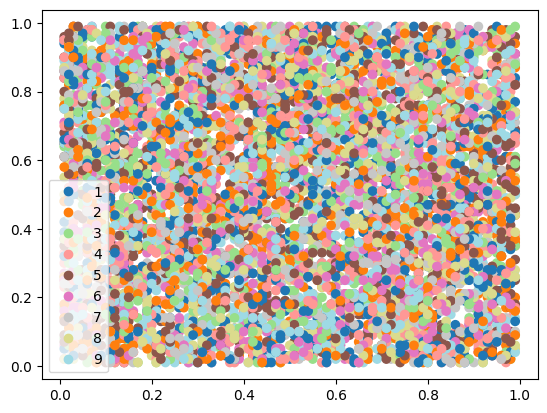
u\_labels = np.unique(data\_normalized['y\_kmeans']).tolist()

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/datasciencedojo/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

**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

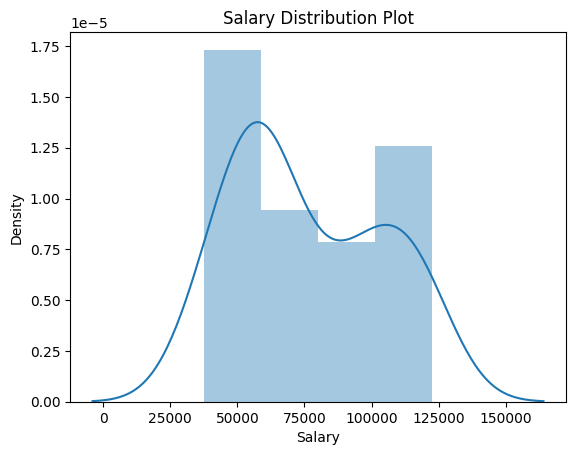
df\_sal.describe()

# 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

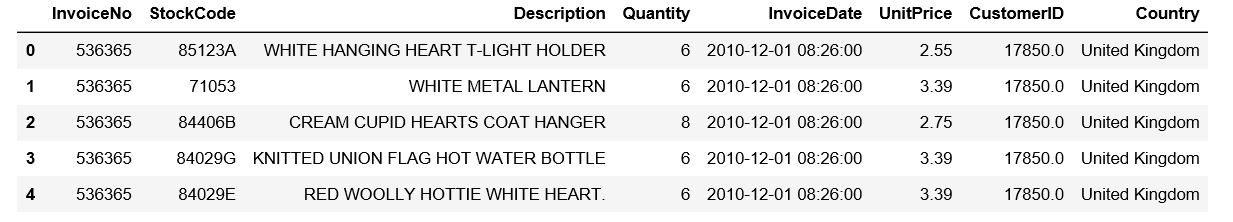
cd C:\Users\Dev\Desktop\Kaggle\Apriori Algorithm

# Loading the Data

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

data.head()

**Output:**

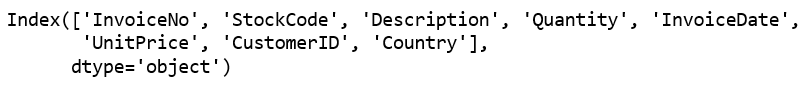


**2 Code:**

**# Exploring the columns of the data**

**data.columns**

**Output:**



**3 Code:**

**# Exploring the different regions of transactions**

**data.Country.unique()**

**Output:**



**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

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)

