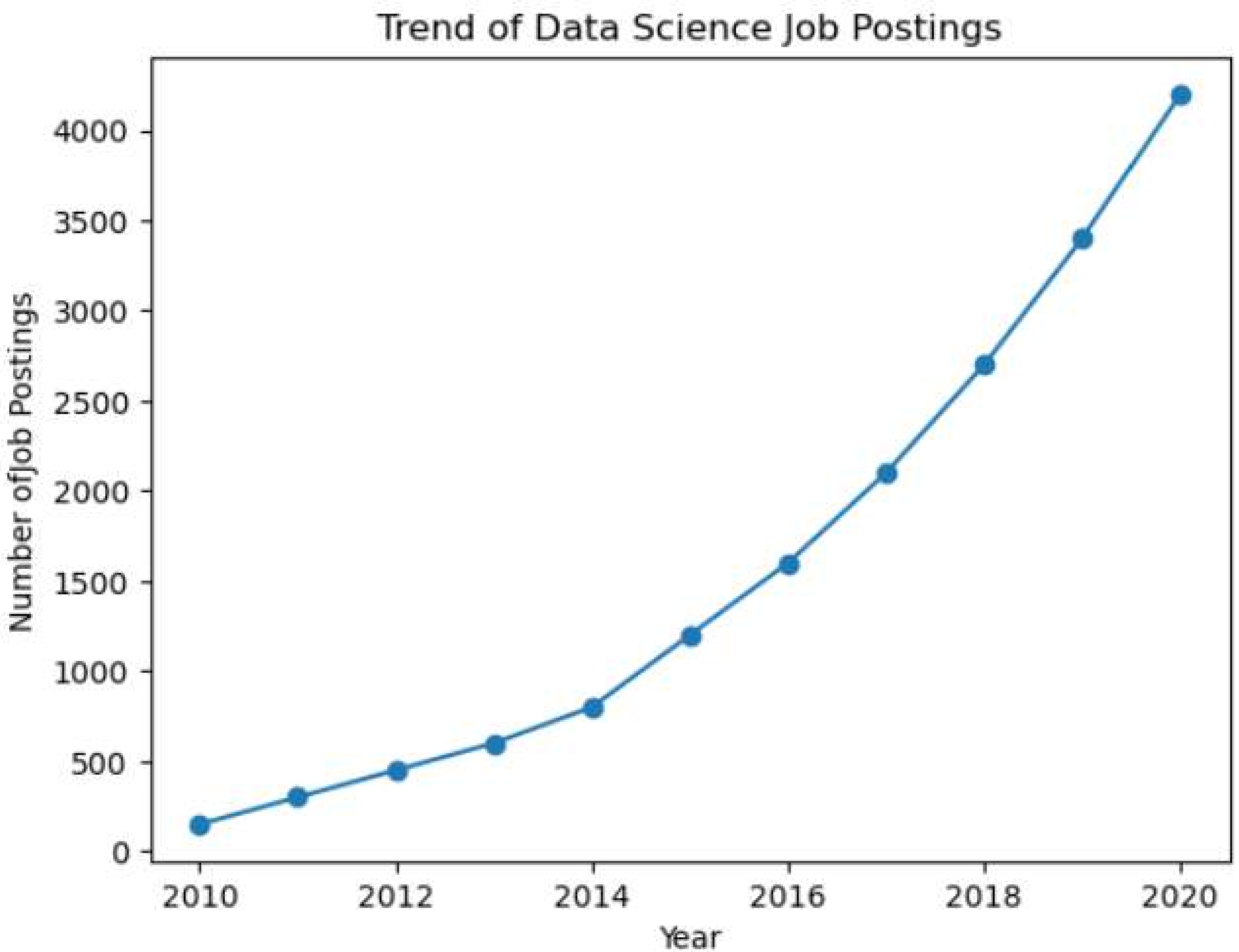
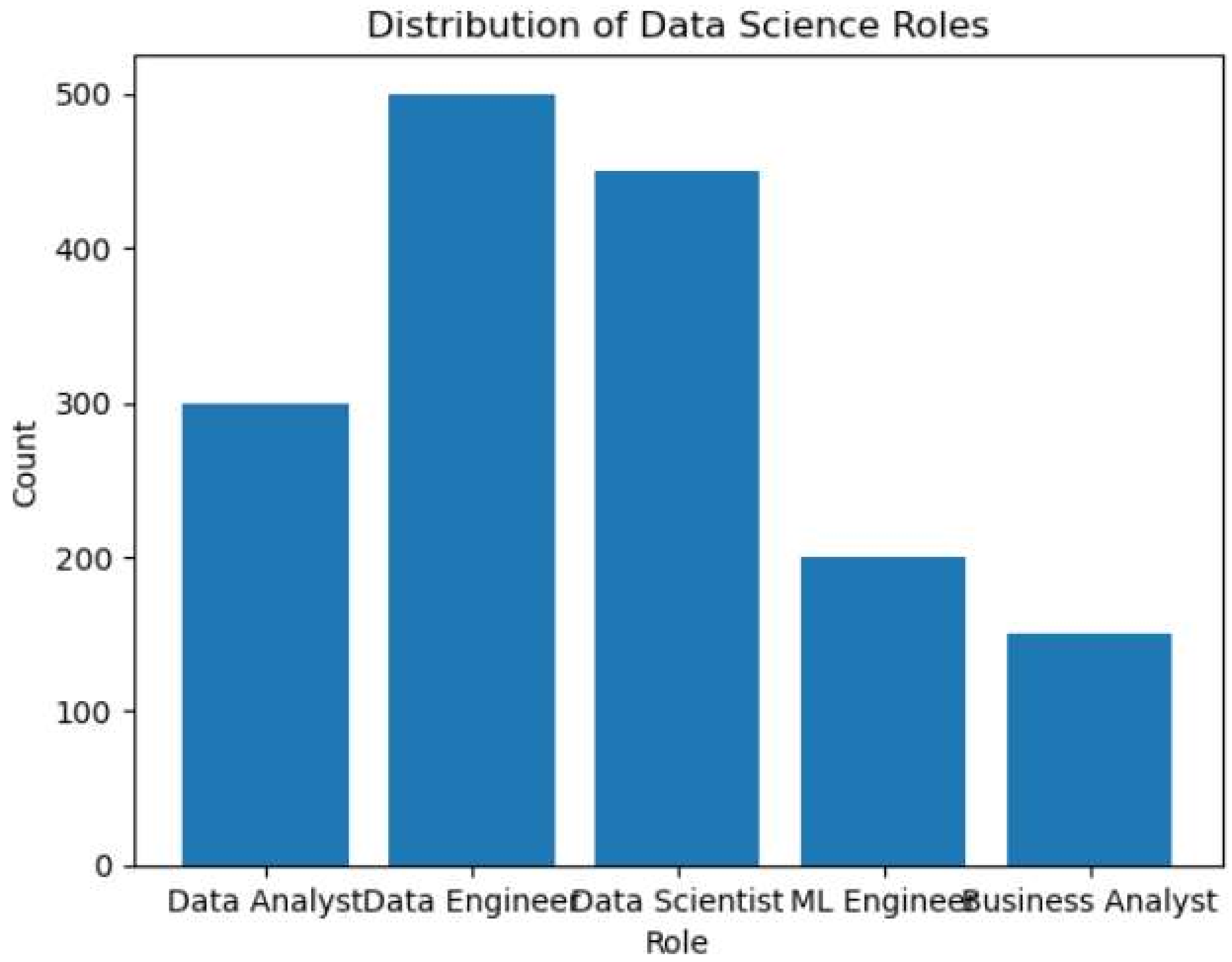
[1]:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'Year': list(range(2010, 2021)),
'Job Postings': [150, 300, 450, 600, 800, 1200, 1600, 2100, 2700, 3400, 4200]}

df = pd.DataFrame(data)
plt.plot(df['Year'], df['Job Postings'], marker='o')
plt.title('Trend of Data Science Job Postings')
plt.xlabel('Year')
plt.ylabel('Number ofJob Postings')
plt.show()
```



[2]:

```python
roles = ['Data Analyst', 'Data Engineer', 'Data Scientist', 'ML Engineer',
'Business Analyst']
counts = [300, 500, 450, 200, 150]
plt.bar(roles, counts)
plt.title('Distribution of Data Science Roles')
plt.xlabel('Role')
plt.ylabel('Count')
plt.show()
```



[4]:

```python
structured_data = pd.DataFrame({'ID': [1, 2, 3],'Name': ['Alice', 'Bob', 'Charlie'],'Age': [25, 30, 35]})
print("Structured Data:\n", structured_data)
unstructured_data = "This is an example of unstructured data. It can be a piece of text, an image, or a video file."
print("\nUnstructured Data:\n", unstructured_data)

semi_structured_data = {'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
print("\nSemi-structured Data:\n", semi_structured_data)
```

```
Structured Data:
    ID     Name  Age
0   1    Alice   25
1   2      Bob   30
2   3  Charlie   35

Unstructured Data:
 This is an example of unstructured data. It can be a piece of text, an image, or a video file.

Semi-structured Data:
 {'ID': 1, 'Name': 'Alice', 'Attributes': {'Height': 165, 'Weight': 68}}
```

[5]:

```python
from cryptography.fernet import Fernet
key = Fernet.generate_key()
f = Fernet(key)
token = f.encrypt(b"Rajalakshmi Engineering College")
token
b'...'
f.decrypt(token)
b'Rajalakshmi Engineering College'
key = Fernet.generate_key()
cipher_suite = Fernet(key)
plain_text = b"Rajalakshmi Engineering College."
cipher_text = cipher_suite.encrypt(plain_text)
decrypted_text = cipher_suite.decrypt(cipher_text)
print("Original Data:", plain_text)
print("Encrypted Data:", cipher_text)
print("Decrypted Data:", decrypted_text)
```

```
Original Data: b'Rajalakshmi Engineering College.'
Encrypted Data: b'gAAAAABomxmEqTZRrJQz2mFmMGLRPbXNgZ_nJnqj3jGOmha5Qqs28412cuLazlCTnsTuo29VLTSe8SIikaZuzd9pOjGqA7skcsXoAisnlpFQ2yCGWUY2EAVsKOCnkLASBdD9
1EZkMXYp'
Decrypted Data: b'Rajalakshmi Engineering College.'
```

[2]:

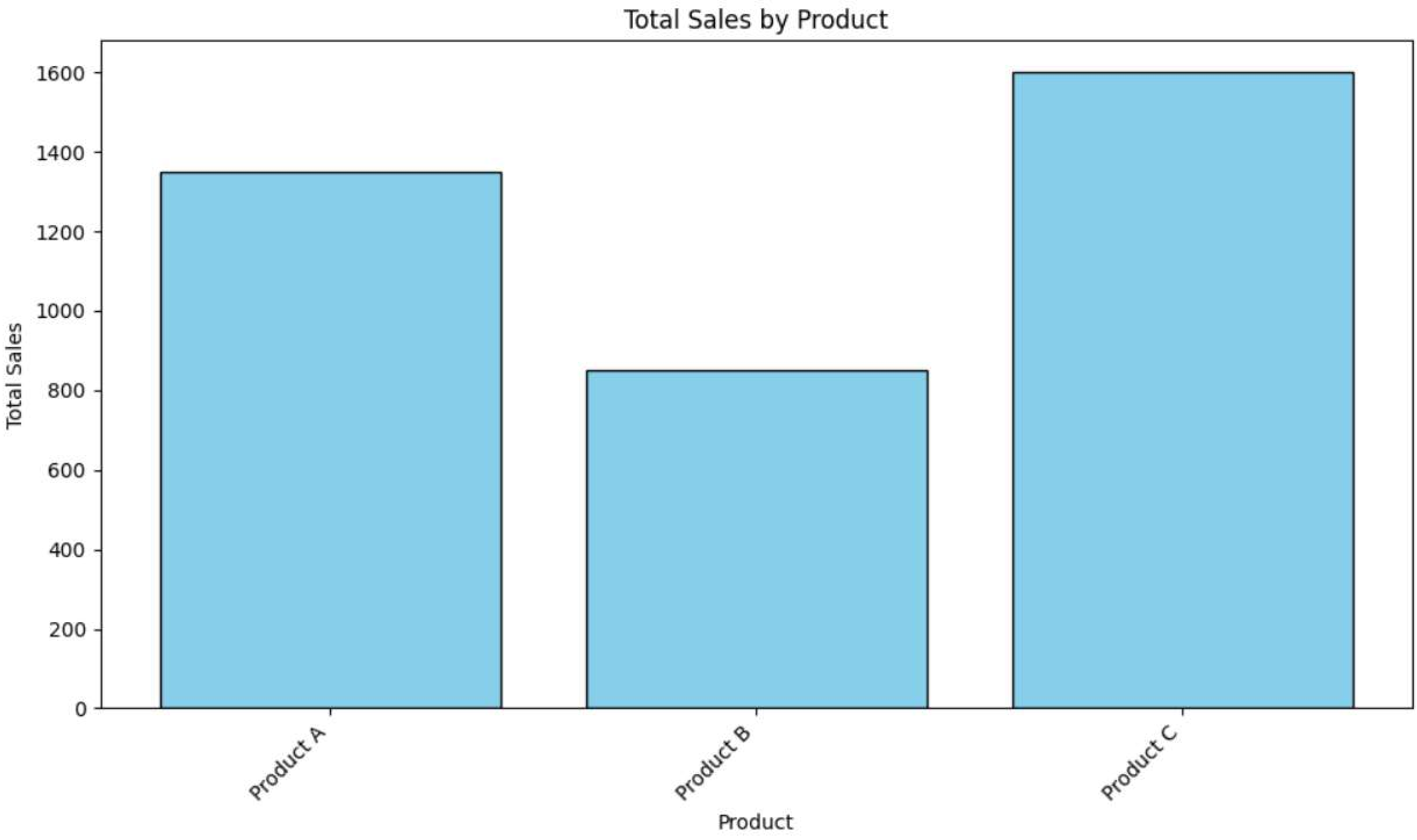
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

# === File path ===
file_path = r"C:\Users\A R KRISHNA\Downloads\sales_data.xlsx"

# === Check if file exists ===
if not os.path.exists(file_path):
    raise FileNotFoundError(f"File not found: {file_path}")

# === Load Excel file ===
try:
    df = pd.read_excel(file_path)
    print("File loaded successfully!")
except Exception as e:
    raise RuntimeError(f"Error reading Excel file: {e}")

# === Display basic info ===
print("\nFirst few rows:\n", df.head())
print("\nAvailable Columns:\n", df.columns.tolist())
print("\nMissing Values:\n", df.isnull().sum())

# === Validate required columns ===
required_cols = {'Sales', 'Quantity', 'Product', 'Region', 'Date'}
missing_cols = required_cols - set(df.columns)
if missing_cols:
    raise ValueError(f"Missing required columns: {missing_cols}")

# === Convert data types safely ===
df['Sales'] = pd.to_numeric(df['Sales'], errors='coerce')
df['Quantity'] = pd.to_numeric(df['Quantity'], errors='coerce')
df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# === Handle missing values ===
if df['Sales'].isnull().all():
    raise ValueError("All Sales values are missing or non-numeric.")
```

```python
df['Sales'] = df['Sales'].fillna(df['Sales'].mean())
df.dropna(subset=['Product', 'Quantity', 'Region', 'Date'], inplace=True)

# === Ensure data is not empty ===
if df.empty:
    raise ValueError("No valid data available after cleaning.")

# === Descriptive statistics ===
print("\nDescriptive Statistics:\n", df.describe())

# === Product Summary ===
product_summary = df.groupby('Product', as_index=False)[['Sales', 'Quantity']].sum()
print("\nProduct Summary:\n", product_summary)

# === Bar Chart: Total Sales by Product ===
if not product_summary.empty:
    plt.figure(figsize=(10, 6))
    plt.bar(product_summary['Product'], product_summary['Sales'], color='skyblue', edgecolor='black')
    plt.xlabel('Product')
    plt.ylabel('Total Sales')
    plt.title('Total Sales by Product')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
else:
    print("Skipping bar chart — product summary is empty.")

# === Line Chart: Sales Over Time ===
sales_over_time = df.groupby('Date', as_index=False)['Sales'].sum().sort_values('Date')

if not sales_over_time.empty:
    plt.figure(figsize=(10, 6))
    plt.plot(sales_over_time['Date'], sales_over_time['Sales'], marker='o', linestyle='-', color='green')
    plt.xlabel('Date')
    plt.ylabel('Total Sales')
    plt.title('Sales Over Time')
    plt.tight_layout()
    plt.show()
else:
    print("Skipping time series plot — no valid date data found.")
```

