 **NURTURING POTENTIAL**

**SAKET GYANPEETH’S**

**SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE KALYAN (EAST)**

**ACADEMIC YEAR 2024-25**

# B.Sc. Computer Science

## Semester VI

**SUBMITTED BY**

**Ayush Satya Prakash Singh**



**AS PRESCRIBED BY UNIVERSITY OF MUMBAI**





**NURTURING POTENTIAL**

**SAKET GYANPEETH’S**

## SAKET COLLEGE OF ARTS, SCIENCE AND

## COMMERCE

(Permanently Affiliated to University of Mumbai) NAAC Accredited B Grade

Saket Vidyanagri Marg, Chinchpada Road, Katemanivali, Kalyan (East) -421306(Mah)

## Department of Computer Science

This is to certify that

**Mr. Ayush satya Prakash singh Roll no. 243107 of**

# B.Sc. Computer Science

## Semester VI

has satisfactory carried out the required practical in the subject

of **Data Science**

For the Academic year 2024 – 2025.

**External Examiner**

**Head of the Department**

**Practical In-Charge**

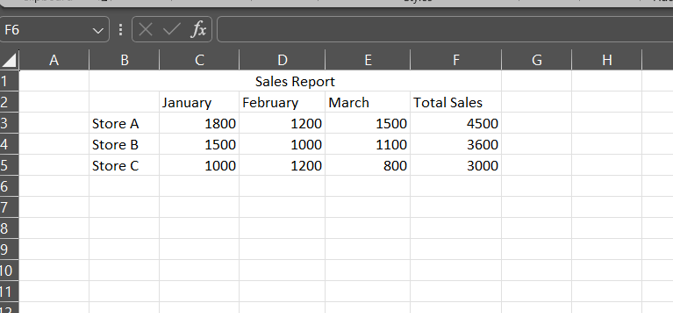
**College Seal**

# INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **SR.**  **NO.** | **TOPIC** | **DATE** | **SIGNATURE** |
| 1 | 1. Perform conditional formatting on a datasheet using various criteria 2. Creative pivot table to analyse and summarise data 3. Use VLOOKUP function to retrieve information from a different worksheet or a table 4. Perform what if analysis using goal sick to determine input values for   desired output |  |  |
| 2. | Data frames and basic data pre processing |  |  |
| 3. | Feature scaling and dummification |  |  |
| 4. | Hypothesis testing |  |  |
| 5. | ANOVA (Analysis of Variance) |  |  |
| 6. | Regression and its types |  |  |
| 7. | Logistic regression and decision tree |  |  |
| 8. | K means clustering |  |  |
| 9. | Principal component analysis PCA |  |  |
| 10. | Data visualisation and story telling |  |  |

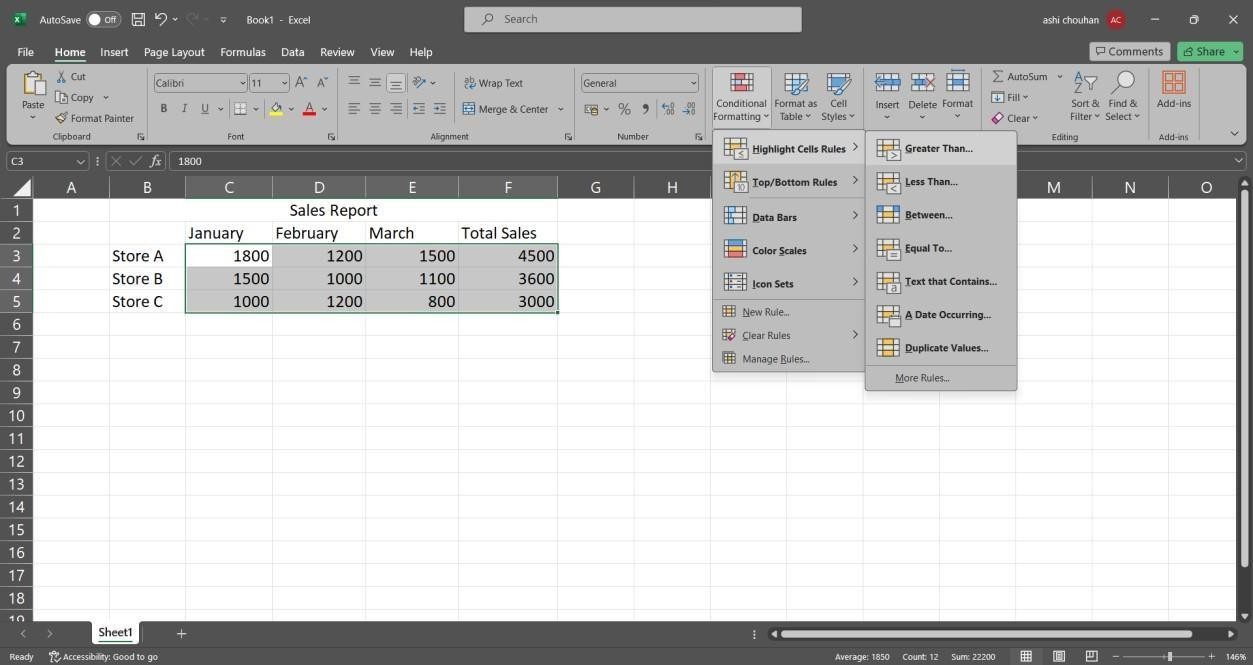
**PRACTICAL NO. – 1**

## Perform conditional formatting on a dataset using various criteria.

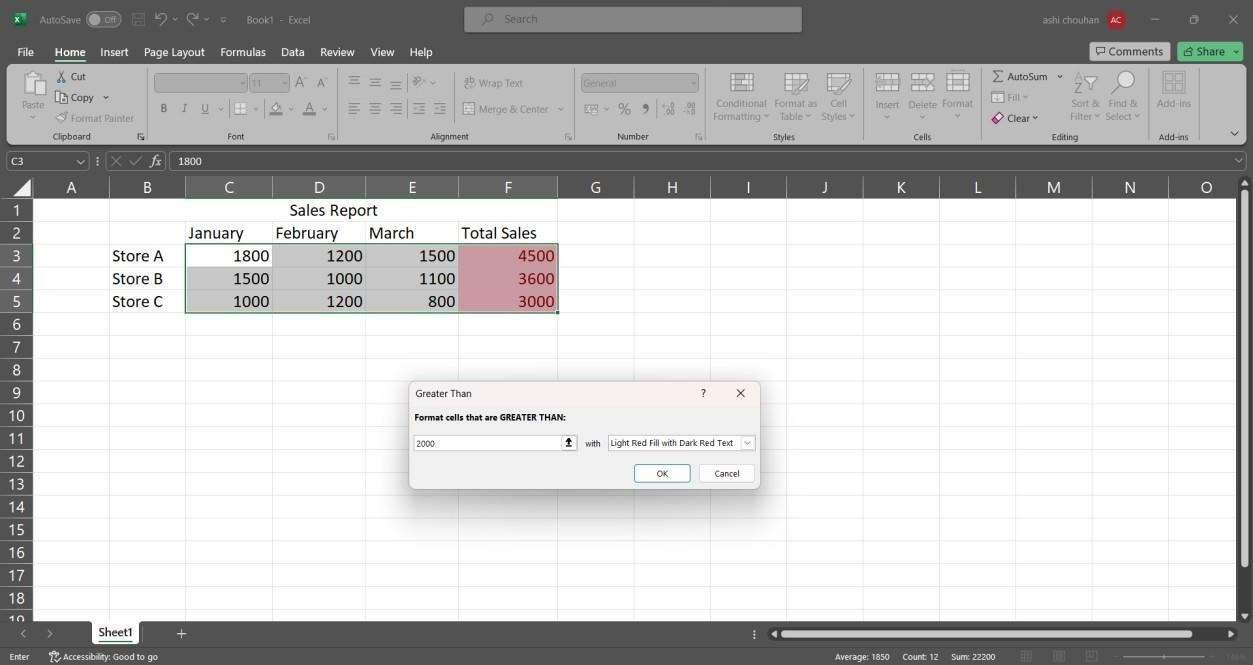
****

Steps

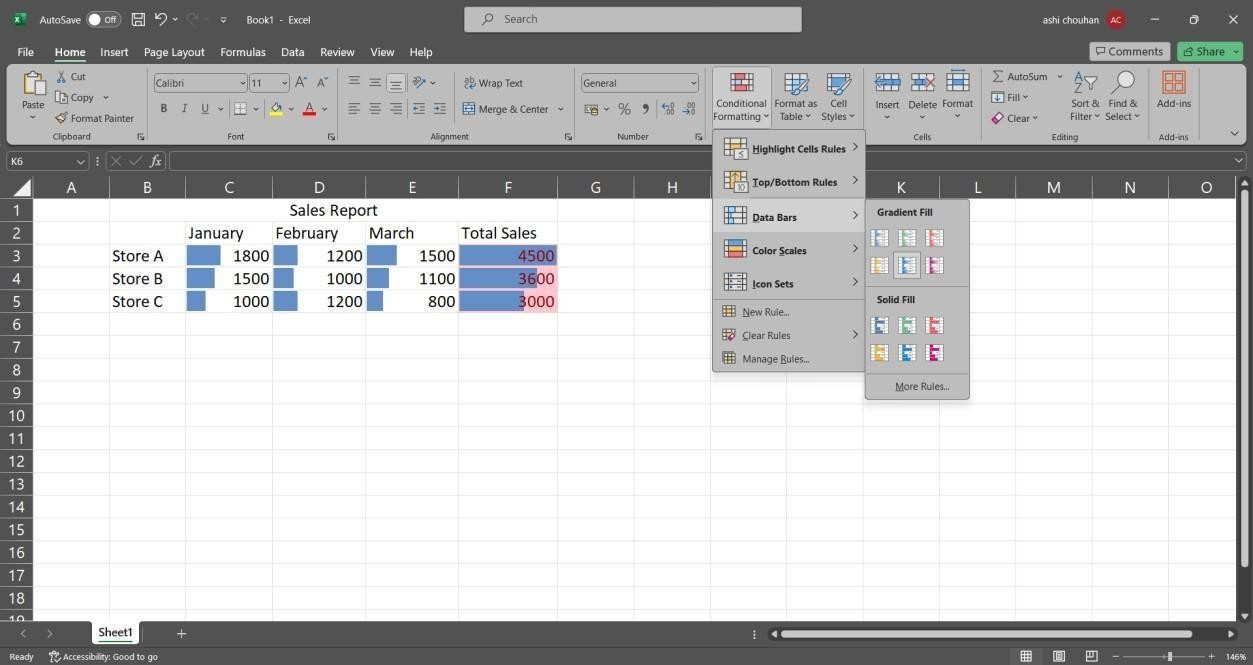
Step 1: Go to conditional formatting > Greater Than



Step 2: Enter the greater than filter value for example 2000.



Step 3: Go to Data Bars > Solid Fill in conditional formatting.

