# INDEX

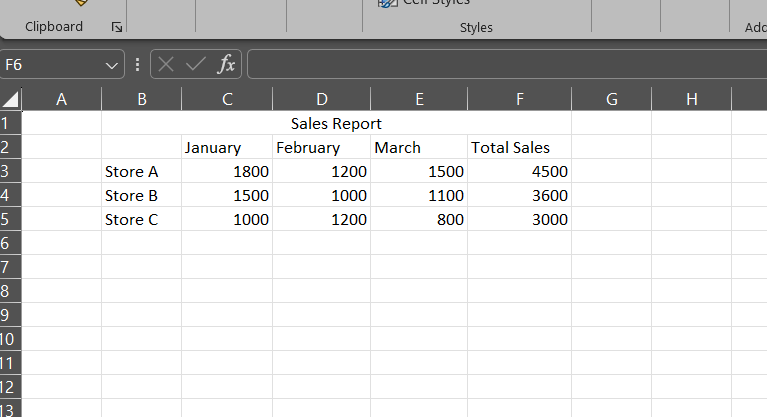
|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. no** | **Practical** | **Date** | **Signature** |
| 1 | Introduction to Excel | 8/01/24 |  |
| 2 | Data Frames and Basic Data Pre-processing | 15/01/24 |  |
| 3 | Feature Scaling and  Dummification | 29/01/24 |  |
| 4 | Hypothesis Testing | 05/02/24 |  |
| 5 | ANOVA (Analysis of Variance) | 12/02/24 |  |
| 6 | Regression and its Types. | 19/02/24 |  |
| 7 | Logistic  Regression and Decision Tree | 26/02/24 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | K-Means clustering | 04/03/24 |  |
| 9 | Principal Component Analysis (PCA) | 11/03/24 |  |