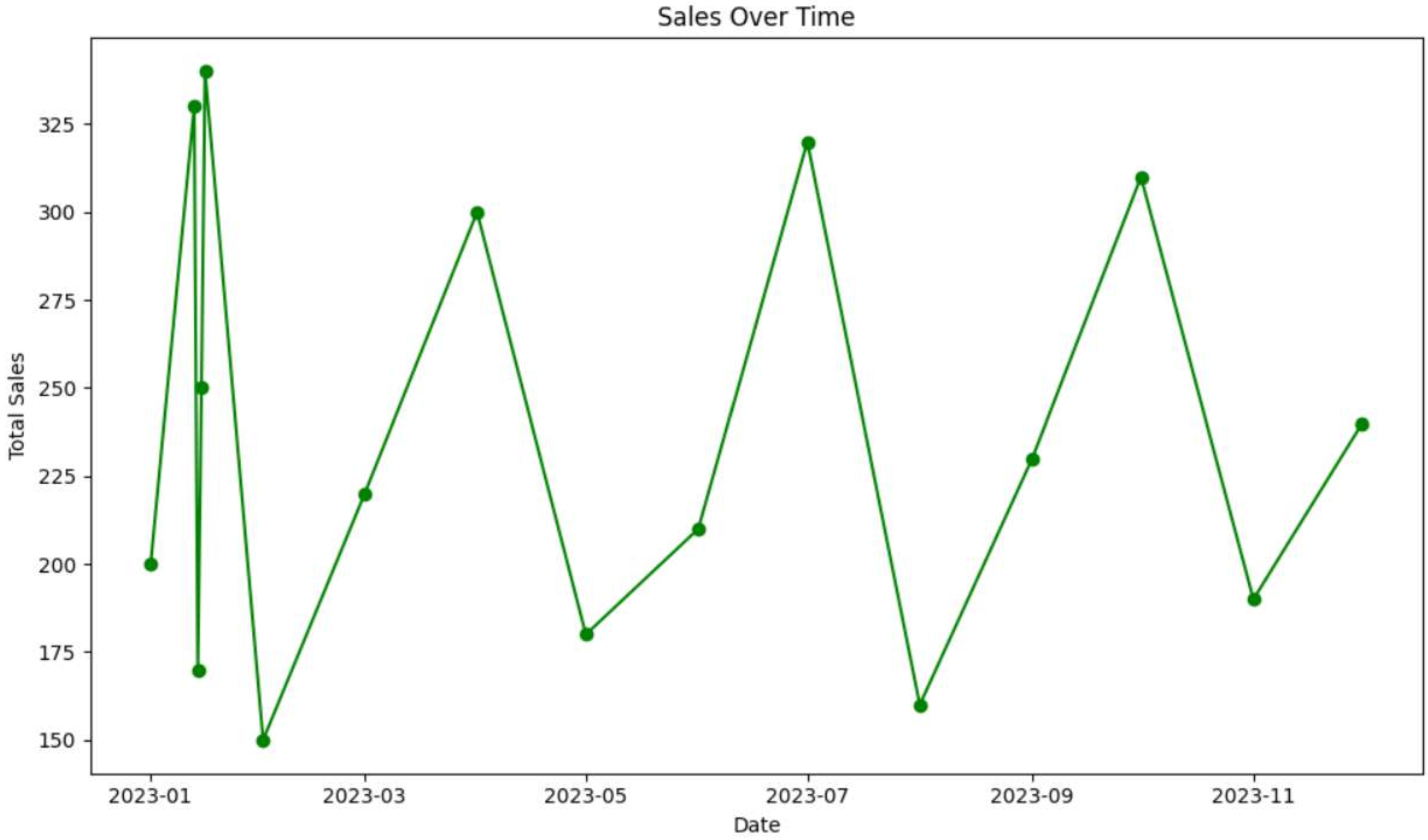
```python
# === Pivot Table (Sales by Region and Product) ===
pivot_table = df.pivot_table(values='Sales', index='Region', columns='Product', aggfunc='sum', fill_value=0)
print("\nPivot Table (Sales by Region and Product):\n", pivot_table)

if not pivot_table.empty:
    plt.figure(figsize=(8, 6))
    sns.heatmap(pivot_table, annot=True, fmt=".0f", cmap='Blues')
    plt.title('Sales by Region and Product')
    plt.tight_layout()
    plt.show()

# === Correlation Matrix ===
corr_cols = ['Sales', 'Quantity']
if all(col in df.columns for col in corr_cols):
    correlation_matrix = df[corr_cols].corr()
    print("\n🔗 Correlation Matrix:\n", correlation_matrix)

    plt.figure(figsize=(6, 5))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1)
    plt.title('Correlation Matrix')
    plt.tight_layout()
    plt.show()
else:
    print("Correlation plot skipped – missing numeric columns.")
```

```
File loaded successfully!

First few rows:
                   Date    Product  Sales  Quantity Region
0  2023-01-01 00:00:00  Product A    200         4  North
1  2023-02-01 00:00:00  Product B    150         3  South
2  2023-03-01 00:00:00  Product A    220         5  North
3  2023-04-01 00:00:00  Product C    300         6   East
4  2023-05-01 00:00:00  Product B    180         4   West
```

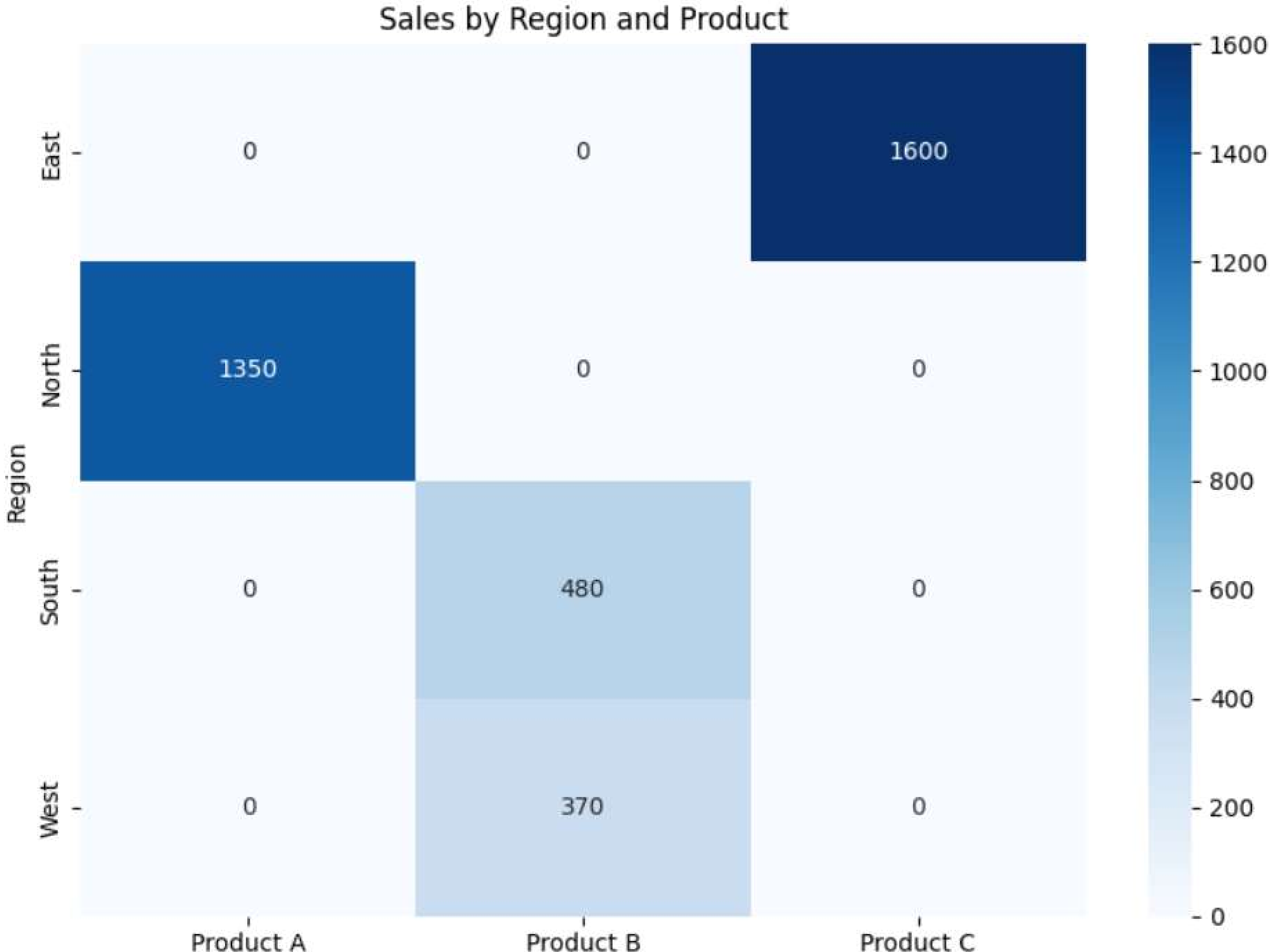
```
Available Columns:
 ['Date', 'Product', 'Sales', 'Quantity', 'Region']

Missing Values:
 Date        0
Product     0
Sales       0
Quantity    0
Region      0
dtype: int64

Descriptive Statistics:
                       Date       Sales   Quantity
count                   16   16.000000  16.000000
mean   2023-05-09 06:00:00  237.500000   5.375000
min    2023-01-01 00:00:00  150.000000   3.000000
25%    2023-01-15 18:00:00  187.500000   4.000000
50%    2023-04-16 00:00:00  225.000000   5.500000
75%    2023-08-08 18:00:00  302.500000   7.000000
max    2023-12-01 00:00:00  340.000000   8.000000
std                    NaN   64.031242   1.746425

Product Summary:
      Product  Sales  Quantity
0  Product A   1350        33
1  Product B    850        17
2  Product C   1600        36
```



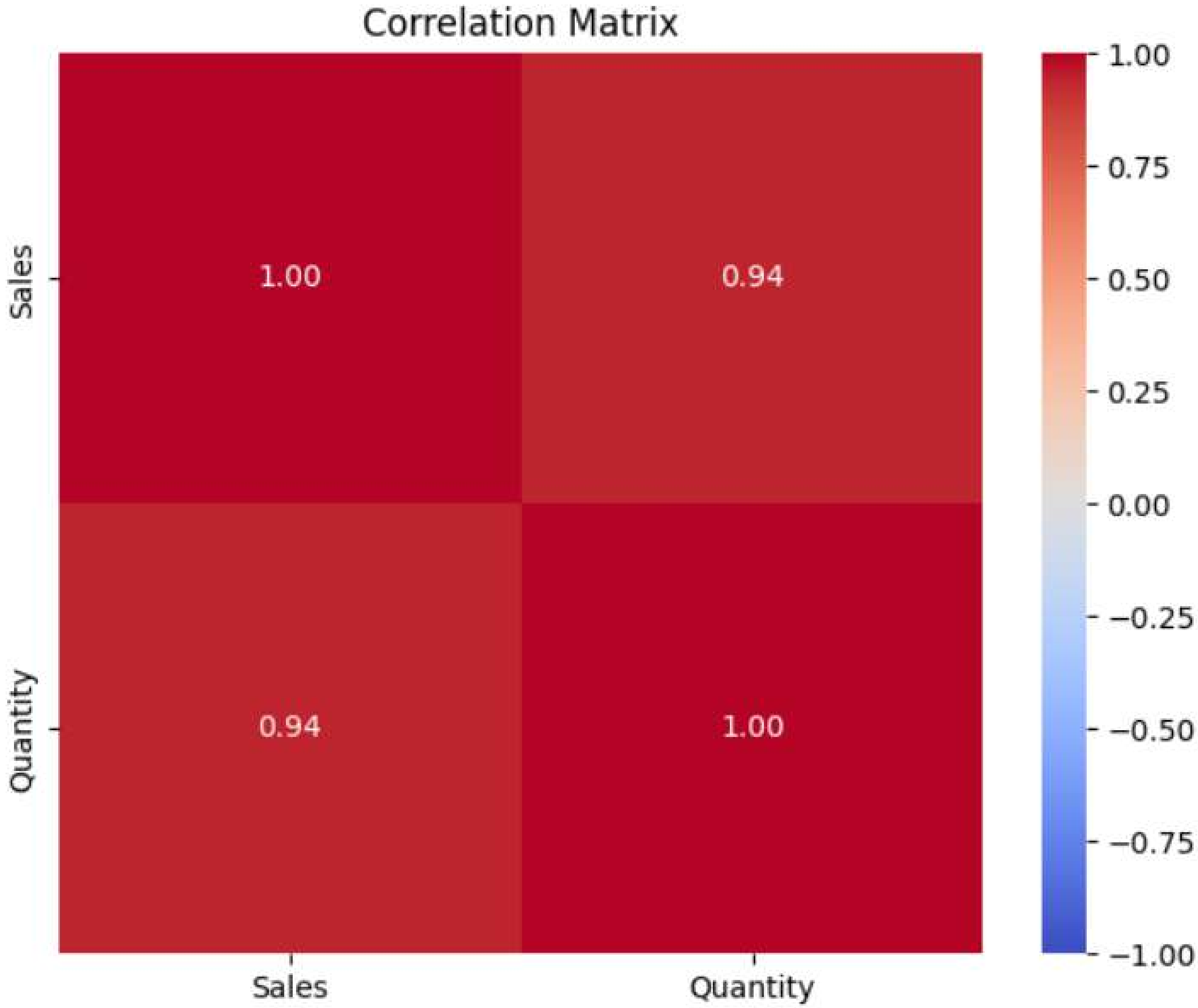
Total Sales by Product
Total Sales
0
200
400
600
800
1000
1200
1400
1600
Product A
Product B
Product C
Product


Sales Over Time
Total Sales
325
300
275
250
225
200
175
150
2023-01
2023-03
2023-05
2023-07
2023-09
2023-11
Date

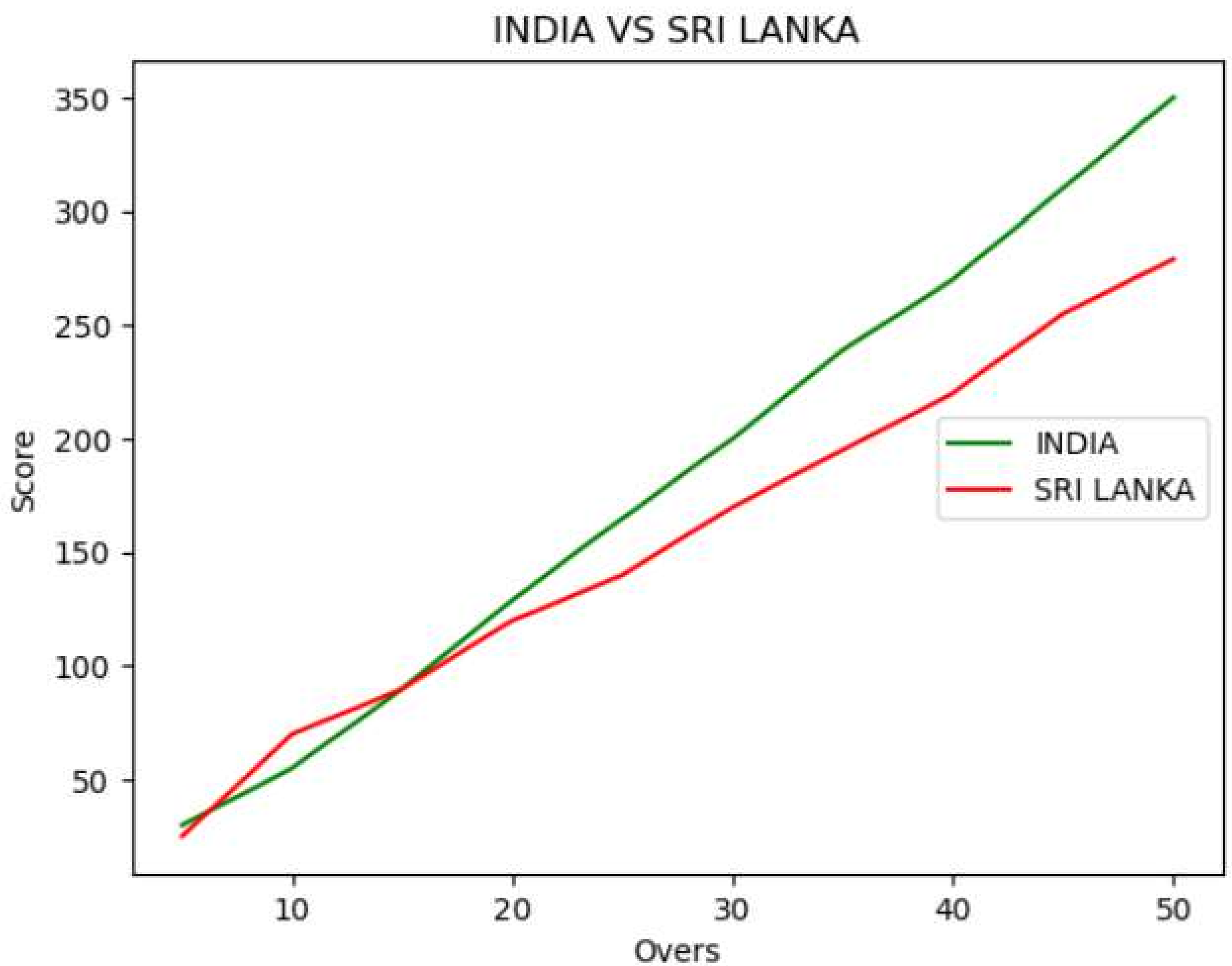
```
Product  Product A  Product B  Product C
Region
East             0          0       1600
North         1350          0          0
South            0        480          0
West             0        370          0
```



```
🔗 Correlation Matrix:
             Sales  Quantity
Sales     1.000000  0.944922
Quantity  0.944922  1.000000
```

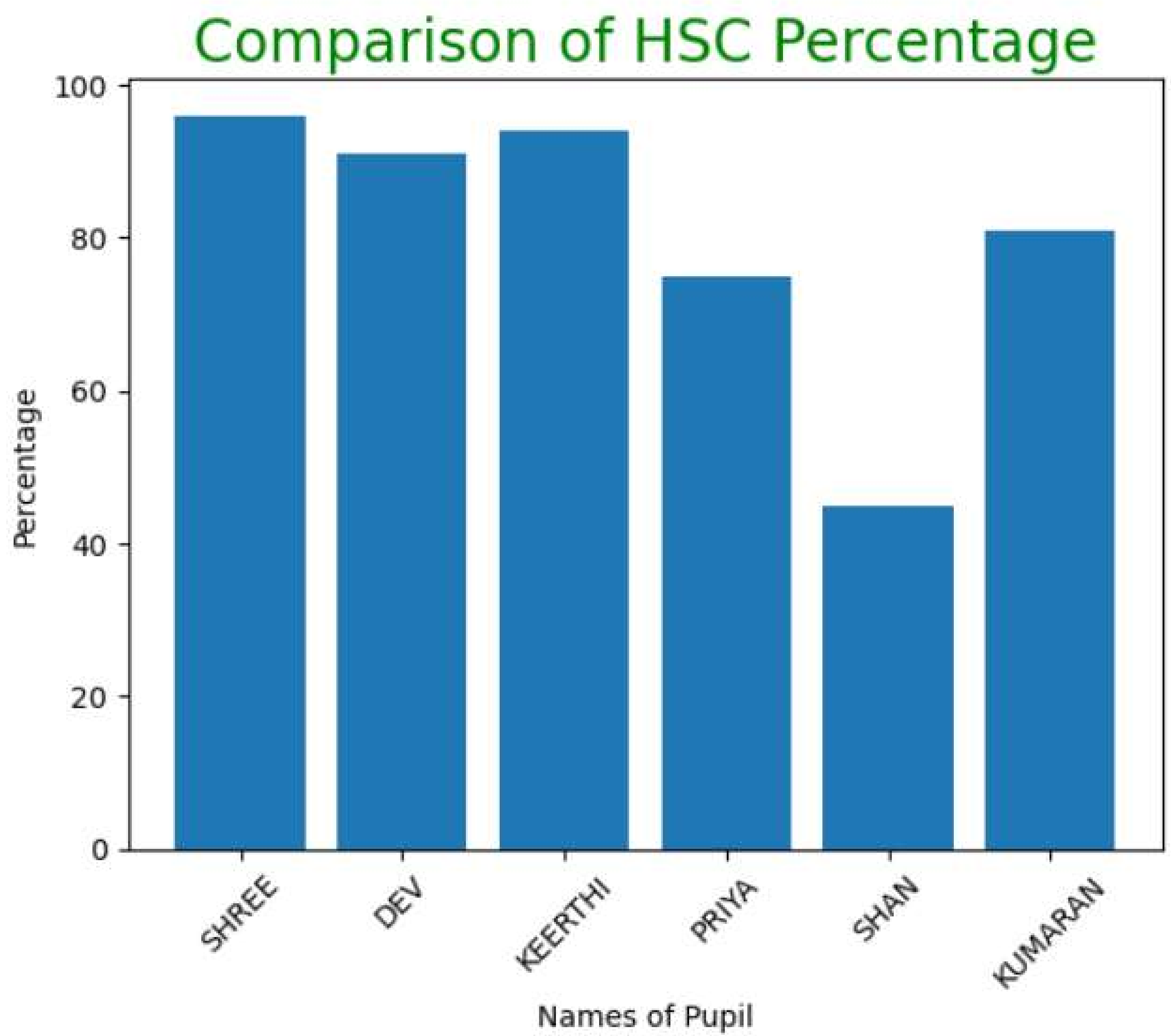


[1]:
```
import matplotlib.pyplot as cricket

Overs = list(range(5, 51, 5))
Indian_Score = [30, 55, 90, 129, 165, 200, 239, 270, 310, 350]
Srilankan_Score = [25, 70, 90, 120, 140, 170, 195, 220, 255, 279]
cricket.plot(Overs, Indian_Score, color="green", label="INDIA")
cricket.plot(Overs, Srilankan_Score, color="red", label="SRI LANKA")
cricket.title("INDIA VS SRI LANKA")
cricket.xlabel("Overs")
cricket.ylabel("Score")
cricket.legend(loc="center right")
cricket.show()
```