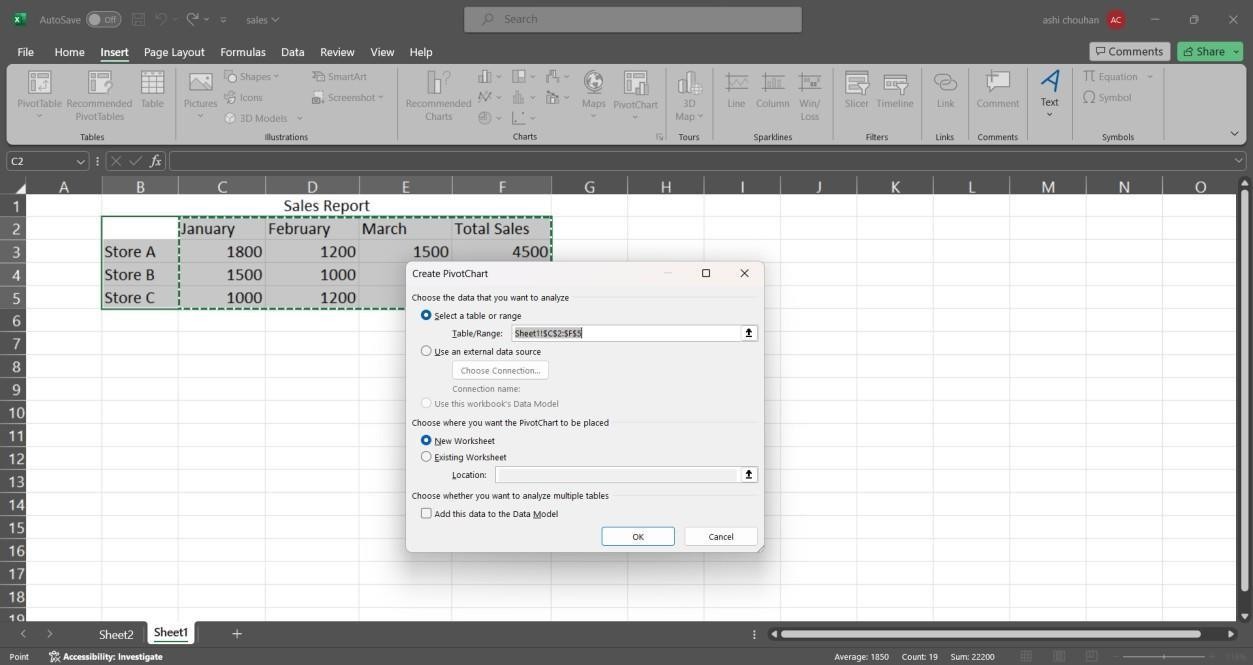


## Create a pivot table to analyse and summarize data.

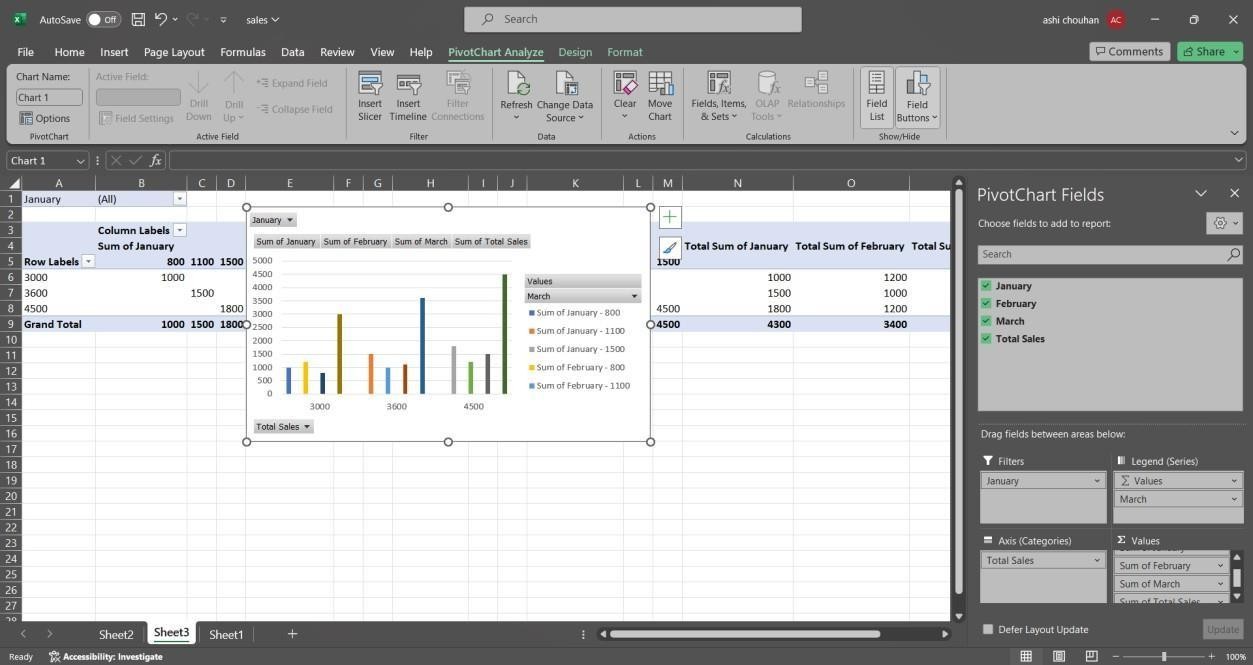
Steps

Step1: - Select the entire table and go to Insert Tab PivotChart > Pivotchart

Step2:- Select “New Worksheet” in the create pivot chart window.



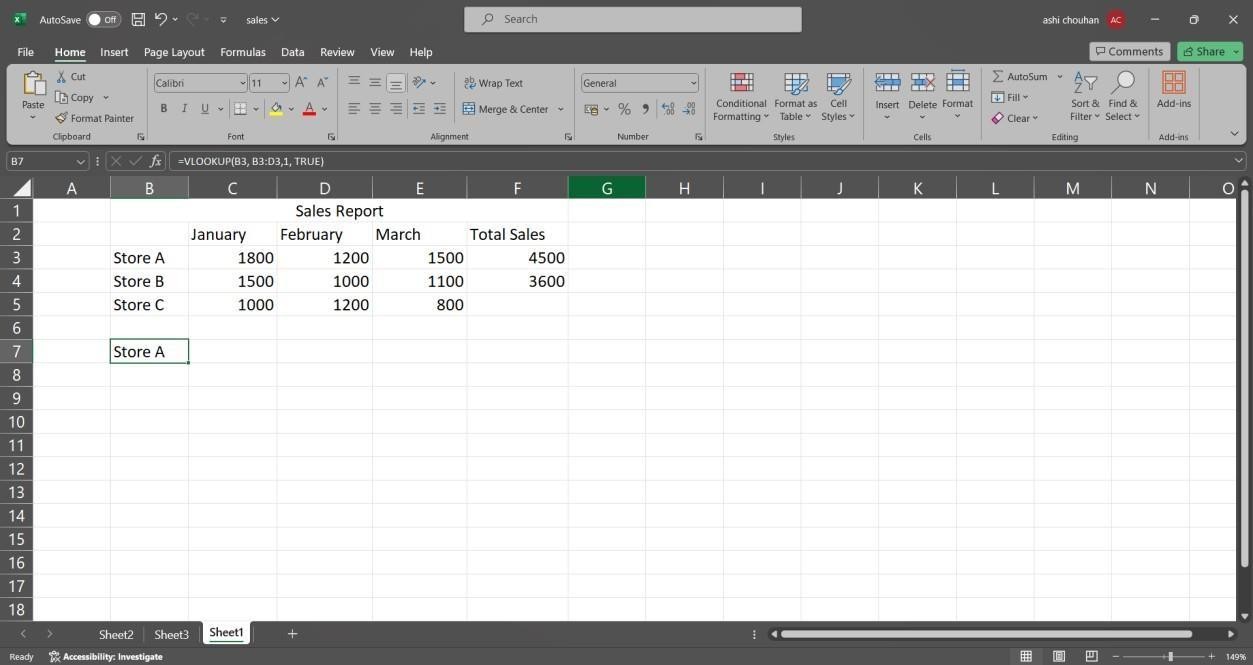
Step 3: Select and drag attributes in the below boxes.



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

Steps

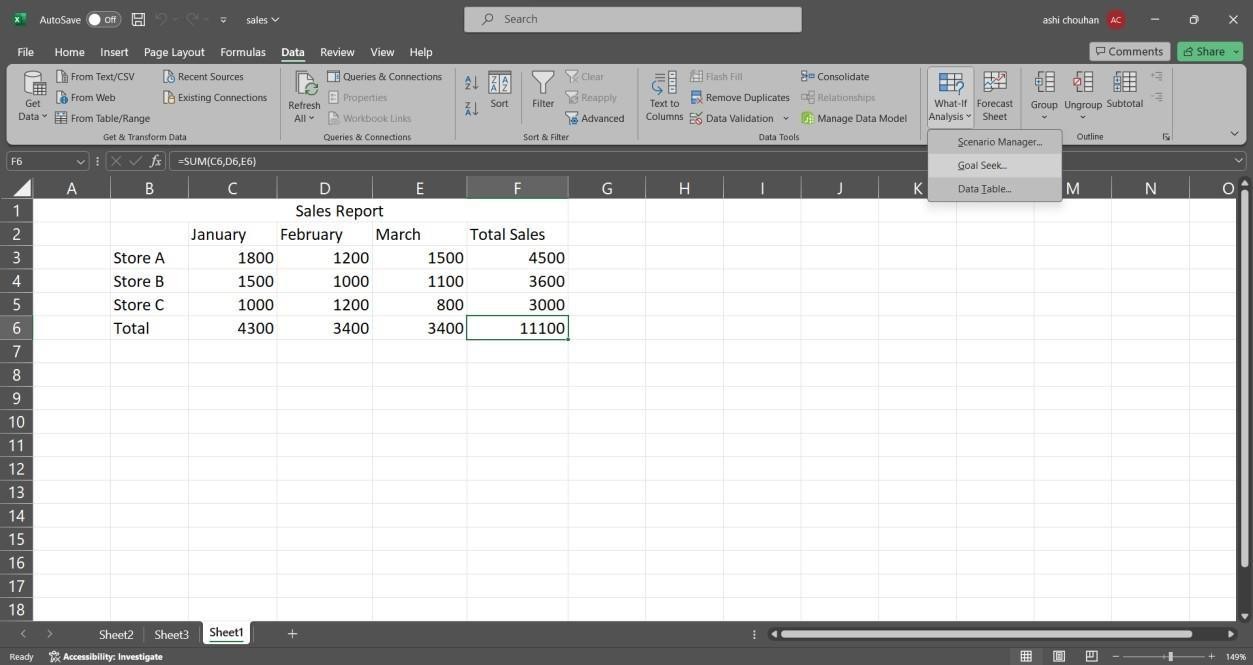
Step 1: click on an empty cell and type the following command. =VLOOKUP(B3, B3:D3,1, TRUE)



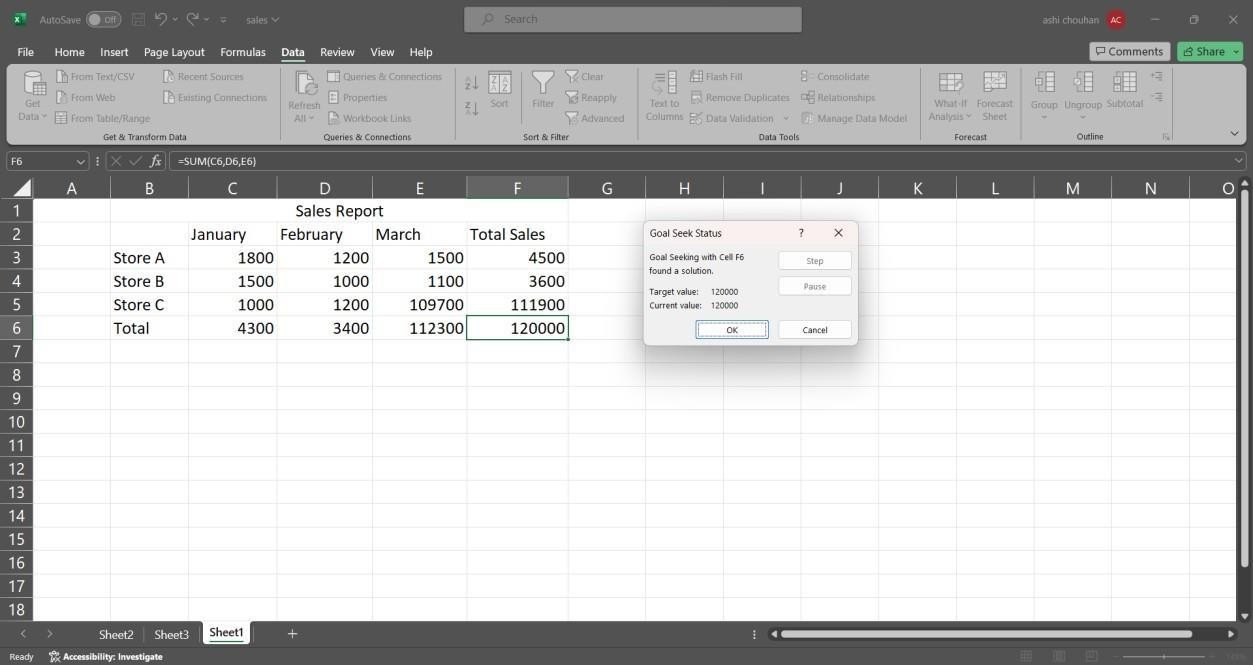
## Perform what-if analysis using Goal Seek to determine input values for desired output.

