**Practical No.01**

**Date:10/12/2024 Roll No.:17**

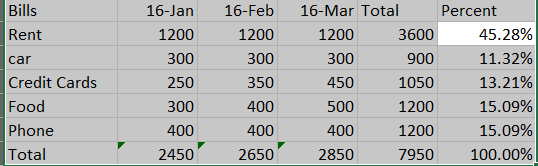
**Aim: Introduction to Excel**

* **Perform conditional formatting on a dataset using various criteria.**
* **Create a pivot table to analyse 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.**

**Highlight Cells Using Conditional Formatting.**

**Greater Than Rule:**

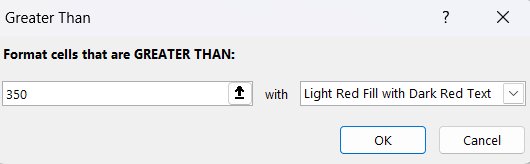
**Step 1:** Select the range of cells you want to apply the highlight.



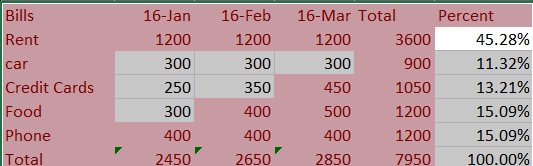
**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

**Step 3:** Click Highlight Cells Rule > Greater Than

**Step 4:** Enter the desired value and select the formatting style.

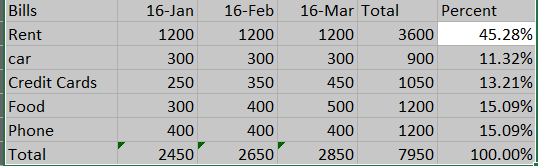


**Result:**

****

**Less Than Rule:**

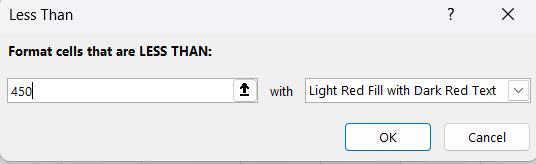
**Step 1:** Select the range of cells you want to apply the highlight.



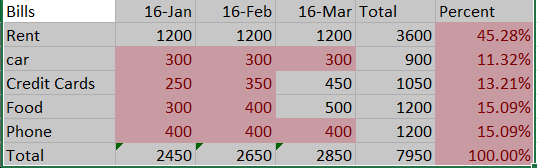
**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

**Step 3:** Click Highlight Cells Rule > Less Than

**Step 4:** Enter the desired value and select formatting style.

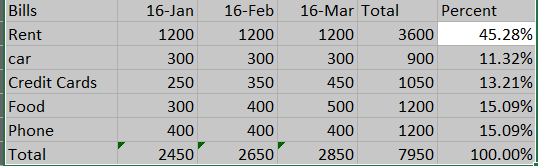
****

**Result:**

****

**Between Rule:**

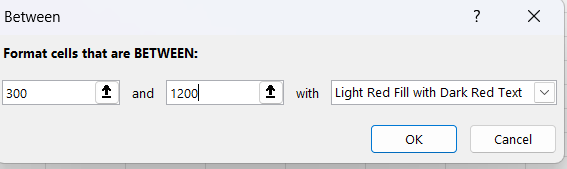
**Step 1:** Select the range of cells you want to apply the highlight.



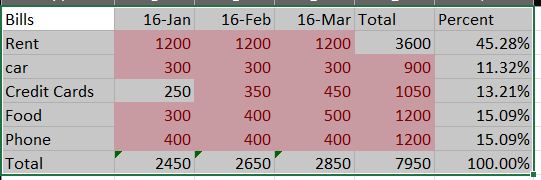
**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

**Step 3:** Click Highlight Cells Rule > Between

**Step 4:** Enter the desired value and select the formatting style.

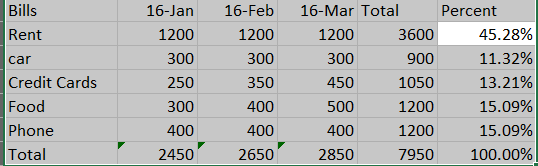


**Result:**



**Equal To Rule:**

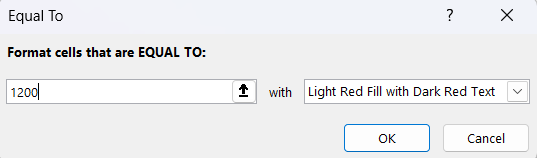
**Step 1:** Select the range of cells you want to apply the highlight.



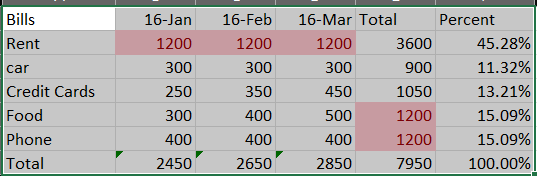
**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

**Step 3:** Click Highlight Cells Rule > Equal To

**Step 4:** Enter the desired value and select the formatting style.

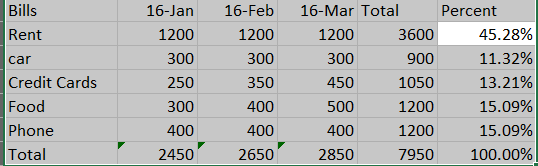


**Result:**



**Duplicate Rule:**

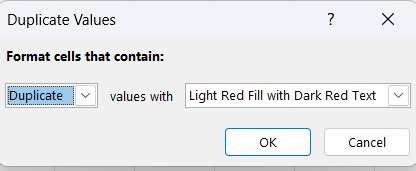
**Step 1:** Select the range of cells you want to apply the highlight.



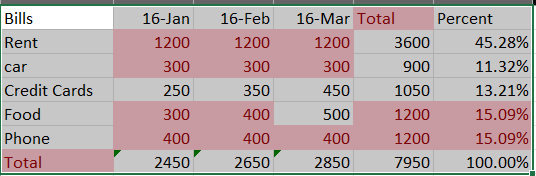
**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

**Step 3:** Click Highlight Cells Rule > Duplicate

**Step 4:** Enter the desired value and select the formatting style.



**Result:**

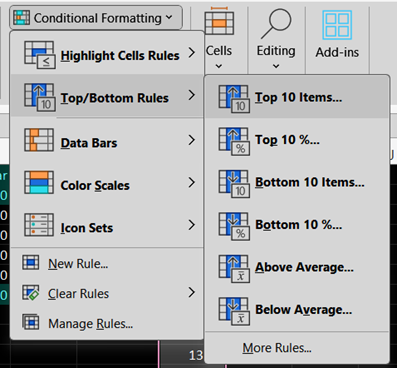


**Top 10 Items:**

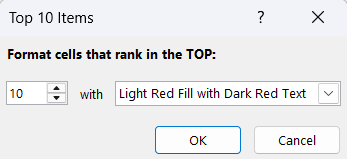
**Step 1:** Select the range of cells where you want to apply conditional formatting.

**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

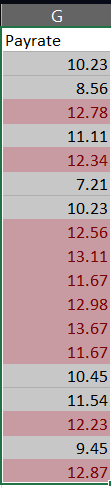
**Step 3:** Click Top/Bottom rules > Top 10 Items



**Step 4:** Mention the number of highest records you want to highlight.



**Result:**

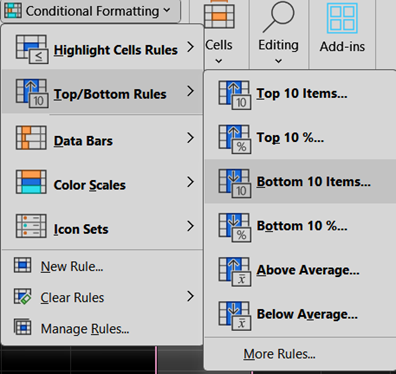


**Bottom 10 Items:**

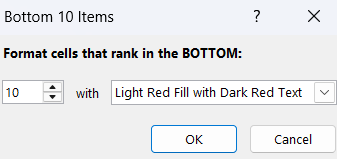
**Step 1:** Select the range of cells where you want to apply conditional formatting.

**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

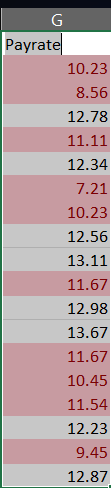
**Step 3:** Click Top/Bottom rules > Bottom 10 Items



**Step 4:** Mention the number of highest records you want to highlight.



**Result:**

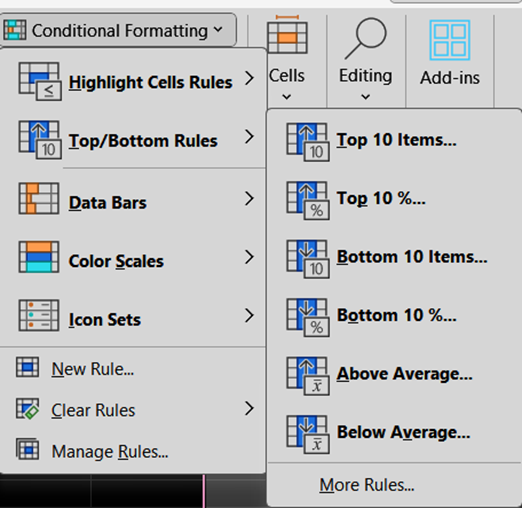


**Above Average:**

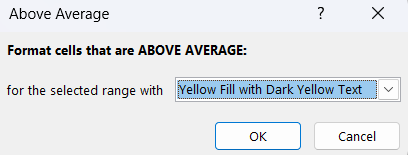
**Step 1:** Select the range of cells where you want to apply conditional formatting.

**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

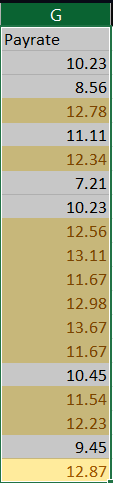
**Step 3:** Click Top/Bottom rules > Above Average

****

**Step 4:** Mention the number of highest records you want to highlight.



**Result:**

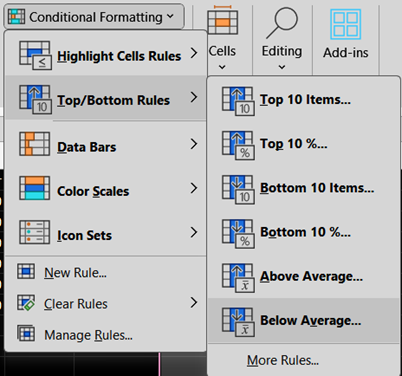


**Below Average:**

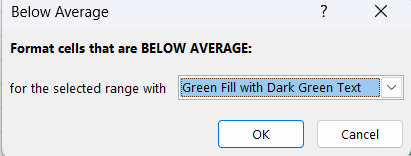
**Step 1:** Select the range of cells where you want to apply conditional formatting.

