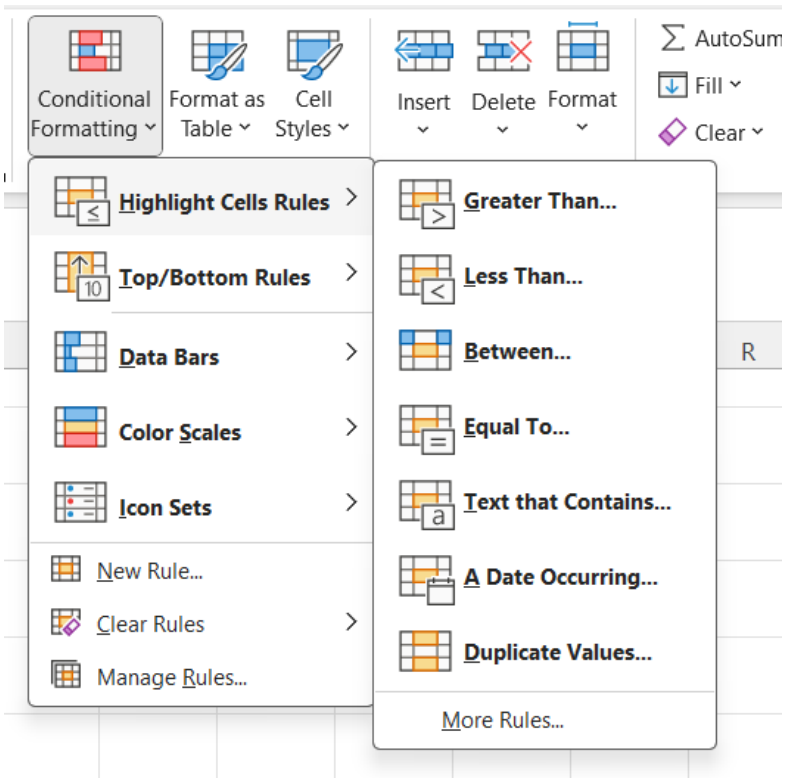
Practical No: 1(a)

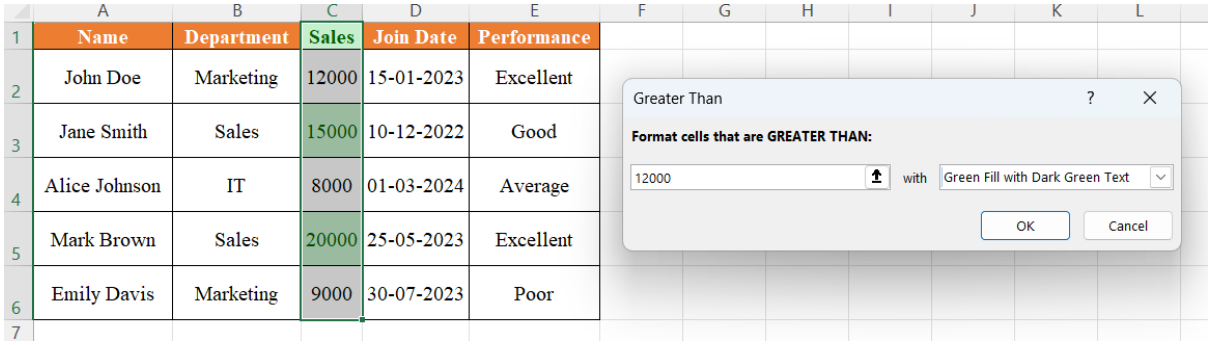
Aim: Perform conditional formatting on a dataset using various criteria.

Dataset:

Name	Department	Sales	Join Date	Performance
John Doe	Marketing	12000	15-01-2023	Excellent
Jane Smith	Sales	15000	10-12-2022	Good
Alice Johnson	IT	8000	01-03-2024	Average
Mark Brown	Sales	20000	25-05-2023	Excellent
Emily Davis	Marketing	9000	30-07-2023	Poor

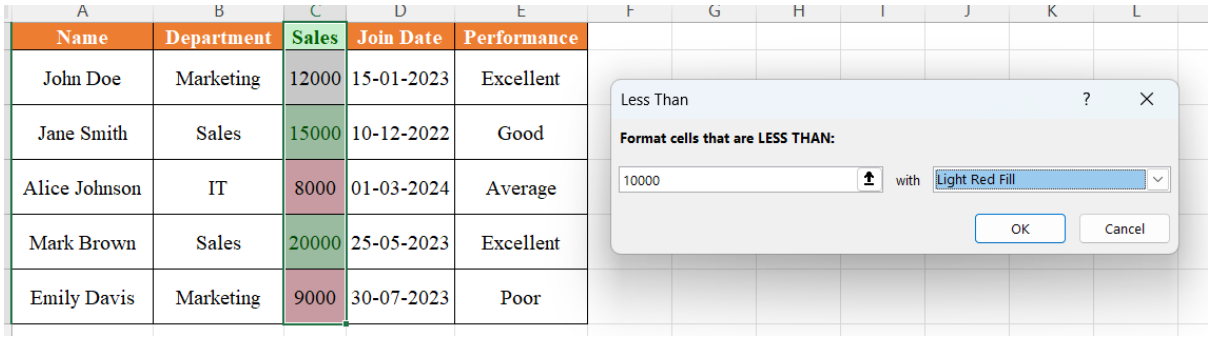
Step 1: Open Excel Go in Home tab and then go in Conditional Formatting Select Highlight Cells Rules and Select Greater Than Option in Options.





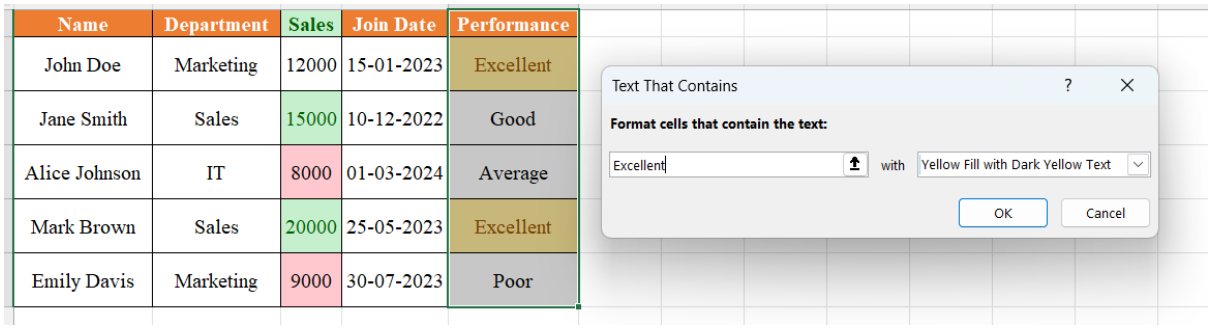
Name	Department	Sales	Join Date	Performance
John Doe	Marketing	12000	15-01-2023	Excellent
Jane Smith	Sales	15000	10-12-2022	Good
Alice Johnson	IT	8000	01-03-2024	Average
Mark Brown	Sales	20000	25-05-2023	Excellent
Emily Davis	Marketing	9000	30-07-2023	Poor

Then Select Lesser Than Options



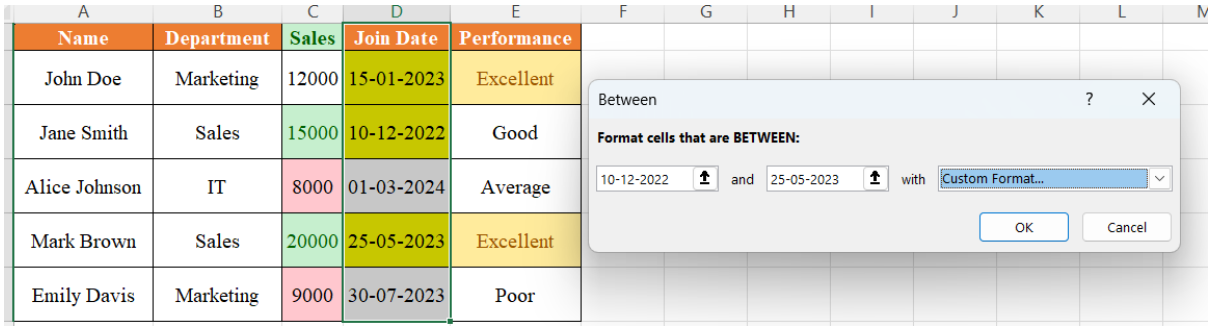
Name	Department	Sales	Join Date	Performance
John Doe	Marketing	12000	15-01-2023	Excellent
Jane Smith	Sales	15000	10-12-2022	Good
Alice Johnson	IT	8000	01-03-2024	Average
Mark Brown	Sales	20000	25-05-2023	Excellent
Emily Davis	Marketing	9000	30-07-2023	Poor

Then Select Format Cells that Contain the text Options



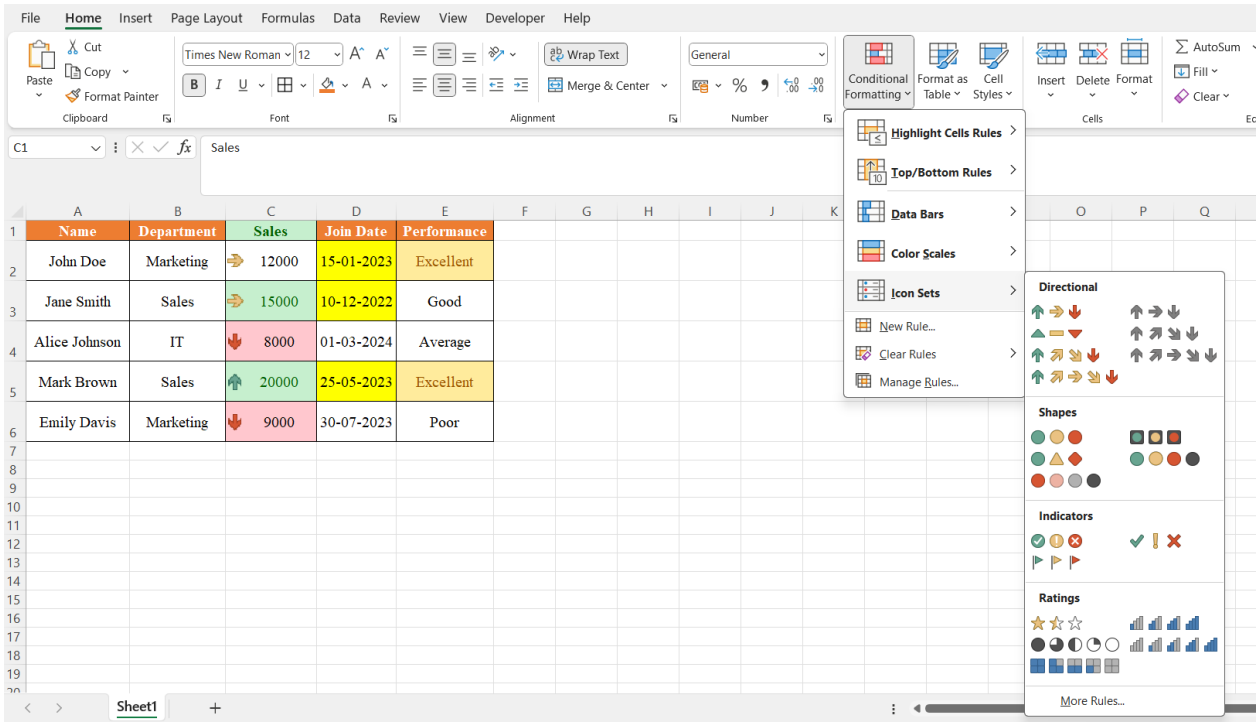
Name	Department	Sales	Join Date	Performance
John Doe	Marketing	12000	15-01-2023	Excellent
Jane Smith	Sales	15000	10-12-2022	Good
Alice Johnson	IT	8000	01-03-2024	Average
Mark Brown	Sales	20000	25-05-2023	Excellent
Emily Davis	Marketing	9000	30-07-2023	Poor

Select Between Options



Name	Department	Sales	Join Date	Performance
John Doe	Marketing	12000	15-01-2023	Excellent
Jane Smith	Sales	15000	10-12-2022	Good
Alice Johnson	IT	8000	01-03-2024	Average
Mark Brown	Sales	20000	25-05-2023	Excellent
Emily Davis	Marketing	9000	30-07-2023	Poor

Use Icon Set


File Home Insert Page Layout Formulas Data Review View Developer Help
Paste Cut Copy Format Painter
Times New Roman 12
Wrap Text
Merge & Center
General
Conditional Formatting
Format as Table
Cell Styles
Insert Delete Format
AutoSum Fill Clear
Clipboard Font Alignment Number Cells
C1
Sales
Highlight Cells Rules
Top/Bottom Rules
Data Bars
Color Scales
Icon Sets
New Rule...
Clear Rules
Manage Rules...
Directional
Shapes
Indicators
Ratings
More Rules...
Name Department Sales Join Date Performance
John Doe Marketing 12000 15-01-2023 Excellent
Jane Smith Sales 15000 10-12-2022 Good
Alice Johnson IT 8000 01-03-2024 Average
Mark Brown Sales 20000 25-05-2023 Excellent
Emily Davis Marketing 9000 30-07-2023 Poor
Sheet1

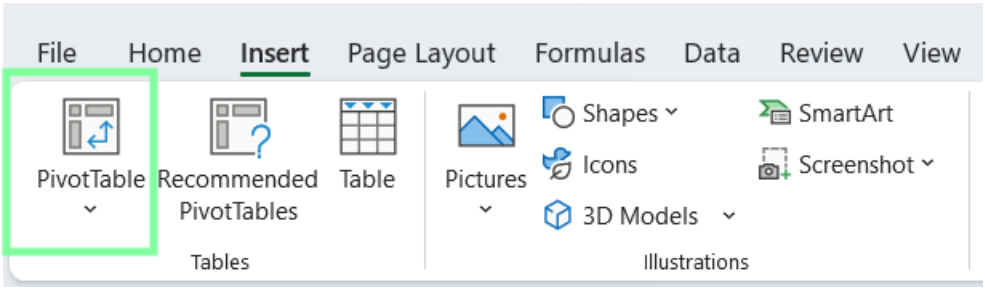
Practical No: 1(b)

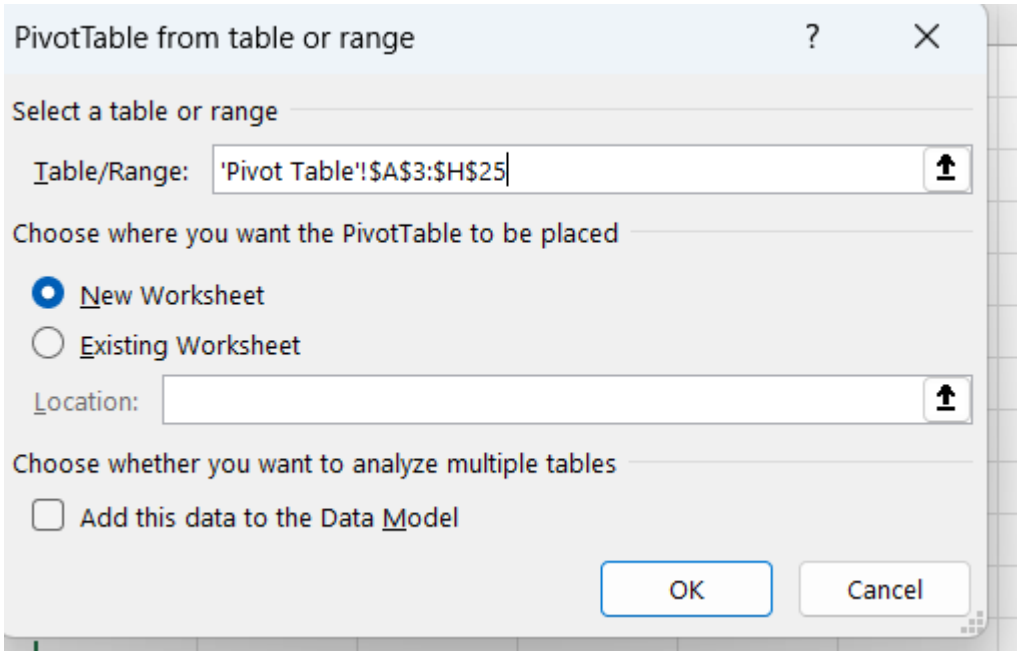
Aim: Create a pivot table to analyze and summarize data.

Dataset:

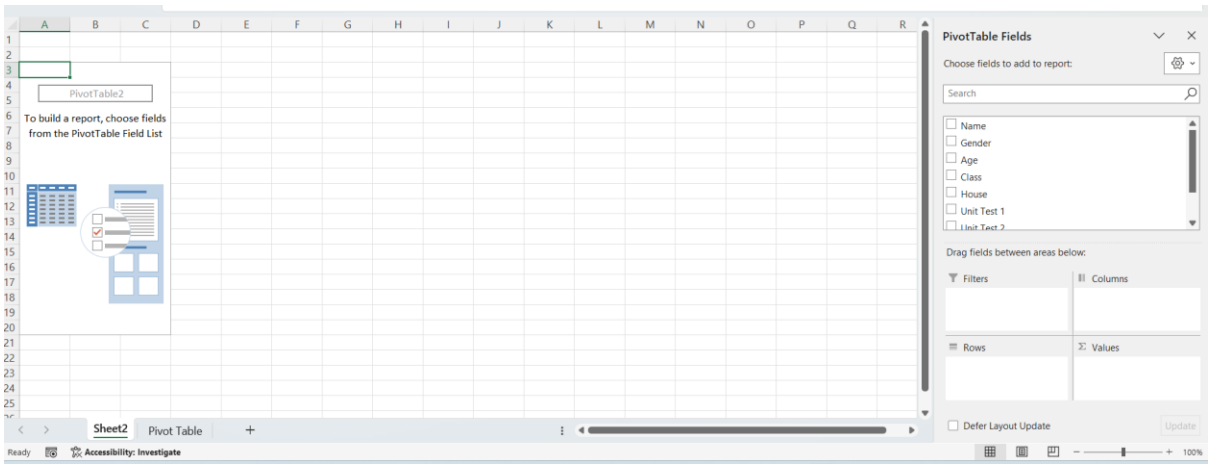
Pivot Table							
Name	Gender	Age	Class	House	Unit Test 1	Unit Test 2	Final Test
Abhimanyu	M	16	10	Bhoomi	84	79	81
Arjun	M	11	5	Vayu	82	83	91
Champa	F	15	8	Jal	81	78	88
Gopal	M	14	8	Bhoomi	70	75	79
Gopi	F	16	10	Agni	88	92	96
Hari	M	16	10	Bhoomi	82	81	80
Indu	F	14	8	Vayu	90	86	89
Keshav	M	15	9	Agni	87	89	96
Lalita	F	17	10	Vayu	70	90	92
Madhav	M	12	7	Jal	86	92	89
Sam	M	11	6	Agni	91	81	94
RNM	M	16	10	Agni	86	81	77
Student1	M	15	9	Agni	87	89	95
Student8	F	15	8	Vayu	81	90	95
Student2	F	17	10	Vayu	70	90	92
Student4	F	12	7	Jal	86	92	89
Student5	F	16	10	Jal	81	80	87
Sudevi	F	16	10	Jal	81	80	87
Varun	M	15	9	Vayu	87	89	95
Vidya	F	11	6	Vayu	88	90	92
Visakha	F	16	10	Bhoomi	70	87	85
Vrinda	F	14	8	Agni	91	96	98

Go in Insert tab and Select Pivot Table Option.

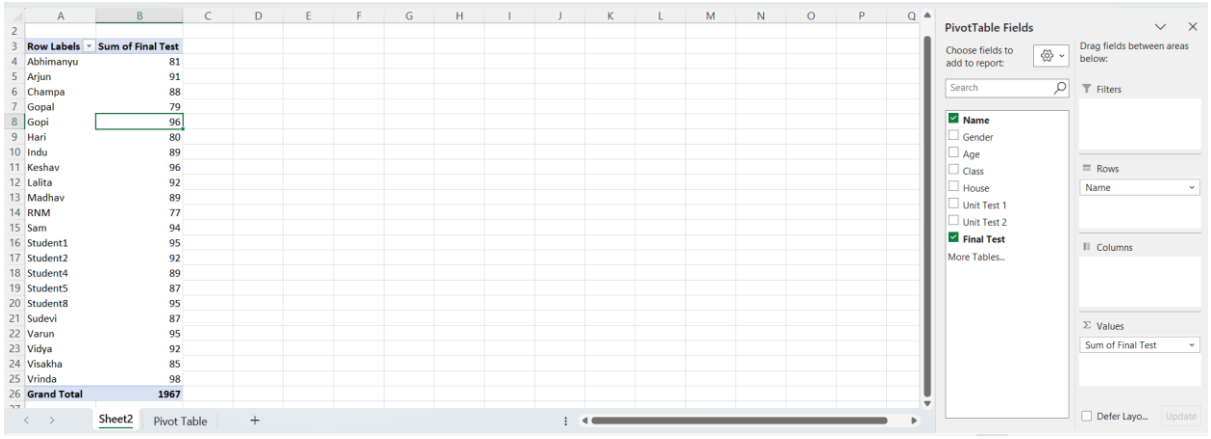




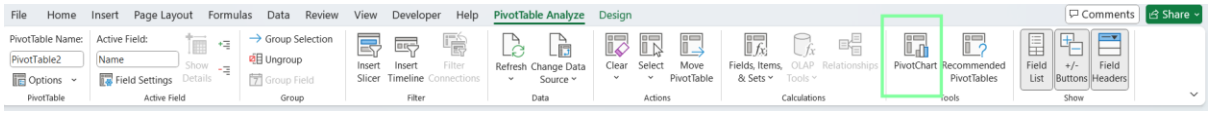
Note: Then we will get this type of Interface by Default.