**PRACTICAL 1**

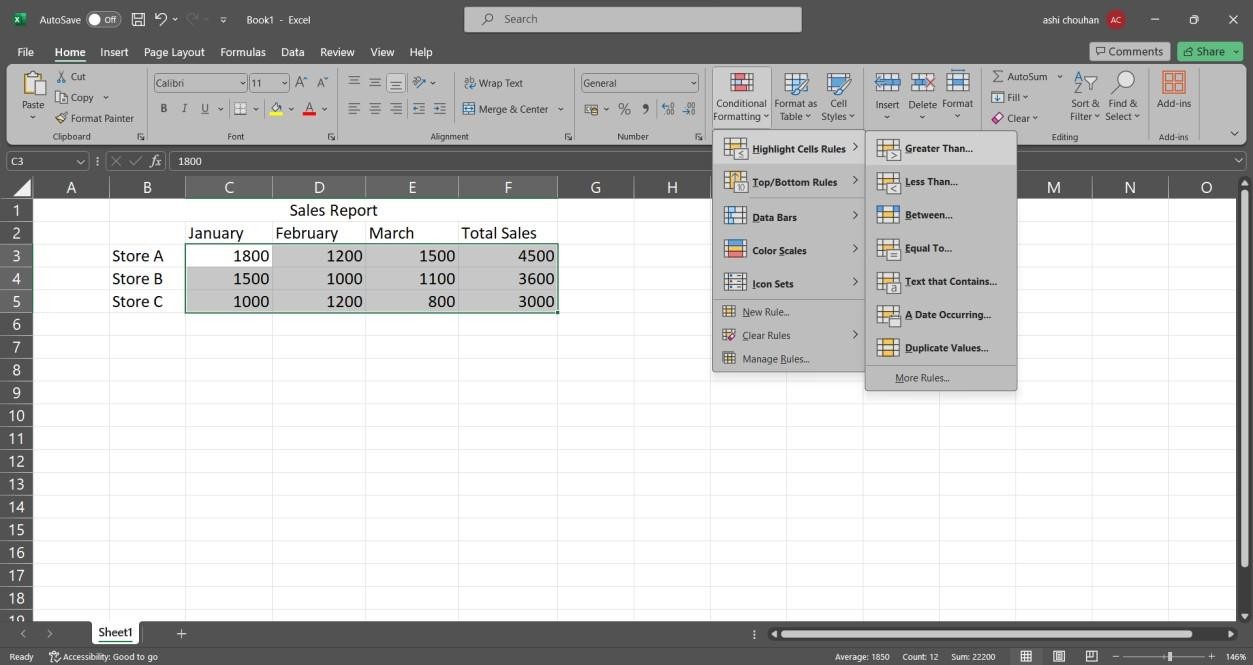
Introduction to Excel

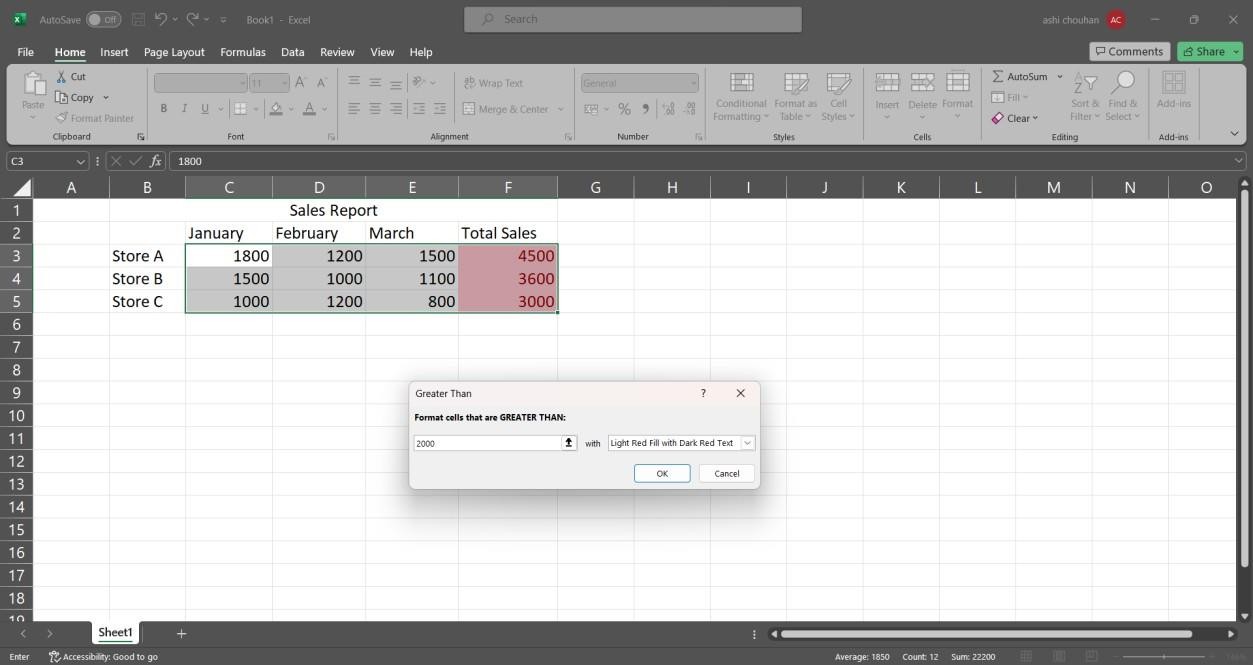
1. Perform conditional formatting on a dataset using various criteria.



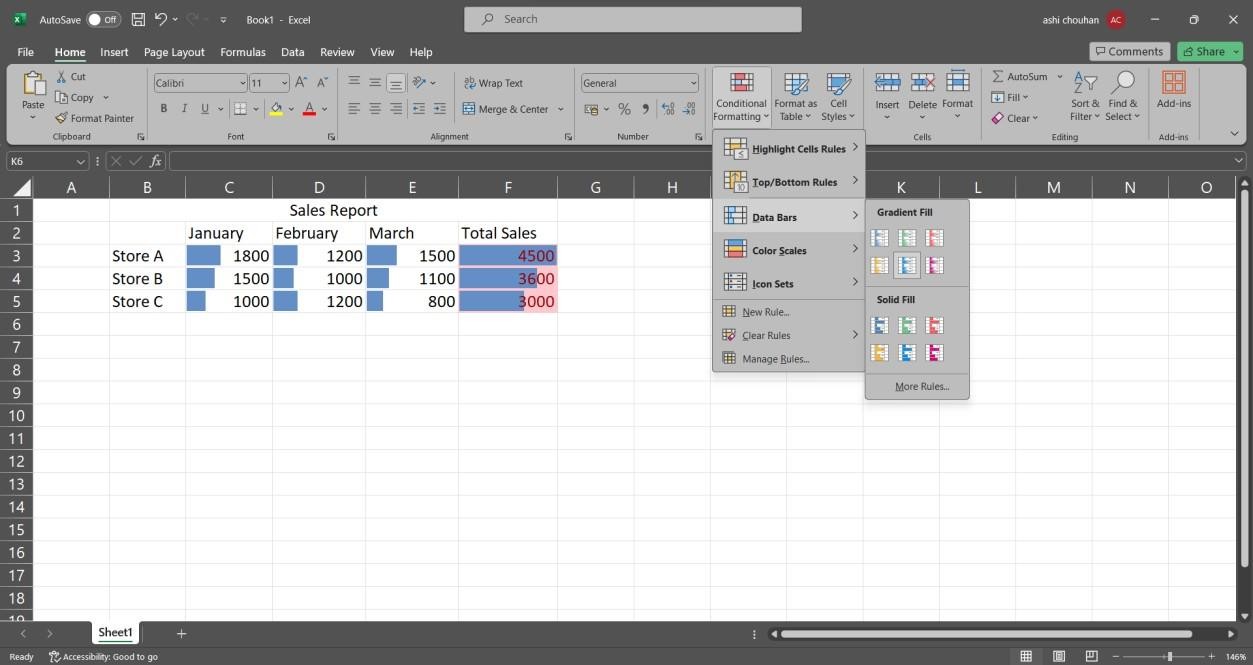
Steps

Step 1: Go to conditional formatting > Greater Than



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

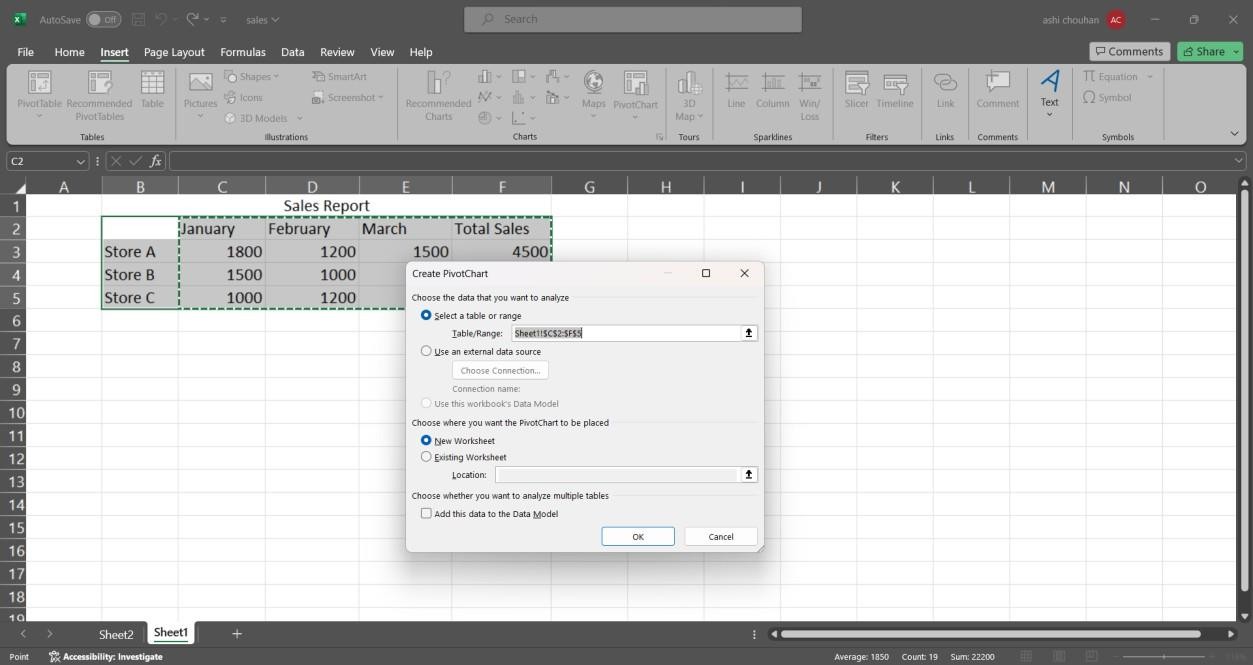
Step 3: Go to Data Bars > Solid Fill in conditional formatting.

