Name: Pavithra B

Register Number. 240701380

Department: CSE

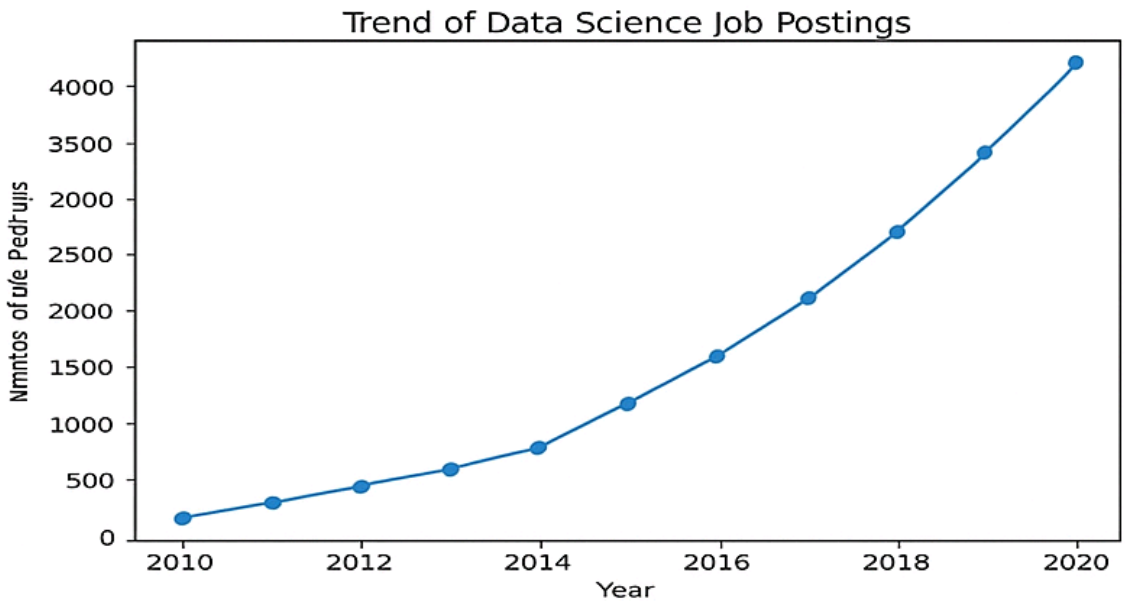
Exp No: 1.a Analyse the trend of data science job postings over the last decade

Description: Use web scraping (e.g.. BeautifulSoup) ot APIs (e.g.. LinkedIn API) to gather data on the humber of data science job postings each year. Use pandas for data manipulation and matplotlib/seaborn for visualization.

Program:

```
import pand»> as pd
import matpiotila,pyplot.as.plɨ
dala = ['ve¤-  ' [ictlrange|zato, 2021)],
        ¦-ıUb Postings [ [150, 300, 460, 600, 800, 1200, 1600, 2100, 2700, 2400, 4200])
df = pd.Dataframe(dato)
plt plot(df['vew-'], dft'Job Postings'[, marker '»')
pit:ɪ:tls['frend of Data Science Job Postings')
plt:vlobe]('vear')
plt:vlobe]('Number of Job Postings')
plt:shom(]
```

Output:



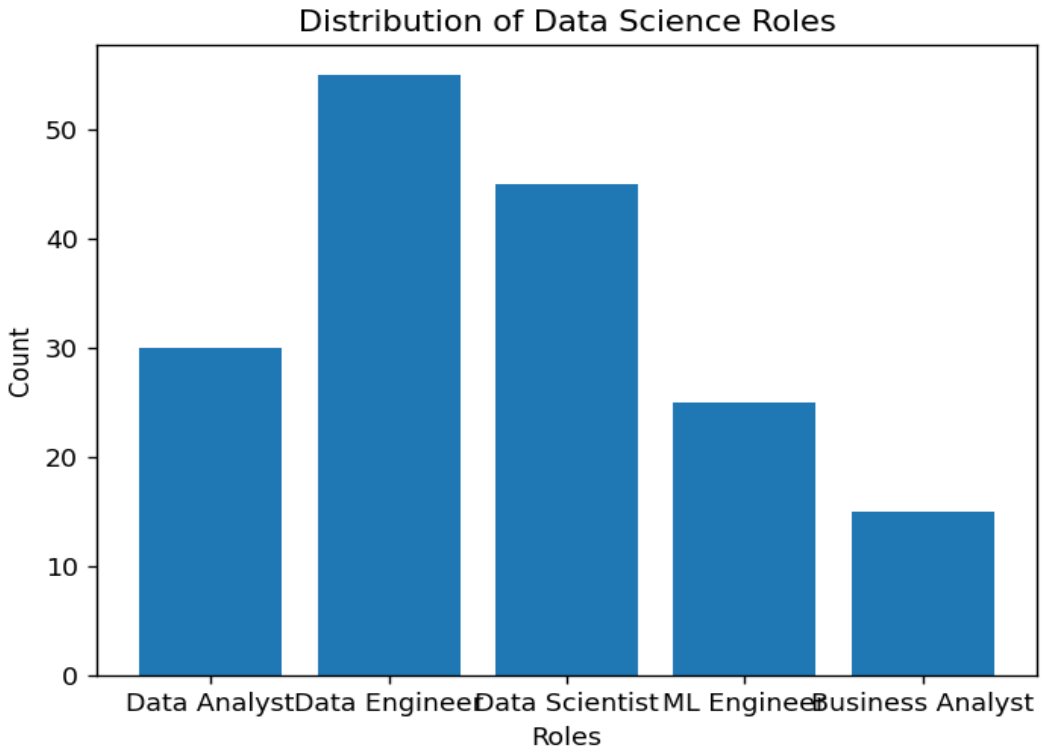
Exp No: 1.b Analyse and visualize the distribution of various data science roles (Data Analyst, Data Engineer, Data Scientist, etc.) from a dataset.

Description: Use a dataset of job postings and categorize them into different roles. Visualize the distribution using pie charts or bar plots.

Program:

```
import matplotlib.pyplot as plt
roles = ['Data Analyst', 'Data Engineer', 'Data Scientist', 'ML Engineer', 'Business Analyst']
counts = [30, 55, 45, 25, 15]
plt.bar(roles, counts)
plt.title('Distribution of Data Science Roles')
plt.xlabel('Roles')
plt.ylabel('Count')
plt.show()
```

Output:



Exp No: 1.c Conduct an experiment to differentiate Structured, Un-structured and Semi-structured data based on data sets given.

Description: Create small datasets for each type and explain their characteristics.

Program:

```
import pandas as pd
structured_data = pd.DataFrame({'ID': [100, 101, 102],
                                'Name': ['Jane', 'Tina', 'Jenny'],'Age': [20, 25, 30]})
print("Structured Data:\n", structured_data)
unstructured_data = "Unstructured data can be a piece of text, an image,or a video file."
print("\nUnstructured Data:\n", unstructured_data)
semi_structured_data = {'ID': 100, 'Name': 'Jane',
                        'Attributes': {'Height': 160, 'Weight': 55}}
print("\nSemi-structured Data:\n", semi_structured_data)
```

Output:

```
Structured Data:
     ID   Name  Age
0  100   Jane   20
1  101   Tina   25
2  102  Jenny   30

Unstructured Data:
 Unstructured data can be a piece of text, an image,or a video file.

Semi-structured Data:
 {'ID': 100, 'Name': 'Jane', 'Attributes': {'Height': 160, 'Weight': 55}}
```

Exp No: 1.d Conduct an experiment to encrypt and decrypt given sensitive data

Description: Use the cryptography library to encrypt and decrypt a piece of data.

Program:

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
cipher_suite = Fernet(key)
user_data = input("Enter sensitive data to encrypt: ")
plain_text = user_data.encode()
cipher_text = cipher_suite.encrypt(plain_text)
decrypted_text = cipher_suite.decrypt(cipher_text)
print("\nOriginal Data:", plain_text)
print("Encrypted Data:", cipher_text)
print("Decrypted Data:", decrypted_text)
```

Output:

```
Enter sensitive data to encrypt: Rajalakshmi Engineering College

Original Data: b'Rajalakshmi Engineering College'
Encrypted Data: b'gAAAAABo3oiy2w9f2omCE3RbQgR1Yx07VT8hTqFJcOdIW59DuGit-_9YDfP9Dy81TAWXiQS4dO7H8j
X4OzfEnGLa_axQmQdsCK1_bBUckd--rDkkqrRk7Ts='
Decrypted Data: b'Rajalakshmi Engineering College'
```

Exp No: 2 Upload and Analyse the data set given in csv format and perform data preprocessing and visualization.

Program:

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

data = {
    'Date': pd.date_range(start='2023-01-01', periods=16, freq='D'),
    'Product': ['Product A', 'Product B', 'Product C', 'Product A', 'Product B', 'Product C',
                'Product A', 'Product B', 'Product C', 'Product A', 'Product B', 'Product C',
                'Product A', 'Product B', 'Product C', 'Product A'],
    'Sales': [200, 150, 300, 220, 180, 340, 210, 170, 310, 230, 160, 320, 240, 190, 330, 250],
    'Quantity': [4, 3, 6, 5, 4, 8, 4, 3, 7, 5, 3, 6, 5, 4, 8, 6],
    'Region': ['North', 'South', 'East', 'North', 'West', 'East', 'North', 'South',
               'East', 'West', 'South', 'East', 'North', 'West', 'East', 'North']
}

df = pd.DataFrame(data)

print(df.head())
print(df.isnull().sum())
df['Sales'].fillna(df['Sales'].mean(), inplace=True)
df.dropna(subset=['Product', 'Quantity', 'Region'], inplace=True)
print(df.describe())

product_summary = df.groupby('Product').agg({
    'Sales': 'sum',
    'Quantity': 'sum'
}).reset_index()
print(product_summary)

plt.figure(figsize=(10, 6))
plt.bar(product_summary['Product'], product_summary['Sales'])
plt.xlabel('Product')
plt.ylabel('Total Sales')
plt.title('Total Sales by Product')
plt.show()

sales_over_time = df.groupby('Date').agg({'Sales': 'sum'}).reset_index()
plt.figure(figsize=(10, 6))
plt.plot(sales_over_time['Date'], sales_over_time['Sales'])
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.title('Sales Over Time')
plt.show()

```

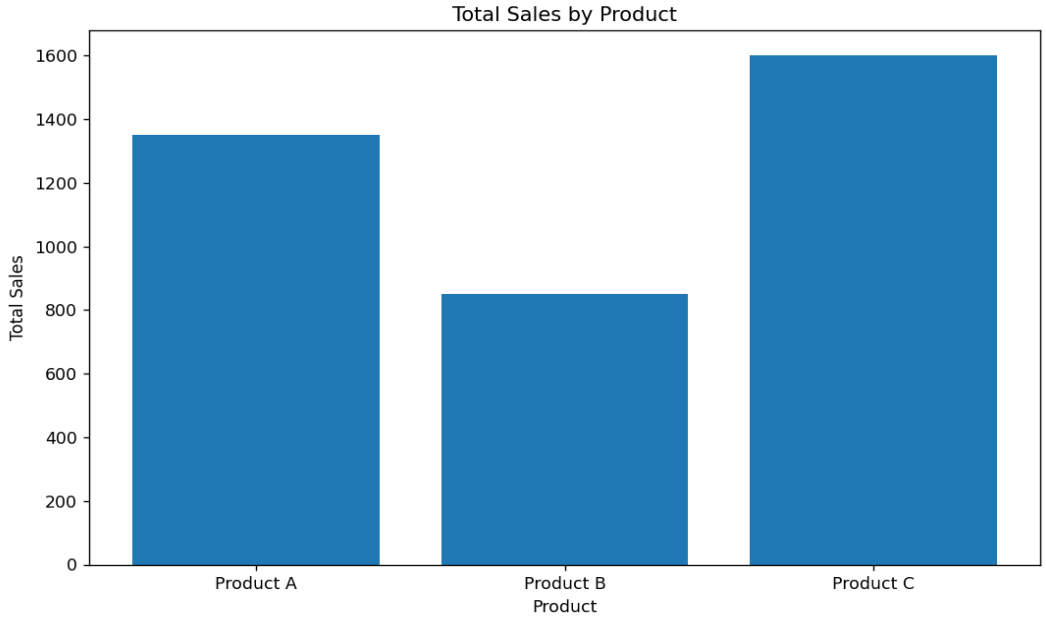
```python
pivot_table = df.pivot_table(values='Sales', index='Region', columns='Product', aggfunc=np.sum, fill_value=0)
print(pivot_table)

correlation_matrix = df[['Sales', 'Quantity']].corr()
print(correlation_matrix)

plt.figure(figsize=(8, 6))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
for i in range(len(correlation_matrix)):
    for j in range(len(correlation_matrix)):
        plt.text(j, i, round(correlation_matrix.iloc[i, j], 2),
                 ha='center', va='center', color='black')
plt.title('Correlation Matrix')
plt.show()
```

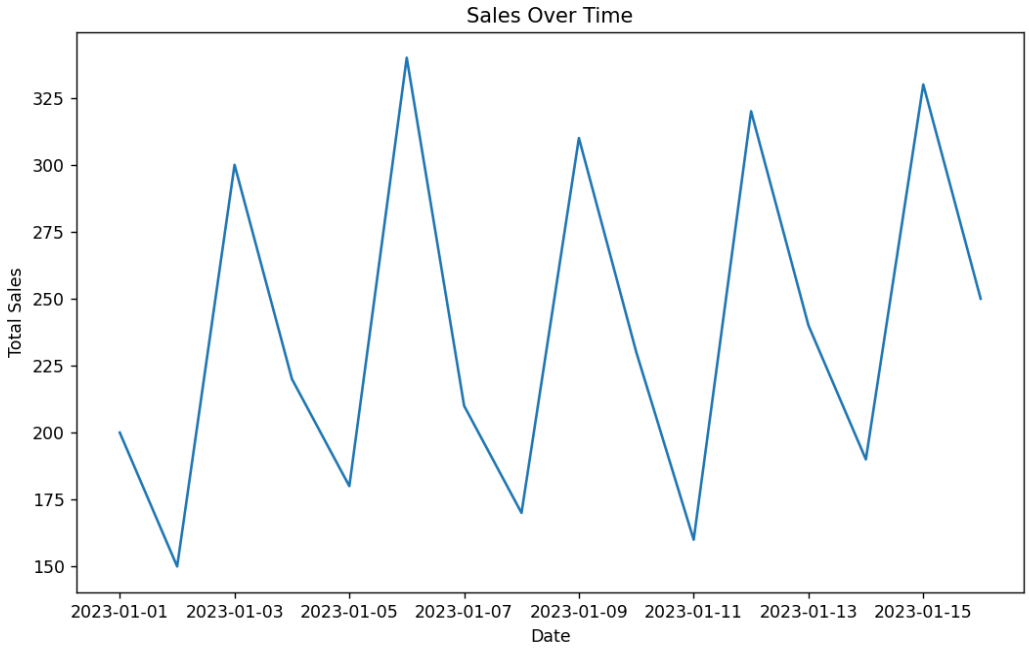
Output:

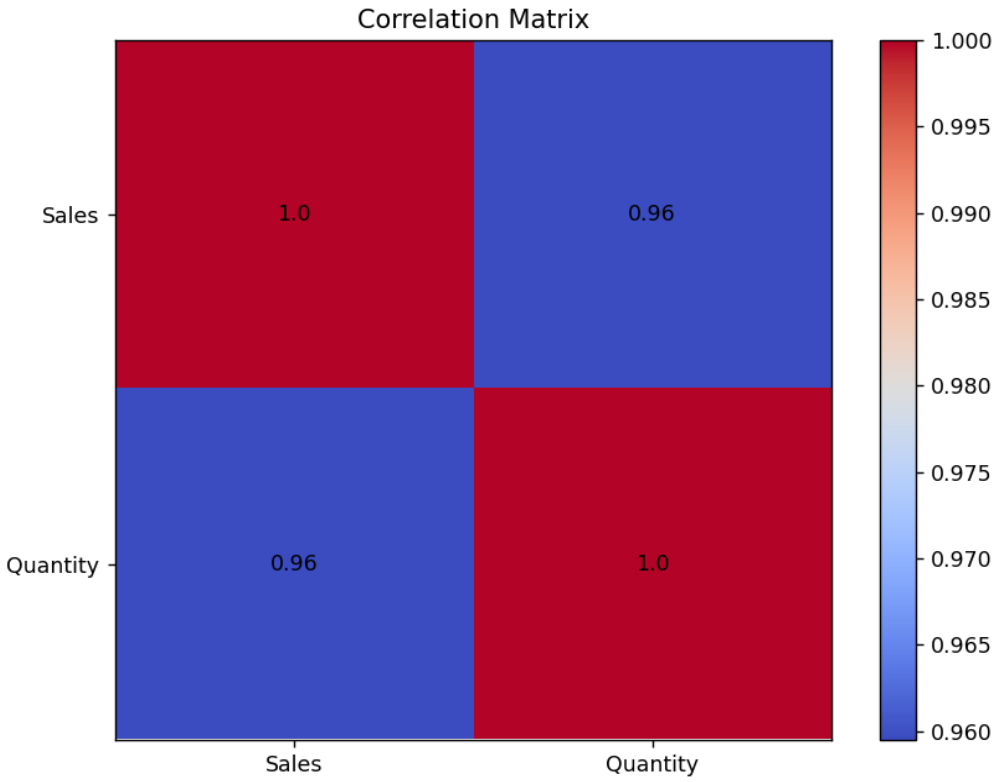
```
         Date    Product  Sales  Quantity Region
0 2023-01-01  Product A    200         4  North
1 2023-01-02  Product B    150         3  South
2 2023-01-03  Product C    300         6   East
3 2023-01-04  Product A    220         5  North
4 2023-01-05  Product B    180         4   West
Date        0
Product     0
Sales       0
Quantity    0
Region      0
dtype: int64
```

```
  df['Sales'].fillna(df['Sales'].mean(), inplace=True)
                      Date       Sales   Quantity
count                   16   16.000000  16.000000
mean   2023-01-08 12:00:00  237.500000   5.062500
min    2023-01-01 00:00:00  150.000000   3.000000
25%    2023-01-04 18:00:00  187.500000   4.000000
50%    2023-01-08 12:00:00  225.000000   5.000000
75%    2023-01-12 06:00:00  302.500000   6.000000
max    2023-01-16 00:00:00  340.000000   8.000000
std                    NaN   64.031242   1.652019
     Product  Sales  Quantity
0  Product A   1350        29
1  Product B    850        17
2  Product C   1600        35
```

```
  pivot_table = df.pivot_table(values='Sales', index='Region', columns='Product', aggfunc=np.sum, fill_value=0)
Product  Product A  Product B  Product C
Region
East             0          0       1600
North         1120          0          0
South            0        480          0
West           230        370          0
             Sales  Quantity
Sales     1.000000  0.959531
Quantity  0.959531  1.000000
```






Correlation Matrix
Sales
Quantity
1.0
0.96
0.96
1.0
Sales
Quantity
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.965
0.960

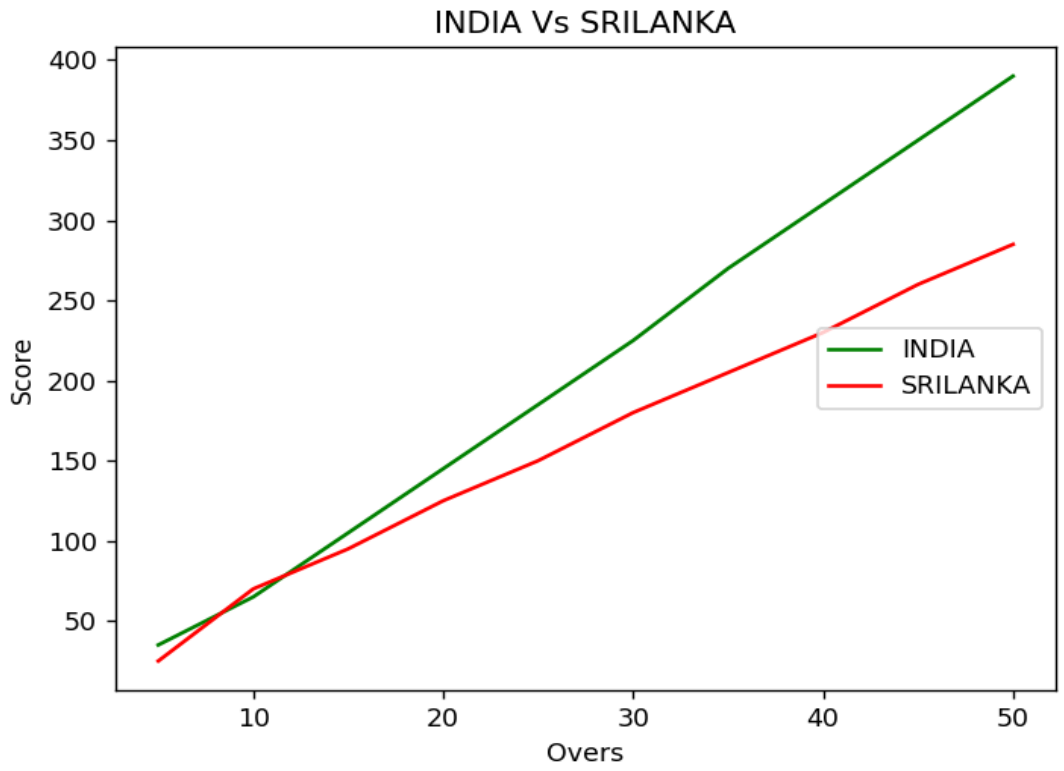
Exp No: 3.a Conduct an experiment to show data visualization using line plot

Description: Take any sample data either through csv file or data fetched directly through code.