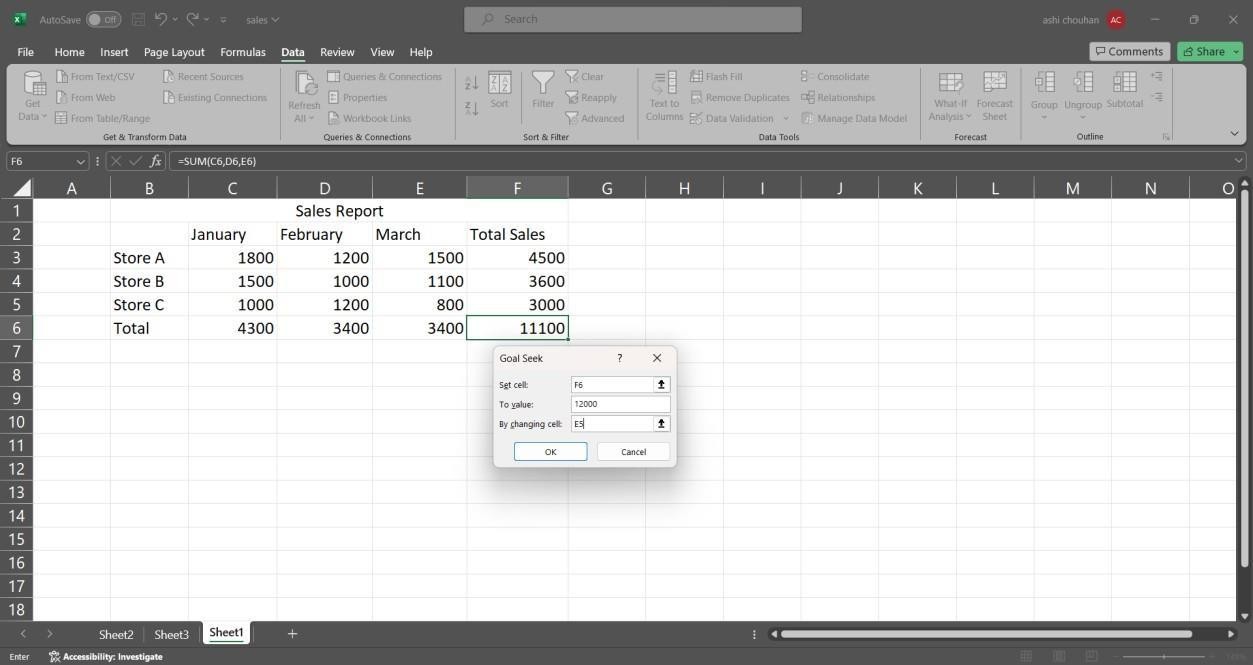
Steps

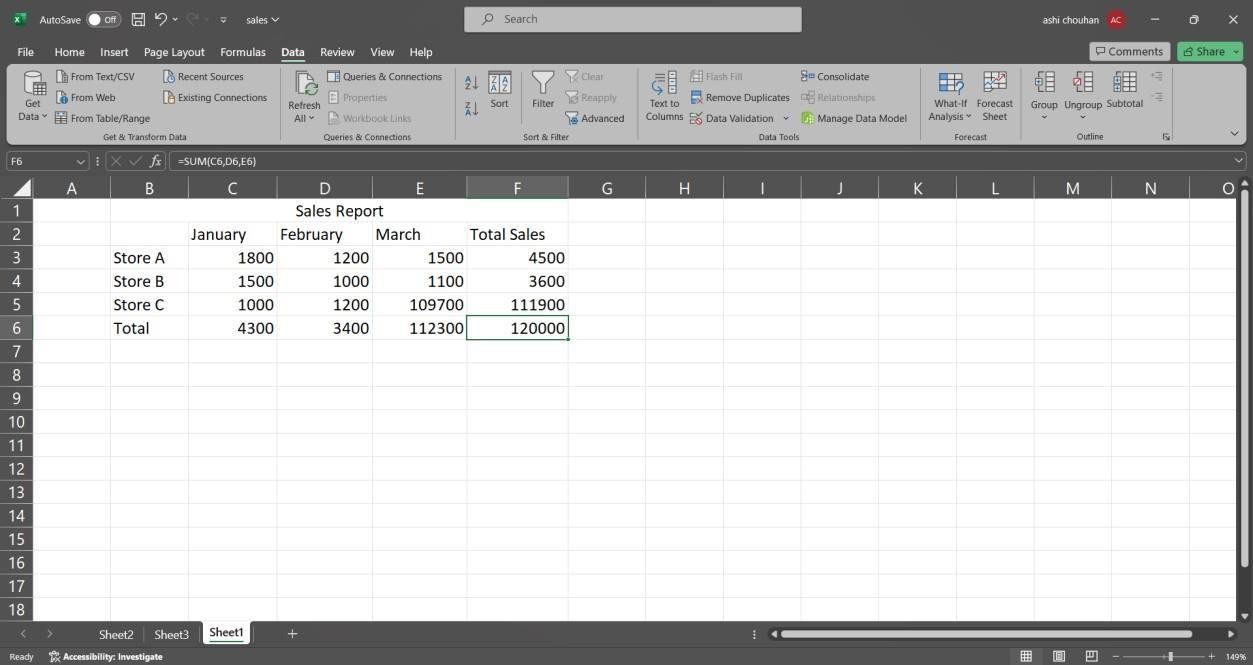
Step 1: In the Data tab go to the what if analysis>Goal seek.



Step 2: Fill the information in the window accordingly and click ok.







# PRACTICAL NO. 2

AIM:- Data Frames and Basic Data Pre-Processing

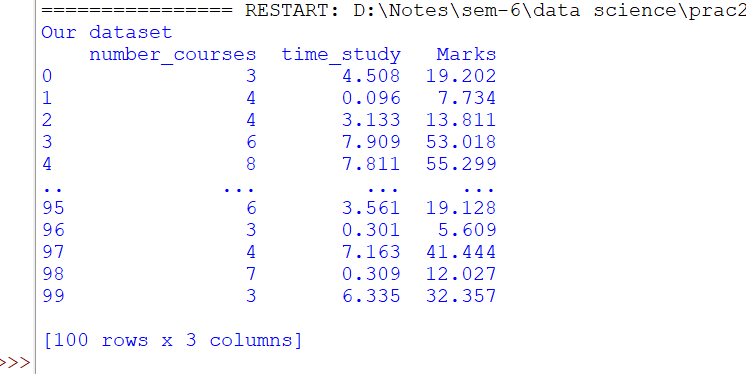
1. Read Data from CSV and JSON files into a data frame.

SOURCE CODE:-

#Read data from a csv file import pandas as pd

df = pd.read\_csv(‘Student\_Marks.csv’) print(“Our dataset”)

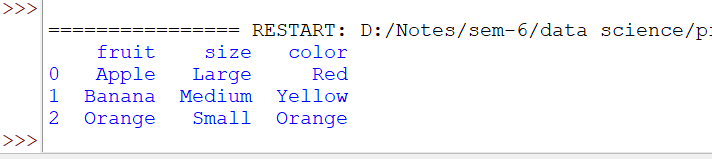
print(df) OUTPUT:-



SOURCE CODE:-

#Reading data from a JSON Import pandas as pd data=pd.read\_json(‘dataset.json’) print(data)

OUTPUT:-



1. Perform basic data pre-processing tasks such as handling missing values and outliers code:
   1. #Replacing NA values using fillna()

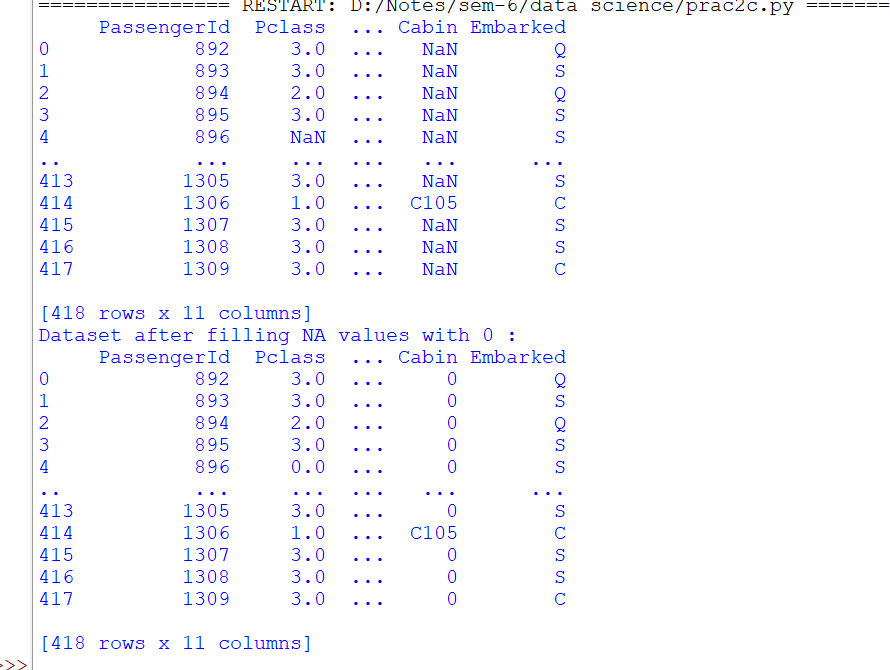
Import pandas as pd

df = pd.read\_csv(‘titanic.csv’) Print(df)

df.head(10)

print(“Dataset after filling NA values with 0:”) df2 = df.fillna(value = 0)

print(df2) OUTPUT:-



* 1. #Dropping NA values using dropna()

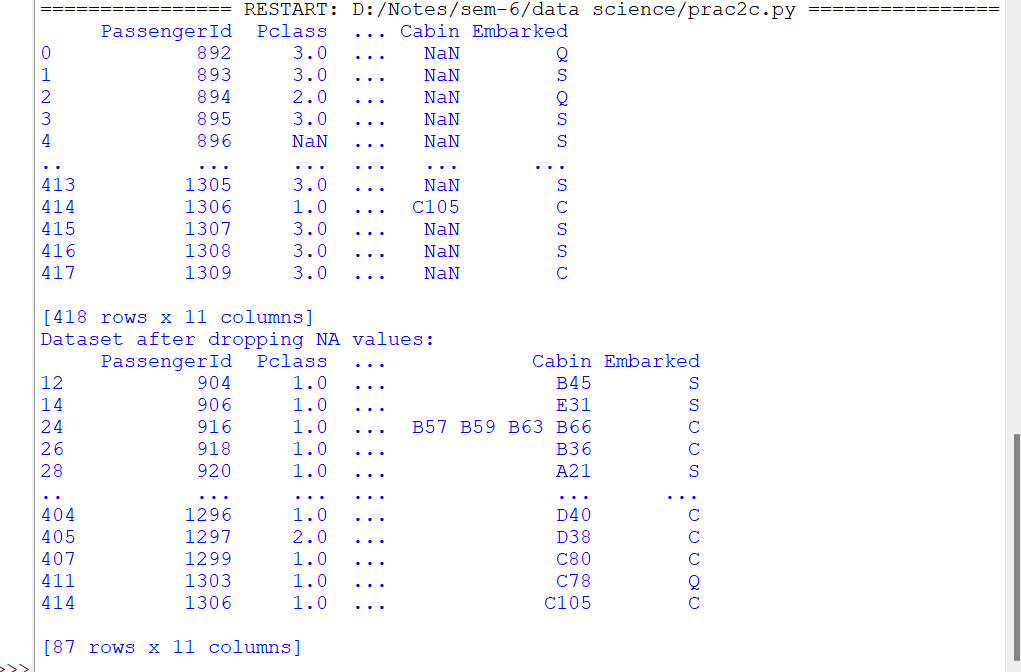
Import pandas as pd

df= pd.read\_csv(‘titanic.csv’) print(df)

df.head(10)

print(“Dataset after dropping NA values:”) df.dropna(inplace = True)

print(df) OUTPUT:-



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

SOURCE CODE:-

import pandas as pd # Load iris dataset

iris = pd.read\_csv('Iris.csv')

# Filtering data based on a

condition setosa = iris[iris['Species'] == 'setosa'] print("Setosa samples:") print(setosa.head())

# Sorting data sorted\_iris =

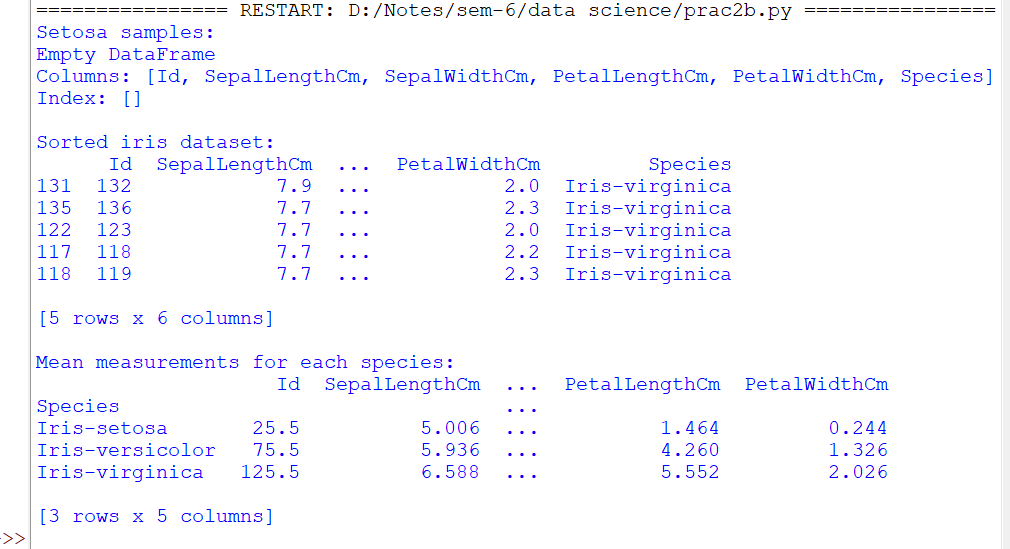
iris.sort\_values(by='SepalLengthCm',ascendin

g=False)

print("\nSorted iris dataset:") print(sorted\_iris.head())

# Grouping data grouped\_species = iris.groupby('Species').mean() print("\nMean measurements for each species:") print(grouped\_species)

OUTPUT:-



# PRACTICAL NO. 3

## AIM:- Feature Scaling and Dummification

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

SOURCE CODE:-

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler,

StandardScaler # Load dataset

df = pd.read\_csv('wine.csv', header=None, usecols=[0, 1, 2], skiprows=1)

df.columns = ['classlabel', 'Alcohol', 'Malic Acid'] # Display original DataFrame

print("Original DataFrame:") print(df)

# Apply Min-Max Scaling min\_max\_scaler = MinMaxScaler() df[['Alcohol', 'Malic Acid']] =

min\_max\_scaler.fit\_transform(df[['Alcohol', 'Malic Acid']])