**Step 2:** On the Home tab, under Styles Group, click Conditional Formatting.

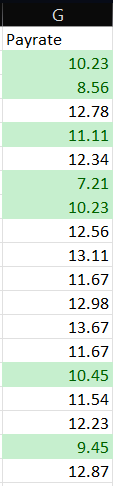
**Step 3:** Click Top/Bottom rules > Below Average.



**Step 4:** Format cells as you want.



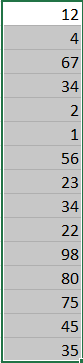
**Result:**



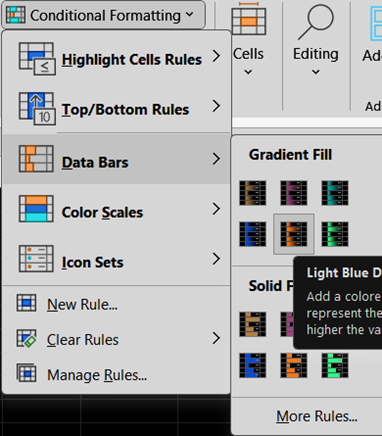
**Data Bars in Conditional Formatting**

Data bars in Excel are used to visualize the range of cells. The longer bar represents higher value. To add the data bars, follow these steps:

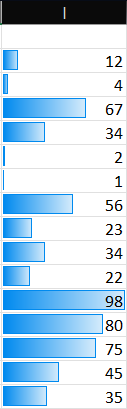
* Select the range of Cells.



* On the Home tab, go to Conditional Formatting > Data Bars and select a subtype.



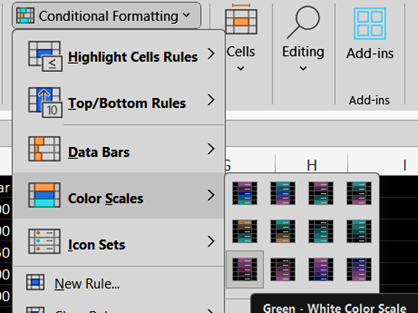
**Result:**



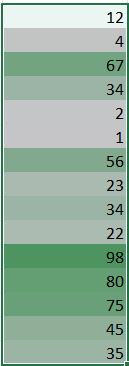
**Color Scales in Conditional Formatting**

Color Scales in Excel make the visualization of values in a range of cells very easy. To add a color scale, follow these steps:

* Select the range of cells.
* On the Home tab, go to Styles Group > Conditional Formatting.
* Click Color Scales and select a subtype.

****

**Result:**

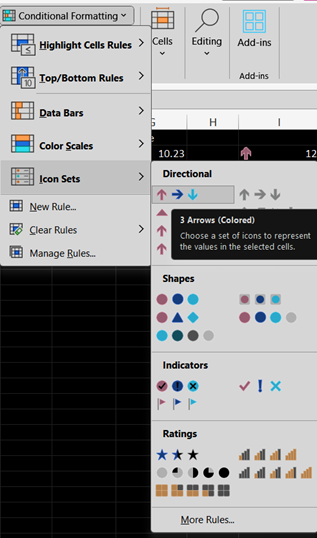


* The white color represents the minimum value in the range.
* The medium light green color represents the median value.
* The green color represents the maximum value.
* All the other values are colored proportionally.

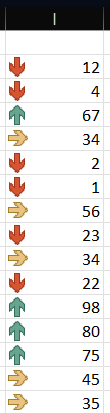
**Icon Sets in Conditional Formatting**

Excel Conditional Formatting icon sets are used to visualize the data with the help of shapes arrows, checks marks, and other objects. To add an icon sets, follow these steps:

* Select the range of cells.
* On the Home tab, go to Styles > Conditional Formatting.
* Click Icon Sets and select a subtype.

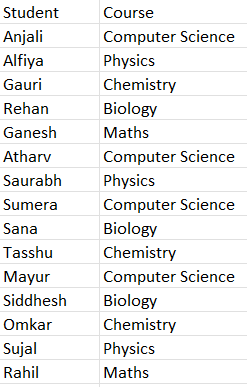


**Result:**

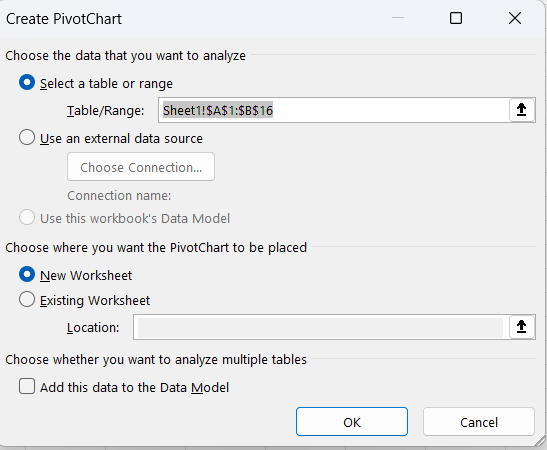


The update the rules, go to Conditional Formatting > Manage Rules > Edit rules. You can change the rules according to your preferences

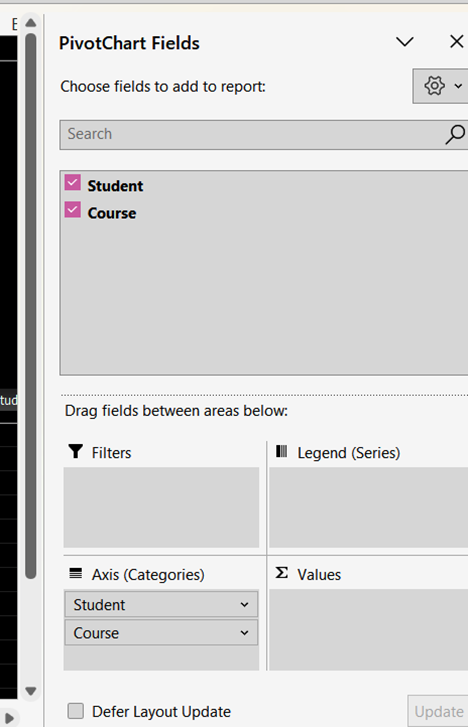
**Pivot Table and Chart**

****

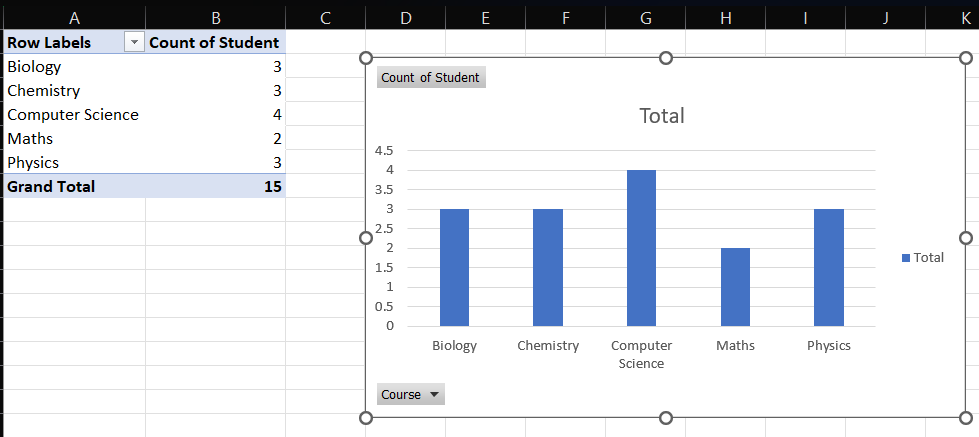
Insert -> Pivot Chart -> Ok



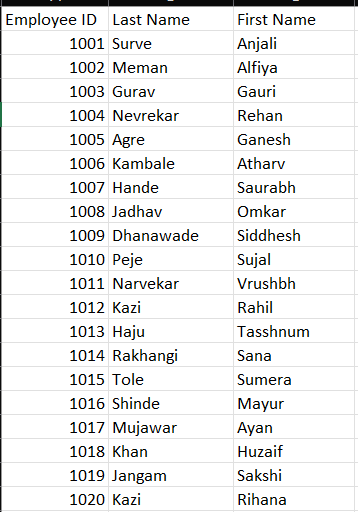
Select pivot chart fields.



Pivot chart field -> Students -> Move to values.



**VLOOKUP**

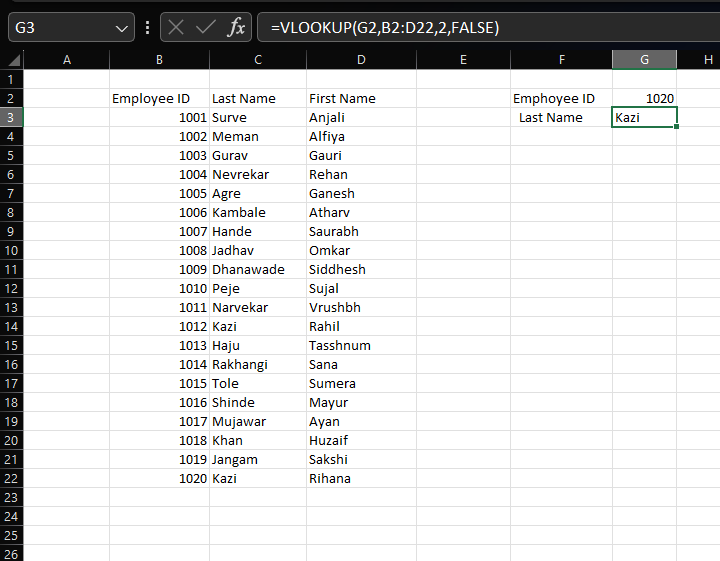


Create new table with Employee id & Last Name. Enter the given formula of VLOOKUP in Last Name Row.