Program:

```
import matplotlib.pyplot as cricket
Overs = list(range(5, 51, 5))
Indian_Score = [35, 65, 105, 145, 185, 225, 270, 310, 350, 390]
Srilankan_Score = [25, 70, 95, 125, 150, 180, 205, 230, 260, 285]
cricket.plot(Overs, Indian_Score, color="green", label="INDIA")
cricket.plot(Overs, Srilankan_Score, color="red", label="SRILANKA")
cricket.title("INDIA Vs SRILANKA")
cricket.xlabel("Overs")
cricket.ylabel("Score")
cricket.legend(loc="center right")
cricket.show()
```

Output:



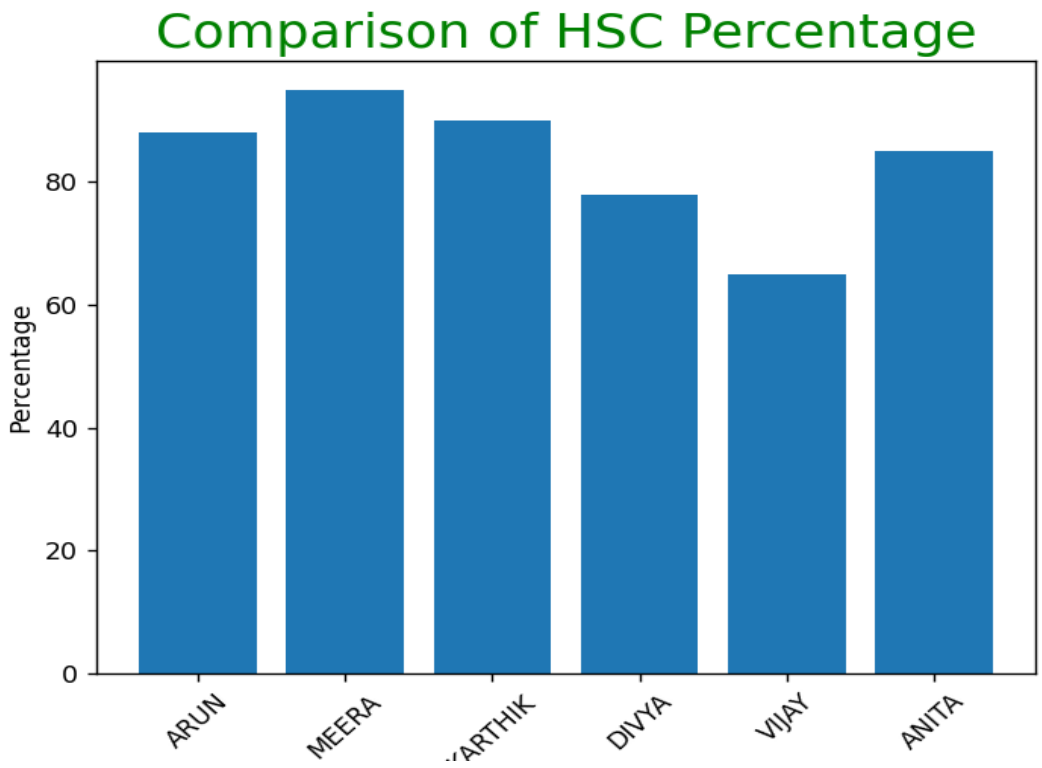
Exp No: 3.b Conduct an experiment to show data visualization using bar chart

Description: Take any sample data either through csv file or data fetched directly through code.

Program:

```
import matplotlib.pyplot as hscmark
import numpy as np
Names = ['ARUN', 'MEERA', 'KARTHIK', 'DIVYA', 'VIJAY', 'ANITA']
xaxis = np.arange(len(Names))
Percentage_hsc = [88, 95, 90, 78, 65, 85]
hscmark.bar(Names, Percentage_hsc)
hscmark.xticks(xaxis, Names, rotation=45)
hscmark.xlabel("Names of Pupil")
hscmark.ylabel("Percentage")
hscmark.title("Comparison of HSC Percentage", fontsize=20, color="green")
hscmark.show()
```

Output:



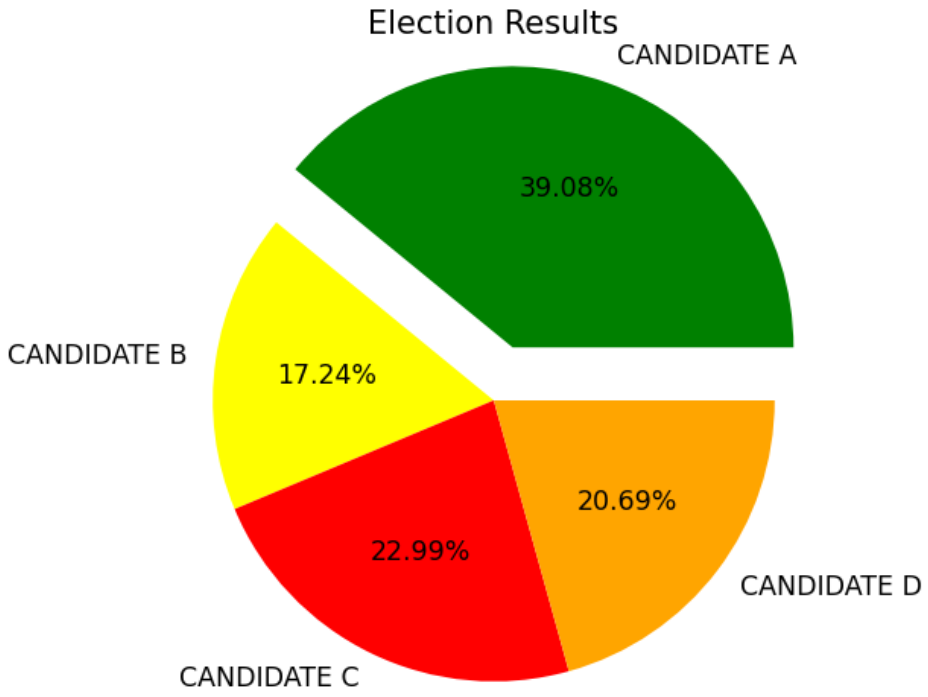
Exp No: 3.c Conduct an experiment to show data visualization using pie chart

Description: Take any sample data either through csv file or data fetched directly through code.

Program:

```
import matplotlib.pyplot as election
labels = ['CANDIDATE A', 'CANDIDATE B', 'CANDIDATE C', 'CANDIDATE D']
Votes = [340, 150, 200, 180]
colors = ['green', 'yellow', 'red', 'orange']
explode = (0.2, 0, 0, 0)
election.pie(Votes, labels=labels, colors=colors, explode=explode, autopct='%0.2f%%')
election.title('Election Results')
election.show()
```

Output:



Exp No: 4 To Count the frequency of occurrence of a word in a body of text is often needed during text processing

Description: Import the word_tokenize function and gutenberg.

Program:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg
from collections import Counter
nltk.download('gutenberg')
nltk.download('punkt')
nltk.download('punkt_tab')
sample = gutenberg.raw("austen-emma.txt")
tokens = word_tokenize(sample)
wlist = tokens[:50]
freq = Counter(wlist)
print("Pairs\n", list(freq.items()))
```

Output:

```
Pairs
 [('[', 1), ('Emma', 2), ('by', 1), ('Jane', 1), ('Austen', 1), ('1816', 1), (']', 1), ('VOLUME', 1), ('I', 2), ('CHAPTER', 1), ('Woodhouse', 1), (',', 5), ('handsome', 1), ('clever', 1), ('and', 3), ('rich', 1), ('with', 2), ('a', 1), ('comfortable', 1), ('home', 1), ('happy', 1), ('disposition', 1), ('seemed', 1), ('to', 1), ('unite', 1), ('some', 1), ('of', 2), ('the', 2), ('best', 1), ('blessings', 1), ('existence', 1), (';', 1), ('had', 1), ('lived', 1), ('nearly', 1), ('twenty-one', 1), ('years', 1), ('in', 1), ('world', 1)]
```

Exp No: 5 Data Collection and Initial Exploration

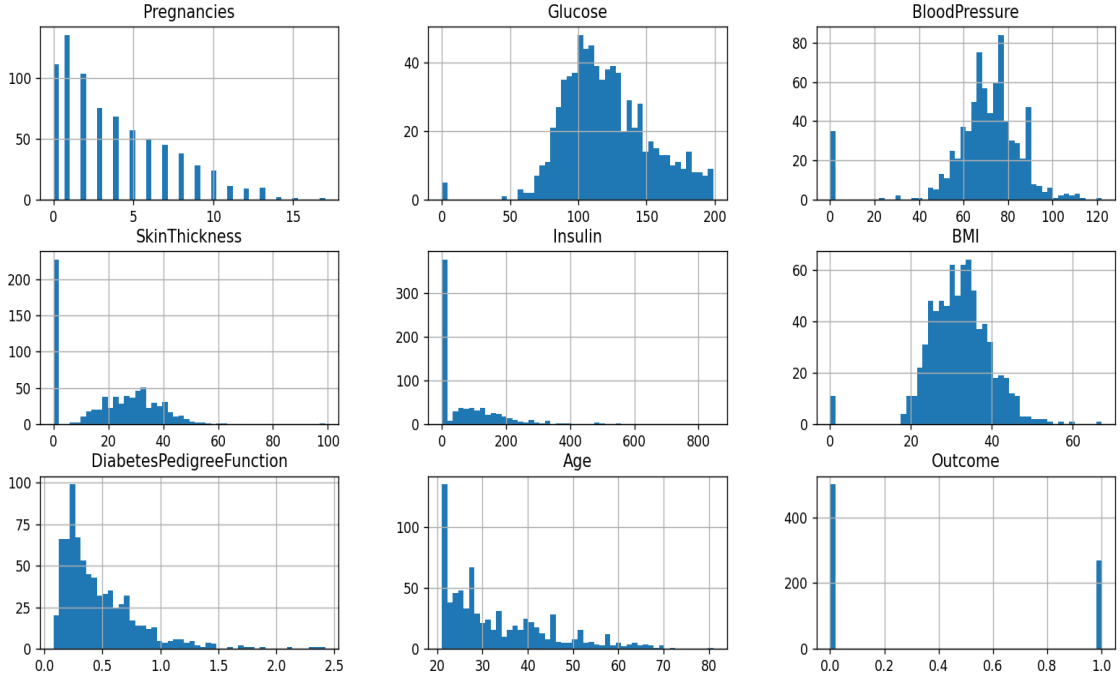
Objective: To collect, load, and perform initial exploration of the diabetes dataset

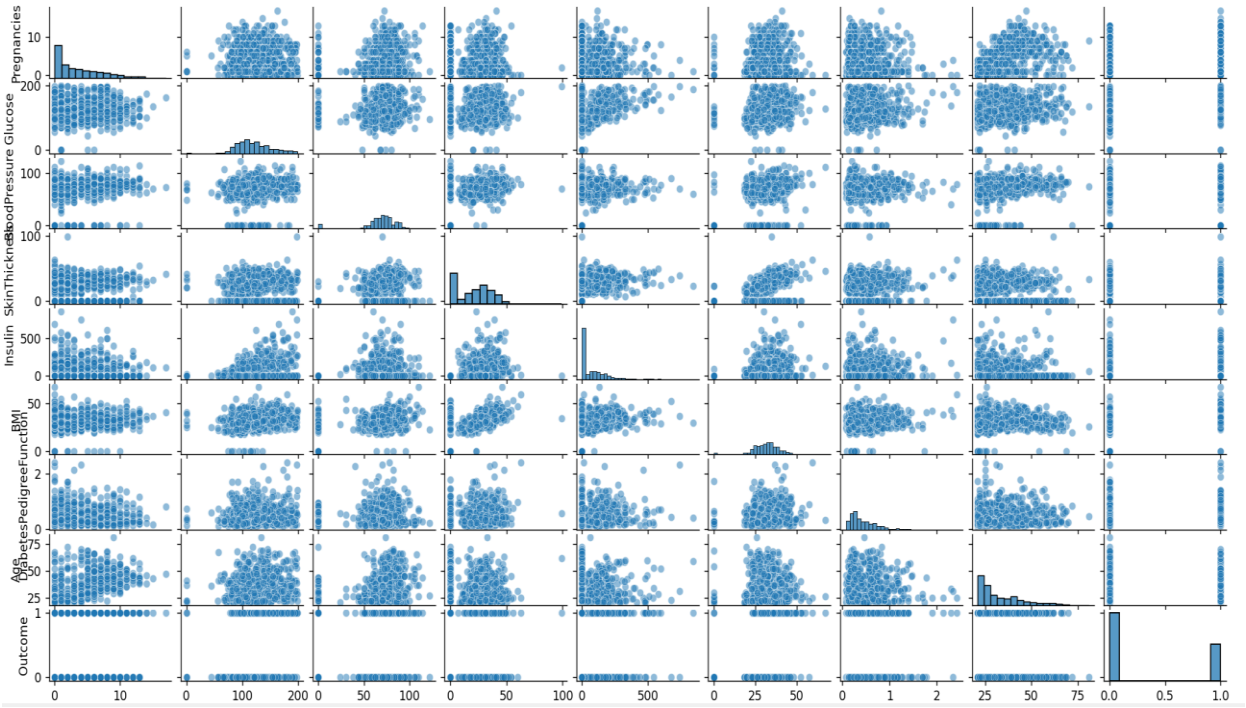
Program:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
cols = ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin",
        "BMI","DiabetesPedigreeFunction","Age","Outcome"]
db = pd.read_csv(url, names=cols)
print(db.head())
print(db.info())
print(db.describe())
db.hist(bins=50, figsize=(20, 15))
plt.show()
sns.pairplot(db, plot_kws={'alpha':0.5}, height=2.5)
plt.show()
```

Output:

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72             35        0  33.6                     0.627   50        1
1            1       85             66             29        0  26.6                     0.351   31        0
2            8      183             64              0        0  23.3                     0.672   32        1
3            1       89             66             23       94  28.1                     0.167   21        0
4            0      137             40             35      168  43.1                     2.288   33        1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin         BMI  DiabetesPedigreeFunction         Age     Outcome
count   768.000000  768.000000     768.000000     768.000000  768.000000  768.000000                768.000000  768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479   31.992578                  0.471876   33.240885    0.348958
std       3.369578   31.972618      19.355807      15.952218  115.244002    7.884160                  0.331329   11.760232    0.476951
min       0.000000    0.000000       0.000000       0.000000    0.000000    0.000000                  0.078000   21.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000   27.300000                  0.243750   24.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000   32.000000                  0.372500   29.000000    0.000000
75%       6.000000  140.250000      80.000000      32.000000  127.250000   36.600000                  0.626250   41.000000    1.000000
max      17.000000  199.000000     122.000000      99.000000  846.000000   67.100000                  2.420000   81.000000    1.000000
```


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

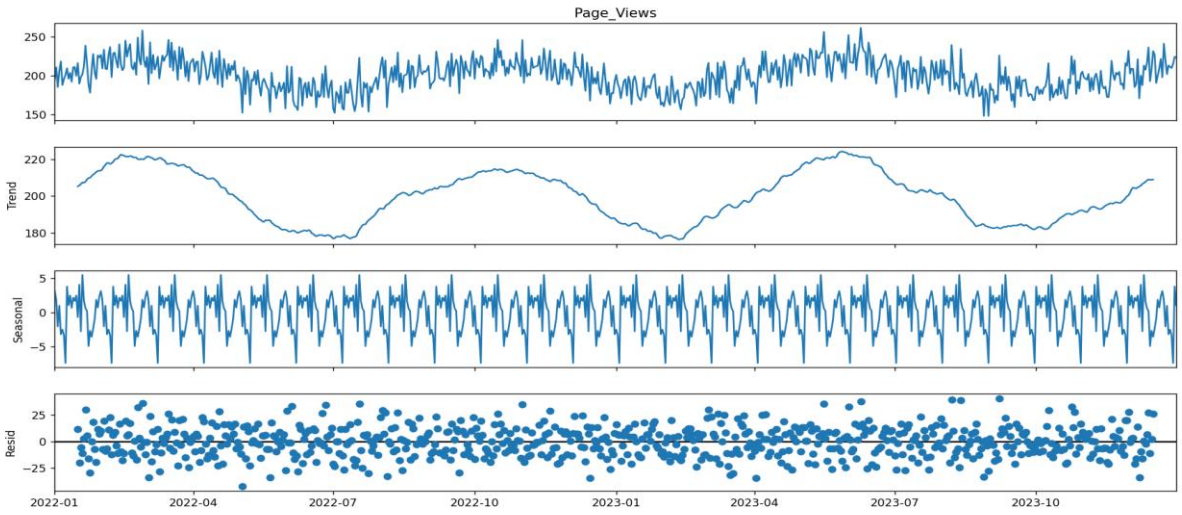
Exp No: 6 Time Series Analysis of Website Traffic

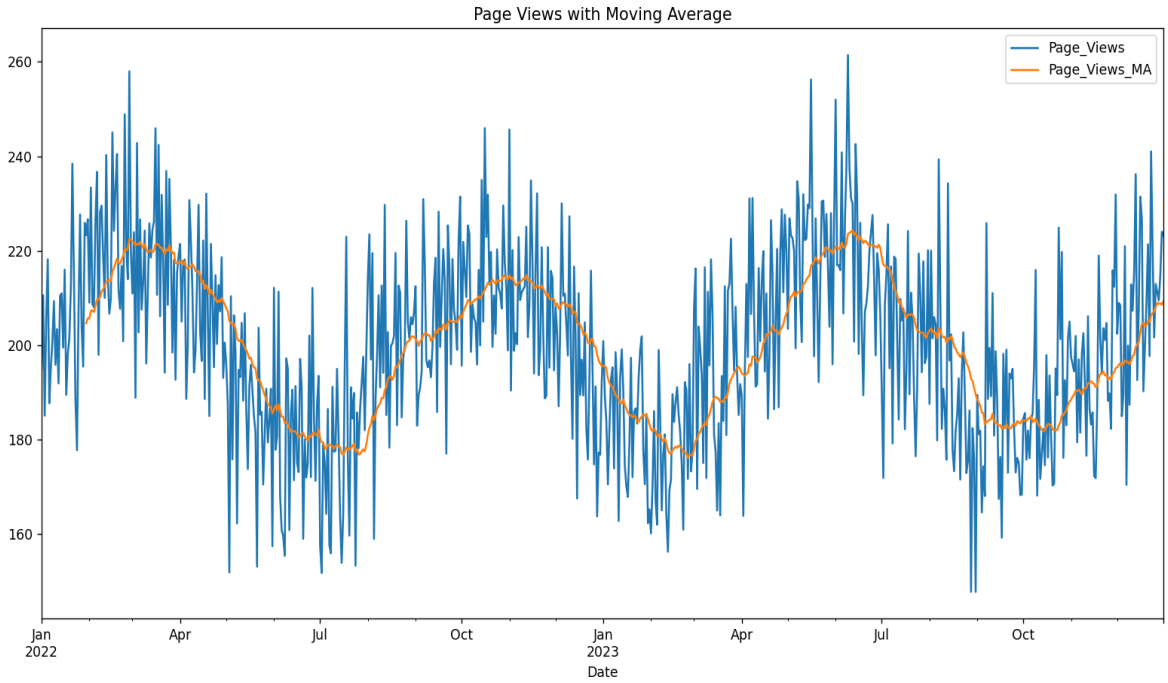
Objective: To analyse the time series data of website traffic to identify trends, seasonality, and anomalies

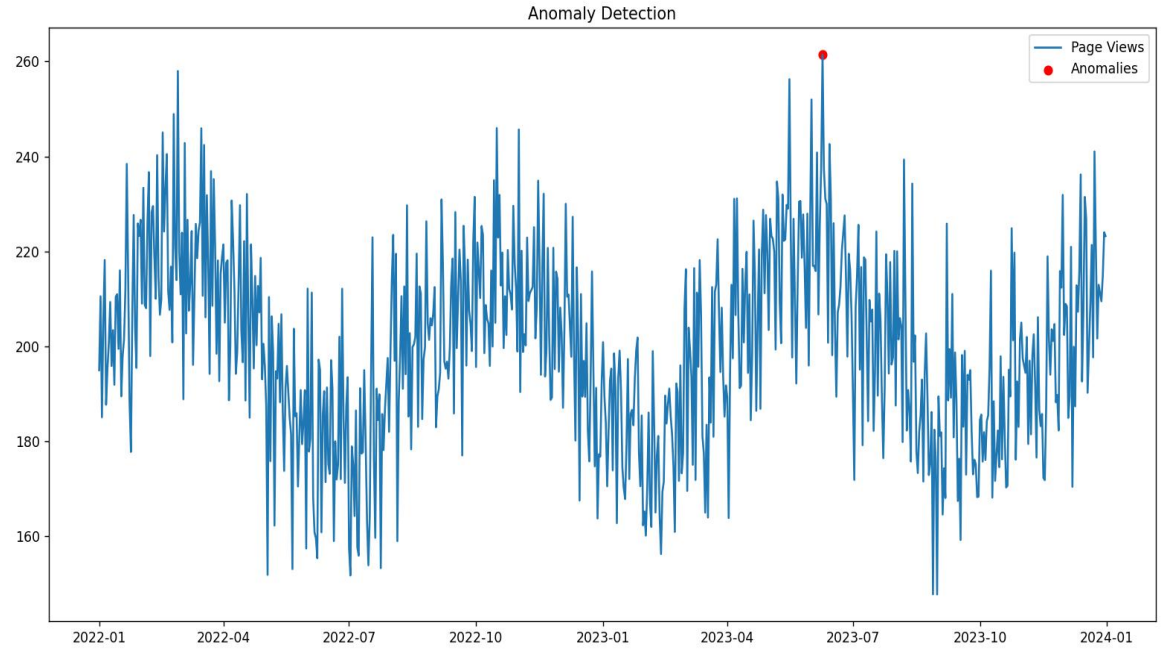
Program:

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
import numpy as np
date_rng = pd.date_range(start="2022-01-01", end="2023-12-31", freq="D")
np.random.seed(42)
page_views = np.random.poisson(lam=200, size=len(date_rng)) + \
             20*np.sin(np.linspace(0, 20, len(date_rng)))
data = pd.DataFrame({'Date': date_rng, 'Page_Views': page_views})
data.set_index('Date', inplace=True)
decomposition = seasonal_decompose(data['Page_Views'], model='additive', period=30)
decomposition.plot()
plt.show()
data['Page_Views_MA'] = data['Page_Views'].rolling(window=30).mean()
data[['Page_Views', 'Page_Views_MA']].plot()
plt.title('Page Views with Moving Average')
plt.show()
data['Page_Views_Z'] = (data['Page_Views'] - data['Page_Views'].mean()) / data['Page_Views'].std()
anomalies = data[data['Page_Views_Z'].abs() > 3]
plt.plot(data.index, data['Page_Views'], label='Page Views')
plt.scatter(anomalies.index, anomalies['Page_Views'], color='red', label='Anomalies')
plt.legend()
plt.title('Anomaly Detection')
plt.show()
```