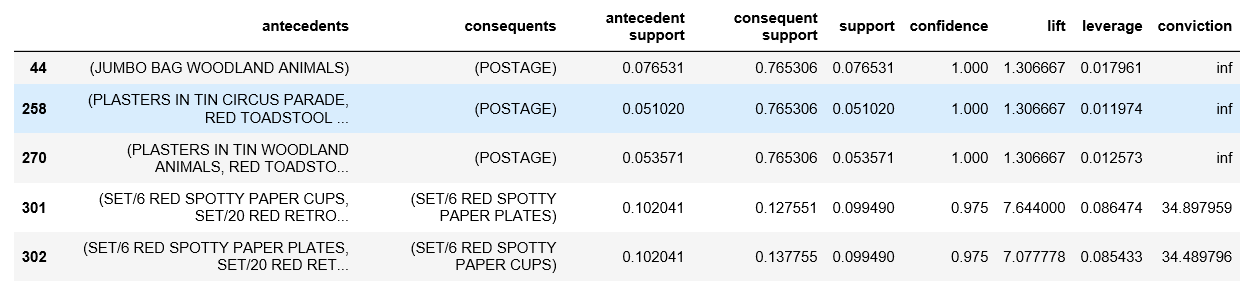
# Collecting the inferred rules in a dataframe

rules = association\_rules(frq\_items, metric ="lift", min\_threshold = 1)

rules = rules.sort\_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

**Output:**



**8 Code:**

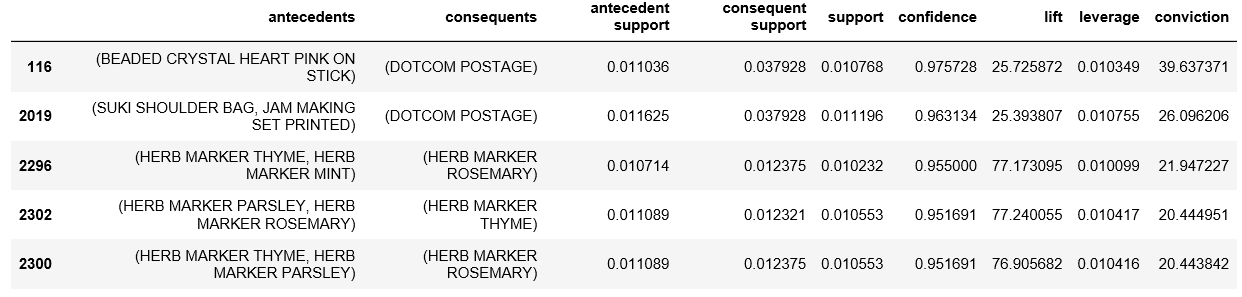
frq\_items = apriori(basket\_UK, min\_support = 0.01, use\_colnames = True)

rules = association\_rules(frq\_items, metric ="lift", min\_threshold = 1)

rules = rules.sort\_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

**Output:**



**9 Code:**

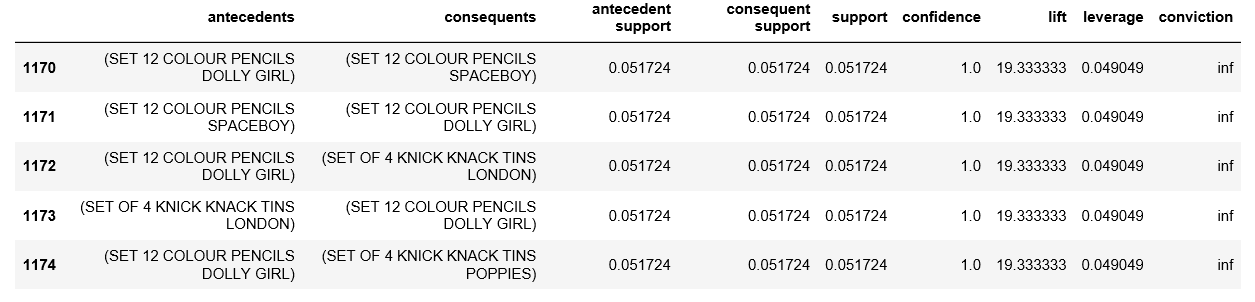
frq\_items = apriori(basket\_Por, min\_support = 0.05, use\_colnames = True)

rules = association\_rules(frq\_items, metric ="lift", min\_threshold = 1)