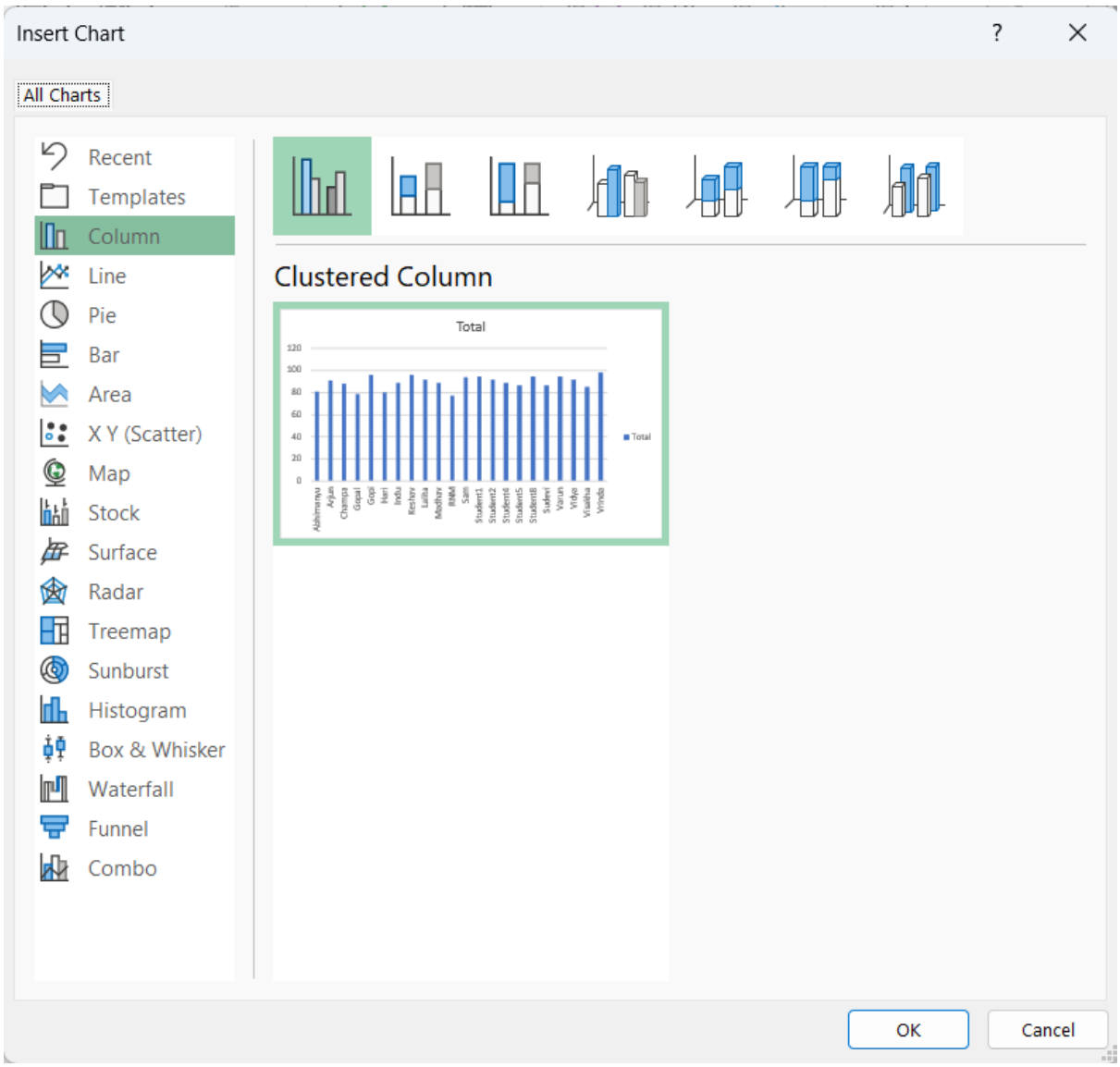
After that, Put Name Column in rows & Final Test in Values.

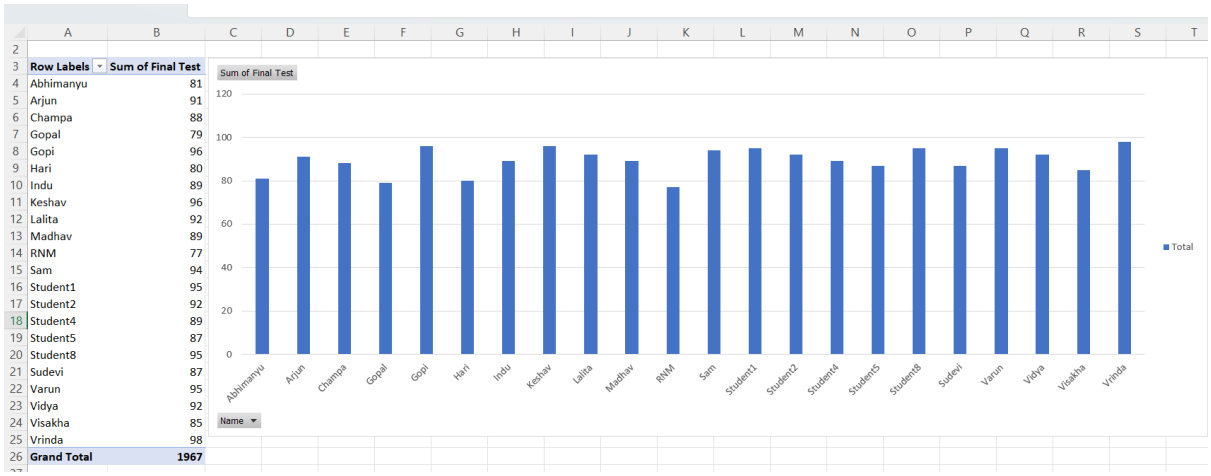


Then Go in PivotTable Analyze and Select Pivot Chart Option.



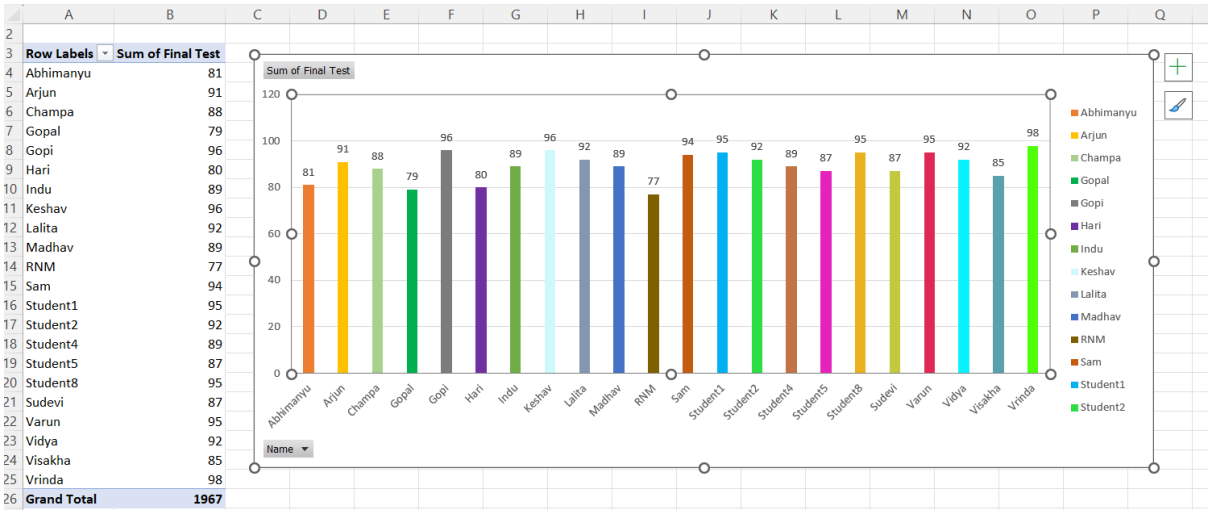
Select Column Chart & Use it.





Row Labels	Sum of Final Test
Abhimanyu	81
Arjun	91
Champa	88
Gopal	79
Gopi	96
Hari	80
Indu	89
Keshav	96
Lalita	92
Madhav	89
RNM	77
Sam	94
Student1	95
Student2	92
Student4	89
Student5	87
Student8	95
Sudevi	87
Varun	95
Vidya	92
Visakha	85
Vrinda	98
Grand Total	**1967**

Output:



Row Labels	Sum of Final Test
Abhimanyu	81
Arjun	91
Champa	88
Gopal	79
Gopi	96
Hari	80
Indu	89
Keshav	96
Lalita	92
Madhav	89
RNM	77
Sam	94
Student1	95
Student2	92
Student4	89
Student5	87
Student8	95
Sudevi	87
Varun	95
Vidya	92
Visakha	85
Vrinda	98
Grand Total	**1967**

Practical No: 1(c)

Aim: Use VLOOKUP function to retrieve information from a different worksheet or table.

Use VLOOKUP in Excel to match the Department Name from the second table with the Department ID in the first table.

Create a new column in the Employee Details table called Department Name, and use VLOOKUP to populate it.

Employee ID	Name	Age	Department ID	Department Name
E001	Shivam Chaubey	30	D001	
E002	Nikhil Shah	25	D002	
E003	Ashutosh Mishra	35	D003	
E004	Vikas Maurya	28	D004	

Department ID	Department Name
D001	Human Resources
D002	Finance
D003	IT
D004	Marketing

Formula of VLookup:

The VLOOKUP formula in Excel is used to look up values in a table:

=VLOOKUP(lookup_value, table_array, column_index_number, [range_lookup])

Employee ID	Name	Age	Department ID	Department Name
E001	Shivam Chaubey	30	D001	=VLOOKUP(D2,G2:H5,2,0)
E002	Nikhil Shah	25	D002	
E003	Ashutosh Mishra	35	D003	
E004	Vikas Maurya	28	D004	

Department ID	Department Name
D001	Human Resources
D002	Finance
D003	IT
D004	Marketing

Employee ID	Name	Age	Department ID	Department Name
E001	Shivam Chaubey	30	D001	Human Resources
E002	Nikhil Shah	25	D002	
E003	Ashutosh Mishra	35	D003	
E004	Vikas Maurya	28	D004	

Department ID	Department Name
D001	Human Resources
D002	Finance
D003	IT
D004	Marketing

Output:

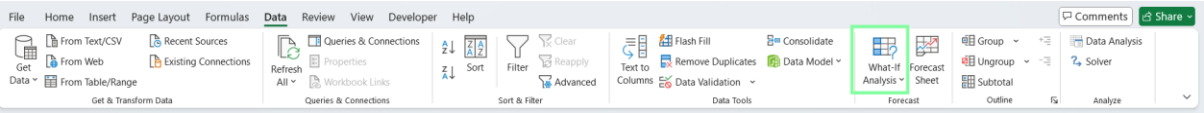
Employee ID	Name	Age	Department ID	Department Name
E001	Shivam Chaubey	30	D001	Human Resources
E002	Nikhil Shah	25	D002	Finance
E003	Ashutosh Mishra	35	D003	IT
E004	Vikas Maurya	28	D004	Marketing

Department ID	Department Name
D001	Human Resources
D002	Finance
D003	IT
D004	Marketing

Practical No: 1(d)

Aim: Perform what-if analysis using Goal Seek to determine input values for desired output.

Step 1: Go in Data tab in Excel and Select **What-if Analysis.**



What-if Analysis

Total Revenue = Price*Qty

Transport Cost = 10% of Total Revenue

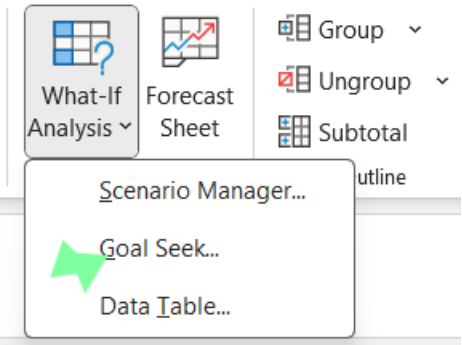
Item Cost= 20 rs of Qty

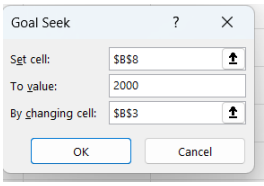
Total Cost = Transport cost + Item Cost

Profit = Total Revenue – Total Cost

What if Analysis	
Price	$ 32.00
Qty	227
Total Revenue	$ 7,272.73
Transport Cost	$ 727.27
Item Cost	$ 4,545.45
Total Cost	$ 5,272.73
Profit	$2,000.00

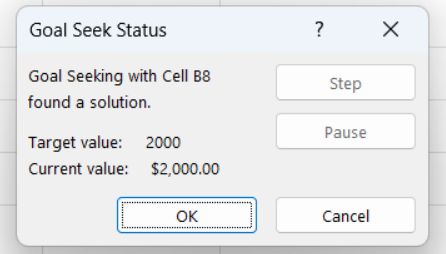
Click on Goal Seek





Output:

	A	B	C	D	E
1	What if Analysis				
2	Price	$ 32.00			
3	Qty	227			
4	Total Revenue	$ 7,272.73			
5	Transport Cost	$ 727.27			
6	Item Cost	$ 4,545.45			
7	Total Cost	$ 5,272.73			
8	Profit	$ 2,000.00			



Practical No: 2(a)

Aim: Read data from CSV and JSON files into a data frame.

Code:

```
import pandas as pd
csv_file_path = "sample_data.csv"
json_file_path = "sample_data.json"
csv_df = pd.read_csv(csv_file_path)
print("CSV DataFrame:")
print(csv_df)
json_df = pd.read_json(json_file_path)
print("\nJSON DataFrame:")
print(json_df)
combined_df = pd.concat([csv_df, json_df], ignore_index=True)
print("\nCombined DataFrame:")
print(combined_df)
```

Output:

```
CSV DataFrame:
      Name  Age Department
0    Alice   30         HR
1      Bob   25         IT
2  Charlie   35    Finance

JSON DataFrame:
    Name  Age Department
0  David   40  Marketing
1    Eve   28         IT
2  Frank   33    Finance

Combined DataFrame:
      Name  Age Department
0    Alice   30         HR
1      Bob   25         IT
2  Charlie   35    Finance
3    David   40  Marketing
4      Eve   28         IT
5    Frank   33    Finance
PS D:\TYBSC_CS_Data_Science_Practicals>
```

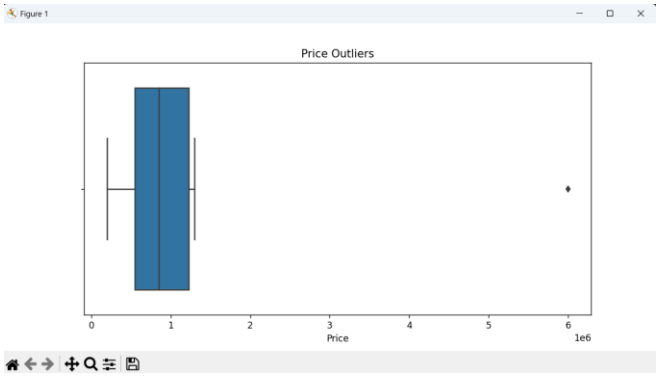
Practical No: 2(b)

Aim: Perform basic data pre-processing tasks such as handling missing values and outliers.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
file_path = "Housing_Prices.csv"
data = pd.read_csv(file_path)
print("Initial Data:")
print(data.head())
print("\nMissing Values:\n", data.isnull().sum())
data['Price'] = data['Price'].fillna(data['Price'].mean())
data['Location'] = data['Location'].fillna(data['Location'].mode()[0])
print("\nAfter Handling Missing Values:")
print(data.isnull().sum())
plt.figure(figsize=(10, 5))
sns.boxplot(x=data['Price'])
plt.title("Price Outliers")
plt.show()
Q1 = data['Price'].quantile(0.25)
Q3 = data['Price'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
data = data[(data['Price'] >= lower_bound) & (data['Price'] <= upper_bound)]
print("\nData After Removing Outliers:")
print(data.describe())
cleaned_file_path = "Cleaned_Housing_Prices.csv"
data.to_csv(cleaned_file_path, index=False)
print(f"\nCleaned dataset saved to: {cleaned_file_path}")
```

Output:




```
Initial Data:
   House_ID Location  Size_in_sqft      Price
0         1        A          1000   200000.0
1         2        B          1500   300000.0
2         3      NaN          2000        NaN
3         4        D          2500   500000.0
4         5        E          3000  6000000.0

Missing Values:
 House_ID        0
Location         2
Size_in_sqft     0
Price            2
dtype: int64

After Handling Missing Values:
House_ID         0
Location         0
Size_in_sqft     0
Price            0
dtype: int64
```

```
Data After Removing Outliers:
        House_ID  Size_in_sqft         Price
count   9.000000      9.000000  9.000000e+00
mean    5.555556   3277.777778  7.777778e+05
std     3.205897   1602.948672  3.961621e+05
min     1.000000   1000.000000  2.000000e+05
25%     3.000000   2000.000000  5.000000e+05
50%     6.000000   3500.000000  8.000000e+05
75%     8.000000   4500.000000  1.000000e+06
max    10.000000   5500.000000  1.300000e+06

Cleaned dataset saved to: Cleaned_Housing_Prices.csv
```

Practical No: 2(c)

Aim: Manipulate and transform data using functions like filtering, sorting, and grouping.

Code:

```
import pandas as pd
from datetime import datetime
file_path = "sample_dataset.csv"
df = pd.read_csv(file_path)
df["Joining Date"] = pd.to_datetime(df["Joining Date"])
print("Original Dataset:")
print(df)
hr_employees = df[df["Department"] == "HR"]
print("\nEmployees in the HR Department:")
print(hr_employees)
high_salary = df[df["Salary"] > 55000]
print("\nEmployees with Salary Greater than 55000:")
print(high_salary)
sorted_salary = df.sort_values(by="Salary")
print("\nEmployees Sorted by Salary (Ascending):")
print(sorted_salary)
sorted_date = df.sort_values(by="Joining Date", ascending=False)
print("\nEmployees Sorted by Joining Date (Descending):")
print(sorted_date)
grouped_salary = df.groupby("Department")["Salary"].mean().reset_index()
print("\nAverage Salary by Department:")
print(grouped_salary)
employee_count = df.groupby("Department")["ID"].count().reset_index()
employee_count.rename(columns={"ID": "Employee Count"}, inplace=True)
print("\nNumber of Employees in Each Department:")
print(employee_count)
filtered_sorted = df[df["Department"] == "HR"].sort_values(by="Salary", ascending=False)
print("\nHR Employees Sorted by Salary (Descending):")
print(filtered_sorted)
df["Years of Service"] = datetime.now().year - df["Joining Date"].dt.year
```

```
print("\nDataset with Years of Service Added:")
print(df)
pivot_table = df.pivot_table(values="Salary", index="Department",
aggfunc="mean").reset_index()
print("\nPivot Table - Average Salary by Department:")
print(pivot_table)
```

Output:

```
Original Dataset:
   ID     Name Department  Salary Joining Date
0   1    Alice         HR   50000   2020-01-15
1   2      Bob    Finance   60000   2021-07-10
2   3  Charlie         IT   70000   2020-05-22
3   4    David  Marketing   55000   2019-11-05
4   5      Eve         HR   50000   2021-03-18

Employees in the HR Department:
   ID   Name Department  Salary Joining Date
0   1  Alice         HR   50000   2020-01-15
4   5    Eve         HR   50000   2021-03-18

Employees with Salary Greater than 55000:
   ID     Name Department  Salary Joining Date
1   2      Bob    Finance   60000   2021-07-10
2   3  Charlie         IT   70000   2020-05-22

Employees Sorted by Salary (Ascending):
   ID     Name Department  Salary Joining Date
0   1    Alice         HR   50000   2020-01-15
4   5      Eve         HR   50000   2021-03-18
3   4    David  Marketing   55000   2019-11-05
1   2      Bob    Finance   60000   2021-07-10
2   3  Charlie         IT   70000   2020-05-22
```

```
Employees Sorted by Joining Date (Descending):
   ID     Name Department  Salary Joining Date
1   2      Bob    Finance   60000   2021-07-10
4   5      Eve         HR   50000   2021-03-18
2   3  Charlie         IT   70000   2020-05-22
0   1    Alice         HR   50000   2020-01-15
3   4    David  Marketing   55000   2019-11-05

Average Salary by Department:
  Department   Salary
0    Finance  60000.0
1         HR  50000.0
2         IT  70000.0
3  Marketing  55000.0

Number of Employees in Each Department:
  Department  Employee Count
0    Finance               1
1         HR               2
2         IT               1
3  Marketing               1
```

```
HR Employees Sorted by Salary (Descending):
   ID   Name Department  Salary Joining Date
0   1  Alice         HR   50000   2020-01-15
4   5    Eve         HR   50000   2021-03-18

Dataset with Years of Service Added:
   ID     Name Department  Salary Joining Date  Years of Service
0   1    Alice         HR   50000   2020-01-15                 4
1   2      Bob    Finance   60000   2021-07-10                 3
2   3  Charlie         IT   70000   2020-05-22                 4
3   4    David  Marketing   55000   2019-11-05                 5
4   5      Eve         HR   50000   2021-03-18                 3

Pivot Table - Average Salary by Department:
  Department  Salary
0    Finance   60000
1         HR   50000
2         IT   70000
3  Marketing   55000
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical No: 3(a)

Aim: Apply feature-scaling techniques like standardization and normalization to numerical features.