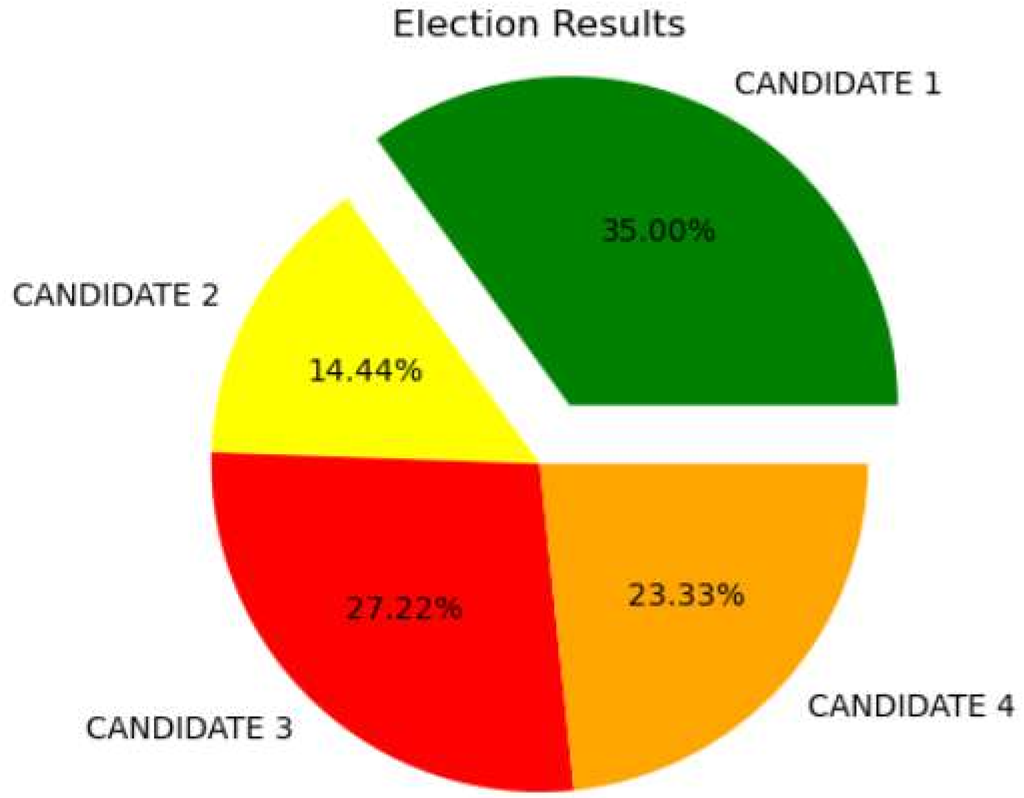


[1]:

```
import matplotlib.pyplot as hscmark
import numpy as np
Names = ['SHREE', 'DEV', 'KEERTHI', 'PRIYA', 'SHAN', 'KUMARAN']
xaxis = np.arange(len(Names))
Percentage_hsc = [96, 91, 94, 75, 45, 81]
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()
```



[2]:
```python
import matplotlib.pyplot as election
# Election data
labels = ['CANDIDATE 1', 'CANDIDATE 2', 'CANDIDATE 3', 'CANDIDATE 4']
Votes = [315, 130, 245, 210]
colors = ['green', 'yellow', 'red', 'orange']
explode = (0.2, 0, 0, 0)
# Plotting the pie chart
election.pie(Votes, labels=labels, colors=colors, explode=explode, autopct='%0.2f%%')
election.title('Election Results')
election.show()
```

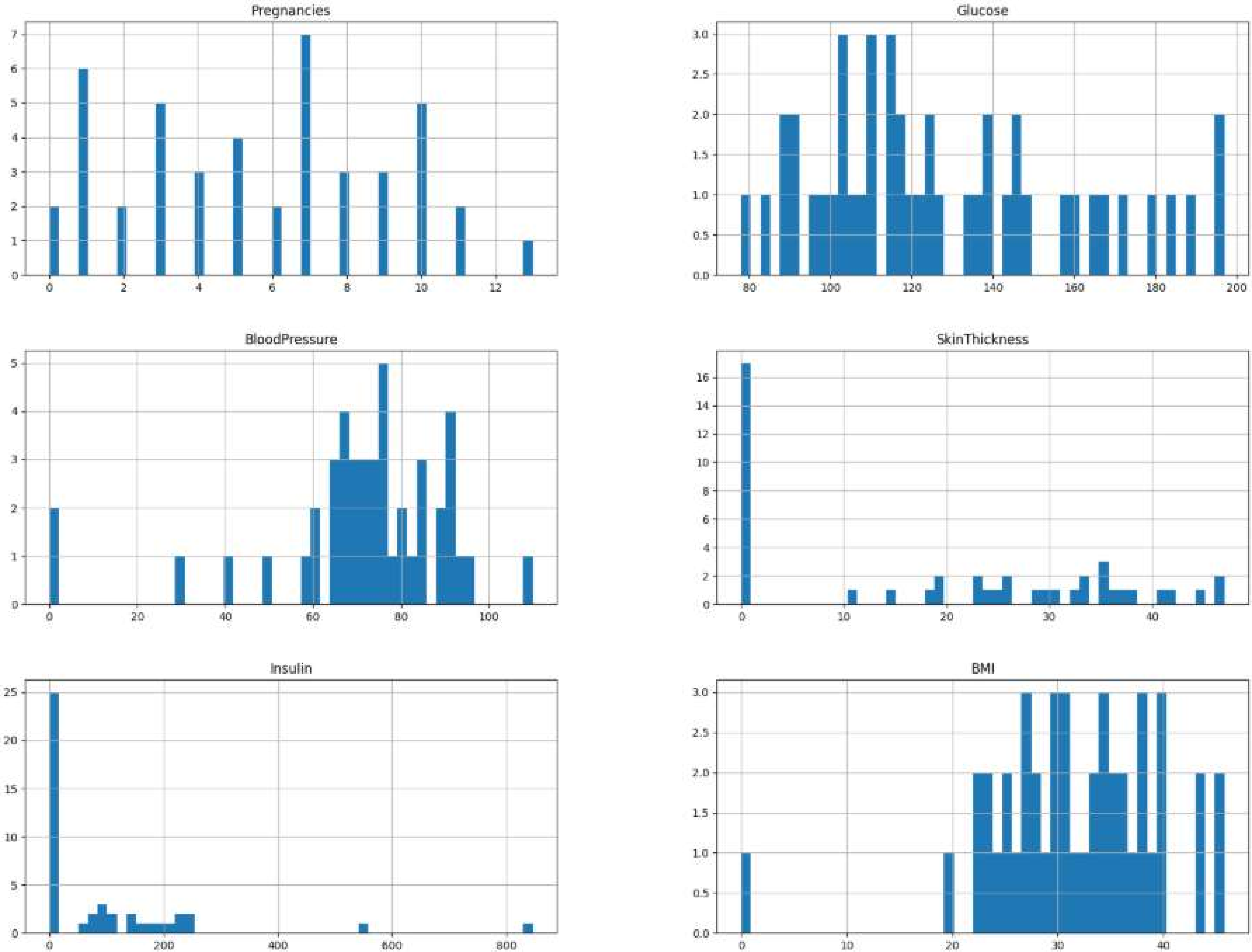


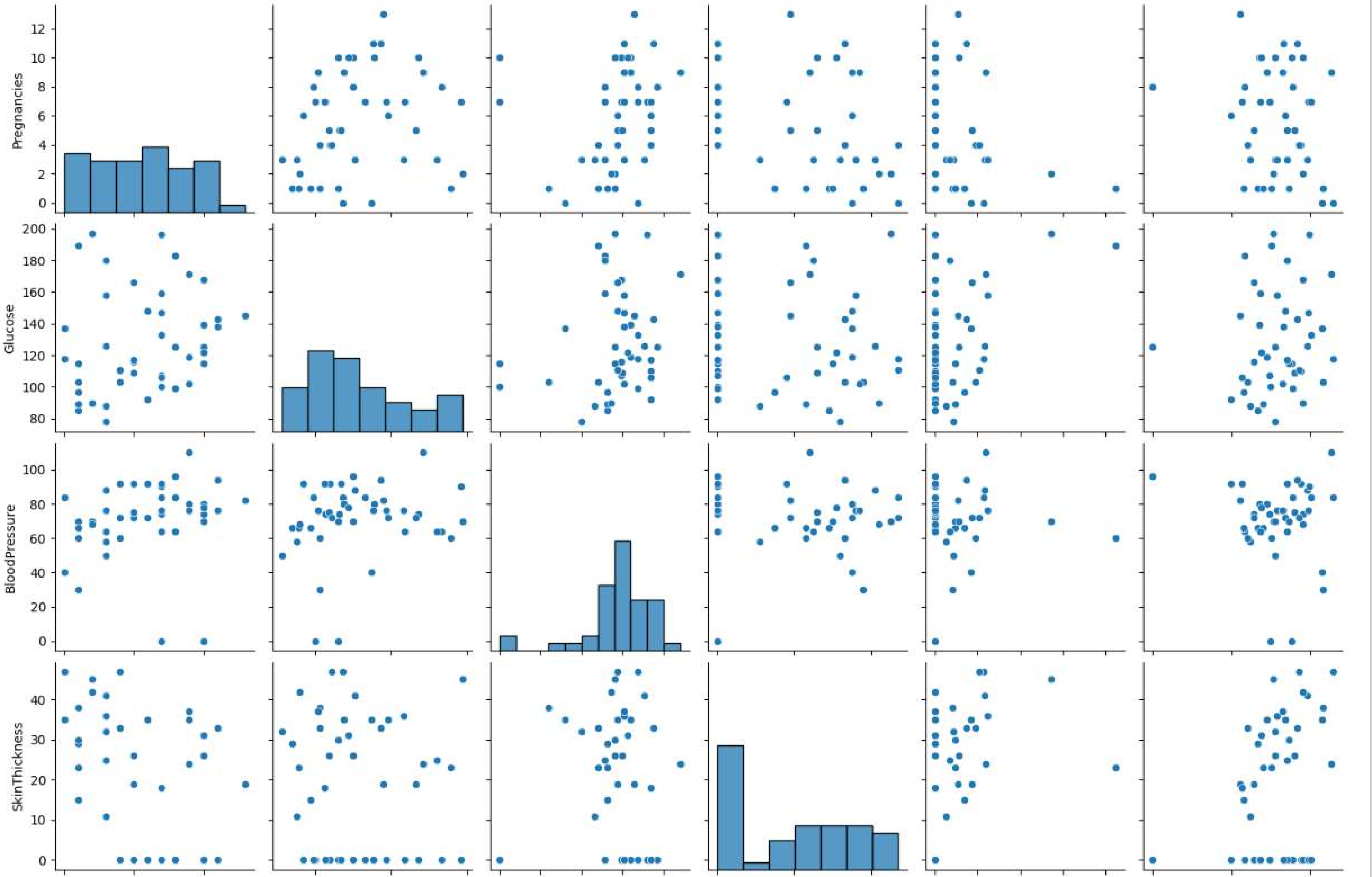
[7]:
```python
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg
from collections import Counter

nltk.download('gutenberg')
nltk.download('punkt')
nltk.download('punkt_tab')
sample = gutenberg.raw("austen-emma.txt")
tokens = word_tokenize(sample)
wlist = tokens[:50]
wordfreq = Counter(wlist)
print("Pairs\n" + str(list(wordfreq.items())))
```

```
[nltk_data] Downloading package gutenberg to C:\Users\A R
[nltk_data]     KRISHNA\AppData\Roaming\nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package punkt to C:\Users\A R
[nltk_data]     KRISHNA\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to C:\Users\A R
[nltk_data]     KRISHNA\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt_tab.zip.
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)]
```

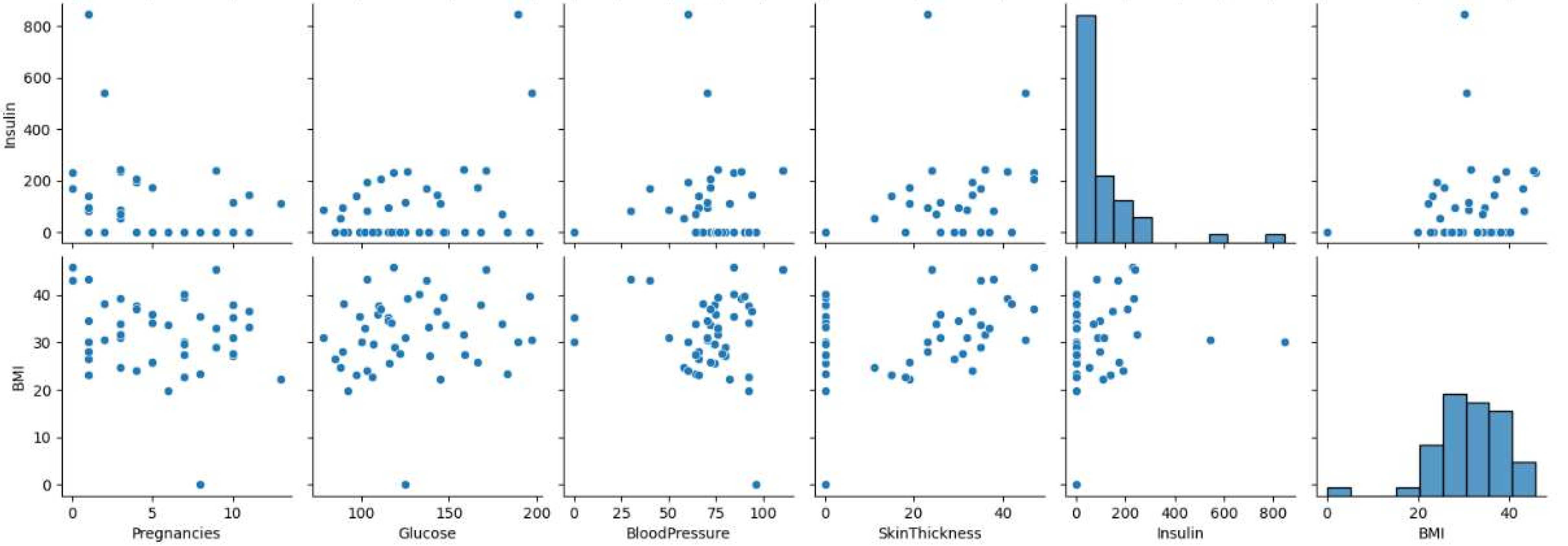
[7]:
```python
import pdfplumber
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

with pdfplumber.open(r"C:\Users\A R KRISHNA\Downloads\diabetes.pdf") as pdf:
    first_page = pdf.pages[0]
    table = first_page.extract_table()

df = pd.DataFrame(table[1:], columns=table[0])
df = df.apply(pd.to_numeric, errors='ignore')

print(df.head())
print(df.info())
print(df.describe())

df.hist(bins=50, figsize=(20, 15))
plt.show()

sns.pairplot(df.select_dtypes(include='number'))
plt.show()
```

```
C:\Users\A R KRISHNA\AppData\Local\Temp\ipykernel_4084\1601767611.py:11: FutureWarning: errors='ignore' is deprecated and will raise in a future versi
on. Use to_numeric without passing `errors` and catch exceptions explicitly instead
  df = df.apply(pd.to_numeric, errors='ignore')
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Pregnancies    45 non-null     int64  
 1   Glucose        45 non-null     int64  
 2   BloodPressure  45 non-null     int64  
 3   SkinThickness  45 non-null     int64  
 4   Insulin        45 non-null     int64  
 5   BMI            45 non-null     float64
dtypes: float64(1), int64(5)
memory usage: 2.2 KB
None
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count    45.000000   45.000000      45.000000      45.000000   45.000000   
mean      5.644444  128.088889      71.133333      19.000000   90.600000   
std       3.484743   31.856798      21.257726      16.742705  158.991338   
min       0.000000   78.000000       0.000000       0.000000    0.000000   
25%       3.000000  103.000000      66.000000       0.000000    0.000000   
50%       6.000000  119.000000      74.000000      23.000000    0.000000   
75%       8.000000  147.000000      84.000000      33.000000  140.000000   
max      13.000000  197.000000     110.000000      47.000000  846.000000   

             BMI  
count  45.000000  
mean   31.646667  
std     8.117898  
min     0.000000  
25%    27.100000  
50%    31.600000  
75%    37.100000  
max    45.800000  
```