rules = rules.sort\_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

**Output:**



**9 Code:**

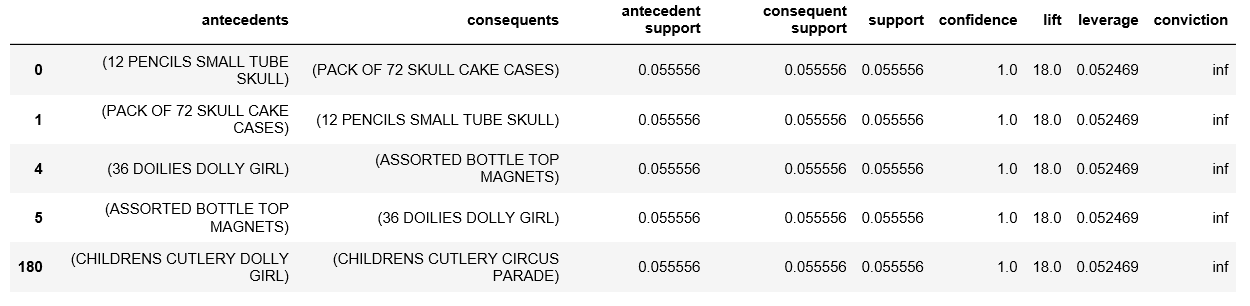
frq\_items = apriori(basket\_Sweden, min\_support = 0.05, use\_colnames = True)

rules = association\_rules(frq\_items, metric ="lift", min\_threshold = 1)

rules = rules.sort\_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

**Output:**



**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)

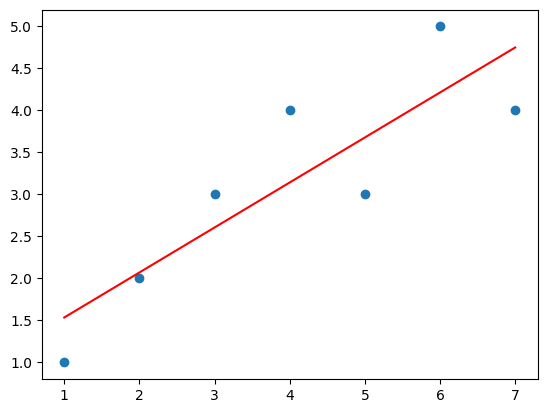
# This will fit the best line into the graph

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

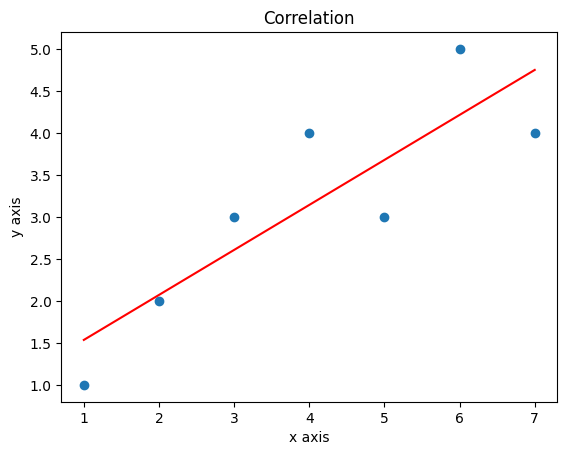
plt.scatter(x, y)

# 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

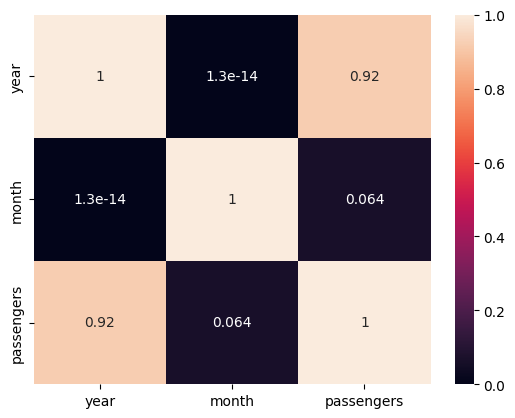
flights = sns.load\_dataset("flights")