Code:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
data = {
    'Feature1': [10, 20, 30, 40, 50],
    'Feature2': [100, 150, 200, 250, 300],
    'Feature3': [1, 2, 3, 4, 5]
}
df = pd.DataFrame(data)
print("Original Data:")
print(df)
scaler_standard = StandardScaler()
df_standardized = pd.DataFrame(scaler_standard.fit_transform(df), columns=df.columns)
print("\nStandardized Data:")
print(df_standardized)
scaler_minmax = MinMaxScaler()
df_normalized = pd.DataFrame(scaler_minmax.fit_transform(df), columns=df.columns)
print("\nNormalized Data:")
print(df_normalized)
```

Output:

```
Original Data:
   Feature1  Feature2  Feature3
0        10       100         1
1        20       150         2
2        30       200         3
3        40       250         4
4        50       300         5

Standardized Data:
   Feature1  Feature2  Feature3
0 -1.414214 -1.414214 -1.414214
1 -0.707107 -0.707107 -0.707107
2  0.000000  0.000000  0.000000
3  0.707107  0.707107  0.707107
4  1.414214  1.414214  1.414214

Normalized Data:
   Feature1  Feature2  Feature3
0      0.00      0.00      0.00
1      0.25      0.25      0.25
2      0.50      0.50      0.50
3      0.75      0.75      0.75
4      1.00      1.00      1.00
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical: 3(b)

Aim: Perform feature dummification to convert categorical variables into numerical representations.

Code:

```
import pandas as pd
data = {
    'ID': [1, 2, 3, 4, 5],
    'Category': ['A', 'B', 'C', 'A', 'B'],
    'Value': [100, 200, 300, 400, 500]
}
df = pd.DataFrame(data)
print("Original Dataset:")
print(df)
df_dummified = pd.get_dummies(df, columns=['Category'], prefix='Cat', drop_first=False)
print("\nDummified Dataset:")
print(df_dummified)
```

Output:

```
Original Dataset:
   ID Category  Value
0   1        A    100
1   2        B    200
2   3        C    300
3   4        A    400
4   5        B    500

Dummified Dataset:
   ID  Value  Cat_A  Cat_B  Cat_C
0   1    100      1      0      0
1   2    200      0      1      0
2   3    300      0      0      1
3   4    400      1      0      0
4   5    500      0      1      0
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical No: 4(a)

Aim: Formulate null and alternative hypotheses for a given problem.

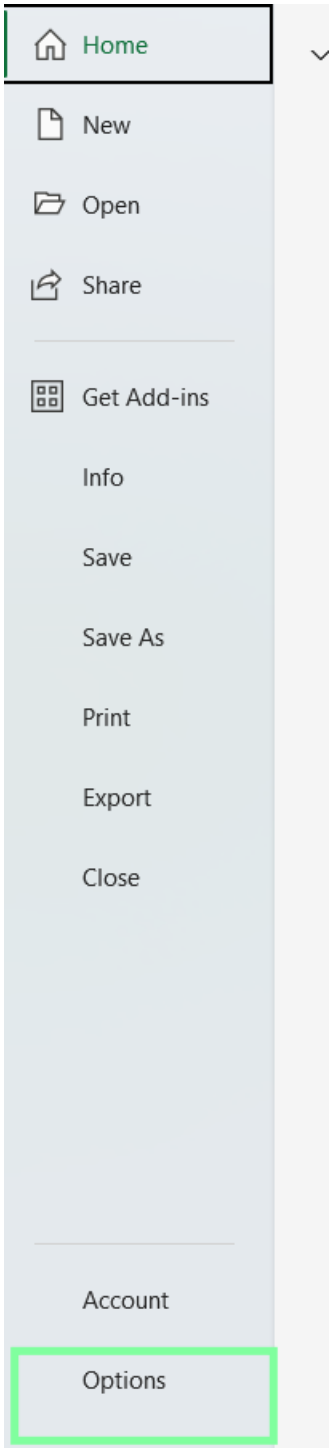
Dataset:

Monthly Sales Report	
Branch 1	Branch 2
₹ 52,862.30	₹ 52,872.96
₹ 54,486.28	₹ 57,356.41
₹ 58,664.41	₹ 44,636.01
₹ 58,430.42	₹ 70,831.49
₹ 38,978.65	₹ 42,945.84
₹ 42,497.40	₹ 41,072.30
₹ 54,120.28	₹ 62,423.00
₹ 54,110.29	₹ 59,124.96
₹ 54,120.38	₹ 57,617.08
₹ 80,821.85	₹ 57,655.11
₹ 54,567.12	₹ 51,889.78
₹ 59,084.53	₹ 43,924.71
₹ 57,632.01	₹ 52,682.24
₹ 55,211.13	₹ 45,905.54
₹ 47,477.85	₹ 60,776.08
₹ 56,071.75	₹ 50,676.48
₹ 43,817.40	₹ 44,570.53
₹ 48,105.45	₹ 49,107.53

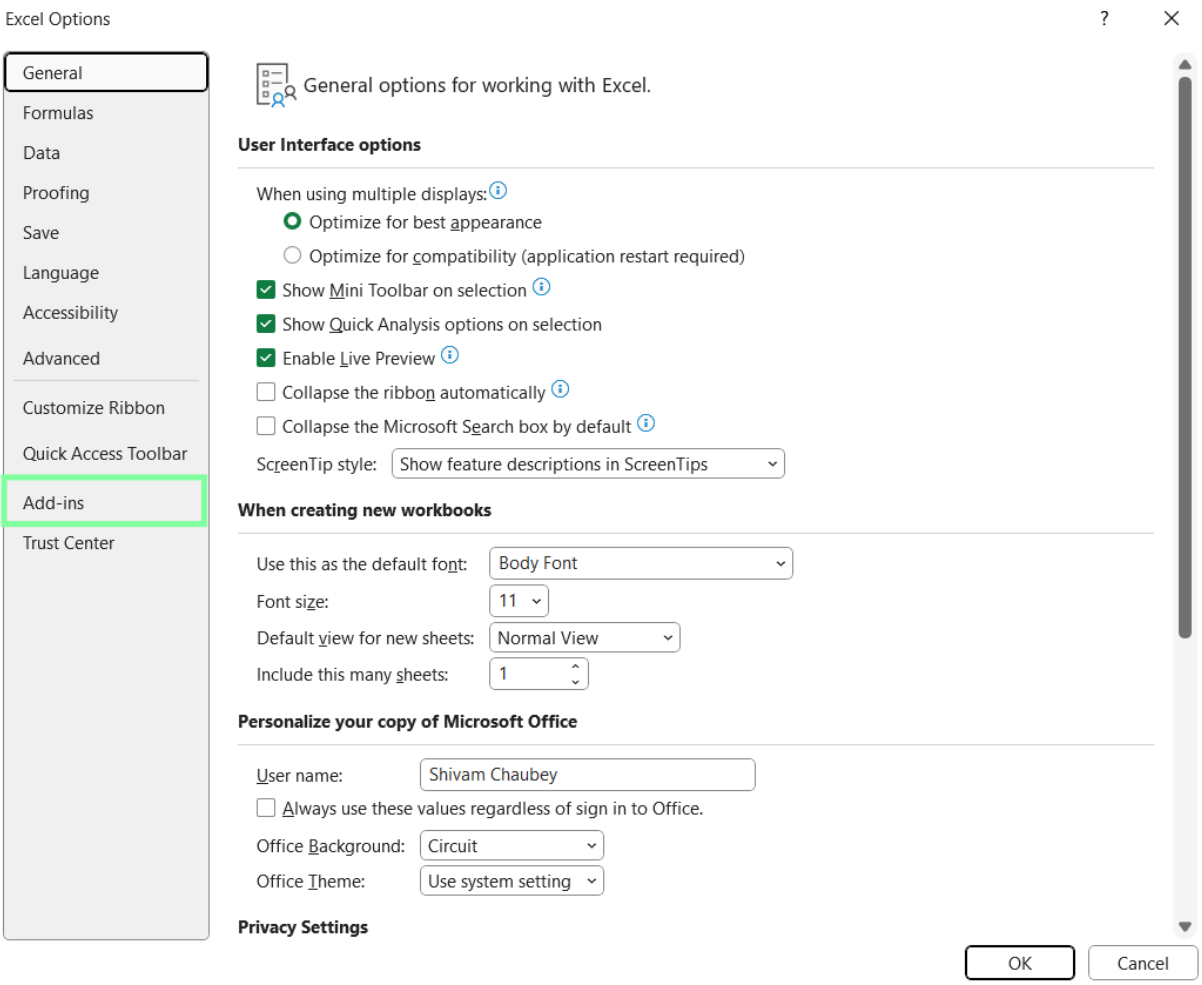
a. Open the Excel file and ensure the dataset is organized.
b. Go to the "Data" tab in Excel.
c. Select "Data Analysis" (if not available, add it via Excel Add-ins).
d. Choose "t-Test: Two-Sample Assuming Equal Variances" or "t-Test: Two-Sample Assuming Unequal Variances," depending on your assumption about variances.

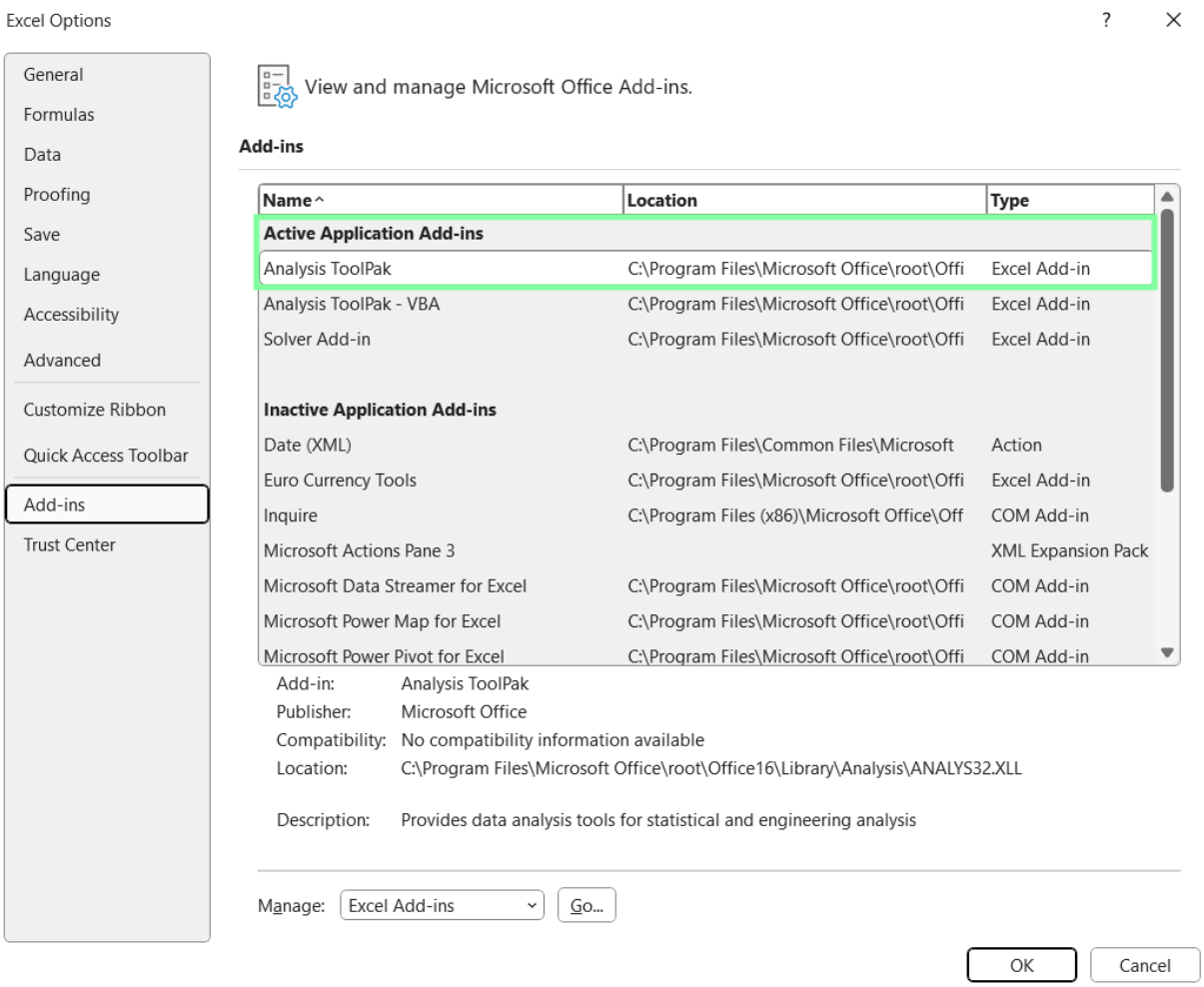
For Enable Data Analysis Tab.

Go in File and Click Options.

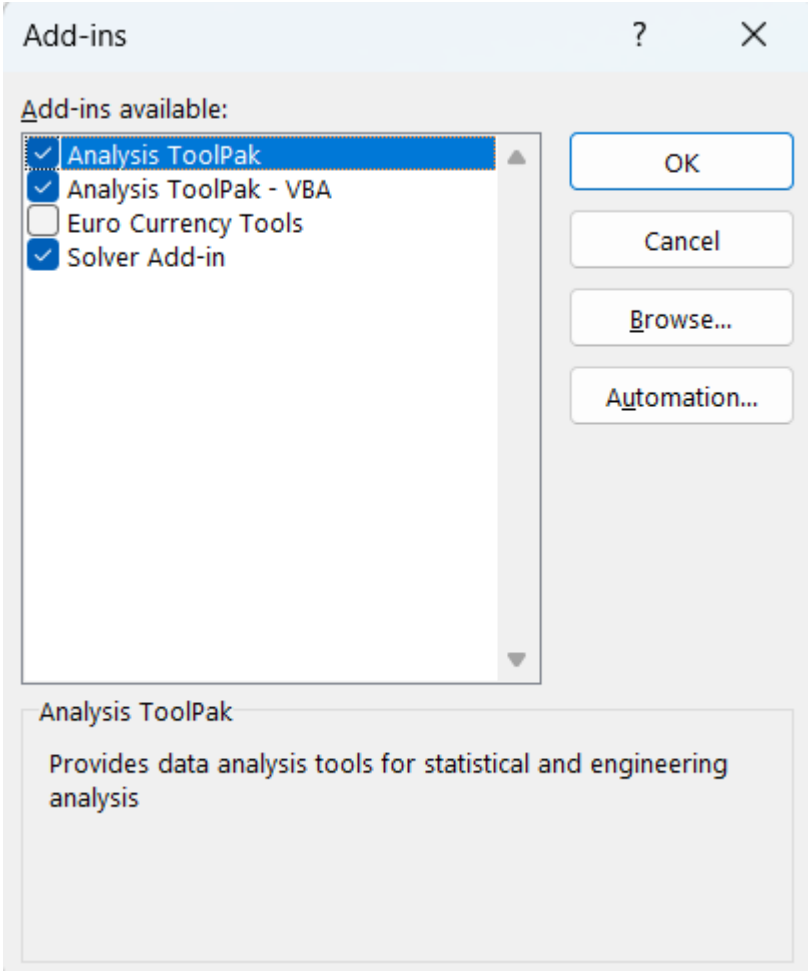


Then it will show another interface Select Add-ins Option then Enable Tool Pack and Choose First Options.


Excel Options
General
Formulas
Data
Proofing
Save
Language
Accessibility
Advanced
Customize Ribbon
Quick Access Toolbar
Add-ins
Trust Center
General options for working with Excel.
User Interface options
When using multiple displays:
Optimize for best appearance
Optimize for compatibility (application restart required)
Show Mini Toolbar on selection
Show Quick Analysis options on selection
Enable Live Preview
Collapse the ribbon automatically
Collapse the Microsoft Search box by default
ScreenTip style: Show feature descriptions in ScreenTips
When creating new workbooks
Use this as the default font: Body Font
Font size: 11
Default view for new sheets: Normal View
Include this many sheets: 1
Personalize your copy of Microsoft Office
User name: Shivam Chaubey
Always use these values regardless of sign in to Office.
Office Background: Circuit
Office Theme: Use system setting
Privacy Settings
OK
Cancel



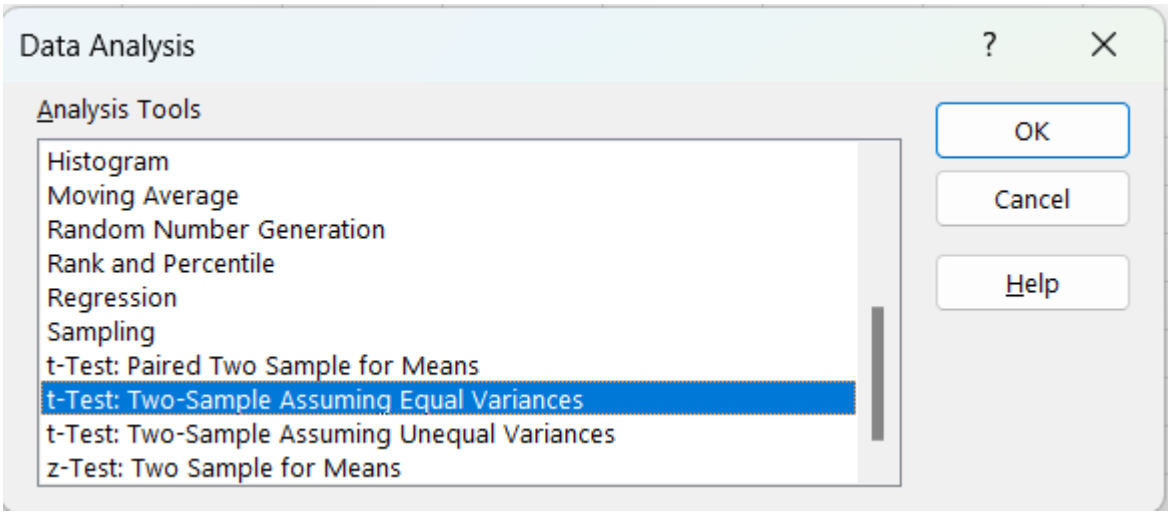
Then Select in Go Option.

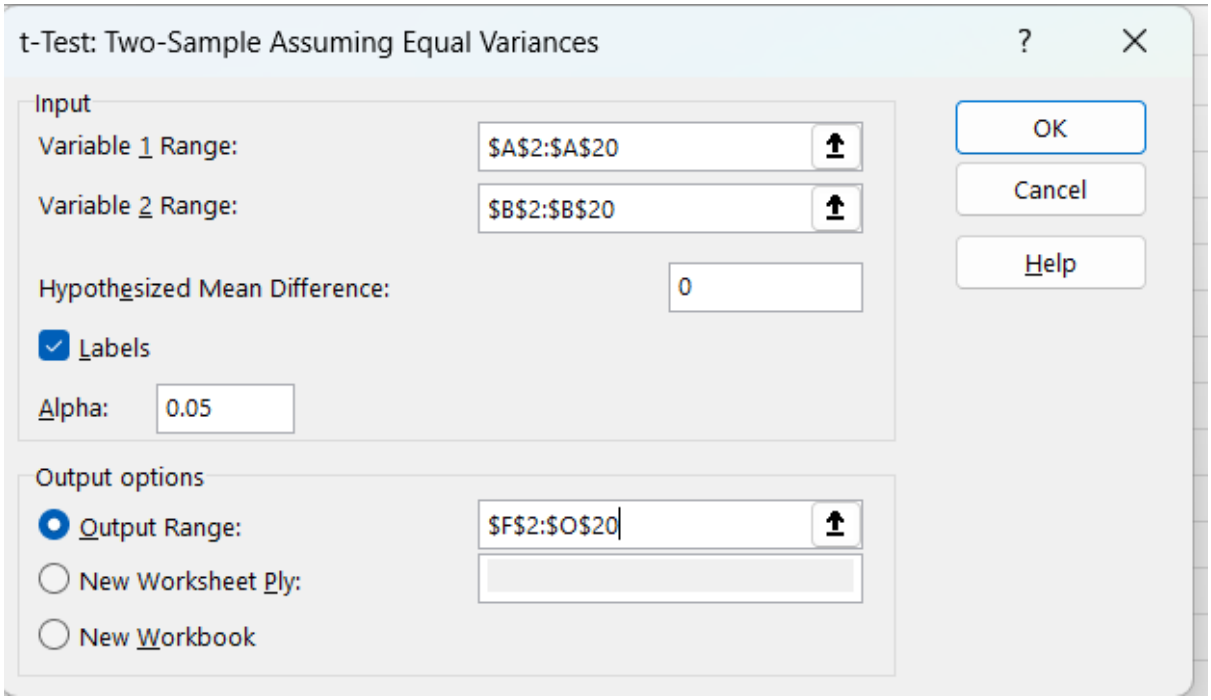


Check The Check Boxes and Click Ok.

Then you will see a “Data” tab in Excel Icon then you get Data Analysis tool Pack.

Then Open Data Analysis and Choose "t-Test: Two-Sample Assuming Equal Variances".





t-Test: Two-Sample Assuming Equal Variances

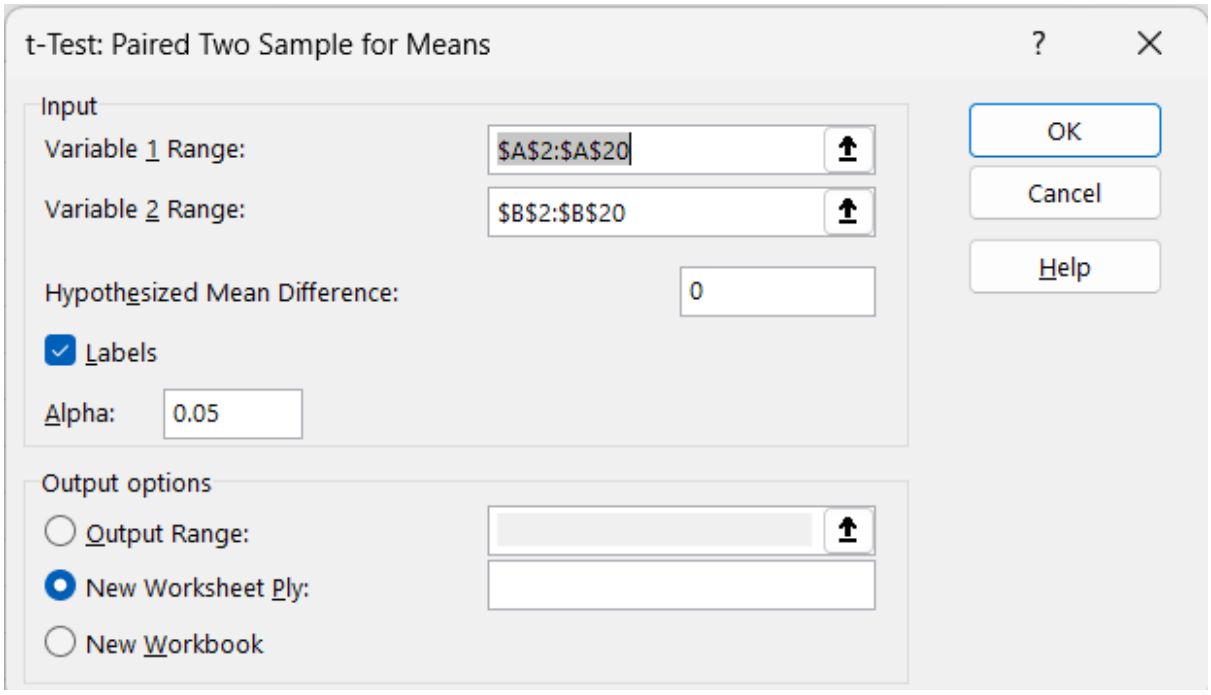
	Branch 1	*Branch 2*
Mean	53947.74955	52559.33675
Variance	79197935.64	64833417.48
Observations	18	18
Pooled Variance	72015676.56	
Hypothesized Mean Difference	0	
df	34	
t Stat	0.490824623	
P(T<=t) one-tail	0.313351816	
t Critical one-tail	1.690924255	
P(T<=t) two-tail	0.626703631	
t Critical two-tail	2.032244509	

t-Test: Two-Sample Assuming Unequal Variances

	Branch 1	*Branch 2*
Mean	53947.74955	52559.33675
Variance	79197935.64	64833417.48
Observations	18	18
Hypothesized Mean Difference	1	
df	34	
t Stat	0.490471108	
P(T<=t) one-tail	0.313475553	
t Critical one-tail	1.690924255	
P(T<=t) two-tail	0.626951106	
t Critical two-tail	2.032244509	

For Alternative Hypothesis:

Go to the **Data tab** > Click **Data Analysis** > Select **t-Test: Paired Two Sample for Means.**



t-Test: Paired Two Sample for Means		
	Branch 1	*Branch 2*
Mean	53947.74955	52559.33675
Variance	79197935.64	64833417.48
Observations	18	18
Pearson Correlation	0.372704372	
Hypothesized Mean Difference	0	
df	17	
t Stat	0.618796614	
P(T<=t) one-tail	0.272126886	
t Critical one-tail	1.739606726	
P(T<=t) two-tail	0.544253772	
t Critical two-tail	2.109815578	

Practical No: 4(b)

Aim: Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chisquare test).

Code:

```
from scipy import stats
class_1_scores = [80, 85, 90, 87, 91]
class_2_scores = [75, 79, 84, 88, 80]
t_stat, p_value = stats.ttest_ind(class_1_scores, class_2_scores)
print("t-statistic:", t_stat)
print("p-value:", p_value)
if p_value < 0.05:
    print("Reject the null hypothesis: There is a significant difference.")
else:
    print("Fail to reject the null hypothesis: No significant difference.")
print()
import numpy as np
from scipy.stats import chi2_contingency
data = np.array([[50, 30],
                 [20, 40]])
chi2, p_value, dof, expected = chi2_contingency(data)
alpha = 0.05  # Significance level
print(f"Chi-Square: {chi2:.2f}, P-value: {p_value:.4f}, Degrees of Freedom: {dof}")
print("Expected Frequencies:")
print(expected)
if p_value < alpha:
    print("Reject the null hypothesis: There's a significant association between gender and product choice.")
else:
    print("Fail to reject the null hypothesis: No significant association between gender and product choice.")
```

Output:

```
t-statistic: 1.820339628650751
p-value: 0.1061968500647213
Fail to reject the null hypothesis: No significant difference.

Chi-Square: 10.53, P-value: 0.0012, Degrees of Freedom: 1
Expected Frequencies:
[[40. 40.]
 [30. 30.]]
Reject the null hypothesis: There's a significant association between gender and product choice.
PS D:\TYBSC_CS_Data_Science_Practicals>
```

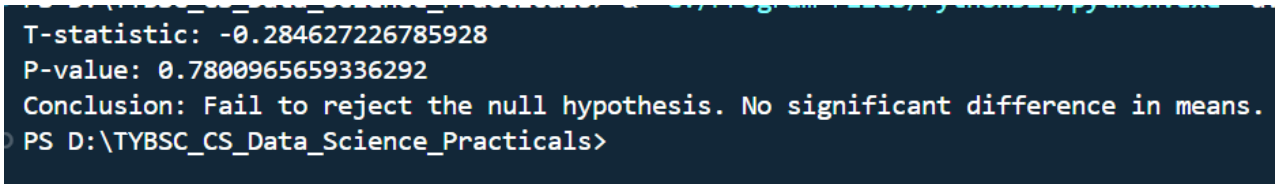
Practical No: 4(c)

Aim: Interpret the results and draw conclusions based on the test outcomes.

Code:

```
import numpy as np
from scipy.stats import ttest_ind
group1 = [23, 45, 67, 89, 12, 56, 78, 90]
group2 = [34, 56, 78, 45, 23, 67, 89, 100]
t_stat, p_value = ttest_ind(group1, group2)
alpha = 0.05
print("T-statistic:", t_stat)
print("P-value:", p_value)
if p_value < alpha:
    print("Conclusion: Reject the null hypothesis. The means are significantly different.")
else:
    print("Conclusion: Fail to reject the null hypothesis. No significant difference in means.")
```

Output:



Practical No: 5(a)

Aim: Perform one-way ANOVA to compare means across multiple groups.

Code:

```
import numpy as np
import scipy.stats as stats
group1 = [23, 45, 67, 45, 34, 34, 56, 76, 34]
group2 = [78, 90, 67, 56, 78, 90, 98, 67]
group3 = [45, 67, 23, 56, 45, 43, 29, 37]
f_statistic, p_value = stats.f_oneway(group1, group2, group3)
print("F-statistic:", f_statistic)
print("P-value:", p_value)
if p_value < 0.05:
    print("Reject the null hypothesis: There are significant differences between the groups.")
else:
    print("Fail to reject the null hypothesis: There are no significant differences between the groups.")
```

Output:

```
F-statistic: 12.868311270211178
P-value: 0.00019919613829838252
Reject the null hypothesis: There are significant differences between the groups.
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical No: 5(b)

Aim: Conduct post-hoc tests to identify significant differences between group means.

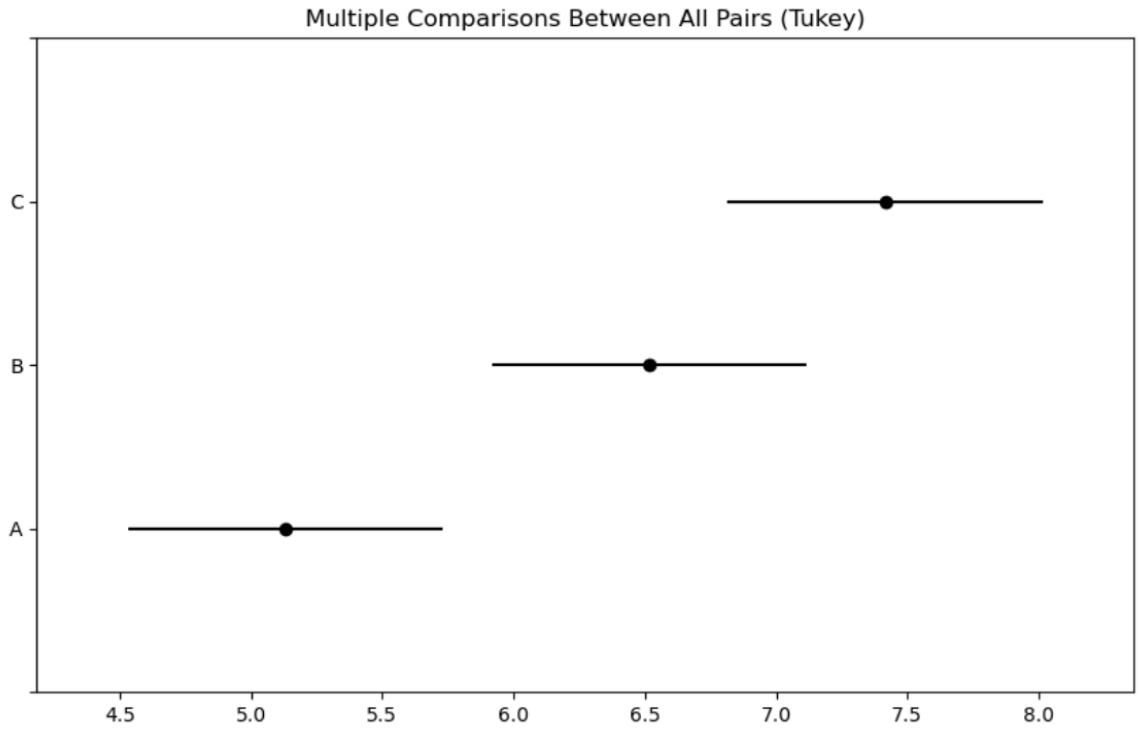
Code:

```
import pandas as pd
import numpy as np
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
data = pd.DataFrame({
    "group": np.repeat(['A', 'B', 'C'], 10),
    "values": np.concatenate([
        np.random.normal(5, 1, 10),
        np.random.normal(6, 1, 10),
        np.random.normal(7, 1, 10)
    ])
})
anova_model = ols('values ~ group', data=data).fit()
anova_results = anova_lm(anova_model)
print("ANOVA Results:")
print(anova_results)
tukey = pairwise_tukeyhsd(endog=data['values'], groups=data['group'], alpha=0.05)
print("\nTukey HSD Results:")
print(tukey)
tukey.plot_simultaneous()
```

Output:

```
ANOVA Results:
            df     sum_sq    mean_sq          F    PR(>F)
group      2.0  25.133413  12.566706  10.419518  0.000443
Residual  27.0  32.563990   1.206074        NaN       NaN

Tukey HSD Results:
Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj   lower  upper  reject
-------------------------------------------------
     A      B   1.8783  0.002  0.6605  3.096   True
     A      C   1.9994  0.001  0.7816 3.2171   True
     B      C   0.1211 0.9671 -1.0966 1.3388  False
-------------------------------------------------
```


Multiple Comparisons Between All Pairs (Tukey)
C
B
A
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

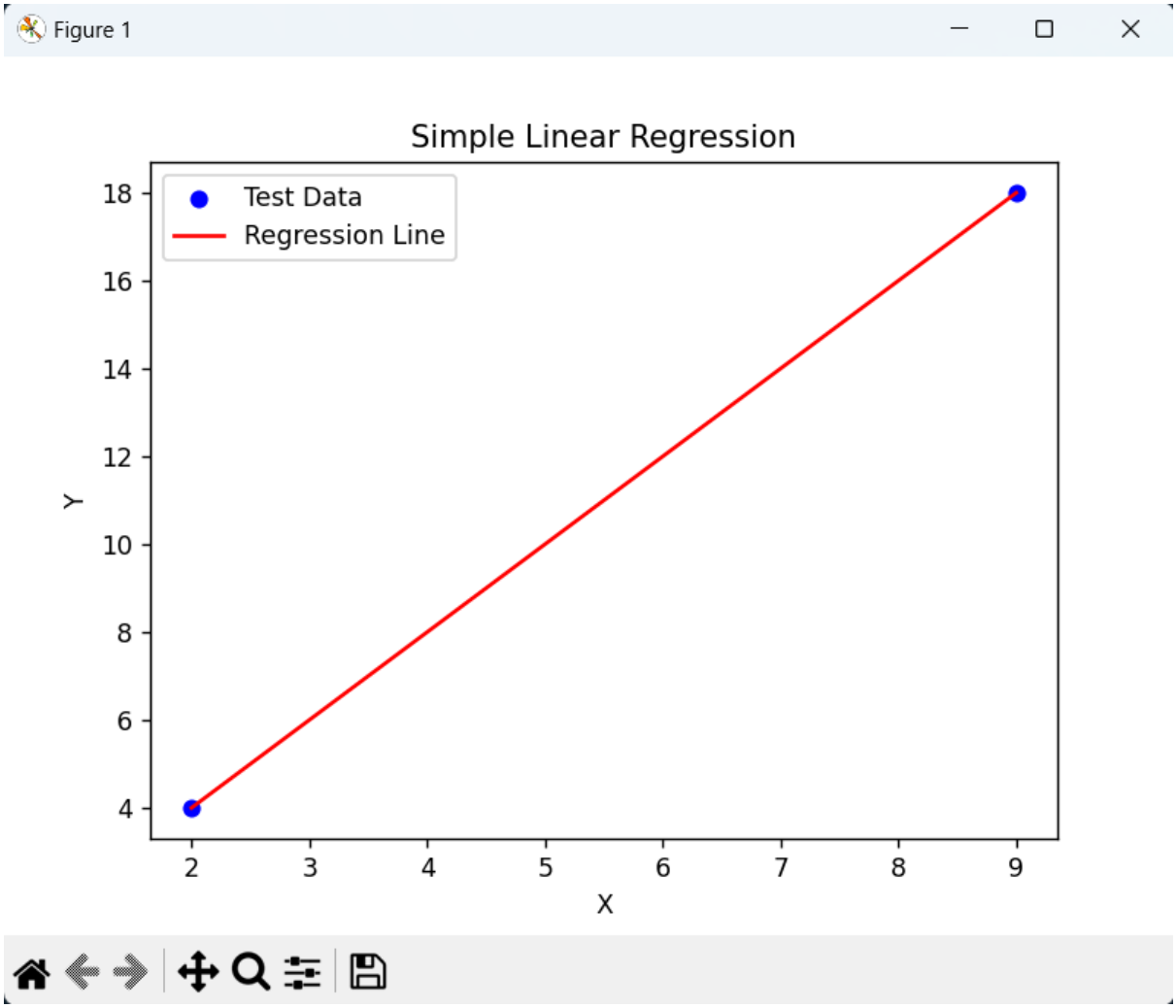
Practical No: 6(a)

Aim: Implement simple linear regression using a dataset.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('dataset.csv')
X = df[['X']]  # Independent variable
Y = df['Y']    # Dependent variable
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
plt.scatter(X_test, Y_test, color='blue', label='Test Data')
plt.plot(X_test, Y_pred, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

Output:




```
Mean Squared Error: 3.9443045261050 59e-31
R-squared: 1.0
```

Practical No: 6(b)

Aim: Explore and interpret the regression model coefficients and goodness-of-fit measures.

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
import matplotlib.pyplot as plt
data = fetch_california_housing(as_frame=True)
df = data.frame
X = df.drop(columns=["MedHouseVal"])
y = df["MedHouseVal"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
coefficients = model.coef_
intercept = model.intercept_
print("Coefficients (scikit-learn):", coefficients)
print("Intercept (scikit-learn):", intercept)
X_train_sm = sm.add_constant(X_train)
X_test_sm = sm.add_constant(X_test)
model_sm = sm.OLS(y_train, X_train_sm).fit()
print("\nStatsmodels Summary:")
print(model_sm.summary())
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```
print("\nGoodness-of-Fit Measures:")
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R2):", r2)
print("R-squared (statsmodels):", model_sm.rsquared)
print("Adjusted R-squared (statsmodels):", model_sm.rsquared_adj)
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted")
plt.show()
```

Output:

```
Coefficients (scikit-learn): [ 4.48674910e-01  9.72425752e-03 -1.23323343e-01  7.83144907e-01
 -2.02962058e-06 -3.52631849e-03 -4.19792487e-01 -4.33708065e-01]
Intercept (scikit-learn): -37.02327770606416

Statsmodels Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:            MedHouseVal   R-squared:                       0.613
Model:                            OLS   Adj. R-squared:                  0.612
Method:                 Least Squares   F-statistic:                     3261.
Date:                Sat, 28 Dec 2024   Prob (F-statistic):               0.00
Time:                        10:57:03   Log-Likelihood:                -17998.
No. Observations:               16512   AIC:                         3.601e+04
Df Residuals:                   16503   BIC:                         3.608e+04
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	-37.0233	0.728	-50.835	0.000	-38.451	-35.596
MedInc	0.4487	0.005	95.697	0.000	0.439	0.458
HouseAge	0.0097	0.000	19.665	0.000	0.009	0.011
AveRooms	-0.1233	0.007	-18.677	0.000	-0.136	-0.110
AveBedrms	0.7831	0.033	23.556	0.000	0.718	0.848
Population	-2.03e-06	5.25e-06	-0.387	0.699	-1.23e-05	8.26e-06
AveOccup	-0.0035	0.000	-7.253	0.000	-0.004	-0.003
Latitude	-0.4198	0.008	-52.767	0.000	-0.435	-0.404
Longitude	-0.4337	0.008	-52.117	0.000	-0.450	-0.417

Omnibus:	3333.187	Durbin-Watson:	1.962
Prob(Omnibus):	0.000	Jarque-Bera (JB):	9371.466
Skew:	1.071	Prob(JB):	0.00
Kurtosis:	6.006	Cond. No.	2.38e+05

```
Goodness-of-Fit Measures:
Mean Squared Error (MSE): 0.555891598695244
Root Mean Squared Error (RMSE): 0.7455813830127761
R-squared (R2): 0.5757877060324511
R-squared (statsmodels): 0.6125511913966952
Adjusted R-squared (statsmodels): 0.6123633715779455
```

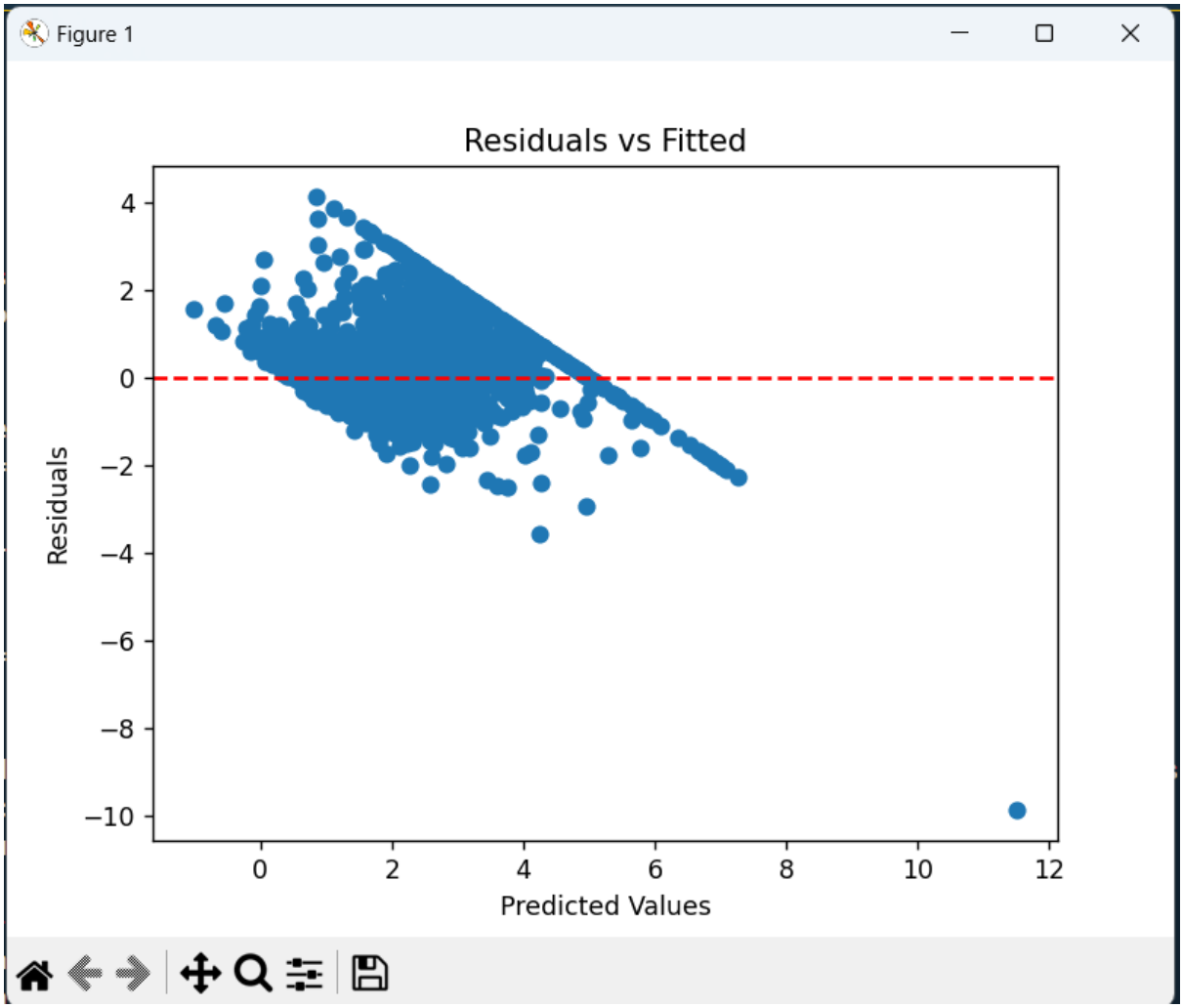


Figure 1
Residuals vs Fitted
Residuals
Predicted Values
4
2
0
−2
−4
−6
−8
−10
0
2
4
6
8
10
12

Practical No: 6(c)

Aim: Extend the analysis to multiple linear regression and assess the impact of additional predictors.