Pregnancies
0 1 2 3 4 5 6 7
0 2 4 6 8 10 12
Glucose
0.0 0.5 1.0 1.5 2.0 2.5 3.0
80 100 120 140 160 180 200
BloodPressure
0 1 2 3 4 5
0 20 40 60 80 100
SkinThickness
0 2 4 6 8 10 12 14 16
0 10 20 30 40
Insulin
0 5 10 15 20 25
0 200 400 600 800
BMI
0.0 0.5 1.0 1.5 2.0 2.5 3.0
0 10 20 30 40



Pregnancies
Glucose
BloodPressure
SkinThickness
0
2
4
6
8
10
12
80
100
120
140
160
180
200
20
40
60
80
100


Insulin
BMI
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
0
200
400
600
800
0
10
20
30
40
0
5
10
100
150
200
0
25
50
75
100
0
20
40
0
200
400
600
800
0
20
40

[10]:
```python
import numpy as np
import pandas as pd

# Load dataset
df = pd.read_csv(r"C:\Users\A R KRISHNA\OneDrive\Documents\Hotel_Dataset.csv")

print("Initial Data:")
print(df)
print("\n" + "-"*80 + "\n")

# Remove duplicate rows
df = df.drop_duplicates()

# Drop unwanted column if present
if 'Age_Group.1' in df.columns:
    df = df.drop(['Age_Group.1'], axis=1)

# Replace invalid negative values with NaN
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['NoOfPax'] < 1) | (df['NoOfPax'] > 20), 'NoOfPax'] = np.nan

# Fix text inconsistencies
df['Hotel'] = df['Hotel'].replace(['Ibys'], 'Ibis')
df['FoodPreference'] = df['FoodPreference'].replace(['Vegetarian', 'veg'], 'Veg')
df['FoodPreference'] = df['FoodPreference'].replace(['non-Veg'], 'Non-Veg')

# Fill missing numerical values with mean or median
df['EstimatedSalary'] = df['EstimatedSalary'].fillna(round(df['EstimatedSalary'].mean()))
df['NoOfPax'] = df['NoOfPax'].fillna(round(df['NoOfPax'].median()))
df['Rating(1-5)'] = df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()))
df['Bill'] = df['Bill'].fillna(round(df['Bill'].mean()))

# Display cleaned dataset
print("Cleaned Data:")
print(df)

print("\nDataFrame Info:")
df.info()
```

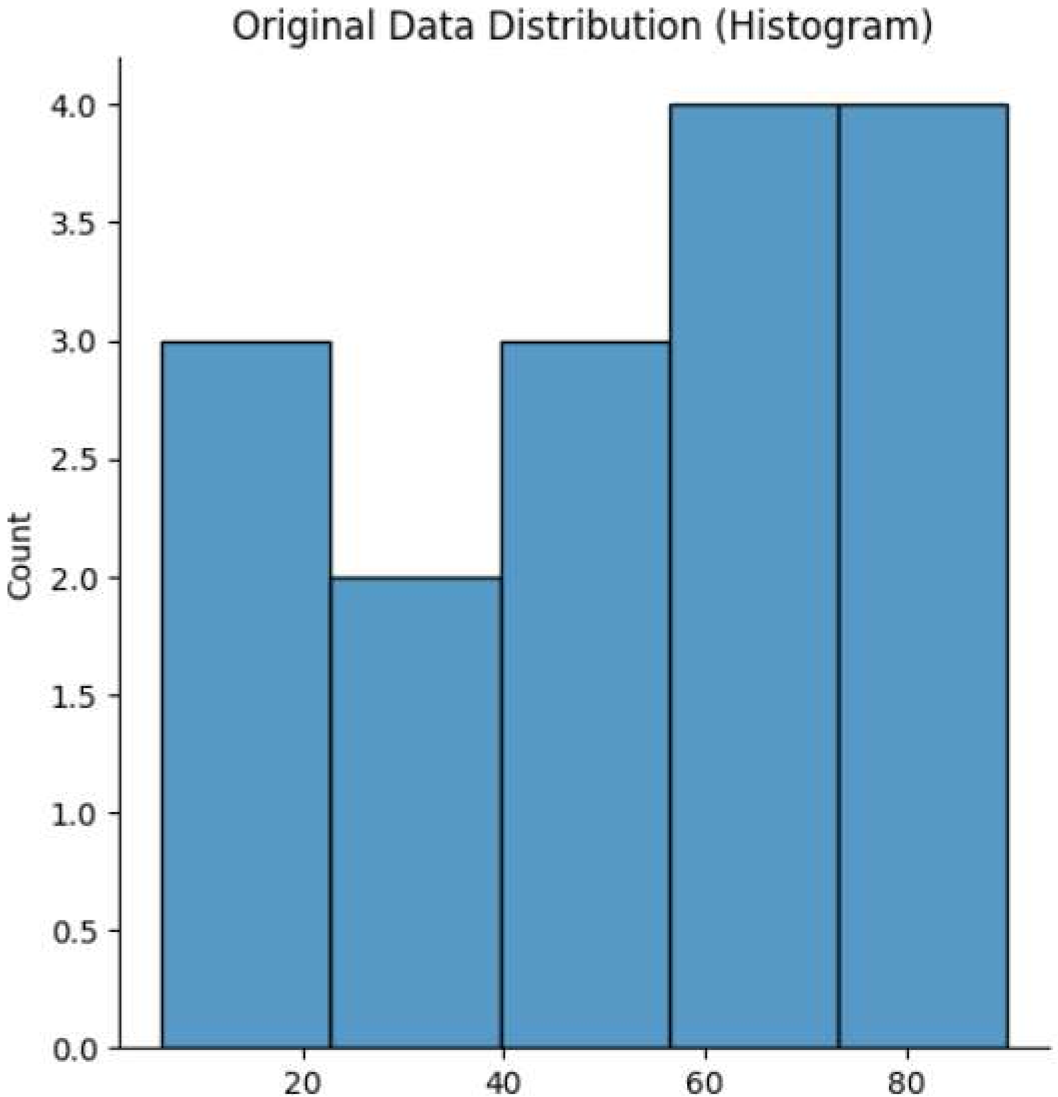
```
Initial Data:
    CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  \
0            1     20-25            4       Ibis            veg  1300
1            2     30-35            5  LemonTree        Non-Veg  2000
2            3     25-30            6     RedFox            Veg  1322
3            4     20-25           -1  LemonTree            Veg  1234
4            5       35+            3       Ibis     Vegetarian   989
5            6       35+            3       Ibys        Non-Veg  1909
6            7       35+            4     RedFox     Vegetarian  1000
7            8     20-25            7  LemonTree            Veg  2999
8            9     25-30            2       Ibis        Non-Veg  3456
9            9     25-30            2       Ibis        Non-Veg  3456
10          10     30-35            5     RedFox        non-Veg -6755

    NoOfPax  EstimatedSalary Age_Group.1
0         2            40000       20-25
1         3            59000       30-35
2         2            30000       25-30
3         2           120000       20-25
4         2            45000         35+
5         2           122220         35+
6        -1            21122         35+
7       -10           345673       20-25
8         3           -99999       25-30
9         3           -99999       25-30
10        4            87777       30-35

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

Cleaned Data:
    CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference    Bill  \
0          1.0     20-25            4       Ibis            Veg  1300.0
1          2.0     30-35            5  LemonTree        Non-Veg  2000.0
2          3.0     25-30            6     RedFox            Veg  1322.0
3          4.0     20-25           -1  LemonTree            Veg  1234.0
4          5.0       35+            3       Ibis            Veg   989.0
5          6.0       35+            3       Ibis        Non-Veg  1909.0
6          7.0       35+            4     RedFox            Veg  1000.0
7          8.0     20-25            7  LemonTree            Veg  2999.0
8          9.0     25-30            2       Ibis        Non-Veg  3456.0
10        10.0     30-35            5     RedFox        Non-Veg  1801.0
```

```
    NoOfPax  EstimatedSalary
0       2.0          40000.0
1       3.0          59000.0
2       2.0          30000.0
3       2.0         120000.0
4       2.0          45000.0
5       2.0         122220.0
6       2.0          21122.0
7       2.0         345673.0
8       3.0          96755.0
10      4.0          87777.0

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, 0 to 10
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CustomerID       10 non-null     float64
 1   Age_Group        10 non-null     object 
 2   Rating(1-5)      10 non-null     int64  
 3   Hotel            10 non-null     object 
 4   FoodPreference   10 non-null     object 
 5   Bill             10 non-null     float64
 6   NoOfPax          10 non-null     float64
 7   EstimatedSalary  10 non-null     float64
dtypes: float64(4), int64(1), object(3)
memory usage: 720.0+ bytes
```

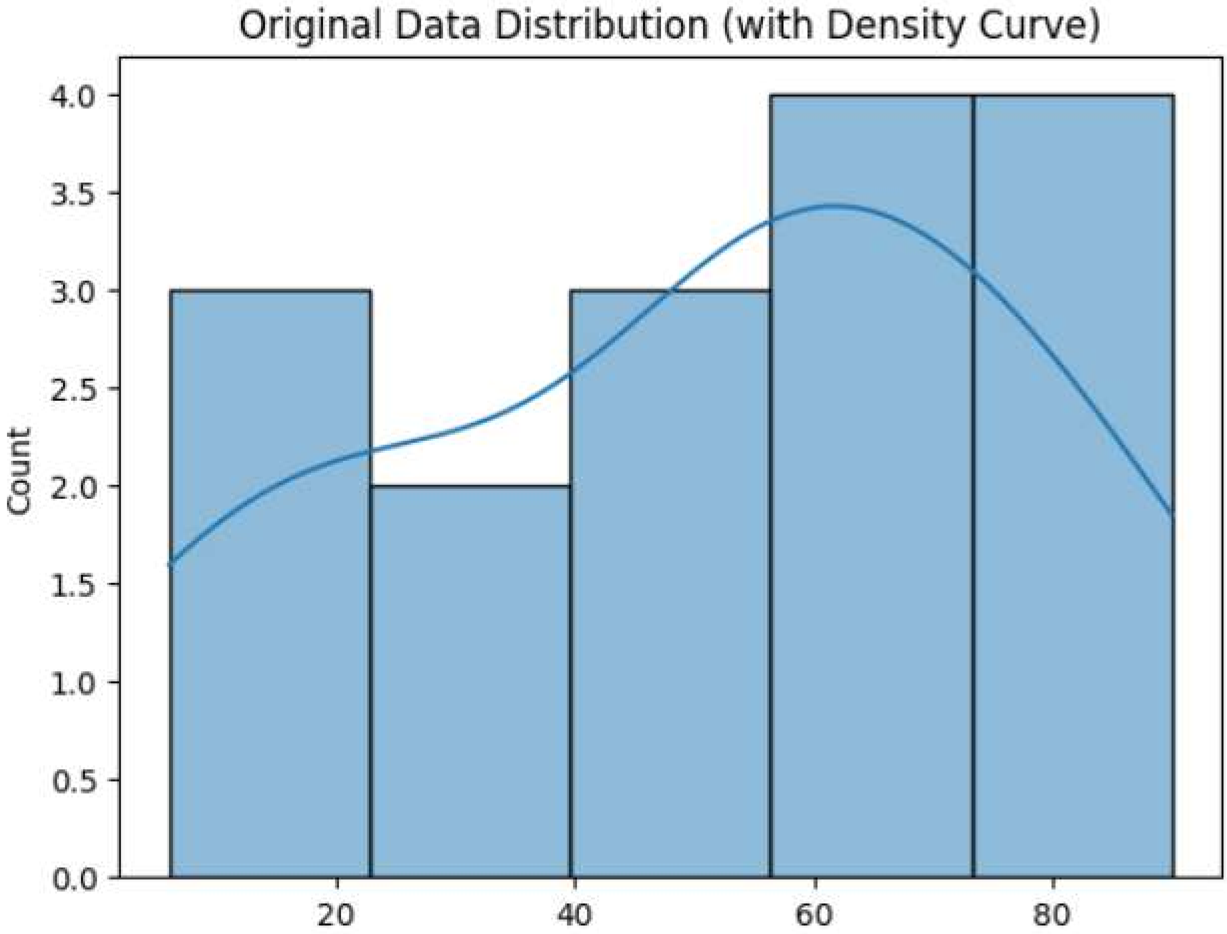
[4]:

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

array = np.array([27, 50, 44, 6, 58, 61, 23, 86, 67, 20, 75, 7, 79, 61, 90, 54])
print("--- Initial Data Analysis ---")
print(f"Array: {array}")
print(f"Mean: {array.mean()}")
print(f"25th Percentile (Q1): {np.percentile(array, 25)}")
print(f"50th Percentile (Median): {np.percentile(array, 50)}")
print(f"75th Percentile (Q3): {np.percentile(array, 75)}")
print(f"100th Percentile (Max): {np.percentile(array, 100)}")
print("-" * 30)
print("\nDisplaying Original Data Histogram...")
sns.displot(array, kind="hist", bins=5)
plt.title("Original Data Distribution (Histogram)")
plt.show()
print("\nDisplaying Original Data Density Plot...")
sns.histplot(array, kde=True, bins=5) # 'bins=5' matches the visual in the file
plt.title("Original Data Distribution (with Density Curve)")
plt.show()
def outDetection(data):
    Q1, Q3 = np.percentile(data, [25, 75])
    IQR = Q3 - Q1
    lr = Q1 - (1.5 * IQR)
    ur = Q3 + (1.5 * IQR)
    return lr, ur

print("\n--- Outlier Range Calculation ---")
lr, ur = outDetection(array)
print(f"Calculated Outlier Range (lr, ur): ({lr}, {ur})")
print("-" * 30)
final_array = array[(array > lr) & (array < ur)]

print("\n--- Filtered Data Result ---")
print(f"Final Array: {final_array}")
print("\nDisplaying Filtered Data Distribution...")
sns.histplot(final_array, kde=True, bins=5)
plt.title("Filtered Data Distribution (No Outliers Found)")
plt.show()
```

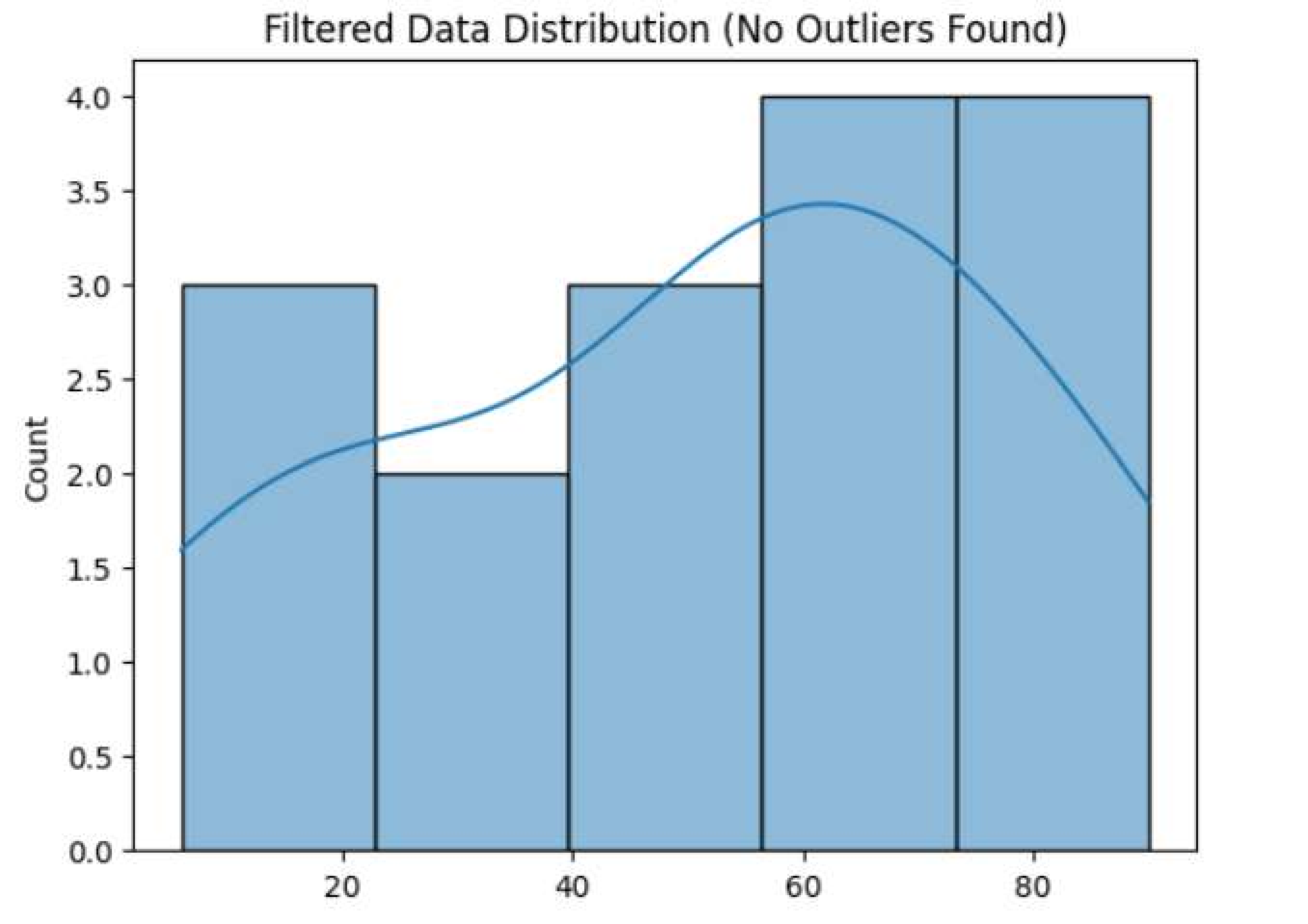
```
--- Initial Data Analysis ---
Array: [27 50 44  6 58 61 23 86 67 20 75  7 79 61 90 54]
Mean: 50.5
25th Percentile (Q1): 26.0
50th Percentile (Median): 56.0
75th Percentile (Q3): 69.0
100th Percentile (Max): 90.0
------------------------------

Displaying Original Data Histogram...
```




```
Displaying Original Data Density Plot...
```




```
--- Outlier Range Calculation ---
Calculated Outlier Range (lr, ur): (-38.5, 133.5)
--------------------------------

--- Filtered Data Result ---
Final Array: [27 50 44  6 58 61 23 86 67 20 75  7 79 61 90 54]