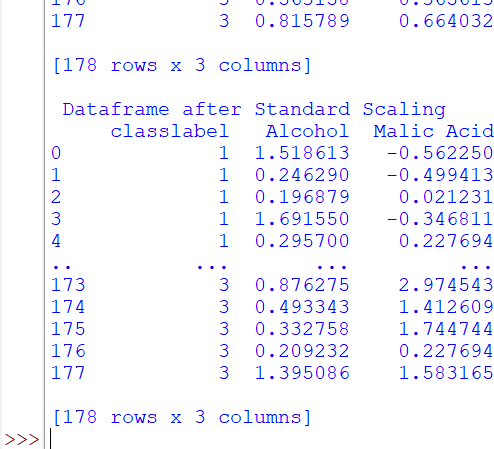
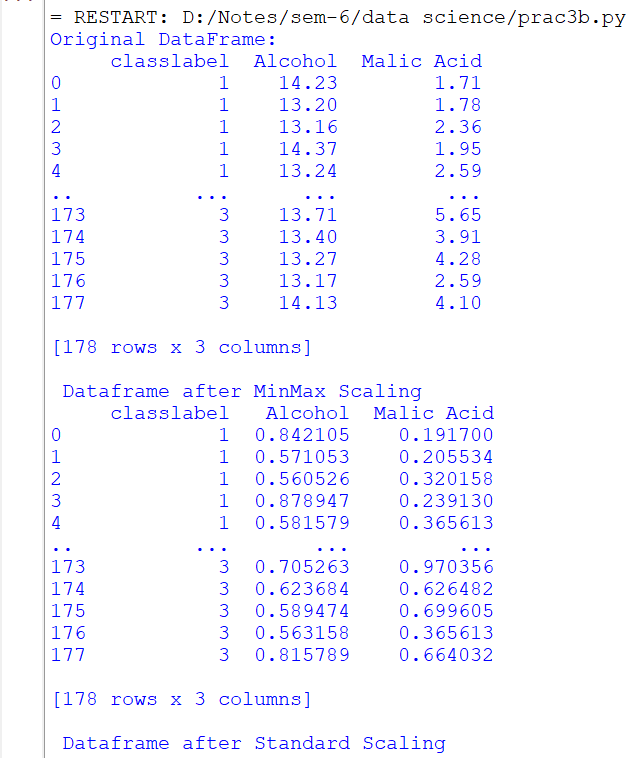
print("\nDataFrame after Min-Max Scaling:") print(df)

# Apply Standard Scaling standard\_scaler = StandardScaler() df[['Alcohol', 'Malic Acid']] =

standard\_scaler.fit\_transform(df[['Alcohol', 'Malic Acid']])

print("\nDataFrame after Standard Scaling:") print(df)

OUTPUT:-



1. Perform feature Dummification to convert categorical variables into numerical representations.

SOURCE CODE:-

import pandas as pd

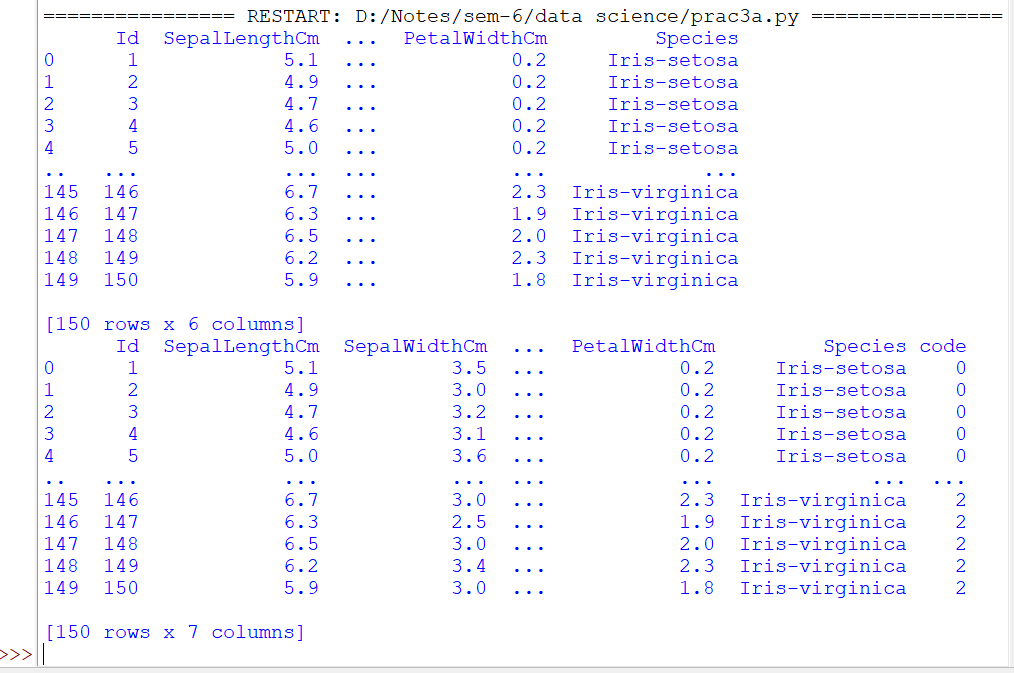
from sklearn.preprocessing import LabelEncoder # Load the dataset

iris = pd.read\_csv("Iris.csv") print(iris)

# Apply Label Encoding le = LabelEncoder()

iris['code'] = le.fit\_transform(iris['Species']) print(iris)

OUTPUT:-



# PRACTICAL NO. – 4

## AIM:- Hypothesis Testing SOURCE CODE:-

import numpy as np from scipy import stats

import matplotlib.pyplot as plt

# Generate two samples for demonstration purposes np.random.seed(42)

sample1 = np.random.normal(loc=10, scale=2, size=30) sample2 = np.random.normal(loc=12, scale=2, size=30)

# Perform a two-sample t-test

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

# Set the significance level alpha = 0.05

print("Results of Two-Sample t-test:") print(f'T-statistic: {t\_statistic}') print(f'P-value: {p\_value}')

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

# Plot the distributions plt.figure(figsize=(10, 6))

plt.hist(sample1, alpha=0.5, label='Sample 1', color='blue') plt.hist(sample2, alpha=0.5, label='Sample 2', color='orange')

plt.axvline(np.mean(sample1), color='blue', linestyle='dashed', linewidth=2)

plt.axvline(np.mean(sample2), color='orange', linestyle='dashed', linewidth=2)

plt.title('Distributions of Sample 1 and Sample 2') plt.xlabel('Values')

plt.ylabel('Frequency') plt.legend()

# Highlight the critical region if null hypothesis is rejected if p\_value < alpha:

critical\_region = np.linspace(min(sample1.min(), sample2.min()), max(sample1.max(), sample2.max()), 1000)

plt.fill\_between(critical\_region, 0, 5, color='red', alpha=0.3, label='Critical Region')

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

# Show the plot plt.show()

# Draw Conclusions 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 of Sample 1 is significantly higher than that of Sample 2.")

else:

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

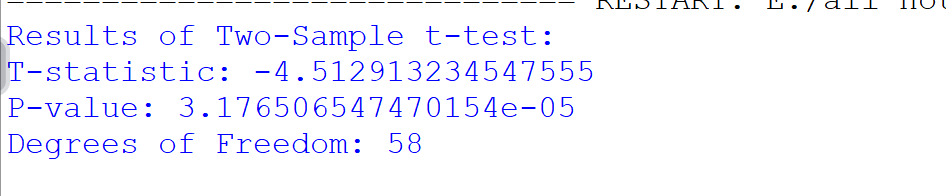
print("Interpretation: The mean of Sample 2 is significantly higher than that of Sample 1.")

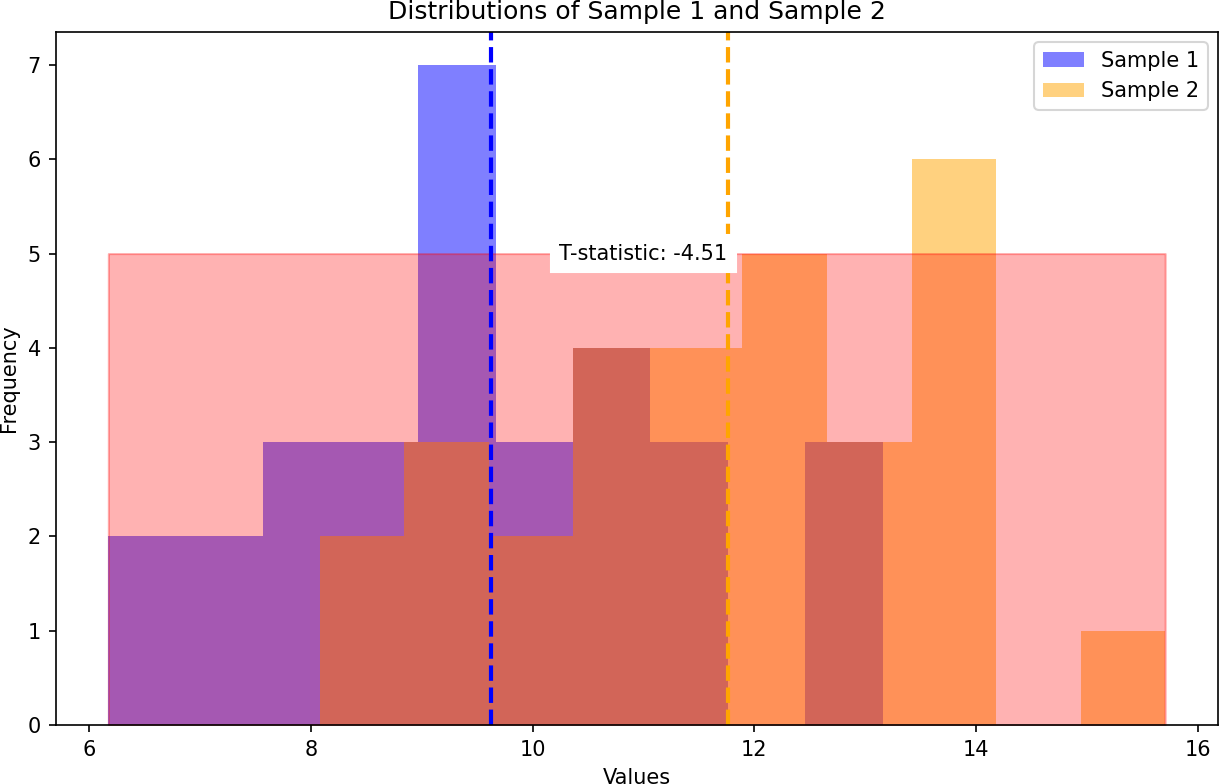
else:

print("Conclusion: Fail to reject the null hypothesis.") print("Interpretation: There is not enough evidence to claim a

significant difference between the means.")

**OUTPUT:-**

****



**SOURCE CODE:-**

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sb

import warnings

from scipy import stats

# Suppress warnings warnings.filterwarnings('ignore')

# Load dataset

df = sb.load\_dataset('mpg') print(df)

# Describe horsepower and model year columns print(df['horsepower'].describe())

print(df['model\_year'].describe())

# Categorize horsepower into bins bins = [0, 75, 150, 240]

df['horsepower\_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h']) c = df['horsepower\_new']

print(c)

# Categorize model year into bins ybins = [69, 72, 74, 84]

labels = ['t1', 't2', 't3']

df['modelyear\_new'] = pd.cut(df['model\_year'], bins=ybins, labels=labels) newyear = df['modelyear\_new']

print(newyear)

# Create a contingency table

df\_chi = pd.crosstab(df['horsepower\_new'], df['modelyear\_new']) print(df\_chi)

