PRACTICAL NO 1

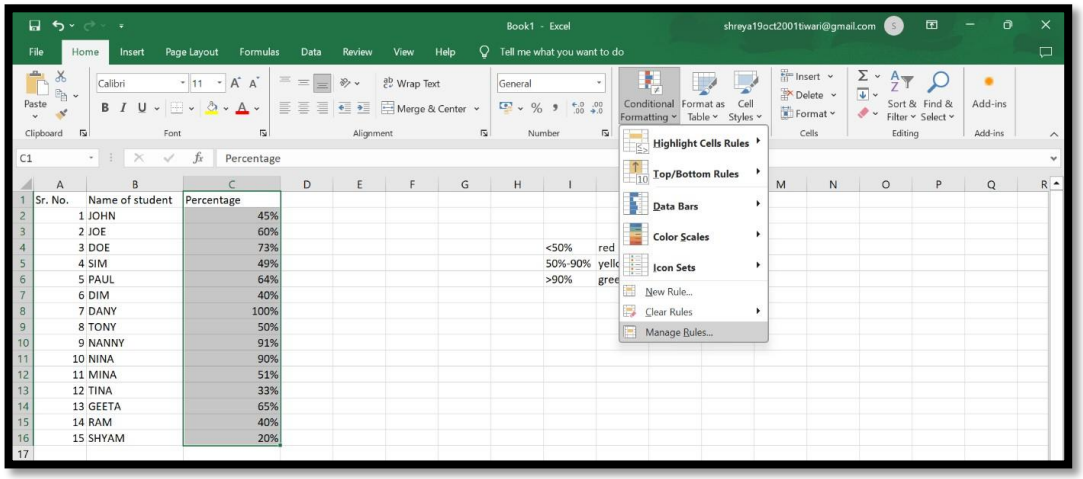
AIM: INTRODUCTION TO EXCEL

a) Perform conditional formatting on a dataset using various criteria

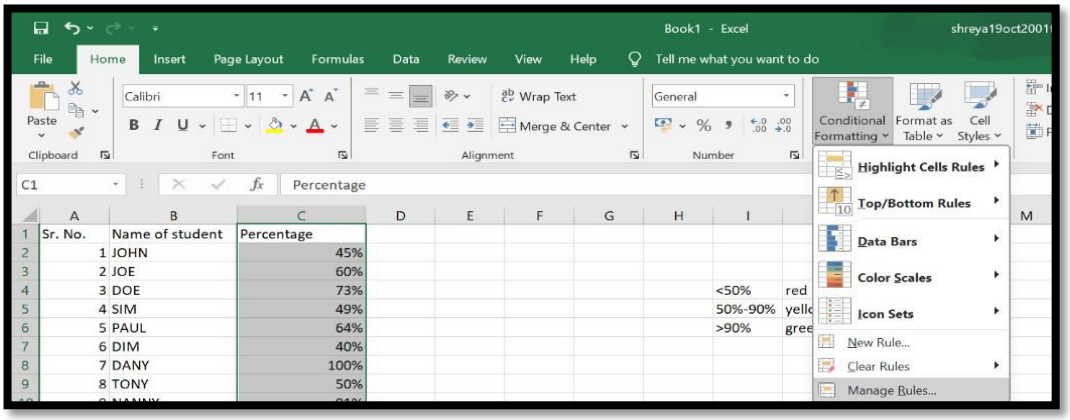
We perform conditional formatting on the "Percentage" column to highlight cells to RED if percentage is less than 50%, YELLOW if percentage is between 50% to 90% and Green if percentage is greater than 90%

Steps:

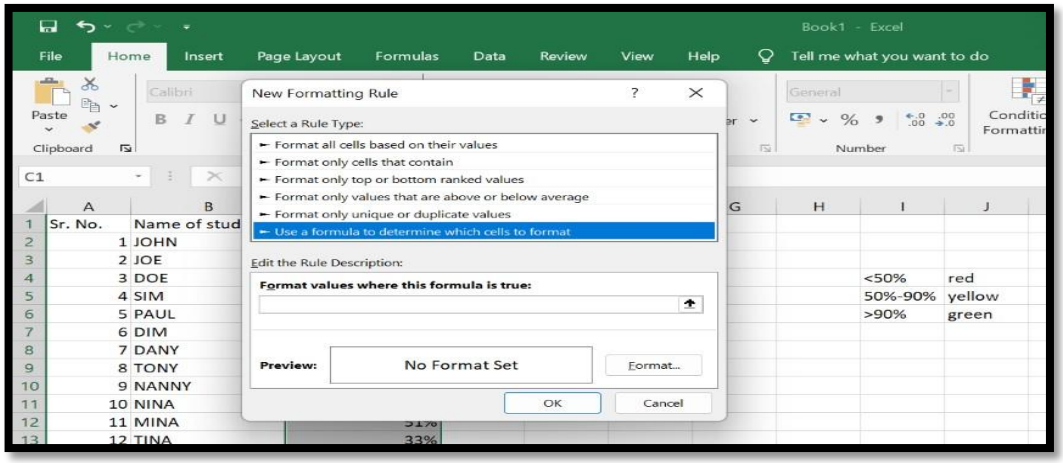
1. Select the "Percentage" column
2. Go to the "Home" tab on the ribbon
3. Click on the "Conditional formatting



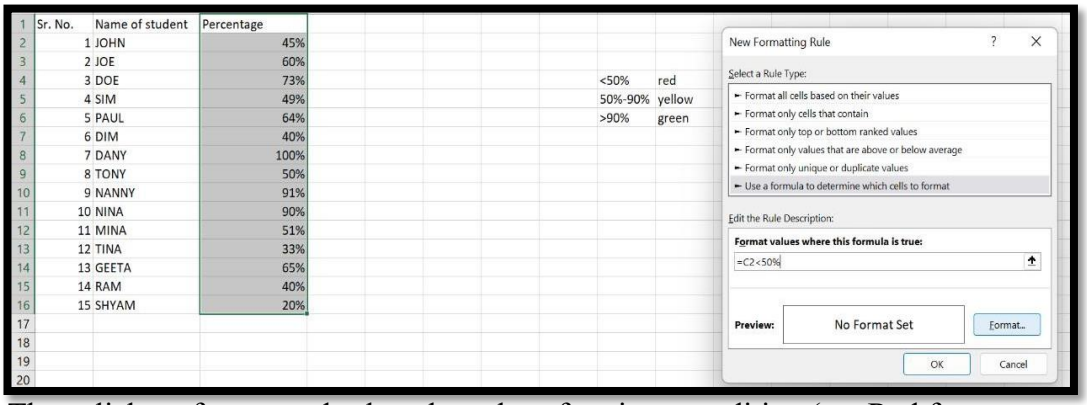
4. Choose "Manage rules"



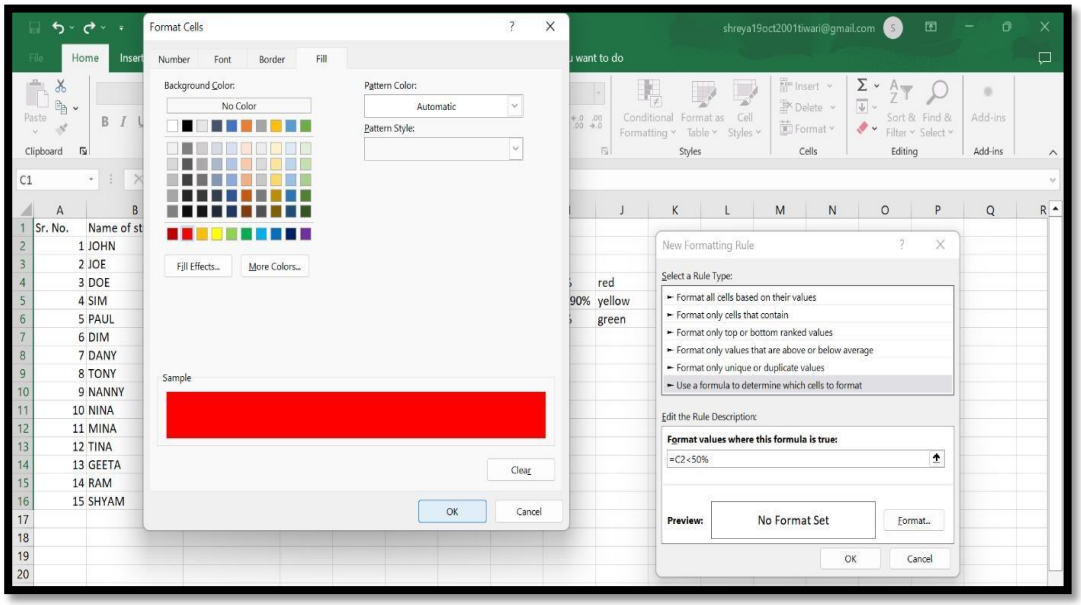
5. Then click on "Use a formula to determine which cells to format"

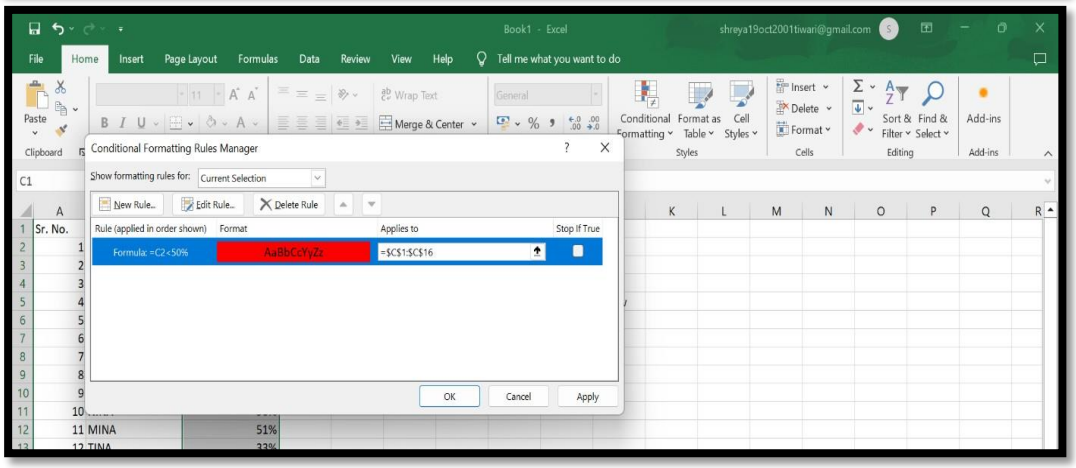


6. Write the condition of highlighting by selecting the cell (eg. =C2<50%)



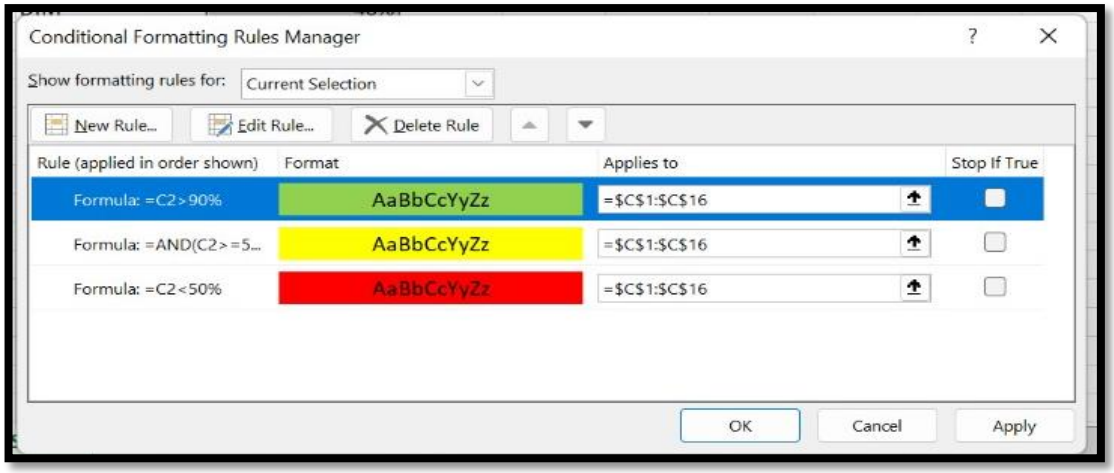
7. Then click on format and select the colour for given condition (eg: Red for percentage less than 50) and then click ok.

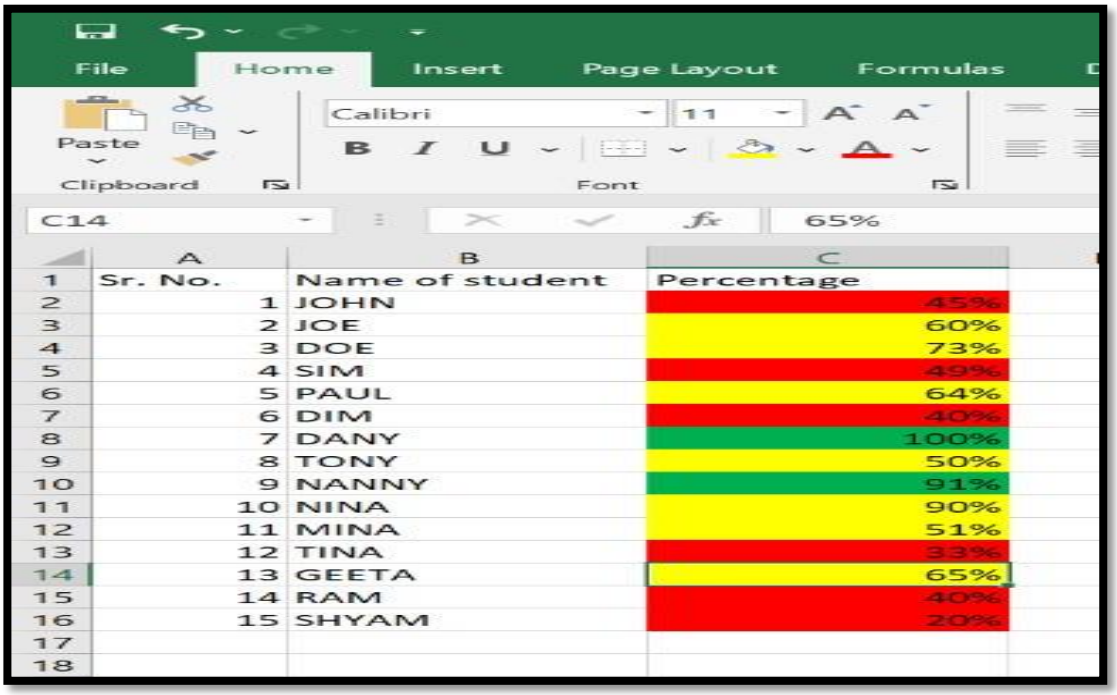


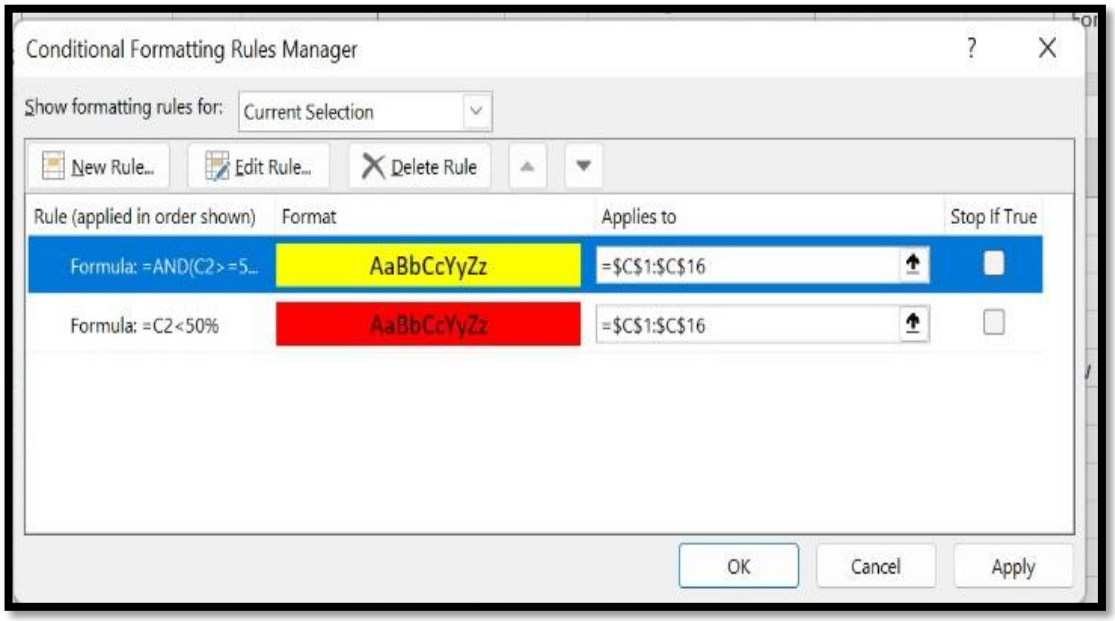


8. Next click on apply and repeat the same procedure for applying other conditions (eg: = AND(C2>=50%,C2<=90%)) selecting yellow colour.