Code:

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(42)
n = 100
X1 = np.random.rand(n)
X2 = np.random.rand(n)
X3 = np.random.rand(n)
X4 = np.random.rand(n)
beta_0 = 2
beta_1 = 3
beta_2 = -1
beta_3 = 1.5
beta_4 = -2
noise = np.random.randn(n)
y = beta_0 + beta_1 * X1 + beta_2 * X2 + beta_3 * X3 + beta_4 * X4 + noise
df = pd.DataFrame({'X1': X1, 'X2': X2, 'X3': X3, 'X4': X4, 'y': y})
X = df[['X1', 'X2', 'X3', 'X4']]
X = sm.add_constant(X)
y = df['y']
model = sm.OLS(y, X).fit()
print("Multiple Linear Regression Summary:")
print(model.summary())
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
```

```
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print("\nVariance Inflation Factor (VIF):")
print(vif_data)
y_pred = model.predict(X)
residuals = y - y_pred
sns.histplot(residuals, kde=True)
plt.title('Residuals Histogram')
plt.show()
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Fitted Values')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
print(f"R-squared: {model.rsquared}")
print(f"Adjusted R-squared: {model.rsquared_adj}")
```

Output:

```
Multiple Linear Regression Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.650
Model:                            OLS   Adj. R-squared:                  0.635
Method:                 Least Squares   F-statistic:                     44.04
Date:                Sat, 28 Dec 2024   Prob (F-statistic):           7.38e-21
Time:                        11:02:30   Log-Likelihood:                -131.23
No. Observations:                 100   AIC:                             272.5
Df Residuals:                      95   BIC:                             285.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.1678      0.394      5.504      0.000       1.386       2.950
X1             2.8118      0.320      8.775      0.000       2.176       3.448
X2            -1.0858      0.320     -3.389      0.001      -1.722      -0.450
X3             1.2937      0.329      3.936      0.000       0.641       1.946
X4            -1.7786      0.332     -5.350      0.000      -2.439      -1.119
==============================================================================
Omnibus:                        0.962   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.618   Jarque-Bera (JB):                0.962
Skew:                           0.228   Prob(JB):                        0.618
Kurtosis:                       2.848   Cond. No.                         8.45
==============================================================================
```

```
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.1678      0.394      5.504      0.000       1.386       2.950
X1             2.8118      0.320      8.775      0.000       2.176       3.448
X2            -1.0858      0.320     -3.389      0.001      -1.722      -0.450
X3             1.2937      0.329      3.936      0.000       0.641       1.946
X4            -1.7786      0.332     -5.350      0.000      -2.439      -1.119
==============================================================================
Omnibus:                        0.962   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.618   Jarque-Bera (JB):                0.962
Skew:                           0.228   Prob(JB):                        0.618
Kurtosis:                       2.848   Cond. No.                         8.45
==============================================================================
X1             2.8118      0.320      8.775      0.000       2.176       3.448
X2            -1.0858      0.320     -3.389      0.001      -1.722      -0.450
X3             1.2937      0.329      3.936      0.000       0.641       1.946
X4            -1.7786      0.332     -5.350      0.000      -2.439      -1.119
==============================================================================
Omnibus:                        0.962   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.618   Jarque-Bera (JB):                0.962
Skew:                           0.228   Prob(JB):                        0.618
Kurtosis:                       2.848   Cond. No.                         8.45
==============================================================================
X3             1.2937      0.329      3.936      0.000       0.641       1.946
X4            -1.7786      0.332     -5.350      0.000      -2.439      -1.119
==============================================================================
Omnibus:                        0.962   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.618   Jarque-Bera (JB):                0.962
Skew:                           0.228   Prob(JB):                        0.618
Kurtosis:                       2.848   Cond. No.                         8.45
==============================================================================
```

```
==============================================================================
Omnibus:                        0.962   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.618   Jarque-Bera (JB):                0.962
Skew:                           0.228   Prob(JB):                        0.618
Kurtosis:                       2.848   Cond. No.                         8.45
==============================================================================
Omnibus:                        0.962   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.618   Jarque-Bera (JB):                0.962
Skew:                           0.228   Prob(JB):                        0.618
Kurtosis:                       2.848   Cond. No.                         8.45
==============================================================================
Skew:                           0.228   Prob(JB):                        0.618
Kurtosis:                       2.848   Cond. No.                         8.45
==============================================================================
```

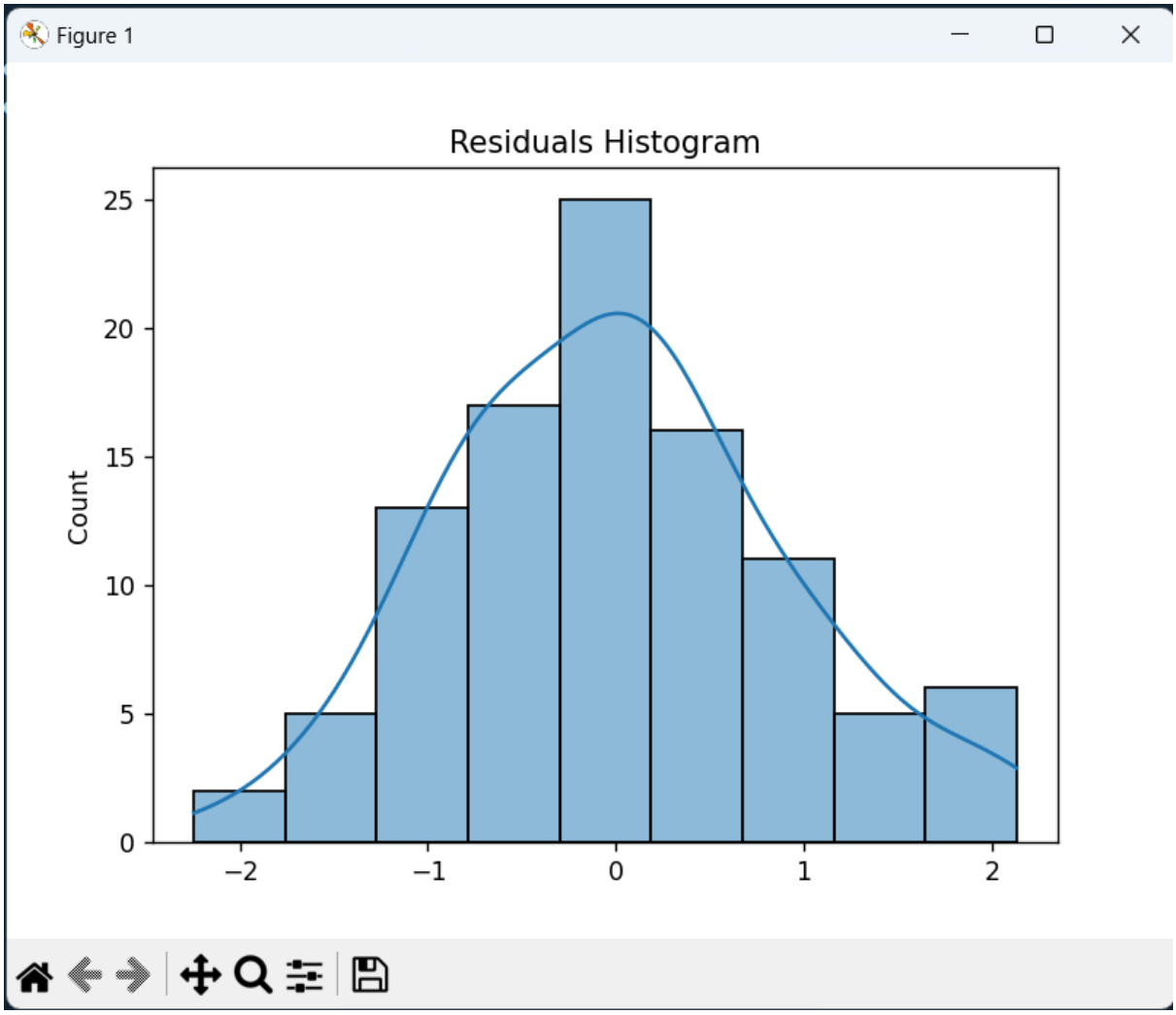
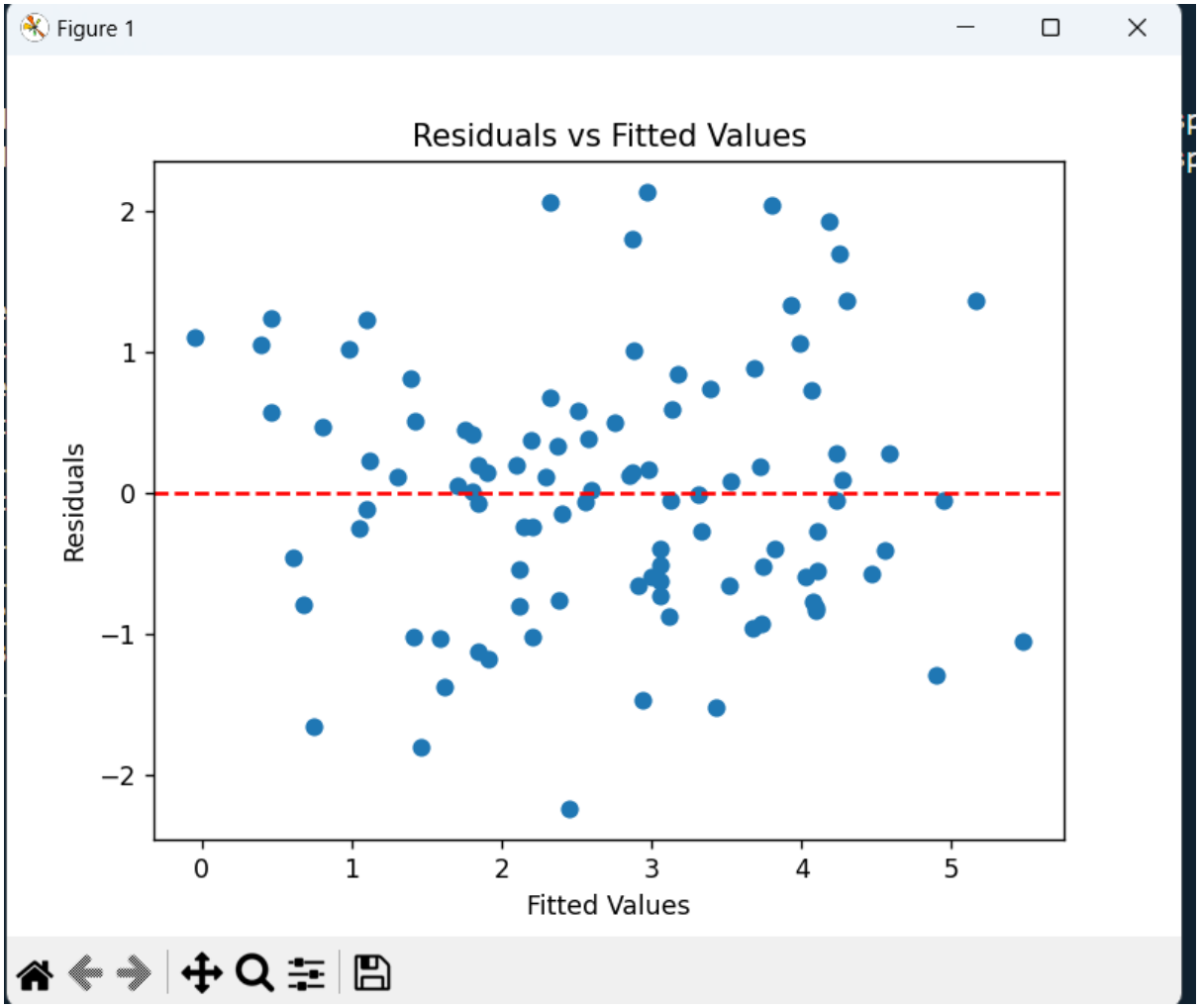


Figure 1
Residuals Histogram
Count
25
20
15
10
5
0
−2
−1
0
1
2




```
Variance Inflation Factor (VIF):
  Feature        VIF
0   const  18.235638
  Feature        VIF
0   const  18.235638
1      X1   1.057816
0   const  18.235638
1      X1   1.057816
1      X1   1.057816
2      X2   1.026656
3      X3   1.082893
4      X4   1.107864
R-squared: 0.6496320068365531
Adjusted R-squared: 0.6348796702823027
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical No: 7(a)

Aim: Build a logistic regression model to predict a binary outcome.

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
print("Dataset Head:")
print(df.head())
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy:.2f}")
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(class_report)
coefficients = pd.DataFrame({'Feature': X.columns, 'Coefficient': logreg.coef_[0]})
coefficients = coefficients.sort_values(by='Coefficient', ascending=False)
print("\nFeature Importance:")
print(coefficients)
```

Output:

```
Dataset Head:
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625
```

```
   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]

Accuracy: 0.96

Confusion Matrix:
[[39  4]
 [ 1 70]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

```
Feature Importance:
                     Feature  Coefficient
0                mean radius     2.311667
11             texture error     1.388432
20              worst radius     1.230929
1                mean texture     0.125152
15         compactness error     0.049276
19  fractal dimension error     0.013465
3                  mean area     0.001752
16           concavity error    -0.006685
14          smoothness error    -0.024195
9     mean fractal dimension    -0.026227
23                worst area    -0.027561
22           worst perimeter    -0.037782
18            symmetry error    -0.043392
17      concave points error    -0.045517
10              radius error    -0.069128
13                area error    -0.078406
29  worst fractal dimension    -0.106881
4            mean smoothness    -0.179505
2             mean perimeter    -0.199868
12           perimeter error    -0.217337
8              mean symmetry    -0.247409
24          worst smoothness    -0.335239
21             worst texture    -0.400024
5            mean compactness    -0.423808
7         mean concave points    -0.431823
27      worst concave points    -0.740505
6              mean concavity    -0.747530
28            worst symmetry    -0.883911
25         worst compactness    -1.156242
26           worst concavity    -1.744126
```

Practical No: 7(b)

Aim: Evaluate the model's performance using classification metrics (e.g., accuracy, precision, recall).

Code:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Output:

```
Accuracy: 0.8566666666666667
Precision: 0.8888888888888888
Recall: 0.8258064516129032
F1-Score: 0.8561872909698997

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       145
           1       0.89      0.83      0.86       155

    accuracy                           0.86       300
   macro avg       0.86      0.86      0.86       300
weighted avg       0.86      0.86      0.86       300


Confusion Matrix:
[[129  16]
 [ 27 128]]
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical No: 7(c)

Aim: Construct a decision tree model and interpret the decision rules for classification.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
tree_rules = export_text(clf, feature_names=X.columns.tolist())
print("\nDecision Tree Rules:\n")
print(tree_rules)
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=iris.target_names)
plt.show()
```

Output:

```
Accuracy: 1.00

Decision Tree Rules:

|--- petal length (cm) <= 2.45
|   |--- class: 0
|--- petal length (cm) >  2.45
|   |--- petal length (cm) <= 4.75
|   |   |--- petal width (cm) <= 1.60
|   |   |   |--- class: 1
|   |   |--- petal width (cm) >  1.60
|   |   |   |--- class: 2
|   |--- petal length (cm) >  4.75
|   |   |--- petal width (cm) <= 1.75
|   |   |   |--- petal length (cm) <= 4.95
|   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.95
|   |   |   |   |--- petal width (cm) <= 1.55
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- petal width (cm) >  1.55
|   |   |   |   |   |--- petal length (cm) <= 5.45
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- petal length (cm) >  5.45
|   |   |   |   |   |   |--- class: 2
|   |   |--- petal width (cm) >  1.75
|   |   |   |--- petal length (cm) <= 4.85
|   |   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |   |--- class: 1
|   |   |   |--- petal length (cm) >  4.85
|   |   |   |   |--- class: 2
```

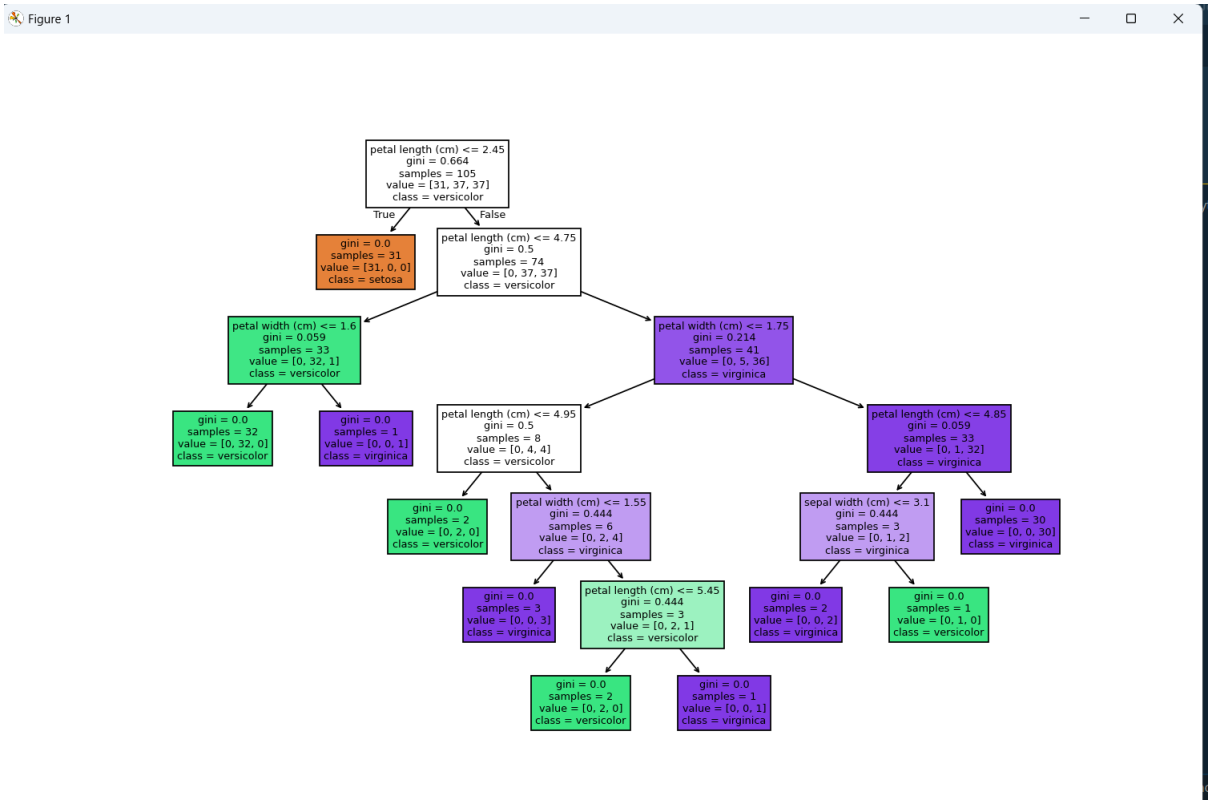


Figure 1
petal length (cm) <= 2.45
gini = 0.664
samples = 105
value = [31, 37, 37]
class = versicolor
True
False
gini = 0.0
samples = 31
value = [31, 0, 0]
class = setosa
petal length (cm) <= 4.75
gini = 0.5
samples = 74
value = [0, 37, 37]
class = versicolor
petal width (cm) <= 1.6
gini = 0.059
samples = 33
value = [0, 32, 1]
class = versicolor
petal width (cm) <= 1.75
gini = 0.214
samples = 41
value = [0, 5, 36]
class = virginica
gini = 0.0
samples = 32
value = [0, 32, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
petal length (cm) <= 4.95
gini = 0.5
samples = 8
value = [0, 4, 4]
class = versicolor
petal length (cm) <= 4.85
gini = 0.059
samples = 33
value = [0, 1, 32]
class = virginica
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
petal width (cm) <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 30
value = [0, 0, 30]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
petal length (cm) <= 5.45
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

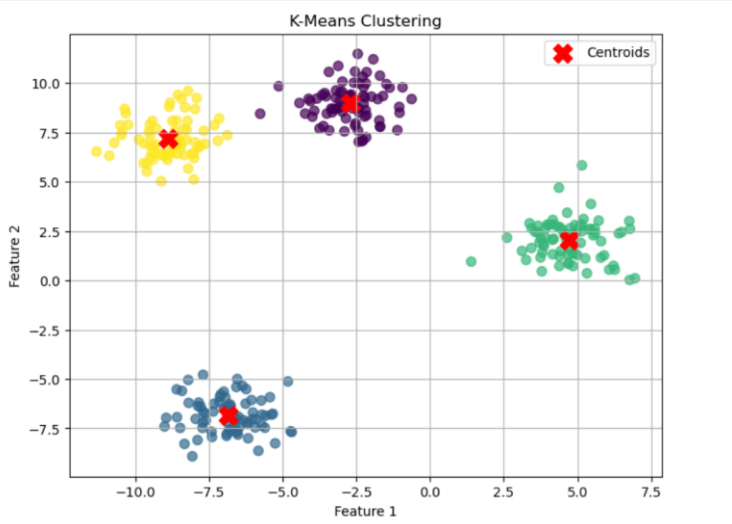
Practical No: 8(a)

Aim: Apply the K-Means algorithm to group similar data points into clusters.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=300, centers=4, random_state=42)
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', s=50, alpha=0.7)
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', marker='X', s=200,
label='Centroids')
plt.title("K-Means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Practical No: 8(b)

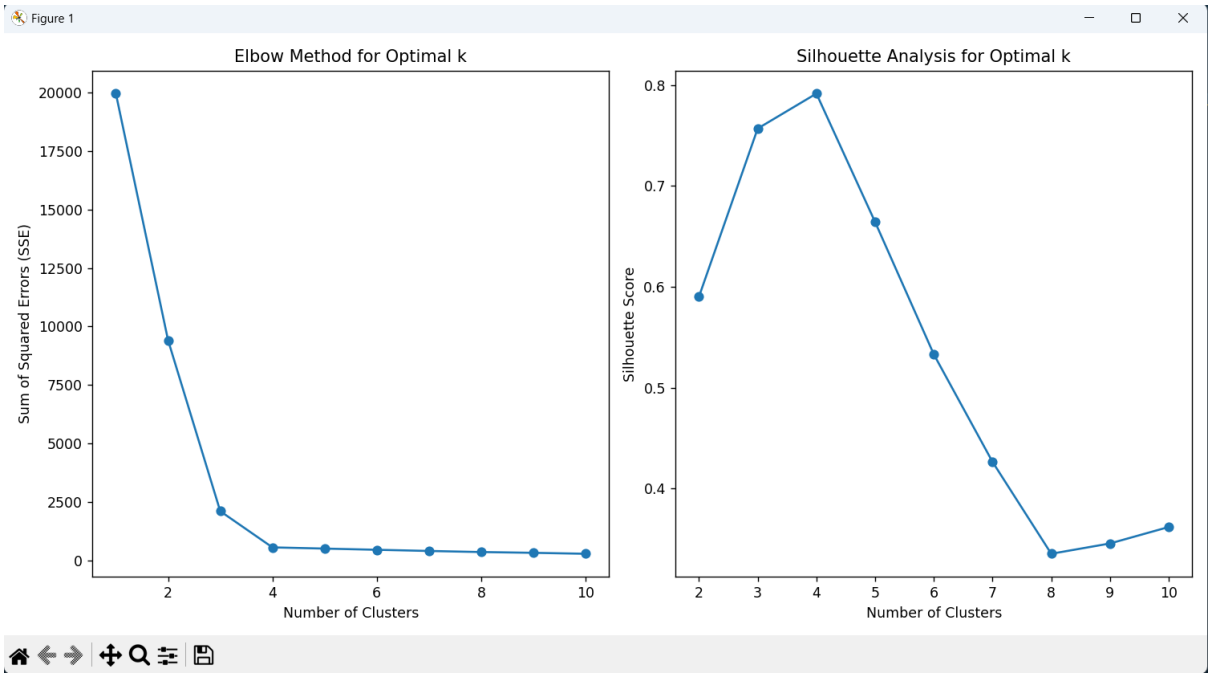
Aim: Determine the optimal number of clusters using elbow method or silhouette analysis.

