Practical No. 01

Aim: Introduction to Excel

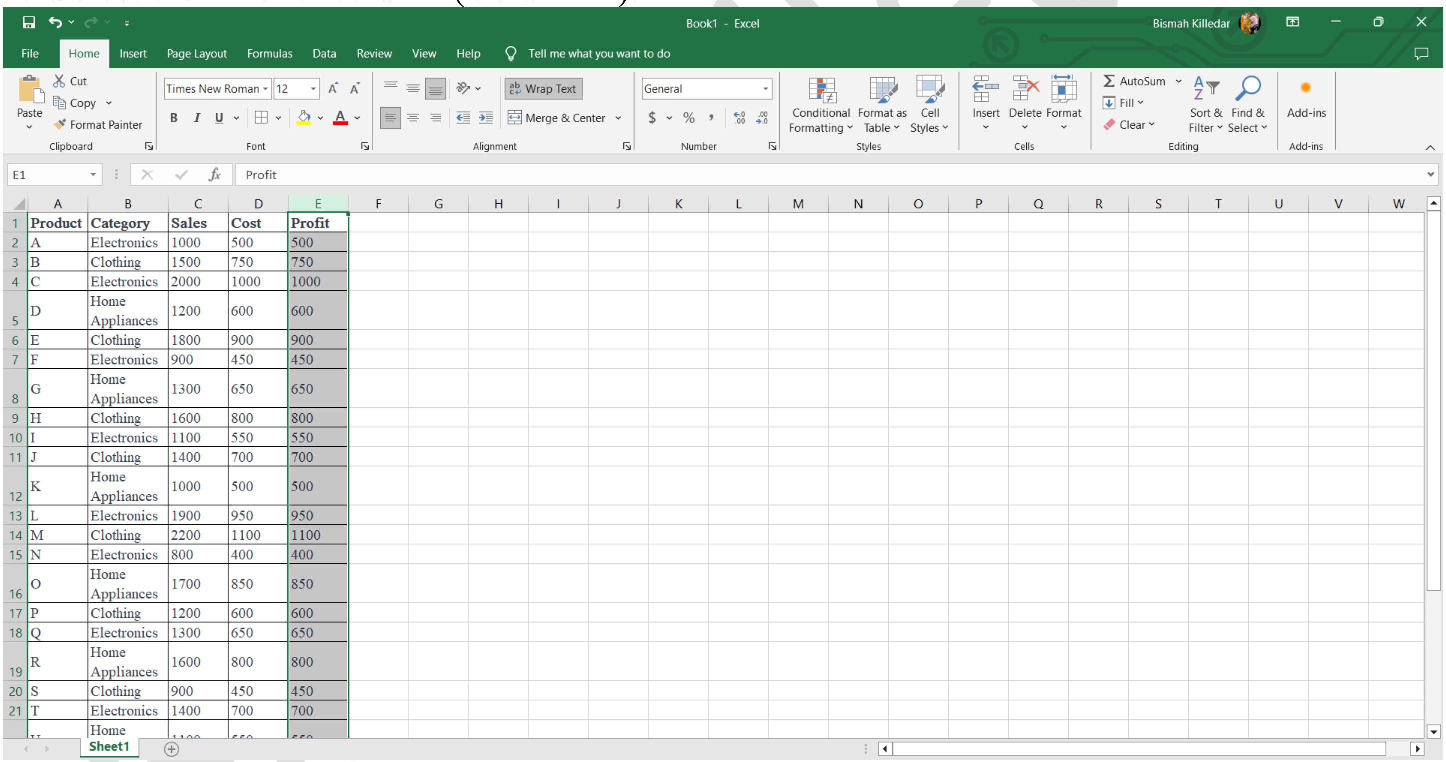
- Perform conditional formatting on a dataset using various criteria.
- Create a pivot table to analyze and summarize data.
- Use VLOOKUP function to retrieve information from a different worksheet or table.
- Perform what-if analysis using Goal Seek to determine input values for desired output.

➢ Perform conditional formatting on a dataset using various criteria.

We perform conditional formatting on the "Profit" column to highlight cells with a profit greater than 800 using following steps:

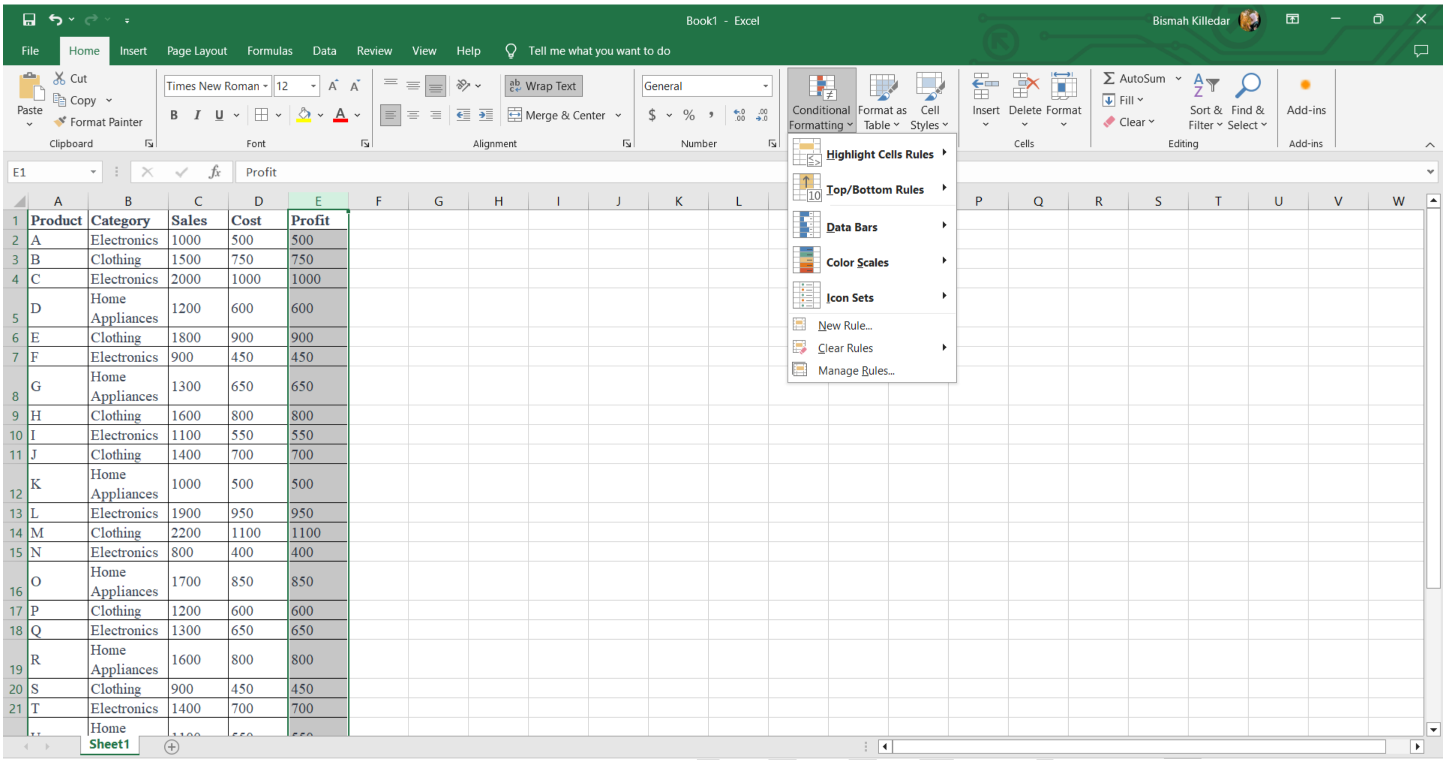
Steps:

1. Select the "Profit" column (Column E).

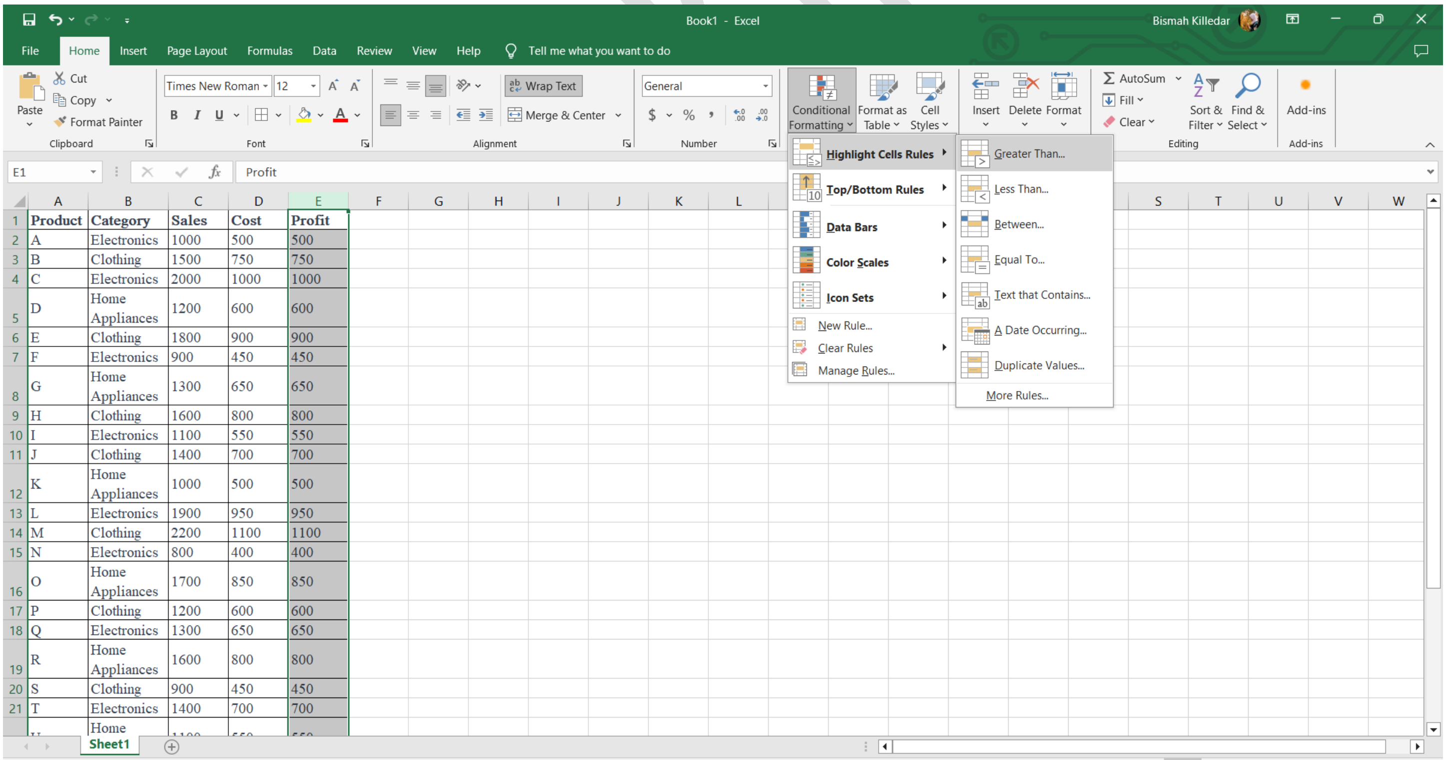


Product	Category	Sales	Cost	Profit
A	Electronics	1000	500	500
B	Clothing	1500	750	750
C	Electronics	2000	1000	1000
D	Home Appliances	1200	600	600
E	Clothing	1800	900	900
F	Electronics	900	450	450
G	Home Appliances	1300	650	650
H	Clothing	1600	800	800
I	Electronics	1100	550	550
J	Clothing	1400	700	700
K	Home Appliances	1000	500	500
L	Electronics	1900	950	950
M	Clothing	2200	1100	1100
N	Electronics	800	400	400
O	Home Appliances	1700	850	850
P	Clothing	1200	600	600
Q	Electronics	1300	650	650
R	Home Appliances	1600	800	800
S	Clothing	900	450	450
T	Electronics	1400	700	700

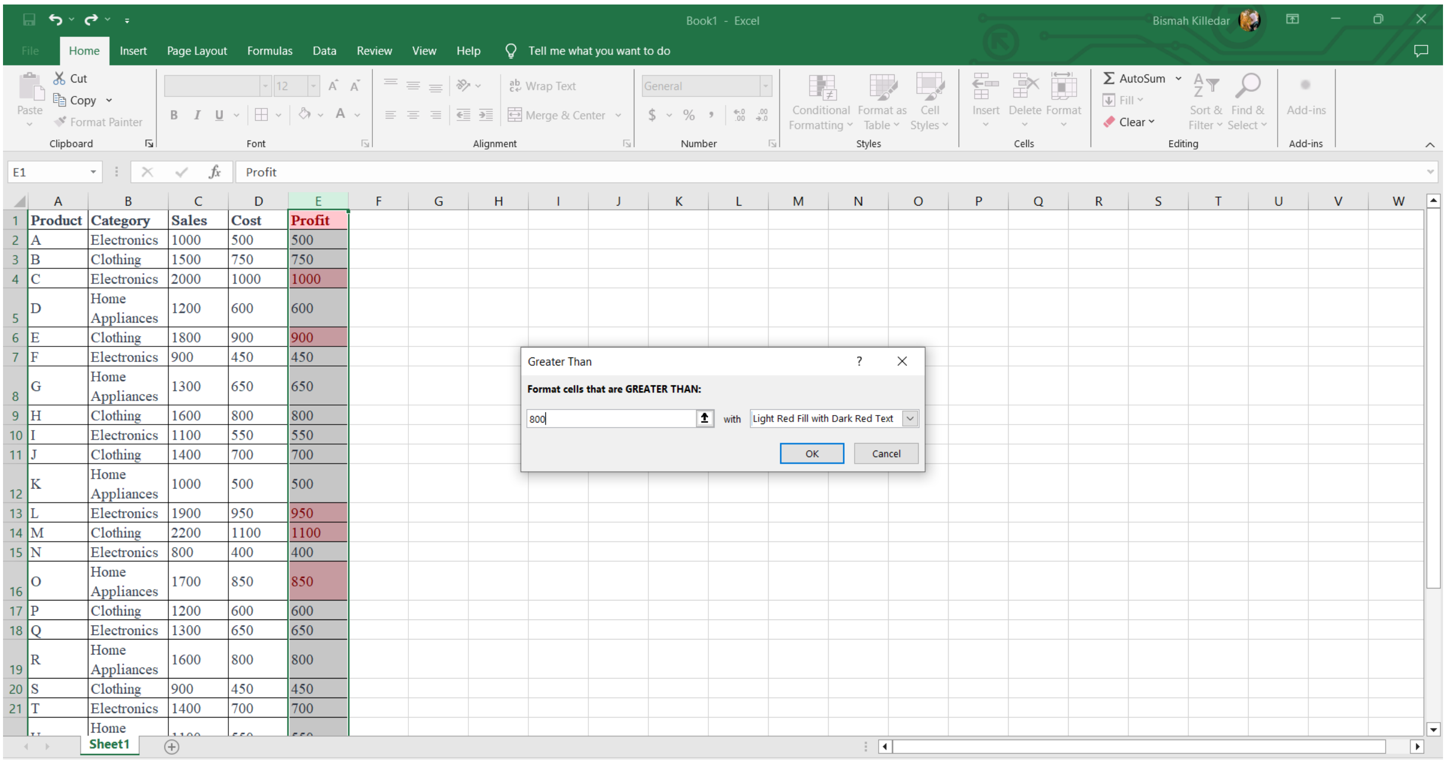
2. Go to the "Home" tab on the ribbon.
3. Click on "Conditional Formatting" in the toolbar.



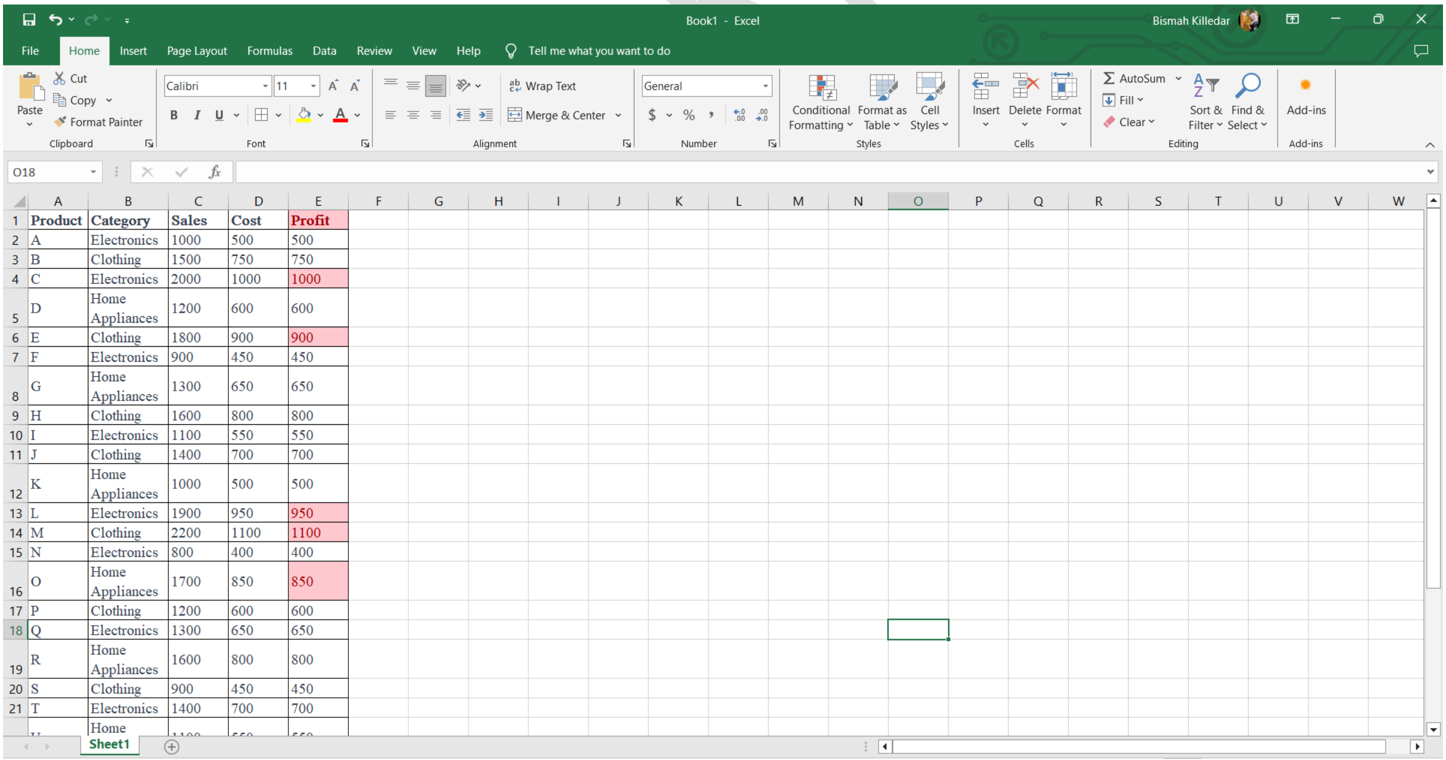
4. Choose "Highlight Cells Rules" and then "Greater Than."