9. Click on “ok” to apply the rule



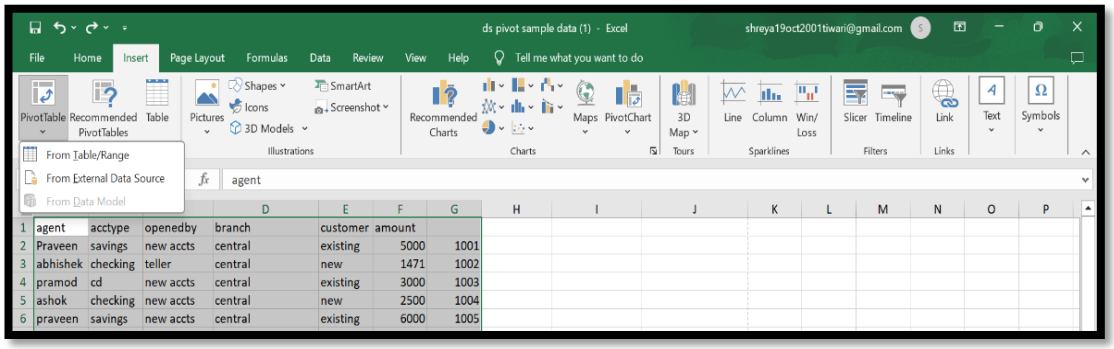




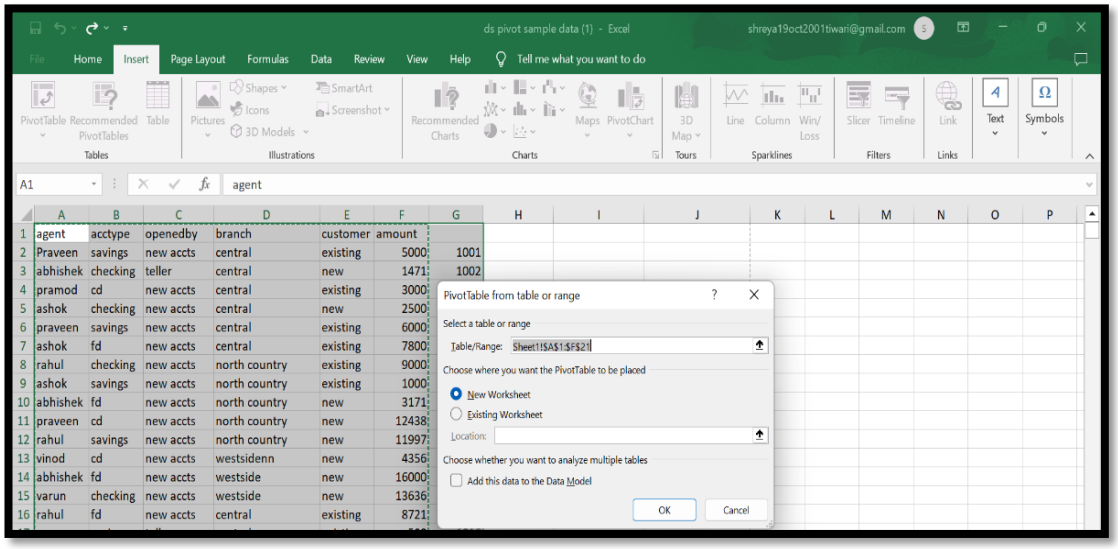
b) Create a pivot table to analyze and summarize data.

Steps:

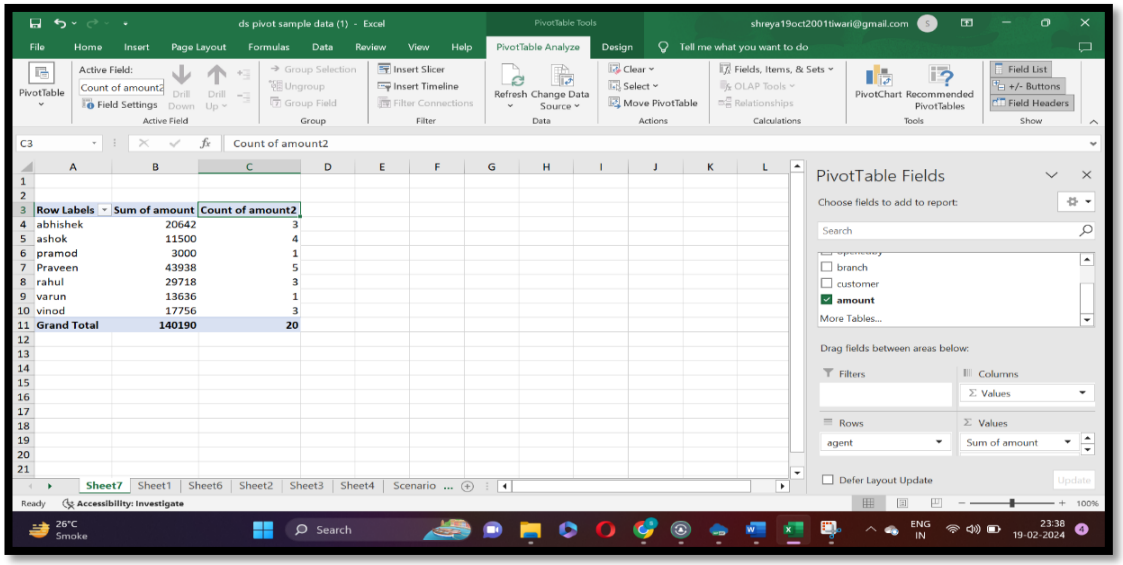
1. Select the entire dataset including headers.
2. Go to the “Insert” tab on the ribbon.
3. Click on “PivotTable”



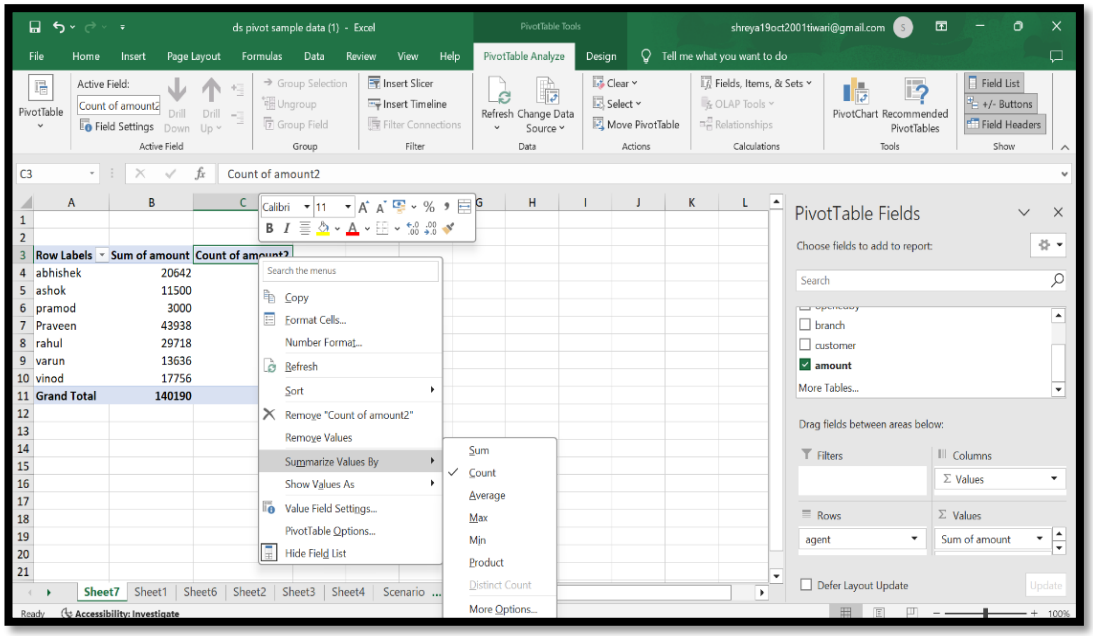
4. Choose whether you want to place the PivotTable (e.g. new worksheet)

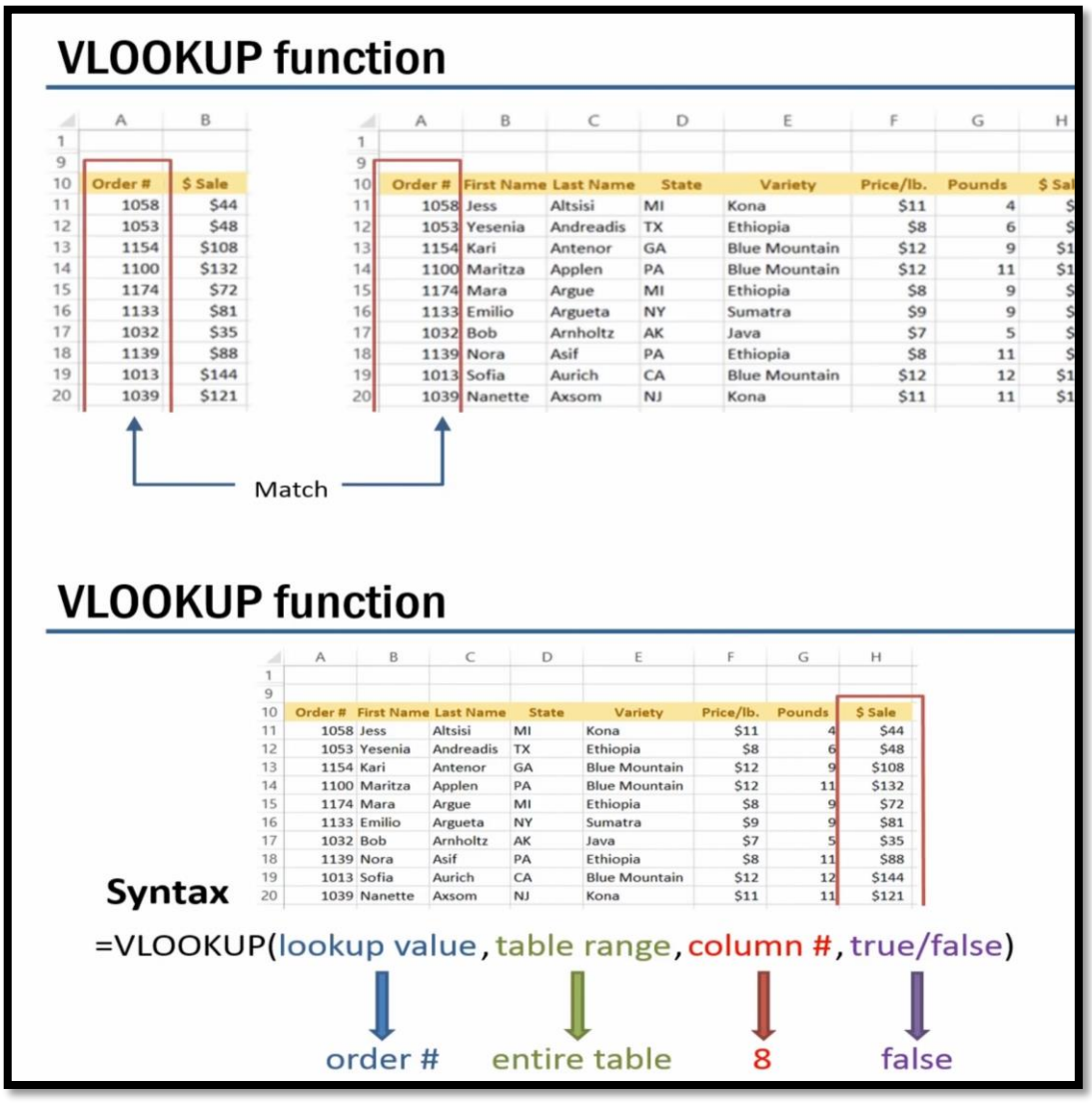


5. Drag “agent” to the Rows area
6. Drag “amount” to the Values area choosing sum function



7. Click on “summarize values by” on “amount” to apply various function (e.g. sum,count etc.)



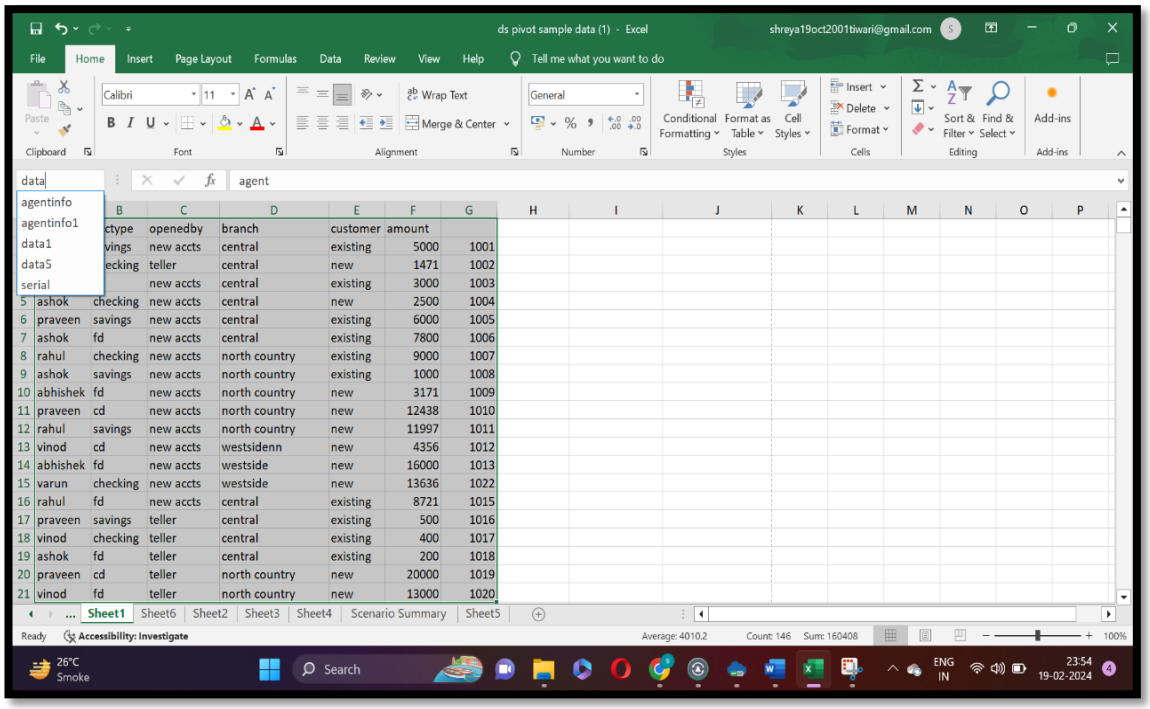


c) Use VLOOKUP function to retrieve information from a different worksheet or table