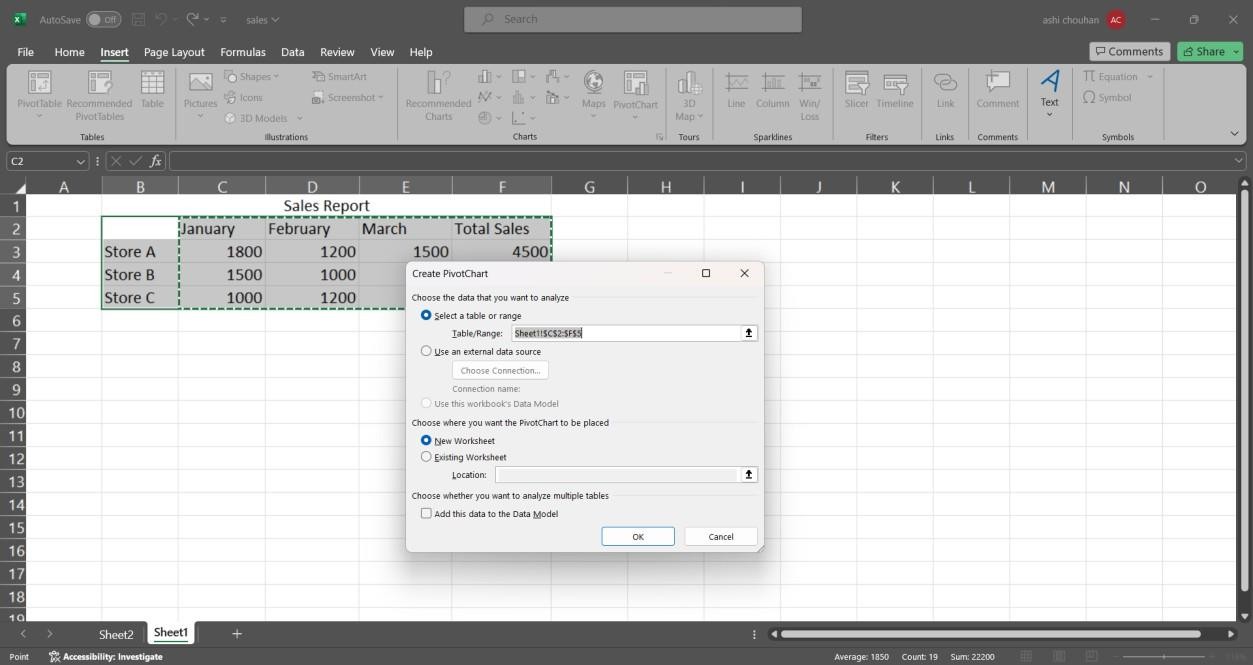


1. Create a pivot table to analyse and summarize data.

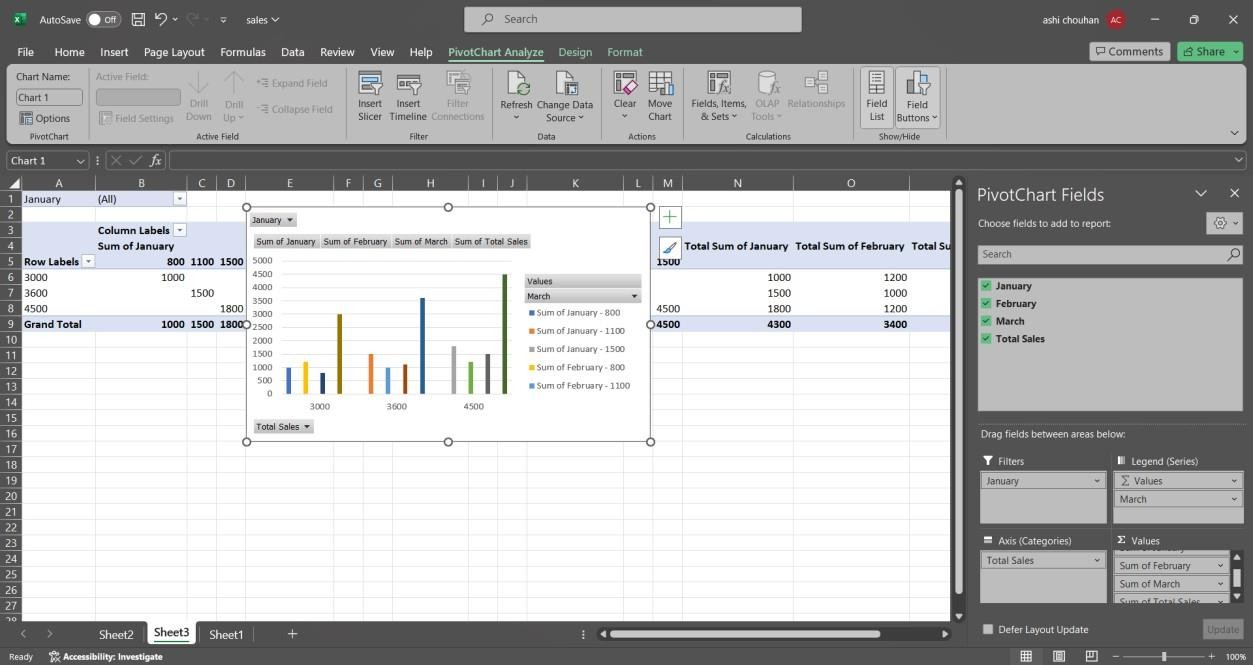
Steps

Step 1: select the entire table and go to Insert tab PivotChart > Pivotchart Step 2: Select “New worksheet” in the create pivot chart window.