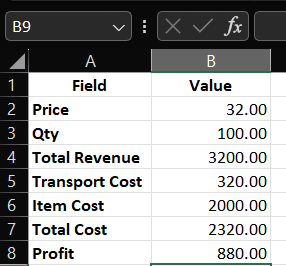
Enter the Employee ID and then Last Name get appear.



**What-If Analysis:**

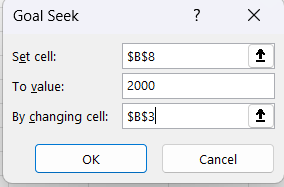
Create a table by using following formulas:

1. Total Revenue = Price\*Quantity
2. Transport Cost = Total Revenue\*10%
3. Item Cost = 20\*Quantity
4. Total Cost = Transport Cost + Item Cost
5. Profit = Total Revenue – Total Cost

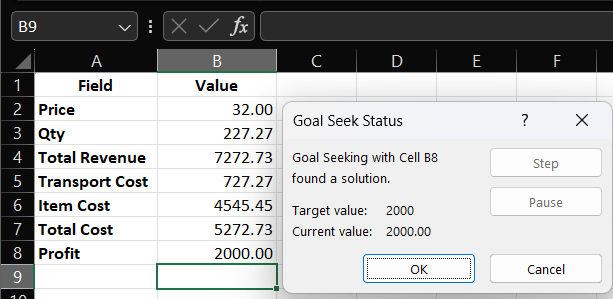


Go to Data -> What – If Analysis -> Goal Seek.

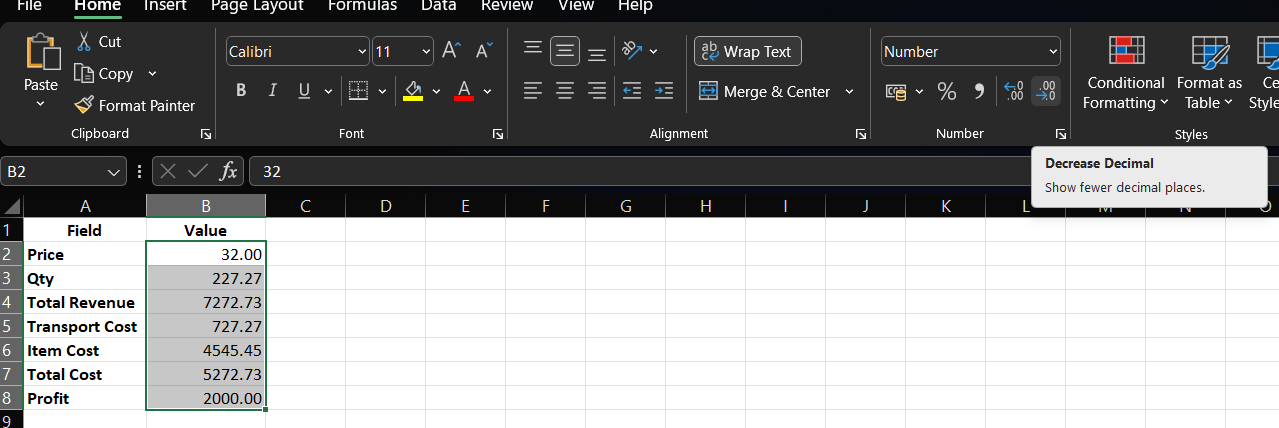
In set call select Profit & changing cell select Quantity. In To value type 2000. Then click OK.

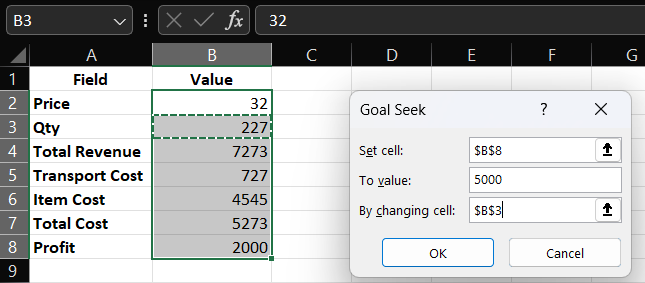


After that we see the changes in our table.

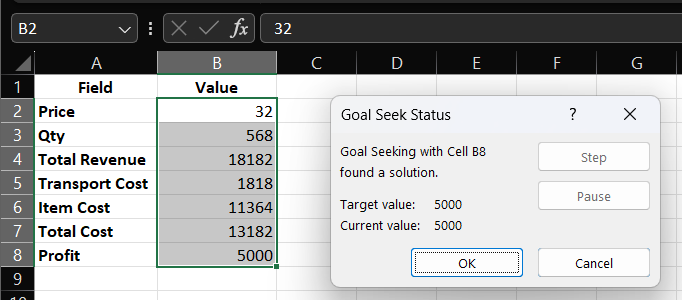


Remove the decimal Quantity and enter.





In Goal Seek select profit -> value 5000 -> select changing cell with Quantity.



**Practical No.02**

**Date: / /202 Roll No.:17**

**Aim: Feature Scaling and Dummification.**

Apply feature scaling and dummification like standardization and normalization to numerical features Perform feature dummification to convert categorical variables into numerical representation

**• Feature Scaling :**

It is a preprocessing technique used to standardize the range of independent variables or features

It is essential for certain machine learning algorithm that are sensitive to the scale of input features , ensuring that all features contribute equally to the learning machines .

• **Feature Dummification :**

One-Hot encoder is a technique use to convert categorical variables into numerical representation these is necessary because many machine learning algorithms required numerical inputs 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 Scale :** Apply standardization and normalization to numeric fetaures.

**3.Feature Dummification:** Convert categorical variables into numeric representation using One-Hot Encoder.

**4.Combine Feature:** Combine scale numeric feature with One-Hot categorical feature.

**5.Display Resulting Dataset:** Display the find dataset after both Feature scalling and Dummification.

**Program:**

import pandas as pd

from sklearn.preprocessing import StandardScaler,MinMaxScaler,OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

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]

}

df=pd.DataFrame(data)

print("Original Dataset:")

print(df)

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=pd.concat([df\_scaled\_standardized,df.drop(numeric\_columns,axis=1)],axis=1)

print("\nDataset after Feature Scaling:")

print(df\_scaled)

categorical\_columns=['Product','Category']

preprocessor=ColumnTransformer(

transformers=[

('categorical',OneHotEncoder(),categorical\_columns)

],

remainder='passthrough'

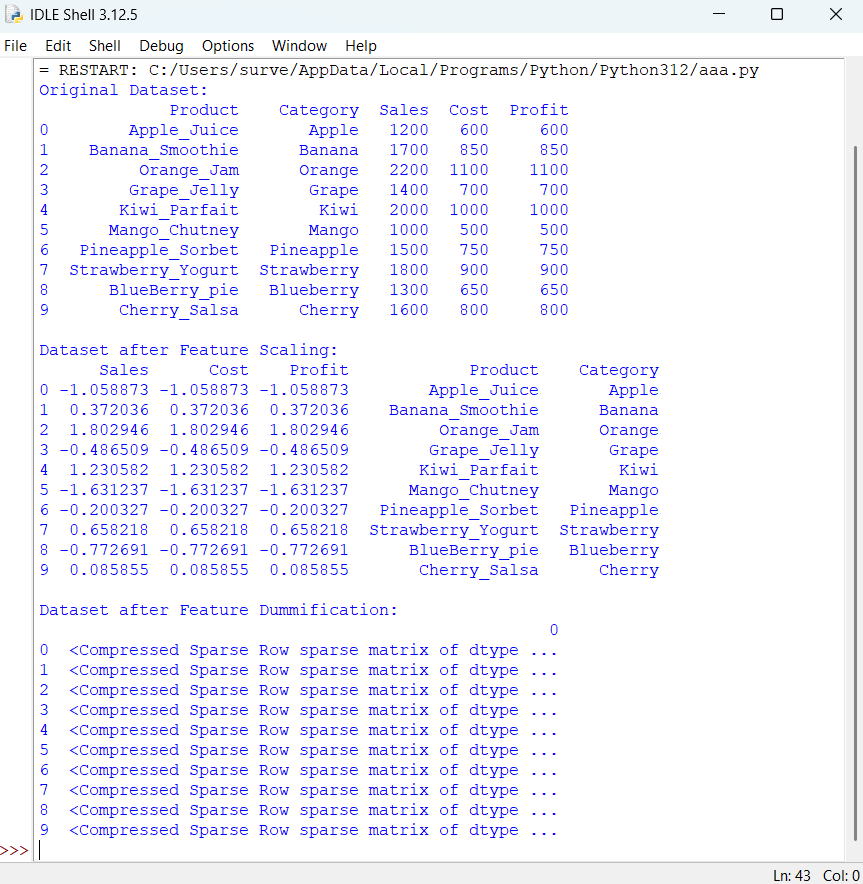
)

df\_dummified=pd.DataFrame(preprocessor.fit\_transform(df))

print("\nDataset after Feature Dummification:")

print(df\_dummified)

**Output:**



**Practical No.03**

**Date: / /2024 Roll No.:17**

**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.**

**Code:**

import numpy as np

from scipy import stats

from scipy import stats

import matplotlib.pyplot as plt

np.random.seed(42)

sample1=np.random.normal(loc=10,scale=2,size=30)

sample2=np.random.normal(loc=12,scale=2,size=30)

t\_statistic,p\_value=stats.ttest\_ind(sample1,sample2)

alpha=0.05

print("Results of Two-Sample t-test:")

print(f"t-statistic:{t\_statistic}")

print(f"Degrees of Freedom:{len(sample1)+len(sample2)-2}")

plt.figure(figsize=(10,6))

plt.hist(sample1,alpha=0.5,label='sample1',color='blue')

plt.hist(sample2,alpha=0.5,label='sample2',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()

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')

plt.text(11,5,f'T-statistic:{t\_statistic:.2f}',ha='center',va='bottom',backgroundcolor='white')

plt.show()

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 2 is significantly higher then theat of sample 2.")

else:

print("Conclusion:There is significnt evidence to reject the null hypothesis.")

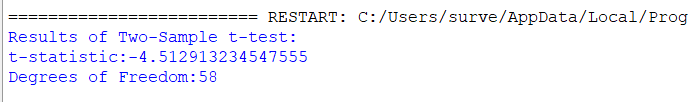
print("Interpretation:The mean caffeine content of Sample 2 is significantly higher than that of Sample 1.")