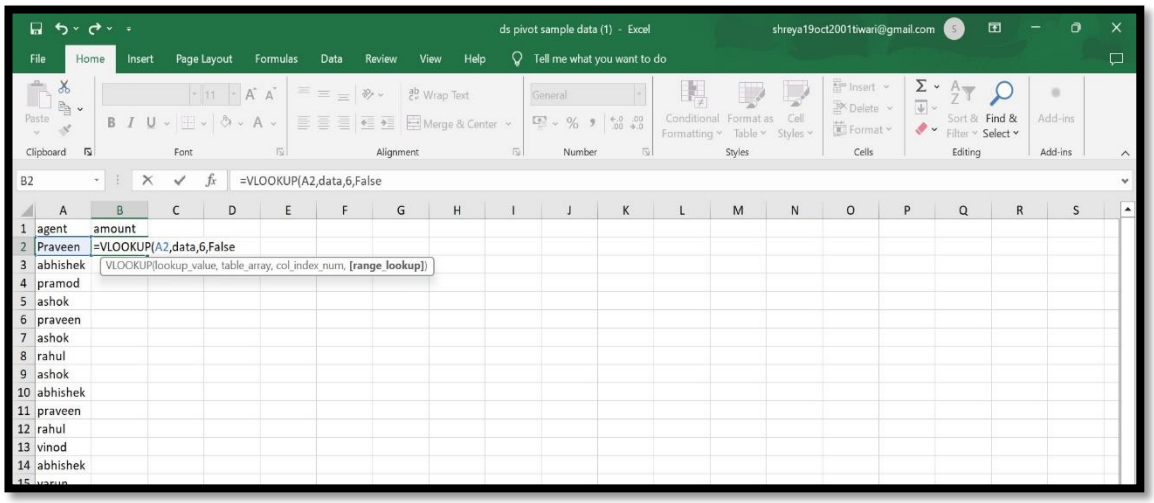
Use the VLOOKUP function to retrieve the agents “amount” from a separate table named “data” using following steps:

Steps:

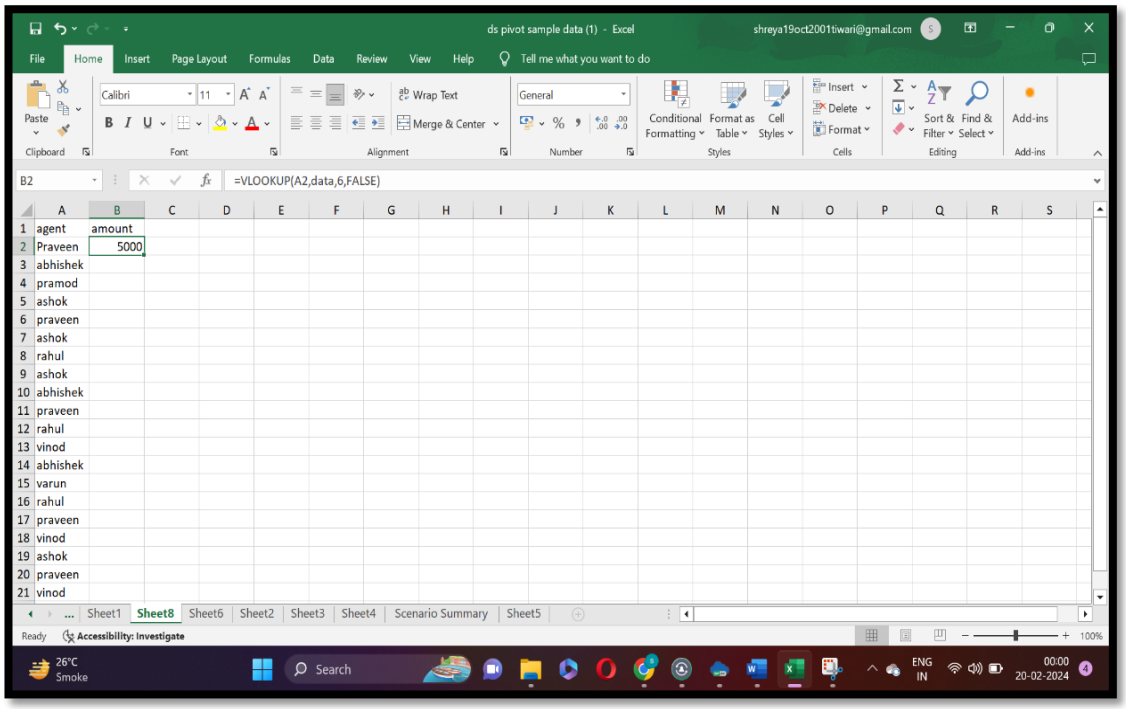
1) Assuming your “data” is in different sheet by selecting the entire table and changing the name to “data”

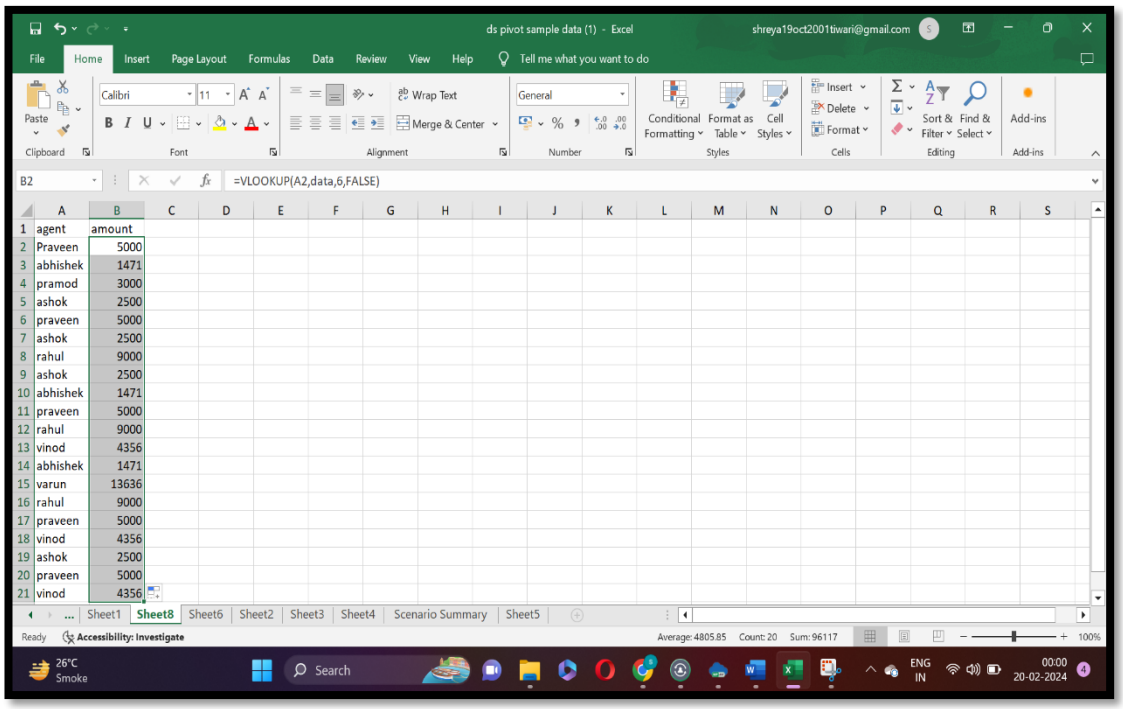


2) Open a new sheet with same column(agent) i.e common column, enter the formula
=VLOOKUP(A2,data,6,FALSE) for retrieving the agents “amount” data to the news sheet



3) Then click on Enter



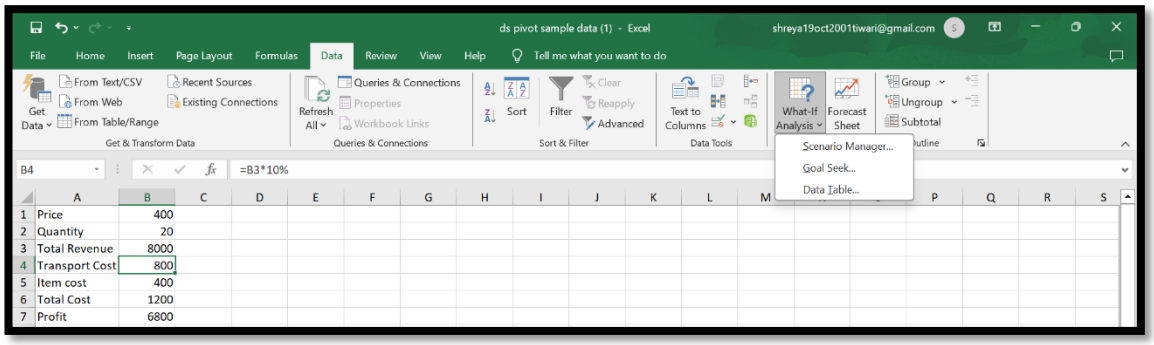


d) Perform what-if analysis using Goal seek and scenario manager to determine input vales for desired output.

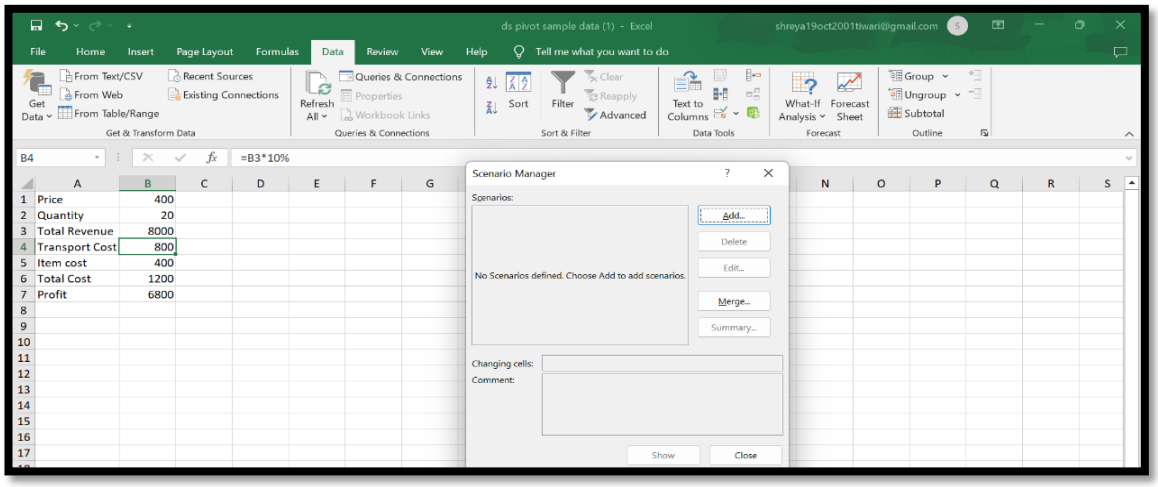
Use Scenario manger to see the scenario of change in quantities of Product

Steps:

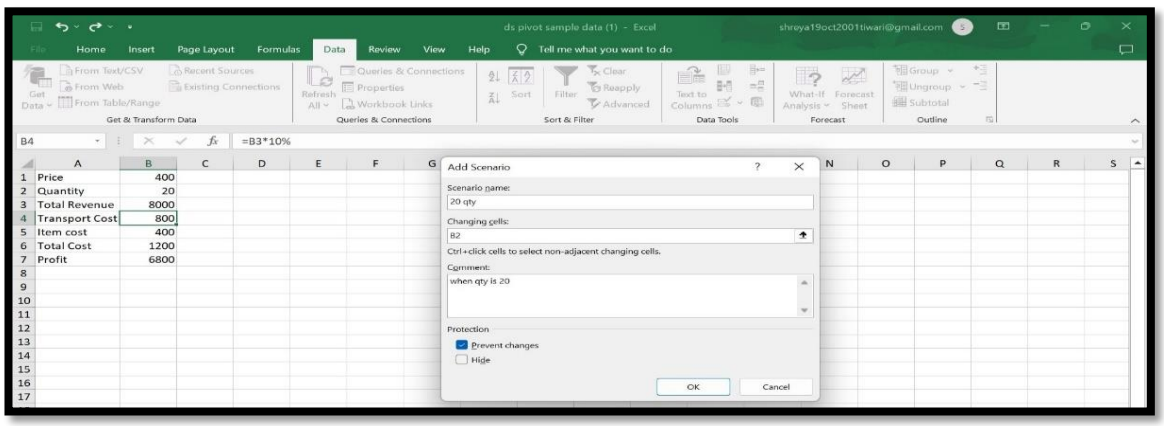
1. Identify the quantity cell from the given table and click on what-if-analysis



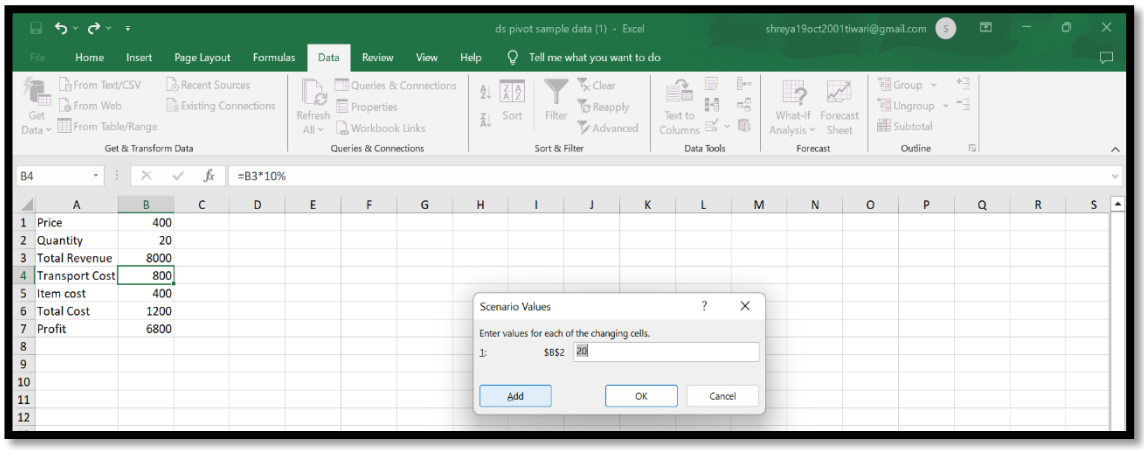
2. Click on “Scenario Manager”