# Perform chi-square test

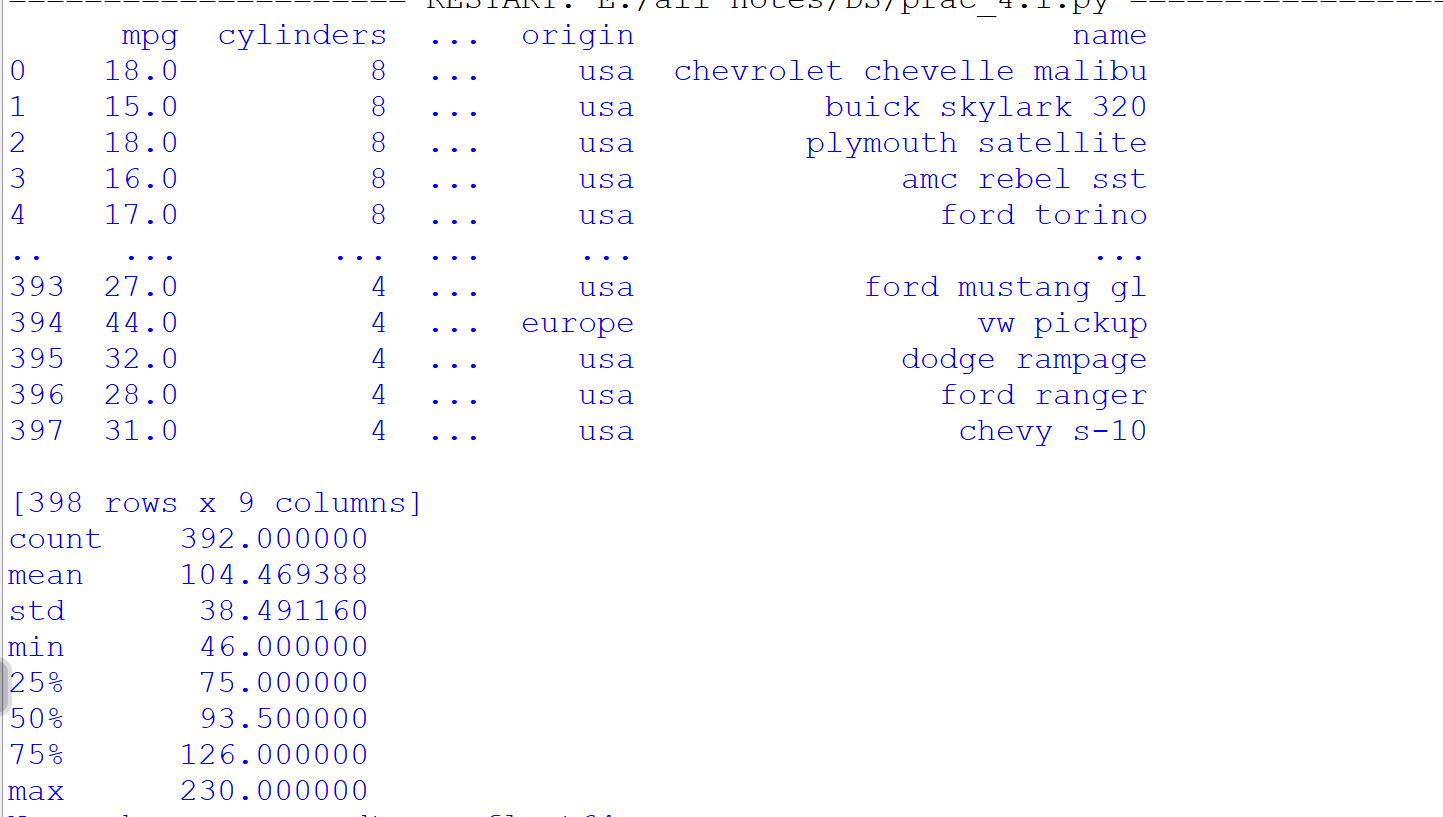
chi2\_stat, p\_value, dof, expected = stats.chi2\_contingency(df\_chi) print(f'Chi-Square Statistic: {chi2\_stat}')

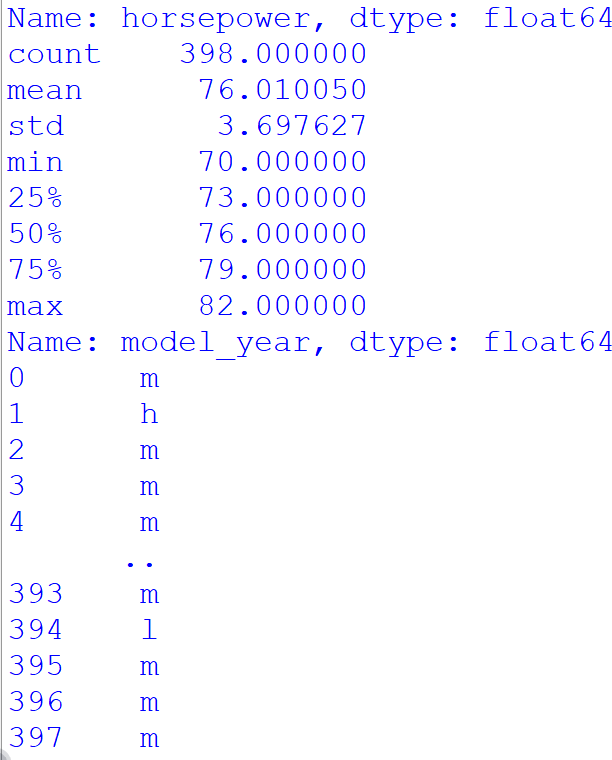
print(f'P-value: {p\_value}')

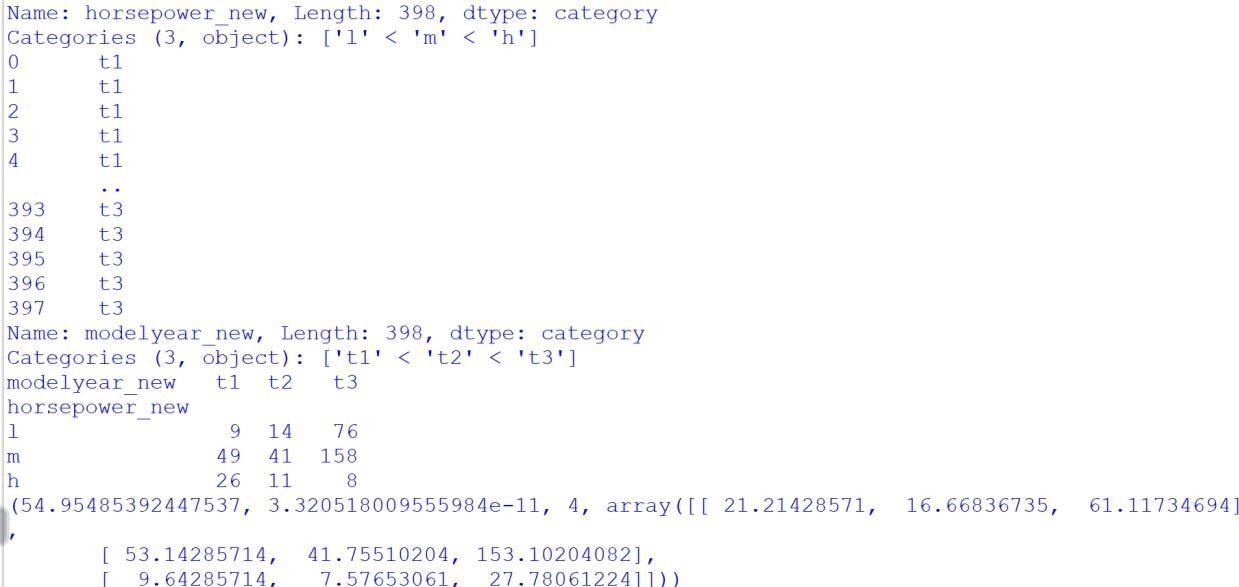
print(f'Degrees of Freedom: {dof}')

print(f'Expected Frequencies:\n{expected}')

# OUTPUT:-

****





**PRACTICAL NO. – 5**

**AIM:- ANOVA(Analysis of Variance) SOURCE CODE:-**

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sb

import warnings

from scipy import stats

from statsmodels.stats.multicomp import pairwise\_tukeyhsd

# Suppress warnings warnings.filterwarnings('ignore')

# Load dataset

df = sb.load\_dataset('mpg') print(df)

# Describe horsepower and model year columns print(df['horsepower'].describe()) print(df['model\_year'].describe())

# Categorize horsepower into bins bins = [0, 75, 150, 240]

df['horsepower\_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h'])

c = df['horsepower\_new'] print(c)

# Categorize model year into bins ybins = [69, 72, 74, 84]

labels = ['t1', 't2', 't3']

df['modelyear\_new'] = pd.cut(df['model\_year'], bins=ybins, labels=labels)

newyear = df['modelyear\_new'] print(newyear)

# Create a contingency table

df\_chi = pd.crosstab(df['horsepower\_new'], df['modelyear\_new']) print(df\_chi)

# Perform chi-square test

chi2\_stat, p\_value, dof, expected = stats.chi2\_contingency(df\_chi) print(f'Chi-Square Statistic: {chi2\_stat}')

print(f'P-value: {p\_value}') print(f'Degrees of Freedom: {dof}')

print(f'Expected Frequencies:\n{expected}')

# Define groups for ANOVA group1 = [23, 25, 29, 34, 30]

group2 = [19, 20, 22, 24, 25]

group3 = [15, 18, 20, 21, 17]

group4 = [28, 24, 26, 30, 29]

# Combine data into a DataFrame

data = pd.DataFrame({'value': group1 + group2 + group3 + group4, 'group': ['Group1'] \* len(group1) + ['Group2'] \*

len(group2) +

['Group3'] \* len(group3) + ['Group4'] \*

len(group4)})

# Perform one-way ANOVA

f\_statistics, p\_value\_anova = stats.f\_oneway(group1, group2, group3, group4)

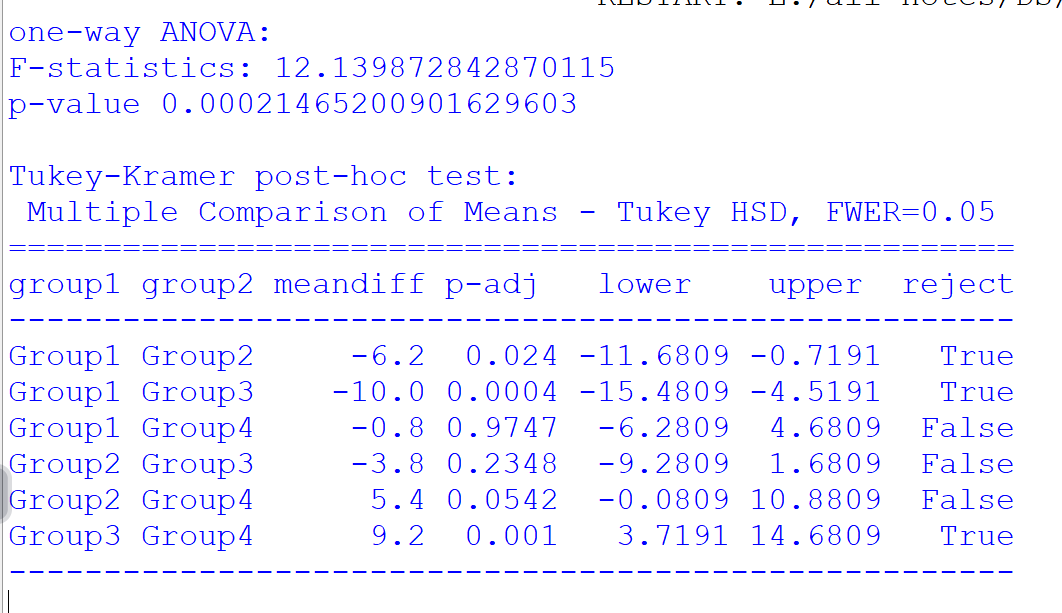
print("\nOne-way ANOVA:") print("F-statistics:", f\_statistics) print("P-value:", p\_value\_anova)

# Perform Tukey-Kramer post-hoc test

tukey\_results = pairwise\_tukeyhsd(data['value'], data['group']) print("\nTukey-Kramer post-hoc test:")

print(tukey\_results)

**OUTPUT:-**

****

# PRACTICAL NO. – 6

## AIM:- Regression and its Types SOURCE CODE:-

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sb

import warnings

from scipy import stats

from statsmodels.stats.multicomp import pairwise\_tukeyhsd from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Suppress warnings warnings.filterwarnings('ignore')

# Load dataset

df = sb.load\_dataset('mpg') print(df)

# Describe horsepower and model year columns

print(df['horsepower'].describe()) print(df['model\_year'].describe())

# Categorize horsepower into bins bins = [0, 75, 150, 240]

df['horsepower\_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h'])

c = df['horsepower\_new'] print(c)

# Categorize model year into bins ybins = [69, 72, 74, 84]

labels = ['t1', 't2', 't3']

df['modelyear\_new'] = pd.cut(df['model\_year'], bins=ybins, labels=labels)

newyear = df['modelyear\_new'] print(newyear)

# Create a contingency table

df\_chi = pd.crosstab(df['horsepower\_new'], df['modelyear\_new']) print(df\_chi)

# Perform chi-square test

chi2\_stat, p\_value, dof, expected = stats.chi2\_contingency(df\_chi)

print(f'Chi-Square Statistic: {chi2\_stat}') print(f'P-value: {p\_value}') print(f'Degrees of Freedom: {dof}')

print(f'Expected Frequencies:\n{expected}')

# Define groups for ANOVA group1 = [23, 25, 29, 34, 30]

group2 = [19, 20, 22, 24, 25]

group3 = [15, 18, 20, 21, 17]

group4 = [28, 24, 26, 30, 29]

# Combine data into a DataFrame

data = pd.DataFrame({'value': group1 + group2 + group3 + group4, 'group': ['Group1'] \* len(group1) + ['Group2'] \*

len(group2) +

['Group3'] \* len(group3) + ['Group4'] \*

len(group4)})

# Perform one-way ANOVA

f\_statistics, p\_value\_anova = stats.f\_oneway(group1, group2, group3, group4)

print("\nOne-way ANOVA:") print("F-statistics:", f\_statistics) print("P-value:", p\_value\_anova)

# Perform Tukey-Kramer post-hoc test

tukey\_results = pairwise\_tukeyhsd(data['value'], data['group']) print("\nTukey-Kramer post-hoc test:")

print(tukey\_results)