Displaying Filtered Data Distribution...
```



Filtered Data Distribution (No Outliers Found)
Count
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
20
40
60
80

[5]:
```python
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
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("--- Original DataFrame 'df' ---")
print(df)
country_mode = df['Country'].mode()[0]
df['Country'] = df['Country'].fillna(country_mode)
features = df.iloc[:, :-1].values
label = df.iloc[:, -1].values
age_imputer = SimpleImputer(strategy="mean", missing_values=np.nan)
salary_imputer = SimpleImputer(strategy="mean", missing_values=np.nan)
features[:, [1]] = age_imputer.fit_transform(features[:, [1]])
features[:, [2]] = salary_imputer.fit_transform(features[:, [2]])

print("\n--- 'features' array after imputing missing Age/Salary ---")
print(features)
oh = OneHotEncoder(sparse_output=False)
Country_encoded = oh.fit_transform(features[:, [0]])

print("\n--- 'Country' array after OneHotEncoding ---")
print(Country_encoded)
final_set = np.concatenate((Country_encoded, features[:, [1, 2]]), axis=1)

print("\n--- 'final_set' array (concatenated) ---")
print(final_set)
sc = StandardScaler()
feat_standard_scaler = sc.fit_transform(final_set)

print("\n--- 'feat_standard_scaler' (StandardScaler output) ---")
print(feat_standard_scaler)
mms = MinMaxScaler(feature_range=(0, 1))
feat_minmax_scaler = mms.fit_transform(final_set)

print("\n--- 'feat_minmax_scaler' (MinMaxScaler output) ---")
print(feat_minmax_scaler)
```

```
--- Original DataFrame 'df' ---
   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

--- 'features' array after imputing missing Age/Salary ---
[['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]]

--- 'Country' array after OneHotEncoding ---
[[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_set' array (concatenated) ---
[[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]]

--- 'feat_standard_scaler' (StandardScaler output) ---
[[ 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]]

--- 'feat_minmax_scaler' (MinMaxScaler output) ---
[[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]]
```

[10]:

```python
import numpy as np
import pandas as pd

# 1. Re-create the initial DataFrame
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("--- 1. Original DataFrame 'df' (with missing values) ---")
print(df)


# 2. Impute (fill) missing values (Warning-free method)
df.fillna({
    'Country': df['Country'].mode()[0],
    'Age': df['Age'].median(),
    'Salary': round(df['Salary'].mean())
}, inplace=True)

print("\n--- 2. DataFrame 'df' after Imputation ---")
print(df)


# 3. Apply One-Hot Encoding to 'Country' and concatenate
updated_dataset = pd.concat([pd.get_dummies(df.Country), df.iloc[:, [1, 2, 3]]], axis=1)

print("\n--- 3. 'updated_dataset' after One-Hot Encoding 'Country' ---")
print(updated_dataset)


# 4. Replace and explicitly infer types (Warning-free method)
# First, perform the replacement as before
updated_dataset['Purchased'] = updated_dataset['Purchased'].replace(['No', 'Yes'], [0, 1])

# NOW, apply the fix exactly as suggested by the warning
updated_dataset['Purchased'] = updated_dataset['Purchased'].infer_objects(copy=False)


print("\n--- 4. Final 'updated_dataset' after replacing 'Purchased' ---")
print(updated_dataset)

print("\n--- Final 'updated_dataset.info()' (to check dtypes) ---")
updated_dataset.info()
```

```
--- 1. Original DataFrame 'df' (with 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      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

--- 2. DataFrame 'df' after Imputation ---
   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

--- 3. 'updated_dataset' after One-Hot Encoding 'Country' ---
   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
```