5. Enter the threshold value as 800.



6. Customize the formatting options (e.g., choose a fill color).
7. Click "OK" to apply the rule.

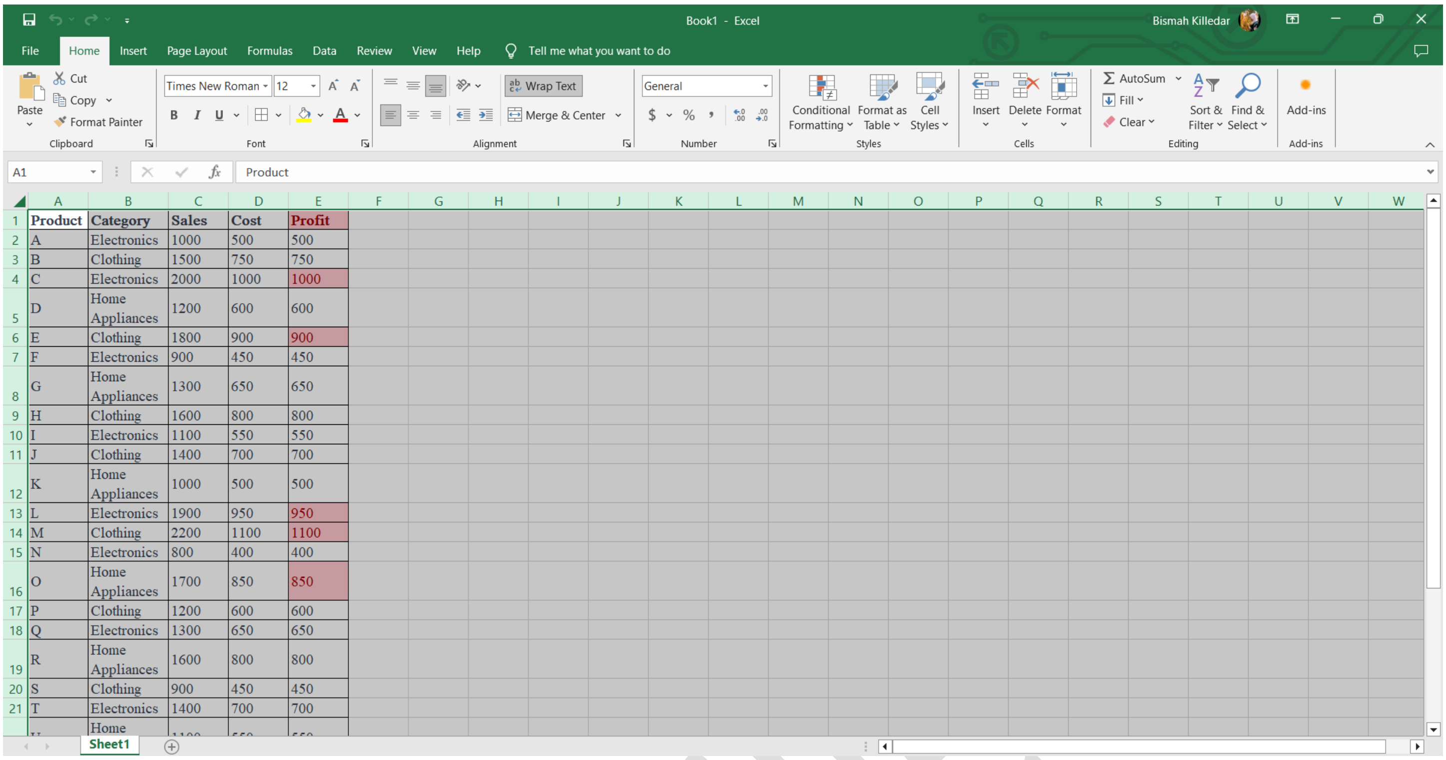


➢ Create a pivot table to analyze and summarize data.

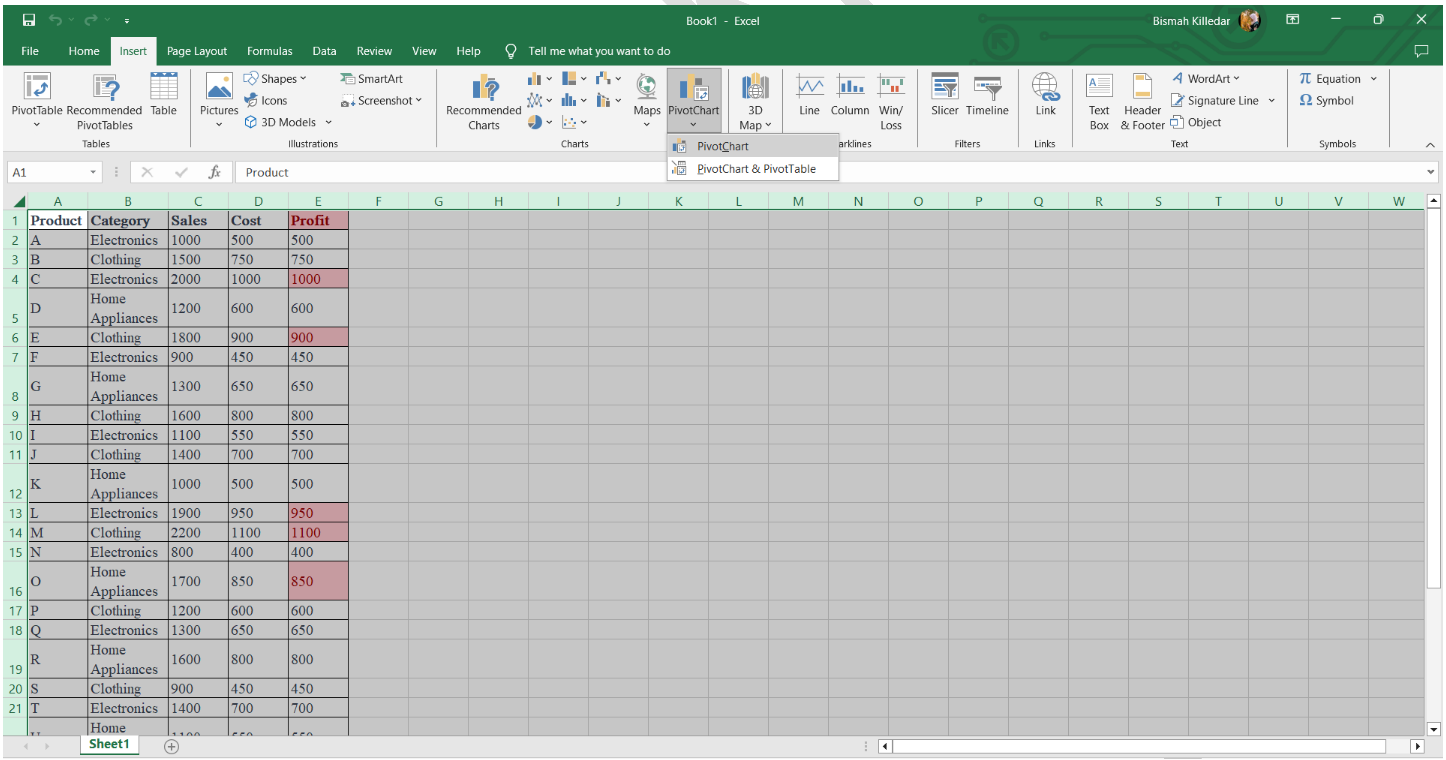
Following are the steps to create a pivot table to analyze total sales by category.

Steps:

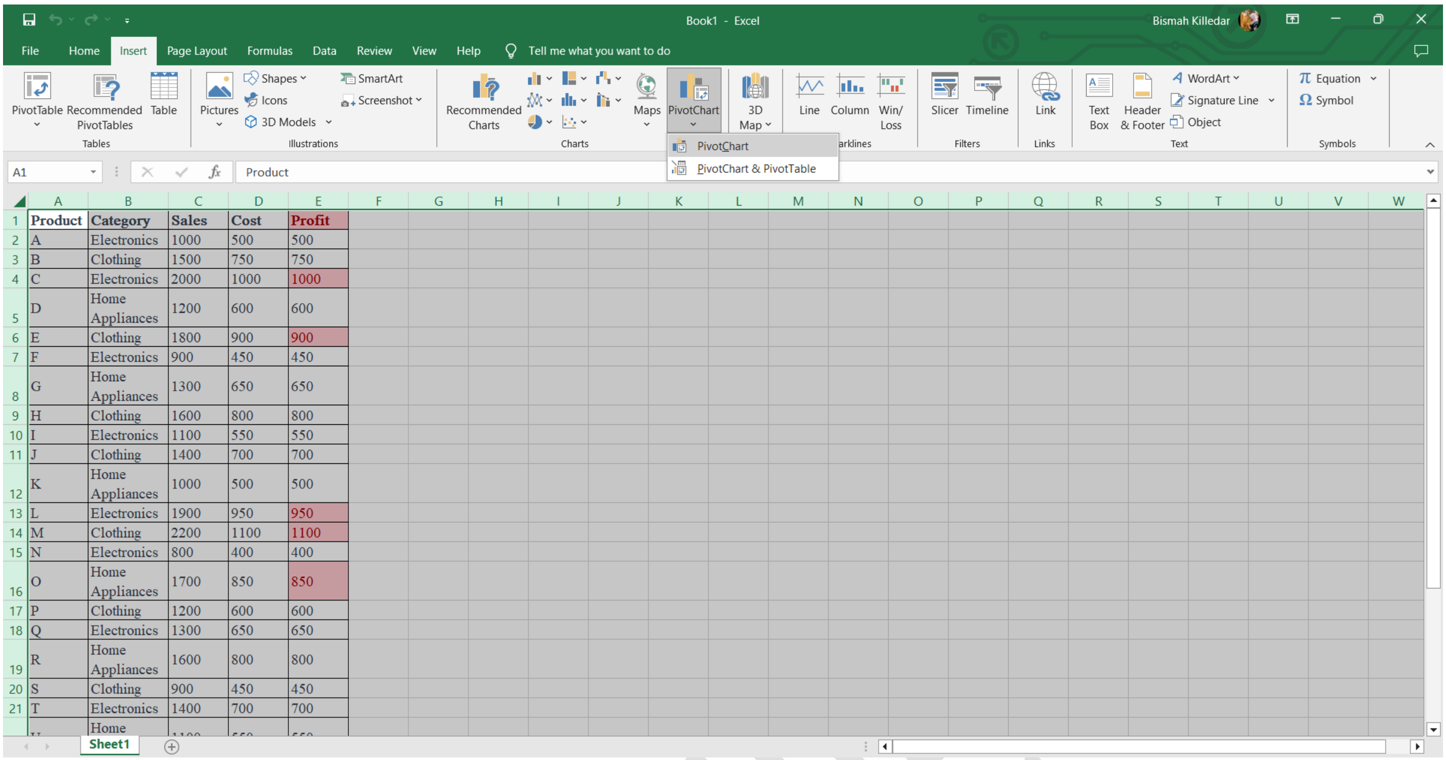
1. Select the entire dataset including headers.



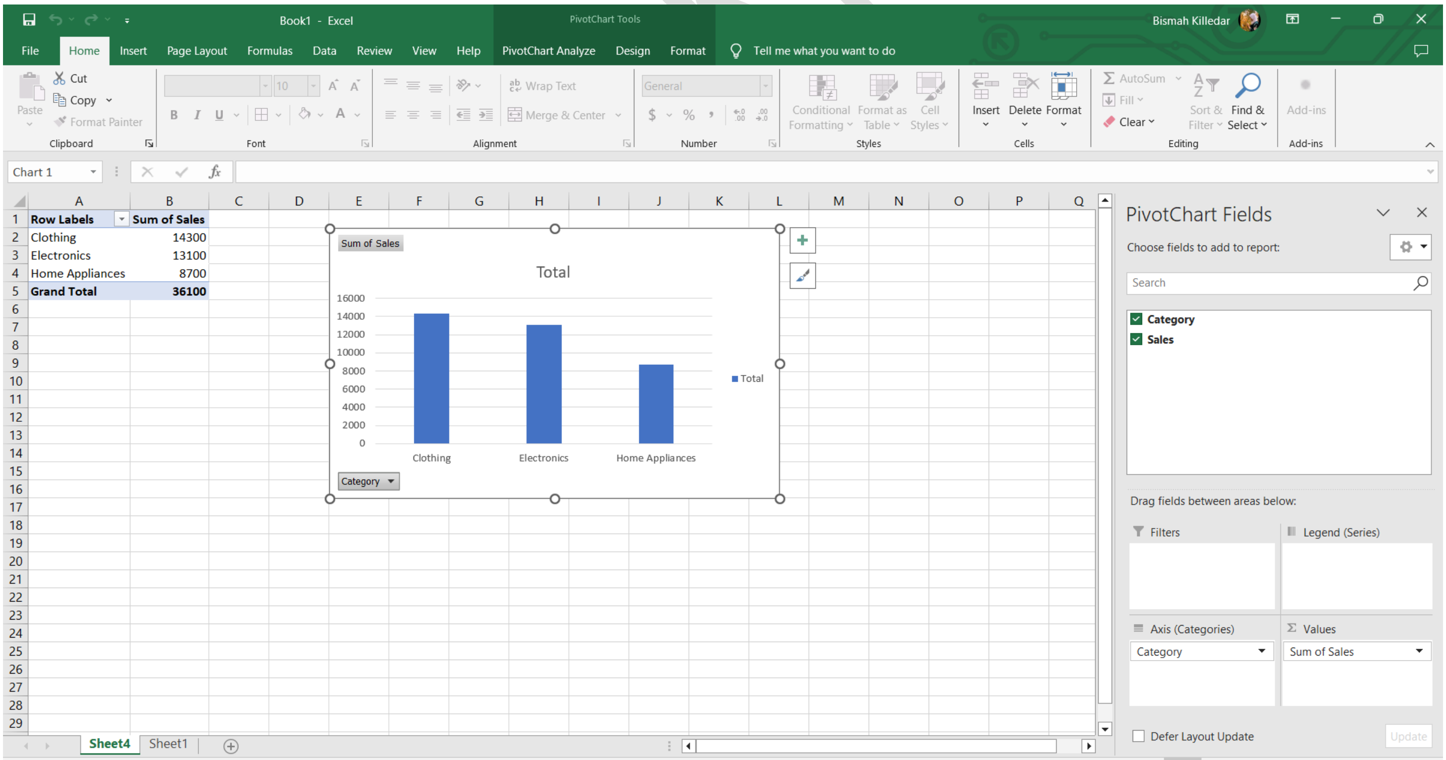
2. Go to the "Insert" tab on the ribbon.
3. Click on "PivotTable."