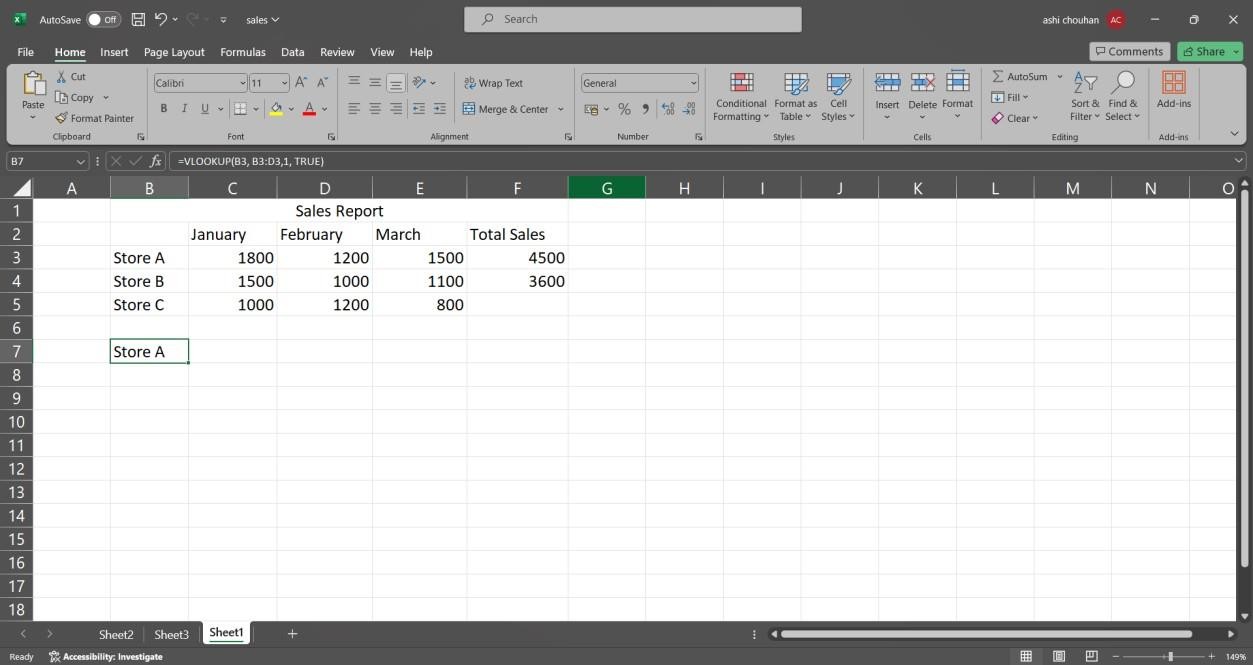
Step 3: Select and drag attributes in the below boxes.



1. 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.

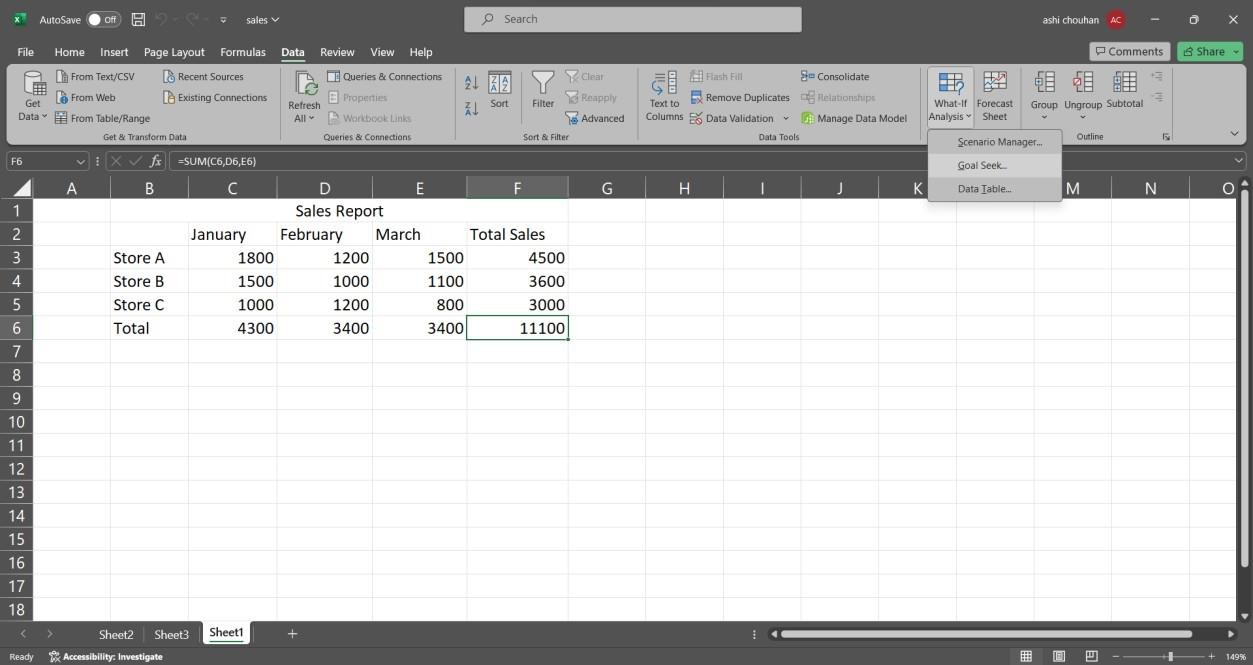
=VLOOKUP(B3, B3:D3,1, TRUE)