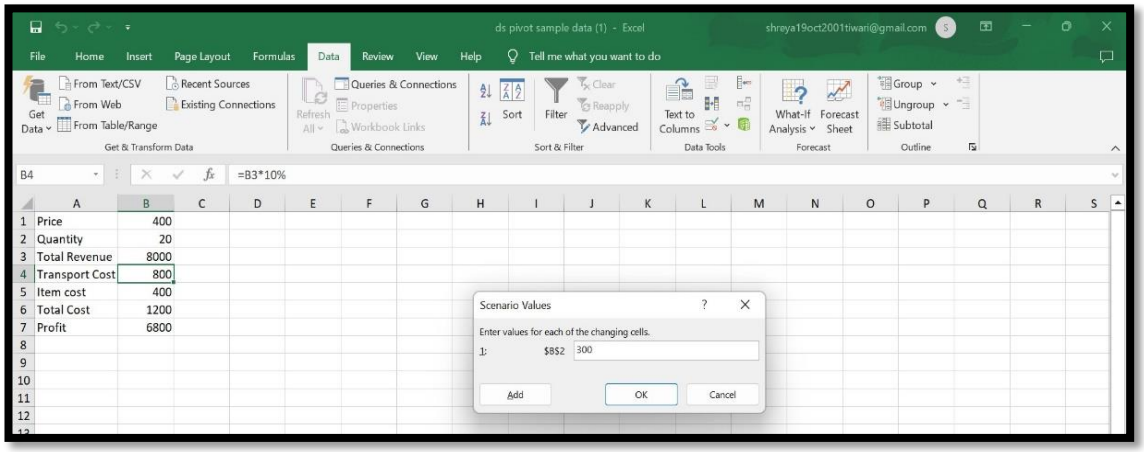
3. Click on “Add” and give the required name and comment and specify the cell of “quantity”



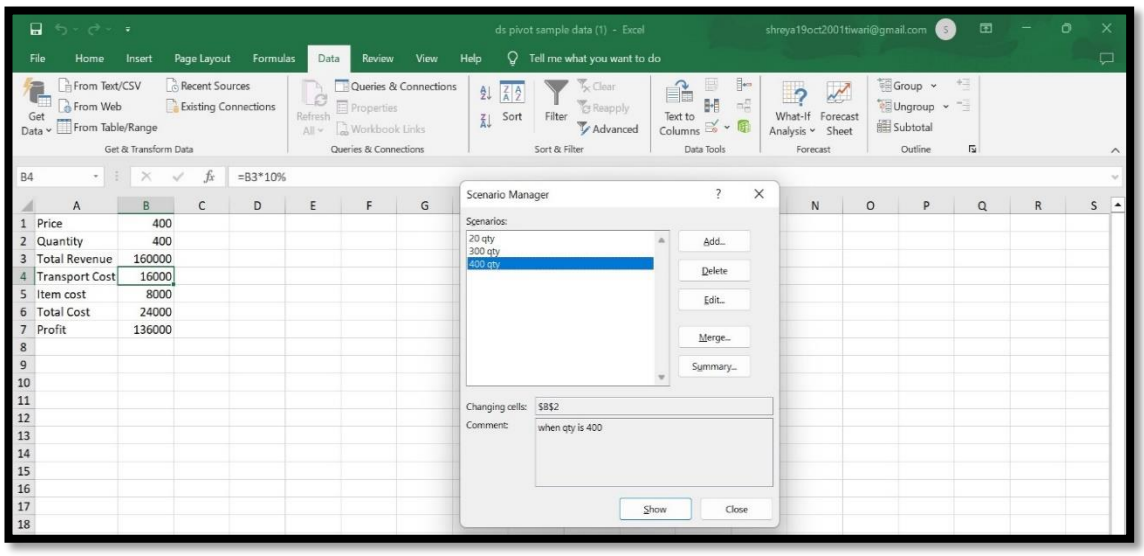
4. Click on “ok” and specify the quantity (e.g 20)



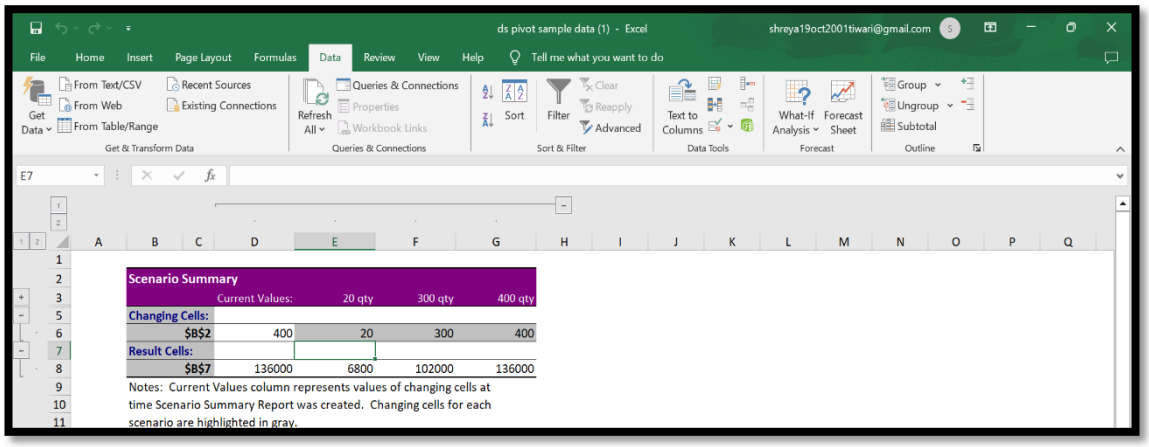
5. Then click on “add” and repeat the same procedure for various quantities (eg. 300, 400)



6. To see the scenario for changing quantities, select any quantity and click on show.(e.g scenario of 400 quantity)



7. Then click on “summary” to see the entire scenario

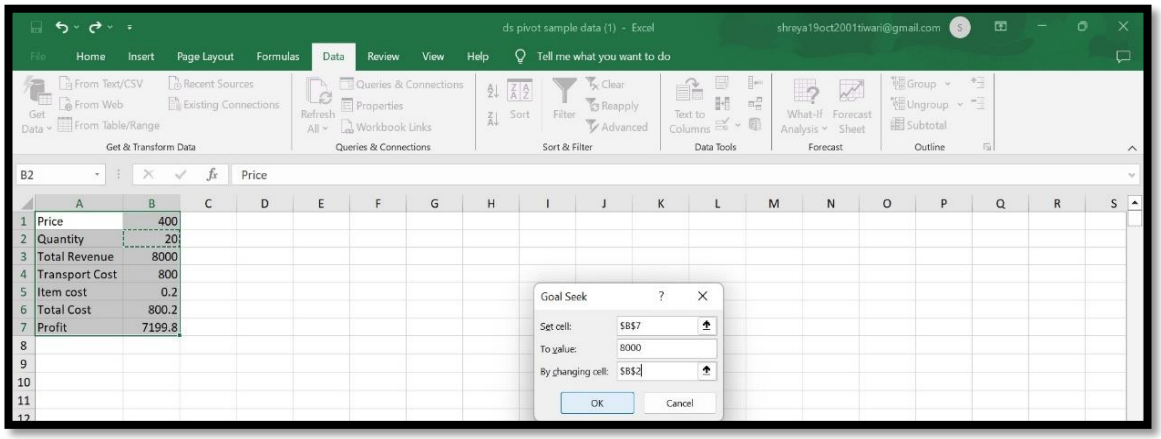


Scenario Summary				
	Current Values:	20 qty	300 qty	400 qty
Changing Cells:				
B2	400	20	300	400
Result Cells:				
B7	136000	6800	102000	136000

Notes: Current Values column represents values of changing cells at time Scenario Summary Report was created. Changing cells for each scenario are highlighted in gray.

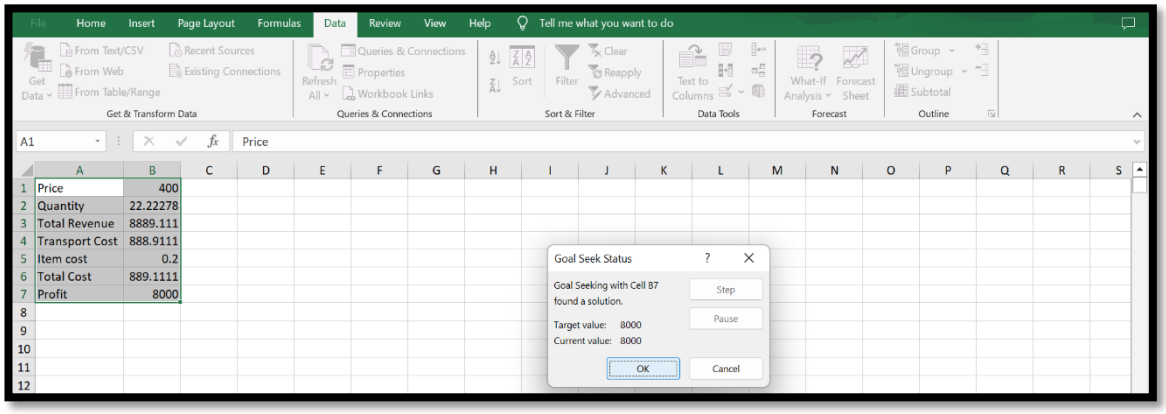
8. Use goal seek to achieve the profit of 8000

   i. Click on “Goal Seek” , then click on “Profit” in “set cell” , “8000” in values” and click on “Quantity” in “By Changing cell” to get the required output.



	A	B
1	Price	400
2	Quantity	20
3	Total Revenue	8000
4	Transport Cost	800
5	Item cost	0.2
6	Total Cost	800.2
7	Profit	7199.8

   ii. Click on “OK”



	A	B
1	Price	400
2	Quantity	22.22278
3	Total Revenue	8889.111
4	Transport Cost	888.9111
5	Item cost	0.2
6	Total Cost	889.1111
7	Profit	8000

PRACTICAL NO 2

AIM: DATA FRAMES AND BASIC DATA PRE-PROCESSING

a) Read data from CSV and JSON file into data frame

CSV file: Book1.csv

customer	amount	bank
existing	5000	boi
new	1471	boi
existing	3000	boi
new		boi
existing		boi
existing		boi
existing		msb
existing		msb
new		msb
new	12438	msb
new	11997	msb
new	4356	india bank
new	16000	india bank
new	13636	india bank
existing	8721	india bank
existing	500	india bank
existing	400	india bank
existing	200	india bank
new	20000	india bank
new	13000	india bank

JSON file: tycs.json

```
{
 "name": "my-react-app",
 "version": "0.1.0",
 "private": true,
 "dependencies":
{
        "@testing-library/jest-dom": "^5.17.0",
        "@testing-library/react": "^13.4.0",
        "@testing-library/user-event": "^13.5.0",
        "antd": "^5.13.3",
        "history": "^5.3.0",
        "react": "^18.2.0",
        "react-dom": "^18.2.0",
        "react-router-dom": "^6.21.3",
        "react-scripts": "5.0.1",
        "web-vitals": "^2.1.4"
 }
}
```

CODE:

```
import pandas as pd
dataFrame = pd.read_csv("C:\\Users\\user\\Downloads\\Book1.csv")
print("Data frame from CSV file:\n", dataFrame)
dataFrame1 =
pd.read_json("C:\\Users\\user\\OneDrive\\Documents\\practicefolder\\tycs.json")
print("Data frame from JSON file:\n", dataFrame1)
```

OUTPUT:

```
Data frame from CSV file:
     customer   amount
0   existing   5000.0
1        new   1471.0
2   existing   3000.0
3        new      NaN
4   existing      NaN
5   existing      NaN
6   existing      NaN
7   existing      NaN
8        new      NaN
9        new  12438.0
10       new  11997.0
11       new   4356.0
12       new  16000.0
13       new  13636.0
14  existing   8721.0
15  existing    500.0
16  existing    400.0
17  existing    200.0
18       new  20000.0
19       new  13000.0
Data frame from JSON file:
                                    name version  private dependencies
@testing-library/jest-dom    my-react-app   0.1.0     True      ^5.17.0
@testing-library/react       my-react-app   0.1.0     True      ^13.4.0
@testing-library/user-event  my-react-app   0.1.0     True      ^13.5.0
antd                         my-react-app   0.1.0     True      ^5.13.3
history                      my-react-app   0.1.0     True       ^5.3.0
react                        my-react-app   0.1.0     True      ^18.2.0
react-dom                    my-react-app   0.1.0     True      ^18.2.0
react-router-dom             my-react-app   0.1.0     True      ^6.21.3
react-scripts                my-react-app   0.1.0     True        5.0.1
web-vitals                   my-react-app   0.1.0     True       ^2.1.4
```

b) Handling missing values

c) Manipulate and transform data using functions like filtering sorting and grouping

CODE:

```
# Read CSV file into data frame
dataFrame = pd.read_csv("C:\\Users\\Sanjay Tiwari\\Downloads\\Book1.csv")
print("Data frame from CSV file:\n", dataFrame)
print("\nShape of the data frame:", dataFrame.shape)
print("\nNull values in the data frame:\n", dataFrame.isnull())
print("\nNumber of null values in each column:\n", dataFrame.isnull().sum())
filled_dataFrame = dataFrame.fillna(value=0)
print("\nData frame after filling null values with 0:\n", filled_dataFrame)
filled_dataFrame = dataFrame.fillna(method="pad")
print("\nData frame after filling null values with previous values:\n", filled_dataFrame)
filled_dataFrame = dataFrame.fillna(method="bfill")
print("\nData frame after filling null values with next values:\n", filled_dataFrame)
filled_dataFrame = dataFrame.fillna(method="pad", axis=1)
print("\nData frame after filling null values with previous values of column:\n",
filled_dataFrame)
filled_dataFrame = dataFrame.fillna(method="bfill", axis=1)
print("\nData frame after filling null values with next values of next column:\n",
filled_dataFrame)

# Filtering
filtered_data = dataFrame.loc[dataFrame["amount"] < 500]
print("\nFiltered data where amount is less than 500:\n", filtered_data)
filtered_data = dataFrame.loc[(dataFrame["amount"] < 500) & (dataFrame["amount"] <
500)]
print("\nFiltered data where amount is less than 500 and amount is less than 500:\n",
filtered_data)
filtered_data = dataFrame.loc[dataFrame["customer"].str.contains("ist")]
print("\nFiltered data where customer column contains 'ist':\n", filtered_data)
filtered_data = dataFrame.loc[~dataFrame["customer"].str.contains("ist")]
print("\nFiltered data where customer column does not contain 'ist':\n", filtered_data)
filtered_data = dataFrame.loc[dataFrame["customer"].str.startswith("e")]
print("\nFiltered data where customer column starts with 'e':\n", filtered_data)
filtered_data = dataFrame.loc[dataFrame["customer"].str.endswith("w")]
print("\nFiltered data where customer column ends with 'w':\n", filtered_data)
# Sorting
dataFrame.sort_values(by='amount', ascending=False, inplace=True)
print("\nData frame after sorting by amount in descending order:\n", dataFrame)
# Sorting by multiple columns
dataFrame.sort_values(by=['customer', 'amount'], ascending=[True, False], inplace=True)
print("\nData frame after sorting by customer (ascending) and amount (descending):\n",
dataFrame)
# Sorting with null values at the end
dataFrame.sort_values(by='amount', ascending=False, na_position='last', inplace=True)
print("\nData frame after sorting by amount in descending order with null values at the
end:\n", dataFrame)
```