4. Choose where you want to place the PivotTable (e.g., new worksheet).



5. Drag "Category" to the Rows area.
6. Drag "Sales" to the Values area, choosing the sum function.

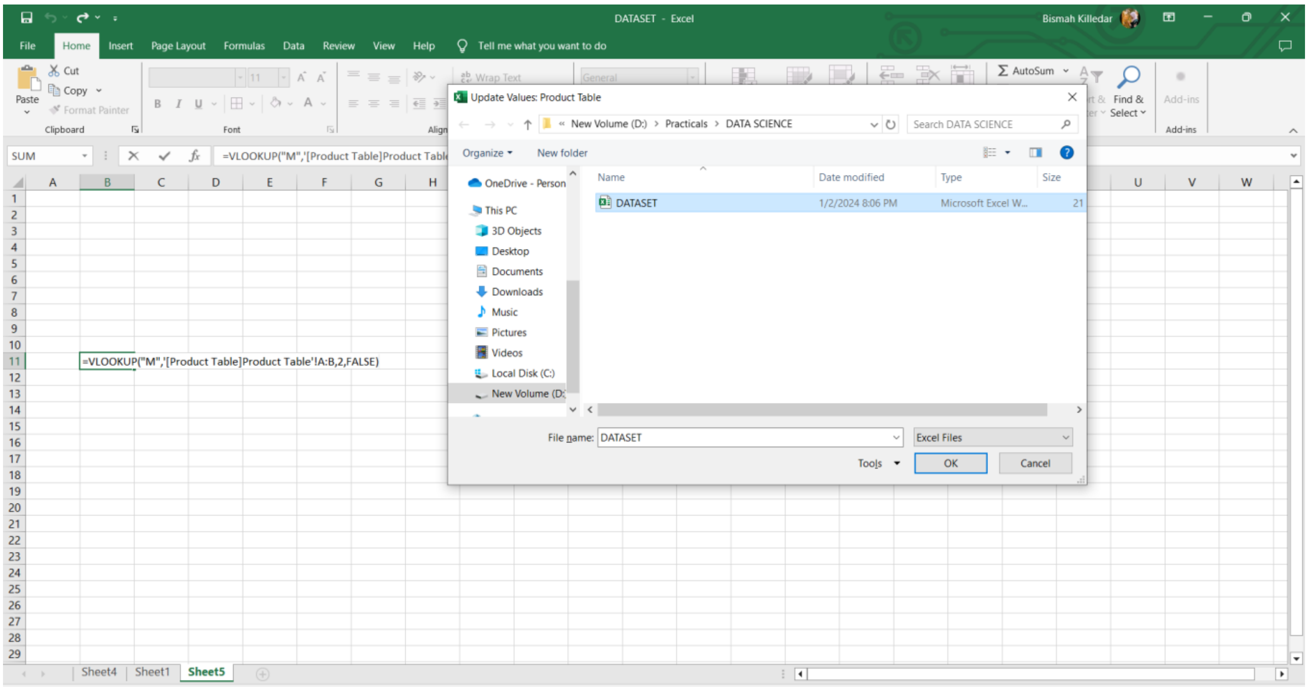


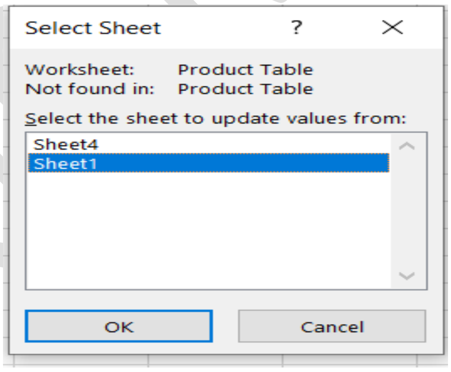
➢ Use VLOOKUP function to retrieve information from a different worksheet or table.

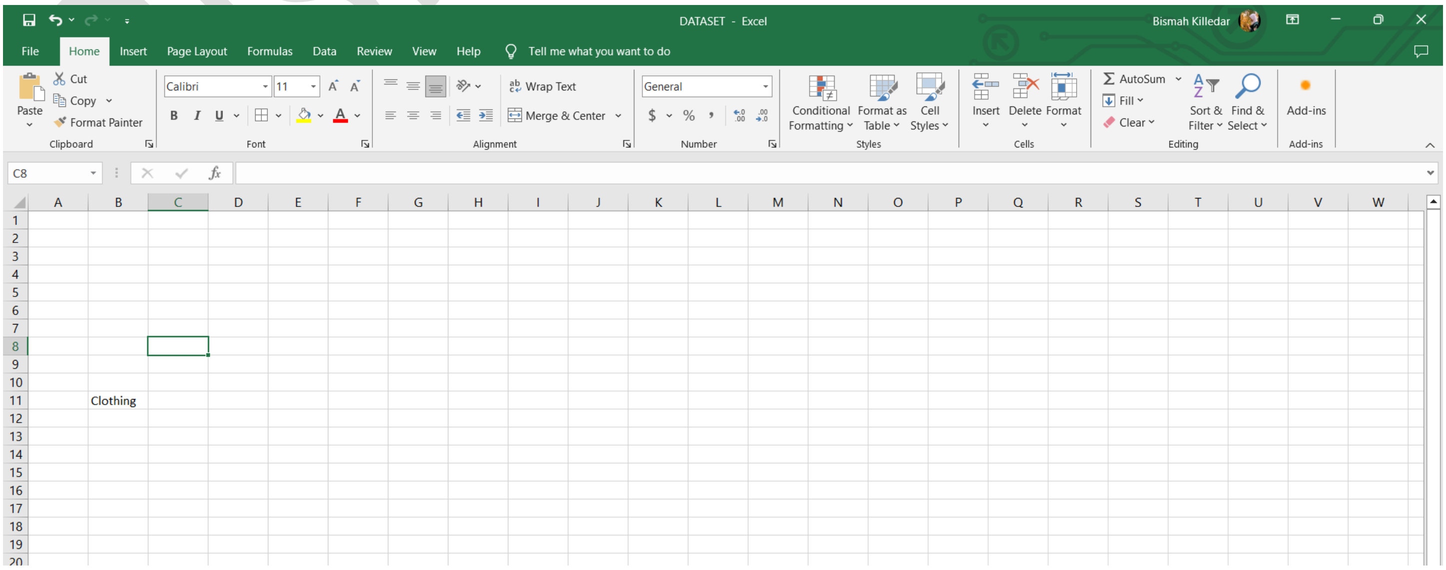
Use the VLOOKUP function to retrieve the category of "Product M" from a separate worksheet named "Product Table" using following steps:

Steps:

1. Assuming your "Product Table" is in a different worksheet.
2. In a cell in your main dataset, enter the formula:
   =VLOOKUP("M", 'Product Table'!A:B, 2, FALSE)





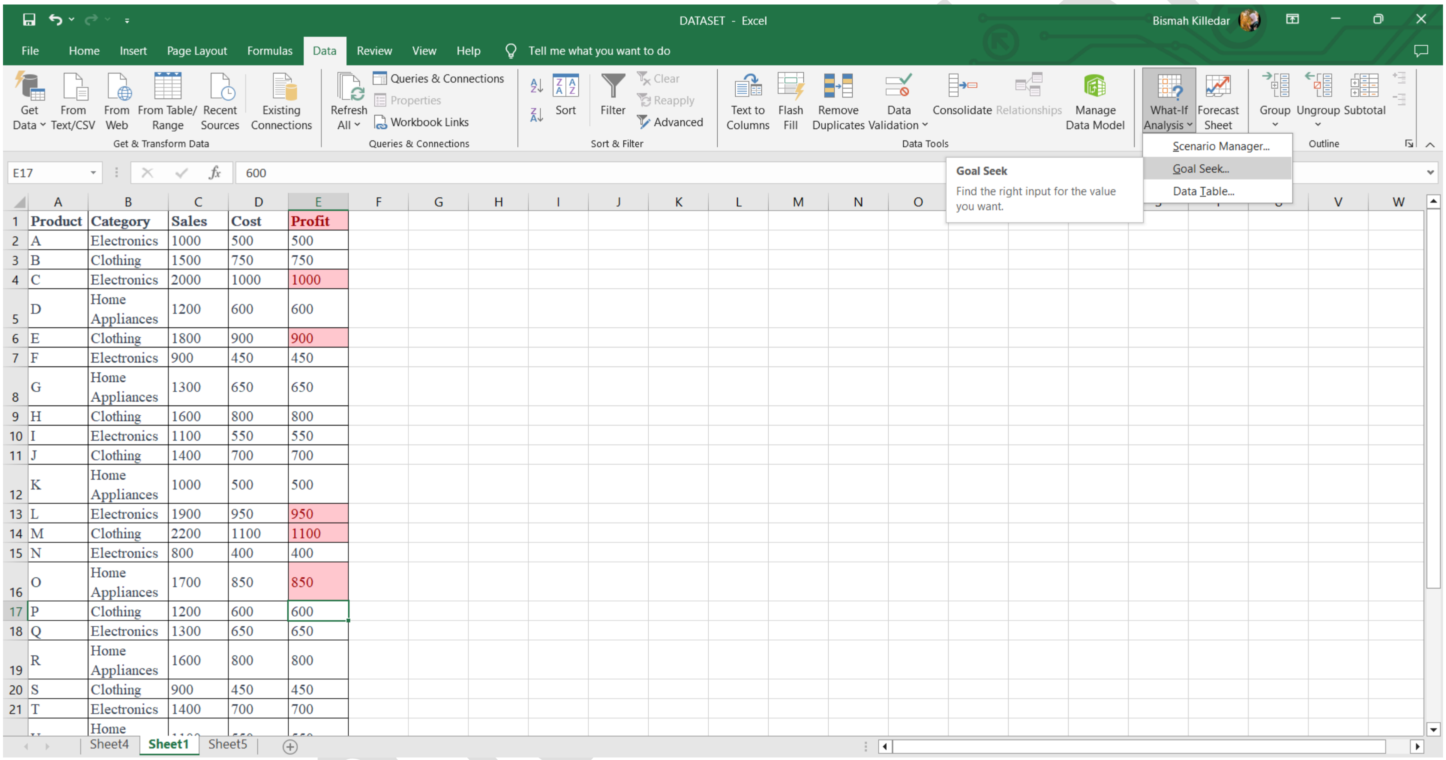


➢ Perform what-if analysis using Goal Seek to determine input values for desired output.

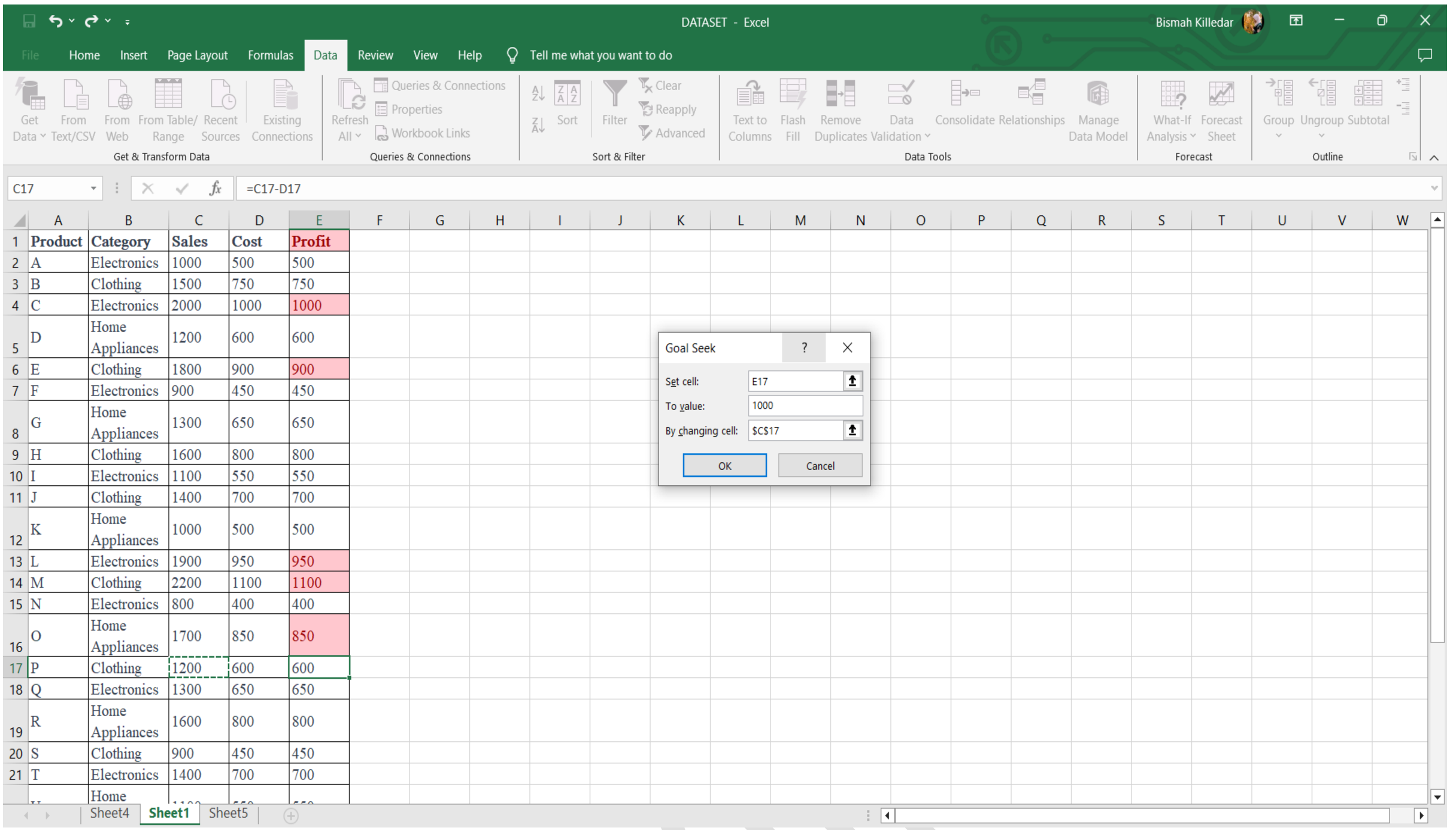
Use Goal Seek to find the required sales for "Product P" to achieve a profit of 1000 using the following steps.

Steps:

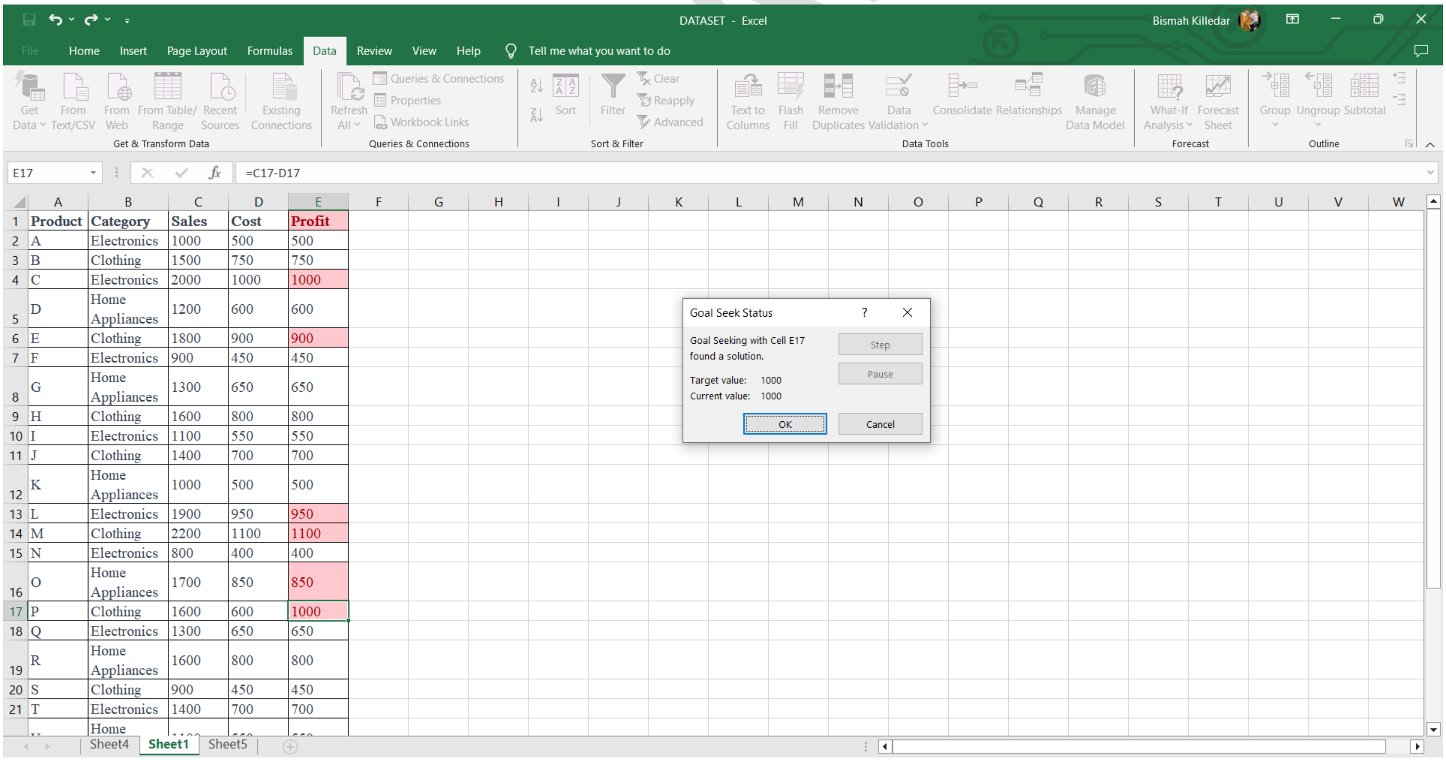
1. Identify the cell containing the formula for "Profit" for "Product P" (let's assume it's in cell E17).
2. Go to the "Data" tab on the ribbon.
3. Click on "What-If Analysis" and select "Goal Seek."



4. Set "Set cell" to the profit cell (E17), "To value" to 1000, and "By changing cell" to the sales cell (C17).



5. Click "OK" to let Excel determine the required sales.



Practical No.02

Aim: Data Frames and Basic Data Pre-processing

- Read data from CSV and JSON files into a data frame.
- Perform basic data pre-processing tasks such as handling missing values and outliers.
- Manipulate and transform data using functions like filtering, sorting, and grouping.