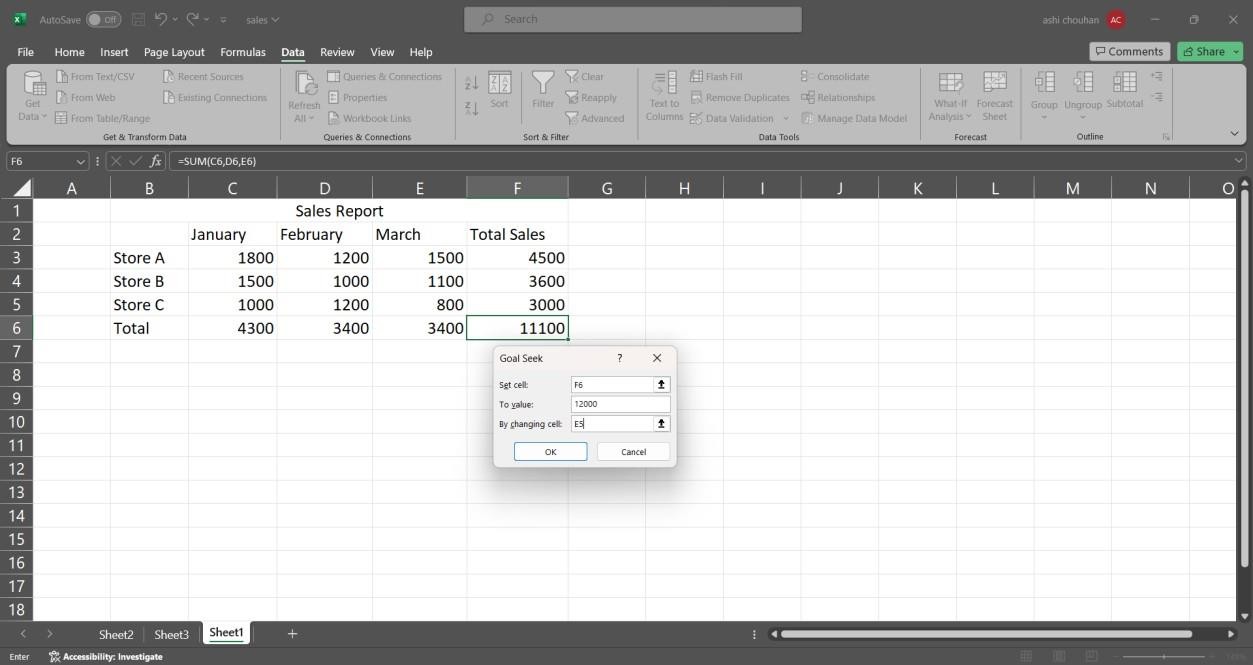
1. Perform what-if analysis using Goal Seek to determine input values for desiredoutput.