```
--- 4. Final 'updated_dataset' after replacing 'Purchased' ---
   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 'updated_dataset.info()' (to check dtypes) ---
<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
C:\Users\A R KRISHNA\AppData\Local\Temp\ipykernel_4764\2742519366.py:37: FutureWarning: Downcasting behavior in `replace` is deprecated and will be re
moved in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.s
et_option('future.no_silent_downcasting', True)`
  updated_dataset['Purchased'] = updated_dataset['Purchased'].replace(['No', 'Yes'], [0, 1])
```

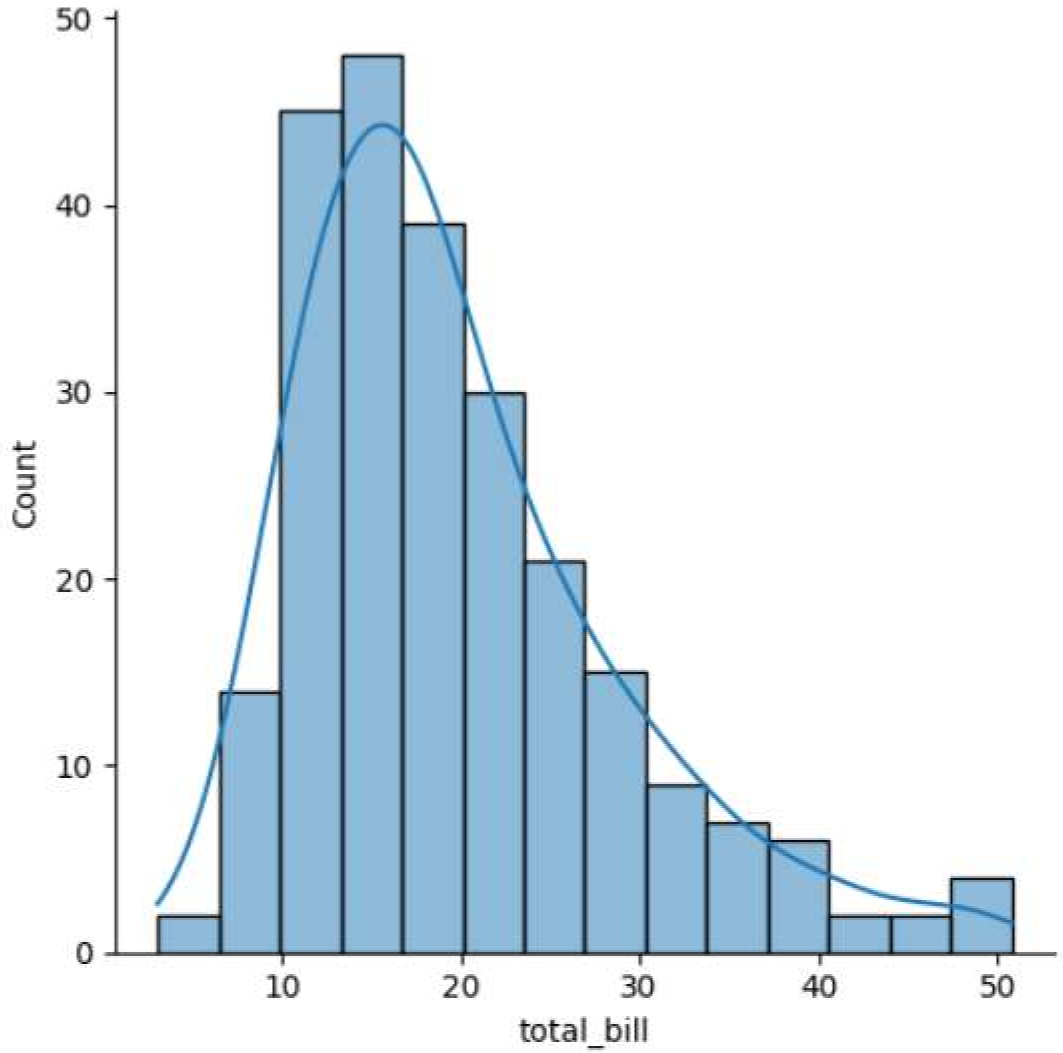
[42]:

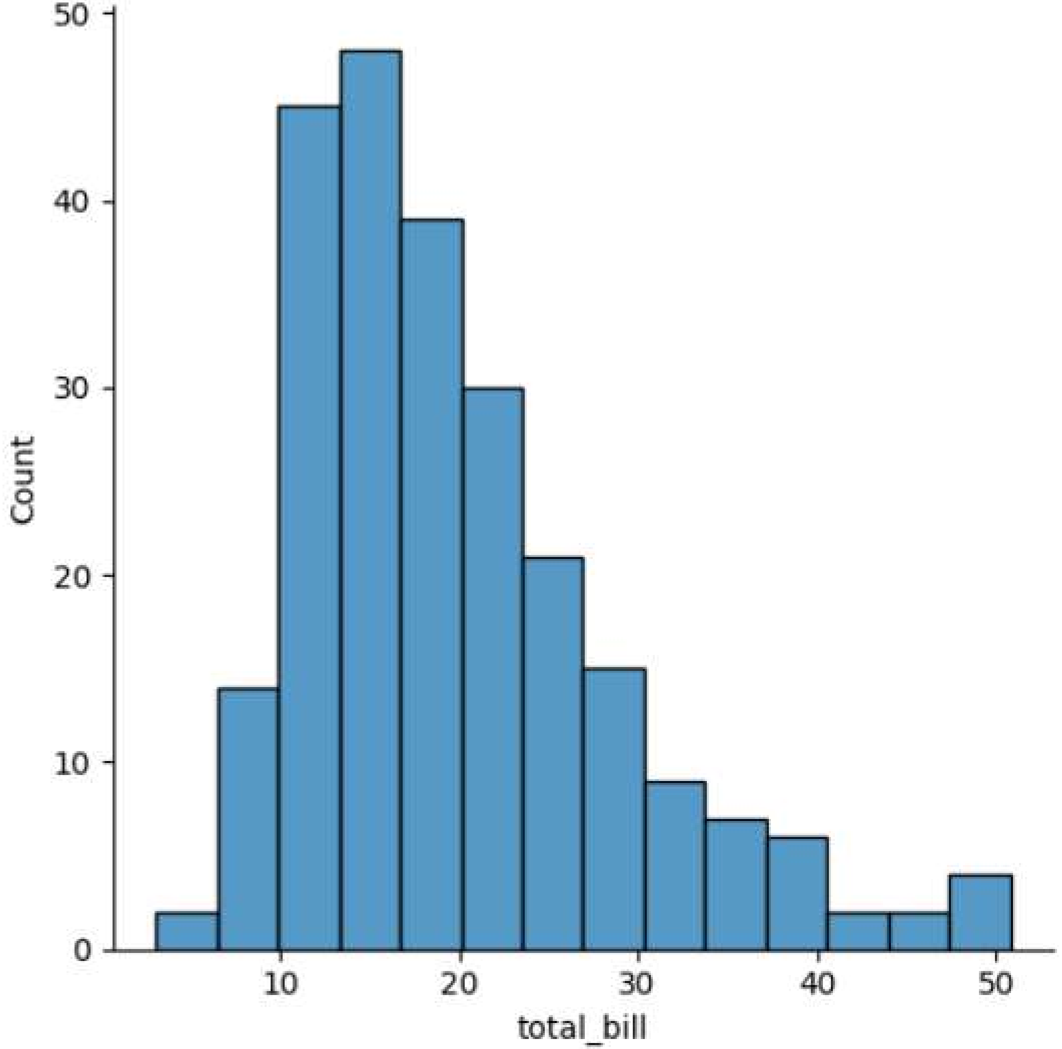
```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
tips=sns.load_dataset('tips')
tips.head()
sns.displot(tips.total_bill,kde=True)
sns.displot(tips.total_bill,kde=False)
sns.jointplot(x=tips.tip,y=tips.total_bill)
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="reg")
sns.jointplot(x=tips.tip,y=tips.total_bill,kind="hex")
sns.pairplot(tips)
tips.time.value_counts()
sns.pairplot(tips,hue='time')
sns.pairplot(tips,hue='day')
sns.heatmap(tips.corr(numeric_only=True),annot=True)
sns.boxplot(tips.total_bill)
sns.boxplot(tips.tip)
sns.countplot(tips.day)
sns.countplot(tips.sex)
tips.sex.value_counts().plot(kind='pie')
tips.sex.value_counts().plot(kind='bar')
sns.countplot(tips[tips.time=='Dinner']['day'])
```

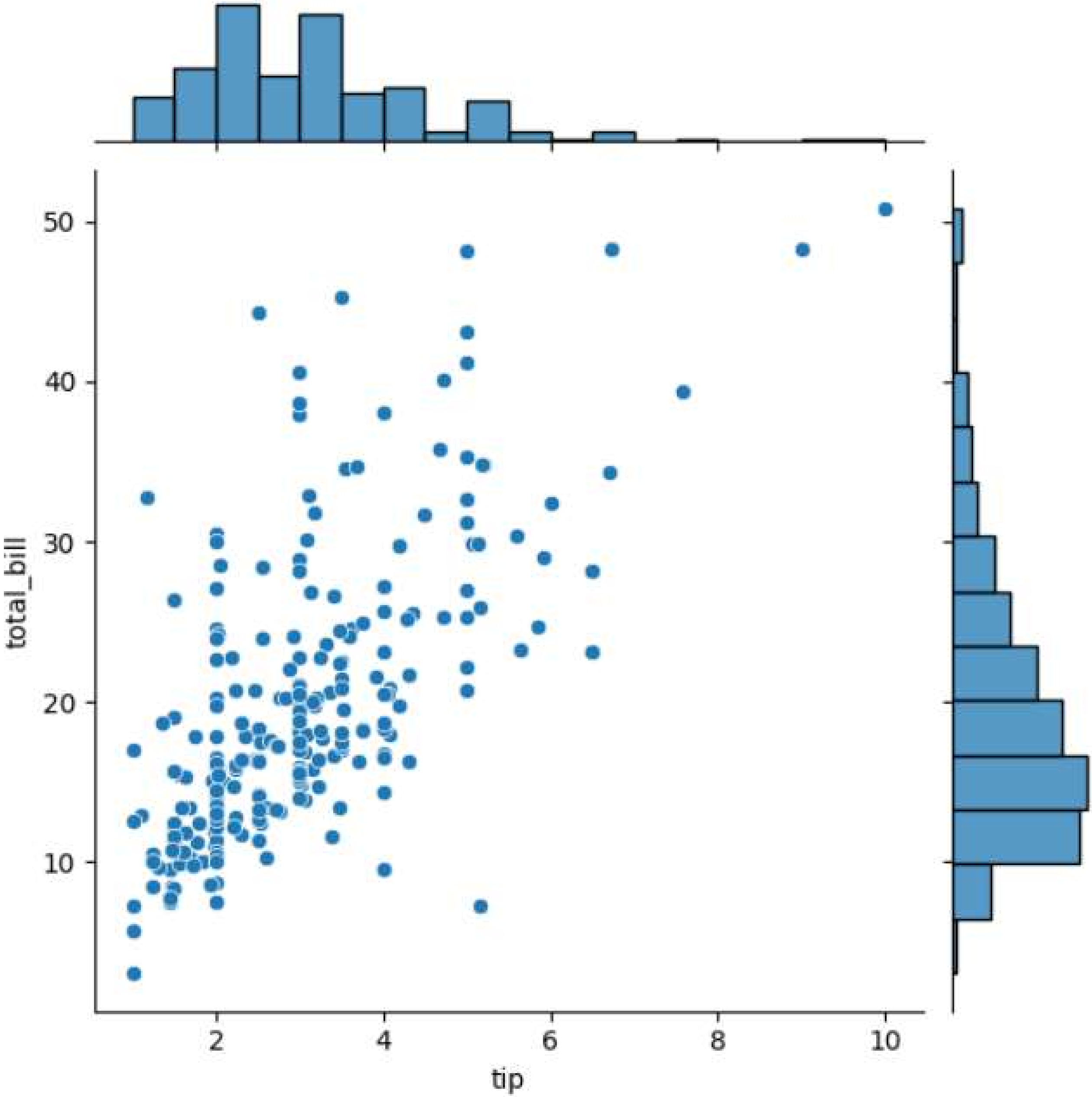
```
Ignoring fixed y limits to fulfill fixed data aspect with adjustable data limits.
C:\Users\A R KRISHNA\AppData\Local\Programs\Python\Python312\Lib\site-packages\seaborn\categorical.py:383: UserWarning: Attempting to set identical lo
w and high ylims makes transformation singular; automatically expanding.
  ax.set_ylim(n - .5, -.5, auto=None)
```

[42]:
```
<Axes: xlabel='sex', ylabel='count'>
```

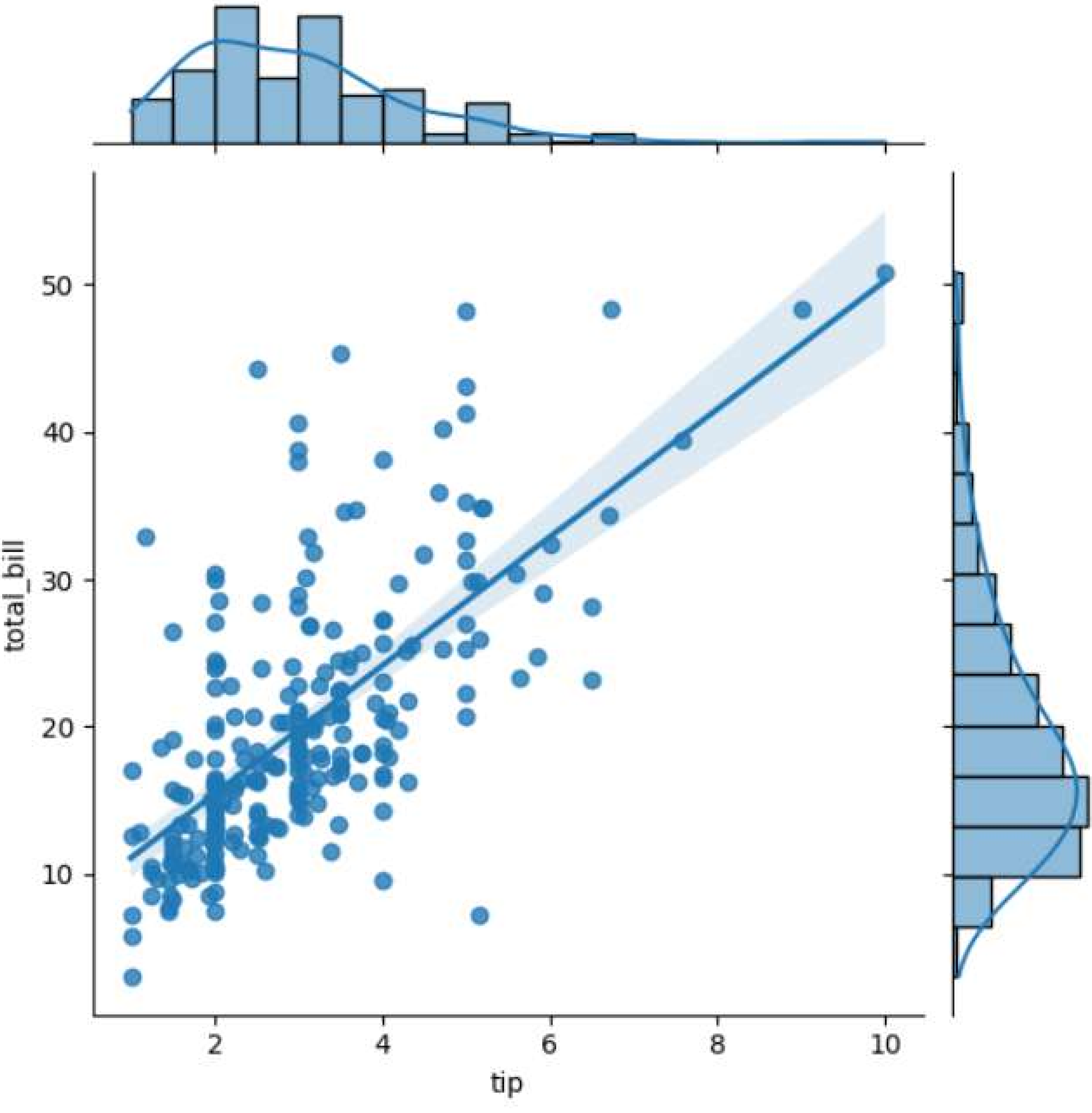
```
Ignoring fixed y limits to fulfill fixed data aspect with adjustable data limits.
```



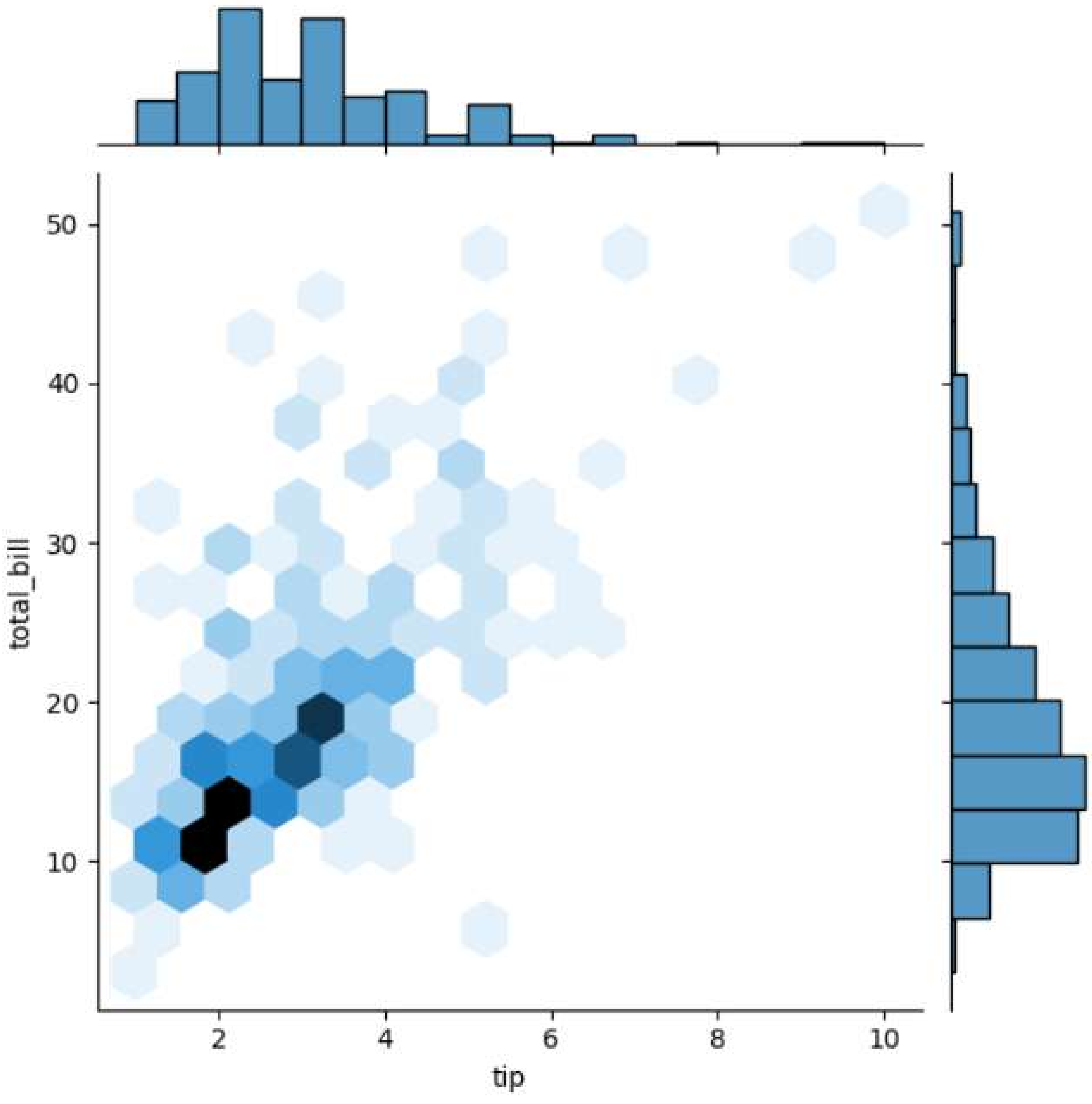

Count
total_bill
0
10
20
30
40
50
10
20
30
40
50



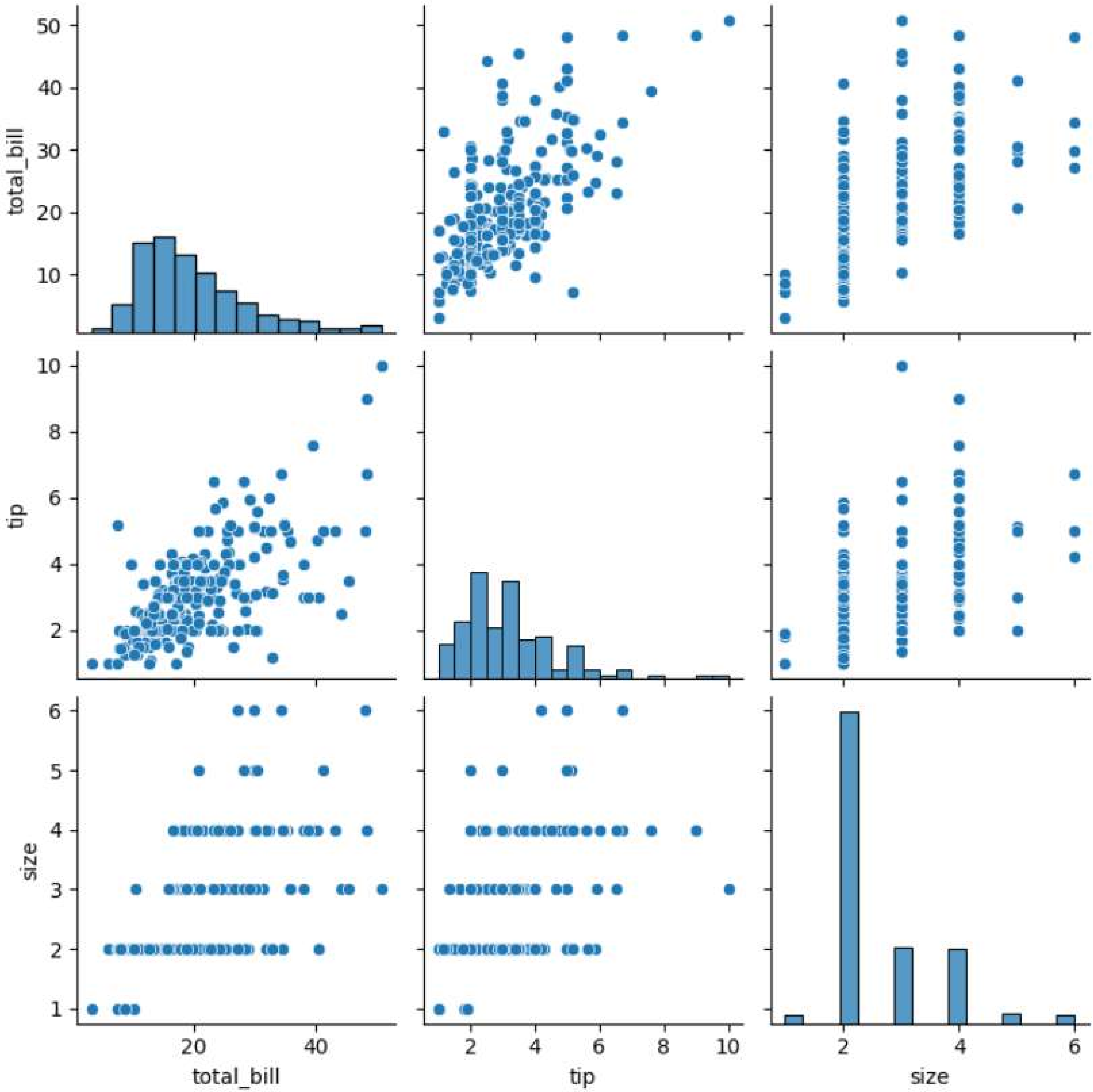
total_bill
tip
50
40
30
20
10
2
4
6
8
10



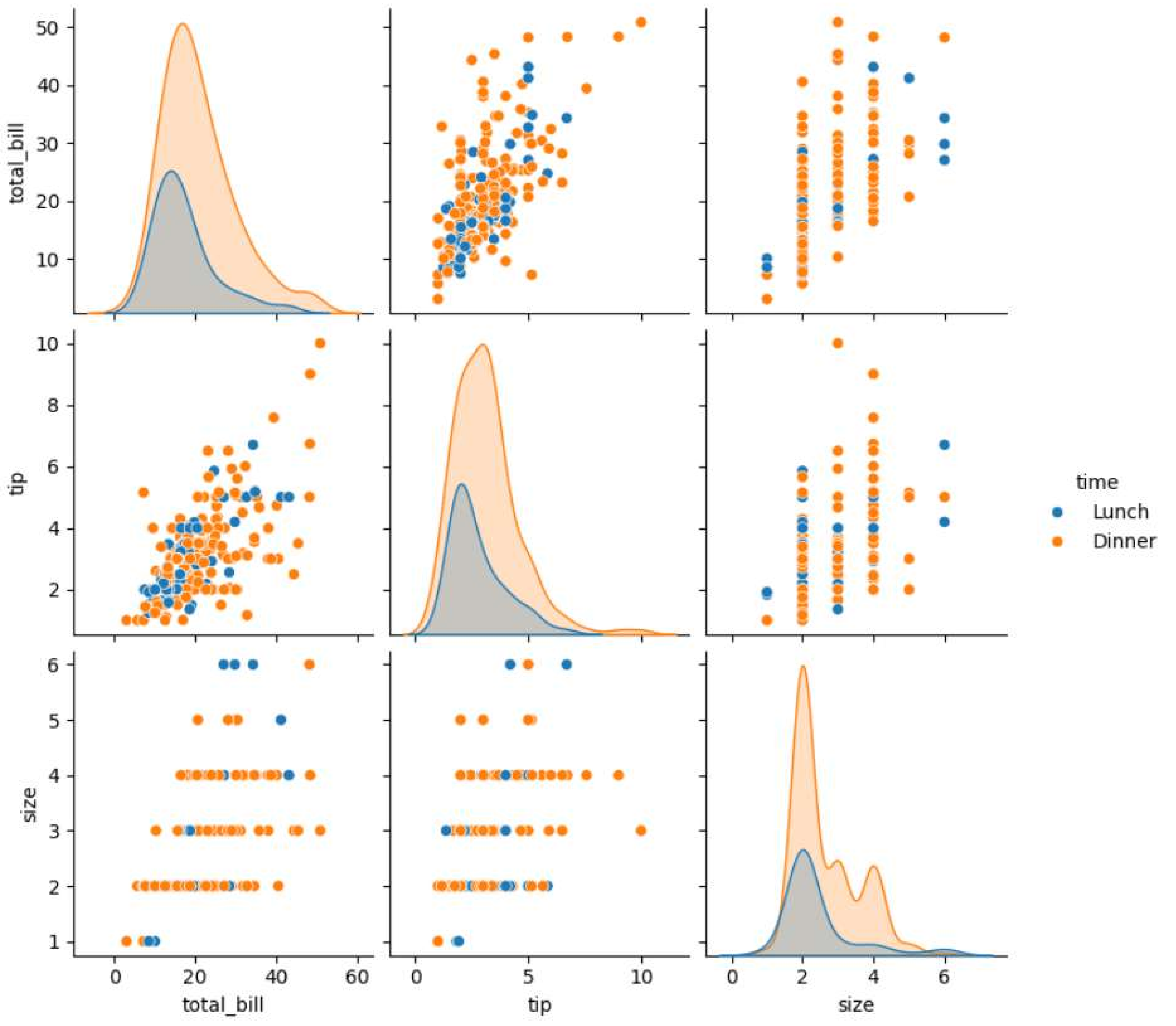
total_bill
tip
10
20
30
40
50
2
4
6
8
10



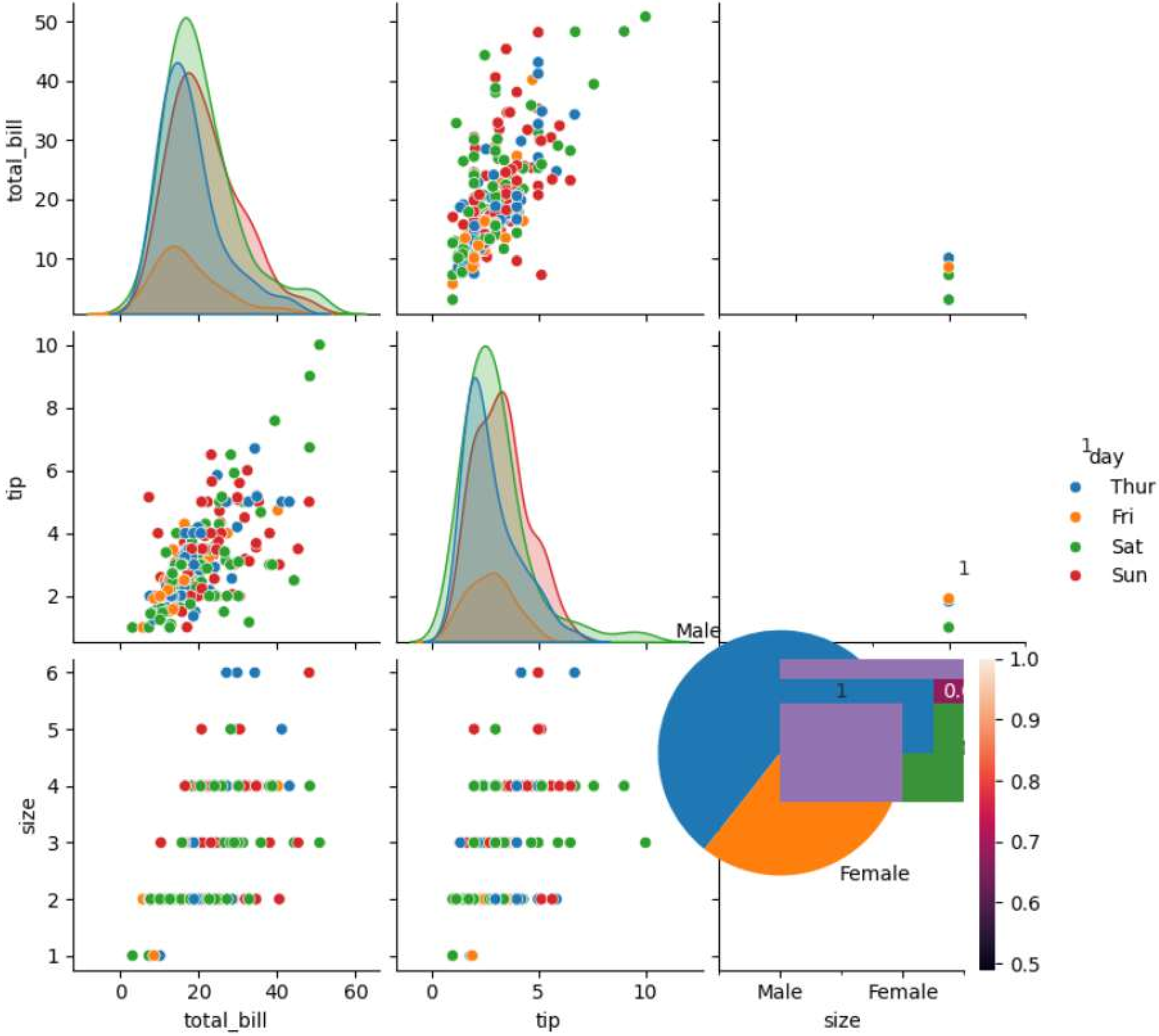
total_bill
tip
50
40
30
20
10
2
4
6
8
10



total_bill
tip
size
total_bill
tip
size



total_bill
tip
size
time
Lunch
Dinner



total_bill
tip
size
day
Thur
Fri
Sat
Sun
Male
Female
1
0.5
0.6
0.7
0.8
0.9
1.0

[46]:

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pickle

df = pd.read_csv(r"C:\Users\A R KRISHNA\Downloads\Salary_data.csv")
df.dropna(inplace=True)
features = df.iloc[:, [0]].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=42)
model = LinearRegression()
model.fit(x_train, y_train)
print("Training Score:", model.score(x_train, y_train))
print("Testing Score:", model.score(x_test, y_test))
print("Model Coefficients:", model.coef_)
print("Model Intercept:", model.intercept_)
pickle.dump(model, open('SalaryPred.model', 'wb'))
model = pickle.load(open('SalaryPred.model', 'rb'))

yr_of_exp = float(input("Enter Years of Experience: "))
yr_of_exp_NP = np.array([[yr_of_exp]])
Salary = model.predict(yr_of_exp_NP)
print("Estimated Salary for {} years of experience is {}:".format(yr_of_exp, Salary[0][0]))
```

```
Training Score: 0.9645401573418146
Testing Score: 0.9024461774180497
Model Coefficients: [[9423.81532303]]
Model Intercept: [25321.58301178]
Enter Years of Experience:  26
Estimated Salary for 26.0 years of experience is 270340.7814105822:
```