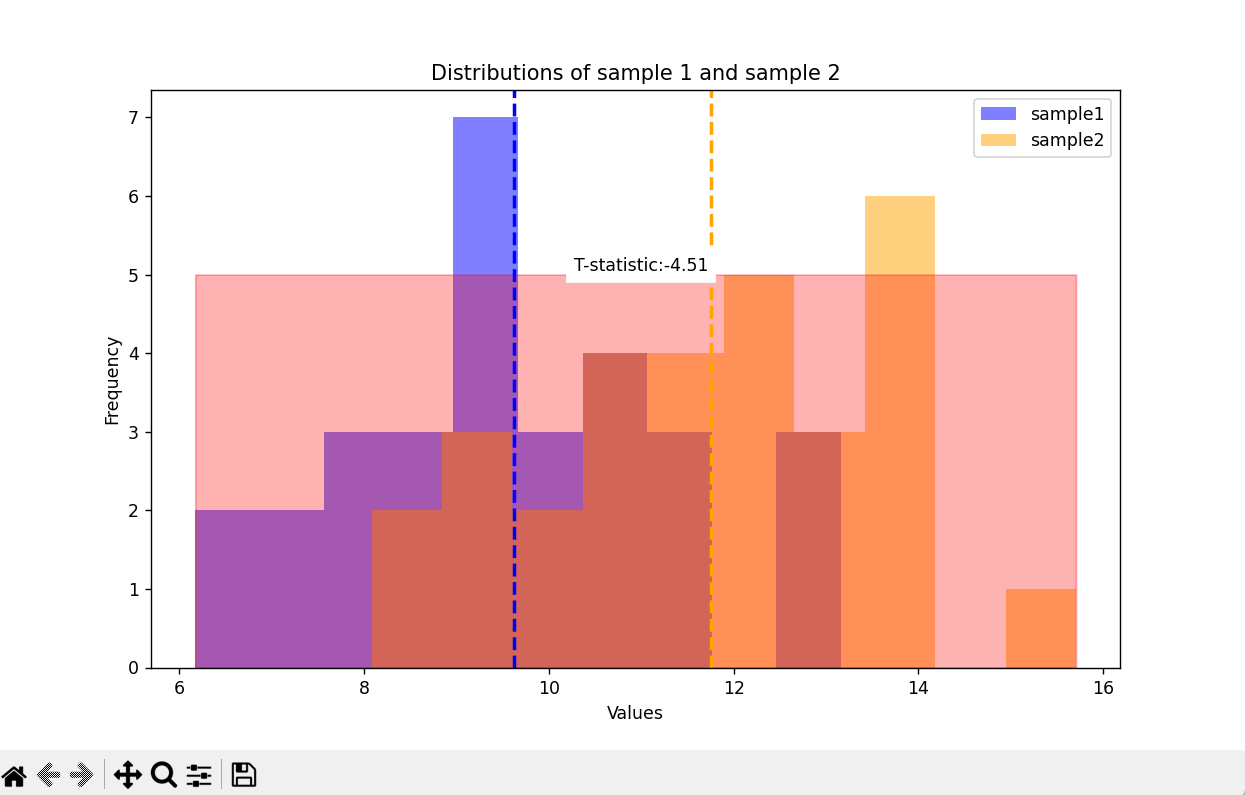
else:

print("conclusion:Fail to reject the null hypothesis.")

print("Interpretation:There is not enough evidence to claim a significant difference between the means.")

**Output:**





**Practical No.04**

**Date: / /2025 Roll No.:17**

**Aim: Regression and Its Types**

* **Implement simple linear regression using a dataset.**
* **Explore and interpret the regression model coefficients 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 modelling, hypothesis testing, and understanding casual 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 in Y.

The model equation for simple linear regression is typically represented as:

**Y=β0+ β1\*X+ε**

Where: Y is the dependent variable

X is the independent variable

β0is the intercept (the value of Y when X is 0)

β1 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 β0 and β1 the minimize the sum of 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.

1. **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 = β0 + β1 \* X1 + β2 \* X2 + … + βn \* Xn + ε**

Where: Y is the dependent variable

X1, X2, …, Xn are the independent variables

β0 is the intercept

β1, β2, …, βn 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. 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.

1. **Multiple Linear Regression:**

Select multiple independent variables (predictors) and one dependent variable (outcome).

1. Split the data into training and testing sets.
2. Fit a multiple linear regression model to the training data.
3. Interpret the coefficients of the regression model.
4. Assess the goodness-of-fit using metrics such as mean squared error (MSE) and R-squared.
5. Compare the performance of the multiple linear regression model with the simple linear regression model.
6. **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.

1. **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\_excel('D:\Datasets\DATA SET.xlsx')

#Simpler Linear Regression

#Select independent and dependent variables

X\_simple = data[['Cost']]

y\_simple = data['Sales']

text\_size=0.3

X\_train\_simple, X\_test\_simple, y\_train\_simple, y\_test\_simple = train\_test\_split(X\_simple, y\_simple, test\_size=text\_size, random\_state=42)

model\_simple = LinearRegression()

model\_simple.fit(X\_train\_simple, y\_train\_simple)

y\_pred\_simple = model\_simple.predict(X\_test\_simple)

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()

X\_multiple = data[['Cost', 'Profit']]

y\_multiple = data['Sales']

X\_train\_multiple, X\_test\_multiple, y\_train\_multiple, y\_test\_multiple = train\_test\_split(X\_multiple, y\_multiple, test\_size=text\_size, random\_state=42)

model\_multiple = LinearRegression()

model\_multiple.fit(X\_train\_multiple, y\_train\_multiple)

y\_pred\_multiple = model\_multiple.predict(X\_test\_multiple)

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 (Simple):', r2\_score(y\_test\_multiple, y\_pred\_multiple))

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()

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')

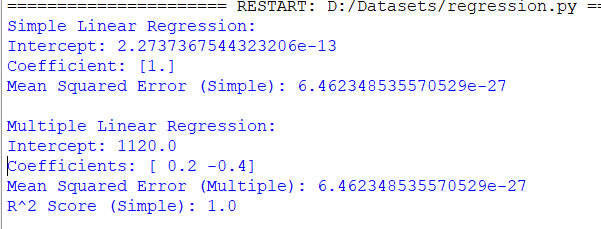
ax.set\_ylabel('Profit')

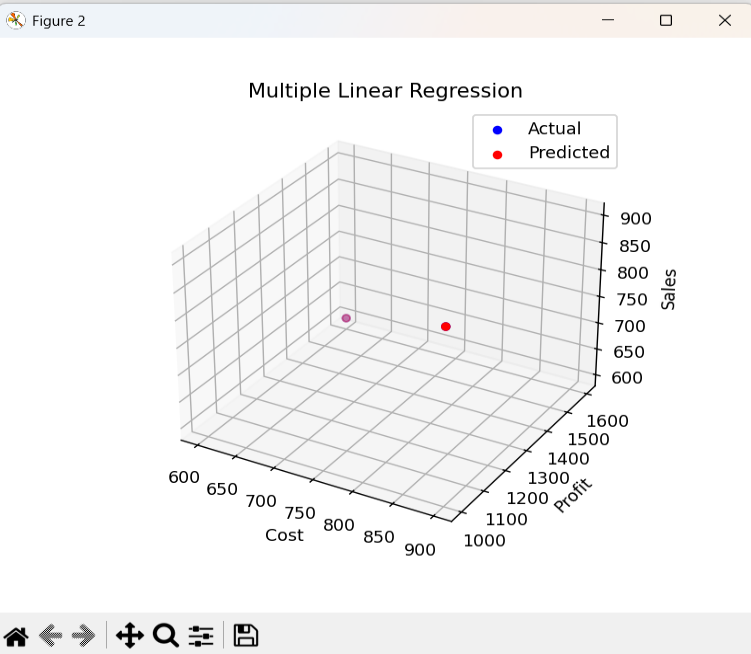
ax.set\_zlabel('Sales')

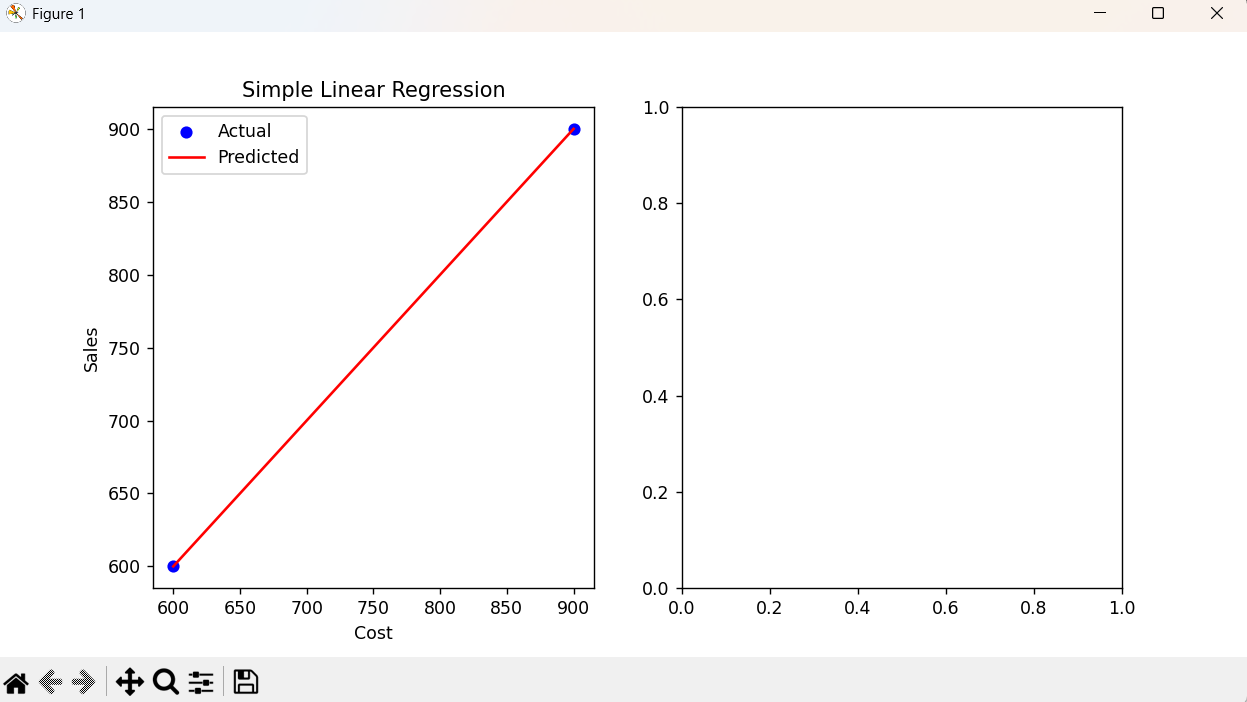
ax.legend()

plt.show()