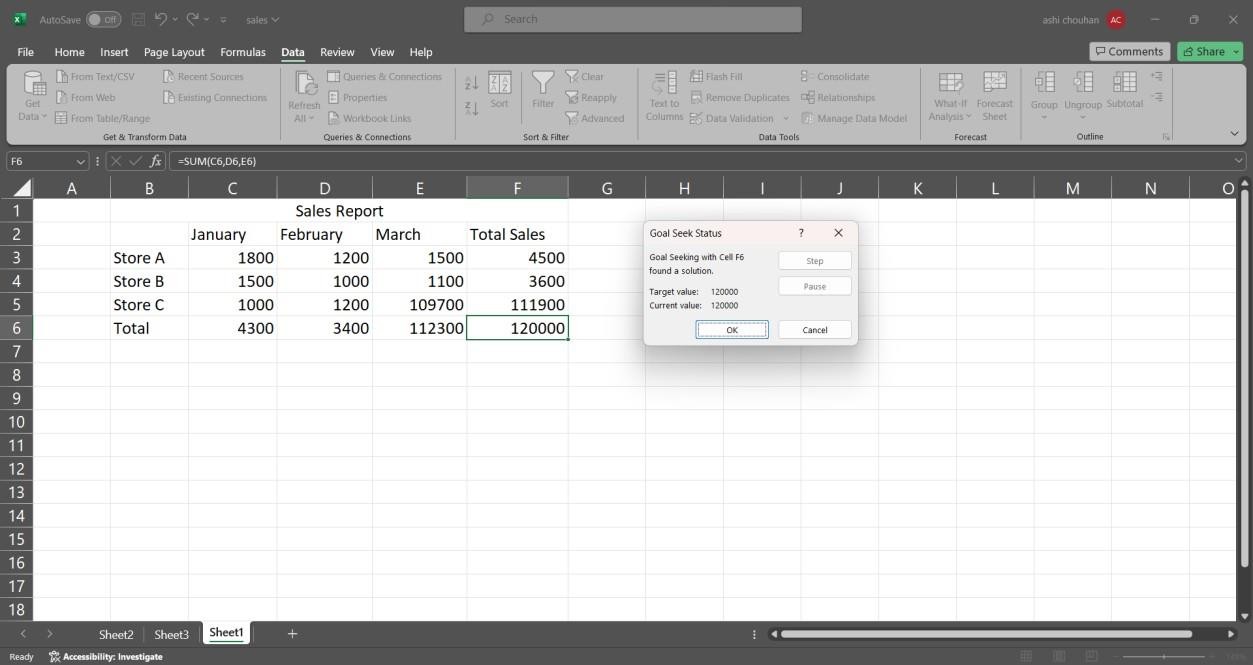
Steps

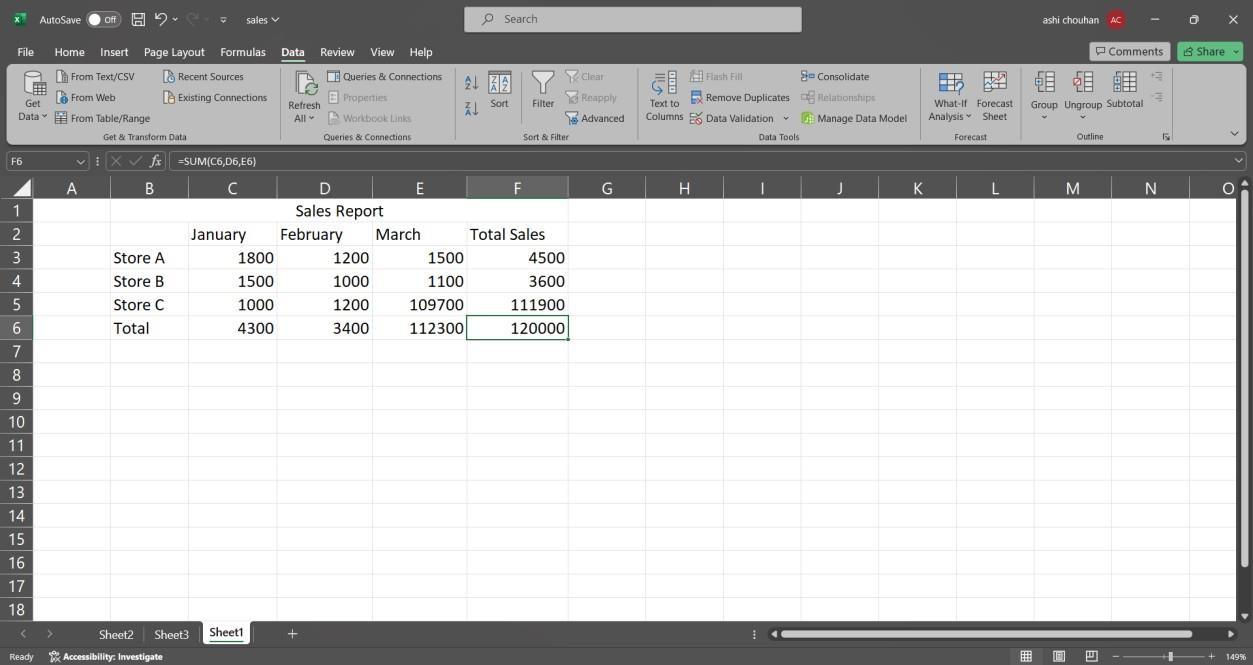
Step 1: In the Data tab go to the what if analysis>Goal seek.



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







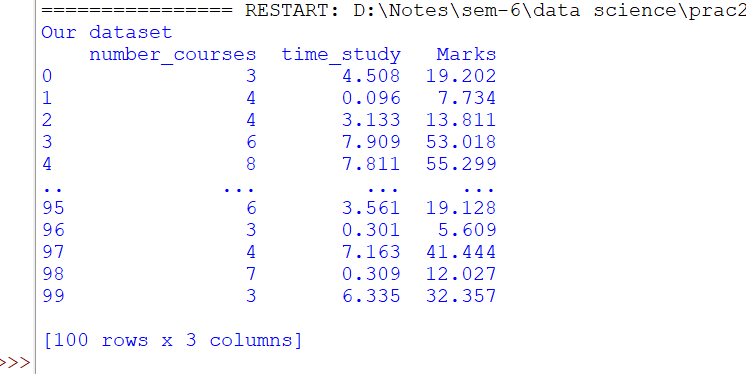
# PRACTICAL 2

Data Frames and Basic Data Pre-processing

* 1. Read data from CSV and JSON files into a data frame. (1)

# Read data from a csv file import pandas as pd df = pd.read\_csv('Student\_Marks.csv') print("Our dataset ")

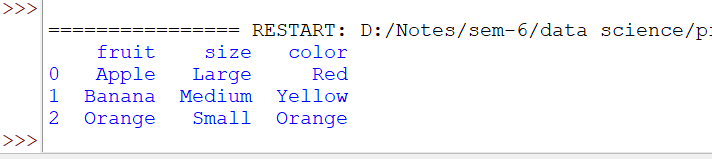
print(df)



(2)

# Reading data from a JSON file import pandas as pd

data = pd.read\_json('dataset.json') print(data)



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

Code:

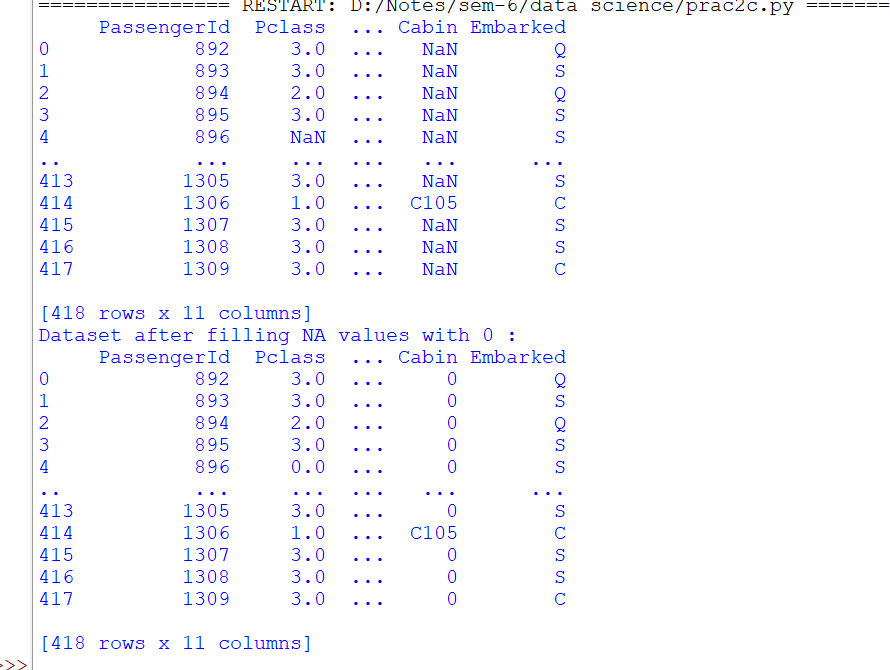
(1)