# Load California housing dataset

housing = fetch\_california\_housing()

housing\_df = pd.DataFrame(housing.data, columns=housing.feature\_names)

print(housing\_df)

# Add target variable housing\_df['PRICE'] = housing.target

# Select feature and target

X = housing\_df[['AveRooms']] y = housing\_df['PRICE']

# Split dataset into training and testing sets

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

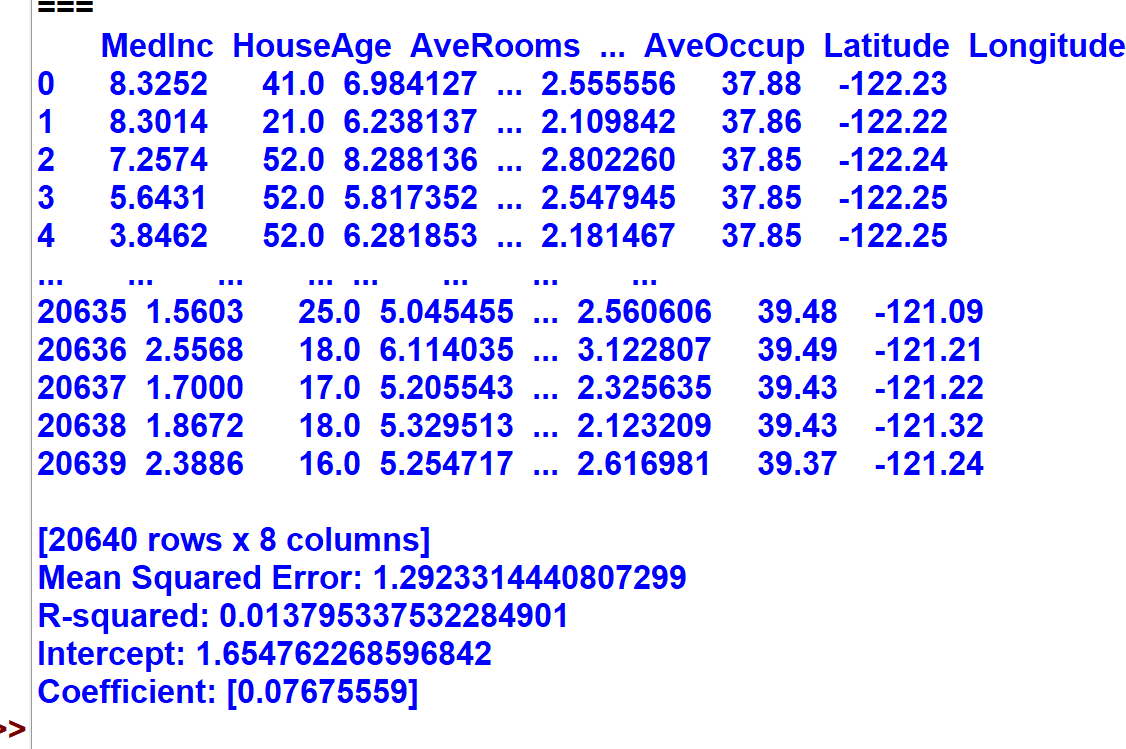
# Train linear regression model model = LinearRegression()

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

# Evaluate model

mse = mean\_squared\_error(y\_test, model.predict(X\_test)) r2 = r2\_score(y\_test, model.predict(X\_test))

print("Mean Squared Error:", mse) print("R-squared:", r2) print("Intercept:", model.intercept\_) print("Coefficient:", model.coef\_) **OUTPUT:-**



SOURCE CODE:-

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sb import warnings

from scipy import stats

from statsmodels.stats.multicomp import pairwise\_tukeyhsd from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Suppress warnings warnings.filterwarnings('ignore')

# Load dataset

df = sb.load\_dataset('mpg') print(df)

# Describe horsepower and model year columns print(df['horsepower'].describe()) print(df['model\_year'].describe())

# Categorize horsepower into bins bins = [0, 75, 150, 240]

df['horsepower\_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h'])

c = df['horsepower\_new'] print(c)

# Categorize model year into bins ybins = [69, 72, 74, 84]

labels = ['t1', 't2', 't3']

df['modelyear\_new'] = pd.cut(df['model\_year'], bins=ybins, labels=labels)

newyear = df['modelyear\_new'] print(newyear)

# Create a contingency table

df\_chi = pd.crosstab(df['horsepower\_new'], df['modelyear\_new'])

print(df\_chi)

# Perform chi-square test

chi2\_stat, p\_value, dof, expected = stats.chi2\_contingency(df\_chi)

print(f'Chi-Square Statistic: {chi2\_stat}') print(f'P-value: {p\_value}')

print(f'Degrees of Freedom: {dof}') print(f'Expected Frequencies:\n{expected}')

# Define groups for ANOVA group1 = [23, 25, 29, 34, 30]

group2 = [19, 20, 22, 24, 25]

group3 = [15, 18, 20, 21, 17]

group4 = [28, 24, 26, 30, 29]

# Combine data into a DataFrame

data = pd.DataFrame({'value': group1 + group2 + group3 + group4,

'group': ['Group1'] \* len(group1) + ['Group2'] \* len(group2) +

['Group3'] \* len(group3) + ['Group4'] \*

len(group4)})

# Perform one-way ANOVA

f\_statistics, p\_value\_anova = stats.f\_oneway(group1, group2, group3, group4)

print("\nOne-way ANOVA:") print("F-statistics:", f\_statistics) print("P-value:", p\_value\_anova)

# Perform Tukey-Kramer post-hoc test

tukey\_results = pairwise\_tukeyhsd(data['value'], data['group']) print("\nTukey-Kramer post-hoc test:")

print(tukey\_results)

# Load California housing dataset housing = fetch\_california\_housing()

housing\_df = pd.DataFrame(housing.data, columns=housing.feature\_names)

print(housing\_df)

# Add target variable housing\_df['PRICE'] = housing.target

# Multiple Linear Regression

X = housing\_df.drop('PRICE', axis=1) y = housing\_df['PRICE']

# Split dataset into training and testing sets

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

# Train linear regression model model = LinearRegression() model.fit(X\_train, y\_train)

# Make predictions

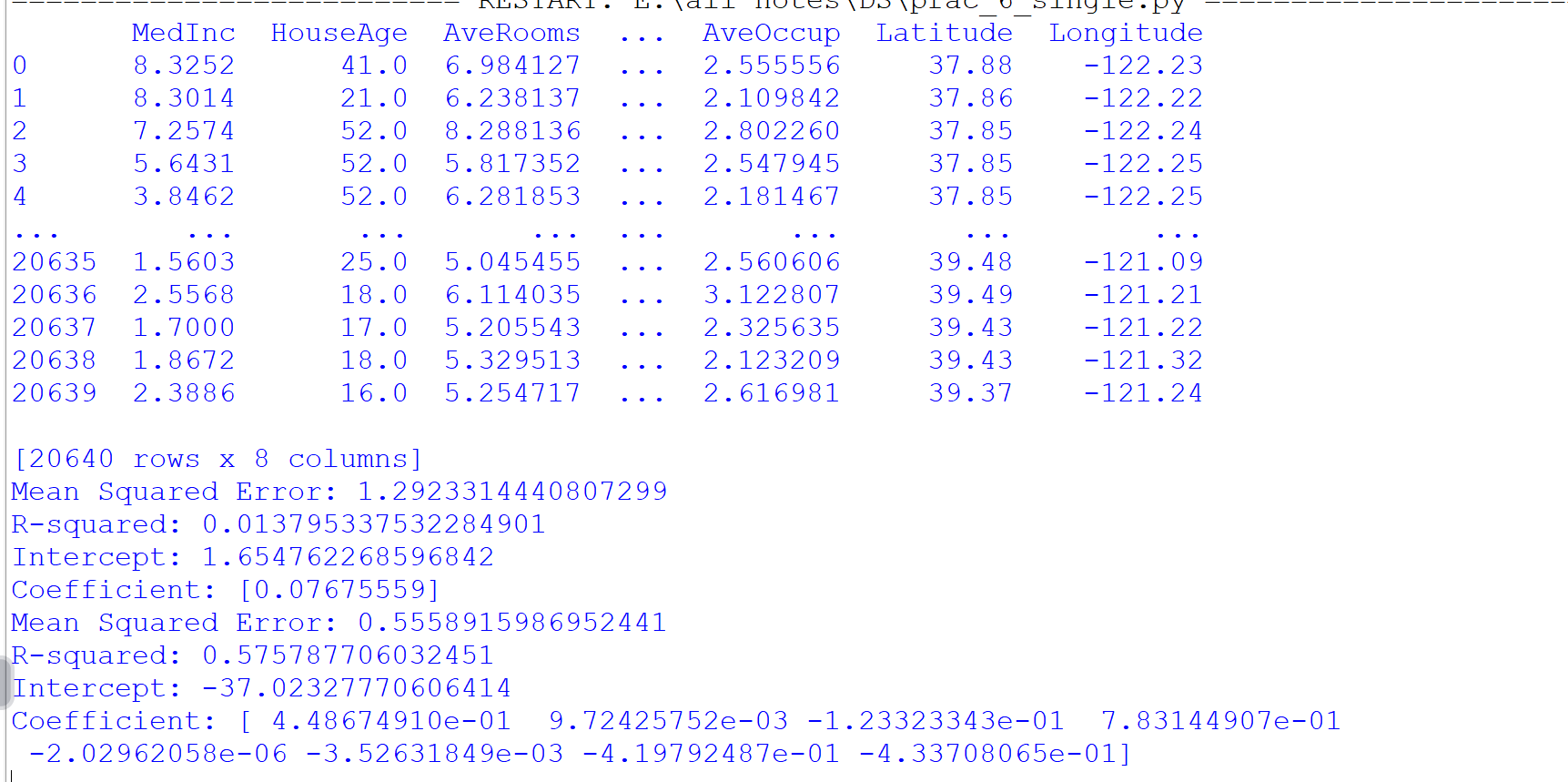
y\_pred = model.predict(X\_test)

# Evaluate model

mse = mean\_squared\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse) print("R-squared:", r2) print("Intercept:", model.intercept\_) print("Coefficient:", model.coef\_)

OUTPUT:-



# PRACTICAL NO. – 7

## AIM:- Logistic Regression and Decision Tree SOURCE CODE:-

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sb

import warnings

from scipy import stats

from statsmodels.stats.multicomp import pairwise\_tukeyhsd from sklearn.datasets import fetch\_california\_housing, load\_iris from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression,

LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import mean\_squared\_error, r2\_score, accuracy\_score, precision\_score, recall\_score, classification\_report

# Suppress warnings warnings.filterwarnings('ignore')

# Load dataset

df = sb.load\_dataset('mpg') print(df)

# Describe horsepower and model year columns print(df['horsepower'].describe()) print(df['model\_year'].describe())

# Categorize horsepower into bins bins = [0, 75, 150, 240]

df['horsepower\_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h'])

c = df['horsepower\_new'] print(c)

# Categorize model year into bins ybins = [69, 72, 74, 84]

labels = ['t1', 't2', 't3']

df['modelyear\_new'] = pd.cut(df['model\_year'], bins=ybins, labels=labels)

newyear = df['modelyear\_new'] print(newyear)

# Create a contingency table

df\_chi = pd.crosstab(df['horsepower\_new'], df['modelyear\_new']) print(df\_chi)

# Perform chi-square test

chi2\_stat, p\_value, dof, expected = stats.chi2\_contingency(df\_chi) print(f'Chi-Square Statistic: {chi2\_stat}')

print(f'P-value: {p\_value}') print(f'Degrees of Freedom: {dof}')

print(f'Expected Frequencies:\n{expected}')

# Define groups for ANOVA group1 = [23, 25, 29, 34, 30]

group2 = [19, 20, 22, 24, 25]

group3 = [15, 18, 20, 21, 17]

group4 = [28, 24, 26, 30, 29]

# Combine data into a DataFrame

data = pd.DataFrame({'value': group1 + group2 + group3 + group4, 'group': ['Group1'] \* len(group1) + ['Group2'] \*

len(group2) +

['Group3'] \* len(group3) + ['Group4'] \*

len(group4)})

# Perform one-way ANOVA

f\_statistics, p\_value\_anova = stats.f\_oneway(group1, group2, group3, group4)

print("\nOne-way ANOVA:")

print("F-statistics:", f\_statistics)

print("P-value:", p\_value\_anova)

# Perform Tukey-Kramer post-hoc test

tukey\_results = pairwise\_tukeyhsd(data['value'], data['group']) print("\nTukey-Kramer post-hoc test:")

print(tukey\_results)

# Load California housing dataset housing = fetch\_california\_housing()

housing\_df = pd.DataFrame(housing.data, columns=housing.feature\_names)

print(housing\_df)

# Add target variable housing\_df['PRICE'] = housing.target

# Multiple Linear Regression

X = housing\_df.drop('PRICE', axis=1) y = housing\_df['PRICE']

# Split dataset into training and testing sets

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

# Train linear regression model model = LinearRegression() model.fit(X\_train, y\_train)

# Make predictions

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

# Evaluate model

mse = mean\_squared\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse) print("R-squared:", r2) print("Intercept:", model.intercept\_) print("Coefficient:", model.coef\_)