**Output:**

****

****

****

**Practical No.05**

**Date: Roll No.:17**

**Aim: Data Frames and Basic Data Pre-processing**

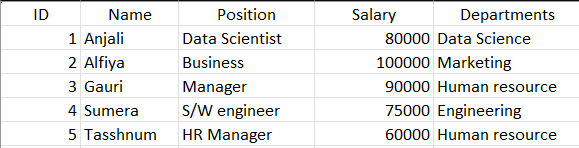
* **Read data from CSV and JSON files into a data frame.**
* **Perform basic data pre-processing tasks such as handling missing values and outliners.**
* **Manipulate and transform data using functions like filtering, sorting, and grouping.**

**Part 1: Read data from CSV and JSON files into a data frame.**

1. **CSV format**

In order to read the file from the CSV format into the data frame, we create the files in the given format

Consider the following CSV file (create a CSV file shown)



**We save the file as C** **Crescentt.csv and then execute the following code**

import pandas as pd

# Function to handle outliers

def handle\_outliers(column, factor=2):

median\_value = column.median()

upper\_threshold = column.mean() + factor \* column.std()

lower\_threshold = column.mean() - factor \* column.std() # Fix: subtract std for lower threshold

return column.apply(lambda x: median\_value if x > upper\_threshold or x < lower\_threshold else x)

# File paths

csv\_file\_path = "E:\\PRACTICAL\\Semester 6\\Data Science\\Data\\Sales.csv"

json\_file\_path = "E:\\PRACTICAL\\Semester 6\\Data Science\\Data\\Sales.JSON"

# Read CSV and JSON data

df\_csv = pd.read\_csv(csv\_file\_path)

df\_json = pd.read\_json(json\_file\_path)

# Print column names to check for 'Sales' column

print("Columns in CSV file:")

print(df\_csv.columns)

# Example of reading the first few rows of both files

print("CSV Data:")

print(df\_csv.head())

print("\nJSON Data:")

print(df\_json.head())

# Cleaning data: dropna and fillna

df\_csv\_cleaned = df\_csv.dropna()

df\_json\_filled = df\_json.fillna(0)

# Check and handle outliers in the 'Sales' column

# Assuming the column name in CSV is 'Sales', you can replace it with the actual column name

df\_csv['Sales'] = handle\_outliers(df\_csv['Sales'])

# Filter data based on 'Sales' value greater than 10

filtered\_data = df\_csv[df\_csv['Sales'] > 10]

# Sort data by 'Sales' in descending order

sorted\_data = df\_csv.sort\_values(by='Sales', ascending=False)

# Group by 'Category' and calculate mean for 'Sales', 'Cost', 'Profit'

numeric\_columns = ['Sales', 'Cost', 'Profit']

grouped\_data = df\_csv.groupby('Category')[numeric\_columns].mean()

# Display 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())

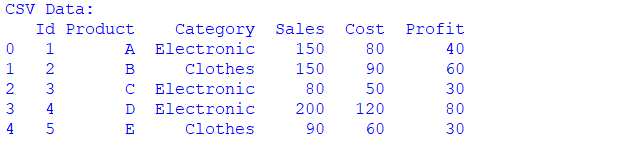
print("\nSorted Data:")

print(sorted\_data.head())

print("\nGrouped Data:")

print(grouped\_data.head())

**Output:**

****

1. **JSON format** (create a JSON file as shown)

In order to read the file from the JSON format into the data frame, we create the files in the given format

Consider the following JSON file,

{

"employees":[

{

"id":1,

"name":"Ganesh Agare",

"position":"Software Engineer",

"salary":75000,

"departments":["Engineering","Technology"]

},

{

"id":2,

"name":"Rehan nevrekar",

"position":"Data Scientist",

"salary":80000,

"departments":["Data Science","Research"]

},

{

"id":3,

"name":"Huzaif Khan",

"position":"Market Specialist",

"salary":65000,

"departments":["Marketing"]

},

{

"id":4,

"name":"Rahil kazi",

"position":"Financial Analyst",

"salary":75000,

"departments":["Finance"]

},

{

"id":5,

"name":"Atharva Kamble",

"position":"HR Manager",

"salary":60000,

"departments":["Human Resource"]

}

]

}

**We save the above files as Crescentt.json and execute the following code**

import json

json\_file\_path = "E:\\PRACTICAL\\Semester 6\\Data Science\\Data\\Crescent.JSON"

def read\_json\_file(file\_path):

with open(file\_path, 'r') as file:

data = json.load(file)

return data

try:

json\_data = read\_json\_file(json\_file\_path)

print("JSON Data:")

print(json.dumps(json\_data, indent=2))

except FileNotFoundError:

print(f"Error: File not found at path '{json\_file\_path}'")

except json.JSONDecodeError:

print(f"Error: Invalid JSON format in file at path '{json\_file\_path}'")

**Output:**

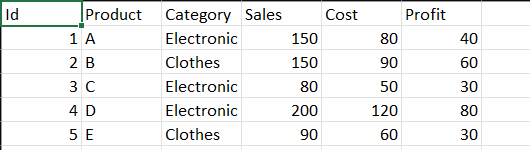
****

**Part 2: Perform basic data pre-processing tasks such as handling missing values and outliers.**

**Manipulate and transform data using functionality like filtering, sorting, and grouping.**

We Use the following CSV and JSON file to perform the above task (create a CSV and JSON file as shown)

**CSV file:**

****

**JSON File:**

{

"sales\_data": [

{

"ID": 1,

"Product": "A",

"Category": "Electronics",

"Sales": 120,

"Cost": 80,

"Profit": 40

},

{

"ID": 2,

"Product": "B",

"Category": "Clothing",

"Sales": 150,

"Cost": 90,

"Profit": 60

},

{

"ID": 3,

"Product": "C",

"Category": "Electronics",

"Sales": 80,

"Cost": 50,

"Profit": 30

},

{

"ID": 4,

"Product": "D",

"Category": "Clothing",

"Sales": 200,

"Cost": 120,

"Profit": 80

},

{

"ID": 5,

"Product": "E",

"Category": "Electronics",

"Sales": 90,

"Cost": 60,

"Profit": 30

}

]

}

**The following is the Python code for the given case**

import pandas as pd

# Function to handle outliers

def handle\_outliers(column, factor=2):

median\_value = column.median()

upper\_threshold = column.mean() + factor \* column.std()

lower\_threshold = column.mean() - factor \* column.std() # Fix: subtract std for lower threshold

return column.apply(lambda x: median\_value if x > upper\_threshold or x < lower\_threshold else x)

# File paths

csv\_file\_path = "E:\\PRACTICAL\\Semester 6\\Data Science\\Data\\Sales.csv"

json\_file\_path = "E:\\PRACTICAL\\Semester 6\\Data Science\\Data\\Sales.JSON"

# Read CSV and JSON data

df\_csv = pd.read\_csv(csv\_file\_path)

df\_json = pd.read\_json(json\_file\_path)

# Print column names to check for 'Sales' column

print("Columns in CSV file:")

print(df\_csv.columns)

# Example of reading the first few rows of both files

print("CSV Data:")

print(df\_csv.head())

print("\nJSON Data:")

print(df\_json.head())

# Cleaning data: dropna and fillna

df\_csv\_cleaned = df\_csv.dropna()

df\_json\_filled = df\_json.fillna(0)

# Check and handle outliers in the 'Sales' column

# Assuming the column name in CSV is 'Sales', you can replace it with the actual column name

df\_csv['Sales'] = handle\_outliers(df\_csv['Sales'])

# Filter data based on 'Sales' value greater than 10

filtered\_data = df\_csv[df\_csv['Sales'] > 10]

# Sort data by 'Sales' in descending order

sorted\_data = df\_csv.sort\_values(by='Sales', ascending=False)

# Group by 'Category' and calculate mean for 'Sales', 'Cost', 'Profit'

numeric\_columns = ['Sales', 'Cost', 'Profit']

grouped\_data = df\_csv.groupby('Category')[numeric\_columns].mean()

# Display 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())

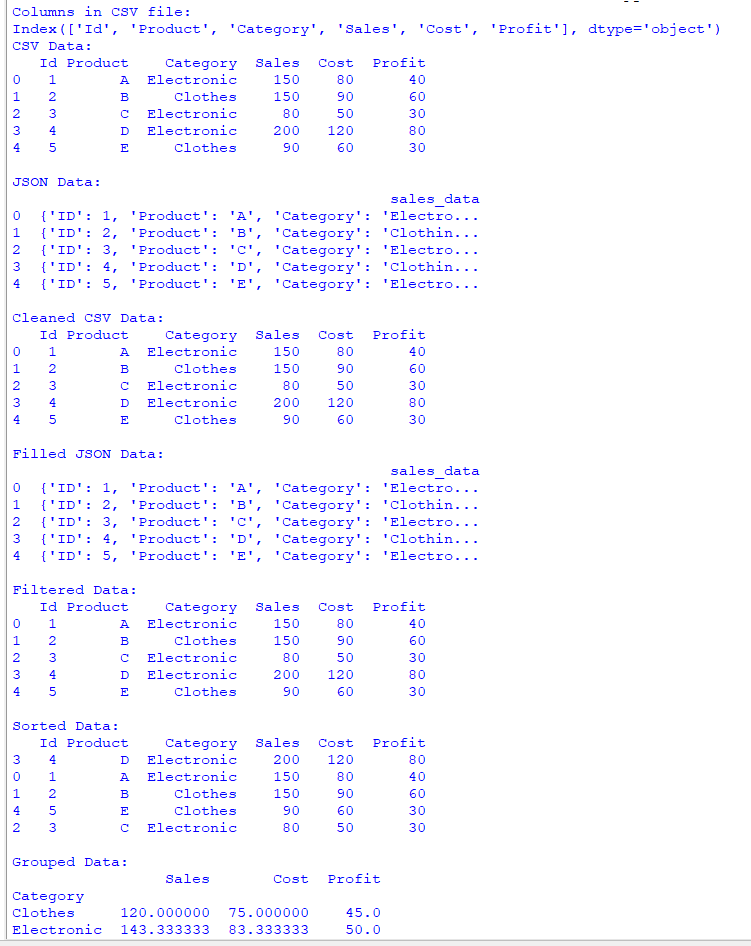
print("\nSorted Data:")

print(sorted\_data.head())

print("\nGrouped Data:")

print(grouped\_data.head())

**Output:**



**Practical No.06**

**Date: Roll No.:17**

**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 modelling the probability of the default class (usually labelled 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).