[2]:
```
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

df = pd.read_csv(r"C:\Users\A R KRISHNA\Downloads\LogisticsRegression.csv")
features = df.iloc[:, [2, 3]].values
label = df.iloc[:, 4].values
for i in range(1, 401):
    x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.8, random_state=i)
    model = LogisticRegression()
    model.fit(x_train, y_train)
    train_score = model.score(x_train, y_train)
    test_score = model.score(x_test, y_test)
    if test_score > train_score:
        print("Test {:.4f} Train {:.4f} Random State {}".format(test_score, train_score, i))

x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=42)
finalModel = LogisticRegression()
finalModel.fit(x_train, y_train)
print("Train Accuracy:", finalModel.score(x_train, y_train))
print("Test Accuracy:", finalModel.score(x_test, y_test))
print("\nClassification Report:\n", classification_report(label, finalModel.predict(features)))
```

```
Test 0.8375 Train 0.8125 Random State 2
Test 0.8313 Train 0.8000 Random State 4
Test 0.8406 Train 0.8000 Random State 8
Test 0.8375 Train 0.8250 Random State 9
Test 0.8531 Train 0.7750 Random State 13
Test 0.8719 Train 0.7875 Random State 14
Test 0.8656 Train 0.8625 Random State 15
Test 0.8688 Train 0.8375 Random State 17
Test 0.8406 Train 0.7750 Random State 18
Test 0.8313 Train 0.8250 Random State 19
Test 0.8375 Train 0.8250 Random State 21
Test 0.8469 Train 0.8375 Random State 22
Test 0.8531 Train 0.8500 Random State 25
Test 0.8438 Train 0.8125 Random State 26
Test 0.8688 Train 0.7750 Random State 27
Test 0.8250 Train 0.8125 Random State 29
Test 0.8219 Train 0.8125 Random State 33
Test 0.8344 Train 0.8250 Random State 34
Test 0.8375 Train 0.8250 Random State 35
Test 0.8406 Train 0.8250 Random State 42
Test 0.8406 Train 0.8125 Random State 47
Test 0.8594 Train 0.8375 Random State 53
Test 0.8313 Train 0.8250 Random State 55
Test 0.8469 Train 0.8375 Random State 56
Test 0.8281 Train 0.8250 Random State 59
Test 0.8375 Train 0.7750 Random State 68
Test 0.8531 Train 0.8500 Random State 71
Test 0.8719 Train 0.8375 Random State 74
Test 0.8594 Train 0.8375 Random State 80
Test 0.8625 Train 0.7500 Random State 82
Test 0.8656 Train 0.8125 Random State 83
Test 0.8469 Train 0.8375 Random State 86
Test 0.8375 Train 0.8250 Random State 93
Test 0.8438 Train 0.8250 Random State 99
Test 0.8562 Train 0.8500 Random State 100
Test 0.8625 Train 0.7625 Random State 105
Test 0.8406 Train 0.8375 Random State 106
Test 0.8406 Train 0.8375 Random State 110
Test 0.8469 Train 0.7375 Random State 111
```

```
Test 0.8469 Train 0.8125 Random State 113
Test 0.8594 Train 0.8125 Random State 118
Test 0.8313 Train 0.8125 Random State 120
Test 0.8469 Train 0.8250 Random State 122
Test 0.8406 Train 0.8375 Random State 123
Test 0.8469 Train 0.8125 Random State 134
Test 0.8562 Train 0.8500 Random State 135
Test 0.8438 Train 0.8125 Random State 136
Test 0.8344 Train 0.8250 Random State 138
Test 0.8313 Train 0.8000 Random State 146
Test 0.8781 Train 0.8250 Random State 150
Test 0.8688 Train 0.8000 Random State 152
Test 0.8500 Train 0.8125 Random State 153
Test 0.8688 Train 0.8375 Random State 154
Test 0.8375 Train 0.7750 Random State 155
Test 0.8625 Train 0.8375 Random State 156
Test 0.8594 Train 0.8250 Random State 161
Test 0.8281 Train 0.8000 Random State 163
Test 0.8594 Train 0.8500 Random State 171
Test 0.8469 Train 0.8250 Random State 173
Test 0.8313 Train 0.8125 Random State 175
Test 0.8438 Train 0.8250 Random State 176
Test 0.8406 Train 0.8250 Random State 180
Test 0.8344 Train 0.8250 Random State 185
Test 0.8375 Train 0.8250 Random State 186
Test 0.8187 Train 0.7875 Random State 187
Test 0.8344 Train 0.8250 Random State 194
Test 0.8406 Train 0.8250 Random State 198
Test 0.8344 Train 0.7875 Random State 200
Test 0.8531 Train 0.8125 Random State 201
Test 0.8344 Train 0.8125 Random State 202
Test 0.8438 Train 0.8375 Random State 207
Test 0.8438 Train 0.8375 Random State 211
Test 0.8438 Train 0.8125 Random State 213
Test 0.8531 Train 0.8125 Random State 215
Test 0.8594 Train 0.8250 Random State 217
Test 0.8375 Train 0.7750 Random State 219
Test 0.8594 Train 0.7750 Random State 223
Test 0.8469 Train 0.8000 Random State 226
```

```
Test 0.8531 Train 0.8500 Random State 227
Test 0.8500 Train 0.7750 Random State 228
Test 0.8500 Train 0.7875 Random State 229
Test 0.8594 Train 0.8375 Random State 232
Test 0.8531 Train 0.8250 Random State 240
Test 0.8469 Train 0.7875 Random State 241
Test 0.8688 Train 0.8500 Random State 242
Test 0.8656 Train 0.8250 Random State 245
Test 0.8688 Train 0.8375 Random State 247
Test 0.8438 Train 0.8375 Random State 251
Test 0.8562 Train 0.8500 Random State 252
Test 0.8781 Train 0.8000 Random State 256
Test 0.8406 Train 0.8250 Random State 259
Test 0.8438 Train 0.8375 Random State 262
Test 0.8562 Train 0.8000 Random State 273
Test 0.8531 Train 0.8500 Random State 276
Test 0.8313 Train 0.8000 Random State 284
Test 0.8156 Train 0.7125 Random State 290
Test 0.8469 Train 0.8250 Random State 292
Test 0.8469 Train 0.8375 Random State 293
Test 0.8313 Train 0.8250 Random State 298
Test 0.8344 Train 0.8250 Random State 301
Test 0.8406 Train 0.8250 Random State 303
Test 0.8656 Train 0.8250 Random State 306
Test 0.8344 Train 0.8000 Random State 307
Test 0.8719 Train 0.7625 Random State 308
Test 0.8562 Train 0.8500 Random State 317
Test 0.8562 Train 0.8375 Random State 318
Test 0.8531 Train 0.8250 Random State 322
Test 0.8250 Train 0.8125 Random State 328
Test 0.8281 Train 0.8125 Random State 329
Test 0.8469 Train 0.8250 Random State 336
Test 0.8375 Train 0.8250 Random State 338
Test 0.8469 Train 0.8375 Random State 344
Test 0.8500 Train 0.8375 Random State 346
Test 0.8750 Train 0.8125 Random State 349
Test 0.8500 Train 0.8000 Random State 352
Test 0.8500 Train 0.7875 Random State 355
Test 0.8531 Train 0.8125 Random State 371
```

```
Test 0.8469 Train 0.8125 Random State 372
Test 0.8375 Train 0.7750 Random State 373
Test 0.8562 Train 0.8375 Random State 381
Test 0.8500 Train 0.8125 Random State 382
Test 0.8250 Train 0.8125 Random State 383
Test 0.8656 Train 0.7625 Random State 386
Test 0.8594 Train 0.7875 Random State 393
Test 0.8531 Train 0.8500 Random State 395
Test 0.8719 Train 0.7625 Random State 397
Test 0.8438 Train 0.8375 Random State 398
Test 0.8438 Train 0.7750 Random State 399
Test 0.8156 Train 0.8125 Random State 400
Train Accuracy: 0.8375
Test Accuracy: 0.8875

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89       257
           1       0.85      0.70      0.77       143

    accuracy                           0.85       400
   macro avg       0.85      0.81      0.83       400
weighted avg       0.85      0.85      0.84       400
```

[12]:

```python
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

try:
    df = pd.read_csv(r"C:\Users\A R KRISHNA\Downloads\Social_Network_Ads.csv")
except FileNotFoundError:
    print("Error: 'Social_Network_Ads.csv' not found.")
    df = pd.DataFrame({
        'Age': [19, 35, 26, 27, 19],
        'EstimatedSalary': [19000, 20000, 43000, 57000, 76000],
        'Purchased': [0, 0, 0, 0, 0]
    })
    print("Using dummy data. Please add the correct CSV file to get real results.")


print("--- Original DataFrame Head ---")
print(df.head())
print("-" * 30)
features = df.iloc[:, [2, 3]].values
label = df.iloc[:, 4].values
print("\n--- Experiment 1 (In [7]) Output ---")
for i in range(1, 401):
    x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2, random_state=i)
    model = LogisticRegression()
    model.fit(x_train, y_train)
    train_score = model.score(x_train, y_train)
    test_score = model.score(x_test, y_test)
    if test_score > train_score:
        print("Test {:.4f} Train {:.4f} Random State {}".format(test_score, train_score, i))
print("-" * 30)
```

```python
print("\n--- Experiment 2 (In [8] - [10]) ---")
x_train, x_test, y_train, y_test = train_test_split(features, label, test_size=0.2)

finalModel = LogisticRegression()
print(f"Model: {finalModel}") #
finalModel.fit(x_train, y_train)
print("\nScores from In [9]:")
print(f"Train Score: {finalModel.score(x_train, y_train)}")
print(f"Test Score: {finalModel.score(x_test, y_test)}")
print("\nClassification Report from In [10]:")
print(classification_report(label, finalModel.predict(features)))
print("-" * 30)
```

```
--- Original DataFrame Head ---
    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
------------------------------

--- Experiment 1 (In [7]) Output ---
Test 0.9000 Train 0.8406 Random State 4
Test 0.8625 Train 0.8500 Random State 5
Test 0.8625 Train 0.8594 Random State 6
Test 0.8875 Train 0.8375 Random State 7
Test 0.8625 Train 0.8375 Random State 9
Test 0.9000 Train 0.8406 Random State 10
Test 0.8625 Train 0.8562 Random State 14
Test 0.8500 Train 0.8438 Random State 15
Test 0.8625 Train 0.8562 Random State 16
Test 0.8750 Train 0.8344 Random State 18
Test 0.8500 Train 0.8438 Random State 19
Test 0.8750 Train 0.8438 Random State 20
```

```
Test 0.8625 Train 0.8344 Random State 21
Test 0.8750 Train 0.8406 Random State 22
Test 0.8750 Train 0.8406 Random State 24
Test 0.8500 Train 0.8344 Random State 26
Test 0.8500 Train 0.8406 Random State 27
Test 0.8625 Train 0.8344 Random State 30
Test 0.8625 Train 0.8562 Random State 31
Test 0.8750 Train 0.8531 Random State 32
Test 0.8625 Train 0.8438 Random State 33
Test 0.8750 Train 0.8313 Random State 35
Test 0.8625 Train 0.8531 Random State 36
Test 0.8875 Train 0.8406 Random State 38
Test 0.8750 Train 0.8375 Random State 39
Test 0.8875 Train 0.8375 Random State 42
Test 0.8750 Train 0.8469 Random State 46
Test 0.9125 Train 0.8313 Random State 47
Test 0.8750 Train 0.8313 Random State 51
Test 0.9000 Train 0.8438 Random State 54
Test 0.8500 Train 0.8438 Random State 57
Test 0.8750 Train 0.8438 Random State 58
Test 0.9250 Train 0.8375 Random State 61
Test 0.8875 Train 0.8344 Random State 65
Test 0.8875 Train 0.8406 Random State 68
Test 0.9000 Train 0.8313 Random State 72
Test 0.8875 Train 0.8375 Random State 75
Test 0.9250 Train 0.8250 Random State 76
Test 0.8625 Train 0.8406 Random State 77
Test 0.8625 Train 0.8594 Random State 81
Test 0.8750 Train 0.8375 Random State 82
Test 0.8875 Train 0.8375 Random State 83
Test 0.8625 Train 0.8531 Random State 84
Test 0.8625 Train 0.8406 Random State 85
Test 0.8625 Train 0.8406 Random State 87
Test 0.8750 Train 0.8469 Random State 88
Test 0.9125 Train 0.8375 Random State 90
Test 0.8625 Train 0.8500 Random State 95
```

```
Test 0.8750 Train 0.8500 Random State 99
Test 0.8500 Train 0.8406 Random State 101
Test 0.8500 Train 0.8406 Random State 102
Test 0.9000 Train 0.8250 Random State 106
Test 0.8625 Train 0.8406 Random State 107
Test 0.8500 Train 0.8344 Random State 109
Test 0.8500 Train 0.8406 Random State 111
Test 0.9125 Train 0.8406 Random State 112
Test 0.8625 Train 0.8500 Random State 115
Test 0.8625 Train 0.8406 Random State 116
Test 0.8750 Train 0.8344 Random State 119
Test 0.9125 Train 0.8281 Random State 120
Test 0.8625 Train 0.8594 Random State 125
Test 0.8500 Train 0.8469 Random State 128
Test 0.8750 Train 0.8500 Random State 130
Test 0.9000 Train 0.8438 Random State 133
Test 0.9250 Train 0.8344 Random State 134
Test 0.8625 Train 0.8500 Random State 135
Test 0.8750 Train 0.8313 Random State 138
Test 0.8625 Train 0.8500 Random State 141
Test 0.8500 Train 0.8469 Random State 143
Test 0.8500 Train 0.8469 Random State 146
Test 0.8500 Train 0.8438 Random State 147
Test 0.8625 Train 0.8500 Random State 148
Test 0.8750 Train 0.8375 Random State 150
Test 0.8875 Train 0.8313 Random State 151
Test 0.9250 Train 0.8438 Random State 152
Test 0.8500 Train 0.8406 Random State 153
Test 0.9000 Train 0.8438 Random State 154
Test 0.9000 Train 0.8406 Random State 155
Test 0.8875 Train 0.8469 Random State 156
Test 0.8875 Train 0.8344 Random State 158
Test 0.8750 Train 0.8281 Random State 159
Test 0.9000 Train 0.8313 Random State 161
Test 0.8500 Train 0.8375 Random State 163
Test 0.8750 Train 0.8313 Random State 164
```

```
Test 0.8625 Train 0.8500 Random State 169
Test 0.8750 Train 0.8406 Random State 171
Test 0.8500 Train 0.8406 Random State 172
Test 0.9000 Train 0.8250 Random State 180
Test 0.8500 Train 0.8344 Random State 184
Test 0.9250 Train 0.8219 Random State 186
Test 0.9000 Train 0.8313 Random State 193
Test 0.8625 Train 0.8500 Random State 195
Test 0.8625 Train 0.8406 Random State 196
Test 0.8625 Train 0.8375 Random State 197
Test 0.8750 Train 0.8406 Random State 198
Test 0.8875 Train 0.8375 Random State 199
Test 0.8875 Train 0.8438 Random State 200
Test 0.8625 Train 0.8375 Random State 202
Test 0.8625 Train 0.8406 Random State 203
Test 0.8875 Train 0.8313 Random State 206
Test 0.8625 Train 0.8344 Random State 211
Test 0.8500 Train 0.8438 Random State 212
Test 0.8625 Train 0.8344 Random State 214
Test 0.8750 Train 0.8313 Random State 217
Test 0.9625 Train 0.8187 Random State 220
Test 0.8750 Train 0.8438 Random State 221
Test 0.8500 Train 0.8406 Random State 222
Test 0.9000 Train 0.8438 Random State 223
Test 0.8625 Train 0.8531 Random State 227
Test 0.8625 Train 0.8344 Random State 228
Test 0.9000 Train 0.8406 Random State 229
Test 0.8500 Train 0.8438 Random State 232
Test 0.8750 Train 0.8469 Random State 233
Test 0.9125 Train 0.8406 Random State 234
Test 0.8625 Train 0.8406 Random State 235
Test 0.8500 Train 0.8469 Random State 236
Test 0.8750 Train 0.8469 Random State 239
Test 0.8500 Train 0.8438 Random State 241
Test 0.8875 Train 0.8500 Random State 242
Test 0.8875 Train 0.8250 Random State 243
```

```
Test 0.8750 Train 0.8469 Random State 244
Test 0.8750 Train 0.8406 Random State 245
Test 0.8750 Train 0.8469 Random State 246
Test 0.8625 Train 0.8594 Random State 247
Test 0.8875 Train 0.8438 Random State 248
Test 0.8625 Train 0.8500 Random State 250
Test 0.8750 Train 0.8313 Random State 251
Test 0.8875 Train 0.8438 Random State 252
Test 0.8625 Train 0.8469 Random State 255
Test 0.9000 Train 0.8406 Random State 257
Test 0.8625 Train 0.8562 Random State 260
Test 0.8625 Train 0.8406 Random State 266
Test 0.8625 Train 0.8375 Random State 268
Test 0.8750 Train 0.8406 Random State 275
Test 0.8625 Train 0.8500 Random State 276
Test 0.9250 Train 0.8375 Random State 277
Test 0.8750 Train 0.8469 Random State 282
Test 0.8500 Train 0.8469 Random State 283
Test 0.8500 Train 0.8438 Random State 285
Test 0.9125 Train 0.8344 Random State 286
Test 0.8500 Train 0.8406 Random State 290
Test 0.8500 Train 0.8406 Random State 291
Test 0.8500 Train 0.8469 Random State 292
Test 0.8625 Train 0.8375 Random State 294
Test 0.8875 Train 0.8281 Random State 297
Test 0.8625 Train 0.8344 Random State 300
Test 0.8625 Train 0.8500 Random State 301
Test 0.8875 Train 0.8500 Random State 302
Test 0.8750 Train 0.8469 Random State 303
Test 0.8625 Train 0.8344 Random State 305
Test 0.9125 Train 0.8375 Random State 306
Test 0.8750 Train 0.8469 Random State 308
Test 0.9000 Train 0.8438 Random State 311
Test 0.8625 Train 0.8344 Random State 313
Test 0.9125 Train 0.8344 Random State 314
Test 0.8750 Train 0.8375 Random State 315
```

```
Test 0.9000 Train 0.8469 Random State 317
Test 0.9125 Train 0.8219 Random State 319
Test 0.8625 Train 0.8500 Random State 321
Test 0.9125 Train 0.8281 Random State 322
Test 0.8500 Train 0.8469 Random State 328
Test 0.8500 Train 0.8375 Random State 332
Test 0.8875 Train 0.8531 Random State 336
Test 0.8500 Train 0.8375 Random State 337
Test 0.8750 Train 0.8406 Random State 343
Test 0.8625 Train 0.8438 Random State 346
Test 0.8875 Train 0.8313 Random State 351
Test 0.8625 Train 0.8500 Random State 352
Test 0.9500 Train 0.8187 Random State 354
Test 0.8625 Train 0.8500 Random State 356
Test 0.9125 Train 0.8406 Random State 357
Test 0.8625 Train 0.8375 Random State 358
Test 0.8500 Train 0.8406 Random State 362
Test 0.9000 Train 0.8438 Random State 363
Test 0.8625 Train 0.8531 Random State 364
Test 0.9375 Train 0.8219 Random State 366
Test 0.9125 Train 0.8406 Random State 369
Test 0.8625 Train 0.8531 Random State 371
Test 0.9250 Train 0.8344 Random State 376
Test 0.9125 Train 0.8281 Random State 377
Test 0.8875 Train 0.8500 Random State 378
Test 0.8875 Train 0.8500 Random State 379
Test 0.8625 Train 0.8406 Random State 382
Test 0.8625 Train 0.8594 Random State 386
Test 0.8500 Train 0.8375 Random State 387
Test 0.8750 Train 0.8281 Random State 388
Test 0.8500 Train 0.8438 Random State 394
Test 0.8625 Train 0.8375 Random State 395
Test 0.9000 Train 0.8438 Random State 397
Test 0.8625 Train 0.8438 Random State 400
----------------------------------------
```

```
--- Experiment 2 (In [8] - [10]) ---
Model: LogisticRegression()

Scores from In [9]:
Train Score: 0.8375
Test Score: 0.85

Classification Report from In [10]:
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       257
           1       0.83      0.69      0.76       143

    accuracy                           0.84       400
   macro avg       0.84      0.81      0.82       400
weighted avg       0.84      0.84      0.84       400

------------------------------
```

```
[4]: import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report

df = pd.read_csv(r"C:\Users\A R KRISHNA\OneDrive\Documents\Iris.csv")
features = df.iloc[:, :-1].values
label = df.iloc[:, 4].values
xtrain, xtest, ytrain, ytest = train_test_split(features, label, test_size=0.2, random_state=42)
model_KNN = KNeighborsClassifier(n_neighbors=5)
model_KNN.fit(xtrain, ytrain)
print("Train Accuracy:", model_KNN.score(xtrain, ytrain))
print("Test Accuracy:", model_KNN.score(xtest, ytest))
print("Confusion Matrix:\n", confusion_matrix(label, model_KNN.predict(features)))
print("\nClassification Report:\n", classification_report(label, model_KNN.predict(features)))
```

```
Train Accuracy: 0.9666666666666667
Test Accuracy: 1.0
Confusion Matrix:
 [[50  0  0]
 [ 0 47  3]
 [ 0  1 49]]

Classification Report:
               precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        50
  Versicolor       0.98      0.94      0.96        50
   Virginica       0.94      0.98      0.96        50

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

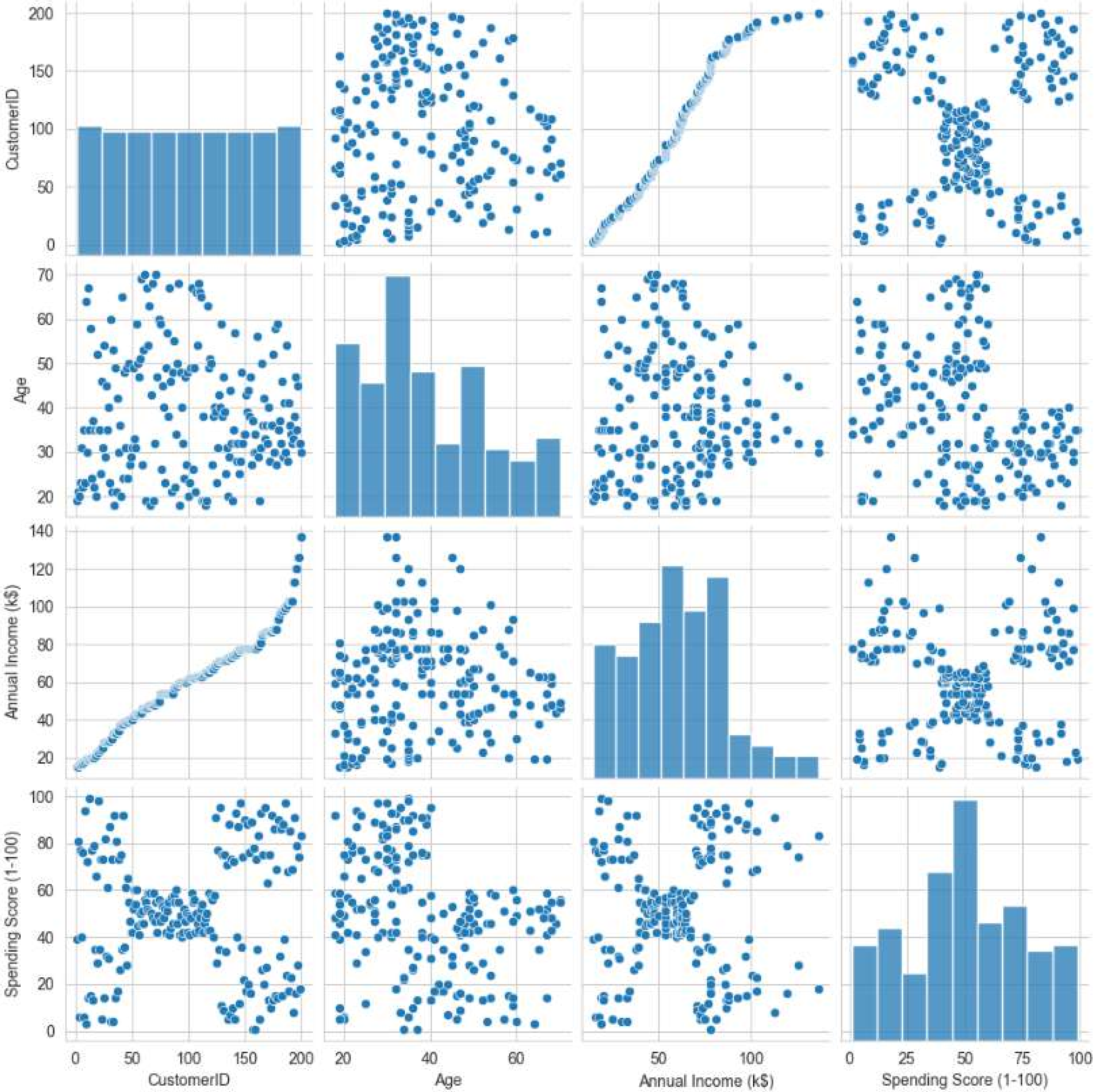
[5]:
```python
import numpy as np
import pandas as pd