OUTPUT:

```
Shape of the data frame: (20, 3)

Null values in the data frame:
    customer  amount   bank
0      False   False  False
1      False   False  False
2      False   False  False
3      False    True  False
4      False    True  False
5      False    True  False
6      False    True  False
7      False    True  False
8      False    True  False
9      False   False  False
10     False   False  False
11     False   False  False
12     False   False  False
13     False   False  False
14     False   False  False
15     False   False  False
16     False   False  False
17     False   False  False
18     False   False  False
19     False   False  False

Number of null values in each column:
 customer     0
amount        6
bank          0
dtype: int64
```

```
Data frame after filling null values with 0:
    customer   amount        bank
0   existing   5000.0         boi
1        new   1471.0         boi
2   existing   3000.0         boi
3        new      0.0         boi
4   existing      0.0         boi
5   existing      0.0         boi
6   existing      0.0         msb
7   existing      0.0         msb
8        new      0.0         msb
9        new  12438.0         msb
10       new  11997.0         msb
11       new   4356.0  india bank
12       new  16000.0  india bank
13       new  13636.0  india bank
14  existing   8721.0  india bank
15  existing    500.0  india bank
16  existing    400.0  india bank
17  existing    200.0  india bank
18       new  20000.0  india bank
19       new  13000.0  india bank
```

```
Data frame after filling null values with previous values of column:
    customer    amount         bank
0   existing    5000.0          boi
1        new    1471.0          boi
2   existing    3000.0          boi
3        new       new          boi
4   existing  existing          boi
5   existing  existing          boi
6   existing  existing          msb
7   existing  existing          msb
8        new       new          msb
9        new   12438.0          msb
10       new   11997.0          msb
11       new    4356.0  india bank
12       new   16000.0  india bank
13       new   13636.0  india bank
14  existing    8721.0  india bank
15  existing     500.0  india bank
16  existing     400.0  india bank
17  existing     200.0  india bank
18       new   20000.0  india bank
19       new   13000.0  india bank

Data frame after filling null values with next values of next column:
    customer   amount         bank
0   existing   5000.0          boi
1        new   1471.0          boi
2   existing   3000.0          boi
3        new      boi          boi
4   existing      boi          boi
5   existing      boi          boi
6   existing      msb          msb
7   existing      msb          msb
```

```
Filtered data where amount is less than 500 and amount is less than 500:
    customer  amount        bank
16  existing   400.0  india bank
17  existing   200.0  india bank

Filtered data where customer column contains 'ist':
    customer  amount         bank
0   existing  5000.0          boi
2   existing  3000.0          boi
4   existing     NaN          boi
5   existing     NaN          boi
6   existing     NaN          msb
7   existing     NaN          msb
14  existing  8721.0  india bank
15  existing   500.0  india bank
16  existing   400.0  india bank
17  existing   200.0  india bank

Filtered data where customer column does not contain 'ist':
    customer   amount         bank
1        new   1471.0          boi
3        new      NaN          boi
8        new      NaN          msb
9        new  12438.0          msb
10       new  11997.0          msb
11       new   4356.0  india bank
```

```
Filtered data where customer column starts with 'e':
      customer   amount          bank
0    existing   5000.0           boi
2    existing   3000.0           boi
4    existing      NaN           boi
5    existing      NaN           boi
6    existing      NaN           msb
7    existing      NaN           msb
14   existing   8721.0   india bank
15   existing    500.0   india bank
16   existing    400.0   india bank
17   existing    200.0   india bank

Filtered data where customer column ends with 'w':
     customer    amount          bank
1         new   1471.0           boi
3         new      NaN           boi
8         new      NaN           msb
9         new  12438.0           msb
10        new  11997.0           msb
11        new   4356.0   india bank
12        new  16000.0   india bank
13        new  13636.0   india bank
18        new  20000.0   india bank
19        new  13000.0   india bank

Data frame after sorting by amount in descending order:
      customer    amount          bank
18         new  20000.0   india bank
12         new  16000.0   india bank
13         new  13636.0   india bank
19         new  13000.0   india bank
9          new  12438.0           msb
10         new  11997.0           msb
```

```
Data frame after sorting by customer (ascending) and amount (descending):
     customer   amount          bank
14  existing   8721.0   india bank
0   existing   5000.0           boi
2   existing   3000.0           boi
15  existing    500.0   india bank
16  existing    400.0   india bank
17  existing    200.0   india bank
4   existing      NaN           boi
5   existing      NaN           boi
6   existing      NaN           msb
7   existing      NaN           msb
18       new  20000.0   india bank
12       new  16000.0   india bank
13       new  13636.0   india bank
19       new  13000.0   india bank
9        new  12438.0           msb
10       new  11997.0           msb
11       new   4356.0   india bank
1        new   1471.0           boi
3        new      NaN           boi
8        new      NaN           msb

Data frame after sorting by amount in descending order with null values at the end:
     customer   amount          bank
18       new  20000.0   india bank
12       new  16000.0   india bank
13       new  13636.0   india bank
19       new  13000.0   india bank
9        new  12438.0           msb
10       new  11997.0           msb
14  existing   8721.0   india bank
0   existing   5000.0           boi
11       new   4356.0   india bank
2   existing   3000.0           boi
1        new   1471.0           boi
15  existing    500.0   india bank
16  existing    400.0   india bank
17  existing    200.0   india bank
4   existing      NaN           boi
5   existing      NaN           boi
```

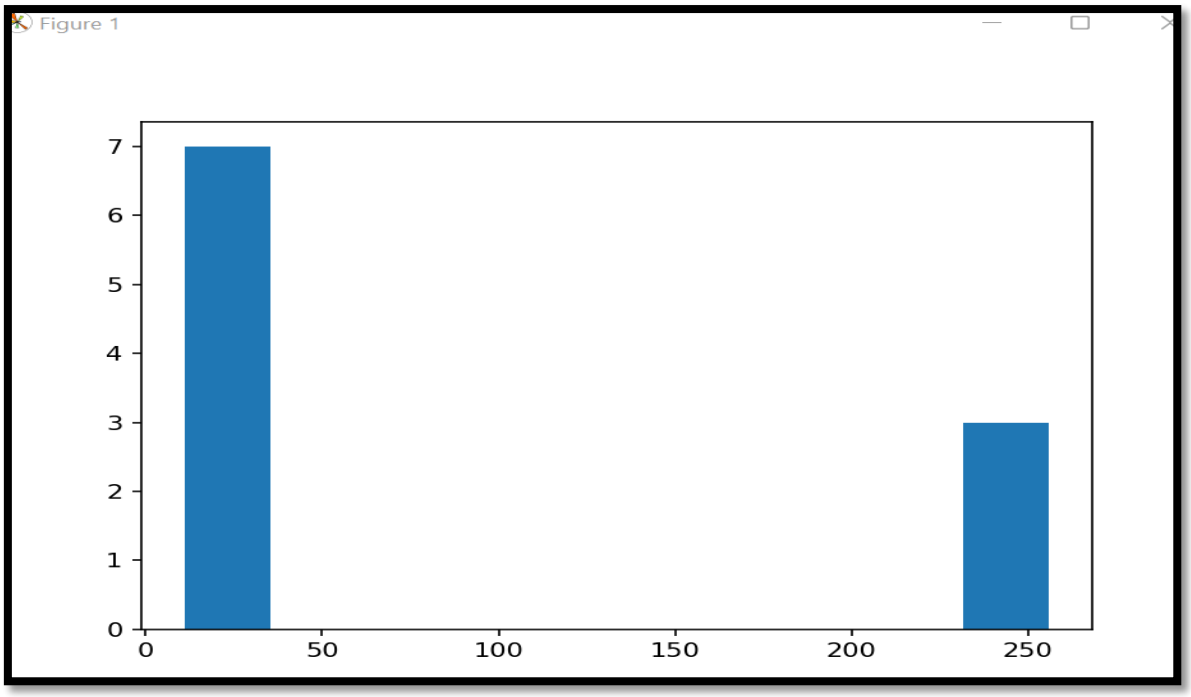
d) Handling outliers

(i) Using Z-score

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
datasets = [11, 17, 16, 15, 14, 13, 12, 256, 255, 250]
plt.hist(datasets)
plt.show()
outliers = []
threshold = 1  # 3rd standard deviation
mean = np.mean(datasets)
std = np.std(datasets)
for i in datasets:
    z_score = (i - mean) / std
    if z_score > threshold:
        outliers.append(i)
print("Outliers in dataset are:", outliers)
```

OUTPUT:



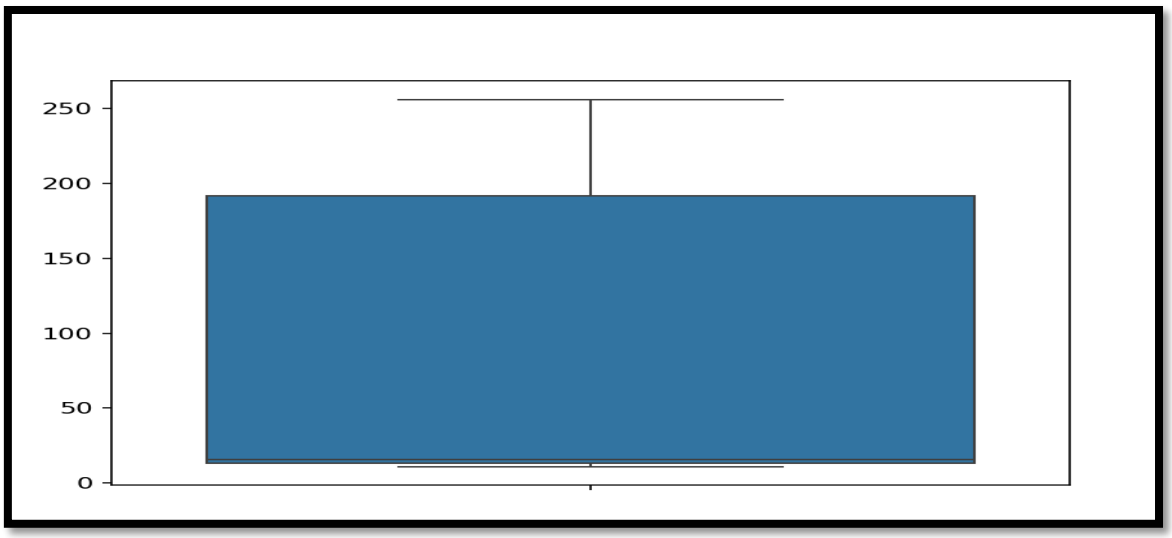

```
==========
Outliers in dataset are: [256, 255, 250]
>>>
```

(ii) Using IQR

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
#detecting outlier using IQR
#SORT THE DATA
#Calculate q1(25%) and q3(75%)
#Iqr(q3-q1)
#Lower fence(q1-1.5(iqr))
#Upper fence(q3+1.5(iqr))
datasets = [11, 17, 16, 15, 14, 13, 12, 256, 255, 250]
dataset=sorted(datasets)
print(dataset)
q1,q3=np.percentile(dataset,[25,75])
print(q1,q3)
iqr=q3-q1
print(iqr)
lower_fence=q1-(1.5*iqr)
upper_fence=q3+(1.5*iqr)
print(lower_fence,upper_fence)
import seaborn as sns
sns.boxplot(datasets)
plt.show()
```

OUTPUT:




```
[11, 12, 13, 14, 15, 16, 17, 250, 255, 256]
13.25 191.75
178.5
-254.5 459.5
>>>
```

PRACTICAL NO 3

AIM: FEATURE SCALING AND DUMMIFICATION

a) Apply feature-scaling techniques like standardization and normalization to numerical features.

CODE:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
data = {
   'Feature1': [10, 20, 30, 40, 50],
   'Feature2': [0.1, 0.5, 1.5, 2.0, 2.5],
   'Feature3': [5, 10, 15, 20, 25]
}
df = pd.DataFrame(data)
X = df[['Feature1', 'Feature2', 'Feature3']]
y = df["Feature3"]  # Replace 'target_column' with the name of your target column
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler_standard = StandardScaler()
X_train_standardized = scaler_standard.fit_transform(X_train)
X_test_standardized = scaler_standard.transform(X_test)
scaler_minmax = MinMaxScaler()
X_train_normalized = scaler_minmax.fit_transform(X_train)
X_test_normalized = scaler_minmax.transform(X_test)
print("Original features:")
print(X)
print("\nScaled features (standardized):")
print(X_train_standardized)
print("\nScaled features (normalized):")
print(X_train_normalized)
```

OUTPUT:

```
================================================
Original features:
   Feature1  Feature2  Feature3
0        10       0.1         5
1        20       0.5        10
2        30       1.5        15
3        40       2.0        20
4        50       2.5        25

Scaled features (standardized):
[[ 1.18321596  1.08880794  1.18321596]
 [-0.16903085 -0.02791815 -0.16903085]
 [-1.52127766 -1.59133468 -1.52127766]
 [ 0.50709255  0.53044489  0.50709255]]

Scaled features (normalized):
[[1.         1.         1.        ]
 [0.5        0.58333333 0.5       ]
 [0.         0.         0.        ]
 [0.75       0.79166667 0.75      ]]
>>>
```

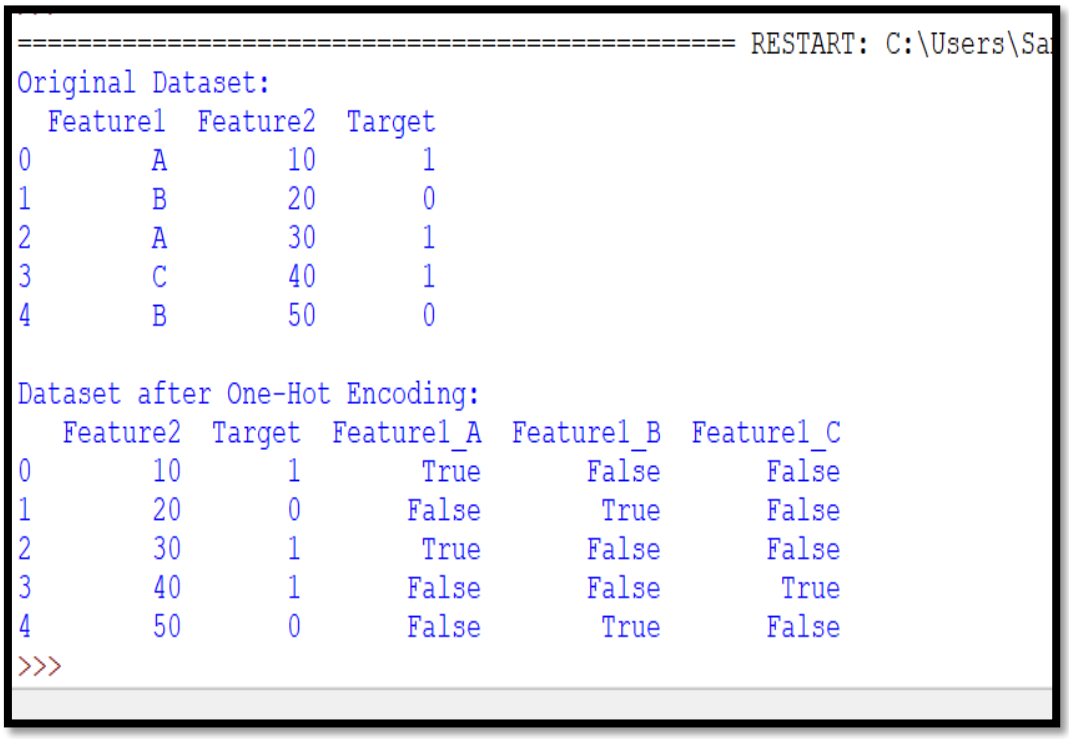
b) Perform feature dummification to convert categorical variables into numerical representations.

CODE:

```
import pandas as pd
data = {
   'Feature1': ['A', 'B', 'A', 'C', 'B'],
   'Feature2': [10, 20, 30, 40, 50],
   'Target': [1, 0, 1, 1, 0]
}

df = pd.DataFrame(data)
print("Original Dataset:")
print(df)
df_encoded = pd.get_dummies(df, columns=['Feature1'], prefix='Feature1')
print("\nDataset after One-Hot Encoding:")
print(df_encoded)
```

OUTPUT:



```
========== RESTART: C:\Users\Sa
Original Dataset:
  Feature1  Feature2  Target
0        A        10       1
1        B        20       0
2        A        30       1
3        C        40       1
4        B        50       0

Dataset after One-Hot Encoding:
   Feature2  Target  Feature1_A  Feature1_B  Feature1_C
0        10       1        True       False       False
1        20       0       False        True       False
2        30       1        True       False       False
3        40       1       False       False        True
4        50       0       False        True       False
>>>
```

PRACTICAL NO 4

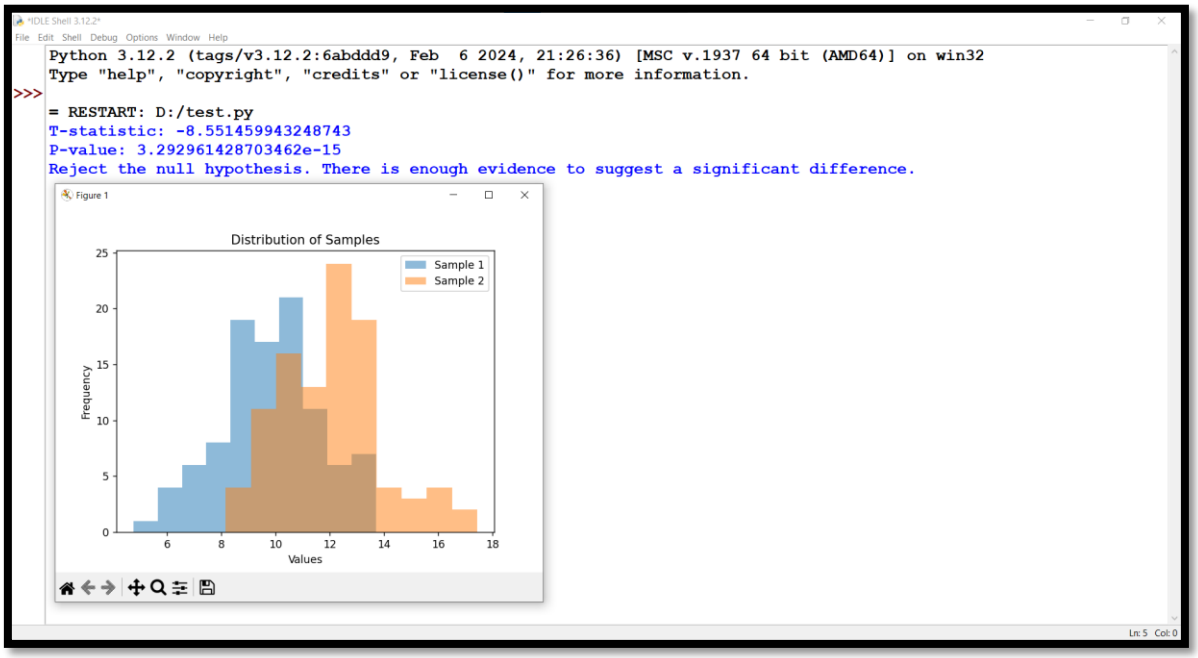
AIM: HYPOTHESIS TESTING

a) Formulate null and alternative hypothesis for a given problem.
b) Conduct a hypothesis test using appropriate statistical tests(t-test)
c) Interpret the results and draw conclusions based on the test outcomes

CODE:

```
import numpy as np
from scipy.stats import ttest_ind, t
import matplotlib.pyplot as plt
np.random.seed(42)
sample1 = np.random.normal(loc=10, scale=2, size=100)
sample2 = np.random.normal(loc=12, scale=2, size=100)
t_statistic, p_value = ttest_ind(sample1, sample2)
# Step 4: Interpret the results
alpha = 0.05 # significance level
print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")
if p_value < alpha:
print("Reject the null hypothesis. There is enough evidence to suggest a significant
difference.")
else:
print("Fail to reject the null hypothesis. There is not enough evidence to suggest a
significant difference.")
plt.hist(sample1, alpha=0.5, label='Sample 1')
plt.hist(sample2, alpha=0.5, label='Sample 2')
plt.legend()
plt.title('Distribution of Samples')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()
x = np.linspace(-5, 5, 1000)
y = t.pdf(x, df=len(sample1) + len(sample2) – 2)
plt.plot(x, y, label='t-distribution')
plt.axvline(t_statistic, color='red', linestyle='dashed', label='T-statistic')
plt.legend()
plt.title('T-distribution with T-statistic')
plt.xlabel('T-statistic')
plt.ylabel('Probability Density')
plt.show()
```

OUTPUT:


Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:/test.py
T-statistic: -8.551459943248743
P-value: 3.292961428703462e-15
Reject the null hypothesis. There is enough evidence to suggest a significant difference.
Figure 1
Distribution of Samples
Sample 1
Sample 2
Frequency
Values

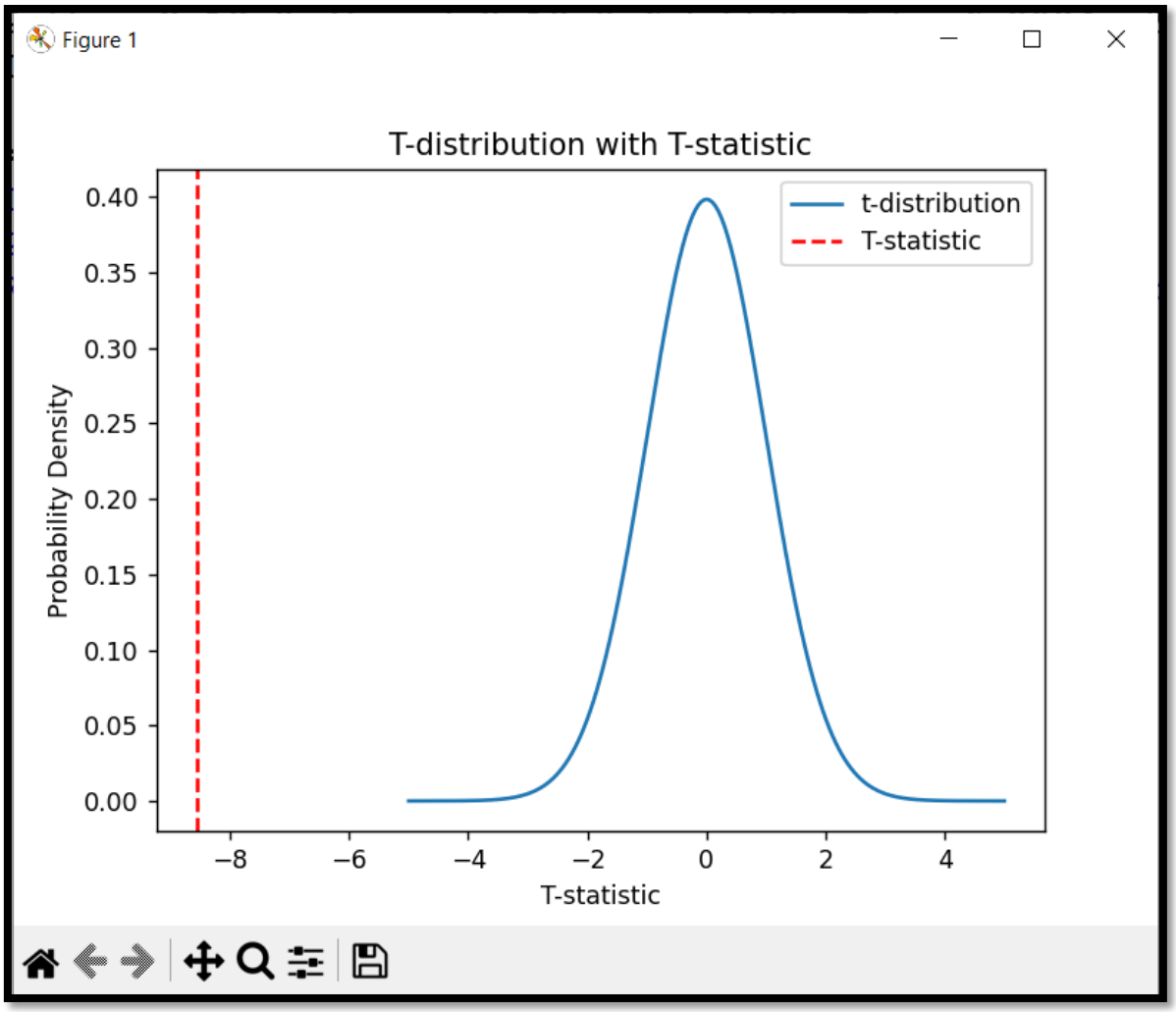


Figure 1
T-distribution with T-statistic
t-distribution
T-statistic
Probability Density
T-statistic

PRACTICAL NO 5

AIM: ANOVA (ANALYSIS OF VARIANCE)

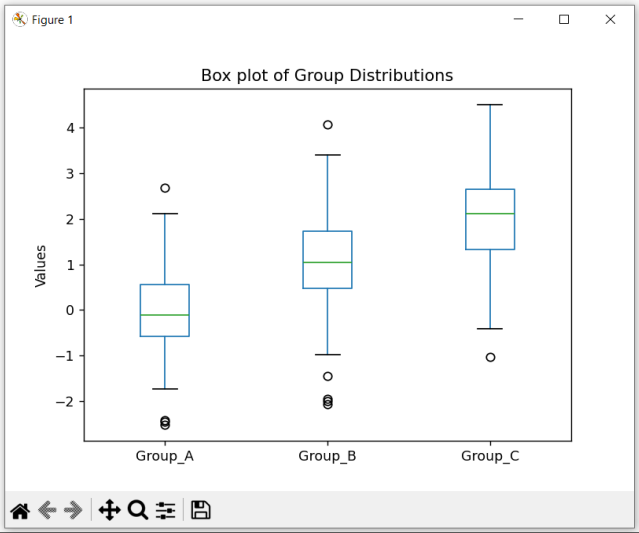
a) **Perform one-way ANOVA to compare across multiple groups.**
b) **Conduct post-hoc tests to identify significant differences between group means.**

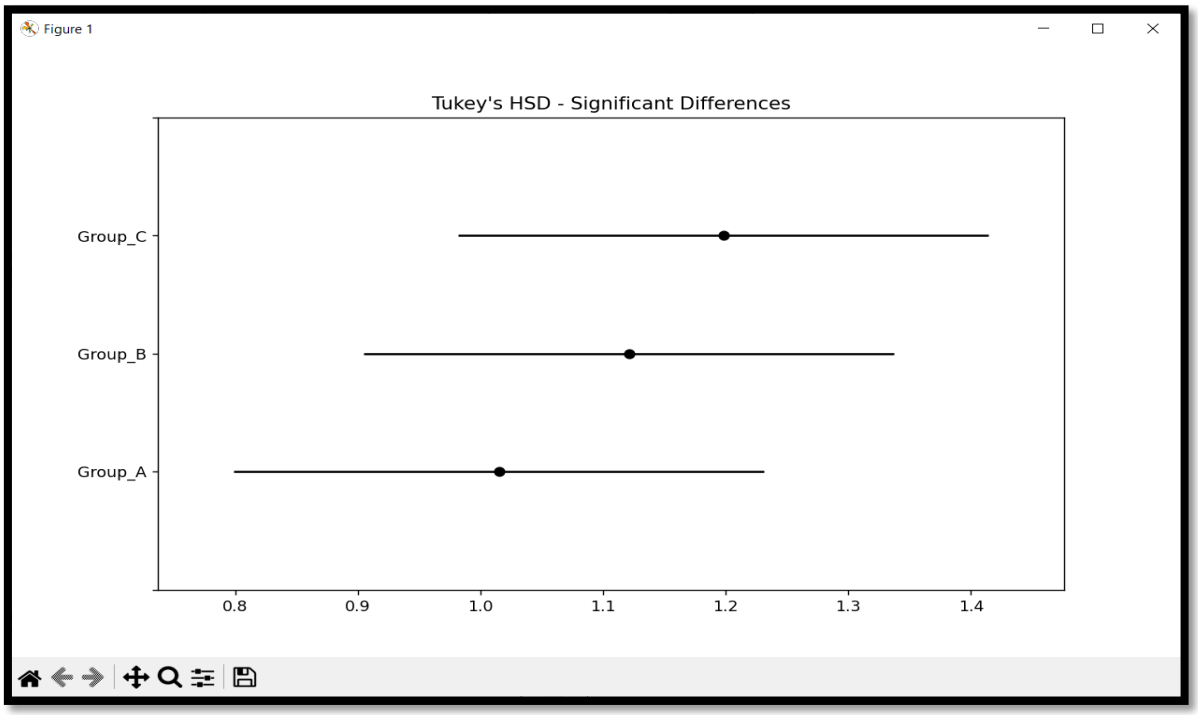
CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import f_oneway
from statsmodels.stats.multicomp import pairwise_tukeyhsd
data = pd.DataFrame({
   'Group_A': np.random.normal(0, 1, 100),
   'Group_B': np.random.normal(1, 1, 100),
   'Group_C': np.random.normal(2, 1, 100),
})
f_statistic, p_value = f_oneway(data['Group_A'], data['Group_B'],
data['Group_C'])
print(f'One-way ANOVA results: F-statistic = {f_statistic}, p-value = {p_value}')
data.boxplot(grid=False)
plt.title('Box plot of Group Distributions')
plt.ylabel('Values')
plt.show()
posthoc = pairwise_tukeyhsd(data.values.flatten(), np.repeat(data.columns,
len(data)))
print(posthoc)
posthoc.plot_simultaneous()
plt.title('Tukey\'s HSD - Significant Differences')
plt.show()
```