# Load the Iris dataset and classification problem iris = load\_iris()

iris\_df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

binary\_df = iris\_df[iris\_df['target'] != 2] X = binary\_df.drop('target', axis=1)

y = binary\_df['target']

# Split the data into training and testing sets

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

# Train a logistic regression model and evaluate its performance logistic\_model = LogisticRegression() logistic\_model.fit(X\_train, y\_train)

y\_pred\_logistic = logistic\_model.predict(X\_test)

print("\nLogistic Regression Metrics")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_logistic)) print("Precision:", precision\_score(y\_test, y\_pred\_logistic)) print("Recall:", recall\_score(y\_test, y\_pred\_logistic)) print("\nClassification Report") print(classification\_report(y\_test, y\_pred\_logistic))

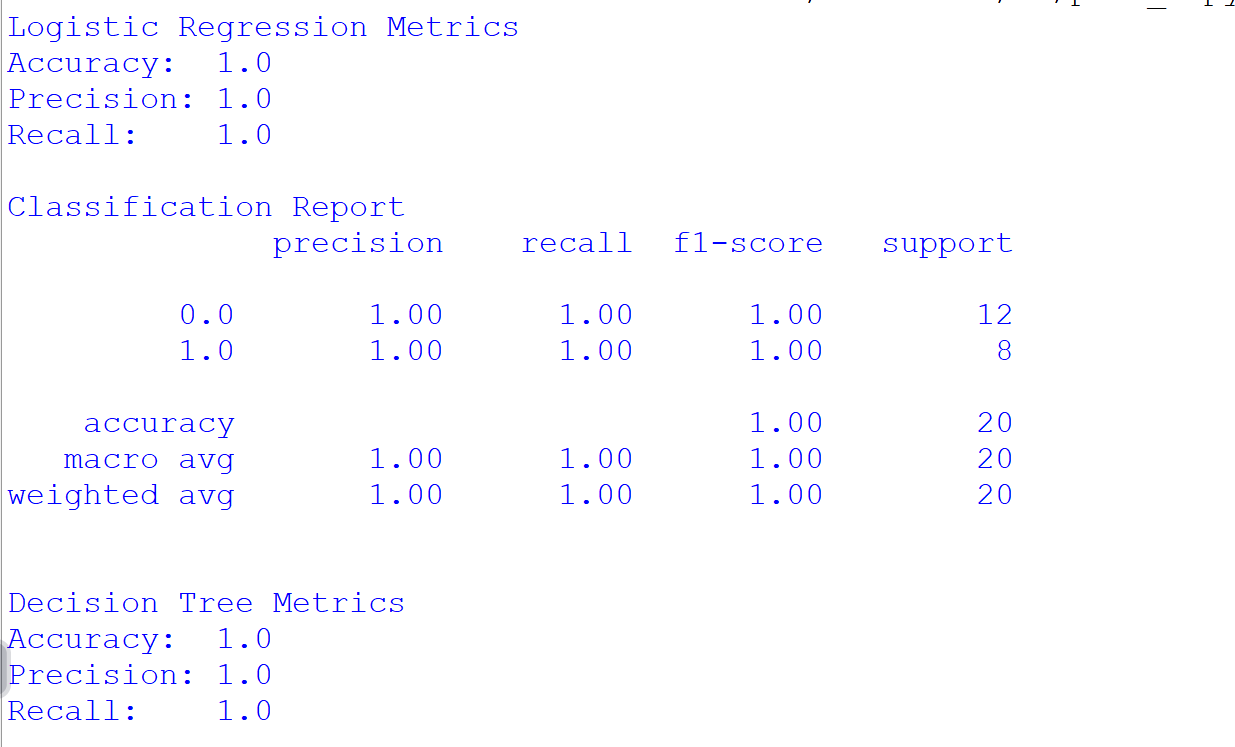
# Train a decision tree model and evaluate its performance decision\_tree\_model = DecisionTreeClassifier() decision\_tree\_model.fit(X\_train, y\_train)

y\_pred\_tree = decision\_tree\_model.predict(X\_test)

print("\nDecision Tree Metrics")

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_tree)) print("Precision:", precision\_score(y\_test, y\_pred\_tree)) print("Recall:", recall\_score(y\_test, y\_pred\_tree))

print("\nClassification Report") print(classification\_report(y\_test, y\_pred\_tree)) **OUTPUT:-**



# PRACTICAL NO. – 8 AIM:- K-MEANS CLUSTERING SOURCE CODE:-

import pandas as pd

from sklearn.preprocessing import MinMaxScaler from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Load dataset data =

pd.read\_csv("C:\\Users\\Reape\\Downloads\\wholesale\\wholesale.csv ")

data.head()

# Define categorical and continuous features categorical\_features = ['Channel', 'Region']

continuous\_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents\_Paper', 'Delicassen']

data[continuous\_features].describe()

# Convert categorical variables into dummy variables for col in categorical\_features:

dummies = pd.get\_dummies(data[col], prefix=col) data = pd.concat([data, dummies], axis=1)

data.drop(col, axis=1, inplace=True)

data.head()

# Scale the data

mms = MinMaxScaler() mms.fit(data)

data\_transformed = mms.transform(data)

# Elbow Method to find optimal k sum\_of\_squared\_distances = []

K = range(1, 15) for k in K:

km = KMeans(n\_clusters=k) km = km.fit(data\_transformed)

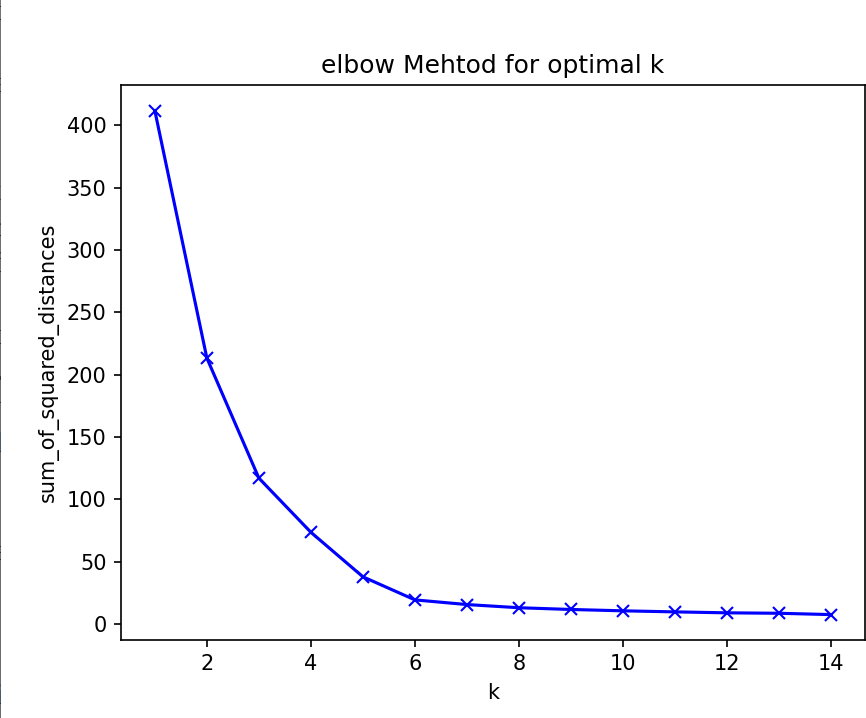
sum\_of\_squared\_distances.append(km.inertia\_)

# Plot the Elbow Method

plt.plot(K, sum\_of\_squared\_distances, 'bx-') plt.xlabel('k')

plt.ylabel('Sum of Squared Distances') plt.title('Elbow Method for Optimal k') plt.show()

# OUTPUT:-



**PRACTICAL NO. 9**

## AIM:- Principal Component Analysis (PCA) SOURCE CODE:-

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA

# Load the Iris dataset iris = load\_iris()

iris\_df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

# Separate features and target

X = iris\_df.drop('target', axis=1) y = iris\_df['target']

# Standardize the features scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Apply PCA

pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled) explained\_variance\_ratio = pca.explained\_variance\_ratio\_

# Plot cumulative explained variance plt.figure(figsize=(8, 6))

plt.plot(np.cumsum(explained\_variance\_ratio), marker='o', linestyle='--')

plt.title('Explained Variance Ratio') plt.xlabel('Number of Principal Components') plt.ylabel('Cumulative Explained Variance Ratio') plt.grid(True)

plt.show()

# Determine the number of components to explain 95% variance cumulative\_variance\_ratio = np.cumsum(explained\_variance\_ratio) n\_components = np.argmax(cumulative\_variance\_ratio >= 0.95) + 1

print(f"Number of principal components to explain 95% variance:

{n\_components}")

# Reduce dimensions using selected number of components pca = PCA(n\_components=n\_components)

X\_reduced = pca.fit\_transform(X\_scaled)

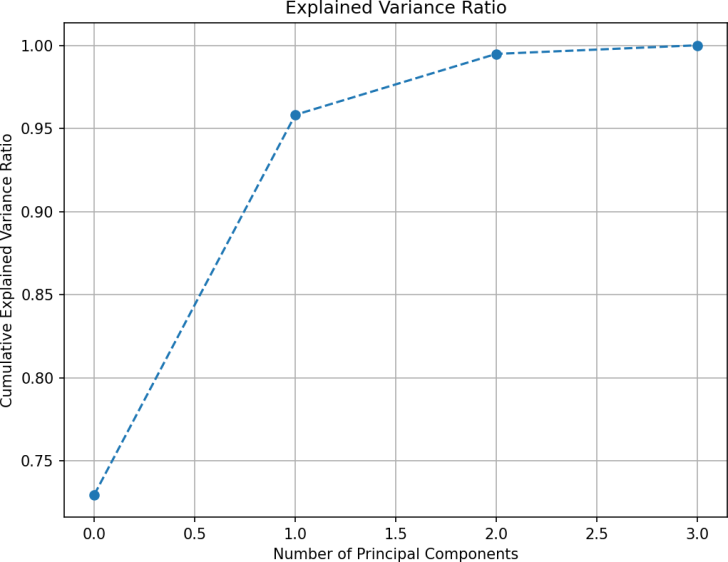
# Scatter plot of the reduced data plt.figure(figsize=(8, 6))

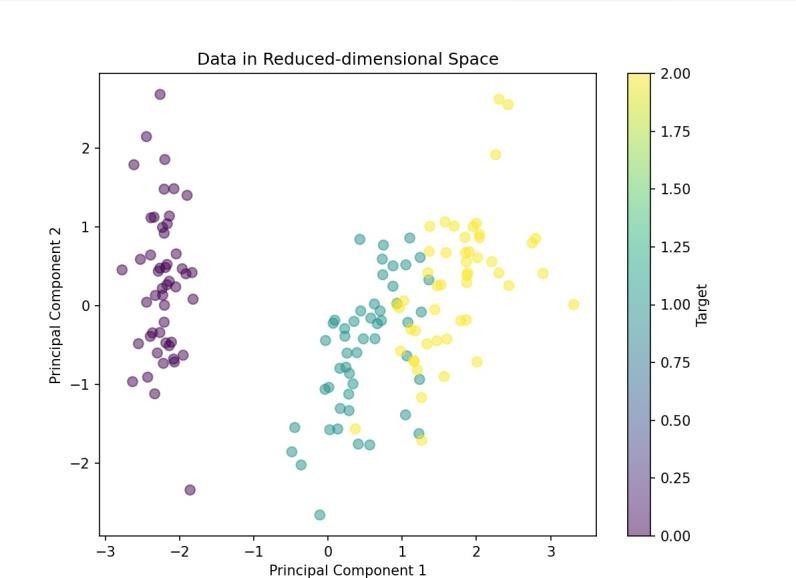
plt.scatter(X\_reduced[:, 0], X\_reduced[:, 1], c=y, cmap='viridis', s=50, alpha=0.5)

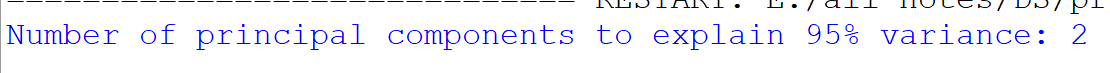
plt.title('Data in Reduced-dimensional Space') plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2') plt.colorbar(label='Target') plt.show()

# OUTPUT:-

****



****

**PRACTICAL NO. – 10**

## AIM:-Data Visualization and Storytelling SOURCE CODE:-

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

import numpy as np

# Generate random data

np.random.seed(42) # Set a seed for reproducibility

# Create a DataFrame with random data data = pd.DataFrame({

'variable1': np.random.normal(0, 1, 1000),

'variable2': np.random.normal(2, 2, 1000) + 0.5 \*

np.random.normal(0, 1, 1000),

'variable3': np.random.normal(-1, 1.5, 1000),

'category': pd.Series(np.random.choice(['A', 'B', 'C', 'D'], size=1000, p=[0.4, 0.3, 0.2, 0.1]),

dtype='category')

})

# Create a scatter plot to visualize the relationship between two variables

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

plt.scatter(data['variable1'], data['variable2'], alpha=0.5) plt.title('Relationship between Variable 1 and Variable 2', fontsize=16) plt.xlabel('Variable 1', fontsize=14)

plt.ylabel('Variable 2', fontsize=14) plt.show()