Data pre-processing:

Data pre-processing is a crucial step in the data analysis pipeline, encompassing tasks such as reading data from various file formats, handling missing values, and managing outliers. This practical guide explores how to execute these tasks using the pandas library in Python.

Steps:

Step 1: Reading from CSV and JSON Files

1. Utilize pandas to read data from a CSV file ('DATA SET.csv') into a data frame.
2. Use pandas to read data from a JSON file ('ds.json') into a data frame.
3. Display the first few rows of each data frame to inspect the data.

Step 2: Handling Missing Values

1. Drop rows with missing values from the CSV data frame.
2. Fill missing values with a specific value (e.g., 0) in the JSON data frame.

Step 3: Handling Outliers

1. Identify outliers in the 'Sales' column of the CSV data frame.
2. Replace outliers with the median value.

Step 4: Manipulating and Transforming Data

1. Filter the CSV data frame to include only rows where 'Sales' is greater than 10.
2. Sort the CSV data frame based on the 'Sales' column in descending order.
3. Group the CSV data frame by the 'Category' column and calculate the mean for numeric columns ('Sales', 'Cost', 'Profit').

Step 5: Displaying Results

1. Display the cleaned CSV data frame after handling missing values.
2. Display the JSON data frame after filling missing values.
3. Display the filtered CSV data frame.
4. Display the sorted CSV data frame.
5. Display the grouped CSV data frame showing the mean values for numeric columns.

Code:

```
import pandas as pd
# Read data from CSV file into a data frame
csv_file_path = 'DATA SET.csv'
df_csv = pd.read_csv(csv_file_path)
# Read data from JSON file into a data frame
json_file_path = 'ds.json'
df_json = pd.read_json(json_file_path)
# Display the first few rows of each data frame to inspect the data
```

```
print("CSV Data:")
print(df_csv.head())
print("\nJSON Data:")
print(df_json.head())
# Handling missing values
# Drop rows with missing values
df_csv_cleaned = df_csv.dropna()
# Fill missing values with a specific value (e.g., 0)
df_json_filled = df_json.fillna(0)
# Handling outliers
# Assume 'Sales' is the column with outliers
# Replace outliers with the median
median_value = df_csv['Sales'].median()
upper_threshold = df_csv['Sales'].mean() + 2 * df_csv['Sales'].std()
lower_threshold = df_csv['Sales'].mean() - 2 * df_csv['Sales'].std()
df_csv['Sales'] = df_csv['Sales'].apply(lambda x: median_value if x > upper_threshold or x < lower_threshold else x)
# Manipulate and transform data
# Filtering
filtered_data = df_csv[df_csv['Sales'] > 10]
# Sorting
sorted_data = df_csv.sort_values(by='Sales', ascending=False)
# Grouping and calculating mean for numeric columns
numeric_columns = ['Sales', 'Cost', 'Profit']
grouped_data = df_csv.groupby('Category')[numeric_columns].mean()
# Display the results
print("\nCleaned CSV Data:")
print(df_csv_cleaned.head())
print("\nFilled JSON Data:")
print(df_json_filled.head())
print("\nFiltered Data:")
print(filtered_data.head())
print("\nSorted Data:")
print(sorted_data.head())
print("\nGrouped Data:")
print(grouped_data.head())
```

Output:

```
CSV Data:
  Product         Category  Sales  Cost  Profit
0       A      Electronics   1000   500     500
1       B         Clothing   1500   750     750
2       C      Electronics   2000  1000    1000
3       D  Home Appliances   1200   600     600
4       E         Clothing   1800   900     900

JSON Data:
  Product         Category  Sales  Cost  Profit
0       A      Electronics   1000   500     500
1       B         Clothing   1500   750     750
2       C      Electronics   2000  1000    1000
3       D  Home Appliances   1200   600     600
4       E         Clothing   1800   900     900

Cleaned CSV Data:
  Product         Category  Sales  Cost  Profit
0       A      Electronics   1000   500     500
1       B         Clothing   1500   750     750
2       C      Electronics   2000  1000    1000
3       D  Home Appliances   1200   600     600
4       E         Clothing   1800   900     900

Filled JSON Data:
  Product         Category  Sales  Cost  Profit
0       A      Electronics   1000   500     500
1       B         Clothing   1500   750     750
2       C      Electronics   2000  1000    1000
3       D  Home Appliances   1200   600     600
4       E         Clothing   1800   900     900

Filtered Data:
  Product         Category   Sales  Cost  Profit
0       A      Electronics  1000.0   500     500
1       B         Clothing  1500.0   750     750
2       C      Electronics  2000.0  1000    1000
3       D  Home Appliances  1200.0   600     600
4       E         Clothing  1800.0   900     900

Sorted Data:
   Product     Category   Sales  Cost  Profit
2        C  Electronics  2000.0  1000    1000
21       V     Clothing  2000.0  1000    1000
11       L  Electronics  1900.0   950     950
4        E     Clothing  1800.0   900     900
23       X     Clothing  1700.0   850     850

Grouped Data:
                       Sales        Cost      Profit
Category
Clothing         1544.444444  794.444444  838.888889
Electronics      1310.000000  655.000000  655.000000
Home Appliances  1242.857143  621.428571  621.428571
```

Practical No.03

Aim: Feature Scaling and Dummification

- Apply feature-scaling techniques like standardization and normalization to numerical features.
- Perform feature dummification to convert categorical variables into numerical representations.

Feature Scaling:

Feature scaling is a preprocessing technique used to standardize the range of independent variables or features of the data. It is essential for certain machine learning algorithms that are sensitive to the scale of input features, ensuring that all features contribute equally to the learning process.

Feature Dummification:

Feature dummification or one-hot encoding is a technique used to convert categorical variables into numerical representations. This is necessary because many machine learning algorithms require numerical input, and representing categorical variables as binary vectors helps maintain their information.

Steps:

1. **Load and Explore Data:** Load the dataset and explore its structure, identify numeric and categorical features.
2. **Feature Scaling:** Apply standardization and normalization to numeric features.
3. **Feature Dummification:** Convert categorical variables into numerical representations using one-hot encoding.
4. **Combine Features:** Combine scaled numeric features with one-hot encoded categorical features.
5. **Display Resulting Dataset:** Display the final dataset after both feature scaling and dummification.

Code:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
# Define the data
data = {
    'Product': ['Apple_Juice', 'Banana_Smoothie', 'Orange_Jam', 'Grape_Jelly', 'Kiwi_Parfait',
'Mango_Chutney', 'Pineapple_Sorbet', 'Strawberry_Yogurt', 'Blueberry_Pie', 'Cherry_Salsa'],
   'Category': ['Apple', 'Banana', 'Orange', 'Grape', 'Kiwi', 'Mango', 'Pineapple', 'Strawberry',
'Blueberry', 'Cherry'],
```

```
    'Sales': [1200, 1700, 2200, 1400, 2000, 1000, 1500, 1800, 1300, 1600],
    'Cost': [600, 850, 1100, 700, 1000, 500, 750, 900, 650, 800],
    'Profit': [600, 850, 1100, 700, 1000, 500, 750, 900, 650, 800]
}
# Create a DataFrame
df = pd.DataFrame(data)
# Display the original dataset
print("Original Dataset:")
print(df)
# Step 1: Feature Scaling (Standardization and Normalization)
numeric_columns = ['Sales', 'Cost', 'Profit']
scaler_standardization = StandardScaler()
scaler_normalization = MinMaxScaler()
df_scaled_standardized =
pd.DataFrame(scaler_standardization.fit_transform(df[numeric_columns]),
columns=numeric_columns)
df_scaled_normalized =
pd.DataFrame(scaler_normalization.fit_transform(df[numeric_columns]),
columns=numeric_columns)
# Combine the scaled numeric features with the categorical features
df_scaled = pd.concat([df_scaled_standardized, df.drop(numeric_columns, axis=1)], axis=1)
# Display the dataset after feature scaling
print("\nDataset after Feature Scaling:")
print(df_scaled)
# Step 2: Feature Dummification
# Identify categorical columns
categorical_columns = ['Product', 'Category']
# Create a column transformer for dummification
preprocessor = ColumnTransformer(
    transformers=[
        ('categorical', OneHotEncoder(), categorical_columns)
    ],
    remainder='passthrough'
)
# Apply the column transformer to the dataset
df_dummified = pd.DataFrame(preprocessor.fit_transform(df))
# Display the dataset after feature dummification
print("\nDataset after Feature Dummification:")
print(df_dummified)
```

Output:

```
IDLE Shell 3.11.3
File  Edit  Shell  Debug  Options  Window  Help
================= RESTART: D:\Practicals\DATA SCIENCE\prac 3.py ================
Original Dataset:
              Product    Category  Sales  Cost  Profit
0         Apple_Juice       Apple   1200   600     600
1     Banana_Smoothie      Banana   1700   850     850
2          Orange_Jam      Orange   2200  1100    1100
3         Grape_Jelly       Grape   1400   700     700
4        Kiwi_Parfait        Kiwi   2000  1000    1000
5       Mango_Chutney       Mango   1000   500     500
6    Pineapple_Sorbet   Pineapple   1500   750     750
7   Strawberry_Yogurt  Strawberry   1800   900     900
8       Blueberry_Pie   Blueberry   1300   650     650
9        Cherry_Salsa      Cherry   1600   800     800

Dataset after Feature Scaling:
      Sales      Cost    Profit            Product    Category
0 -1.058873 -1.058873 -1.058873        Apple_Juice       Apple
1  0.372036  0.372036  0.372036    Banana_Smoothie      Banana
2  1.802946  1.802946  1.802946         Orange_Jam      Orange
3 -0.486509 -0.486509 -0.486509        Grape_Jelly       Grape
4  1.230582  1.230582  1.230582       Kiwi_Parfait        Kiwi
5 -1.631237 -1.631237 -1.631237      Mango_Chutney       Mango
6 -0.200327 -0.200327 -0.200327   Pineapple_Sorbet   Pineapple
7  0.658218  0.658218  0.658218  Strawberry_Yogurt  Strawberry
8 -0.772691 -0.772691 -0.772691      Blueberry_Pie   Blueberry
9  0.085855  0.085855  0.085855       Cherry_Salsa      Cherry

Dataset after Feature Dummification:
                                                  0
0    (0, 0)\t1.0\n  (0, 10)\t1.0\n  (0, 20)\t1200...
1    (0, 1)\t1.0\n  (0, 11)\t1.0\n  (0, 20)\t1700...
2    (0, 7)\t1.0\n  (0, 17)\t1.0\n  (0, 20)\t2200...
3    (0, 4)\t1.0\n  (0, 14)\t1.0\n  (0, 20)\t1400...
4    (0, 5)\t1.0\n  (0, 15)\t1.0\n  (0, 20)\t2000...
5    (0, 6)\t1.0\n  (0, 16)\t1.0\n  (0, 20)\t1000...
6    (0, 8)\t1.0\n  (0, 18)\t1.0\n  (0, 20)\t1500...
7    (0, 9)\t1.0\n  (0, 19)\t1.0\n  (0, 20)\t1800...
8    (0, 2)\t1.0\n  (0, 12)\t1.0\n  (0, 20)\t1300...
9    (0, 3)\t1.0\n  (0, 13)\t1.0\n  (0, 20)\t1600...
>>>
                                                        Ln: 141  Col: 0
```

Practical No.04

Aim: Hypothesis Testing

- Formulate null and alternative hypotheses for a given problem.
- Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chi-square test).
- Interpret the results and draw conclusions based on the test outcomes.

Hypothesis Testing:

Hypothesis testing is a statistical method used to make inferences about population parameters based on sample data. It involves the formulation of a null hypothesis (H0) and an alternative hypothesis (H1), and the collection of sample data to assess the evidence against the null hypothesis. The goal is to determine whether there is enough evidence to reject the null hypothesis in favor of the alternative hypothesis.

1. Formulate Null and Alternative Hypotheses:

- Null Hypothesis (H0): The average productivity levels of employees who underwent the training program are the same as those who did not ($\mu1 = \mu2$).
- Alternative Hypothesis (H1): The average productivity levels of employees who underwent the training program are different from those who did not ($\mu1 \neq \mu2$).

2. Data Collection:

- Randomly select two groups of employees: one group that underwent the training program (Sample 1) and another that did not (Sample 2).
- Measure the productivity levels of each employee in both groups.

3. Conduct Hypothesis Test:

- Perform a two-sample t-test to compare the means of the two groups.
- Set the significance level (α) to 0.05.

4. Analyze Results:

- Print the results of the t-test, including the t-statistic, p-value, and degrees of freedom.
- Visualize the distributions of Sample 1 and Sample 2 using histograms.
- Highlight the critical region if the p-value is less than the significance level.

5. Draw Conclusions:

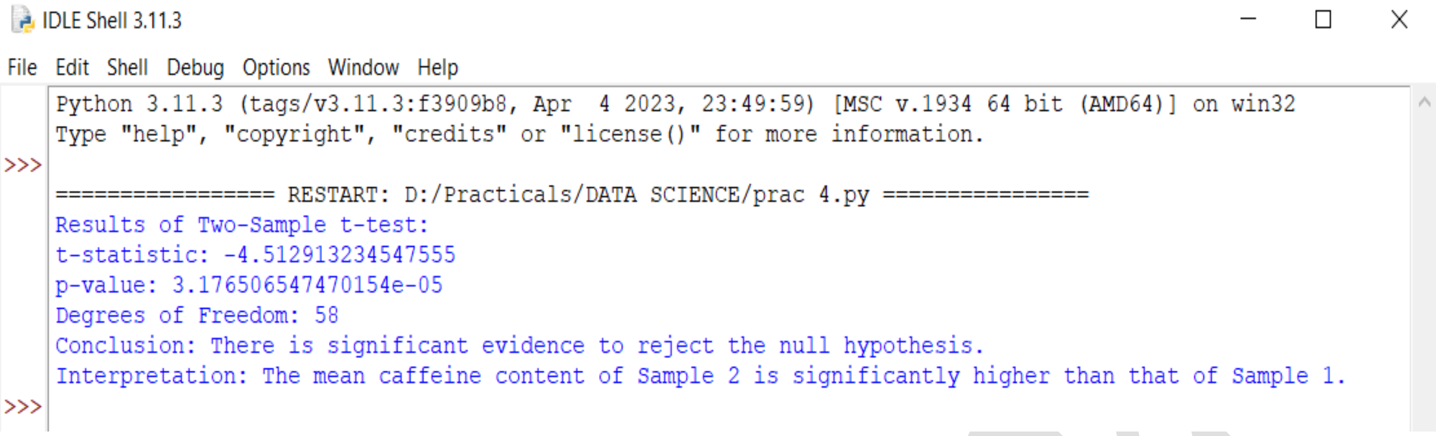
- If p-value $< \alpha$, reject the null hypothesis.
- If p-value $\geq \alpha$, fail to reject the null hypothesis.

Code:

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
# Generate two samples for demonstration purposes
np.random.seed(42)
sample1 = np.random.normal(loc=10, scale=2, size=30)
```

```
sample2 = np.random.normal(loc=12, scale=2, size=30)
# Perform a two-sample t-test
t_statistic, p_value = stats.ttest_ind(sample1, sample2)
# Set the significance level
alpha = 0.05
print("Results of Two-Sample t-test:")
print(f"t-statistic: {t_statistic}")
print(f"p-value: {p_value}")
print(f"Degrees of Freedom: {len(sample1) + len(sample2) - 2}")
# Plot the distributions
plt.figure(figsize=(10, 6))
plt.hist(sample1, alpha=0.5, label='Sample 1', color='blue')
plt.hist(sample2, alpha=0.5, label='Sample 2', color='orange')
plt.axvline(np.mean(sample1), color='blue', linestyle='dashed', linewidth=2)
plt.axvline(np.mean(sample2), color='orange', linestyle='dashed', linewidth=2)
plt.title('Distributions of Sample 1 and Sample 2')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
# Highlight the critical region if null hypothesis is rejected
if p_value < alpha:
    critical_region = np.linspace(min(sample1.min(), sample2.min()), max(sample1.max(), sample2.max()), 1000)
    plt.fill_between(critical_region, 0, 5, color='red', alpha=0.3, label='Critical Region')
# Show the observed t-statistic
plt.text(11, 5, f'T-statistic: {t_statistic:.2f}', ha='center', va='center', color='black', backgroundcolor='white')
# Show the plot
plt.show()
# Draw Conclusions
# Drawing Conclusions
if p_value < alpha:
    if np.mean(sample1) > np.mean(sample2):
        print("Conclusion: There is significant evidence to reject the null hypothesis.")
        print("Interpretation: The mean caffeine content of Sample 1 is significantly higher than that of Sample 2.")
        # Additional context and practical implications can be added here.
    else:
        print("Conclusion: There is significant evidence to reject the null hypothesis.")
        print("Interpretation: The mean caffeine content of Sample 2 is significantly higher than that of Sample 1.")
        # Additional context and practical implications can be added here.
else:
    print("Conclusion: Fail to reject the null hypothesis.")
    print("Interpretation: There is not enough evidence to claim a significant difference between the means.")
```

Output:


IDLE Shell 3.11.3
File Edit Shell Debug Options Window Help
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: D:/Practicals/DATA SCIENCE/prac 4.py ================
Results of Two-Sample t-test:
t-statistic: -4.512913234547555
p-value: 3.176506547470154e-05
Degrees of Freedom: 58
Conclusion: There is significant evidence to reject the null hypothesis.
Interpretation: The mean caffeine content of Sample 2 is significantly higher than that of Sample 1.
>>>