1. **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.

1. **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.

1. **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

data = pd.DataFrame({

'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Feature1': [0, 1, 1, 0, 1, 0, 0, 1, 0, 1],

'Target': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]

})

X = data.drop('Target', axis=1)

y = data['Target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

logistic\_model = LogisticRegression()

logistic\_model.fit(X\_train, y\_train)

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 = DecisionTreeClassifier()

decision\_tree\_model.fit(X\_train, y\_train)

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)

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()

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_dt, annot=True, cmap='Blues', fmt='g')

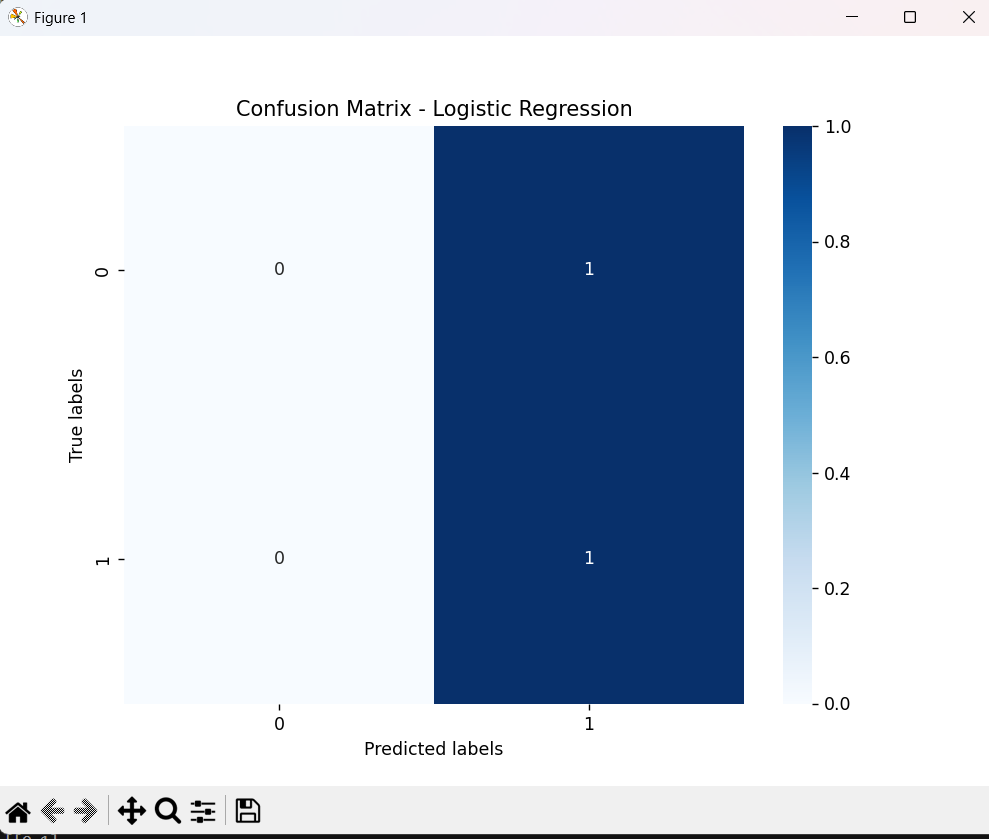
plt.xlabel('Predicted labels')

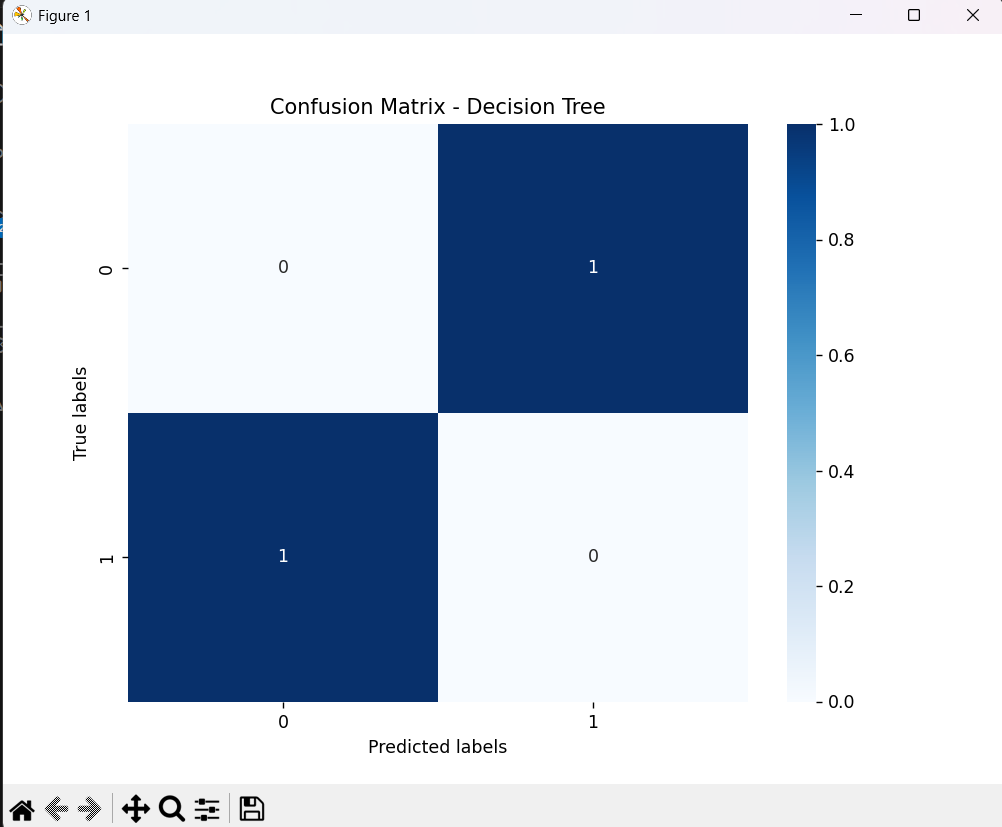
plt.ylabel('True labels')

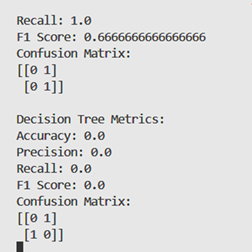
plt.title('Confusion Matrix - Decision Tree')

plt.show()

**Output:**





****

**Practical No.07**

**Date: Roll No.:17**

**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.

1. **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).

1. **Visualize Data in Reduced-Dimensional Space:**
2. Plot the data points in the reduced-dimensional space using the selected principal components.
3. Optionally, visualize the data with different colors or markers to represent different classes or groups.
4. 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

data = load\_iris()

X = data.data

y = data.target

pca = PCA()

X\_pca = pca.fit\_transform(X)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

cumulative\_variance = np.cumsum(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()

n\_components = np.argmax(cumulative\_variance >= 0.95) + 1

print("Number of principal components to retain:", n\_components)

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')

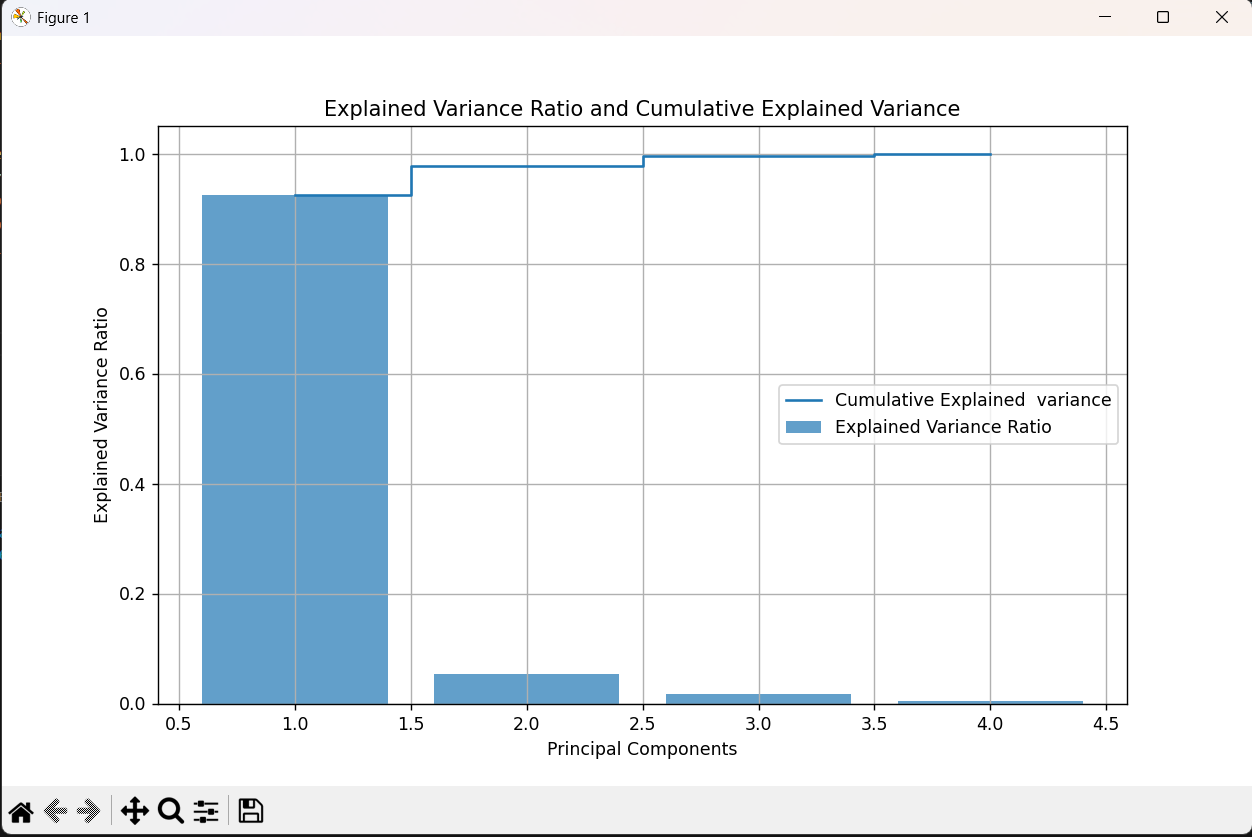
    plt.legend()