Output:




Page Views with Moving Average
Page_Views
Page_Views_MA
260
240
220
200
180
160
Jan
2022
Apr
Jul
Oct
Jan
2023
Apr
Jul
Oct
Date


Anomaly Detection
Page Views
Anomalies
260
240
220
200
180
160
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04
2023-07
2023-10
2024-01

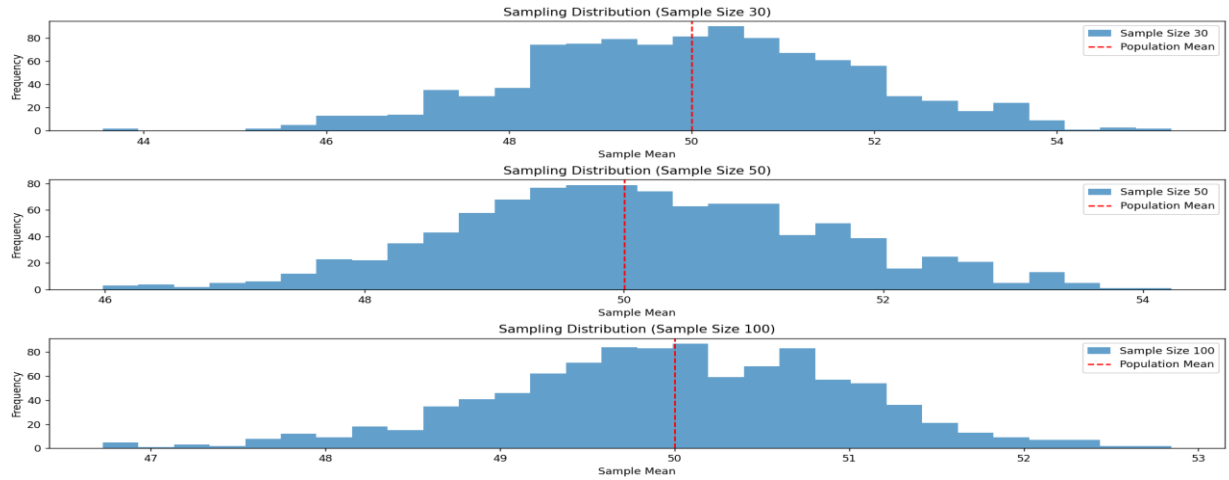
Exp No: 7 Random Sampling and Sampling Distribution

Objective: To explore random sampling from a population and understand the concept of sampling distribution using Python

Program:

```python
import numpy as np
import matplotlib.pyplot as plt
population_mean = 50
population_std = 10
population_size = 100000
population = np.random.normal(population_mean, population_std, population_size)
sample_sizes = [30, 50, 100]
num_samples = 1000
sample_means = {}
for size in sample_sizes:
    sample_means[size] = []
    for _ in range(num_samples):
        sample = np.random.choice(population, size=size, replace=False)
        sample_means[size].append(np.mean(sample))
plt.figure(figsize=(12, 8))
for i, size in enumerate(sample_sizes):
    plt.subplot(len(sample_sizes), 1, i+1)
    plt.hist(sample_means[size], bins=30, alpha=0.7, label=f'Sample Size {size}')
    plt.axvline(np.mean(population), color='red', linestyle='dashed', linewidth=1.5, label='Population Mean')
    plt.title(f'Sampling Distribution (Sample Size {size})')
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.legend()
plt.tight_layout()
plt.show()
```

Output:



Exp No: 8 Conduct Z test for the Data Given using python

Objective: To test whether the average weight of a species of birds differs from 150 grams.

Program:

```python
import numpy as np
from scipy import stats
sample_data = np.array([151, 153, 149, 147, 150, 153, 152, 148, 151, 150, 149, 152,
                        151, 150, 149, 151, 150, 148, 152, 153, 150, 151, 152, 149,
                        150, 153, 152, 148, 150, 151])
population_mean = 150
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
n = len(sample_data)
Z_statistic = (sample_mean - population_mean) / (sample_std / np.sqrt(n))
p_value = 2 * (1 - stats.norm.cdf(abs(Z_statistic)))
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Z-statistic: {Z_statistic:.4f}")
print(f"p-value: {p_value:.4f}")
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average weight is significantly different from 150 grams.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in average weight from 150 grams.")
```

Output:

```
Sample Mean: 150.50
Z-statistic: 1.6540
p-value: 0.0981
Fail to reject the null hypothesis: There is no significant difference in average weight from 150 grams.
```

Exp No: 9 Experiment to understand Matplotlib library use cases in Data Science through visualization

Description: Use Iris data set to understand the Matplotlib library

Program:

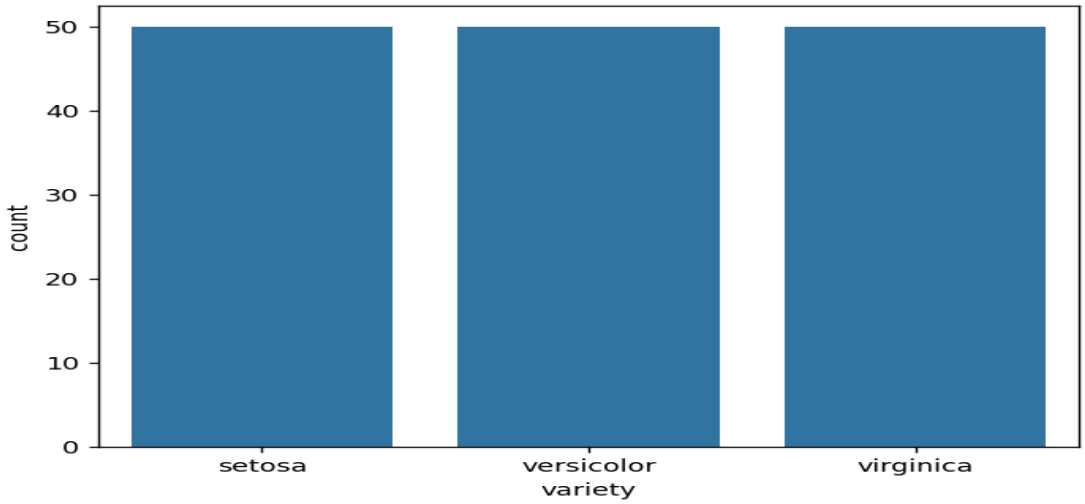
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['variety'] = pd.Categorical.from_codes(iris.target, iris.target_names)
data['Id'] = range(1, len(data) + 1)
cols = ['Id'] + [col for col in data.columns if col != 'Id']
data = data[cols]
print(data.head())
print(data.info())
print(data.describe())
print(data['variety'].value_counts())
sns.countplot(x='variety', data=data)
plt.show()
sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)', hue='variety', data=data)
plt.show()
sns.scatterplot(x='petal length (cm)', y='petal width (cm)', hue='variety', data=data)
plt.show()
sns.pairplot(data, hue='variety', height=3)
plt.show()
sns.FacetGrid(data, hue='variety', height=5).map(sns.histplot, 'petal length (cm)').add_legend()
plt.show()
sns.FacetGrid(data, hue='variety', height=5).map(sns.histplot, 'petal width (cm)').add_legend()
plt.show()
sns.FacetGrid(data, hue='variety', height=5).map(sns.histplot, 'sepal length (cm)').add_legend()
plt.show()
sns.FacetGrid(data, hue='variety', height=5).map(sns.histplot, 'sepal width (cm)').add_legend()
plt.show()
```

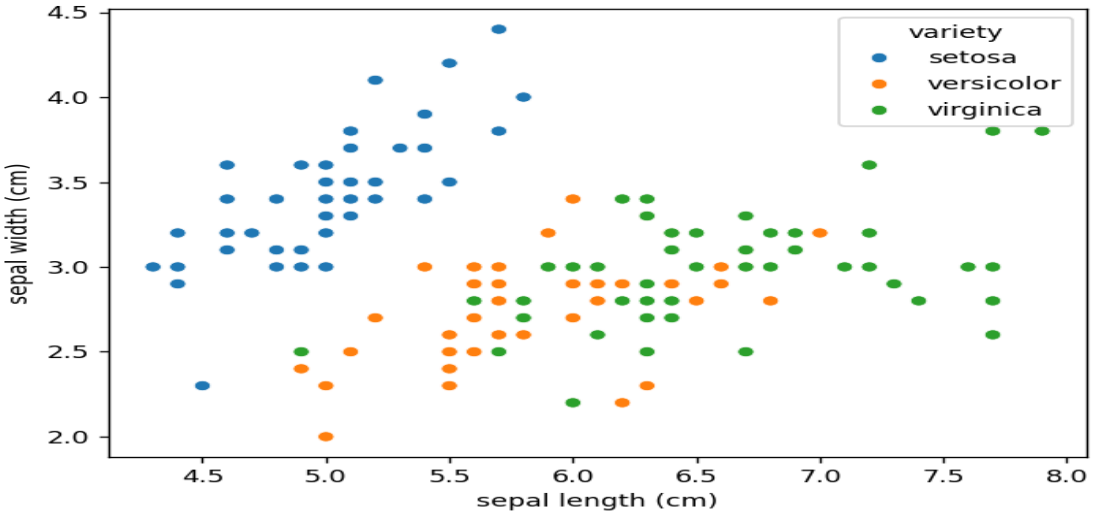
Output:

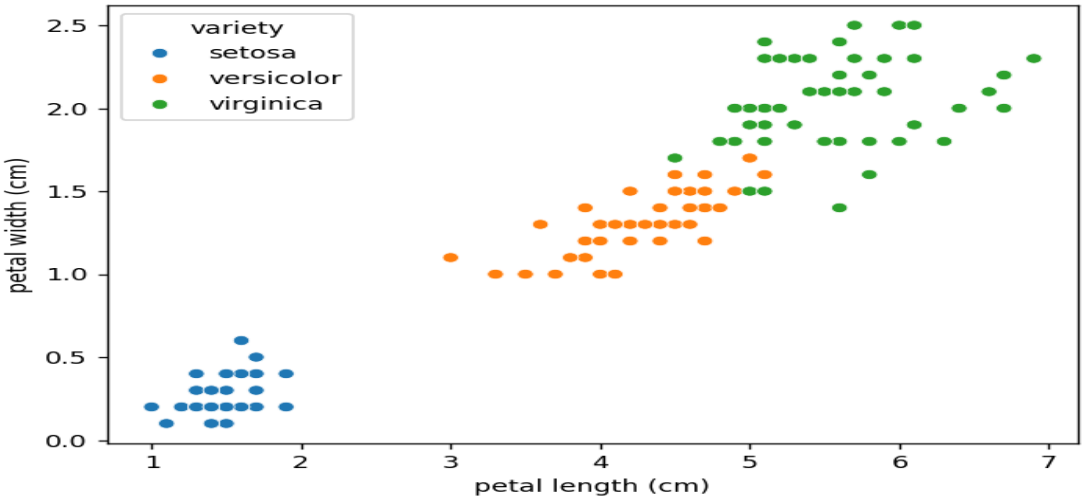
```
   Id  sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) variety
0   1                5.1               3.5                1.4               0.2  setosa
1   2                4.9               3.0                1.4               0.2  setosa
2   3                4.7               3.2                1.3               0.2  setosa
3   4                4.6               3.1                1.5               0.2  setosa
4   5                5.0               3.6                1.4               0.2  setosa
<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   sepal length (cm)  150 non-null    float64
 2   sepal width (cm)   150 non-null    float64
 3   petal length (cm)  150 non-null    float64
 4   petal width (cm)   150 non-null    float64
 5   variety            150 non-null    category
dtypes: category(1), float64(4), int64(1)
memory usage: 6.3 KB
None
```

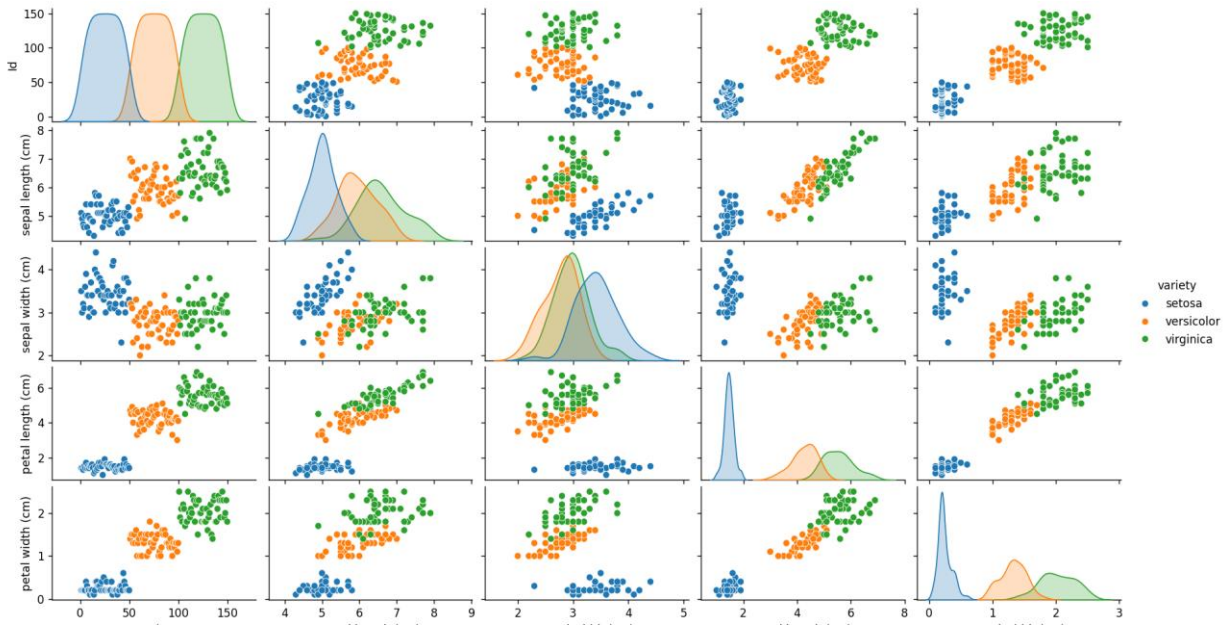
	Id	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.057333	3.758000	1.199333
std	43.445368	0.828066	0.435866	1.765298	0.762238
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	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

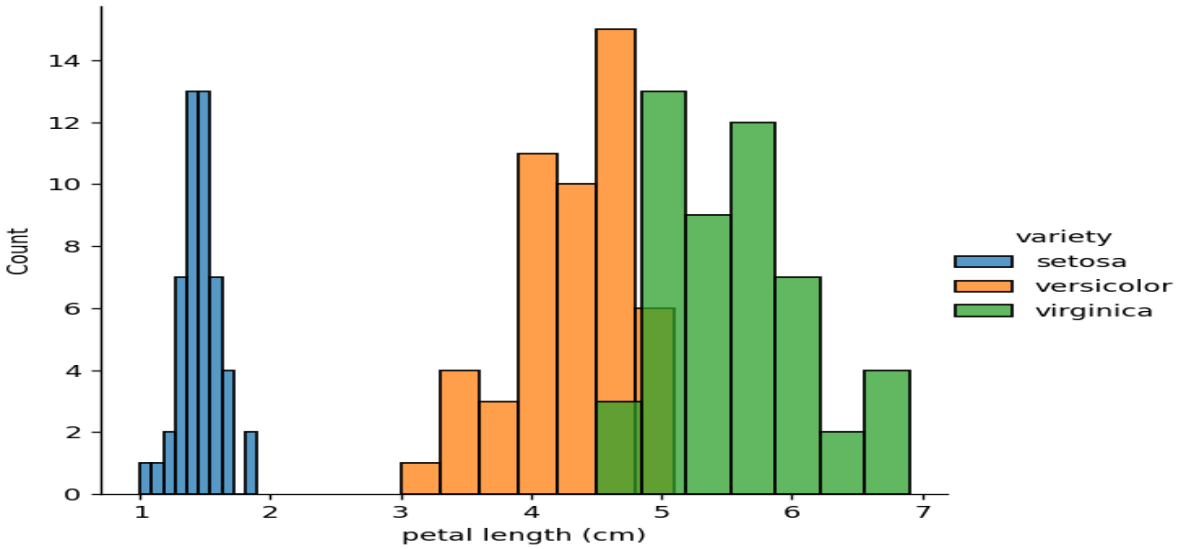
```
variety
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