Code:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(k_range, sse, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Errors (SSE)')
silhouette_scores = []
k_range_sil = range(2, 11)
for k in k_range_sil:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    score = silhouette_score(X, kmeans.labels_)
    silhouette_scores.append(score)
plt.subplot(1, 2, 2)
plt.plot(k_range_sil, silhouette_scores, marker='o')
plt.title('Silhouette Analysis for Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
```

```
plt.tight_layout()
plt.show()
```

Output:



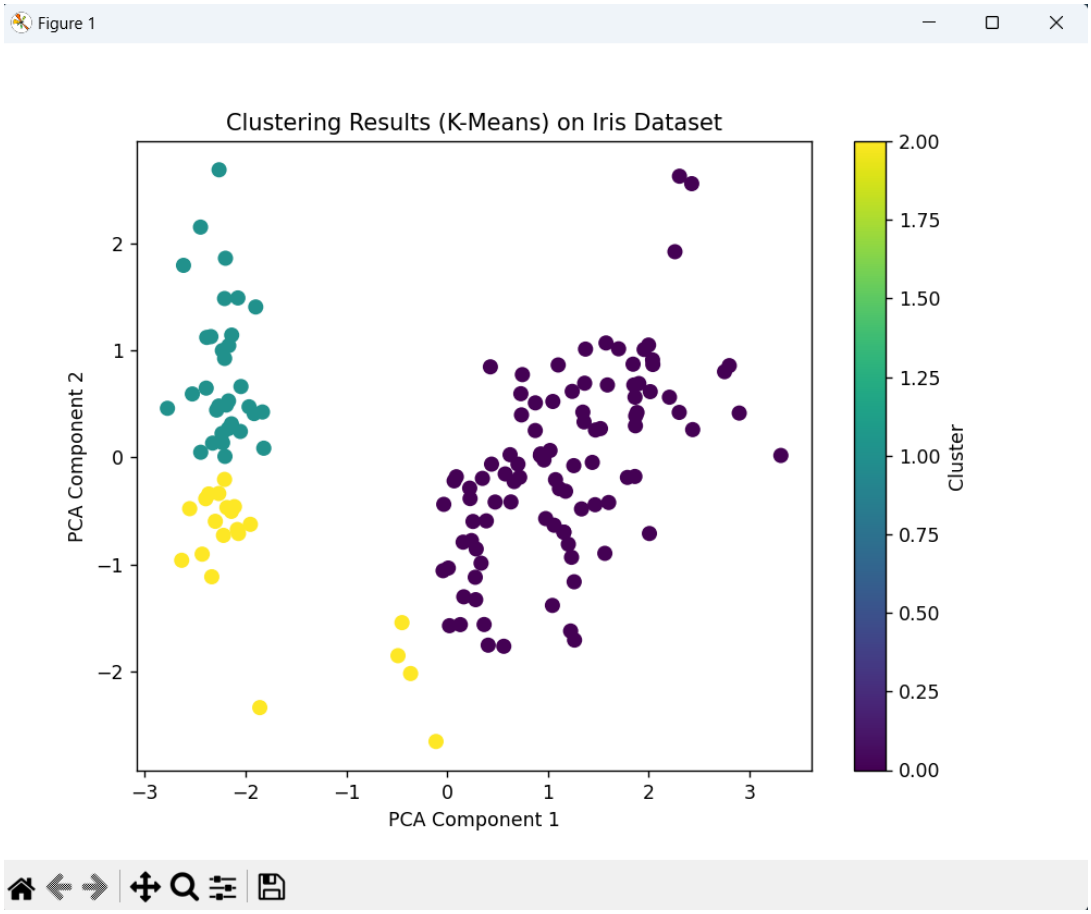
Practical No: 8(c)

Aim: Visualize the clustering results and analyze the cluster characteristics..

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
iris = load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
kmeans = KMeans(n_clusters=3, random_state=42)  # n_clusters set to 3 for Iris dataset
clusters = kmeans.fit_predict(data_scaled)
data['Cluster'] = clusters
pca = PCA(n_components=2)
pca_components = pca.fit_transform(data_scaled)
plt.figure(figsize=(8,6))
plt.scatter(pca_components[:, 0], pca_components[:, 1], c=clusters, cmap='viridis', s=50)
plt.title("Clustering Results (K-Means) on Iris Dataset")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label='Cluster')
plt.show()
cluster_characteristics = data.groupby('Cluster').mean()  # Mean of features per cluster
print("Cluster Characteristics:\n", cluster_characteristics)
```

Output:




```
Cluster Characteristics:
         sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
Cluster
0                 6.314583          2.895833           4.973958          1.703125
1                 5.169697          3.630303           1.493939          0.272727
2                 4.747619          2.895238           1.757143          0.352381
PS D:\TYBSC_CS_Data_Science_Practicals>
```

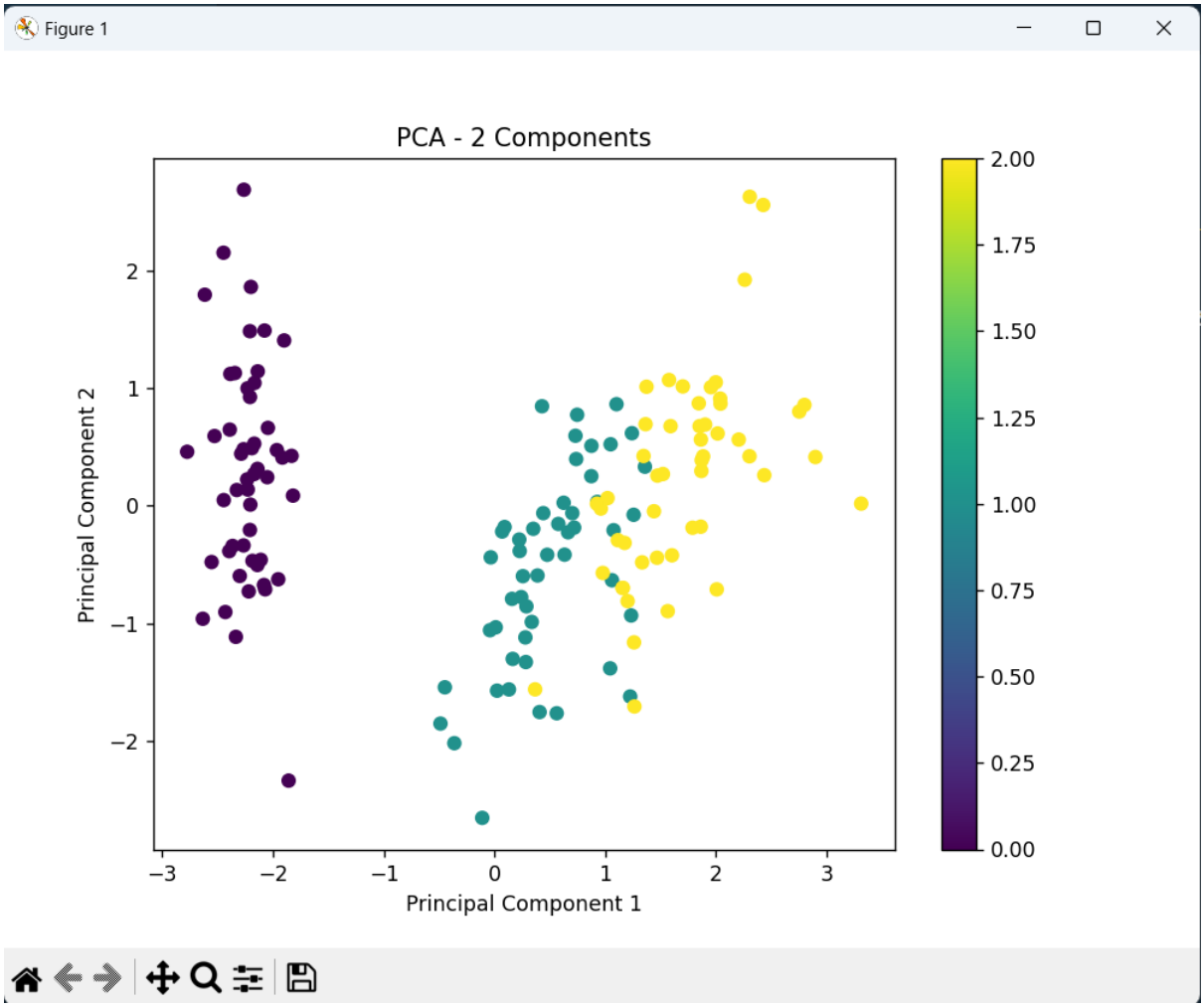
Practical No: 9(a)

Aim: Perform PCA on a dataset to reduce dimensionality.

Code:

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
data = load_iris()
X = data.data
y = data.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
print(f"Explained Variance Ratio: {pca.explained_variance_ratio_}")
print(f"Total Explained Variance: {np.sum(pca.explained_variance_ratio_)}")
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - 2 Components')
plt.colorbar()
plt.show()
```

Output:




```
Explained Variance Ratio: [0.72962445 0.22850762]
Total Explained Variance: 0.9581320720000166
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical No: 9(b)

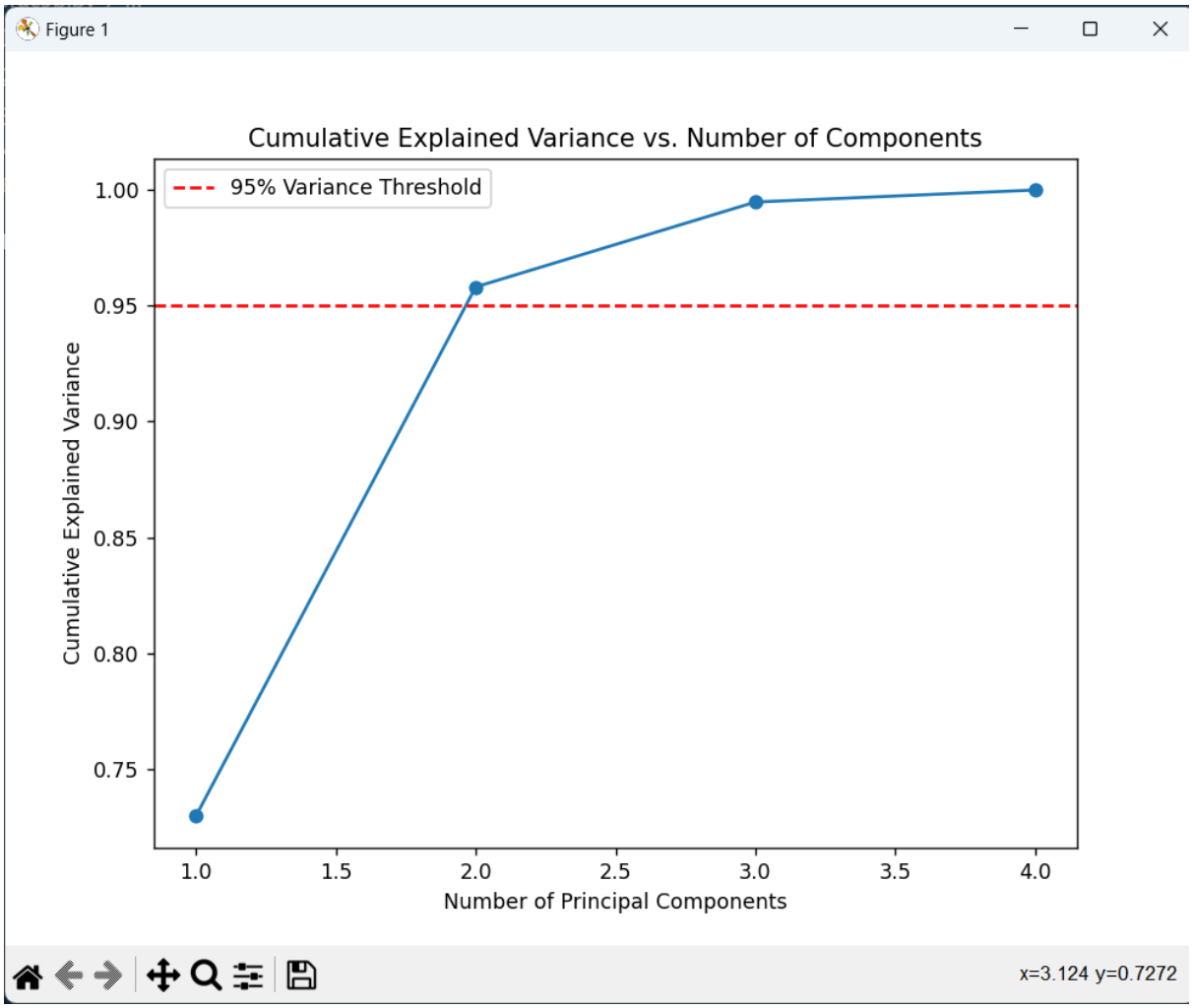
Aim: Evaluate the explained variance and select the appropriate number of principal components.

Code:

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
data = load_iris()
X = data.data
y = data.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA()
pca.fit(X_scaled)
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), cumulative_explained_variance, marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance vs. Number of Components')
plt.axhline(y=0.95, color='r', linestyle='--', label="95% Variance Threshold")
plt.legend()
plt.show()
n_components_95 = np.argmax(cumulative_explained_variance >= 0.95) + 1
print(f"Number of components that explain 95% of the variance: {n_components_95}")
pca = PCA(n_components=n_components_95)
X_pca = pca.fit_transform(X_scaled)
```

```
print(f"Explained Variance Ratio for {n_components_95} components:
{pca.explained_variance_ratio_}")
print(f"Total Explained Variance: {np.sum(pca.explained_variance_ratio_)}")
```

Output:




```
Number of components that explain 95% of the variance: 2
Explained Variance Ratio for 2 components: [0.72962445 0.22850762]
Total Explained Variance: 0.9581320720000166
PS D:\TYBSC_CS_Data_Science_Practicals>
```

Practical No: 9(c)

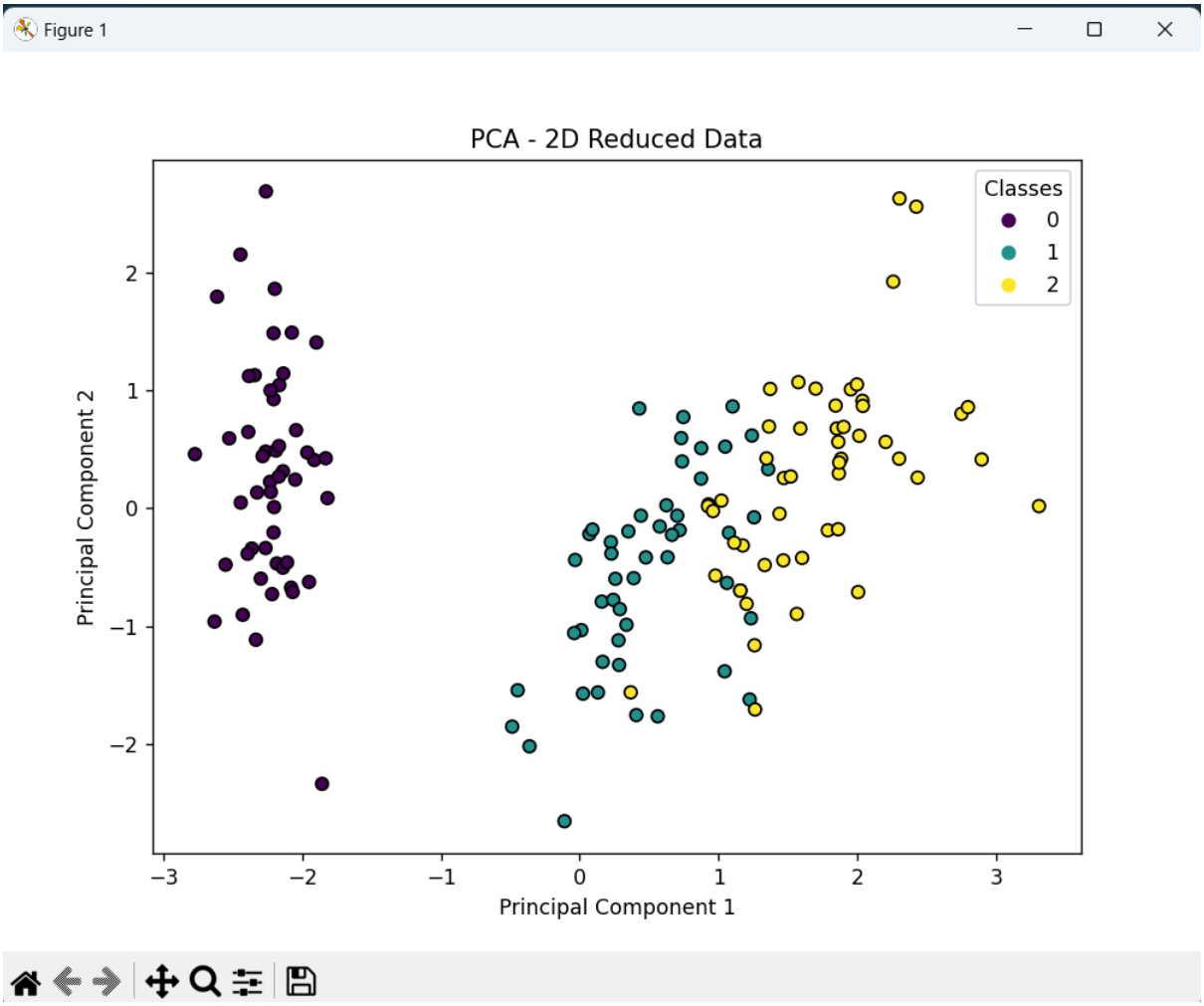
Aim: Visualize the data in the reduced-dimensional space.

Code:

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
labels = iris.target
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
pca = PCA(n_components=2)  # Reduce to 2D
pca_result = pca.fit_transform(scaled_data)
plt.figure(figsize=(8, 6))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1], c=labels, cmap='viridis',
edgecolor='black')
plt.title('PCA - 2D Reduced Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(*scatter.legend_elements(), title="Classes")
plt.show()
tsne = TSNE(n_components=2)
tsne_result = tsne.fit_transform(scaled_data)
plt.figure(figsize=(8, 6))
scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c=labels, cmap='viridis',
edgecolor='black')
plt.title('t-SNE - 2D Reduced Data')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend(*scatter.legend_elements(), title="Classes")
```

```
plt.show()
```

Output:



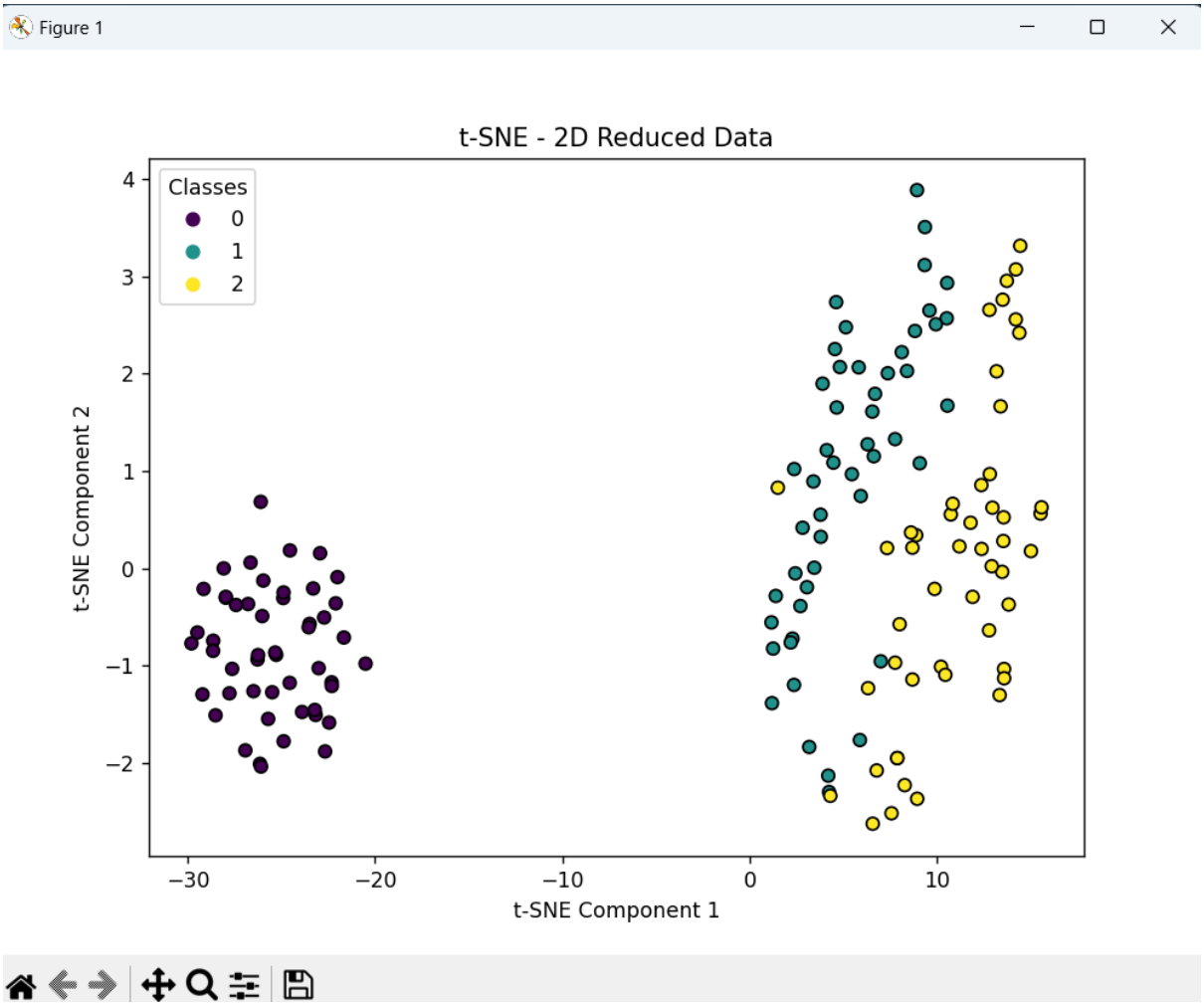


Figure 1
t-SNE - 2D Reduced Data
Classes
0
1
2
t-SNE Component 2
t-SNE Component 1
4
3
2
1
0
−1
−2
−30
−20
−10
0
10