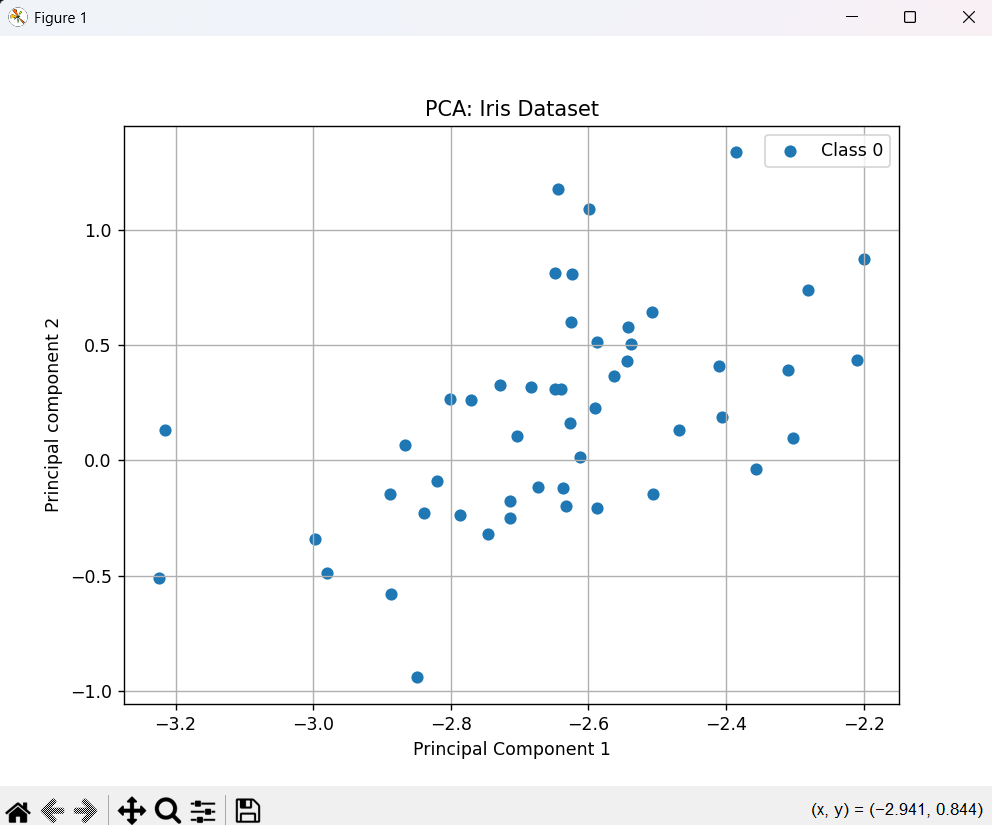
    plt.grid()

    plt.show()

**Output:**

****

****

****

**Practical No.8**

**Date: Roll No: 17**

**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.

1. **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.

1. **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.

1. **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.

1. **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 # type: ignore

import pandas as pd

import matplotlib.pyplot as plt

method\_a = [80, 82, 85, 78, 88]

method\_b = [75, 79, 82, 80, 81]

method\_c = [70, 75, 78, 72, 80]

data = pd.DataFrame({'Method A': method\_a, 'Method B': method\_b, 'Method C': method\_c})

f\_statistic, p\_value = f\_oneway(method\_a, method\_b, method\_c)

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.")

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)

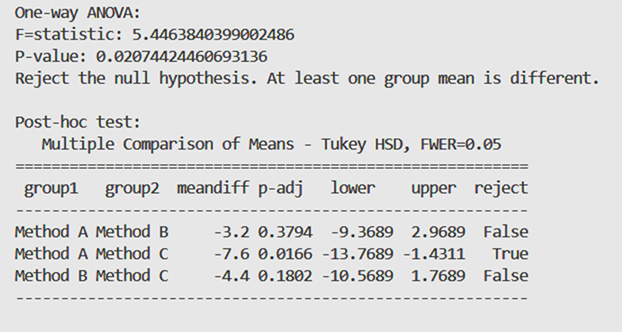
print("\nPost-hoc test:")

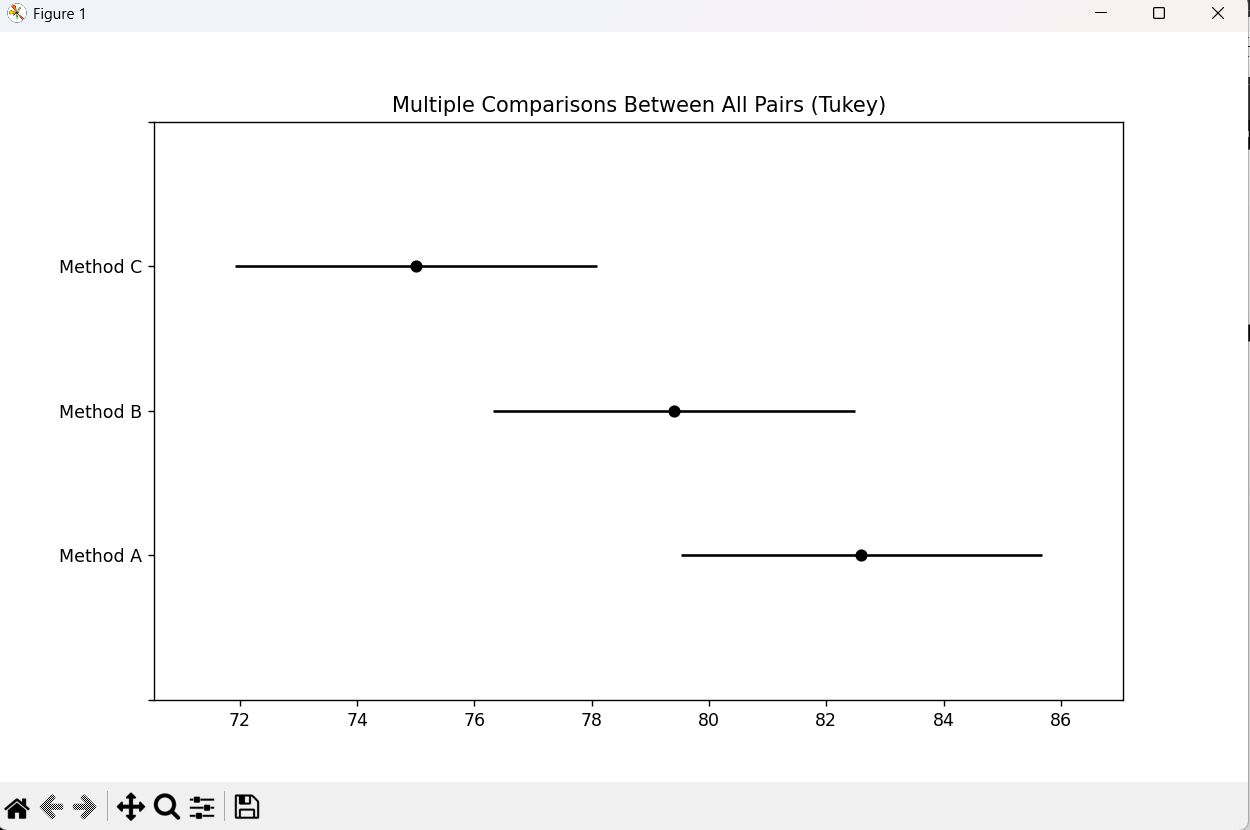
print(posthoc)

posthoc.plot\_simultaneous()

plt.show()

**Output:**

****

****

**PRACTICAL NO.9**

**Date: / /2025 Roll No.: 17**

**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

data = pd.read\_csv('D:\pract7\product\_sales\_data.csv')

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')

plt.show()

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)

plt.show()

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.")

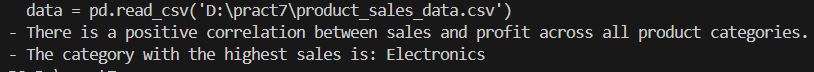
if data['Sales'].max() == data.loc[data['Sales'].idxmax(), 'Sales']:

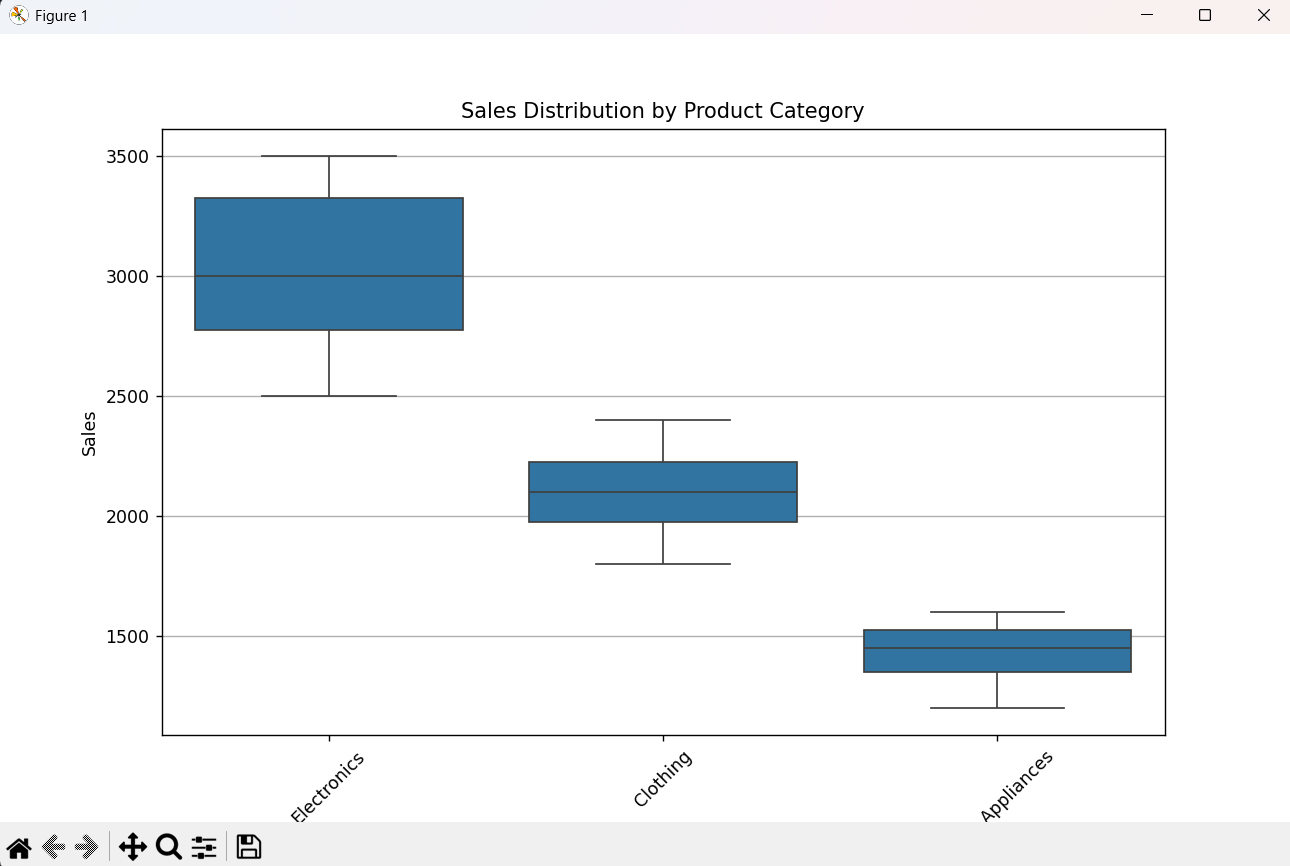
    print("- The category with the highest sales is:", data.loc[data['Sales'].idxmax(), 'Category'])

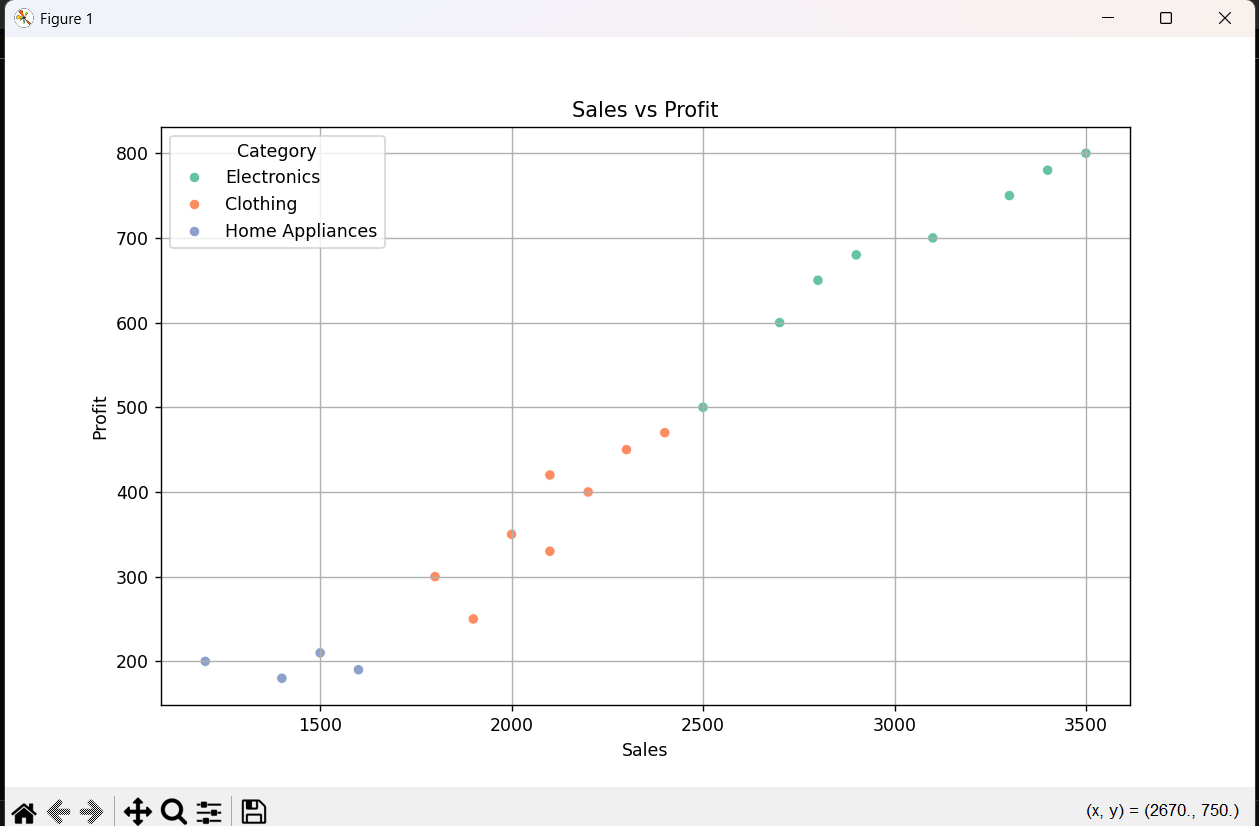
else:

    print("- The data does not meet any specific condition for insights.")

**Output:**

****

****

****

**PRACTICAL NO.10**

**Date:13/03/2024 Roll No.: 17**

**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 (centre 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.

1. **Preprocess Data:**

* If necessary, preprocess the data by handling missing values, scaling features, or encoding categorical variables.

1. **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.

1. **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.

1. **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.

1. **Analyse Cluster Characteristics:**

* Analyse 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.

1. **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

X, \_ = make\_blobs(n\_samples = 300, centers = 4, cluster\_std = 0.60, random\_state = 0)

wcss = []

silhouette\_scores = []

for i in range(2, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 10, n\_init = 10, random\_state = 0)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

silhouette\_scores.append(silhouette\_score(X, kmeans.labels\_))

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')

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()

optimal\_num\_clusters = np.argmax(silhouette\_scores) + 2

print("Optimal number of clusters:", optimal\_num\_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)

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')

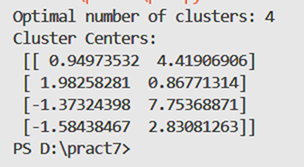
plt.xlabel('Feature 1')

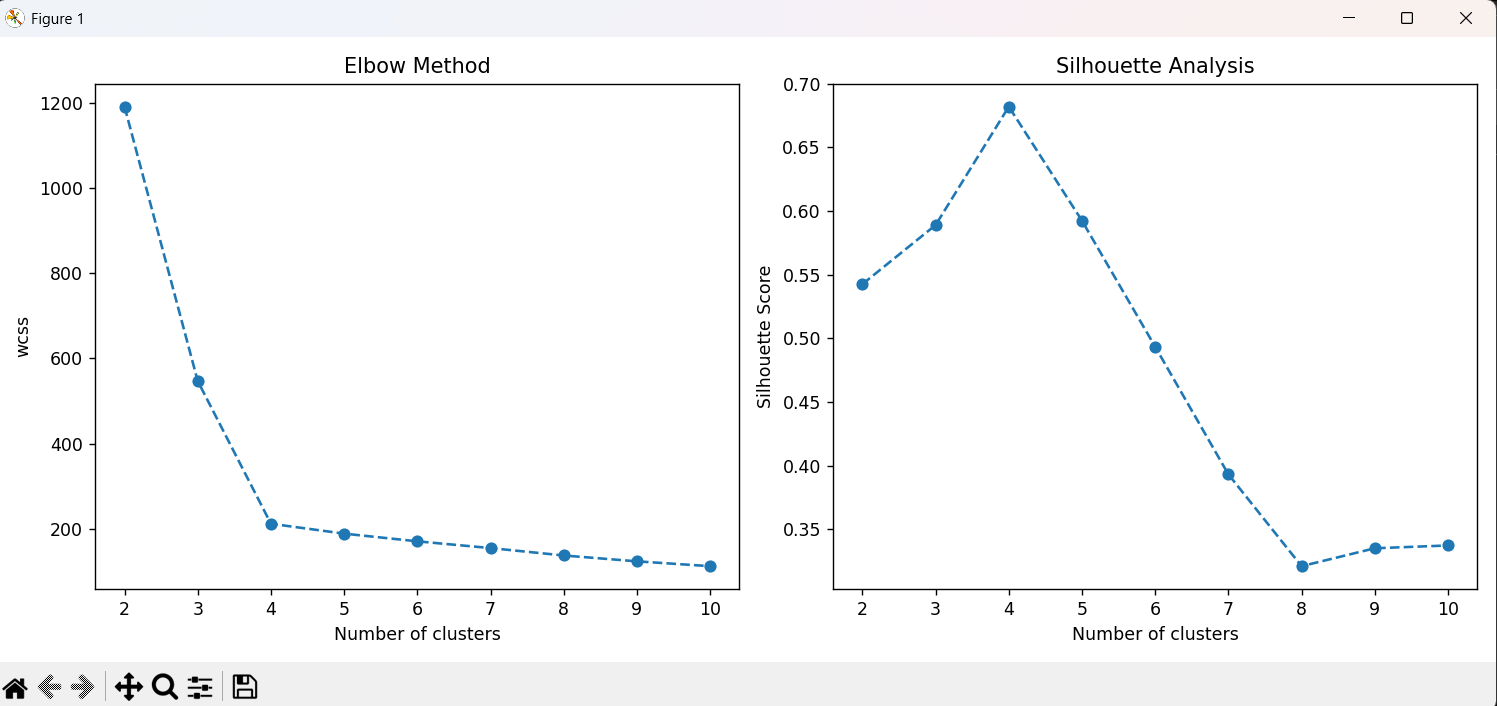
plt.ylabel('Feature 2')

plt.show()

print("Cluster Centers:\n", centers)

**Output:**

****

****

****