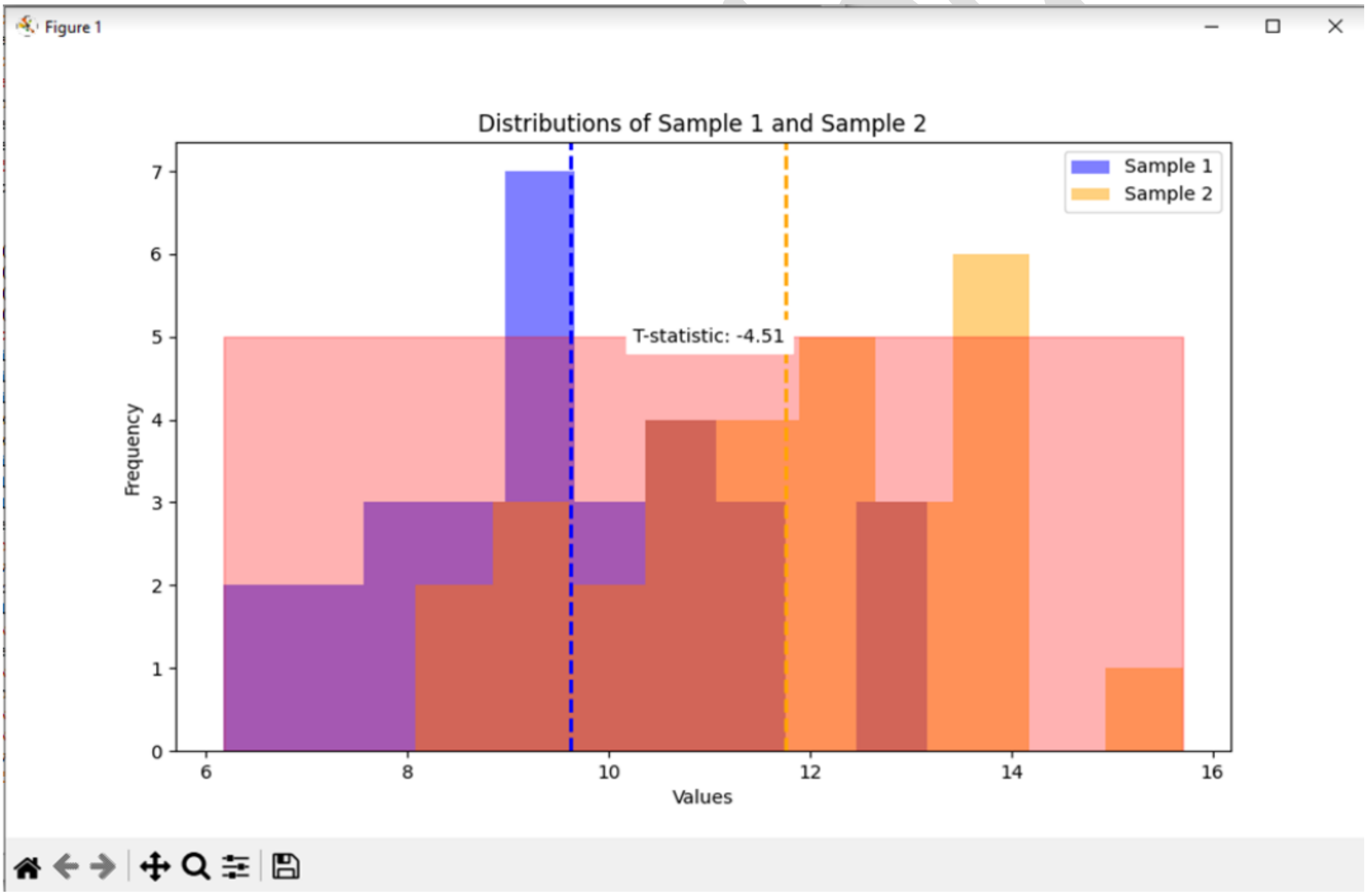


Figure 1
Distributions of Sample 1 and Sample 2
Sample 1
Sample 2
T-statistic: -4.51
Frequency
Values
0
1
2
3
4
5
6
7
6
8
10
12
14
16

Practical No.05

Aim: ANOVA (Analysis of Variance)

- Perform one-way ANOVA to compare means across multiple groups.
- Conduct post-hoc tests to identify significant differences between group means.

ANOVA:

ANOVA, or Analysis of Variance, is a statistical method used to analyze whether there are any statistically significant differences between the means of three or more independent groups. This practical focuses on conducting a one-way ANOVA, followed by post-hoc tests to pinpoint specific group differences.

One-way ANOVA:

It is a statistical test used to determine if there are any significant differences between the means of three or more independent groups. It checks if the variation between group means is greater than the variation within groups. If the test is significant, it suggests that at least one group mean is different from the others.

Post-hoc test:

It is conducted following an Analysis of Variance (ANOVA) when there are three or more groups to compare. ANOVA determines if there are any significant differences in the means of these groups. If the ANOVA result is significant, indicating that at least one group mean differs from others, a post-hoc test is employed to identify which specific group or groups exhibit significant differences.

Steps:

1. **Generate Data:**
   Simulate data for multiple groups, each representing a different experimental condition or treatment.
2. **One-Way ANOVA:**
   - Utilize the **f_oneway** function from the **scipy.stats** module to perform a one-way ANOVA on the data.
   - The F-statistic and p-value are obtained as outputs.
3. **Interpret ANOVA Results:**
   Evaluate the p-value:If p-value < 0.05, there is evidence to reject the null hypothesis, suggesting that at least one group mean is different.
4. **Post-Hoc Testing (Tukey's HSD):**
   - Combine all data into a flat array.
   - Apply the **pairwise_tukeyhsd** function from the **statsmodels.stats.multicomp** module to conduct Tukey's Honestly Significant Difference (HSD) post-hoc tests.
   - The post-hoc test results provide information on significant differences between pairs of groups.
5. **Visualize Post-Hoc Test Results:**

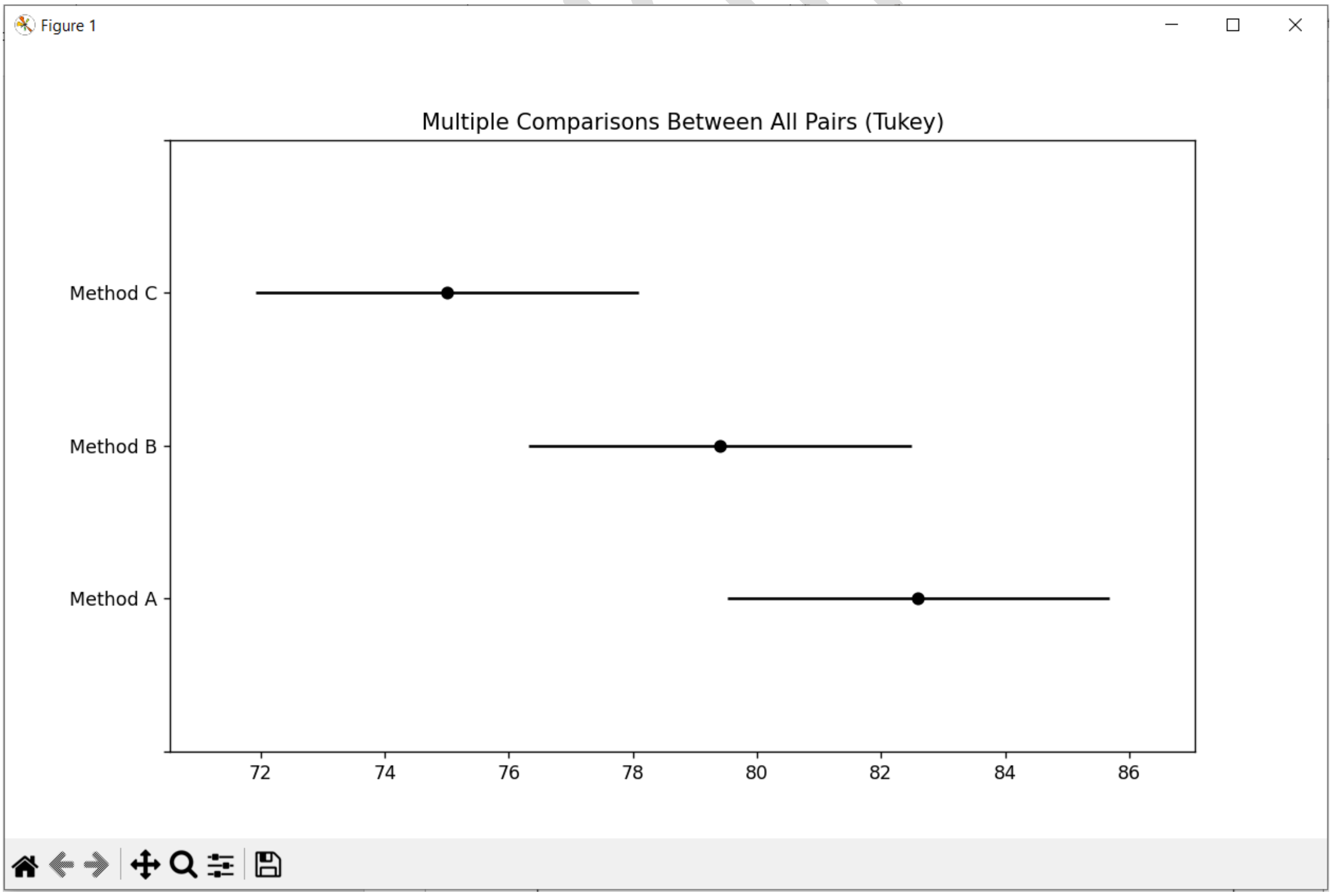
Plot the post-hoc test results using the **plot_simultaneous** function, which helps visualize significant differences between group means.

Code:

```
import numpy as np
from scipy.stats import f_oneway
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import pandas as pd
import matplotlib.pyplot as plt

# Actual data
method_a = [80, 82, 85, 78, 88]
method_b = [75, 79, 82, 80, 81]
method_c = [70, 75, 78, 72, 80]

# Combine data into a DataFrame
data = pd.DataFrame({'Method A': method_a, 'Method B': method_b, 'Method C':
method_c})

# Perform one-way ANOVA
f_statistic, p_value = f_oneway(method_a, method_b, method_c)

# Print ANOVA results
print("One-way ANOVA:")
print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")

if p_value < 0.05:
    print("Reject the null hypothesis. At least one group mean is different.")
else:
    print("Fail to reject the null hypothesis. No significant difference in group means.")

# Perform Tukey's HSD post-hoc test
flatten_data = np.concatenate([method_a, method_b, method_c])
group_labels = np.repeat(['Method A', 'Method B', 'Method C'], len(method_a))
posthoc = pairwise_tukeyhsd(flatten_data, group_labels)

# Print post-hoc results
print("\nPost-hoc test:")
print(posthoc)

# Plot the post-hoc test results
posthoc.plot_simultaneous()
plt.show()
```

Output:

```
================== RESTART: D:\Practicals\DATA SCIENCE\5().py ==================
One-way ANOVA:
F-statistic: 5.4463840399002486
P-value: 0.020744244606931365
Reject the null hypothesis. At least one group mean is different.

Post-hoc test:
   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=========================================================
 group1   group2  meandiff p-adj   lower    upper  reject
---------------------------------------------------------
Method A Method B     -3.2 0.3794  -9.3689  2.9689  False
Method A Method C     -7.6 0.0166 -13.7689 -1.4311   True
Method B Method C     -4.4 0.1802 -10.5689  1.7689  False
---------------------------------------------------------
>>>
```



Practical No.06

Aim: Regression and Its Types

- Implement simple linear regression using a dataset.
- Explore and interpret the regression model coefficients and goodness-of-fit measures.
- Extend the analysis to multiple linear regression and assess the impact of additional predictors.

Regression and its types:

Regression analysis is a statistical method used to examine the relationship between one or more independent variables and a dependent variable. It aims to understand how changes in the independent variables are associated with changes in the dependent variable. Regression analysis is widely used in various fields such as economics, finance, social sciences, and healthcare for predictive modeling, hypothesis testing, and understanding causal relationships.

There are different types of regression, but the two main types are:

1.Simple Linear Regression:

- In simple linear regression, we model the relationship between one independent variable (X) and one dependent variable (Y).
- The relationship between X and Y is assumed to be linear, meaning that changes in X are associated with a constant change in Y.
- The model equation for simple linear regression is typically represented as:

$$Y = \beta_0 + \beta_1 * X + \varepsilon$$

where:Y is the dependent variable

- X is the independent variable
- β_0 is the intercept (the value of Y when X is 0)
- β_1 is the slope (the change in Y for a one-unit change in X)
- ε is the error term, representing the variability in Y that is not explained by the model

- The goal of simple linear regression is to estimate the coefficients β_0 and β_1 that minimize the sum of the squared differences between the observed and predicted values of Y.
- The goodness-of-fit of the model is often assessed using metrics such as the coefficient of determination (R-squared), which measures the proportion of the variance in the dependent variable that is explained by the independent variable.

2.Multiple Linear Regression:

- Multiple linear regression extends the simple linear regression model to include two or more independent variables (X1, X2, ..., Xn) and one dependent variable (Y).
- The model equation for multiple linear regression is:

$$Y = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + ... + \beta_n * X_n + \varepsilon$$

where: Y is the dependent variable

- X1, X2, ..., Xn are the independent variables
- β_0 is the intercept
- $\beta_1, \beta_2, ..., \beta_n$ are the coefficients for the independent variables

    - ε is the error term
- Multiple linear regression allows us to assess the combined effect of multiple predictors on the dependent variable. Each coefficient (β) represents the change in the dependent variable associated with a one-unit change in the corresponding independent variable, holding all other variables constant.
- Similar to simple linear regression, the model's goodness-of-fit can be evaluated using metrics like R-squared.

STEPS:

1. **Load Data**: Import the dataset that contains the variables needed for regression analysis.
2. **Simple Linear Regression**:
    - Choose one independent variable (predictor) and one dependent variable (outcome).
    - Split the data into training and testing sets.
    - Fit a simple linear regression model to the training data.
    - Interpret the coefficients (intercept and slope) of the regression model.
    - Assess the goodness-of-fit using metrics such as mean squared error (MSE) and R-squared.
3. **Multiple Linear Regression**:
    - Select multiple independent variables (predictors) and one dependent variable (outcome).
    - Split the data into training and testing sets.
    - Fit a multiple linear regression model to the training data.
    - Interpret the coefficients of the regression model.
    - Assess the goodness-of-fit using metrics such as mean squared error (MSE) and R-squared.
    - Compare the performance of the multiple linear regression model with the simple linear regression model.
4. **Visualize (Optional)**:
    - Optionally, visualize the relationships between the independent and dependent variables, as well as the model predictions.
    - Visualization can provide insights into the data and model performance.
5. **Interpret Results**:
    - Draw conclusions about the relationships between variables based on the coefficients of the regression models.
    - Evaluate the predictive power of the models based on the goodness-of-fit metrics.
    - Make interpretations and recommendations based on the analysis results.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Load dataset
data = pd.read_csv('D:\\Practicals\\DATA SCIENCE\\DATA SET.csv')
# Simple Linear Regression
# Select independent and dependent variables
X_simple = data[['Cost']]
y_simple = data['Sales']
# Split data into training and testing sets for simple linear regression
X_train_simple, X_test_simple, y_train_simple, y_test_simple = train_test_split(X_simple,
y_simple, test_size=0.2, random_state=42)
# Fit simple linear regression model
model_simple = LinearRegression()
model_simple.fit(X_train_simple, y_train_simple)
# Predict on testing set for simple linear regression
y_pred_simple = model_simple.predict(X_test_simple)
# Evaluate model for simple linear regression
print('Simple Linear Regression:')
print('Intercept:', model_simple.intercept_)
print('Coefficient:', model_simple.coef_)
print('Mean Squared Error (Simple):', mean_squared_error(y_test_simple, y_pred_simple))
print('R^2 Score (Simple):', r2_score(y_test_simple, y_pred_simple))
print()
# Multiple Linear Regression
# Select independent and dependent variables
X_multiple = data[['Cost', 'Profit']]  # Independent variables
y_multiple = data['Sales']  # Dependent variable
# Split data into training and testing sets for multiple linear regression
X_train_multiple, X_test_multiple, y_train_multiple, y_test_multiple =
train_test_split(X_multiple, y_multiple, test_size=0.2, random_state=42)
# Fit multiple linear regression model
model_multiple = LinearRegression()
model_multiple.fit(X_train_multiple, y_train_multiple)
# Predict on testing set for multiple linear regression
y_pred_multiple = model_multiple.predict(X_test_multiple)
# Evaluate model for multiple linear regression
print('Multiple Linear Regression:')
print('Intercept:', model_multiple.intercept_)
print('Coefficients:', model_multiple.coef_)
print('Mean Squared Error (Multiple):', mean_squared_error(y_test_multiple,
y_pred_multiple))
print('R^2 Score (Multiple):', r2_score(y_test_multiple, y_pred_multiple))
# Visualize Simple Linear Regression
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_test_simple, y_test_simple, color='blue', label='Actual')
```