Practical No: 10(a)

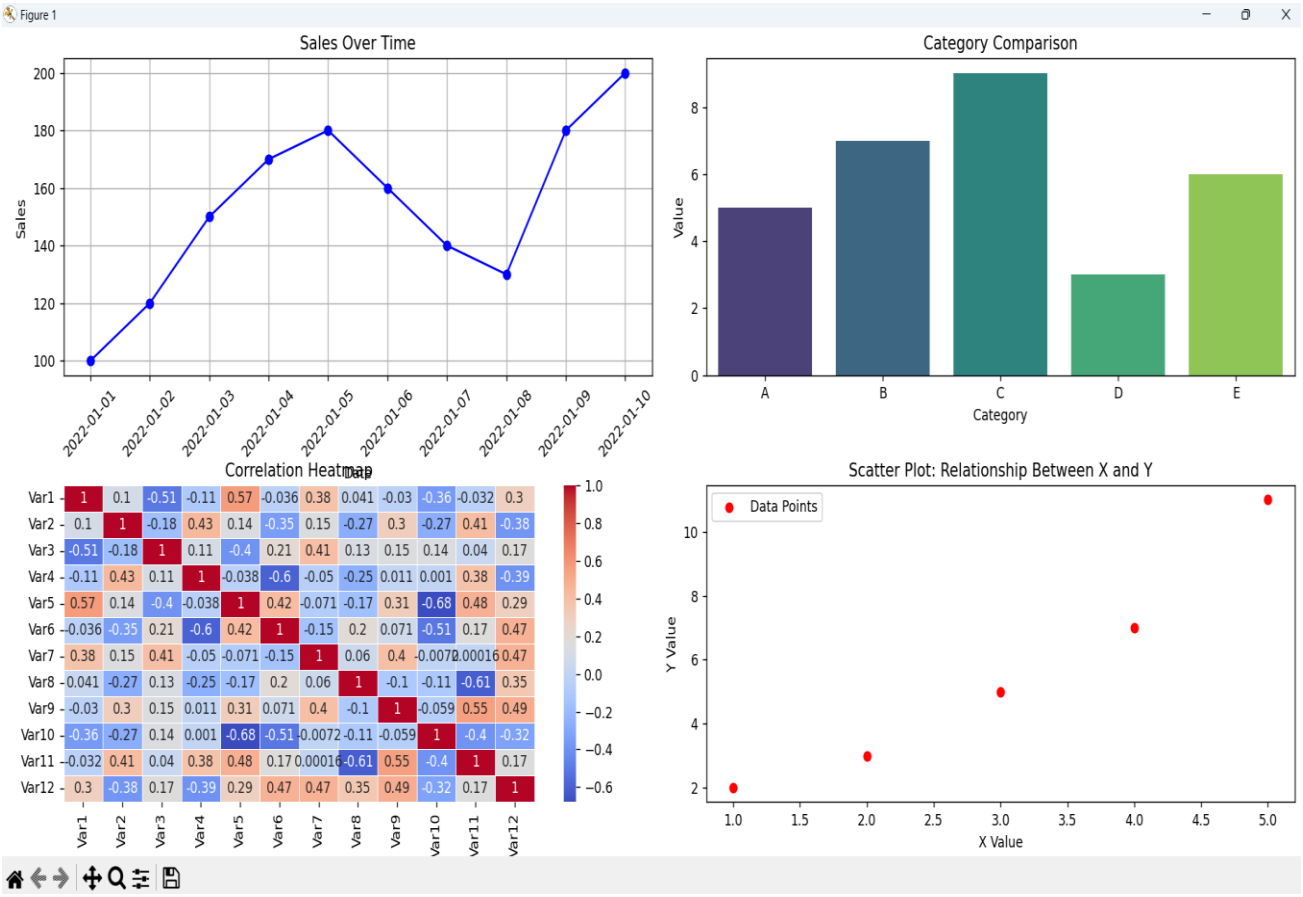
Aim: Create meaningful visualizations using data visualization tools.

Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
data = {'Date': pd.date_range(start='2022-01-01', periods=10, freq='D'),
    'Sales': [100, 120, 150, 170, 180, 160, 140, 130, 180, 200]}
df = pd.DataFrame(data)
categories = ['A', 'B', 'C', 'D', 'E']
values = [5, 7, 9, 3, 6]
data_heatmap = np.random.rand(10, 12)
df_heatmap = pd.DataFrame(data_heatmap, columns=[f'Var{i}' for i in range(1, 13)])
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes[0, 0].plot(df['Date'], df['Sales'], marker='o', linestyle='-', color='b')
axes[0, 0].set_title('Sales Over Time')
axes[0, 0].set_xlabel('Date')
axes[0, 0].set_ylabel('Sales')
axes[0, 0].tick_params(axis='x', rotation=45)
axes[0, 0].grid(True)
sns.barplot(x=categories, y=values, palette='viridis', ax=axes[0, 1])
axes[0, 1].set_title('Category Comparison')
axes[0, 1].set_xlabel('Category')
axes[0, 1].set_ylabel('Value')
sns.heatmap(df_heatmap.corr(), annot=True, cmap='coolwarm', linewidths=0.5, ax=axes[1,
0])
axes[1, 0].set_title('Correlation Heatmap')
axes[1, 1].scatter(x, y, color='r', label='Data Points')
axes[1, 1].set_title('Scatter Plot: Relationship Between X and Y')
axes[1, 1].set_xlabel('X Value')
```

```
axes[1, 1].set_ylabel('Y Value')
axes[1, 1].legend()
plt.tight_layout()
plt.show()
```

Output:



Practical No: 10(b)

Aim: Combine multiple visualizations to tell a compelling data story.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
data = {
    "Month": pd.date_range("2023-01-01", periods=12, freq="M").strftime("%b"),
    "Region": ["North", "South", "East", "West"] * 3,
    "Sales": [2500, 3000, 4000, 2000, 2600, 3200, 4100, 2200, 2700, 3300, 4200, 2300],
    "Category": ["Electronics", "Fashion", "Home", "Books"] * 3,
    "Age_Group": ["18-25", "26-35", "36-45", "46-55"] * 3,
    "Customer_Count": [120, 150, 200, 100, 130, 160, 210, 110, 140, 170, 220, 120],
}
df = pd.DataFrame(data)
month_order = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct",
"Nov", "Dec"]
df["Month"] = pd.Categorical(df["Month"], categories=month_order, ordered=True)
plt.figure(figsize=(10, 6))
sns.lineplot(data=df.groupby("Month")["Sales"].sum().reset_index(), x="Month", y="Sales",
marker="o", color="blue")
plt.title("Monthly Sales Trend", fontsize=16)
plt.xlabel("Month", fontsize=12)
plt.ylabel("Sales ($)", fontsize=12)
plt.grid()
plt.show()
plt.figure(figsize=(10, 6))
sns.barplot(data=df.groupby("Region")["Sales"].sum().reset_index(), x="Region", y="Sales",
palette="viridis")
plt.title("Sales by Region", fontsize=16)
```

```
plt.xlabel("Region", fontsize=12)
plt.ylabel("Sales ($)", fontsize=12)
plt.show()
fig_pie = px.pie(
    df.groupby("Category")["Sales"].sum().reset_index(),
    names="Category",
    values="Sales",
    title="Product Category Contribution",
    color_discrete_sequence=px.colors.sequential.RdBu
)
fig_pie.show()
fig_scatter = px.scatter(
    df, x="Customer_Count", y="Sales", color="Age_Group",
    title="Customer Count vs Sales by Age Group",
    size="Sales", hover_data=["Region"]
)
fig_scatter.show()
geo_data = {
    "Region": ["North", "South", "East", "West"],
    "Latitude": [28.6139, 12.9716, 22.5726, 19.0760],
    "Longitude": [77.2090, 77.5946, 88.3639, 72.8777],
}
geo_df =
pd.DataFrame(geo_data).merge(df.groupby("Region")["Sales"].sum().reset_index(),
on="Region")
fig_map = px.scatter_geo(
    geo_df,
    lat="Latitude",
    lon="Longitude",
    size="Sales",
    hover_name="Region",
    title="Geographic Sales Distribution",
    color="Sales",
```

```
    color_continuous_scale=px.colors.sequential.Plasma,
)
fig_map.show()
fig_dashboard = make_subplots(
    rows=2, cols=2,
    specs=[[{}, {}], [{"type": "domain"}, {}]],  # Define "domain" type for pie chart
    subplot_titles=("Monthly Sales Trend", "Sales by Region", "Product Category Contribution", "Customer Count vs Sales"),
)
monthly_sales = df.groupby("Month")["Sales"].sum().reset_index()
line_fig = go.Scatter(x=monthly_sales["Month"], y=monthly_sales["Sales"], mode='lines+markers', name='Sales')
fig_dashboard.add_trace(line_fig, row=1, col=1)
region_sales = df.groupby("Region")["Sales"].sum().reset_index()
bar_fig = go.Bar(x=region_sales["Region"], y=region_sales["Sales"], name="Sales by Region")
fig_dashboard.add_trace(bar_fig, row=1, col=2)
category_sales = df.groupby("Category")["Sales"].sum().reset_index()
pie_fig = go.Pie(labels=category_sales["Category"], values=category_sales["Sales"], name="Category Share")
fig_dashboard.add_trace(pie_fig, row=2, col=1)
scatter_fig = go.Scatter(
    x=df["Customer_Count"], y=df["Sales"], mode='markers',
    marker=dict(size=10), name="Age vs Sales"
)
fig_dashboard.add_trace(scatter_fig, row=2, col=2)
fig_dashboard.update_layout(height=800, width=1000, title_text="E-commerce Sales Dashboard")
fig_dashboard.show()
```

Output:

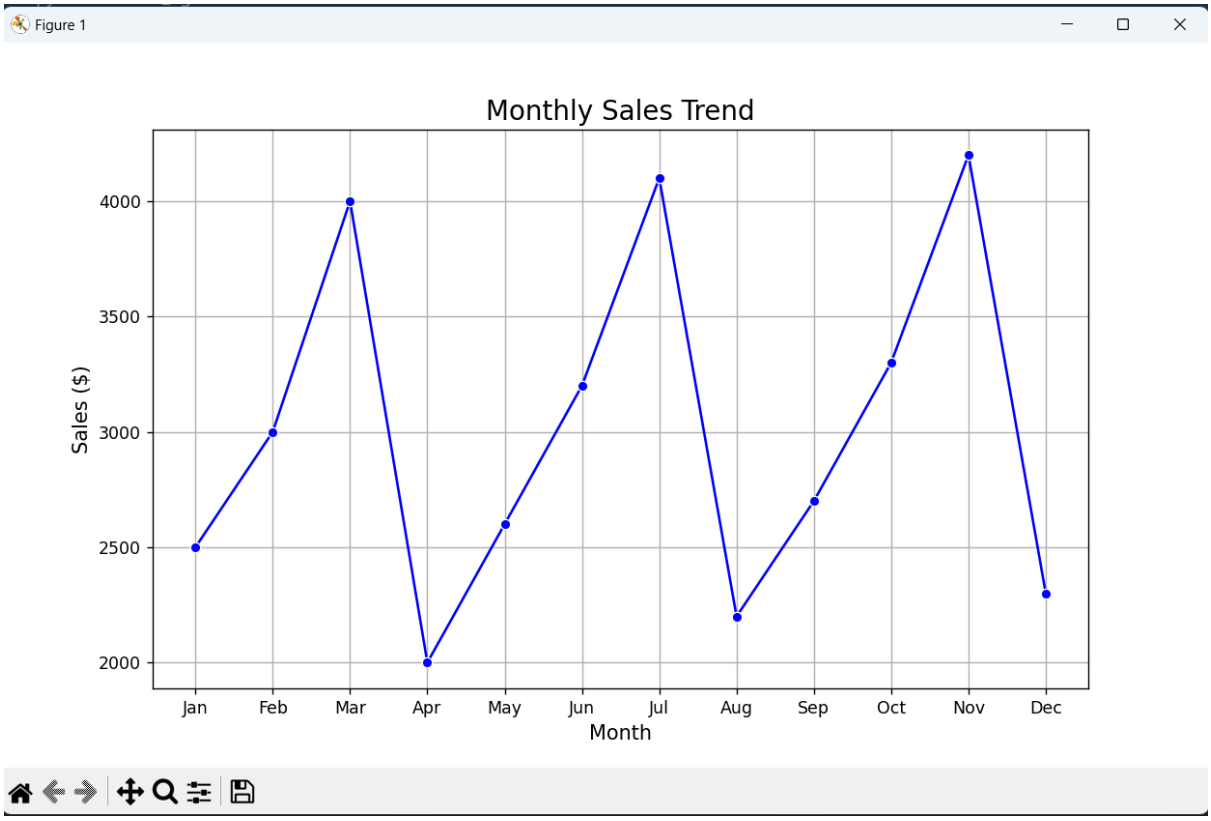


Figure 1
Monthly Sales Trend
Sales ($)
4000
3500
3000
2500
2000
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month

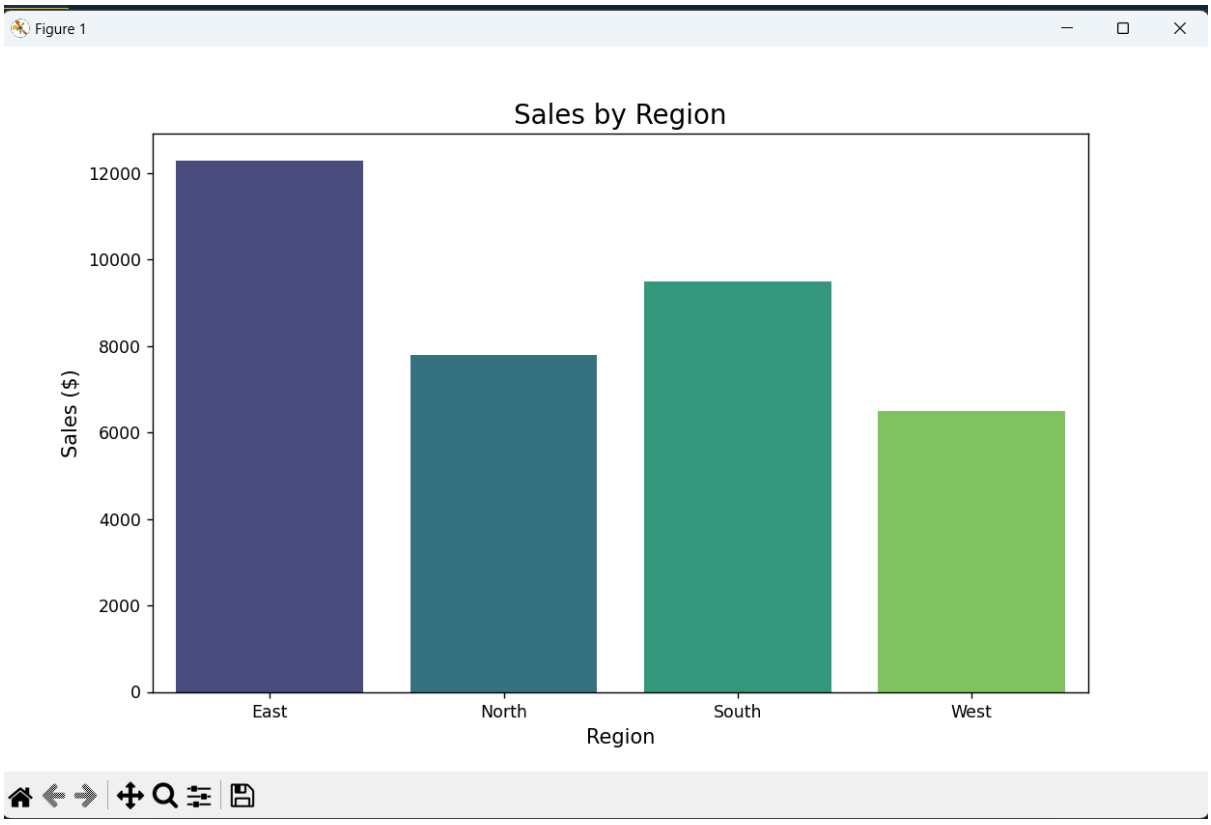
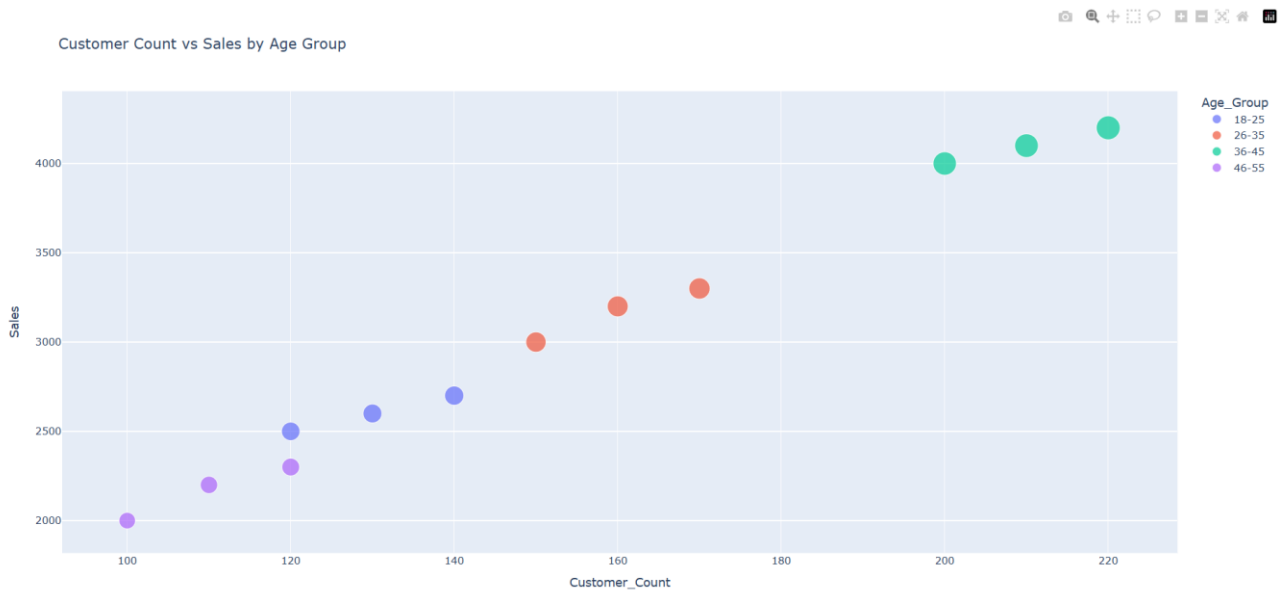
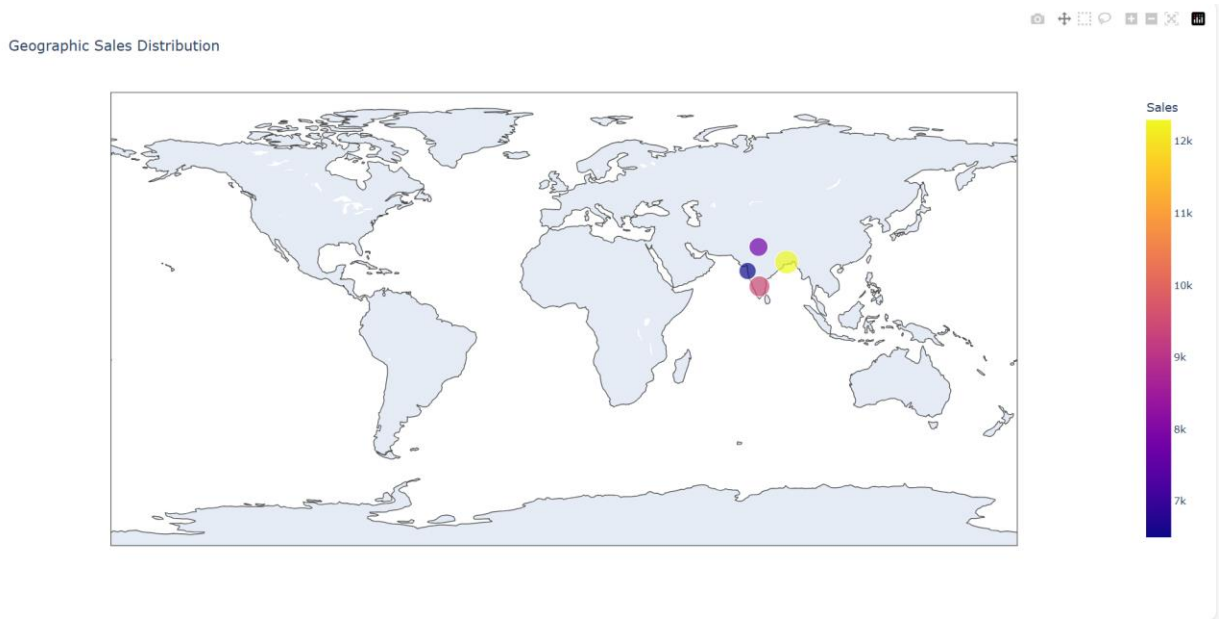
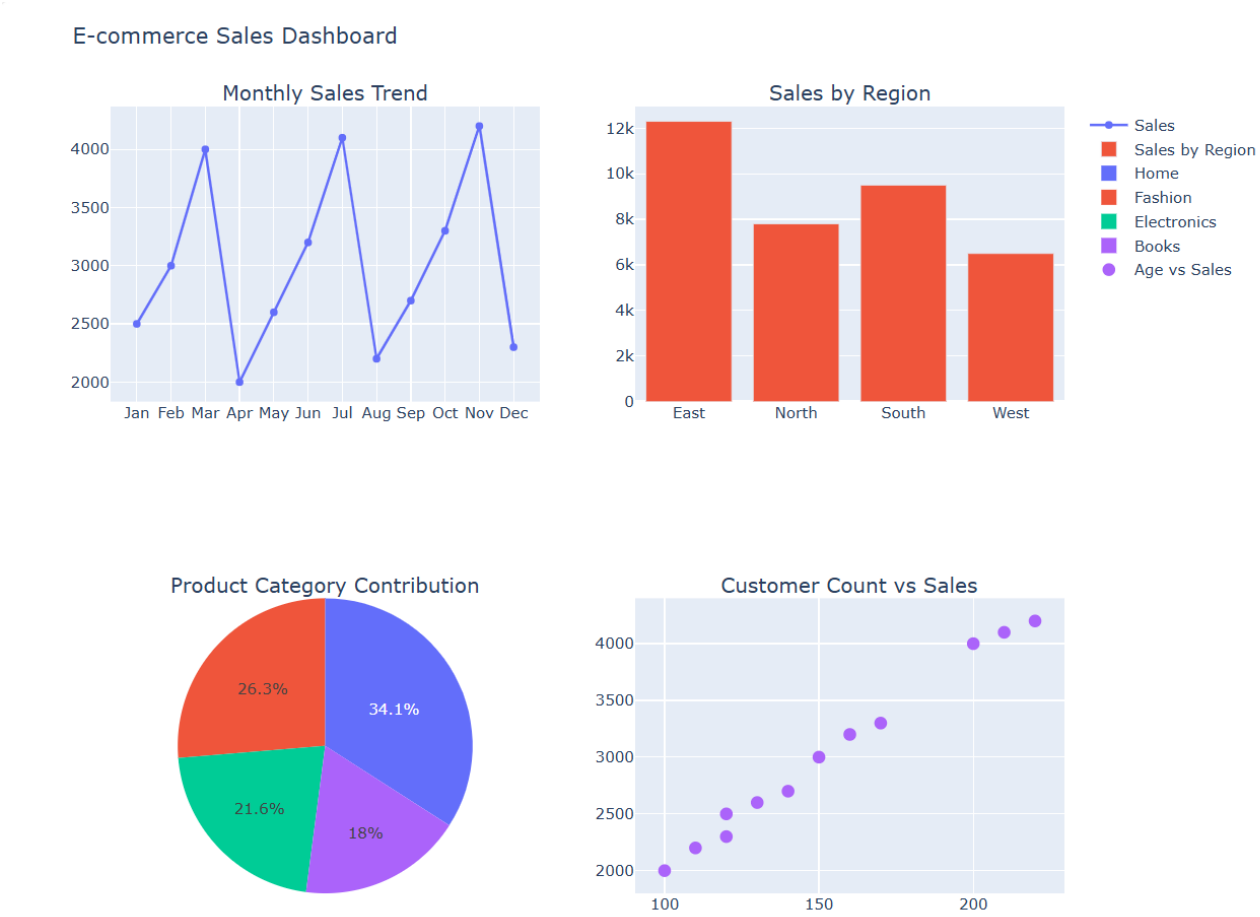