# 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

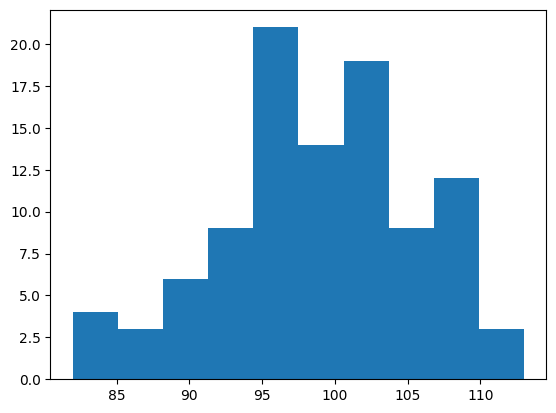
def plot\_degree\_dist(G):

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)

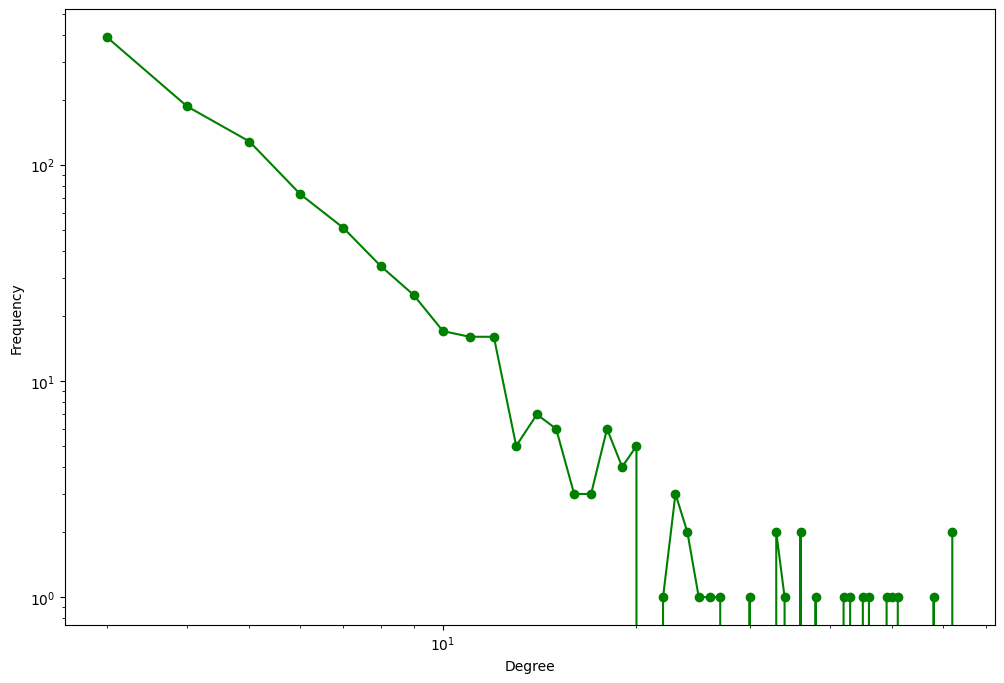
degrees = range(len(degree\_freq))

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)

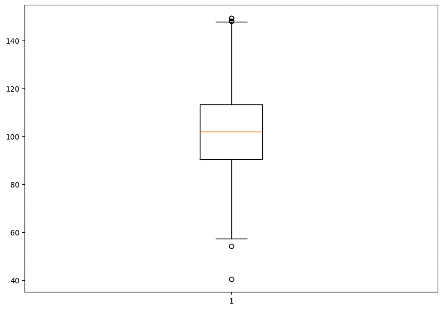
fig = plt.figure(figsize =(10, 7))

# 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()