```
plt.plot(X_test_simple, y_pred_simple, color='red', label='Predicted')
plt.title('Simple Linear Regression')
plt.xlabel('Cost')
plt.ylabel('Sales')
plt.legend()
# Visualize Multiple Linear Regression
plt.subplot(1, 2, 2)
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_test_multiple['Cost'], X_test_multiple['Profit'], y_test_multiple, color='blue',
label='Actual')
ax.scatter(X_test_multiple['Cost'], X_test_multiple['Profit'], y_pred_multiple, color='red',
label='Predicted')
ax.set_title('Multiple Linear Regression')
ax.set_xlabel('Cost')
ax.set_ylabel('Profit')
ax.set_zlabel('Sales')
ax.legend()
plt.show()
```

Output:

```
>>>
================= RESTART: D:/Practicals/DATA SCIENCE/prac 6.py ================
Simple Linear Regression:
Intercept: 55.2709946396667
Coefficient: [1.94997022]
Mean Squared Error (Simple): 551.8543362796051
R^2 Score (Simple): 0.9934863094734211

Multiple Linear Regression:
Intercept: 4.547473508864641e-13
Coefficients: [1. 1.]
Mean Squared Error (Multiple): 1.0770580892617548e-26
R^2 Score (Multiple): 1.0
>>>
```

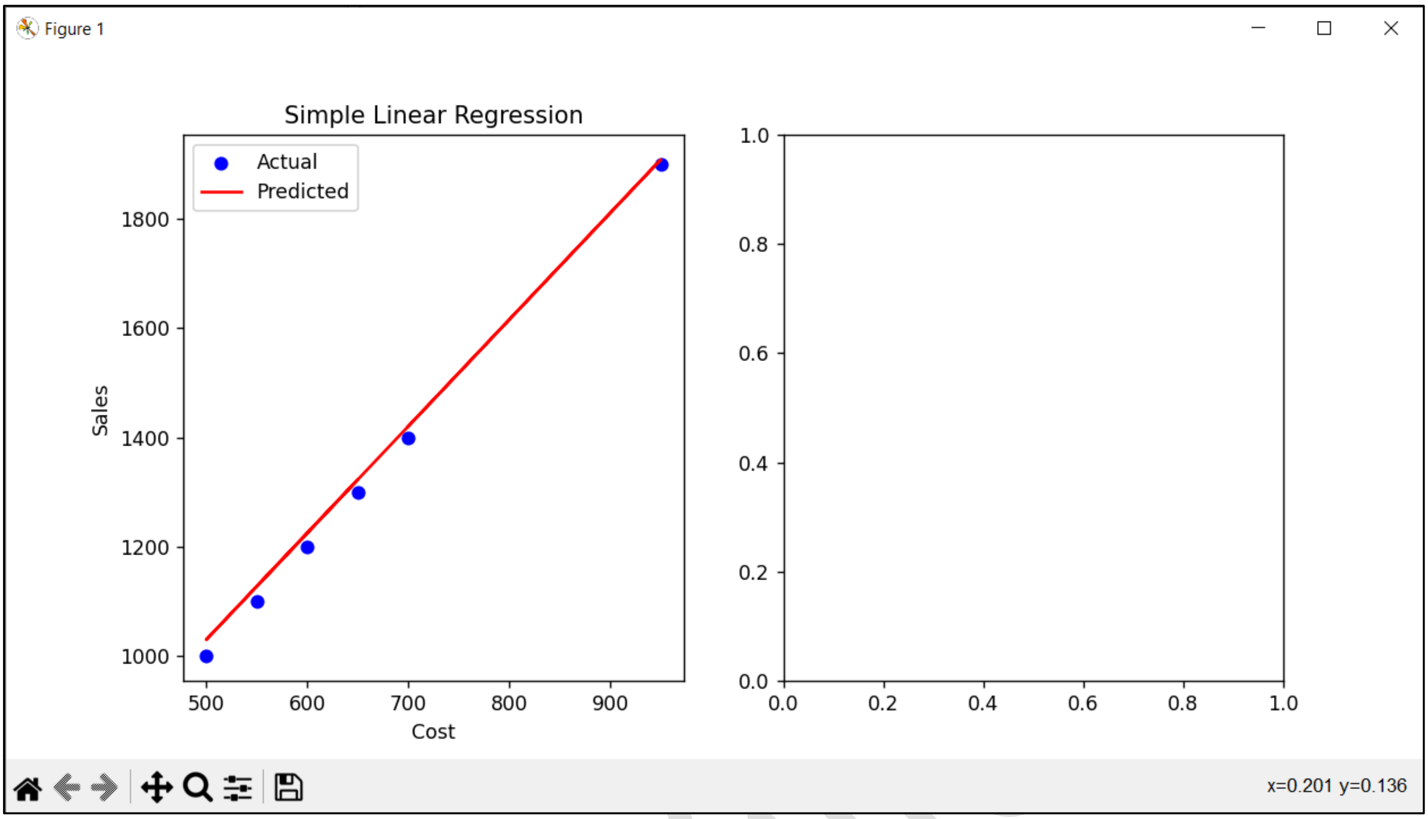


Figure 1
Simple Linear Regression
Actual
Predicted
Sales
Cost
x=0.201 y=0.136

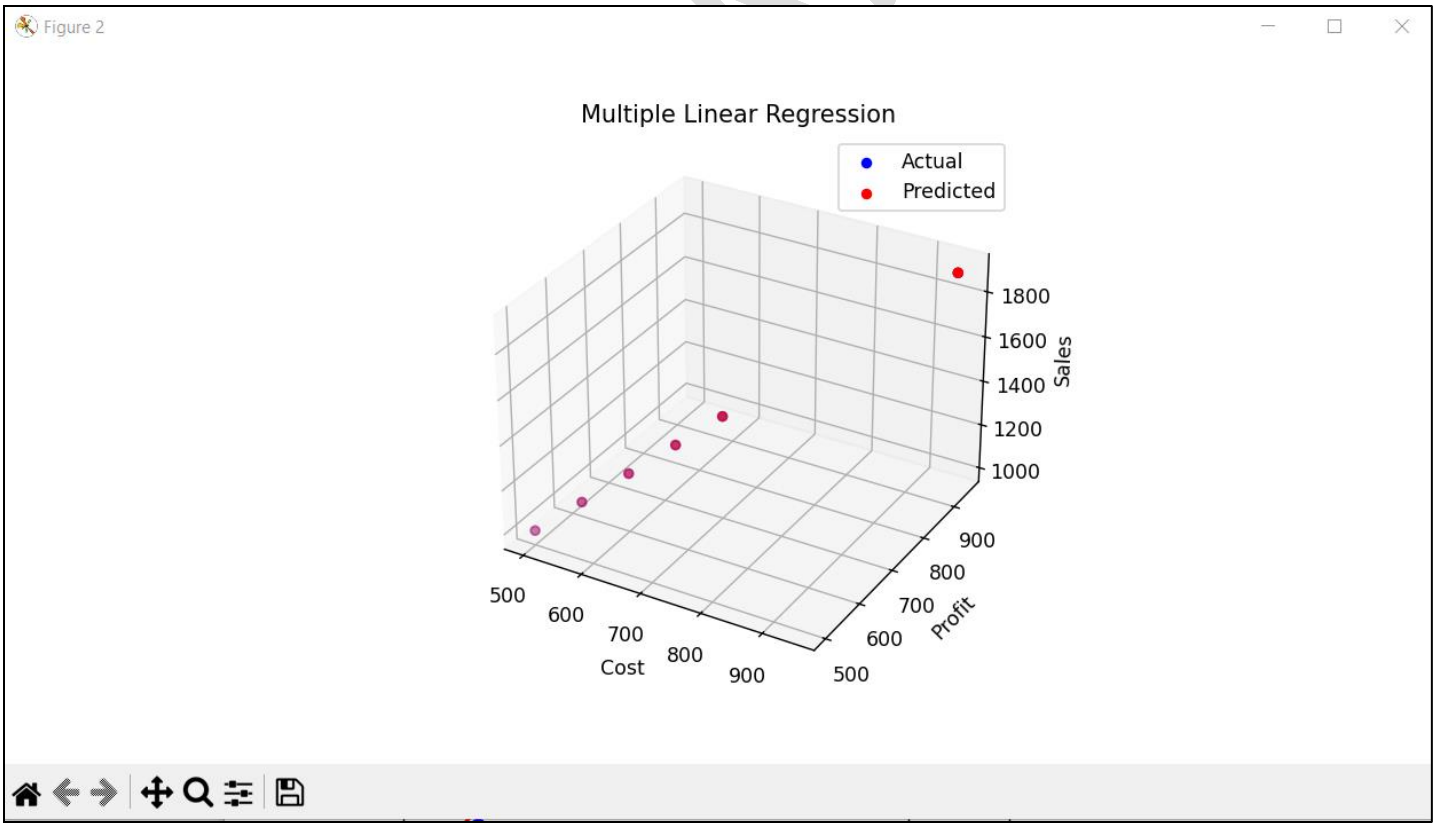


Figure 2
Multiple Linear Regression
Actual
Predicted
Sales
Cost
Profit

Practical No.07

Aim: Logistic Regression and Decision Tree

- Build a logistic regression model to predict a binary outcome.
- Evaluate the model's performance using classification metrics (e.g., accuracy, precision, recall).
- Construct a decision tree model and interpret the decision rules for classification.

Logistic Regression:

Despite its name, logistic regression is a linear model for binary classification. It predicts the probability that an instance belongs to a particular class. It works by modeling the probability of the default class (usually labeled as 1) using the logistic function, also known as the sigmoid function. Logistic regression estimates the parameters of the logistic function through optimization techniques such as gradient descent. Despite its simplicity, logistic regression can be very effective for linearly separable data or data with linear decision boundaries.

Decision Tree:

Decision trees are versatile supervised learning algorithms used for both classification and regression tasks. They work by partitioning the feature space into regions based on the feature values. At each node of the tree, a decision is made about which feature to split on, based on criteria such as information gain or Gini impurity. This process is repeated recursively until a stopping criterion is met, such as reaching a maximum tree depth or when further splitting does not lead to significant improvement in purity. Decision trees are intuitive and easy to interpret, and they can capture complex relationships between features and the target variable. However, they are prone to overfitting, especially when the tree grows too deep. Various techniques such as pruning and limiting the maximum depth of the tree can help mitigate overfitting.

Steps:

1. **Data Preparation**:
   - Creates a synthetic dataset with two features (**Feature1** and **Feature2**) and a binary target variable (**Target**).
   - Splits the dataset into features (X) and the target variable (y).
2. **Logistic Regression**:
   - Initializes and trains a logistic regression model using **LogisticRegression()** and **fit()** on the training data.
   - Makes predictions for the test data using **predict()**.
   - Evaluates the performance of the logistic regression model using classification metrics such as accuracy, precision, recall, and F1-score.
3. **Decision Tree**:
   - Initializes and trains a decision tree classifier using **DecisionTreeClassifier()** and **fit()** on the training data.
   - Makes predictions for the test data using **predict()**.

- Evaluates the performance of the decision tree model using classification metrics.

4. **Visualization**:
    - Plots the confusion matrices for both logistic regression and decision tree models using **sns.heatmap()**.

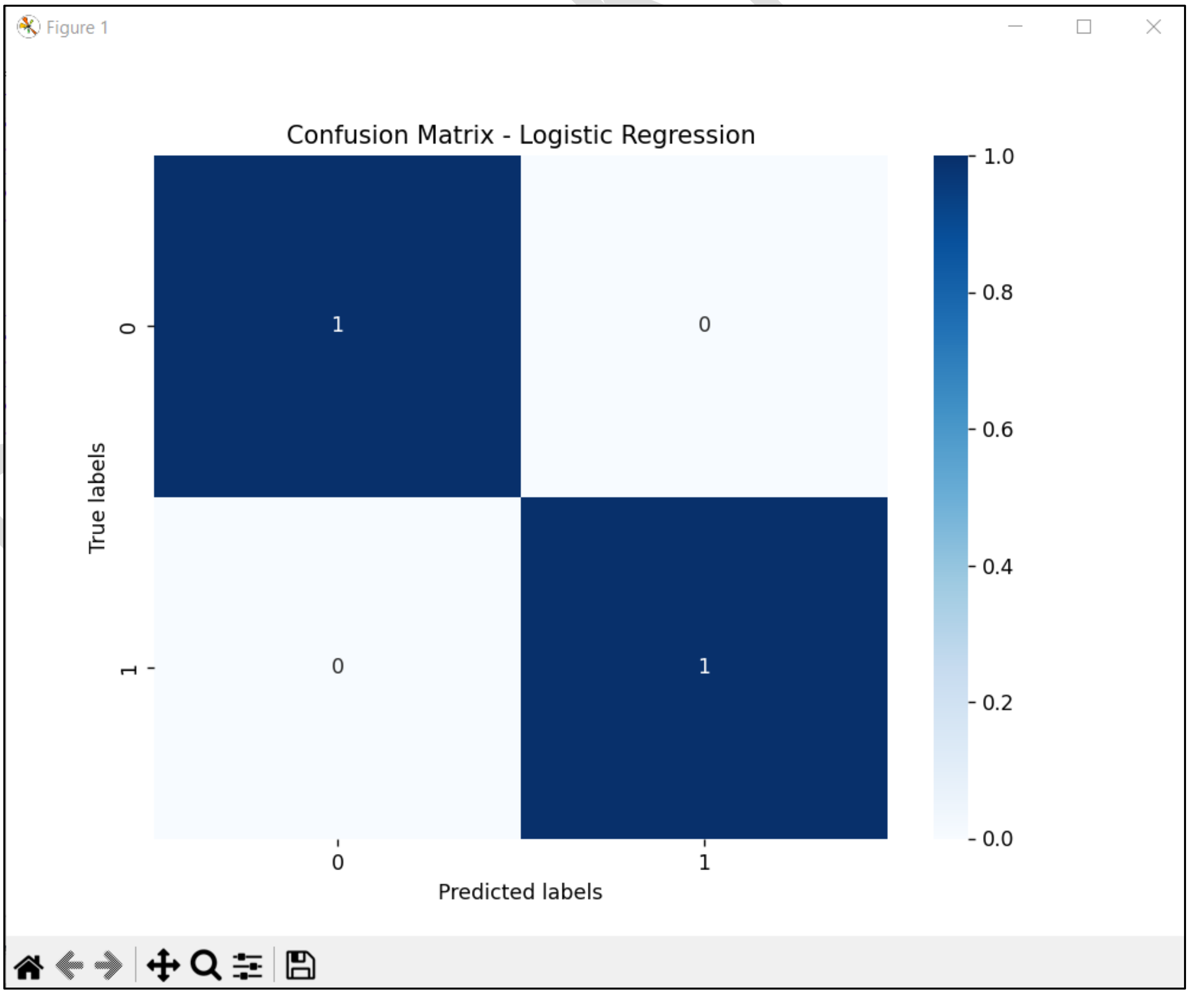
Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Creating a synthetic dataset
data = pd.DataFrame({
    'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Feature2': [0, 1, 1, 0, 1, 0, 0, 1, 0, 1],
    'Target': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]  # Binary outcome
})
# Features and target
X = data.drop('Target', axis=1)
y = data['Target']
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Logistic Regression Model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
# Predictions and Evaluation for Logistic Regression
y_pred_logistic = logistic_model.predict(X_test)
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
precision_logistic = precision_score(y_test, y_pred_logistic)
recall_logistic = recall_score(y_test, y_pred_logistic)
f1_logistic = f1_score(y_test, y_pred_logistic)
conf_matrix_logistic = confusion_matrix(y_test, y_pred_logistic)
print("Logistic Regression Metrics:")
print("Accuracy:", accuracy_logistic)
print("Precision:", precision_logistic)
print("Recall:", recall_logistic)
print("F1 Score:", f1_logistic)
print("Confusion Matrix:")
print(conf_matrix_logistic)
```

```
# Decision Tree Model
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train, y_train)
# Predictions and Evaluation for Decision Tree
y_pred_dt = decision_tree_model.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
print("\nDecision Tree Metrics:")
print("Accuracy:", accuracy_dt)
print("Precision:", precision_dt)
print("Recall:", recall_dt)
print("F1 Score:", f1_dt)
print("Confusion Matrix:")
print(conf_matrix_dt)
# Plot Confusion Matrix for Logistic Regression
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_logistic, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
# Plot Confusion Matrix for Decision Tree
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_dt, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix - Decision Tree')
plt.show()
```

Output:

```
================ RESTART: D:/Practicals/DATA SCIENCE/prac 7.py ================
Logistic Regression Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
Confusion Matrix:
[[1 0]
 [0 1]]

Decision Tree Metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
Confusion Matrix:
[[1 0]
 [0 1]]
>>>
```