Figure 1
Sales by Region
Sales ($)
12000
10000
8000
6000
4000
2000
0
East
North
South
West
Region


Customer Count vs Sales by Age Group
Age_Group
18-25
26-35
36-45
46-55
Sales
Customer_Count
4000
3500
3000
2500
2000
100
120
140
160
180
200
220


Geographic Sales Distribution
Sales
12k
11k
10k
9k
8k
7k


E-commerce Sales Dashboard
Monthly Sales Trend
4000
3500
3000
2500
2000
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Sales by Region
12k
10k
8k
6k
4k
2k
0
East
North
South
West
Sales
Sales by Region
Home
Fashion
Electronics
Books
Age vs Sales
Product Category Contribution
26.3%
34.1%
21.6%
18%
Customer Count vs Sales
4000
3500
3000
2500
2000
100
150
200

Practical No: 10(c)

Aim: Present the findings and insights in a clear and concise manner.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
data = sns.load_dataset('titanic')
print("Summary Statistics:\n", data.describe(include='all'))
print("\nMissing Values:\n", data.isnull().sum())
age_survival = data.groupby('age')['survived'].mean().dropna()
plt.figure(figsize=(10, 6))
plt.plot(age_survival, marker='o')
plt.title('Survival Rate by Age')
plt.xlabel('Age')
plt.ylabel('Survival Rate')
plt.grid()
plt.show()
class_counts = data['class'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Passenger Class Distribution')
plt.show()
class_survival = data.groupby('class')['survived'].mean()
plt.figure(figsize=(8, 6))
class_survival.plot(kind='bar', color='skyblue')
plt.title('Survival Rate by Class')
plt.xlabel('Class')
```

```
plt.ylabel('Survival Rate')
plt.xticks(rotation=0)
plt.show()
num_data = data[['age', 'fare', 'survived']].dropna()
plt.figure(figsize=(10, 8))
sns.heatmap(num_data.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
diabetes = load_diabetes()
X = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
y = diabetes.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
score = model.score(X_test, y_test)
print("\nModel R-squared:", score)
predictions = model.predict(X_test)
print("\nPredictions (first 5):\n", predictions[:5])
peak_fare = num_data['fare'].max()
peak_age = num_data.loc[num_data['fare'].idxmax(), 'age']
plt.figure(figsize=(10, 6))
plt.scatter(num_data['age'], num_data['fare'], alpha=0.6, label='Passengers')
plt.title('Age vs Fare')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.annotate(f"Peak Fare: {peak_fare:.2f}", xy=(peak_age, peak_fare),
        xytext=(peak_age + 5, peak_fare - 50),
        arrowprops=dict(facecolor='red', shrink=0.05))
plt.grid()
plt.legend()
plt.show()
```

Output:

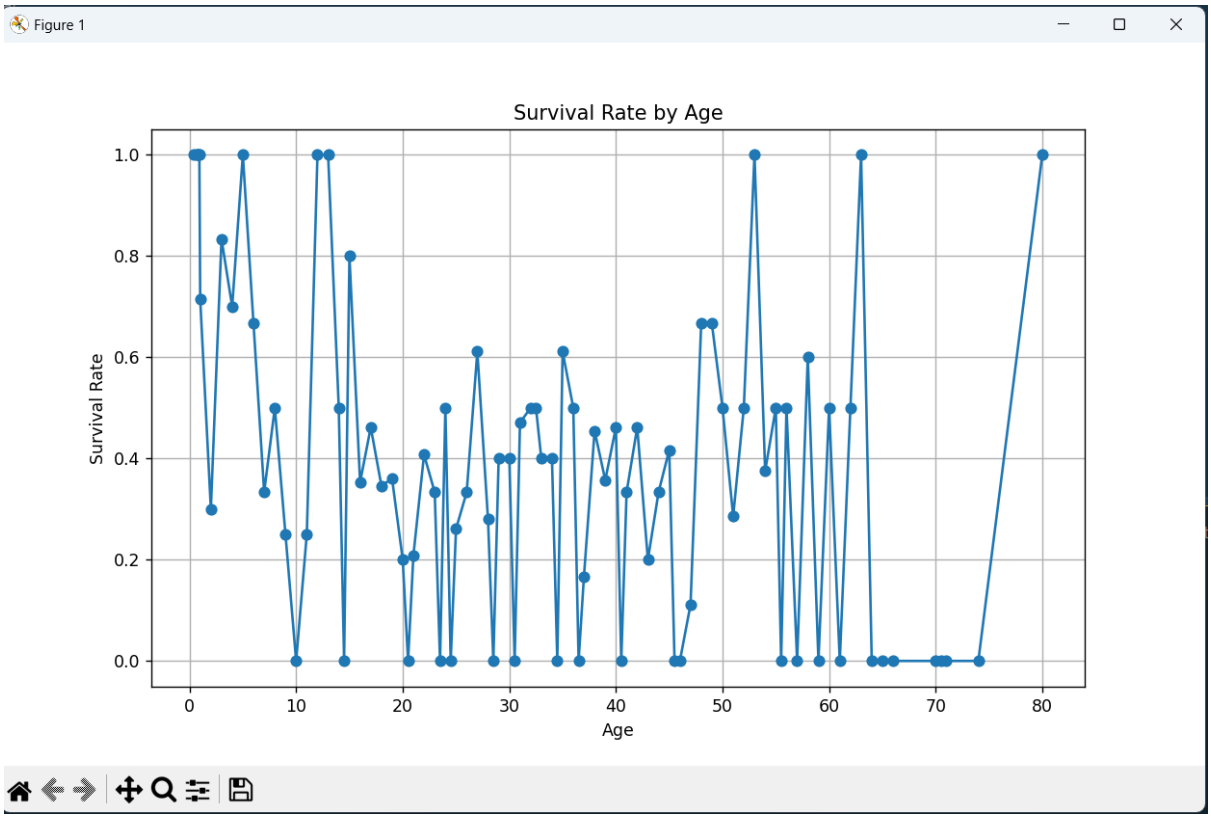


Figure 1
Survival Rate by Age
Survival Rate
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
80
Age

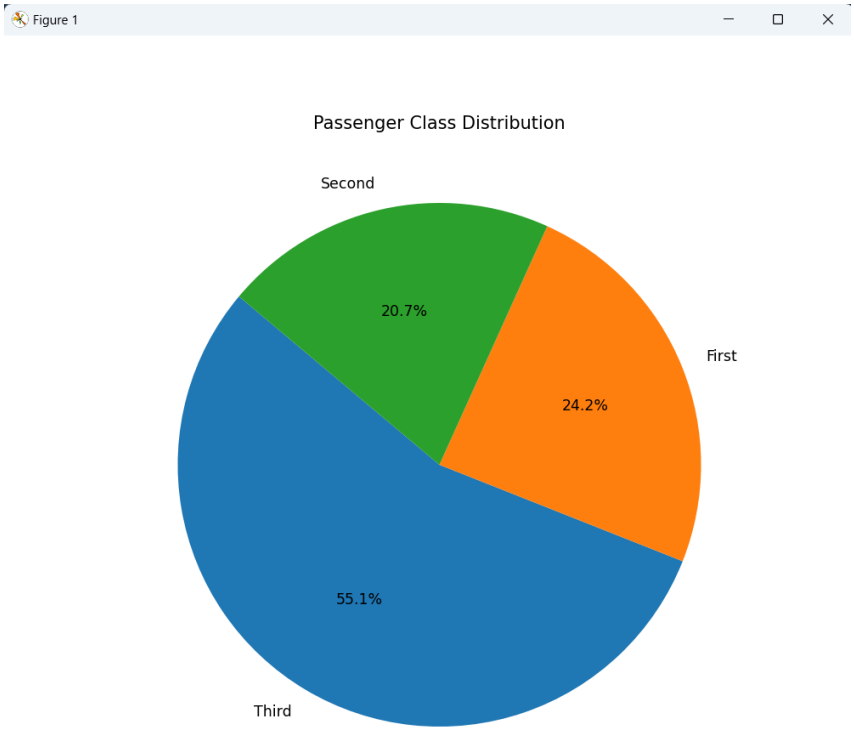


Figure 1
Passenger Class Distribution
Second
20.7%
First
24.2%
55.1%
Third

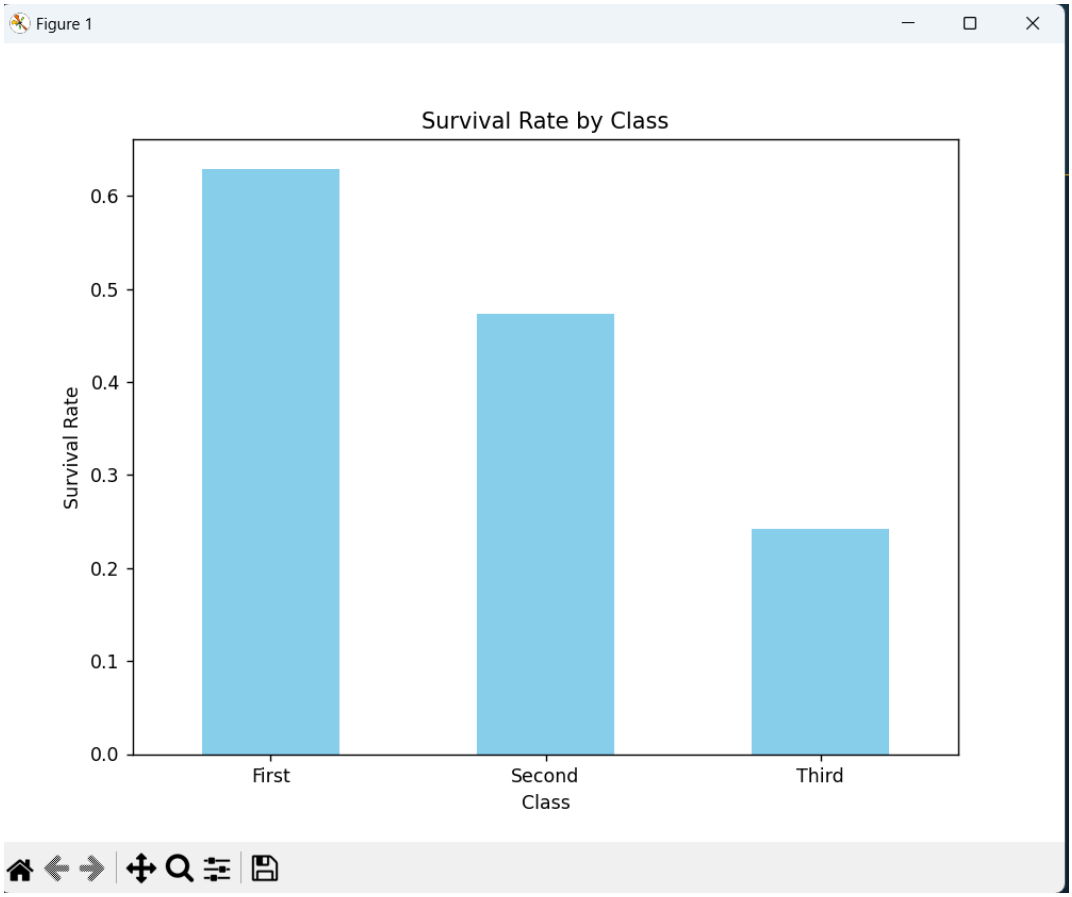


Figure 1
Survival Rate by Class
Survival Rate
0.0
0.1
0.2
0.3
0.4
0.5
0.6
First
Second
Third
Class

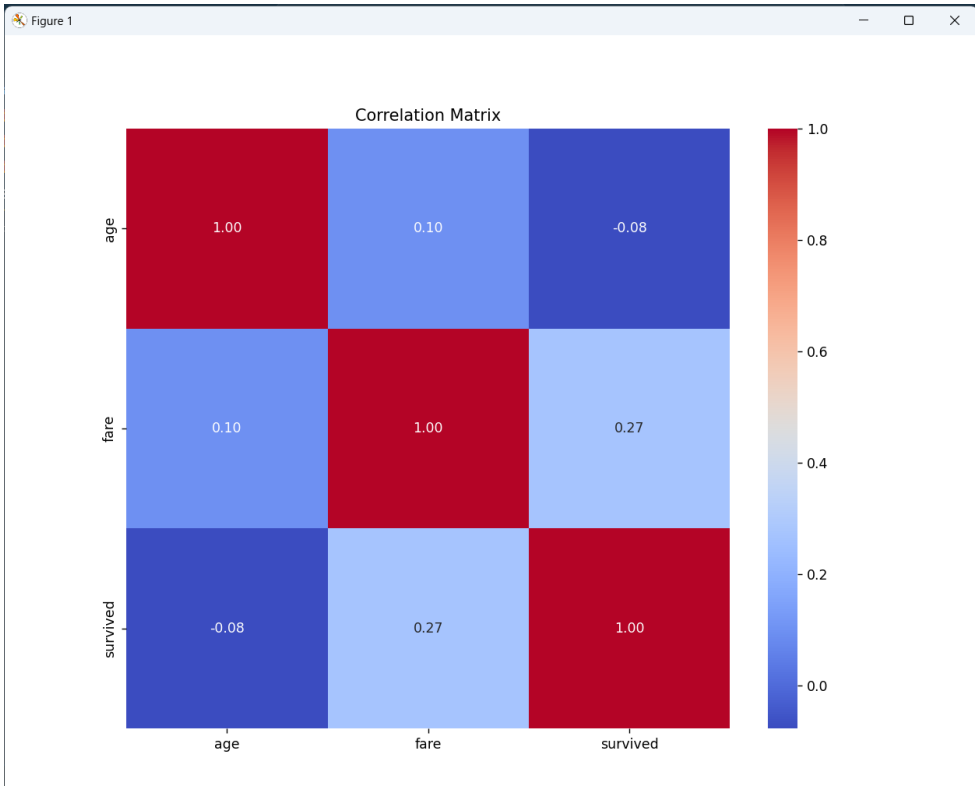
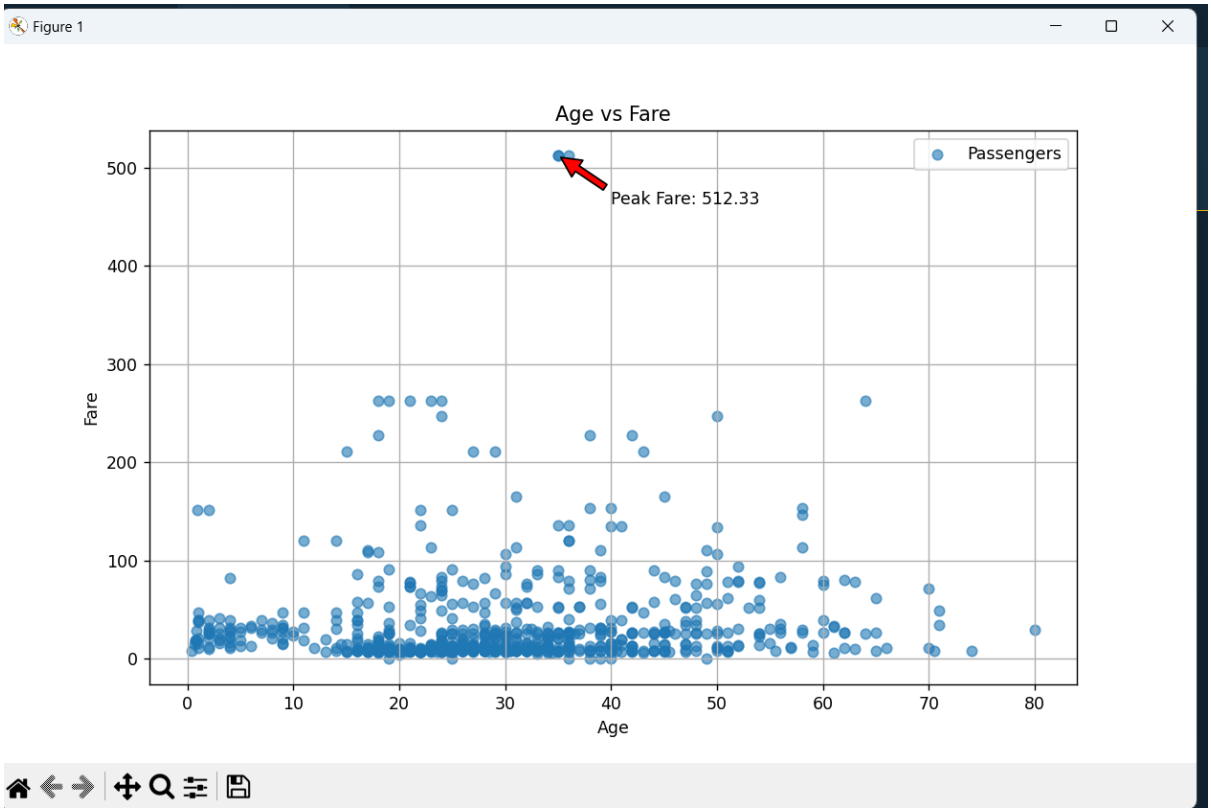


Figure 1
Correlation Matrix
age
fare
survived
1.00
0.10
-0.08
0.10
1.00
0.27
-0.08
0.27
1.00
1.0
0.8
0.6
0.4
0.2
0.0




```
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64

Model R-squared: 0.4526027629719195

Predictions (first 5):
 [139.5475584  179.51720835 134.03875572 291.41702925 123.78965872]
PS D:\TYBSC_CS_Data_Science_Practicals>
```