df = pd.read_csv(r"C:\Users\A R KRISHNA\OneDrive\Documents\Iris.csv")
df.info()
df.variety.value_counts()
df.head()
features=df.iloc[:,:-1].values
label=df.iloc[:,4].values
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
xtrain,xtest,ytrain,ytest=train_test_split(features,label,test_size=.2,random_state=52)
model_KNN=KNeighborsClassifier(n_neighbors=5)
model_KNN.fit(xtrain,ytrain)
print(model_KNN.score(xtrain,ytrain))
print(model_KNN.score(xtest,ytest))
from sklearn.metrics import confusion_matrix
confusion_matrix(label,model_KNN.predict(features))
from sklearn.metrics import classification_report
print(classification_report(label,model_KNN.predict(features)))
```

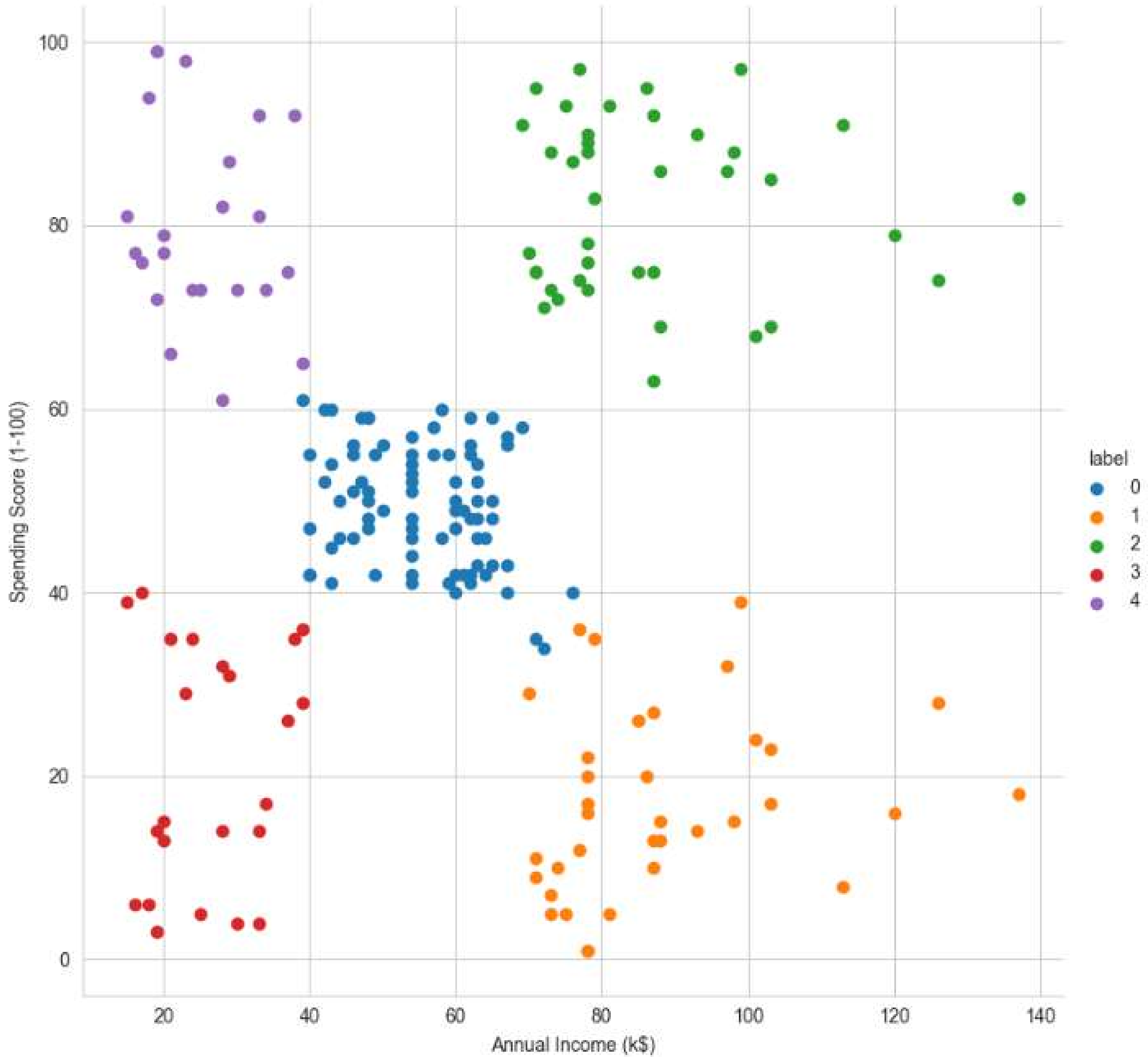
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
0.975
0.9666666666666667
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        50
  Versicolor       0.98      0.94      0.96        50
   Virginica       0.94      0.98      0.96        50

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

[11]:

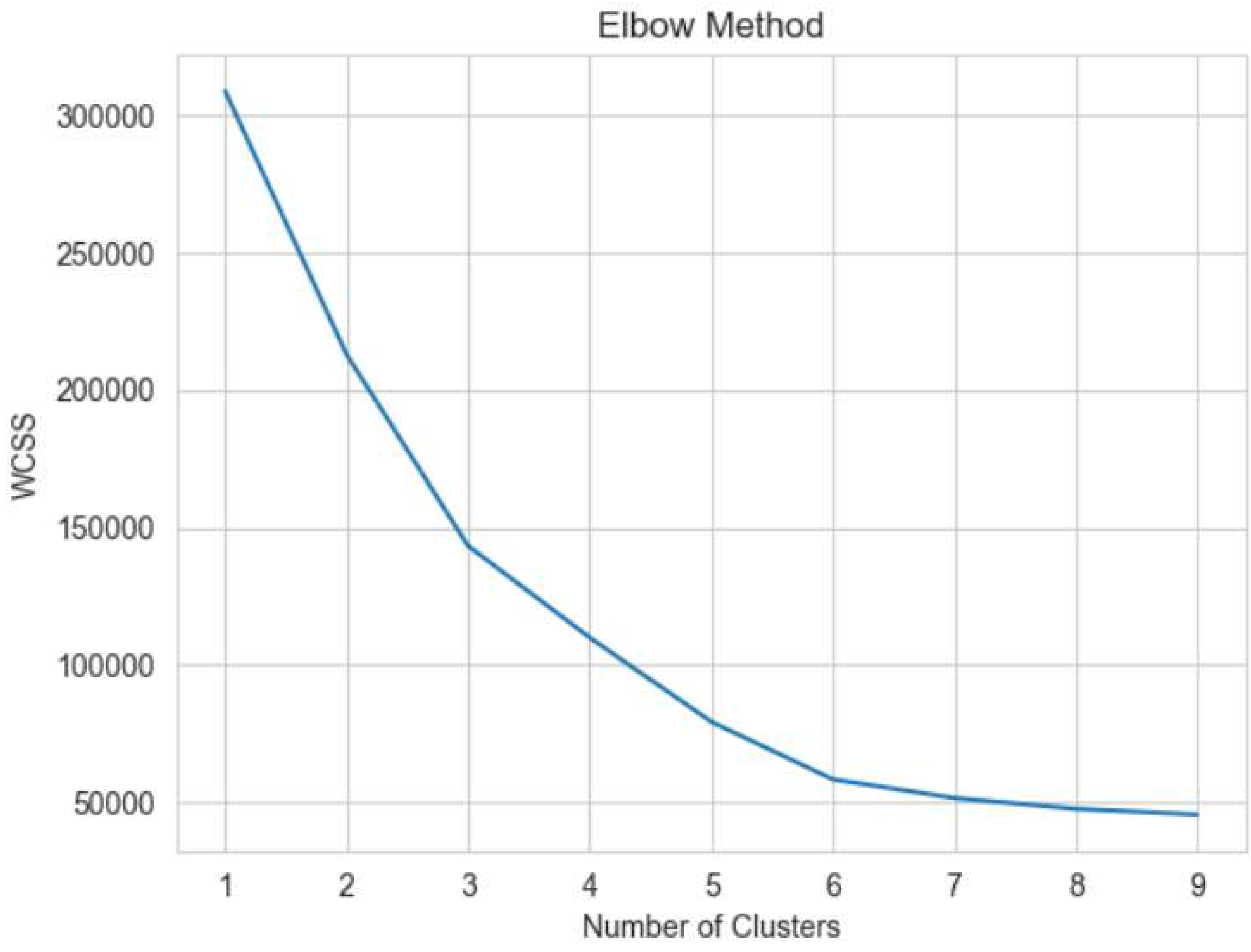
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv(r"C:\Users\A R KRISHNA\Downloads\Mall_Customers.csv")
df.info()
df.head()
sns.pairplot(df)
features = df.iloc[:, [3, 4]].values
from sklearn.cluster import KMeans
model = KMeans(n_clusters=5)
model.fit(features)
Final = df.iloc[:, [3, 4]].copy()
Final.columns = ['Annual Income (k$)', 'Spending Score (1-100)']
Final['label'] = model.predict(features)
sns.set_style("whitegrid")
sns.FacetGrid(Final, hue="label", height=8) \
    .map(plt.scatter, "Annual Income (k$)", "Spending Score (1-100)") \
    .add_legend()
plt.show()
features_el = df.iloc[:, [2, 3, 4]].values
wcss = []
for i in range(1, 10):
    model = KMeans(n_clusters=i)
    model.fit(features_el)
    wcss.append(model.inertia_)

plt.plot(range(1, 10), wcss)
plt.title("Elbow Method")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64 
 3   Annual Income (k$)      200 non-null    int64 
 4   Spending Score (1-100)  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```


CustomerID
Age
Annual Income (k$)
Spending Score (1-100)
CustomerID
Age
Annual Income (k$)
Spending Score (1-100)



Spending Score (1-100)
Annual Income (k$)
label
0
1
2
3
4
0
20
40
60
80
100
20
40
60
80
100
120
140


Elbow Method
WCSS
300000
250000
200000
150000
100000
50000
1
2
3
4
5
6
7
8
9
Number of Clusters

[14]:

```python
import numpy as np
from scipy import stats
marks=np.array([72, 68, 75, 70, 74, 69, 71, 73, 70, 72])
mu_0 = 70
t_stat, p_value = stats.ttest_1samp(marks, mu_0)
print(f"T-statistic: {t_stat:.3f}")
print(f"P-value: {p_value:.4f}")
alpha = 0.05
if p_value<alpha:
    print("Reject Null Hypothesis → Mean is significantly different from 70.")
else:
    print("Fail to Reject Null Hypothesis → No significant difference.")
```

```
T-statistic: 1.993
P-value: 0.0774
Fail to Reject Null Hypothesis → No significant difference.
```

[15]:

```
import numpy as np
from math import sqrt
from scipy.stats import norm

x_bar = 51.2
mu_0 = 50
sigma = 3
n = 36
z_stat = (x_bar - mu_0) / (sigma / sqrt(n))
p_value = 2 * (1 - norm.cdf(abs(z_stat)))
print(f"Z-statistic: {z_stat:.3f}")
print(f"P-value: {p_value:.4f}")
alpha = 0.05
if p_value<alpha:
    print("Reject Null Hypothesis → Mean is significantly different from 50 g.")
else:
    print("Fail to Reject Null Hypothesis → No significant difference.")
```

```
Z-statistic: 2.400
P-value: 0.0164
Reject Null Hypothesis → Mean is significantly different from 50 g.
```

[16]:

```
import numpy as np
from scipy import stats

A = [20, 22, 23]
B = [19, 20, 18]
C = [25, 27, 26]
f_stat, p_value = stats.f_oneway(A, B, C)
print(f"F-statistic: {f_stat:.3f}")
print(f"P-value: {p_value:.4f}")
alpha = 0.05
if p_value<alpha:
    print("Reject Null Hypothesis → Means are significantly different.")
else:
    print("Fail to Reject Null Hypothesis → No significant difference.")
```

```
F-statistic: 25.923
P-value: 0.0011
Reject Null Hypothesis → Means are significantly different.
```