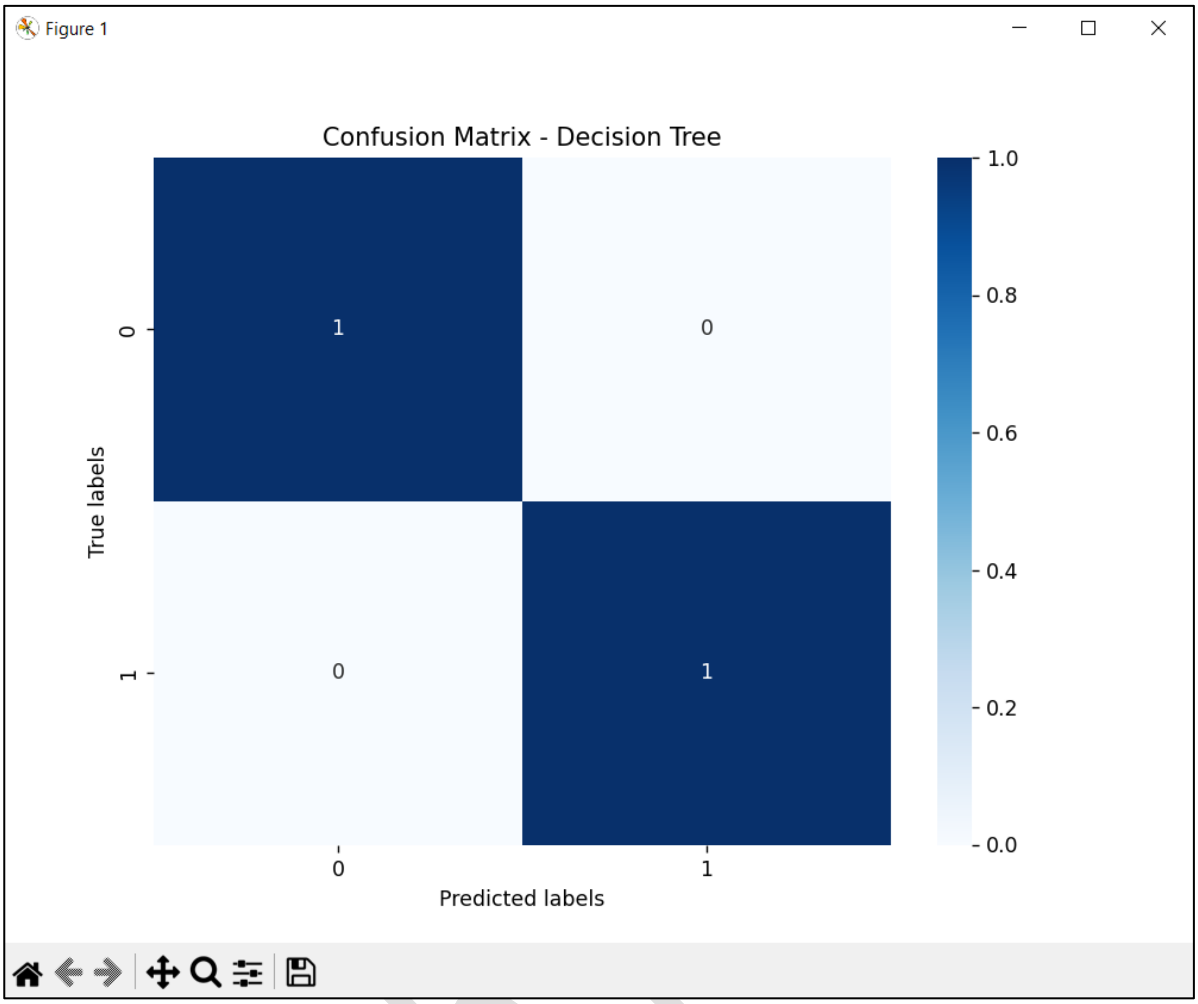


Figure 1
Confusion Matrix - Decision Tree
True labels
Predicted labels
0
1
1
0
0
1
0
1
1.0
0.8
0.6
0.4
0.2
0.0

Practical No.08

Aim: K-Means Clustering

- Apply the K-Means algorithm to group similar data points into clusters.
- Determine the optimal number of clusters using elbow method or silhouette analysis.
- Visualize the clustering results and analyze the cluster characteristics.

K-Means Clustering:

K-Means Clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a set of K clusters. The algorithm aims to group similar data points together and separate dissimilar points into different clusters. It works iteratively to assign each data point to the nearest centroid (center of a cluster) and then update the centroids based on the mean of the data points assigned to each cluster. This process continues until the centroids no longer change significantly or a maximum number of iterations is reached. K-Means clustering is widely used in various applications such as customer segmentation, image compression, anomaly detection, and more.

Elbow method:

It looks for a point in the plot of WCSS (Within-Cluster Sum of Squares) against the number of clusters where the rate of decrease slows down, suggesting the optimal number of clusters.

Silhouette analysis:

It evaluates the quality of clustering by measuring how similar each point is to its own cluster compared to other clusters. The highest average silhouette score suggests the optimal number of clusters.

Steps:

1. **Load Data**: Load your dataset containing features for clustering.
2. **Preprocess Data**: If necessary, preprocess the data by handling missing values, scaling features, or encoding categorical variables.
3. **Apply K-Means Algorithm**:
    - Initialize the K-Means algorithm with an initial guess for the number of clusters (K).
    - Fit the K-Means model to the data.
4. **Determine the Optimal Number of Clusters**:
    - Use either the elbow method or silhouette analysis:
        - **Elbow Method:**
            - Calculate the Within-Cluster Sum of Squares (WCSS) for different values of K.
            - Plot the WCSS against the number of clusters.
            - Identify the "elbow" point in the plot where the rate of decrease in WCSS slows down.
            - The number of clusters at the elbow point is considered optimal.
        - **Silhouette Analysis:**
            - Calculate the silhouette score for different values of K.

- Plot the silhouette score against the number of clusters.
- Choose the number of clusters that maximizes the silhouette score.

5. **Visualize Clustering Results**:
   - Plot the data points with different colors representing the clusters they belong to.
   - Optionally, plot centroids (cluster centers) if desired.
   - Visualize the clusters in 2D or 3D space, depending on the dimensionality of the data.
6. **Analyze Cluster Characteristics**:
   - Analyze the centroids of each cluster to understand the characteristics of the clusters.
   - Evaluate the distribution of data points within each cluster.
   - Interpret the results and draw insights about the data based on the clustering.
7. **Iterate (Optional)**:
   - If necessary, iterate through the process by adjusting parameters or preprocessing steps to refine the clustering results.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
# Generate sample data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
# Apply K-Means algorithm with different number of clusters
wcss = []
silhouette_scores = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X, kmeans.labels_))
# Determine the optimal number of clusters using the elbow method
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(2, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # Within cluster sum of squares
plt.subplot(1, 2, 2)
plt.plot(range(2, 11), silhouette_scores, marker='o', linestyle='--')
plt.title('Silhouette Analysis')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.tight_layout()
```

```
plt.show()
# Based on the analysis, choose the optimal number of clusters
optimal_num_clusters = np.argmax(silhouette_scores) + 2  # Adding 2 because range starts from 2
print("Optimal number of clusters:", optimal_num_clusters)
# Apply K-Means with optimal number of clusters
kmeans = KMeans(n_clusters=optimal_num_clusters, init='k-means++', max_iter=300, n_init=10, random_state=0)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
# Visualize the clustering results
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
# Analyze the cluster characteristics
print("Cluster Centers:\n", centers)
```

Output:

```
>>>
================= RESTART: D:/Practicals/DATA SCIENCE/prac 8.py ================
Optimal number of clusters: 4
Cluster Centers:
 [[ 0.94973532  4.41906906]
 [ 1.98258281  0.86771314]
 [-1.37324398  7.75368871]
 [-1.58438467  2.83081263]]
>>>
```

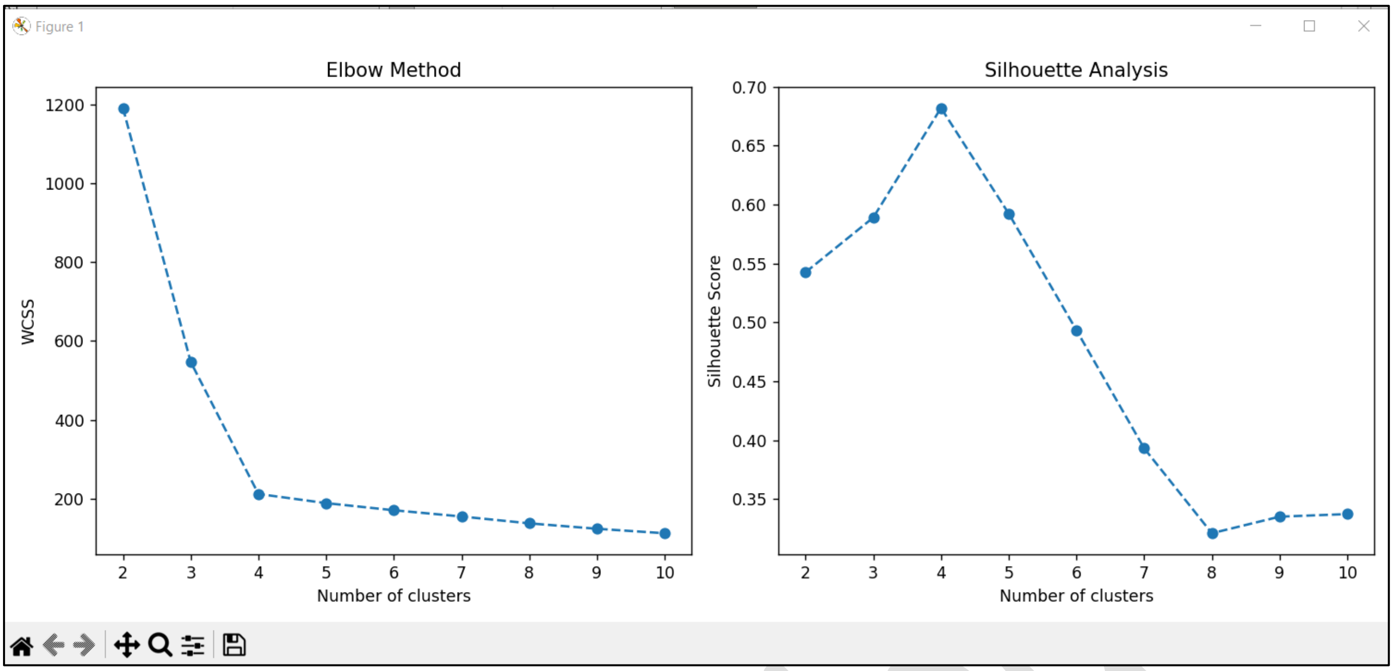


Figure 1
Elbow Method
WCSS
Number of clusters
Silhouette Analysis
Silhouette Score
Number of clusters

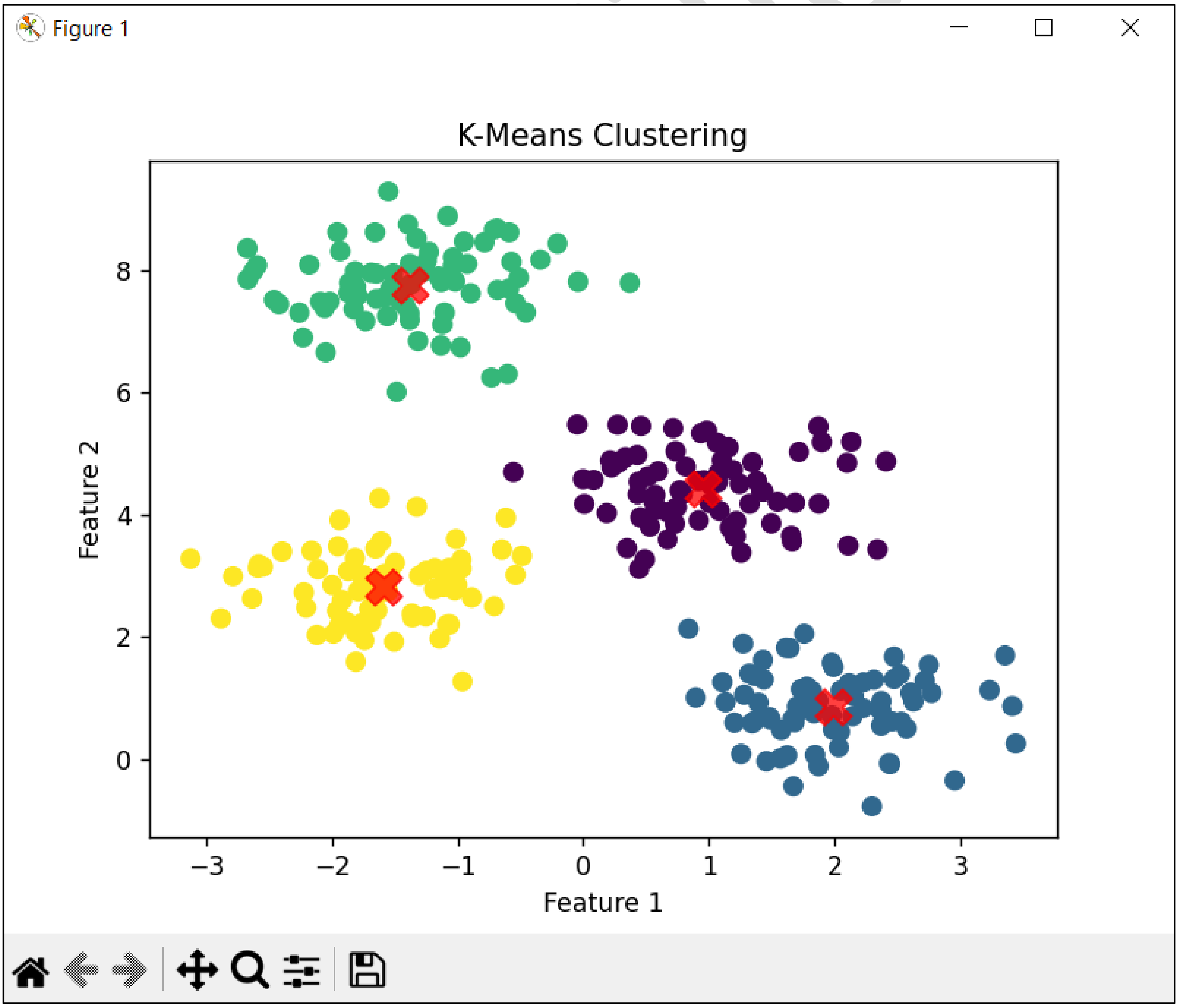


Figure 1
K-Means Clustering
Feature 2
Feature 1

Practical No.09

Aim: Principal Component Analysis (PCA)

- Perform PCA on a dataset to reduce dimensionality.
- Evaluate the explained variance and select the appropriate number of principal components.
- Visualize the data in the reduced-dimensional space.

Principal Component Analysis (PCA):

Principal Component Analysis (PCA) is a mathematical technique used for reducing the dimensionality of high-dimensional datasets while preserving most of the important information. It identifies the directions of maximum variance in the data and projects it onto a lower-dimensional space defined by these directions, called principal components. PCA is widely used for tasks such as visualization, noise reduction, feature extraction, and data compression.

Steps:

1. **Load Data**: Load your dataset containing features for dimensionality reduction.
2. **Preprocess Data**: If necessary, preprocess the data by handling missing values, scaling features, or encoding categorical variables.
3. **Apply PCA**:
   - Initialize the PCA algorithm.
   - Fit the PCA model to the data.
   - Transform the original data into the reduced-dimensional space using the learned principal components.
4. **Evaluate Explained Variance**:
   - Analyze the explained variance ratio of each principal component.
   - Plot the cumulative explained variance to decide on the appropriate number of principal components to retain.
   - Typically, you aim to retain a significant portion of the variance (e.g., 90% or more).
5. **Visualize Data in Reduced-Dimensional Space**:
   - Plot the data points in the reduced-dimensional space using the selected principal components.
   - Optionally, visualize the data with different colors or markers to represent different classes or groups.
   - Interpret the visualization to gain insights into the structure of the data.

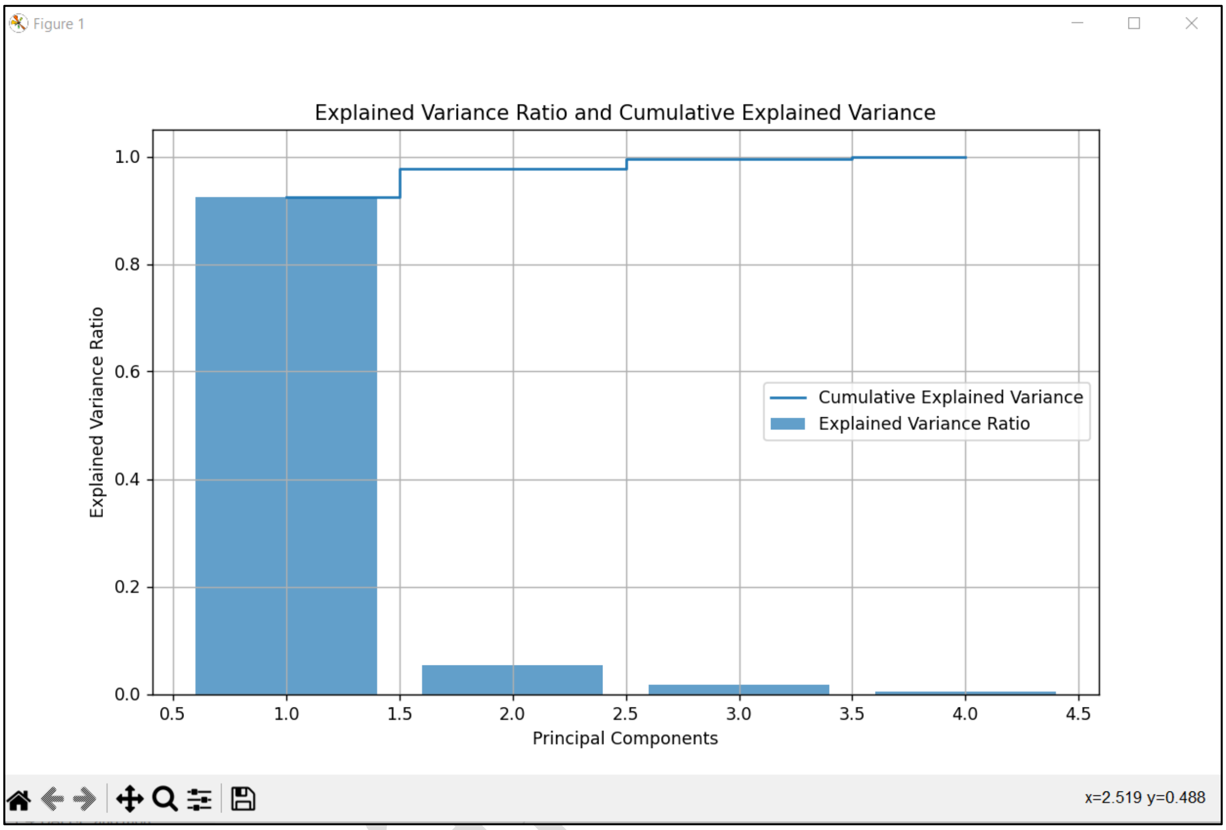
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
```

```
from sklearn.decomposition import PCA

# Load dataset
data = load_iris()
X = data.data
y = data.target
# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X)
# Evaluate explained variance
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratio)
# Plot explained variance ratio
plt.figure(figsize=(10, 6))
plt.bar(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, alpha=0.7,
align='center',
       label='Explained Variance Ratio')
plt.step(range(1, len(cumulative_variance) + 1), cumulative_variance, where='mid',
        label='Cumulative Explained Variance')
plt.xlabel('Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio and Cumulative Explained Variance')
plt.legend()
plt.grid()
plt.show()
# Select appropriate number of principal components based on explained variance
n_components = np.argmax(cumulative_variance >= 0.95) + 1
print("Number of principal components to retain:", n_components)
# Visualize data in reduced-dimensional space
plt.figure(figsize=(8, 6))
for target in np.unique(y):
    plt.scatter(X_pca[y == target, 0], X_pca[y == target, 1], label=f'Class {target}')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA: Iris Dataset')
plt.legend()
plt.grid()
plt.show()
```