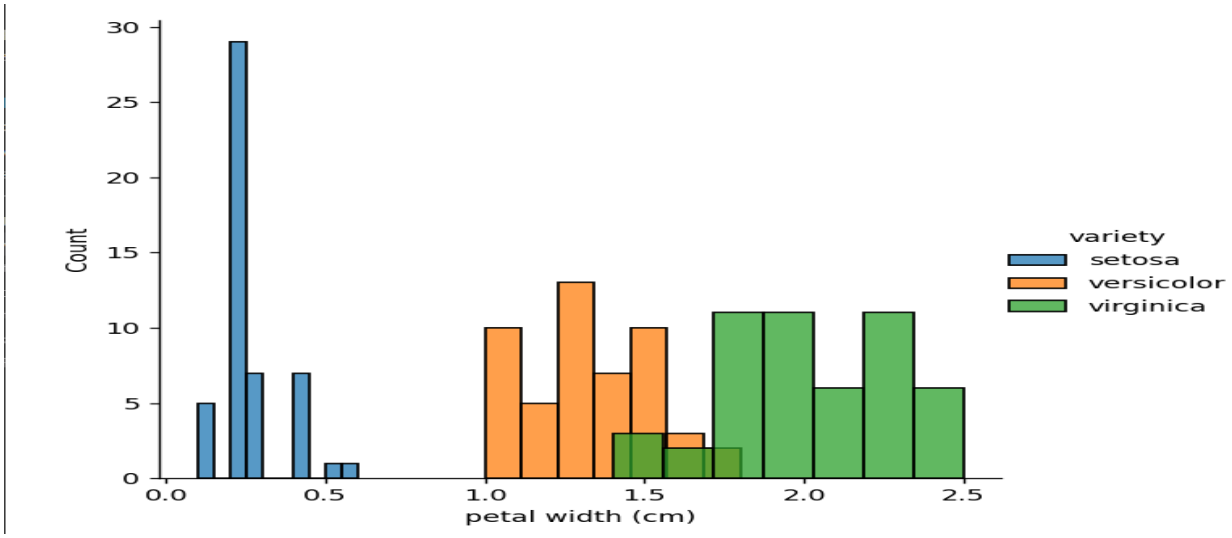


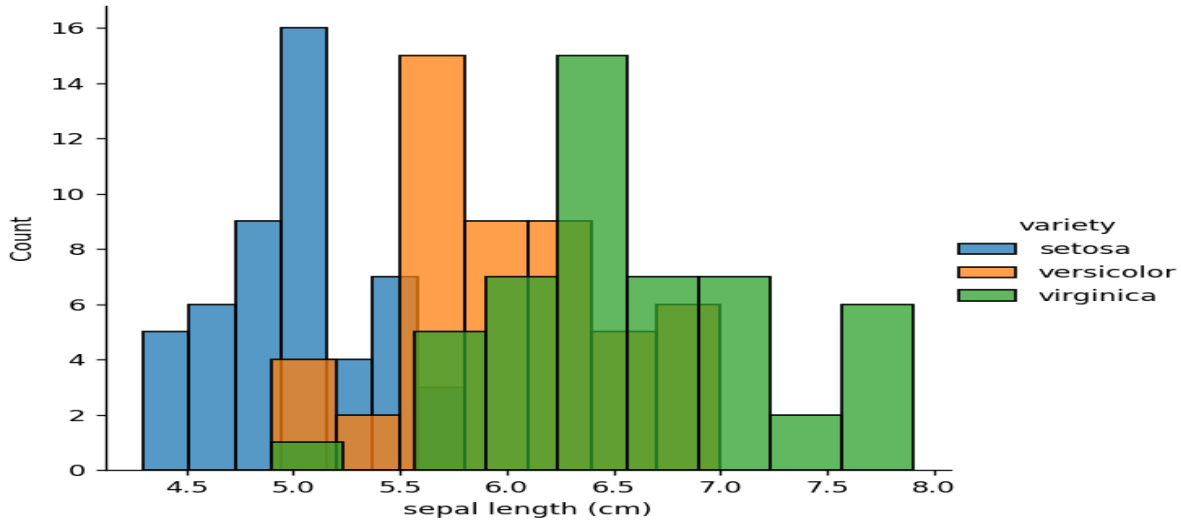


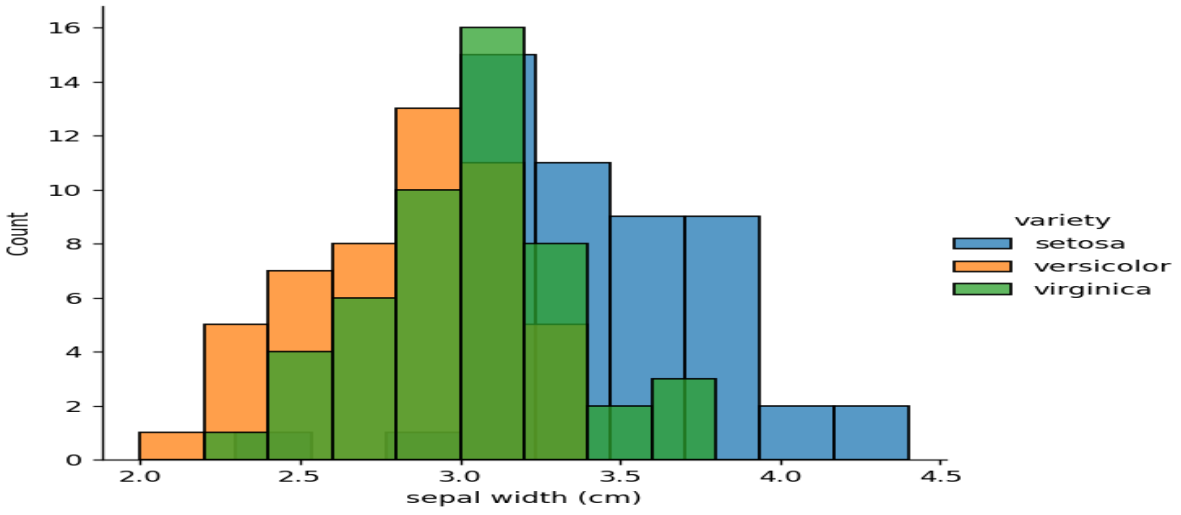

variety
setosa
versicolor
virginica
petal width (cm)
petal length (cm)


Id
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
variety
setosa
versicolor
virginica


Count
petal length (cm)
variety
setosa
versicolor
virginica


Count
petal width (cm)
variety
setosa
versicolor
virginica


Count
sepal length (cm)
variety
setosa
versicolor
virginica


Count
sepal width (cm)
variety
setosa
versicolor
virginica

Exp No: 10 Experiment to understand array function in Data science.

Description: Understand array function using Numpy library

Program:

```
import numpy as np
array = np.random.randint(1, 100, 9)
print(array)
print(np.sqrt(array))
print(array.ndim)
print(array.size)
print(np.sum(array))
print(np.max(array))
print(np.min(array))
new_array = array.reshape(3, 3)
print(new_array)
print(new_array.ndim)
print(new_array.shape)
print(new_array.ravel())
newm = new_array.reshape(3, 3)
print(newm)
print(newm[2, 1:3])
print(newm[1:2, 1:3])
print(new_array[0:3, 0:0])
print(new_array[0:2, 0:1])
print(new_array[0:3, 0:1])
print(new_array[1:3])
```

Output:

```
[72 72 14 22 94 27 75 85 46]
[8.48528137 8.48528137 3.74165739 4.69041576 9.69535971 5.19615242
 8.66025404 9.21954446 6.78232998]
1
9
507
94
14
[[72 72 14]
 [22 94 27]
 [75 85 46]]
2
(3, 3)
[72 72 14 22 94 27 75 85 46]
[[72 72 14]
 [22 94 27]
 [75 85 46]]
[85 46]
[[94 27]]
[]
[[72]
 [22]]
[[72]
 [22]
 [75]]
[[22 94 27]
 [75 85 46]]
```

Exp No: 11 Experiment to understand pandas library use cases in Data science.

Description: Understand data frame use cases using pandas library

Program:

```
import pandas as pd
import numpy as np
df = pd.DataFrame([['1', 'Smith', 50000], ['2', 'Jones', 60000]])
print(df)
df.columns = ['Empid', 'Name', 'Salary']
print(df)
df.info()
df_startups = pd.DataFrame({
    'R&D Spend': [165349.20, 162597.70, 153441.51, 144372.41, 142107.34,
                  1000.23, 1315.46, 0.00, 542.05, 0.00],
    'Administration': [136897.80, 151377.59, 101145.55, 118671.85, 91391.77,
                       124153.04, 115816.21, 135426.92, 51743.15, 116983.80],
    'Marketing Spend': [471784.10, 443898.53, 407934.54, 383199.62, 366168.42,
                        1903.93, 297114.46, 0.00, 0.00, 45173.06],
    'State': ['New York', 'California', 'Florida', 'New York', 'Florida',
              'New York', 'Florida', 'California', 'New York', 'California'],
    'Profit': [192261.83, 191792.06, 191050.39, 182901.99, 166997.43,
               64926.08, 49490.75, 42559.73, 35673.41, 14681.40]
})
df_startups.info()
print(df_startups.head())
print(df_startups.tail())
df_employee = pd.DataFrame({
    'emp id': [1, 2, 3, 4, 5, 6, 7],
    'name': ['SREE VARSSINI K S', 'SREEMATHI B', 'SREYA G', 'SREYASKARI MULLAPUDI',
             'SRI AKASH U G', 'SRI HARSHAVARDHANAN R', 'SRI HARSHAVARDHANAN R'],
    'salary': [5000, 6000, 7000, 5000, 8000, 3000, 6000]
})
```

```
print(df_employee.head())
print(df_employee.tail())
df_employee.info()
print(df_employee['salary'])
print(type(df_employee['salary']))
print(df_employee['salary'].mean())
print(df_employee['salary'].median())
print(df_employee['salary'].mode())
print(df_employee['salary'].var())
print(df_employee['salary'].std())
print(df_employee.describe())
print(df_employee.describe(include='all'))
empCol = df_employee.columns
print(df_employee.columns)
emparray = df_employee.values
print(emparray)
employee_DF = pd.DataFrame(emparray, columns=empCol)
print(employee_DF)
```

Output:

```
   0      1      2
0  1  Smith  50000
1  2  Jones  60000
  Empid   Name  Salary
0     1  Smith   50000
1     2  Jones   60000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Empid   2 non-null      object
 1   Name    2 non-null      object
 2   Salary  2 non-null      int64
dtypes: int64(1), object(2)
memory usage: 180.0+ bytes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        10 non-null     float64
 1   Administration   10 non-null     float64
 2   Marketing Spend  10 non-null     float64
 3   State            10 non-null     object
 4   Profit           10 non-null     float64
dtypes: float64(4), object(1)
memory usage: 532.0+ bytes
```

```
   R&D Spend  Administration  Marketing Spend       State     Profit
0  165349.20       136897.80        471784.10    New York  192261.83
1  162597.70       151377.59        443898.53  California  191792.06
2  153441.51       101145.55        407934.54     Florida  191050.39
3  144372.41       118671.85        383199.62    New York  182901.99
4  142107.34        91391.77        366168.42     Florida  166997.43
   R&D Spend  Administration  Marketing Spend       State    Profit
5    1000.23       124153.04          1903.93    New York  64926.08
6    1315.46       115816.21        297114.46     Florida  49490.75
7       0.00       135426.92             0.00  California  42559.73
8     542.05        51743.15             0.00    New York  35673.41
9       0.00       116983.80         45173.06  California  14681.40
   emp id                 name  salary
0       1    SREE VARSSINI K S    5000
1       2          SREEMATHI B    6000
2       3              SREYA G    7000
3       4  SREYASKARI MULLAPUDI    5000
4       5        SRI AKASH U G    8000
   emp id                   name  salary
2       3                SREYA G    7000
3       4  SREYASKARI MULLAPUDI     5000
4       5         SRI AKASH U G    8000
5       6  SRI HARSHAVARDHANAN R    3000
6       7  SRI HARSHAVARDHANAN R    6000
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   emp id  7 non-null      int64
 1   name    7 non-null      object
 2   salary  7 non-null      int64
dtypes: int64(2), object(1)
memory usage: 300.0+ bytes
0    5000
1    6000
2    7000
3    5000
4    8000
5    3000
6    6000
Name: salary, dtype: int64
<class 'pandas.core.series.Series'>
5714.285714285715
6000.0
0    5000
1    6000
Name: salary, dtype: int64
2571428.5714285714
1603.5674514745463
          emp id       salary
count   7.000000     7.000000
mean    4.000000  5714.285714
std     2.160247  1603.567451
min     1.000000  3000.000000
25%     2.500000  5000.000000
50%     4.000000  6000.000000
75%     5.500000  6500.000000
max     7.000000  8000.000000
```

	emp id	name	salary
count	7.000000	7	7.000000
unique	NaN	6	NaN
top	NaN	SRI HARSHAVARDHANAN R	NaN
freq	NaN	2	NaN
mean	4.000000	NaN	5714.285714
std	2.160247	NaN	1603.567451
min	1.000000	NaN	3000.000000
25%	2.500000	NaN	5000.000000
50%	4.000000	NaN	6000.000000
75%	5.500000	NaN	6500.000000
max	7.000000	NaN	8000.000000

```
Index(['emp id', 'name', 'salary'], dtype='object')
[[1 'SREE VARSSINI K S' 5000]
 [2 'SREEMATHI B' 6000]
 [3 'SREYA G' 7000]
 [4 'SREYASKARI MULLAPUDI' 5000]
 [5 'SRI AKASH U G' 8000]
 [6 'SRI HARSHAVARDHANAN R' 3000]
 [7 'SRI HARSHAVARDHANAN R' 6000]]
   emp id                    name salary
0       1       SREE VARSSINI K S   5000
1       2             SREEMATHI B   6000
2       3                 SREYA G   7000
3       4    SREYASKARI MULLAPUDI   5000
4       5           SRI AKASH U G   8000
5       6  SRI HARSHAVARDHANAN R   3000
6       7  SRI HARSHAVARDHANAN R   6000
```

Exp No: 12 Experiment to detect outliers in a given data set.

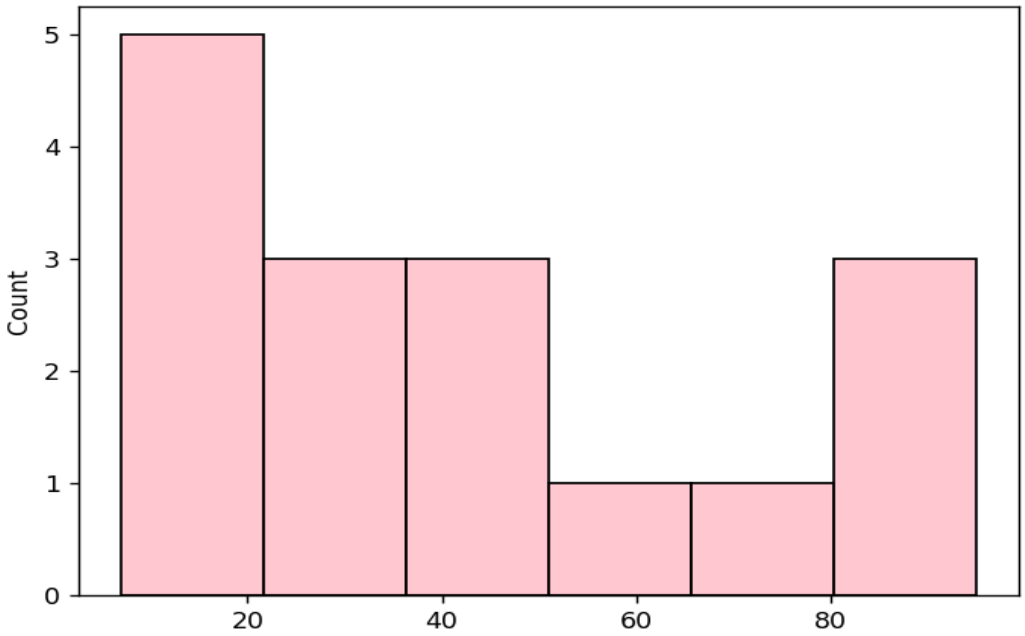
Description: Understand the procedure to identify the outliers in a given dataset

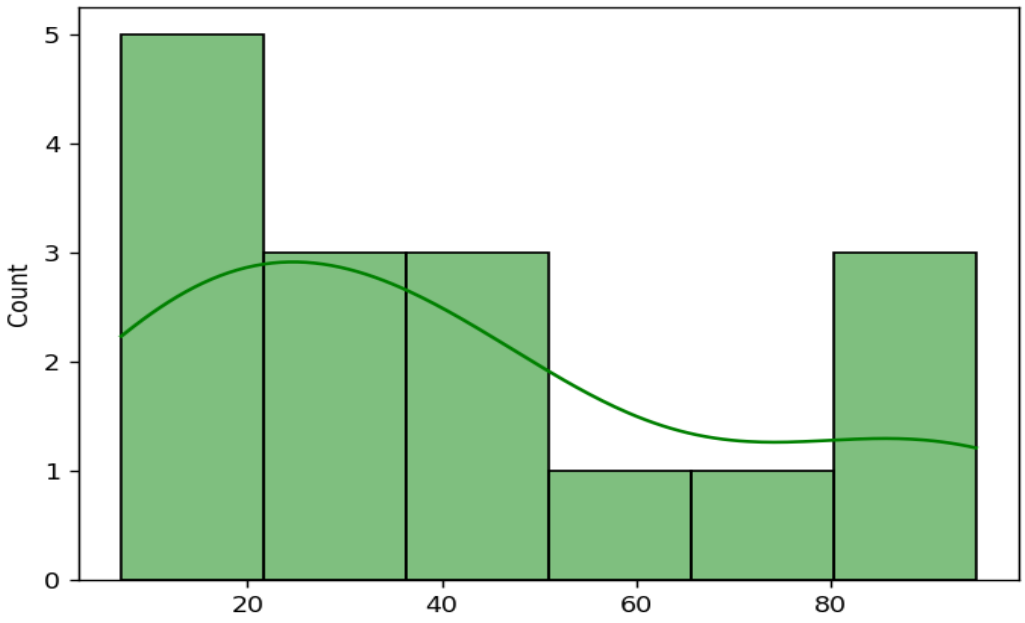
Program:

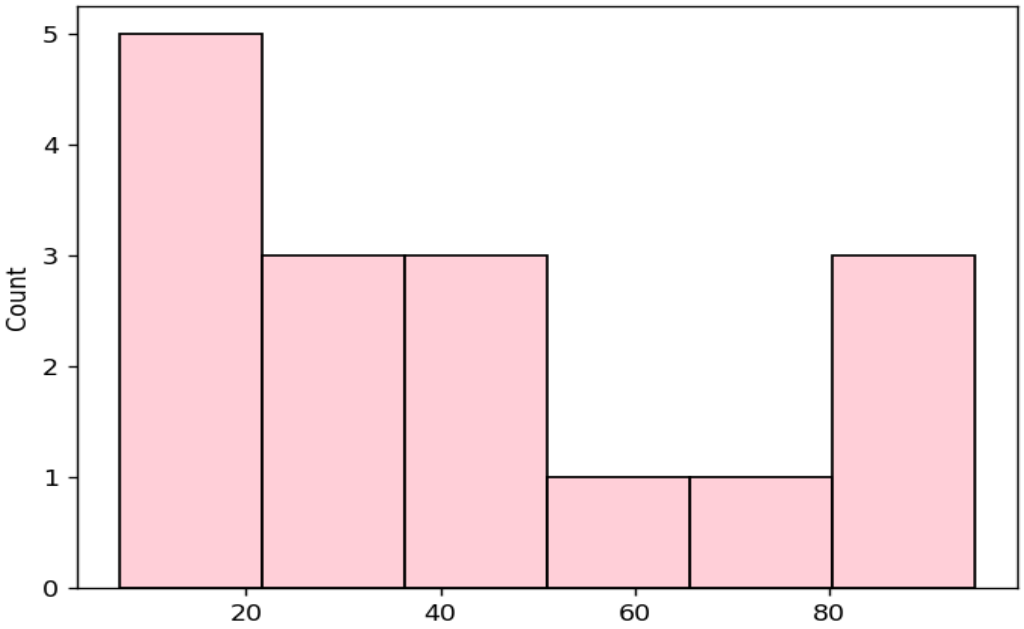
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
array = np.random.randint(1, 100, 16)
print(array)
print(array.mean())
print(np.percentile(array, 25))
print(np.percentile(array, 50))
print(np.percentile(array, 75))
print(np.percentile(array, 100))
def outDetection(array):
    Q1, Q3 = np.percentile(array, [25, 75])
    IQR = Q3 - Q1
    lr = Q1 - (1.5 * IQR)
    ur = Q3 + (1.5 * IQR)
    return lr, ur
lr, ur = outDetection(array)
print(lr, ur)
sns.histplot(array, bins=6, kde=False, color="lightpink")
plt.show()
sns.histplot(array, bins=6, kde=True, color="green")
plt.show()
final_array = array[(array >= lr) & (array <= ur)]
print(final_array)
sns.histplot(final_array, bins=6, kde=False, color="pink")
plt.show()
sns.histplot(final_array, bins=6, kde=True, color="brown")
plt.show()
```

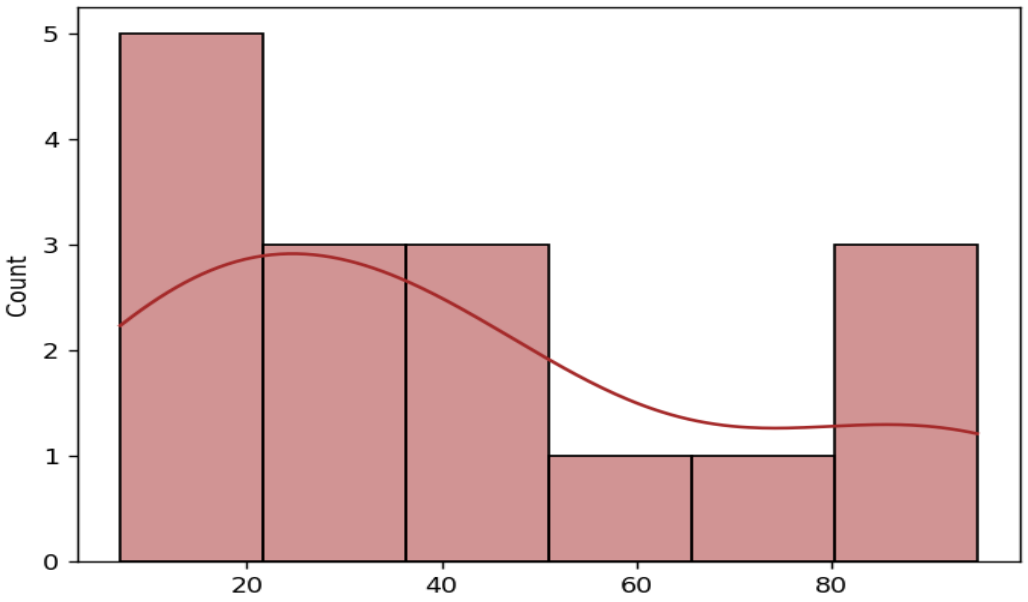
Output:

```
[11 10 41 24 95 78 24 14 47 18 38 95 33  7 52 91]
42.375
17.0
35.5
58.5
95.0
-45.25 120.75
[11 10 41 24 95 78 24 14 47 18 38 95 33  7 52 91]
```


Count
0
1
2
3
4
5
20
40
60
80


Count
0
1
2
3
4
5
20
40
60
80


Count
0
1
2
3
4
5
20
40
60
80


Count
0
1
2
3
4
5
20
40
60
80

Exp No: 13 Experiment to handle missing values and inappropriate data in a given data set

Description: Understand the procedure to handle missing and inappropriate data

Program:

```
import numpy as np
import pandas as pd

data = {
    "CustomerID": [1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 10],
    "Age_Group": ["20-25", "30-35", "25-30", "20-25", "35+", "25-30", "35+", "20-25", "25-30", "25-30", "30-35"],
    "Rating(1-5)": [4, 5, 6, -1, 3, 3, 4, 7, 2, 2, 5],
    "Hotel": ["Ibis", "LemonTree", "RedFox", "LemonTree", "Ibis", "Ibys", "RedFox", "LemonTree", "Ibis", "Ibis", "RedFox"],
    "FoodPreference": ["veg", "Non-Veg", "Veg", "Veg", "Vegetarian", "Non-Veg", "Vegetarian", "Veg", "Non-Veg", "Non-Veg", "non-Veg"],
    "Bill": [1300, 2000, 1322, 1234, 989, 1909, 1000, 2999, 3456, 3456, -6754],
    "NoOfPax": [2, 3, 2, 2, 2, 2, -1, -10, 3, 3, 4],
    "EstimatedSalary": [40000, 50000, 30000, 120000, 45000, 123000, 21122, 345673, -99999, -99999, 81777],
    "Age_Group1": ["20-25", "30-35", "25-30", "20-25", "35+", "25-30", "35+", "20-25", "25-30", "25-30", "30-35"]
}
df = pd.DataFrame(data)
print(df)
print(df.duplicated())
print(df.info())
df.drop_duplicates(inplace = True)
print(df)
print(len(df))
index = np.array(list(range(0, len(df))))
df.set_index(index, inplace = True)
print(index)
print(df)
df.drop(['Age_Group1'], axis = 1, inplace = True)
print(df)
df.loc[df.CustomerID < 0, 'CustomerID'] = np.nan
df.loc[df.Bill < 0, 'Bill'] = np.nan
df.loc[df.EstimatedSalary < 0, 'EstimatedSalary'] = np.nan
df.loc[(df["Rating(1-5)"] < 1) | (df["Rating(1-5)"] > 5), "Rating(1-5)"] = np.nan
df.loc[df["NoOfPax"] <= 0, "NoOfPax"] = np.nan
print(df)
```

```
print(df.Age_Group.unique())
df.loc[:, 'Hotel'] = df['Hotel'].replace(['Ibys'], 'Ibis')
print(df.Hotel.unique())
df.loc[:, 'FoodPreference'] = df['FoodPreference'].str.lower().replace({
    'veg': 'Veg',
    'vegetarian': 'Veg',
    'non-veg': 'Non-Veg'
})
print(df.FoodPreference.unique())
df.loc[:, 'EstimatedSalary'] = df['EstimatedSalary'].fillna(int(round(df['EstimatedSalary'].mean())))
df.loc[:, 'NoOfPax'] = df['NoOfPax'].fillna(int(round(df['NoOfPax'].median())))
df.loc[:, 'Rating(1-5)'] = df['Rating(1-5)'].fillna(int(round(df['Rating(1-5)'].median())))
df.loc[:, 'Bill'] = df['Bill'].fillna(int(round(df['Bill'].mean())))
print(df)
```

Output:

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary	Age_Group1
0	1	20-25	4	Ibis	veg	1300	2	40000	20-25
1	2	30-35	5	LemonTree	Non-Veg	2000	3	50000	30-35
2	3	25-30	6	RedFox	Veg	1322	2	30000	25-30
3	4	20-25	-1	LemonTree	Veg	1234	2	120000	20-25
4	5	35+	3	Ibis	Vegetarian	989	2	45000	35+
5	6	25-30	3	Ibys	Non-Veg	1909	2	123000	25-30
6	7	35+	4	RedFox	Vegetarian	1000	-1	21122	35+
7	8	20-25	7	LemonTree	Veg	2999	-10	345673	20-25
8	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
9	9	25-30	2	Ibis	Non-Veg	3456	3	-99999	25-30
10	10	30-35	5	RedFox	non-Veg	-6754	4	81777	30-35

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9      True
10    False
dtype: bool
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CustomerID       11 non-null     int64
 1   Age_Group        11 non-null     object
 2   Rating(1-5)      11 non-null     int64
 3   Hotel            11 non-null     object
 4   FoodPreference   11 non-null     object
 5   Bill             11 non-null     int64
 6   NoOfPax          11 non-null     int64
 7   EstimatedSalary  11 non-null     int64
 8   Age_Group1       11 non-null     object
dtypes: int64(5), object(4)
memory usage: 924.0+ bytes
None
```

```
    CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  NoOfPax  EstimatedSalary Age_Group1
0            1     20-25            4       Ibis            veg  1300        2            40000      20-25
1            2     30-35            5  LemonTree        Non-Veg  2000        3            50000      30-35
2            3     25-30            6     RedFox            Veg  1322        2            30000      25-30
3            4     20-25           -1  LemonTree            Veg  1234        2           120000      20-25
4            5       35+            3       Ibis     Vegetarian   989        2            45000        35+
5            6     25-30            3       Ibys        Non-Veg  1909        2           123000      25-30
6            7       35+            4     RedFox     Vegetarian  1000       -1            21122        35+
7            8     20-25            7  LemonTree            Veg  2999      -10           345673      20-25
8            9     25-30            2       Ibis        Non-Veg  3456        3           -99999      25-30
9           10     30-35            5     RedFox        non-Veg -6754        4            81777      30-35
10
[0 1 2 3 4 5 6 7 8 9]
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  NoOfPax  EstimatedSalary Age_Group1
0           1     20-25            4       Ibis            veg  1300        2            40000      20-25
1           2     30-35            5  LemonTree        Non-Veg  2000        3            50000      30-35
2           3     25-30            6     RedFox            Veg  1322        2            30000      25-30
3           4     20-25           -1  LemonTree            Veg  1234        2           120000      20-25
4           5       35+            3       Ibis     Vegetarian   989        2            45000        35+
5           6     25-30            3       Ibys        Non-Veg  1909        2           123000      25-30
6           7       35+            4     RedFox     Vegetarian  1000       -1            21122        35+
7           8     20-25            7  LemonTree            Veg  2999      -10           345673      20-25
8           9     25-30            2       Ibis        Non-Veg  3456        3           -99999      25-30
9          10     30-35            5     RedFox        non-Veg -6754        4            81777      30-35
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  NoOfPax  EstimatedSalary
0           1     20-25            4       Ibis            veg  1300        2            40000
1           2     30-35            5  LemonTree        Non-Veg  2000        3            50000
2           3     25-30            6     RedFox            Veg  1322        2            30000
3           4     20-25           -1  LemonTree            Veg  1234        2           120000
4           5       35+            3       Ibis     Vegetarian   989        2            45000
5           6     25-30            3       Ibys        Non-Veg  1909        2           123000
6           7       35+            4     RedFox     Vegetarian  1000       -1            21122
7           8     20-25            7  LemonTree            Veg  2999      -10           345673
8           9     25-30            2       Ibis        Non-Veg  3456        3           -99999
9          10     30-35            5     RedFox        non-Veg -6754        4            81777
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4.0	Ibis	veg	1300.0	2.0	40000.0
1	2.0	30-35	5.0	LemonTree	Non-Veg	2000.0	3.0	50000.0
2	3.0	25-30	NaN	RedFox	Veg	1322.0	2.0	30000.0
3	4.0	20-25	NaN	LemonTree	Veg	1234.0	2.0	120000.0
4	5.0	35+	3.0	Ibis	Vegetarian	989.0	2.0	45000.0
5	6.0	25-30	3.0	Ibys	Non-Veg	1909.0	2.0	123000.0
6	7.0	35+	4.0	RedFox	Vegetarian	1000.0	NaN	21122.0
7	8.0	20-25	NaN	LemonTree	Veg	2999.0	NaN	345673.0
8	9.0	25-30	2.0	Ibis	Non-Veg	3456.0	3.0	NaN
9	10.0	30-35	5.0	RedFox	non-Veg	NaN	4.0	81777.0

```
['20-25' '30-35' '25-30' '35+']
['Ibis' 'LemonTree' 'RedFox']
['Veg' 'Non-Veg']
```

	CustomerID	Age_Group	Rating(1-5)	Hotel	FoodPreference	Bill	NoOfPax	EstimatedSalary
0	1.0	20-25	4.0	Ibis	Veg	1300.0	2.0	40000.0
1	2.0	30-35	5.0	LemonTree	Non-Veg	2000.0	3.0	50000.0
2	3.0	25-30	4.0	RedFox	Veg	1322.0	2.0	30000.0
3	4.0	20-25	4.0	LemonTree	Veg	1234.0	2.0	120000.0
4	5.0	35+	3.0	Ibis	Veg	989.0	2.0	45000.0
5	6.0	25-30	3.0	Ibis	Non-Veg	1909.0	2.0	123000.0
6	7.0	35+	4.0	RedFox	Veg	1000.0	2.0	21122.0
7	8.0	20-25	4.0	LemonTree	Veg	2999.0	2.0	345673.0
8	9.0	25-30	2.0	Ibis	Non-Veg	3456.0	3.0	95175.0
9	10.0	30-35	5.0	RedFox	Non-Veg	1801.0	4.0	81777.0

Exp No: 14 Experiment to understand feature scaling.

Description: Understand the importance of feature scaling.

Program:

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
df = pd.DataFrame({
    'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany',
                'France', 'Spain', 'France', np.nan, 'France'],
    'Age': [44, 27, 30, 38, 40, 35, np.nan, 48, 50, 37],
    'Salary': [72000, 48000, 54000, 61000, np.nan, 58000, 52000, 79000, 83000, 67000],
    'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']
})
print("Initial Dataset:\n", df)
print("\nFirst 5 Rows:\n", df.head())
df['Country'] = df['Country'].fillna(df['Country'].mode()[0])
features = df.iloc[:, :-1].values
labels = df.iloc[:, -1].values
print("\nFeatures before Imputation:\n", features)
age = SimpleImputer(strategy="mean", missing_values=np.nan)
salary = SimpleImputer(strategy="mean", missing_values=np.nan)
features[:, 1:2] = age.fit_transform(features[:, 1:2])
features[:, 2:3] = salary.fit_transform(features[:, 2:3])
print("\nFeatures after Imputation:\n", features)
oh = OneHotEncoder(sparse_output=False)
Country = oh.fit_transform(features[:, 0:1])
print("\nOneHotEncoded Country:\n", Country)
final_set = np.concatenate((Country, features[:, 1:]), axis=1)
print("\nFinal Dataset before Scaling:\n", final_set)
sc = StandardScaler()
feat_standard_scale = sc.fit_transform(final_set)
print("\nStandard Scaled Features:\n", feat_standard_scale)
ms = MinMaxScaler(feature_range=(0, 1))
feat_minmax_scale = ms.fit_transform(final_set)
print("\nMinMax Scaled Features:\n", feat_minmax_scale)
```

Output:

```
Initial Dataset:
    Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain   NaN  52000.0        No
7   France  48.0  79000.0       Yes
8      NaN  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

```
First 5 Rows:
    Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes

Features before Imputation:
 [['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['France' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

```
Features after Imputation:
 [['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['France' 50.0 83000.0]
 ['France' 37.0 67000.0]]

OneHotEncoded Country:
 [[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

```
Final Dataset before Scaling:
 [[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [1.0 0.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]

Standard Scaled Features:
 [[ 1.00000000e+00 -5.00000000e-01 -6.54653671e-01  7.58874362e-01
    7.49473254e-01]
 [-1.00000000e+00 -5.00000000e-01  1.52752523e+00 -1.71150388e+00
   -1.43817841e+00]
 [-1.00000000e+00  2.00000000e+00 -6.54653671e-01 -1.27555478e+00
   -8.91265492e-01]
 [-1.00000000e+00 -5.00000000e-01  1.52752523e+00 -1.13023841e-01
   -2.53200424e-01]
 [-1.00000000e+00  2.00000000e+00 -6.54653671e-01  1.77608893e-01
    6.63219199e-16]
 [ 1.00000000e+00 -5.00000000e-01 -6.54653671e-01 -5.48972942e-01
   -5.26656882e-01]
 [-1.00000000e+00 -5.00000000e-01  1.52752523e+00  0.00000000e+00
   -1.07356980e+00]
 [ 1.00000000e+00 -5.00000000e-01 -6.54653671e-01  1.34013983e+00
    1.38753832e+00]
 [ 1.00000000e+00 -5.00000000e-01 -6.54653671e-01  1.63077256e+00
    1.75214693e+00]
 [ 1.00000000e+00 -5.00000000e-01 -6.54653671e-01 -2.58340208e-01
    2.93712492e-01]]
```

```
MinMax Scaled Features:
 [[1.         0.         0.         0.73913043 0.68571429]
 [0.         0.         1.         0.         0.        ]
 [0.         1.         0.         0.13043478 0.17142857]
 [0.         0.         1.         0.47826087 0.37142857]
 [0.         1.         0.         0.56521739 0.45079365]
 [1.         0.         0.         0.34782609 0.28571429]
 [0.         0.         1.         0.51207729 0.11428571]
 [1.         0.         0.         0.91304348 0.88571429]
 [1.         0.         0.         1.         1.        ]
 [1.         0.         0.         0.43478261 0.54285714]]
```

Exp No: 15 Experiment to understand the data preprocessing in Data science

Description: Understand the importance of Data preprocessing in data science

Program:

```python
import numpy as np
import pandas as pd
data = {
    'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany',
                'France', 'Spain', 'France', np.nan, 'France'],
    'Age': [44.0, 27.0, 30.0, 38.0, 40.0, 35.0, np.nan, 48.0, 50.0, 37.0],
    'Salary': [72000.0, 48000.0, 54000.0, 61000.0, np.nan,
               58000.0, 52000.0, 79000.0, 83000.0, 67000.0],
    'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']
}
df = pd.DataFrame(data)
print("Initial DataFrame")
print(df)
print("\n")
print("Initial DataFrame Info")
df.info()
print("\n")
print(df.Country.mode())
print("\n")
print(type(df.Country.mode()))
print("\n")
mode_country = df['Country'].mode()[0]
df['Country'] = df['Country'].fillna(mode_country)
median_age = df['Age'].median()
df['Age'] = df['Age'].fillna(median_age)
mean_salary = df['Salary'].mean()
df['Salary'] = df['Salary'].fillna(round(mean_salary))
```

```python
print("DataFrame after Filling Missing Values")
print(df)
print("\n")
country_dummies = pd.get_dummies(df['Country'])
print("One-Hot Encoded 'Country' Dummies")
print(country_dummies)
print("\n")
columns_to_keep_intermediate = df.iloc[:, 1:4]
intermediate_dataset = pd.concat([country_dummies, columns_to_keep_intermediate], axis=1)
intermediate_dataset = intermediate_dataset[['France', 'Germany', 'Spain', 'Age', 'Salary', 'Purchased']]

print("Intermediate Dataset")
print(intermediate_dataset)
print("\n")
mapping = {'No': 0, 'Yes': 1}
df['Purchased'] = df['Purchased'].map(mapping).astype('int64')
columns_to_keep_final = df.iloc[:, 1:4]
updated_dataset = pd.concat([country_dummies, columns_to_keep_final], axis=1)
updated_dataset = updated_dataset[['France', 'Germany', 'Spain', 'Age', 'Salary', 'Purchased']]
print("Final Updated Dataset")
print(updated_dataset)
print("\n")
print("Final DataFrame Info")
updated_dataset.info()
```

Output:

```
Initial DataFrame
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain   NaN  52000.0        No
7   France  48.0  79000.0       Yes
8      NaN  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

```
Initial DataFrame Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Country    9 non-null      object
 1   Age        9 non-null      float64
 2   Salary     9 non-null      float64
 3   Purchased  10 non-null     object
dtypes: float64(2), object(2)
memory usage: 452.0+ bytes


0    France
Name: Country, dtype: object


<class 'pandas.core.series.Series'>
```

```
DataFrame after Filling Missing Values
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0  63778.0       Yes
5   France  35.0  58000.0       Yes
6    Spain  38.0  52000.0        No
7   France  48.0  79000.0       Yes
8   France  50.0  83000.0        No
9   France  37.0  67000.0       Yes


One-Hot Encoded 'Country' Dummies
   France  Germany  Spain
0    True    False  False
1   False    False   True
2   False     True  False
3   False    False   True
4   False     True  False
5    True    False  False
6   False    False   True
7    True    False  False
8    True    False  False
9    True    False  False
```

```
Intermediate Dataset
   France  Germany  Spain   Age   Salary Purchased
0    True    False  False  44.0  72000.0        No
1   False    False   True  27.0  48000.0       Yes
2   False     True  False  30.0  54000.0        No
3   False    False   True  38.0  61000.0        No
4   False     True  False  40.0  63778.0       Yes
5    True    False  False  35.0  58000.0       Yes
6   False    False   True  38.0  52000.0        No
7    True    False  False  48.0  79000.0       Yes
8    True    False  False  50.0  83000.0        No
9    True    False  False  37.0  67000.0       Yes


Final Updated Dataset
   France  Germany  Spain   Age   Salary  Purchased
0    True    False  False  44.0  72000.0          0
1   False    False   True  27.0  48000.0          1
2   False     True  False  30.0  54000.0          0
3   False    False   True  38.0  61000.0          0
4   False     True  False  40.0  63778.0          1
5    True    False  False  35.0  58000.0          1
6   False    False   True  38.0  52000.0          0
7    True    False  False  48.0  79000.0          1
8    True    False  False  50.0  83000.0          0
9    True    False  False  37.0  67000.0          1
```

```
Final DataFrame Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   France     10 non-null     bool
 1   Germany    10 non-null     bool
 2   Spain      10 non-null     bool
 3   Age        10 non-null     float64
 4   Salary     10 non-null     float64
 5   Purchased  10 non-null     int64
dtypes: bool(3), float64(2), int64(1)
memory usage: 402.0 bytes
```

Exp No: 16 Experiment to understand EDA Quantitative and Qualitative analysis.

Description: Understand the importance of EDA-Quantitative and Qualitative analysis.