OUTPUT:

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] or
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:/test.py
One-way ANOVA results: F-statistic = 101.86495852273856, p-value = 2.056870961499119e-34
```





Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
Group_A	Group_B	0.019	0.9941	-0.4124	0.4504	False
Group_A	Group_C	0.0796	0.9011	-0.3518	0.511	False
Group_B	Group_C	0.0607	0.9413	-0.3708	0.4921	False

PRACTICAL NO 6

AIM: REGRESSION AND IT'S TYPES

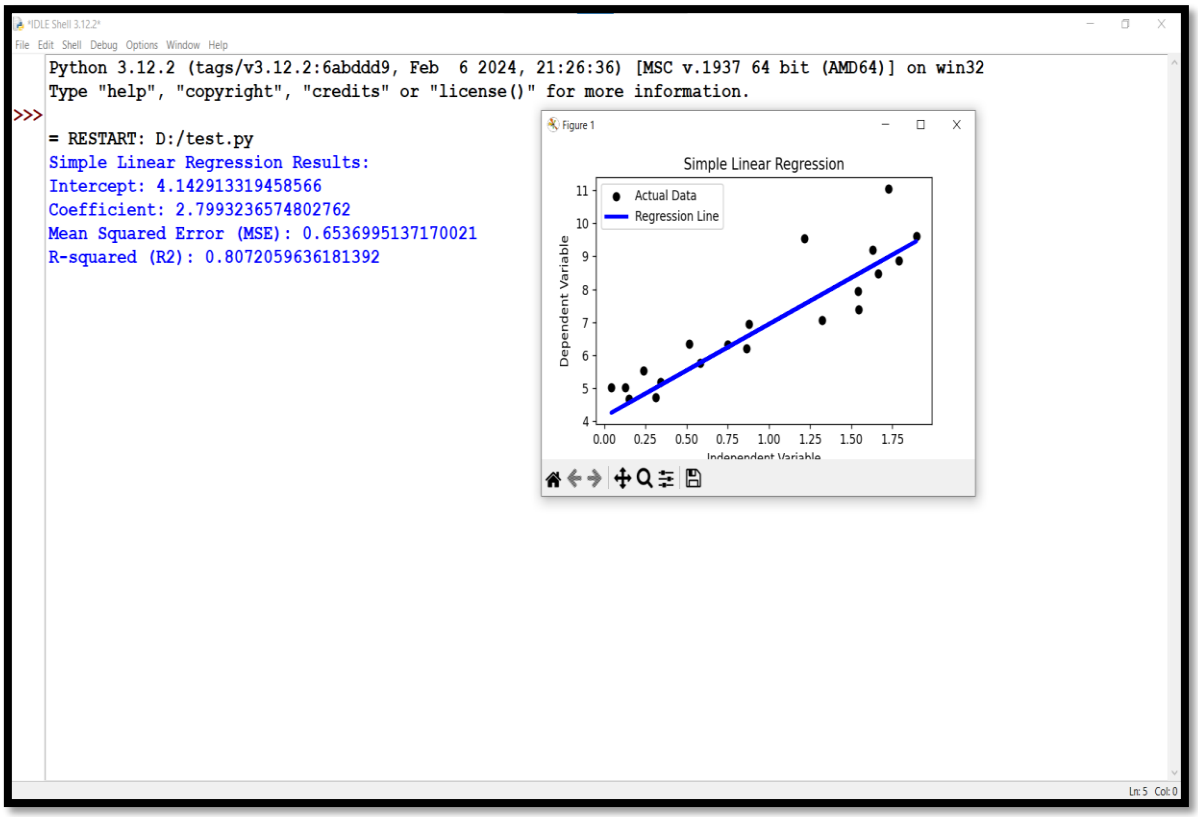
a) **Implement simple linear regression using a dataset.**
b) **Explore and interpret the regression model coefficients and goodness-of-fit measures.**
c) **Extend the analysis to multiple linear regression and assess the impact of additional predictors.**

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
np.random.seed(42)
X_simple = 2 * np.random.rand(100, 1)
y_simple = 4 + 3 * X_simple[:, 0] + np.random.randn(100)
X_simple_train, X_simple_test, y_simple_train, y_simple_test = train_test_split(
    X_simple, y_simple, test_size=0.2, random_state=42
)
simple_model = LinearRegression()
simple_model.fit(X_simple_train, y_simple_train)
y_simple_pred = simple_model.predict(X_simple_test)
mse_simple = mean_squared_error(y_simple_test, y_simple_pred)
r2_simple = r2_score(y_simple_test, y_simple_pred)
print("Simple Linear Regression Results:")
print(f'Intercept: {simple_model.intercept_}')
print(f'Coefficient: {simple_model.coef_[0]}')
print(f'Mean Squared Error (MSE): {mse_simple}')
print(f'R-squared (R2): {r2_simple}')
plt.figure(figsize=(10, 5))
plt.scatter(X_simple_test, y_simple_test, color='black', label='Actual Data')
plt.plot(X_simple_test, y_simple_pred, color='blue', linewidth=3,
label='Regression Line')
plt.xlabel('Independent Variable')
plt.ylabel('Dependent Variable')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
X_multi = 2 * np.random.rand(100, 2)
y_multi = 4 + 3 * X_multi[:, 0] + 2 * X_multi[:, 1] + np.random.randn(100)
X_multi_train, X_multi_test, y_multi_train, y_multi_test = train_test_split(
    X_multi, y_multi, test_size=0.2, random_state=42
)
multi_model = LinearRegression()
multi_model.fit(X_multi_train, y_multi_train)
y_multi_pred = multi_model.predict(X_multi_test)
mse_multi = mean_squared_error(y_multi_test, y_multi_pred)
```

```
r2_multi = r2_score(y_multi_test, y_multi_pred)
print("\nMultiple Linear Regression Results:")
print(f'Intercept: {multi_model.intercept_}')
print(f'Coefficients: {multi_model.coef_}')
print(f'Mean Squared Error (MSE): {mse_multi}')
print(f'R-squared (R2): {r2_multi}')
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(X_multi_test[:, 0], X_multi_test[:, 1], y_multi_test, c='black',
marker='o')
ax1.set_xlabel('Independent Variable 1')
ax1.set_ylabel('Independent Variable 2')
ax1.set_zlabel('Dependent Variable (y)')
ax1.set_title('Actual Data')
ax2 = fig.add_subplot(122, projection='3d')
ax2.scatter(X_multi_test[:, 0], X_multi_test[:, 1], y_multi_pred, c='blue',
marker='o')
ax2.set_xlabel('Independent Variable 1')
ax2.set_ylabel('Independent Variable 2')
ax2.set_zlabel('Predicted Variable (y)')
ax2.set_title('Multiple Linear Regression Prediction')
plt.show()
```

OUTPUT:


IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:/test.py
Simple Linear Regression Results:
Intercept: 4.142913319458566
Coefficient: 2.7993236574802762
Mean Squared Error (MSE): 0.6536995137170021
R-squared (R2): 0.8072059636181392
Ln: 5 Col: 0
Figure 1
Simple Linear Regression
Actual Data
Regression Line
Dependent Variable
Independent Variable

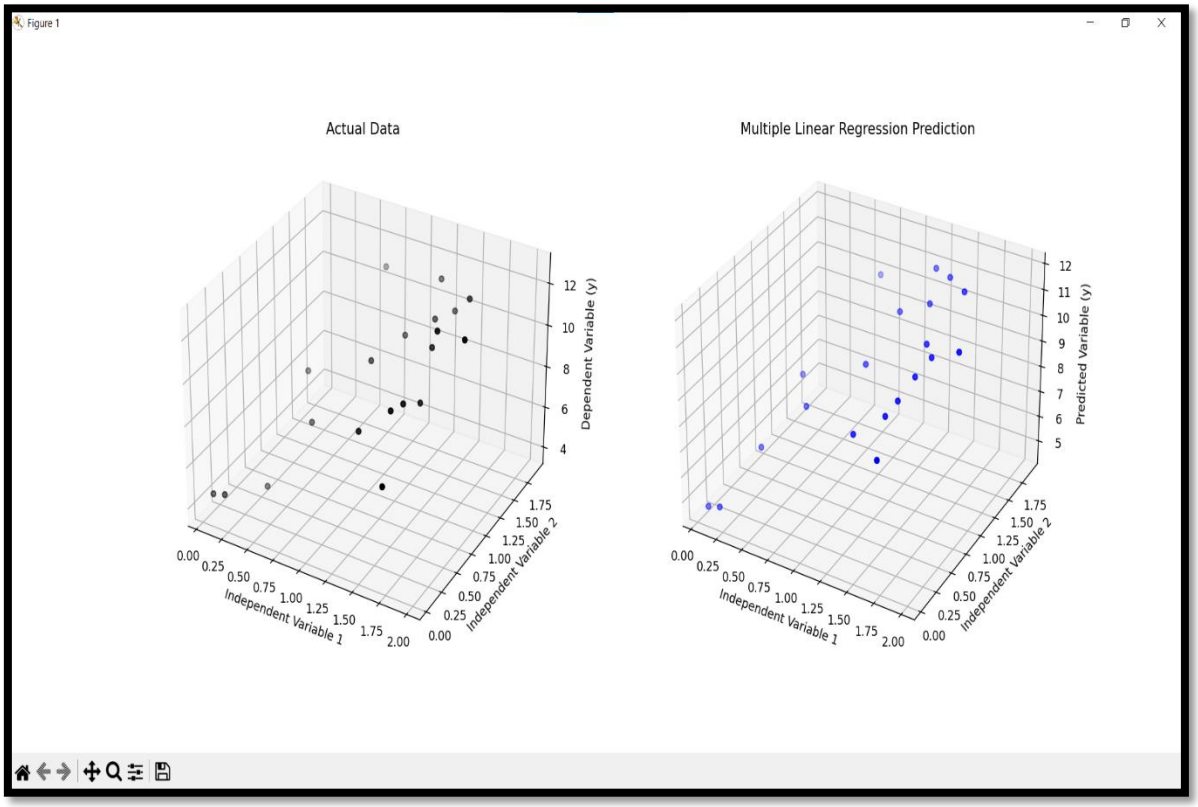


Figure 1
Actual Data
Multiple Linear Regression Prediction
Dependent Variable (y)
Predicted Variable (y)
Independent Variable 1
Independent Variable 2

PRACTICAL NO 7

AIM: LOGISTIC REGRESSION AND DECISION TREE

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

CODE:

```
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,
classification_report
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = (data.target == 2).astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
logreg_model = LogisticRegression()
logreg_model.fit(X_train, y_train)
logreg_predictions = logreg_model.predict(X_test)
print("Logistic Regression Model Performance:")
print("Accuracy:", accuracy_score(y_test, logreg_predictions))
print("Precision:", precision_score(y_test, logreg_predictions))
print("Recall:", recall_score(y_test, logreg_predictions))
print("\nClassification Report:")
print(classification_report(y_test, logreg_predictions))
tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, y_train)
tree_predictions = tree_model.predict(X_test)
print("\nDecision Tree Model Performance:")
print("Accuracy:", accuracy_score(y_test, tree_predictions))
print("Precision:", precision_score(y_test, tree_predictions))
print("Recall:", recall_score(y_test, tree_predictions))
print("\nClassification Report:")
print(classification_report(y_test, tree_predictions))
```

OUTPUT:

```
Logistic Regression Model Performance:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
Decision Tree Model Performance:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

PRACTICAL NO 8

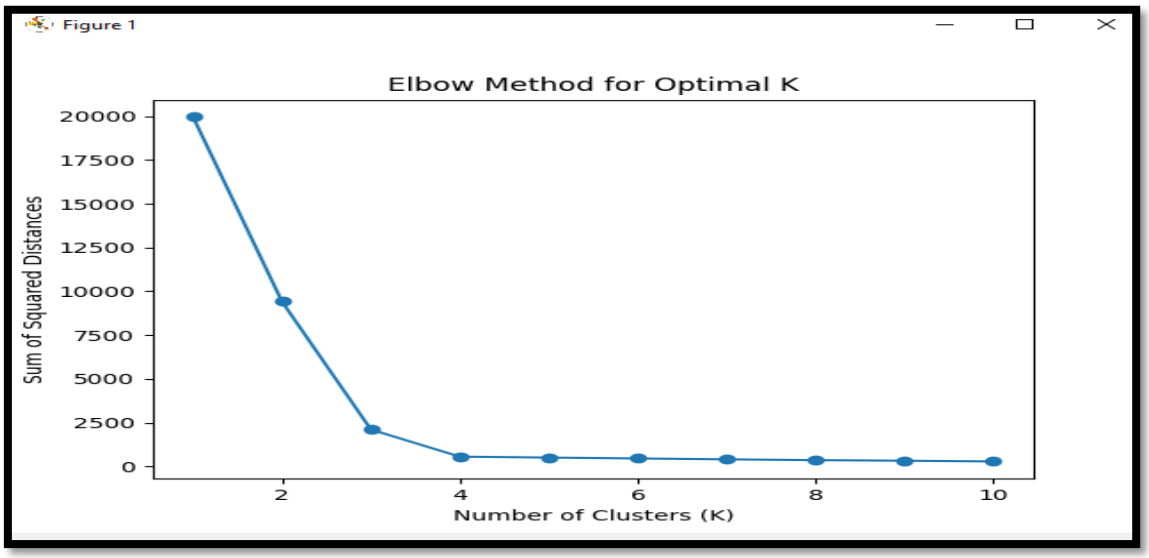
AIM: K-MEANS CLUSTERING

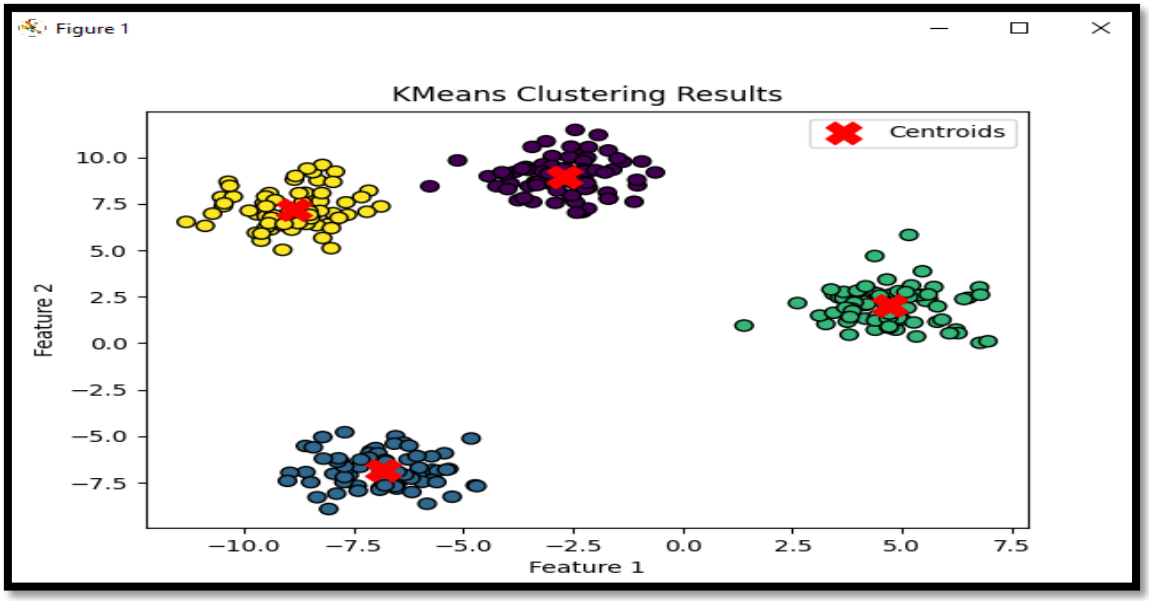
a) **Apply the K-Means algorithm to group similar data points into clusters.**
b) **Determine the optimal number of clusters using elbow method.**
c) **Visualize the clustering results and analyze the cluster characteristics**.

CODE:

```
import os
os.environ["LOKY_MAX_CPU_COUNT"] = "4"
# Adjust the number based on your system configuration
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
data, _ = make_blobs(n_samples=300, centers=4, random_state=42)
def calculate_inertia(data, k_range):
    inertias = []
    for k in k_range:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        inertias.append(kmeans.inertia_)
    return inertias
k_range = range(1, 11)
inertias = calculate_inertia(data, k_range)
plt.plot(k_range, inertias, marker='o')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Sum of Squared Distances')
plt.show()
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
labels = kmeans.fit_predict(data)
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', edgecolors='k', s=50)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='red',
marker='X', s=200, label='Centroids')
plt.title('KMeans Clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
# Analyze cluster characteristics
for i in range(optimal_k):
    cluster_points = data[labels == i]
    print(f"Cluster {i + 1} - Size: {len(cluster_points)}")
    print(f"Centroid: {kmeans.cluster_centers_[i]}")
    print(f"Sample points: {cluster_points[:5]}\n")
```

OUTPUT:






```
Cluster 1 - Size: 76
Centroid: [-2.70981136  8.97143336]
Sample points: [[-1.68665271  7.79344248]
 [-2.98837186  8.82862715]
 [-3.11090424 10.86656431]
 [-2.30033403  7.054616  ]
 [-3.92456837  8.59364081]]

Cluster 2 - Size: 75
Centroid: [-6.83235205 -6.83045748]
Sample points: [[-7.09730839 -5.78133274]
 [-6.02196757 -7.04004812]
 [-6.46669574 -7.44383415]
 [-5.90450746 -7.02716697]
 [-7.69784787 -4.78772232]]
```

```
Cluster 3 - Size: 75
Centroid: [4.7182049  2.04179676]
Sample points: [[4.81305976 2.35848706]
 [3.44857534 2.62972329]
 [4.99689432 1.28026009]
 [2.61473625 2.159624  ]
 [5.79847442 1.15248737]]

Cluster 4 - Size: 74
Centroid: [-8.87357218  7.17458342]
Sample points: [[ -9.29768866   6.47367855]
 [ -9.69874112   6.93896737]
 [-10.87645229   6.3154366 ]
 [ -9.48897033   6.83639753]
 [-10.46587019   7.37160786]]
```

PRACTICAL NO 9

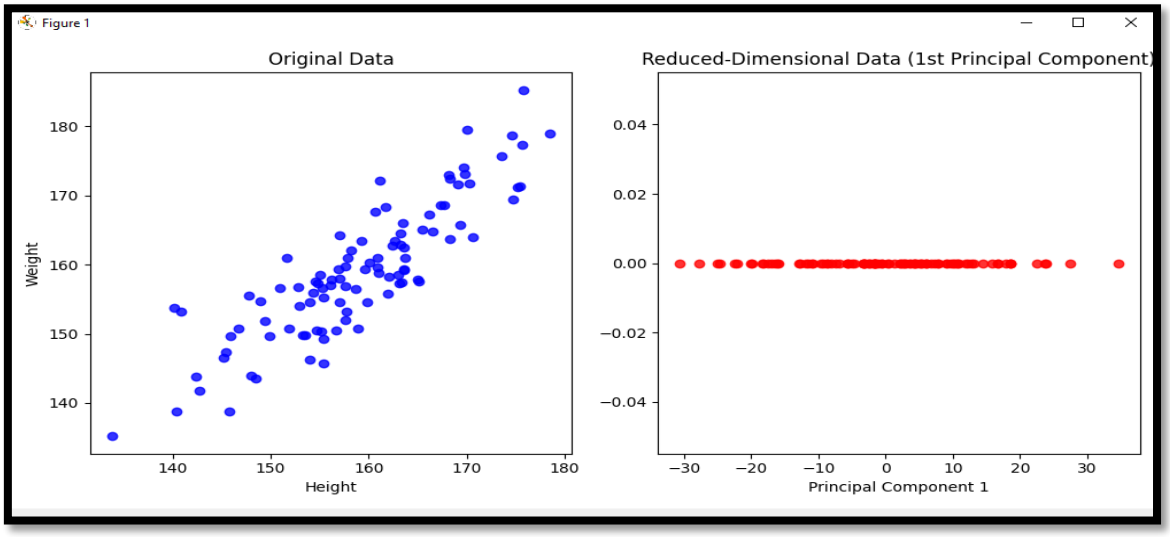
AIM: PRINCIPAL COMPONENT ANALYSIS (PCA)

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

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
np.random.seed(42)
height = np.random.normal(160, 10, 100)  # mean=160, std=10
weight = height + np.random.normal(0, 5, 100)
data = np.column_stack((height, weight))
pca = PCA()
data_pca = pca.fit_transform(data)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(data[:, 0], data[:, 1], c='blue', alpha=0.8)
plt.title('Original Data')
plt.xlabel('Height')
plt.ylabel('Weight')
plt.subplot(1, 2, 2)
plt.scatter(data_pca[:, 0], np.zeros_like(data_pca[:, 0]), c='red', alpha=0.8)
plt.title('Reduced-Dimensional Data (1st Principal Component)')
plt.xlabel('Principal Component 1')
plt.tight_layout()
plt.show()
```

OUTPUT:



PRACTICAL NO 10

AIM: DATA VISUALIZATION AND STORY TELLING

a) **Create meaningful visualization using data visualization tools**
b) **Combine multiple visualizations to tell a compelling data story**
c) **Present the findings and insights in a clear and concise manner**

STEPS:

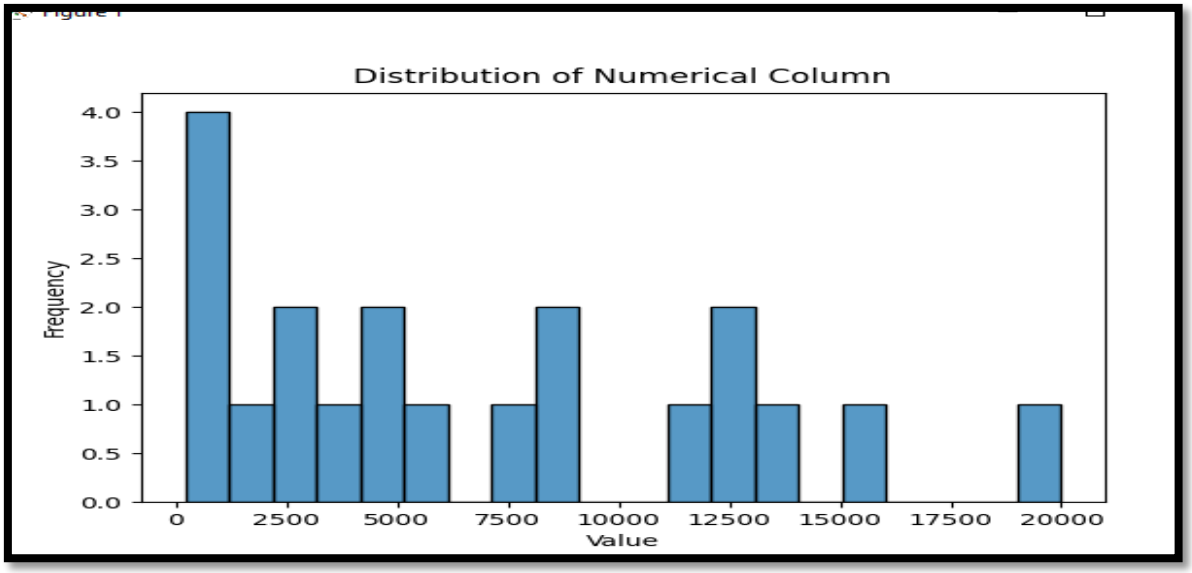
Add a csv file with numerical column (eg. amount) and category column

agent	acctype	openedby	branch	category	amount
25	savings	new accts	central	existing	5000
30	checking	teller	central	new	1471
25	cd	new accts	central	existing	3000
45	checking	new accts	central	new	2500
67	savings	new accts	central	existing	6000
78	fd	new accts	central	existing	7800
90	checking	new accts	north cou	existing	9000
34	savings	new accts	north cou	existing	1000
56	fd	new accts	north cou	new	3171
78	cd	new accts	north cou	new	12438
90	savings	new accts	north cou	new	11997
23	cd	new accts	westsider	new	4356
34	fd	new accts	westside	new	16000
45	checking	new accts	westside	new	13636
67	fd	new accts	central	existing	8721
89	savings	teller	central	existing	500
89	checking	teller	central	existing	499

CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv('C:\\Users\\pc7\\Downloads\\Book1.csv')
sns.histplot(data['amount'], bins=20)
plt.title('Distribution of Numerical Column')
plt.xlabel('Value')
plt.ylabel('Frequency')plt.show()
plt.figure(figsize=(8, 6))
sns.scatterplot(x='agent', y='amount', data=data, hue='branch')
plt.title('Scatter Plot of Feature1 vs Feature2')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend(title='target') plt.show()
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.barplot(x='category', y='value1', data=data) #specify any numerical column
plt.title('Bar Plot 1')
plt.subplot(1, 2, 2)
sns.barplot(x='category', y='value2', data=data) #specify any numerical column
plt.title('Bar Plot 2')
plt.tight_layout()
plt.show()
```

OUTPUT:


Distribution of Numerical Column
Frequency
Value
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
2500
5000
7500
10000
12500
15000
17500
20000

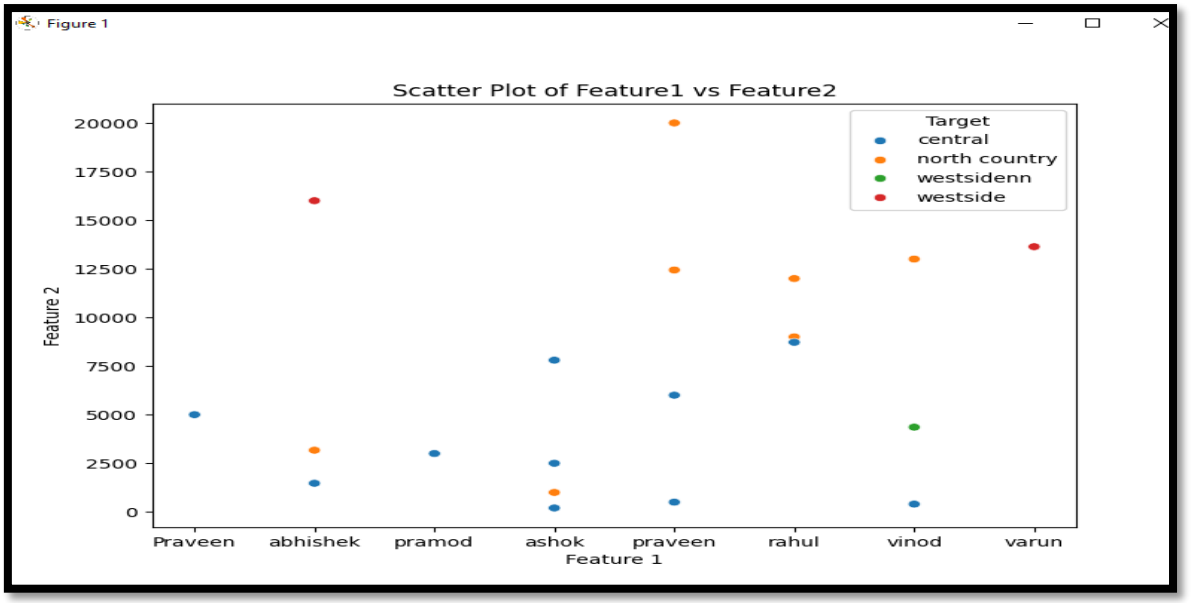


Figure 1
Scatter Plot of Feature1 vs Feature2
Target
central
north country
westsidenn
westside
Feature 2
Feature 1
0
2500
5000
7500
10000
12500
15000
17500
20000
Praveen
abhishek
pramod
ashok
praveen
rahul
vinod
varun

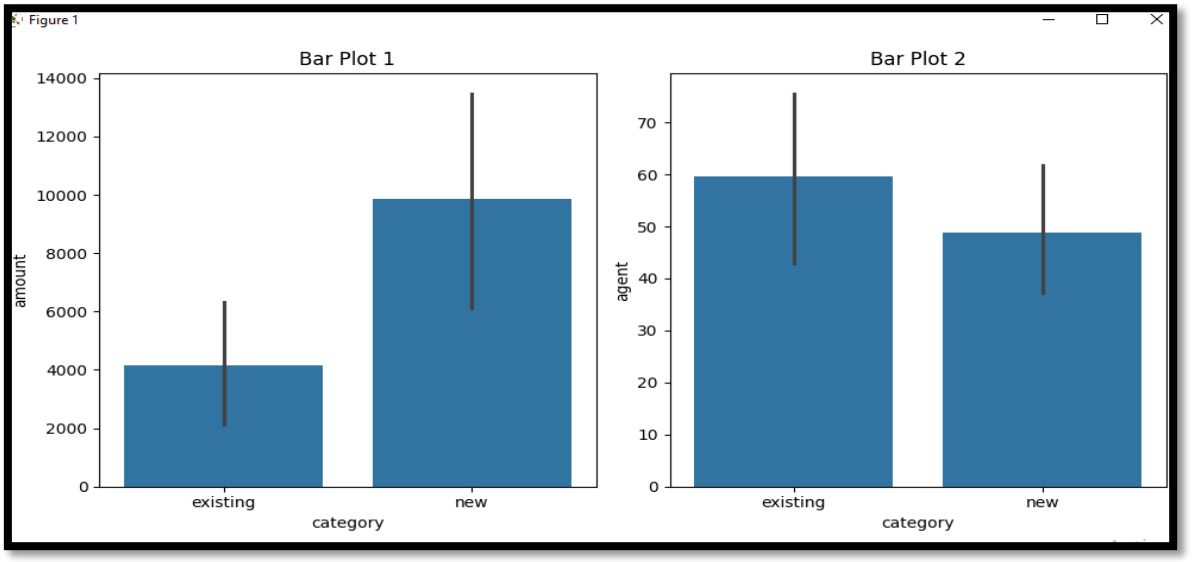


Figure 1
Bar Plot 1
amount
category
existing
new
0
2000
4000
6000
8000
10000
12000
14000
Bar Plot 2
agent
category
existing
new
0
10
20
30
40
50
60
70