Output:

```
>>>
================= RESTART: D:/Practicals/DATA SCIENCE/prac 9.py ================
Number of principal components to retain: 2
>>> |
```



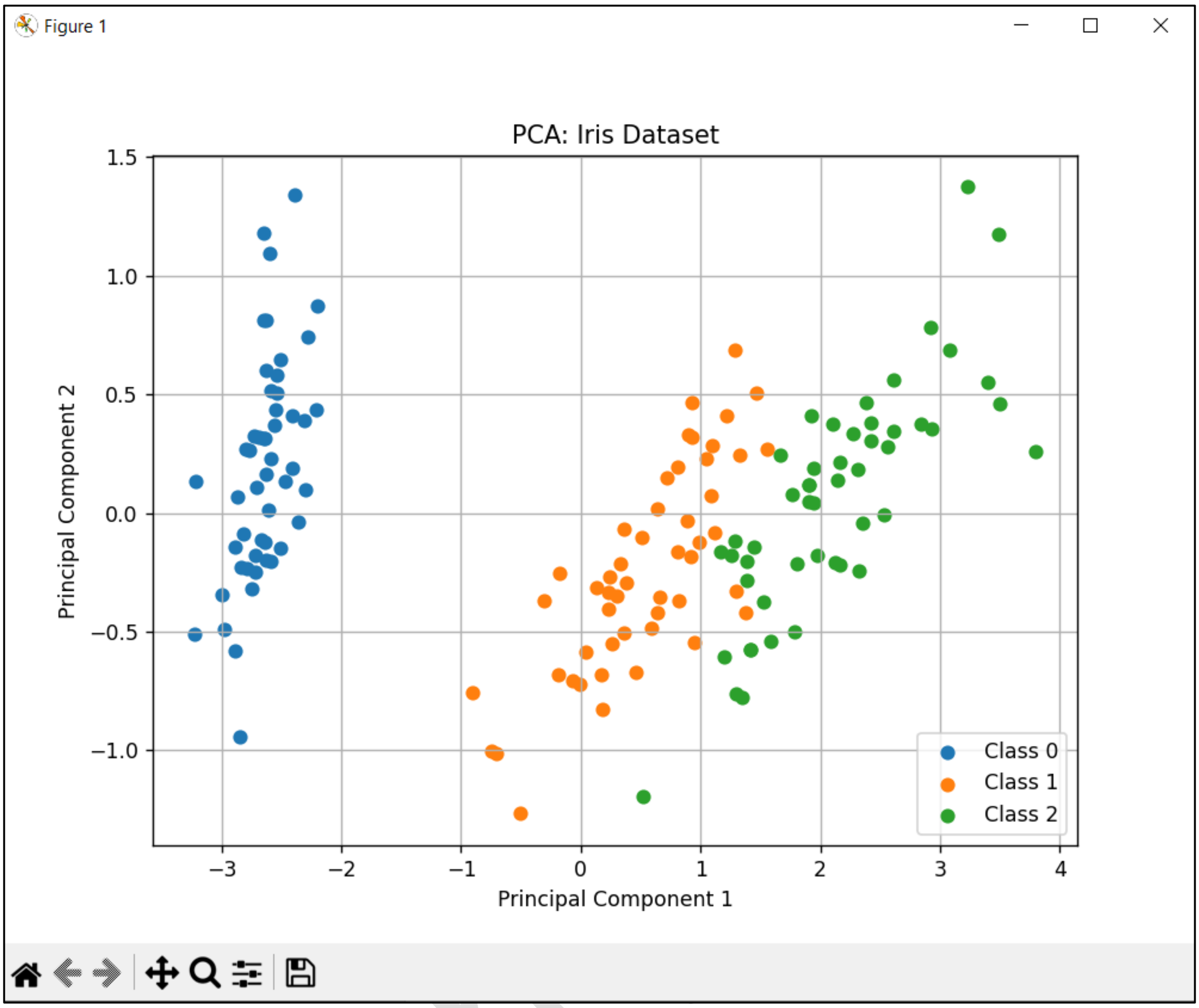


Figure 1
PCA: Iris Dataset
Principal Component 1
Principal Component 2
Class 0
Class 1
Class 2
−3
−2
−1
0
1
2
3
4
1.5
1.0
0.5
0.0
−0.5
−1.0

Practical No.10

Aim: Data Visualization and Storytelling

- Create meaningful visualizations using data visualization tools
- Combine multiple visualizations to tell a compelling data story.
- Present the findings and insights in a clear and concise manner.

Data Visualization and Storytelling:

Data visualization is the process of presenting data in visual formats like charts and graphs, making complex information easier to understand immediately. It is like painting a picture with data, using visuals to convey insights and trends.

Storytelling, on the other hand, involves crafting a narrative around the data, guiding the audience through a journey of discovery. It is about connecting the dots between data points, providing context, and evoking emotions to create a compelling narrative.

When combined, data visualization and storytelling create a powerful way to communicate insights. Visualizations serve as the backbone of the story, while storytelling adds depth and meaning, engaging the audience, and driving understanding and action. Together, they transform raw data into impactful stories that resonate with audiences.

Steps:

1. **Load the Data:** Begin by loading the dataset you want to analyze using a suitable data manipulation library like pandas. Ensure the data is clean and formatted correctly for analysis.
2. **Create Meaningful Visualizations:** Utilize data visualization tools such as matplotlib, seaborn, or plotly to create visual representations of the data. Choose appropriate visualization types (e.g., bar charts, scatter plots, box plots) that effectively convey insights and trends in the data.
3. **Combine Multiple Visualizations:** Integrate multiple visualizations into a cohesive narrative to tell a compelling data story. Arrange the visualizations in a logical sequence, highlighting key insights and relationships between different data points.
4. **Present Findings and Insights:** Communicate the findings and insights derived from the visualizations in a clear and concise manner. Use annotations, captions, or accompanying text to provide context and interpretation for the visualizations. Tailor the presentation to the audience's level of understanding and objectives.
5. **Iterate and Refine:** Review the visualizations and narrative to ensure coherence and effectiveness in conveying the intended message. Iterate on the visualizations and storytelling elements based on feedback or further analysis of the data.

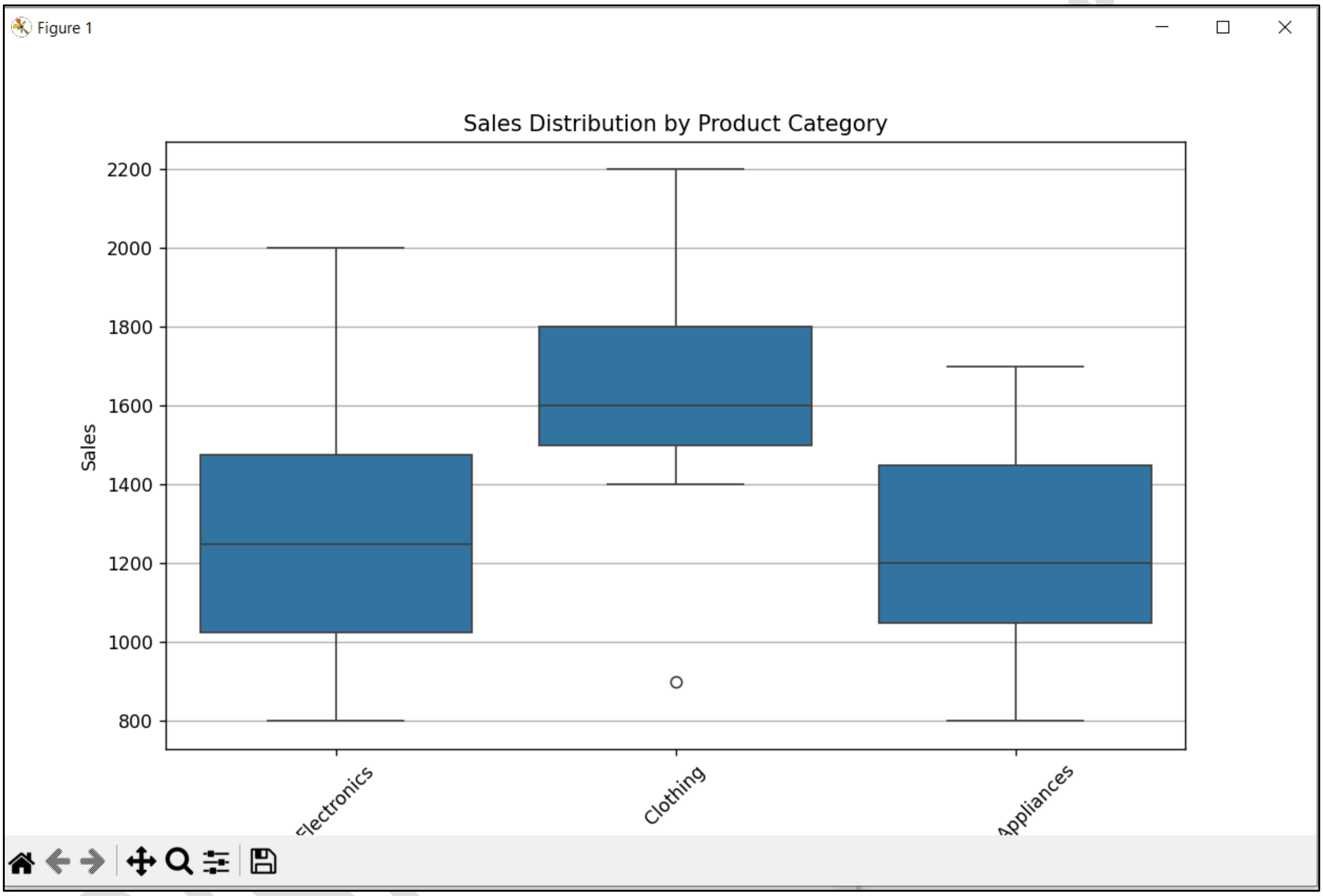
Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the data
data = pd.read_csv('D:\\Practicals\\DATA SCIENCE\\DATA SET.csv')
# Visualize sales distribution by product category
plt.figure(figsize=(10, 6))
sns.boxplot(x='Category', y='Sales', data=data)
plt.title('Sales Distribution by Product Category')
plt.xlabel('Product Category')
plt.ylabel('Sales')
plt.xticks(rotation=45)
plt.grid(axis='y')  # Add gridlines only to the y-axis
plt.show()
# Visualize sales vs. profit
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Sales', y='Profit', data=data, hue='Category', palette='Set2')
plt.title('Sales vs. Profit')
plt.xlabel('Sales')
plt.ylabel('Profit')
plt.grid(True)  # Add gridlines to both axes
plt.show()
# Conditional Insights
median_sales_by_category = data.groupby('Category')['Sales'].median().sort_values(ascending=False)
if median_sales_by_category.idxmax() == 'Clothing':
    print("Insights:")
    print("- The 'Clothing' category has the highest median sales, followed by 'Electronics' and 'Home Appliances'.")
if data[['Sales', 'Profit']].corr().iloc[0, 1] > 0:
    print("- There is a positive correlation between sales and profit across all product categories.")
if data['Sales'].max() == data.loc[data['Sales'].idxmax(), 'Sales']:
    print("- The category with the highest sales is:", data.loc[data['Sales'].idxmax(), 'Category'])
else:
    print("- The data does not meet any specific condition for insights.")
```

Output:

```
=============== RESTART: D:/Practicals/DATA SCIENCE/prac 10().py ===============
Insights:
- The 'Clothing' category has the highest median sales, followed by 'Electronics
' and 'Home Appliances'.
- There is a positive correlation between sales and profit across all product ca
tegories.
- The category with the highest sales is: Clothing
>>>
```



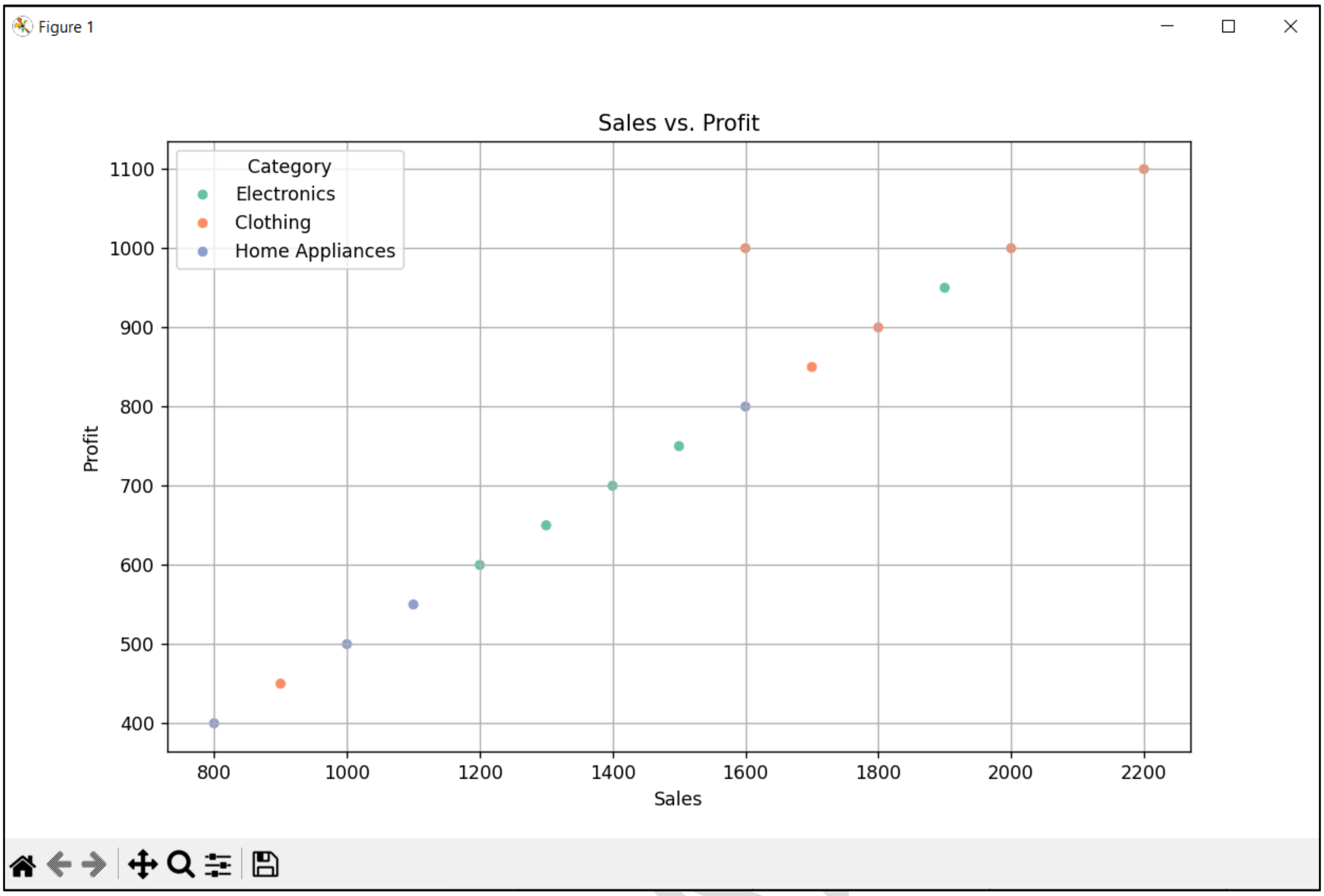


Figure 1
Sales vs. Profit
Category
Electronics
Clothing
Home Appliances
Profit
1100
1000
900
800
700
600
500
400
800
1000
1200
1400
1600
1800
2000
2200
Sales