# Create a bar chart to visualize the distribution of a categorical variable

plt.figure(figsize=(10, 6)) sns.countplot(x='category', data=data) plt.title('Distribution of Categories', fontsize=16) plt.xlabel('Category', fontsize=14) plt.ylabel('Count', fontsize=14) plt.xticks(rotation=45)

plt.show()

# Create a heatmap to visualize the correlation between numerical variables

plt.figure(figsize=(10, 8))

numerical\_cols = ['variable1', 'variable2', 'variable3']

sns.heatmap(data[numerical\_cols].corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap', fontsize=16)

plt.show()

# Data Storytelling

print("Title: Exploring the Relationship between Variable 1 and Variable 2")

print("\nThe scatter plot (Figure 1) shows the relationship between Variable 1 and Variable 2.")

print("\nScatter Plot")

print("Figure 1: Scatter Plot of Variable 1 and Variable 2")

print("\nTo better understand the distribution of the categorical variable 'category', we created a ")

print("\nBar Chart")

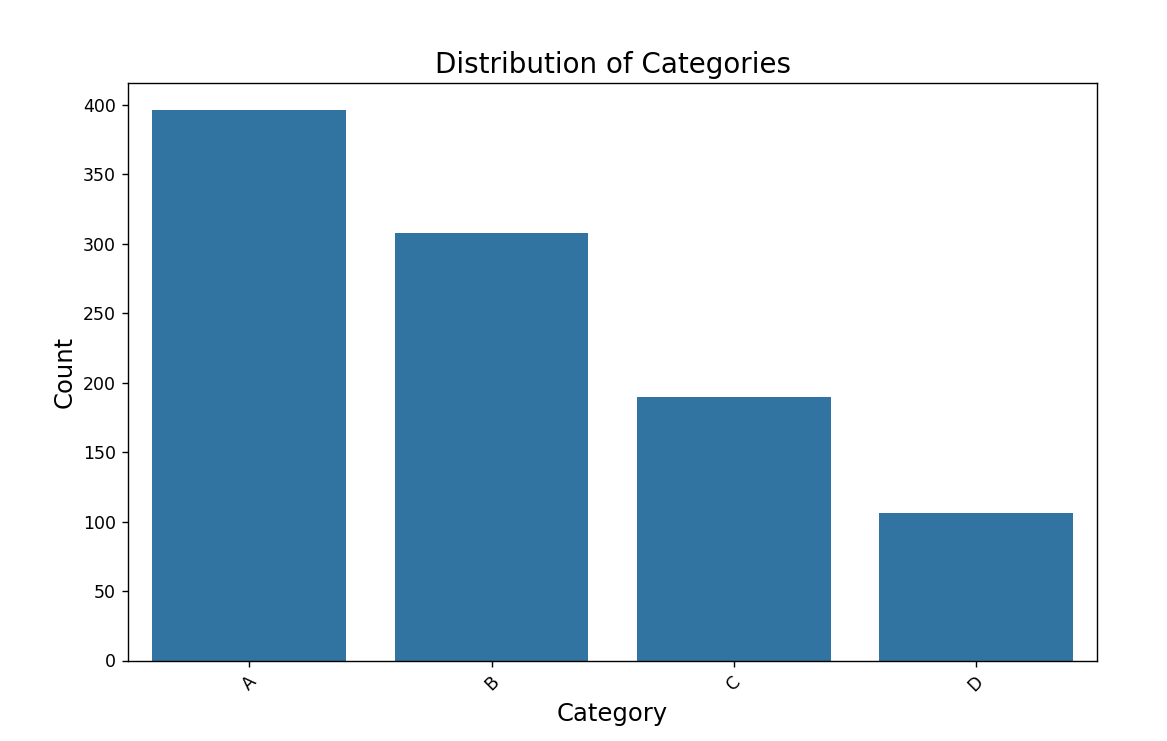
print("Figure 2: Distribution of Categories")

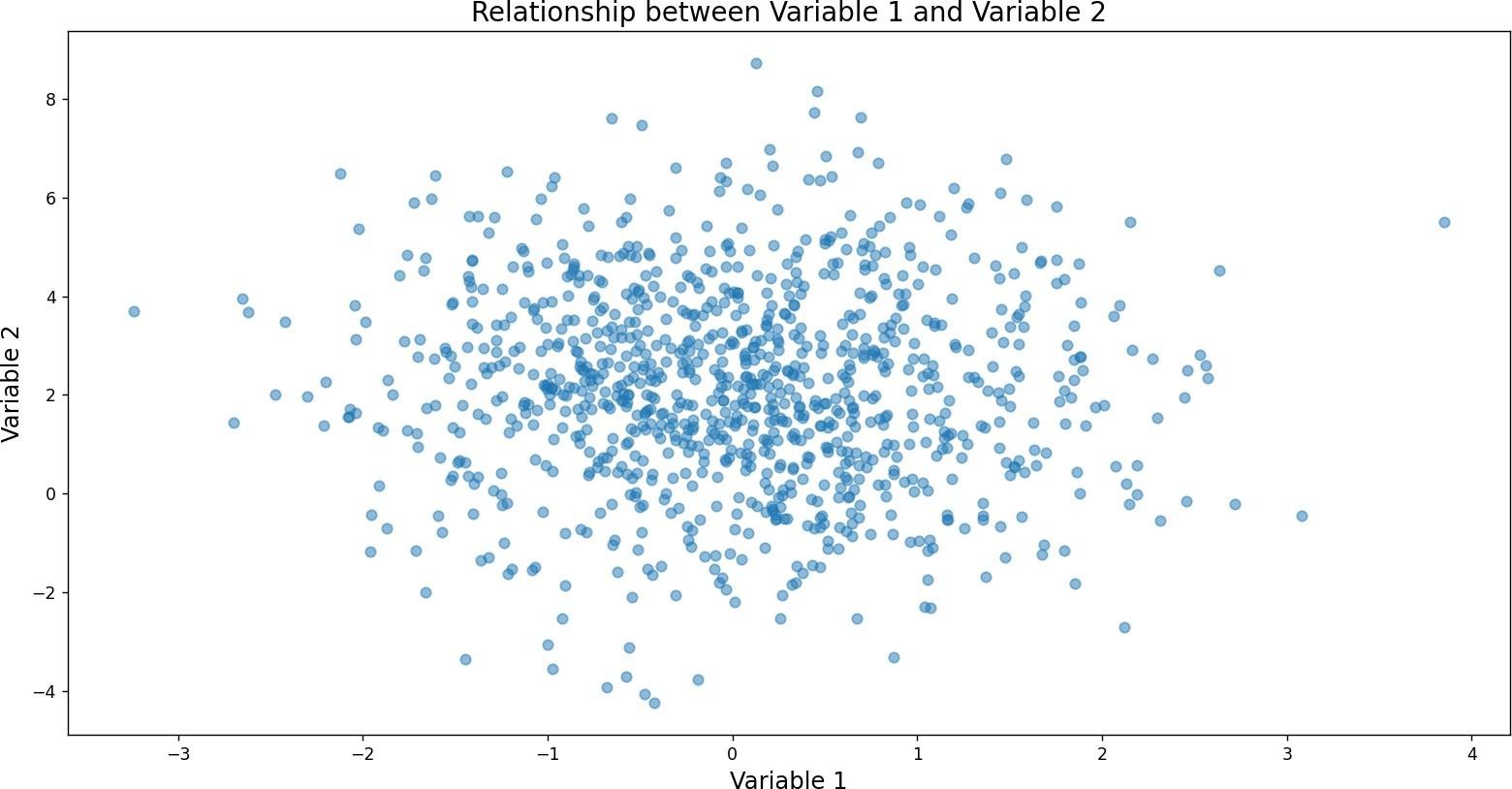
print("\nAdditionally, we explored the correlation between numerical variables using a heatmap")

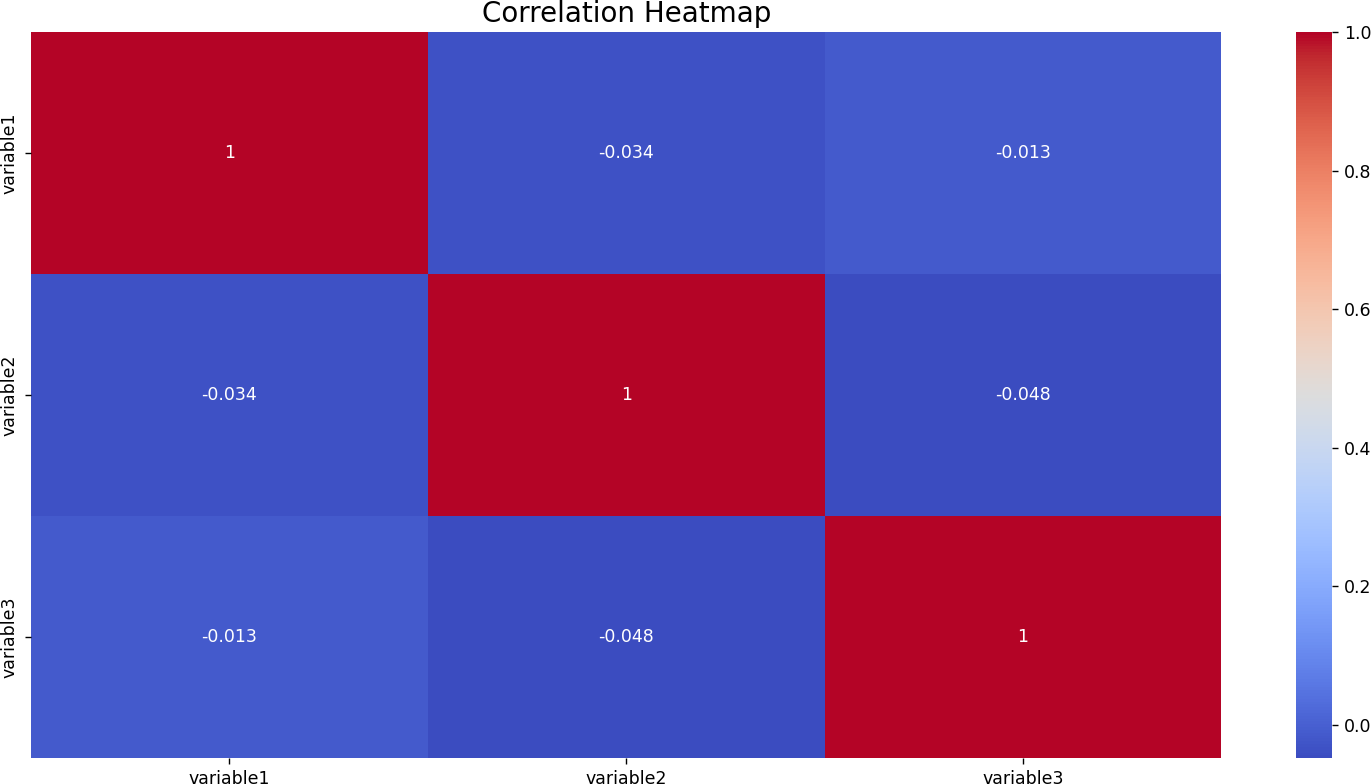
print("\nHeatmap")

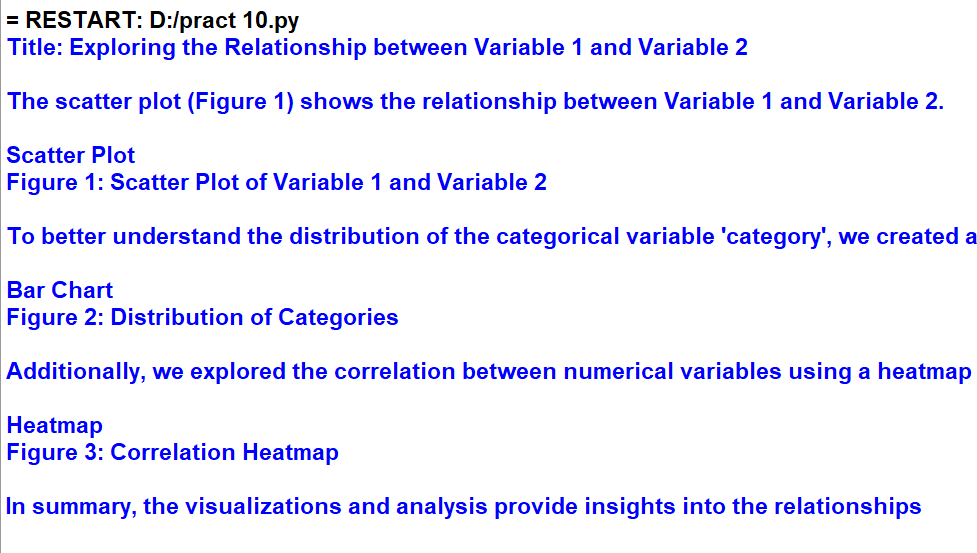
print("Figure 3: Correlation Heatmap")

print("\nIn summary, the visualizations and analysis provide insights into the relationships")



**OUTPUT:-**



****