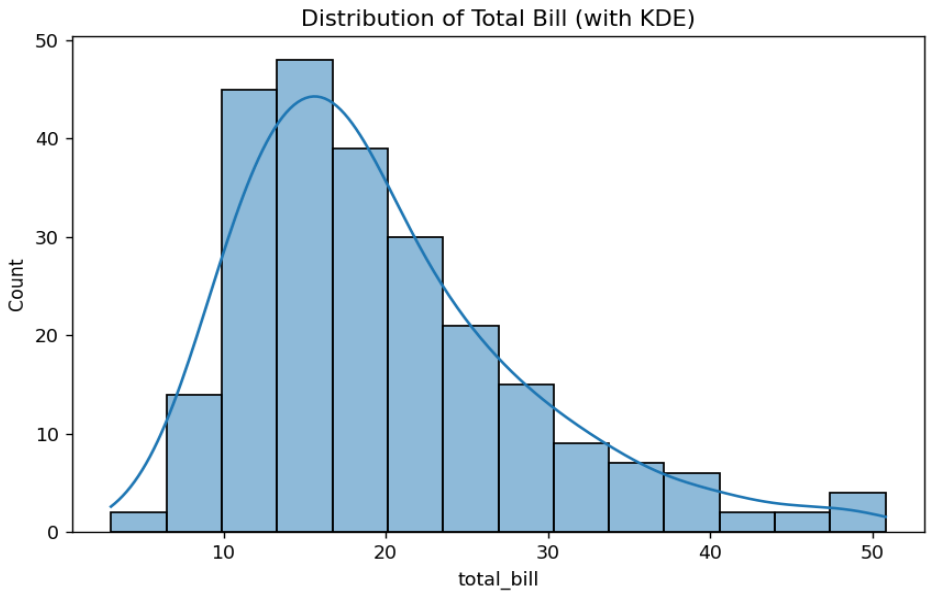
Program:

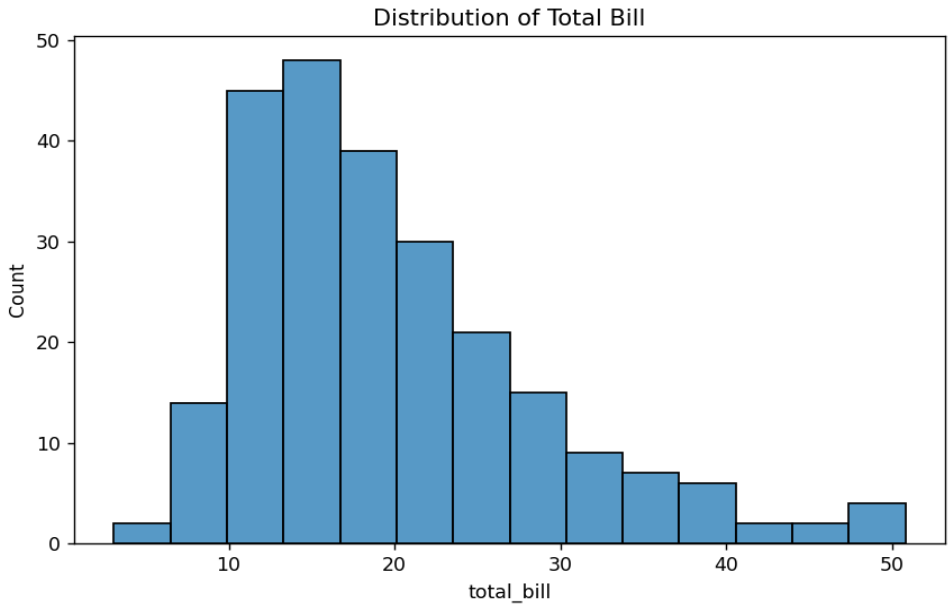
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
tips = sns.load_dataset('tips')
print(tips.head())
plt.figure(figsize=(8, 5))
sns.histplot(tips['total_bill'], kde=True)
plt.title('Distribution of Total Bill (with KDE)')
plt.ylabel('Count')
plt.show()
plt.figure(figsize=(8, 5))
sns.histplot(tips['total_bill'], kde=False)
plt.title('Distribution of Total Bill')
plt.show()
plt.figure(figsize=(4, 6))
sns.boxplot(y=tips['total_bill'])
plt.title("Total Bill Distribution")
plt.ylabel("total_bill")
plt.show()
plt.figure(figsize=(4, 6))
sns.boxplot(y=tips['tip'])
plt.title("Tip Distribution")
plt.ylabel("tip")
plt.show()
```

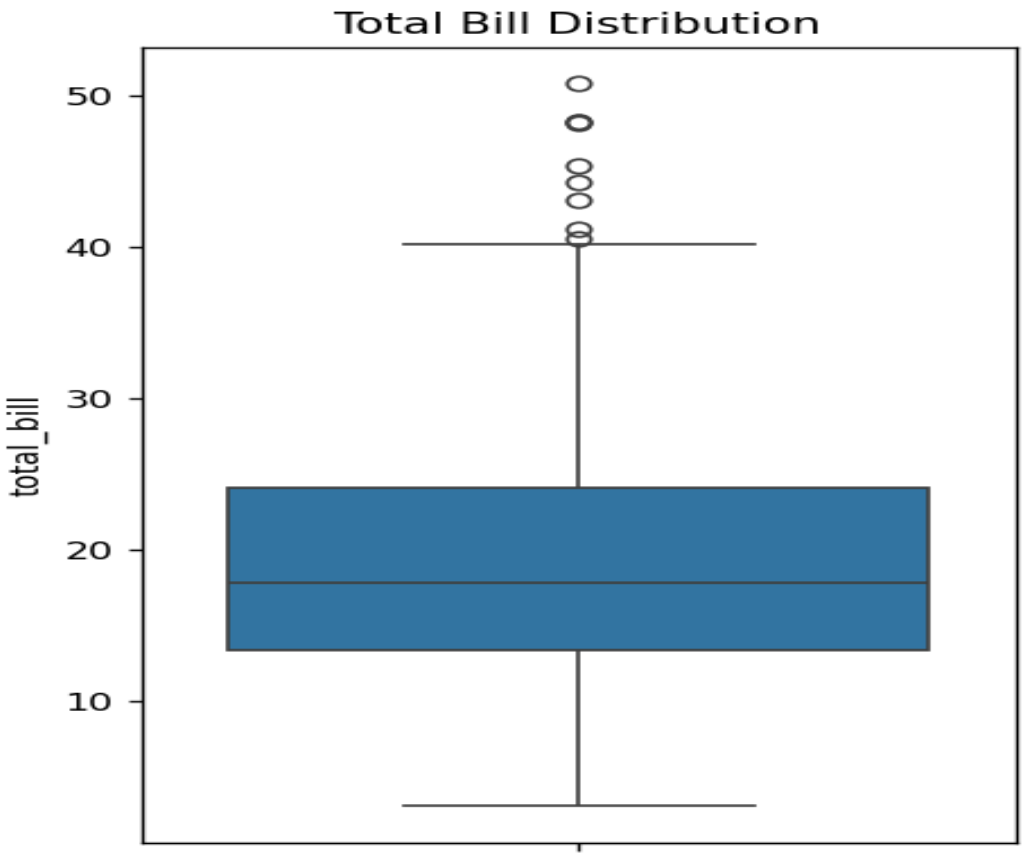
```
plt.figure(figsize=(8, 5))
sns.countplot(y='day', data=tips)
plt.title('Count of Customers by Day')
plt.xlabel('count')
plt.ylabel('day')
plt.show()
plt.figure(figsize=(8, 5))
sns.countplot(y='sex', data=tips)
plt.title('Count of Customers by Sex')
plt.xlabel('count')
plt.ylabel('sex')
plt.show()
plt.figure(figsize=(6, 6))
tips['sex'].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.title('Proportion of Customers by Sex (Pie Chart)')
plt.ylabel('count')
plt.show()
plt.figure(figsize=(8, 5))
tips['sex'].value_counts().plot(kind='bar')
plt.title('Count of Customers by Sex (Bar Chart)')
plt.xlabel('sex')
plt.ylabel('count')
plt.xticks(rotation=0)
plt.show()
print(tips['time'].value_counts())
```

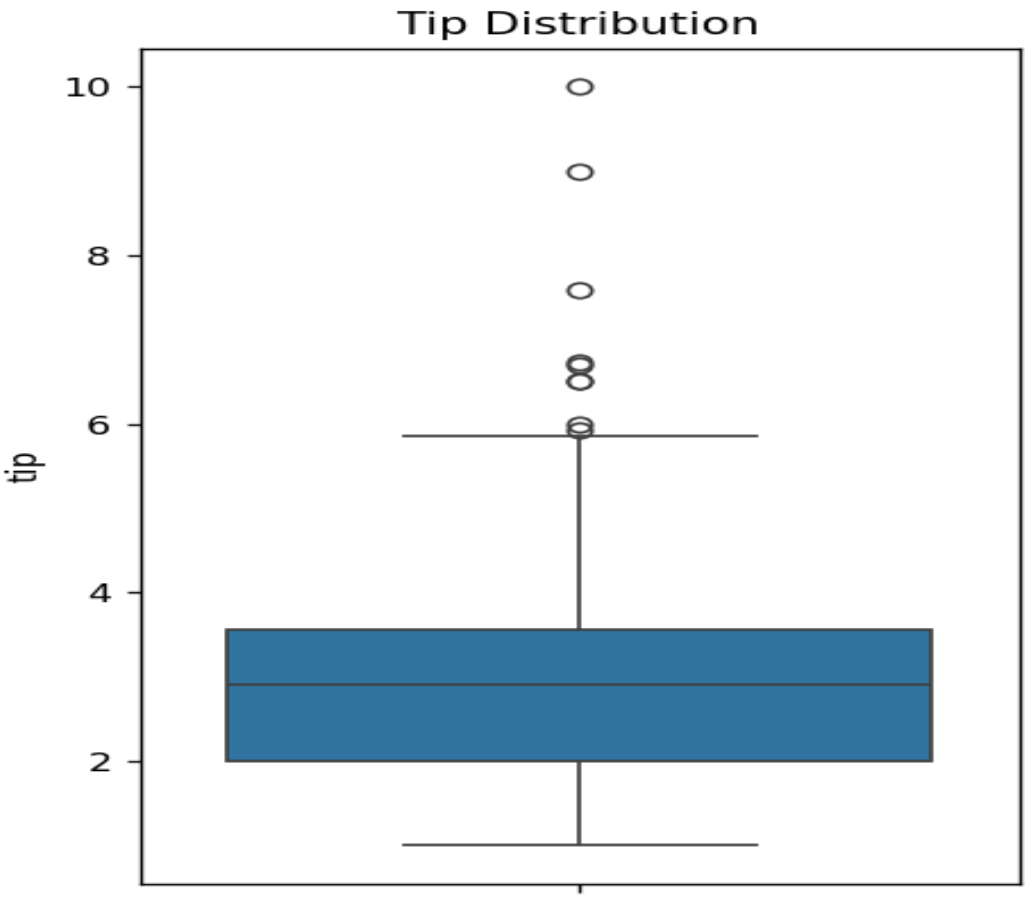
```
correlation_matrix = tips.corr(numeric_only=True)
plt.figure(figsize=(6, 5))
sns.heatmap(correlation_matrix, annot=True, cmap='rocket', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
sns.jointplot(x='tip', y='total_bill', data=tips)
plt.suptitle('Tip vs Total Bill (Scatter)', y=1.02)
plt.show()
sns.jointplot(x='tip', y='total_bill', data=tips, kind='reg')
plt.suptitle('Tip vs Total Bill (Regression)', y=1.02)
plt.show()
sns.jointplot(x='tip', y='total_bill', data=tips, kind='hex')
plt.suptitle('Tip vs Total Bill (Hexbin)', y=1.02)
plt.show()
plt.figure(figsize=(8, 5))
sns.countplot(x='day', data=tips[tips['time'] == 'Dinner'])
plt.title('Count of Dinner Customers by Day')
plt.xlabel('day')
plt.ylabel('count')
plt.show()
sns.pairplot(tips)
plt.suptitle('Pairplot of All Numeric Variables', y=1.02)
plt.show()
sns.pairplot(tips, hue='time')
plt.suptitle('Pairplot colored by Time', y=1.02)
plt.show()
sns.pairplot(tips, hue='day')
plt.suptitle('Pairplot colored by Day', y=1.02)
plt.show()
```

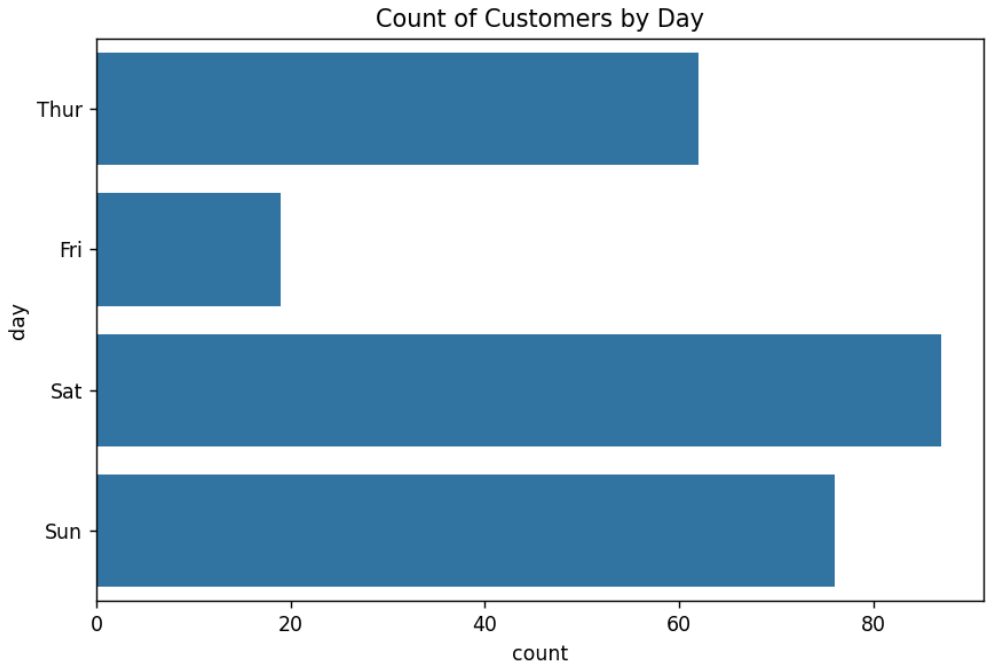
Output:

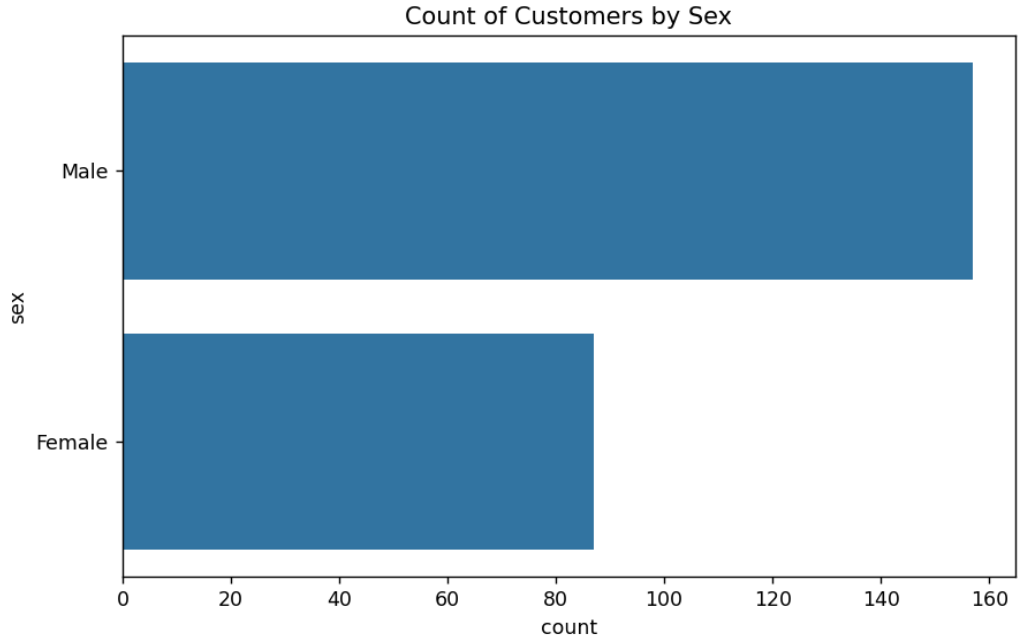


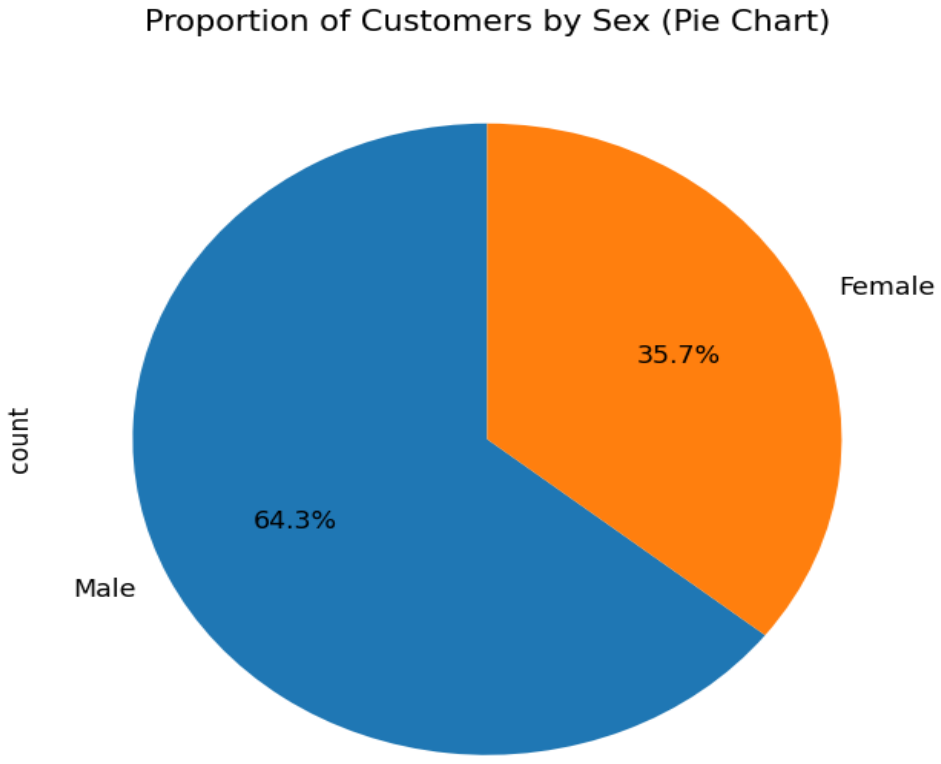

Distribution of Total Bill
Count
total_bill
0
10
20
30
40
50

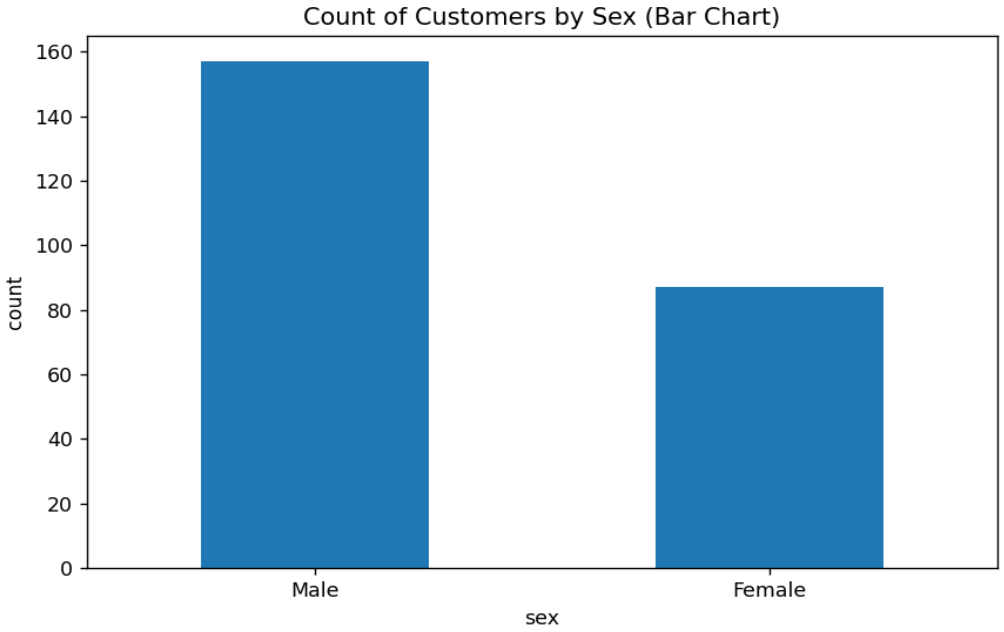

Total Bill Distribution
total_bill
10
20
30
40
50

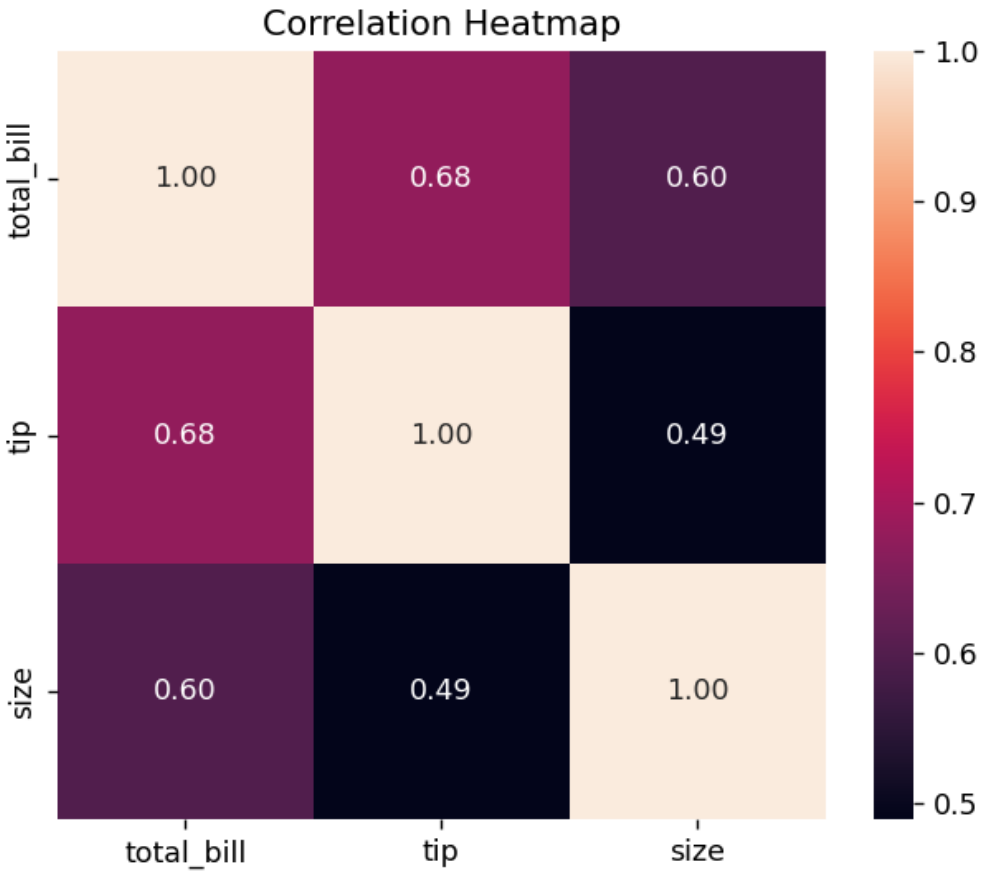

Tip Distribution
tip
10
8
6
4
2

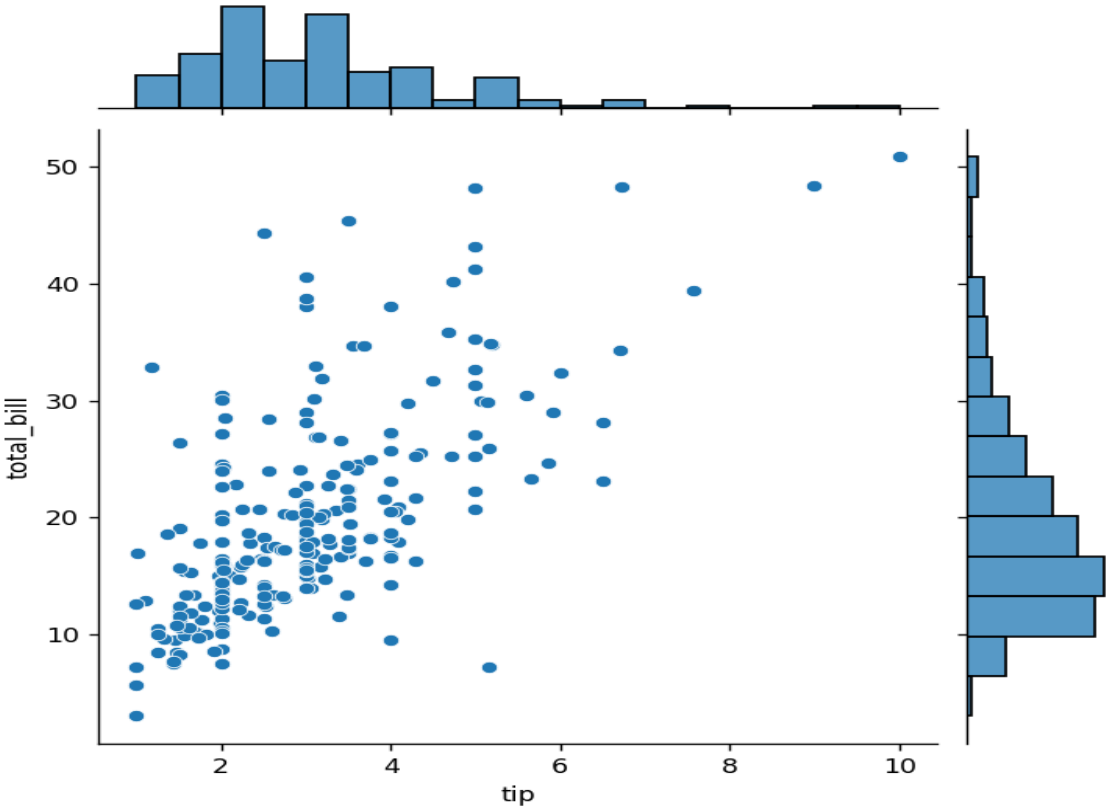

Count of Customers by Day
day
Thur
Fri
Sat
Sun
0
20
40
60
80
count

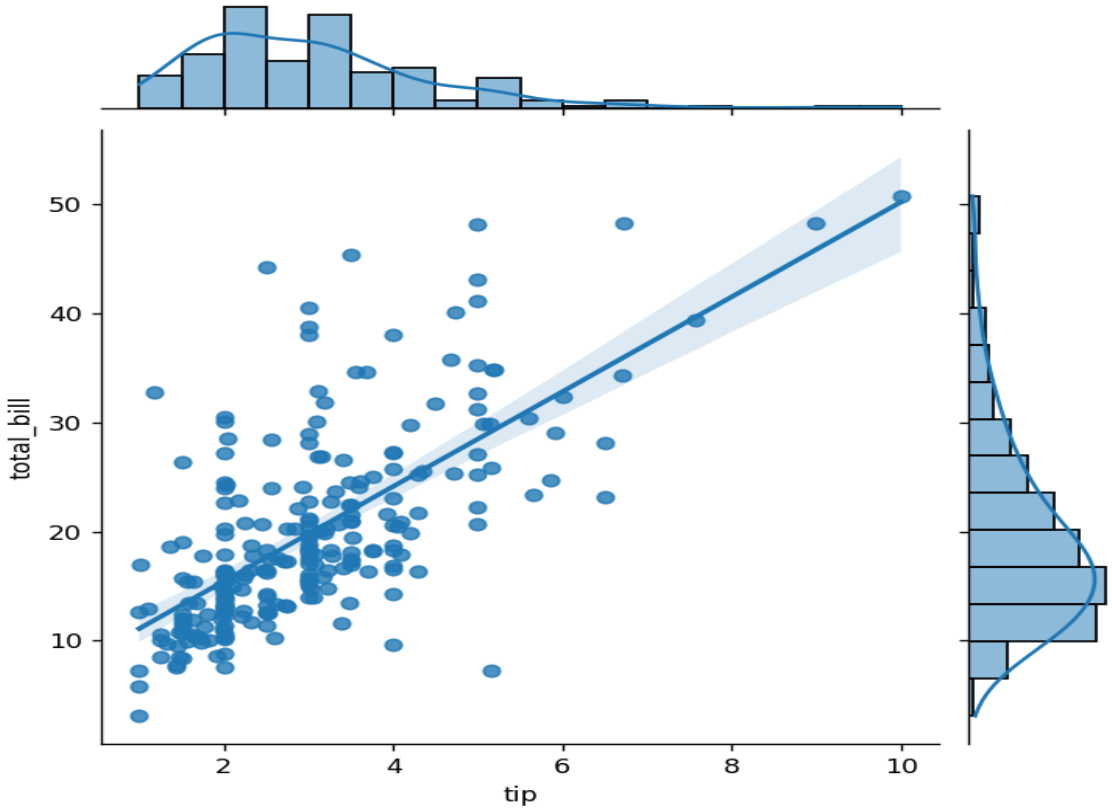

Count of Customers by Sex
Male
Female
sex
0
20
40
60
80
100
120
140
160
count

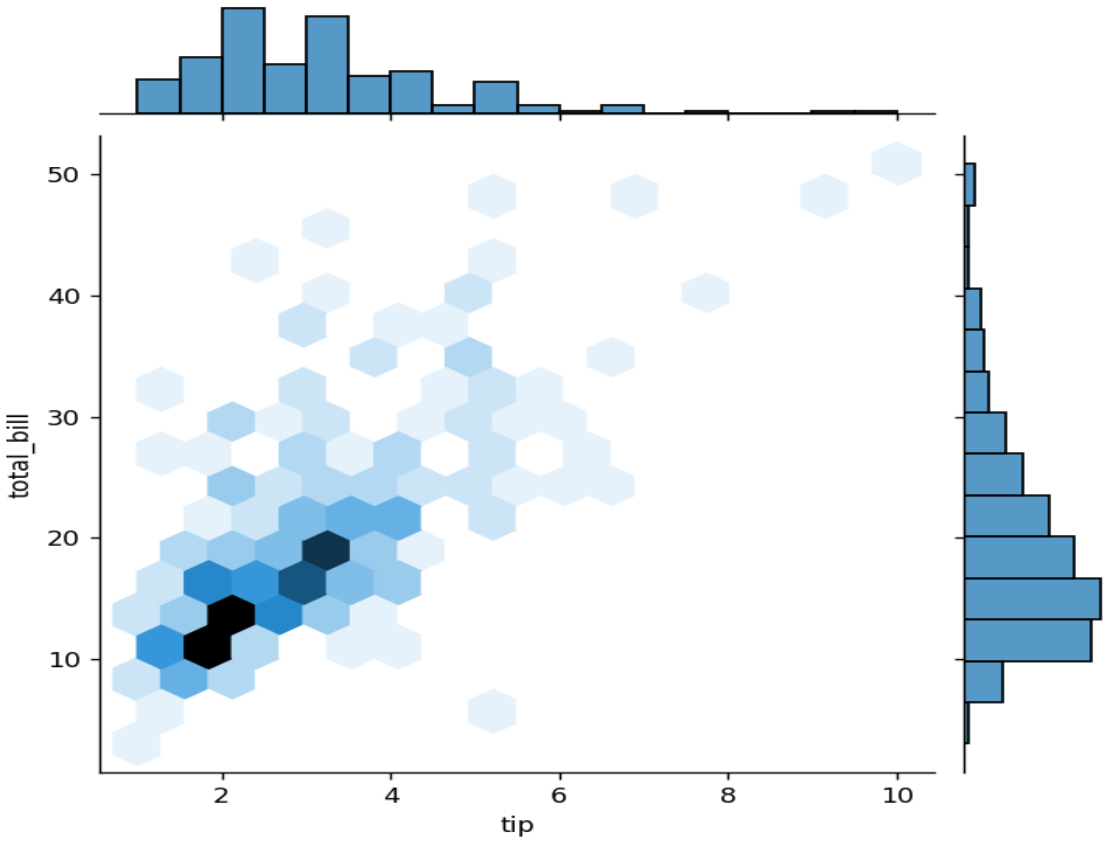

Proportion of Customers by Sex (Pie Chart)
Female
35.7%
count
64.3%
Male

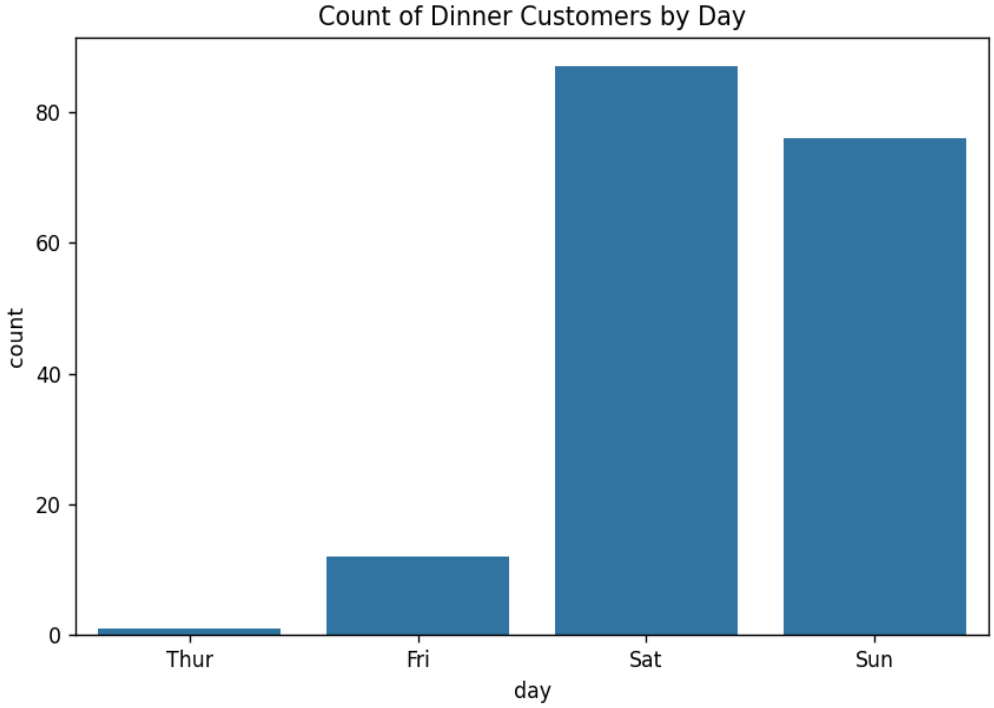

Count of Customers by Sex (Bar Chart)
160
140
120
100
80
60
40
20
0
count
Male
Female
sex

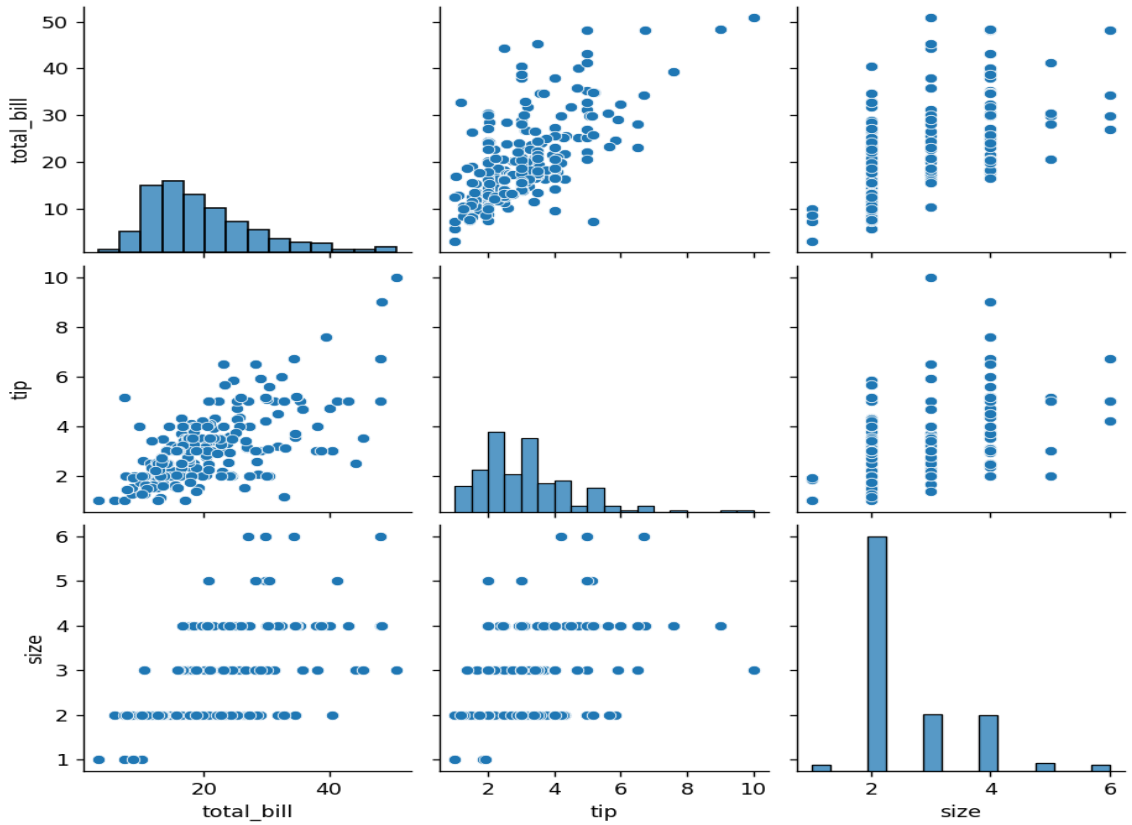

Correlation Heatmap
total_bill
tip
size
1.00
0.68
0.60
0.68
1.00
0.49
0.60
0.49
1.00
total_bill
tip
size
1.0
0.9
0.8
0.7
0.6
0.5

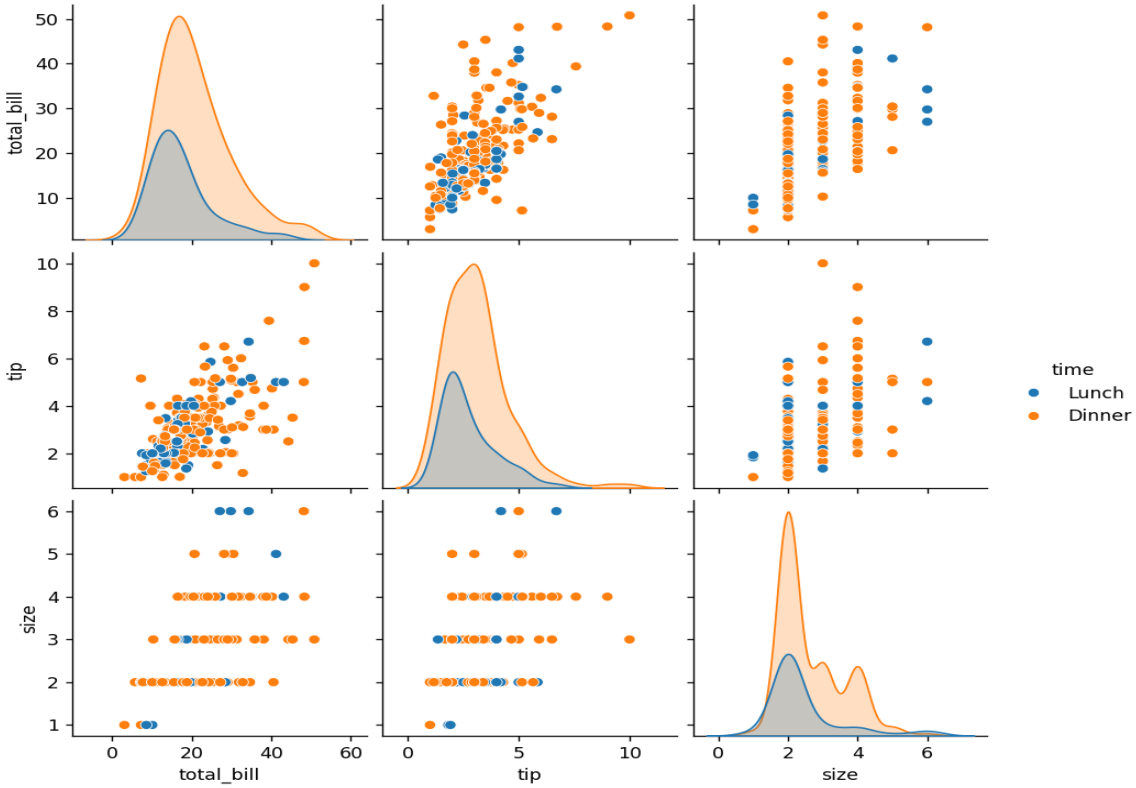

total_bill
tip
50
40
30
20
10
2
4
6
8
10

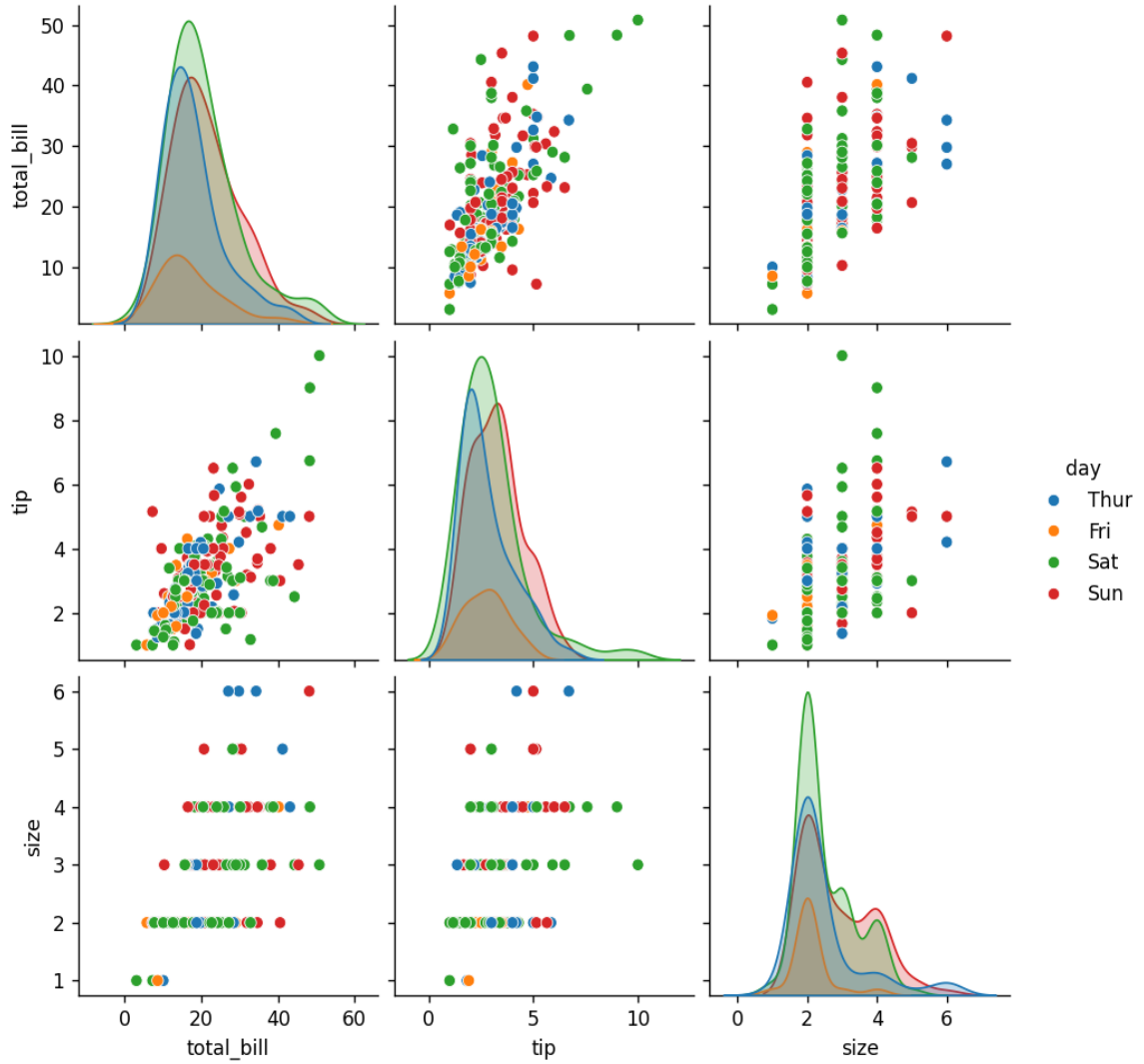

total_bill
tip
50
40
30
20
10
2
4
6
8
10


50
40
30
20
10
total_bill
2
4
6
8
10
tip


Count of Dinner Customers by Day
80
60
40
20
0
count
Thur
Fri
Sat
Sun
day


total_bill
tip
size
total_bill
tip
size


total_bill
tip
size
total_bill
tip
size
time
Lunch
Dinner




```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
time
Dinner    176
Lunch      68
Name: count, dtype: int64
```

Exp No:17 Experiment to understand Linear Regression for a given data set.

Description: Understand the Linear regression for the dataset given.

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pickle
data = {
    'YearsExperience': [1.1, 1.3, 1.5, 2.0, 2.2, 2.9, 3.0, 3.2, 3.2, 3.7,
                        3.9, 4.0, 4.0, 4.1, 4.5, 4.9, 5.1, 5.3, 5.9, 6.0,
                        6.8, 7.1, 7.9, 8.2, 8.7, 9.0, 9.5, 9.6, 10.3, 10.5],
    'Salary': [39343, 46205, 37731, 43525, 39891, 56642, 60150, 54445, 64445, 57189,
               63218, 55794, 56957, 57081, 61111, 67938, 66029, 83088, 81363, 93940,
               91738, 98273, 101302, 113812, 109431, 105582, 116969, 112635, 122391, 121872]
}
df = pd.DataFrame(data)
print("\nDataset Info")
print(df.info())
print("\nDataset Description")
print(df.describe())
features = df.iloc[:, :-1].values
label = df.iloc[:, -1].values
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=0)
model = LinearRegression()
model.fit(x_train, y_train)
print("\nModel Evaluation")
print("Training Score:", model.score(x_train, y_train))
print("Testing Score:", model.score(x_test, y_test))
print("Coefficient (Slope):", model.coef_)
print("Intercept:", model.intercept_)
with open('SalaryPred.model', 'wb') as file:
    pickle.dump(model, file)
with open('SalaryPred.model', 'rb') as file:
    loaded_model = pickle.load(file)
yr_of_exp = float(input("\nEnter Years of Experience: "))
yr_of_exp_np = np.array([[yr_of_exp]])
Salary = loaded_model.predict(yr_of_exp_np)
print("\nEstimated Salary for {:.1f} years of experience is: ₹{:.2f}".format(yr_of_exp, Salary[0]))
```

Output:

```
Dataset Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 612.0 bytes
None

Dataset Description
       YearsExperience         Salary
count        30.000000      30.000000
mean          5.313333   76003.000000
std           2.837888   27414.429785
min           1.100000   37731.000000
25%           3.200000   56720.750000
50%           4.700000   65237.000000
75%           7.700000  100544.750000
max          10.500000  122391.000000

Model Evaluation
Training Score: 0.9411949620562126
Testing Score: 0.988169515729126
Coefficient (Slope): [9312.57512673]
Intercept: 26780.09915062818

Enter Years of Experience: 44

Estimated Salary for 44.0 years of experience is: ₹436533.40
```

Exp No:18 Experiment to understand KNN algorithm for a given data set

Description: Understand the KNN algorithm for the dataset given.

Program:

```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
iris = load_iris(as_frame=True)
df = iris.frame
df.columns = ['sepal.length', 'sepal.width', 'petal.length', 'petal.width', 'variety']
print("\nInitial DataFrame Info")
df.info()
print("\nVariety Value Counts")
df['variety'] = df['variety'].map({i: name for i, name in enumerate(iris.target_names)})
print(df['variety'].value_counts())
print("\nHead of the Data")
print(df.head())
features = df.iloc[:, :-1].values
label = iris.target
x_train, x_test, y_train, y_test = train_test_split(
    features, label, test_size=0.2, random_state=2
)
model_KNN = KNeighborsClassifier(n_neighbors=5)
model_KNN.fit(x_train, y_train)
print("\nModel Scores")
print(model_KNN.score(x_train, y_train))
print(model_KNN.score(x_test, y_test))
print("\nConfusion Matrix")
y_pred_full = model_KNN.predict(features)
conf_matrix = confusion_matrix(label, y_pred_full)
print(conf_matrix)
print("\nClassification Report")
target_names = iris.target_names
class_report = classification_report(label, y_pred_full, target_names=target_names)
print(class_report)
```

Output:

```
Initial DataFrame Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

```
Variety Value Counts
variety
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64

Head of the Data
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa

Model Scores
0.9583333333333334
1.0

Confusion Matrix
[[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]

Classification Report
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        50
  versicolor       0.96      0.94      0.95        50
   virginica       0.94      0.96      0.95        50

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150
```

Exp No:19 Experiment to understand Logistic Regression for a given data set.

Description: Understand the Logistic Regression algorithm for the dataset given.

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
data = {
    'User ID': [
        15624510, 15810944, 15668575, 15603246, 15804002,
        15681234, 15748900, 15762345, 15699999, 15822222,
        15691863, 15706071, 15654296, 15755018, 15594041
    ],
    'Gender': [
        'Male', 'Male', 'Female', 'Female', 'Male',
        'Male', 'Female', 'Male', 'Female', 'Male',
        'Female', 'Male', 'Female', 'Male', 'Female'
    ],
    'Age': [
        19, 35, 26, 27, 19,
        42, 33, 48, 29, 40,
        46, 51, 50, 36, 49
    ],
    'EstimatedSalary': [
        19000, 20000, 43000, 57000, 76000,
        61000, 82000, 79000, 55000, 63000,
        41000, 23000, 20000, 33000, 36000
    ],
    'Purchased': [
        0, 0, 0, 0, 0,
        1, 0, 1, 0, 1,
        1, 1, 1, 0, 1
    ]
}
```

```
33
34  df = pd.DataFrame(data)
35  print(df)
36  print("\n")
37  print(df.head())
38  print("\n")
39  features = df.iloc[:, [2, 3]].values
40  label = df.iloc[:, 4].values
41  print(features)
42  print("\n")
43  print(label)
44  print("\n")
45  for i in range(1, 16):
46      x_train, x_test, y_train, y_test = train_test_split(
47          features, label, test_size=0.2, random_state=i
48      )
49      model = LogisticRegression()
50      model.fit(x_train, y_train)
51      train_score = model.score(x_train, y_train)
52      test_score = model.score(x_test, y_test)
53      if i < 7 or test_score > train_score:
54          print("Test {:.4f} Train {:.4f} Random State {}".format(test_score, train_score, i))
55  x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=0)
56  model = LogisticRegression()
57  model.fit(x_train, y_train)
58  print("\nTraining Accuracy:", model.score(x_train, y_train))
59  print("Testing Accuracy:", model.score(x_test, y_test))
60  y_pred = model.predict(features)
61  print("\nClassification Report:\n", classification_report(label, y_pred, zero_division=0))
```

Output:

```
      User ID  Gender  Age  EstimatedSalary  Purchased
0    15624510    Male   19            19000          0
1    15810944    Male   35            20000          0
2    15668575  Female   26            43000          0
3    15603246  Female   27            57000          0
4    15804002    Male   19            76000          0
5    15681234    Male   42            61000          1
6    15748900  Female   33            82000          0
7    15762345    Male   48            79000          1
8    15699999  Female   29            55000          0
9    15822222    Male   40            63000          1
10   15691863  Female   46            41000          1
11   15706071    Male   51            23000          1
12   15654296  Female   50            20000          1
13   15755018    Male   36            33000          0
14   15594041  Female   49            36000          1


     User ID  Gender  Age  EstimatedSalary  Purchased
0   15624510    Male   19            19000          0
1   15810944    Male   35            20000          0
2   15668575  Female   26            43000          0
3   15603246  Female   27            57000          0
4   15804002    Male   19            76000          0
```

```
[[   19 19000]
 [   35 20000]
 [   26 43000]
 [   27 57000]
 [   19 76000]
 [   42 61000]
 [   33 82000]
 [   48 79000]
 [   29 55000]
 [   40 63000]
 [   46 41000]
 [   51 23000]
 [   50 20000]
 [   36 33000]
 [   49 36000]]


[0 0 0 0 0 1 0 1 0 1 1 1 1 0 1]


Test 0.6667 Train 1.0000 Random State 1
Test 1.0000 Train 1.0000 Random State 2
Test 1.0000 Train 1.0000 Random State 3
Test 0.6667 Train 1.0000 Random State 4
Test 1.0000 Train 1.0000 Random State 5
Test 0.6667 Train 1.0000 Random State 6

Training Accuracy: 1.0
Testing Accuracy: 0.6666666666666666

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.88      0.93         8
           1       0.88      1.00      0.93         7

    accuracy                           0.93        15
   macro avg       0.94      0.94      0.93        15
weighted avg       0.94      0.93      0.93        15
```

Exp No: 20 Experiment to understand K-means clustering algorithm for a given data set.

Description: Understand the K-means clustering algorithm for the dataset given.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
data = {
    'CustomerID': np.arange(1, 31),
    'Gender': ['Male', 'Female'] * 15,
    'Age': [19, 21, 20, 23, 31, 22, 35, 36, 25, 23,
            45, 46, 50, 51, 52, 48, 29, 31, 32, 33,
            40, 42, 51, 45, 38, 37, 30, 29, 28, 27],
    'Annual Income (k$)': [15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
                           25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
                           35, 36, 37, 38, 39, 40, 41, 42, 43, 44],
    'Spending Score (1-100)': [39, 81, 6, 77, 40, 76, 30, 40, 80, 60,
                               50, 40, 20, 30, 50, 70, 80, 60, 40, 20,
                               30, 10, 80, 90, 100, 60, 40, 20, 30, 10]
}
df = pd.DataFrame(data)
print(df)
print(df.info())
print(df.head())
sns.pairplot(df.iloc[:, 2:5])
plt.show()
features = df.iloc[:, [3, 4]].values
wcss = []
```

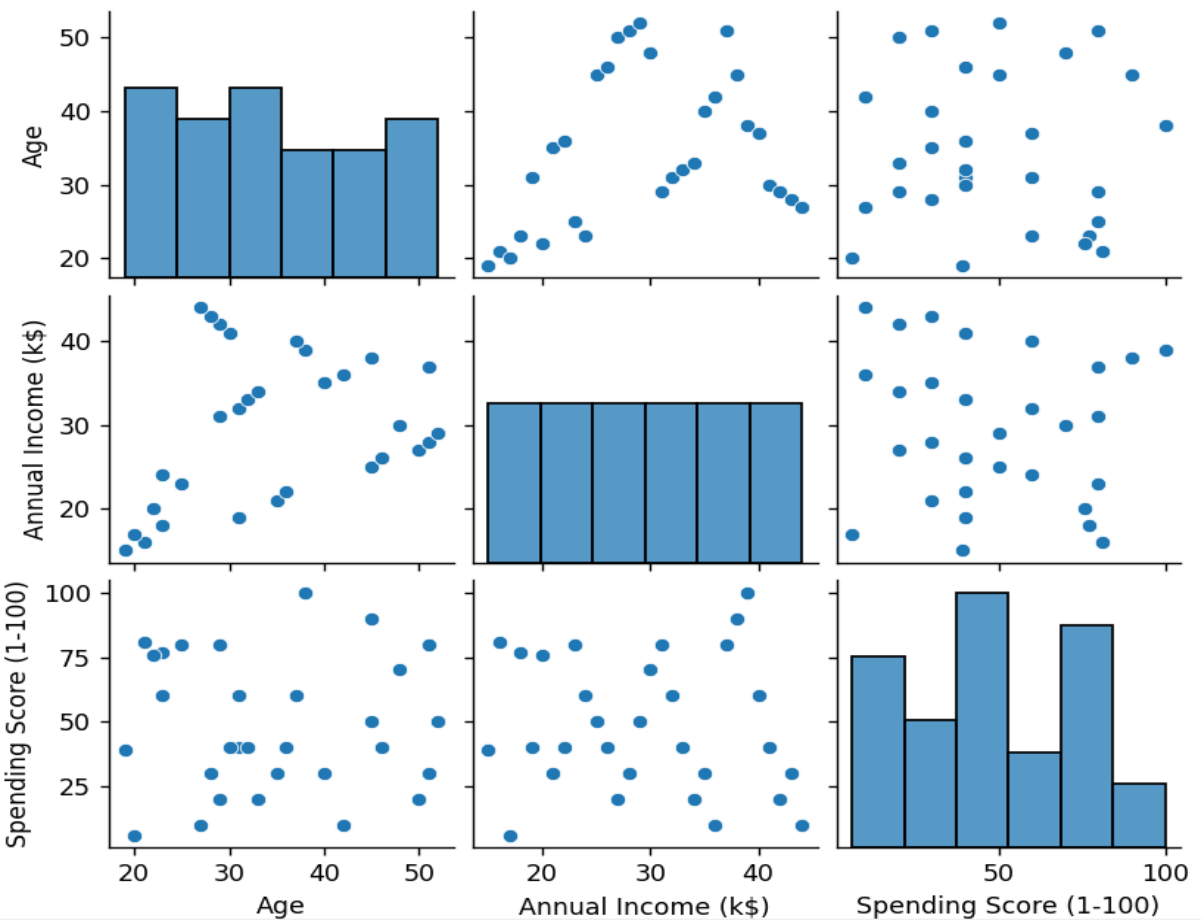
```
for i in range(1, 11):
    model = KMeans(n_clusters=i)
    model.fit(features)
    wcss.append(model.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
optimal_clusters = 5
model = KMeans(n_clusters=optimal_clusters)
df['label'] = model.fit_predict(features)
sns.set_style("whitegrid")
sns.scatterplot(data=df, x='Annual Income (k$)', y='Spending Score (1-100)',
                hue='label', palette='viridis')
plt.title('Clusters of Customers')
plt.legend(title='Cluster')
plt.show()
```

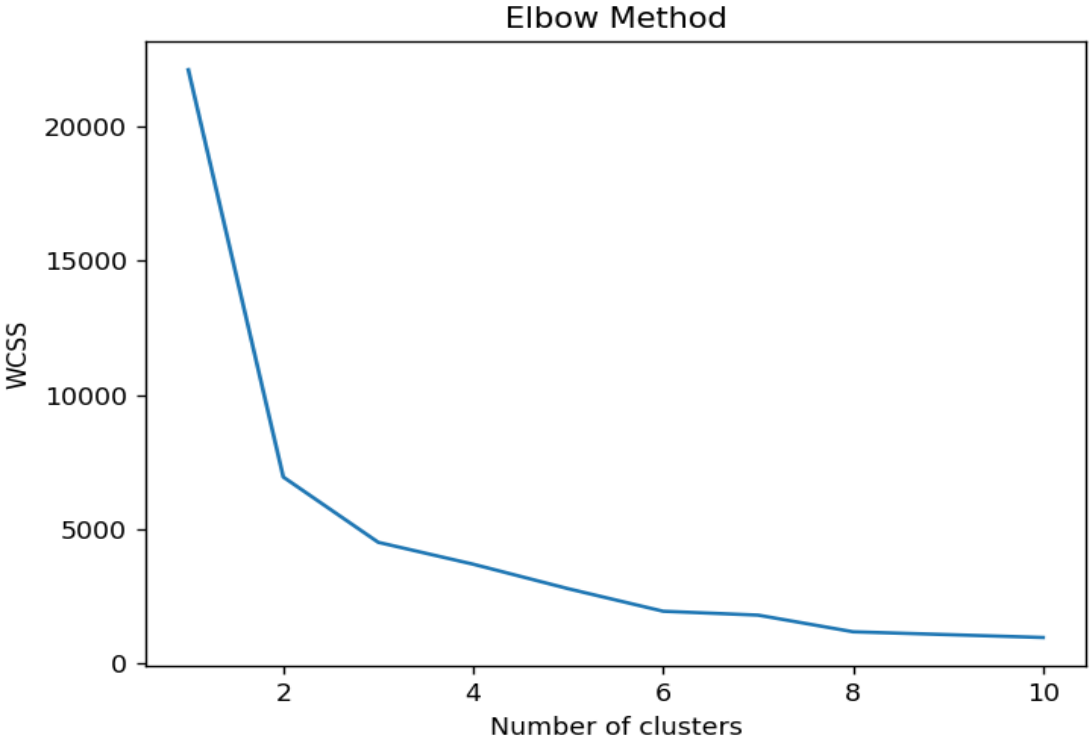
Output:

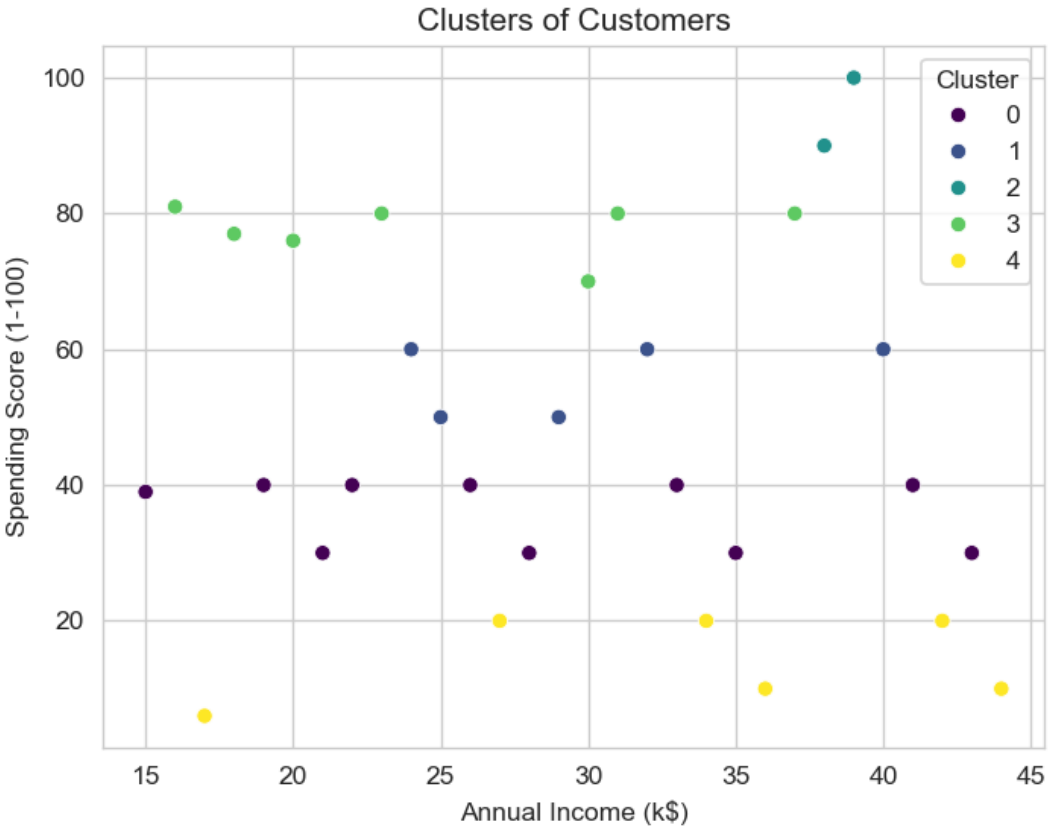
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Female	21	16	81
2	3	Male	20	17	6
3	4	Female	23	18	77
4	5	Male	31	19	40
5	6	Female	22	20	76
6	7	Male	35	21	30
7	8	Female	36	22	40
8	9	Male	25	23	80
9	10	Female	23	24	60
10	11	Male	45	25	50
11	12	Female	46	26	40
12	13	Male	50	27	20
13	14	Female	51	28	30
14	15	Male	52	29	50
15	16	Female	48	30	70
16	17	Male	29	31	80
17	18	Female	31	32	60
18	19	Male	32	33	40
19	20	Female	33	34	20
20	21	Male	40	35	30
21	22	Female	42	36	10
22	23	Male	51	37	80
23	24	Female	45	38	90
24	25	Male	38	39	100
25	26	Female	37	40	60
26	27	Male	30	41	40
27	28	Female	29	42	20
28	29	Male	28	43	30
29	30	Female	27	44	10

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              30 non-null     int64
 1   Gender                  30 non-null     object
 2   Age                     30 non-null     int64
 3   Annual Income (k$)      30 non-null     int64
 4   Spending Score (1-100)  30 non-null     int64
dtypes: int64(4), object(1)
memory usage: 1.3+ KB
None
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Female	21	16	81
2	3	Male	20	17	6
3	4	Female	23	18	77
4	5	Male	31	19	40


Age
Annual Income (k$)
Spending Score (1-100)
20
30
40
50
25
75
100


Elbow Method
WCSS
Number of clusters
0
5000
10000
15000
20000
2
4
6
8
10


Clusters of Customers
Spending Score (1-100)
Annual Income (k$)
Cluster
0
1
2
3
4
100
80
60
40
20
15
20
25
30
35
40
45