# 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)



(2)

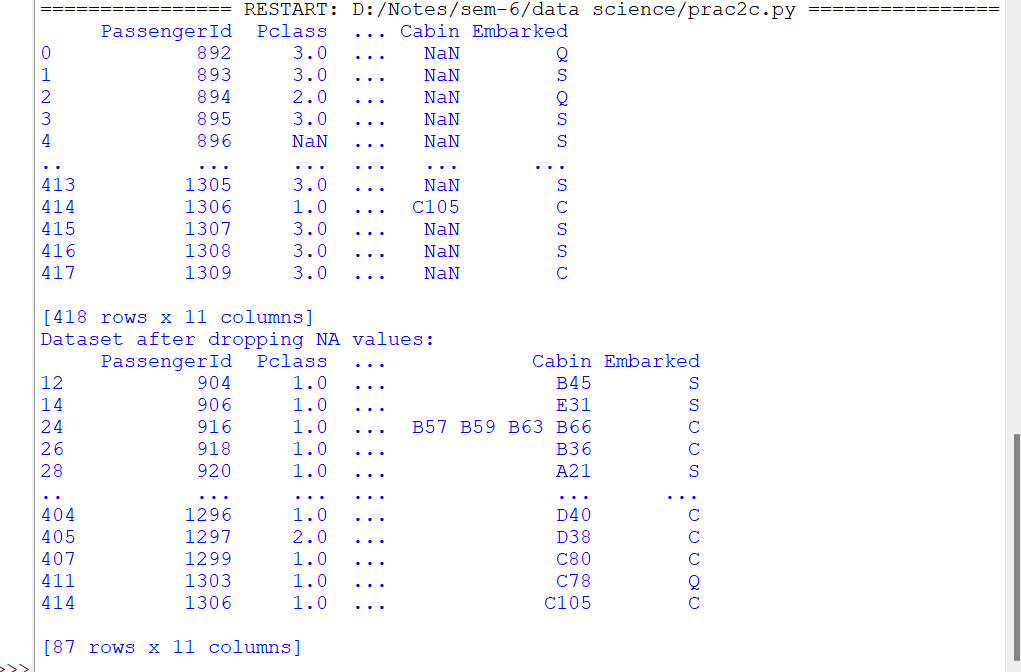
# Dropping Na values using dropna() import pandas as pd

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

df.head(10)

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

print(df)



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

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())

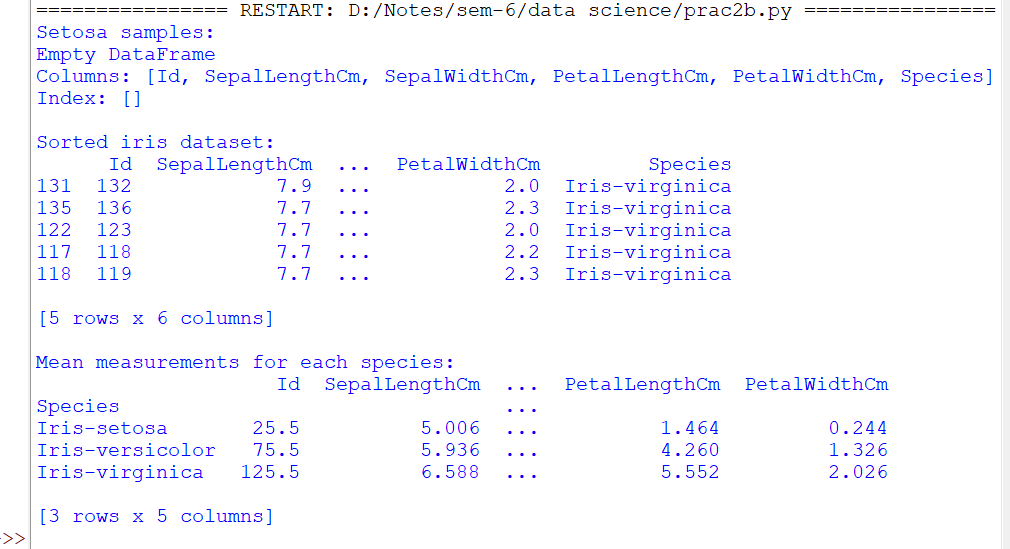
# Sorting data

sorted\_iris = iris.sort\_values(by='SepalLengthCm', ascending=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)



# PRACTICAL 3

Feature Scaling and Dummification

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

Code:

# Standardization and normalization import pandas as pd

import matplotlib.pyplot as plt from sklearn.preprocessing

import MinMaxScaler, StandardScaler

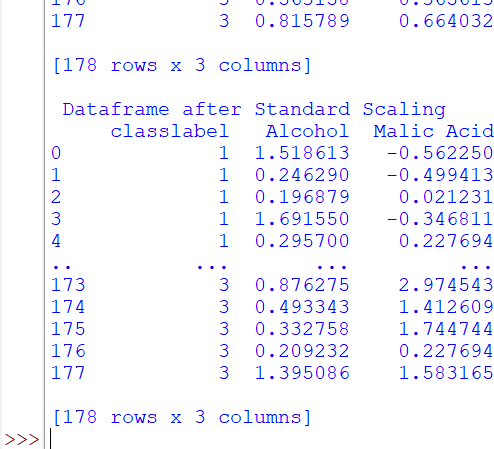
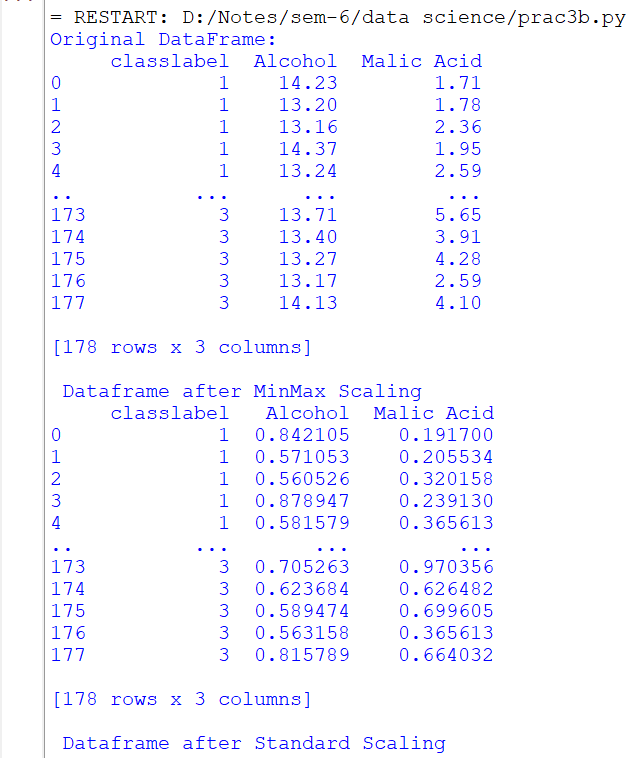
df = pd.read\_csv('wine.csv', header=None, usecols=[0, 1, 2], skiprows=1) df.columns = ['classlabel', 'Alcohol', 'Malic Acid'] print("Original DataFrame:")

print(df) scaling=MinMaxScaler() scaled\_value=scaling.fit\_transform(df[['Alcohol','Malic Acid']])

df[['Alcohol','Malic Acid']]=scaled\_value print("\n Dataframe after MinMax Scaling") print(df)

scaling=StandardScaler() scaled\_standardvalue=scaling.fit\_transform(df[['Alcohol','Mali c Acid']]) df[['Alcohol','Malic Acid']]=scaled\_standardvalue print("\n Dataframe after Standard Scaling")

print(df)

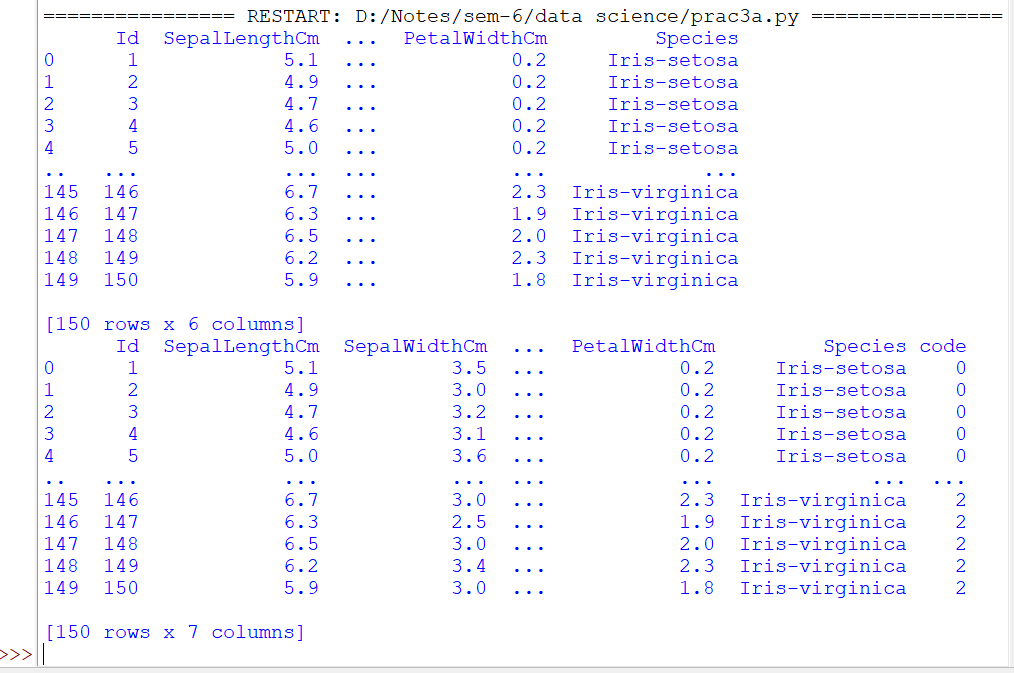


* + 1. Perform feature Dummification to convert categorical variables into numerical representations.

Code:

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

from sklearn.preprocessing import LabelEncoder le=LabelEncoder() iris['code']=le.fit\_transform(iris.Species) print(iris)



# PRACTICAL 4

Hypothesis Testing

Conduct a hypothesis test using appropriate statistical tests (e.g., t- test, chi-square test)

# t-test

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.")

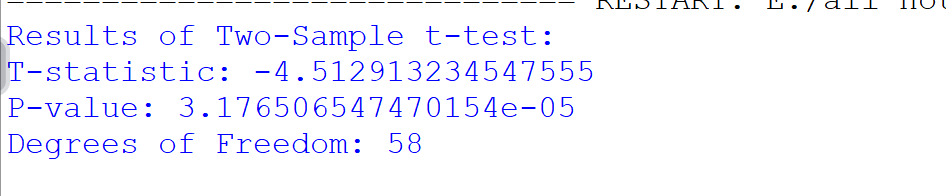
print("Interpretation: The mean of Sample 2 is significantly higher than that of Sample 1.")

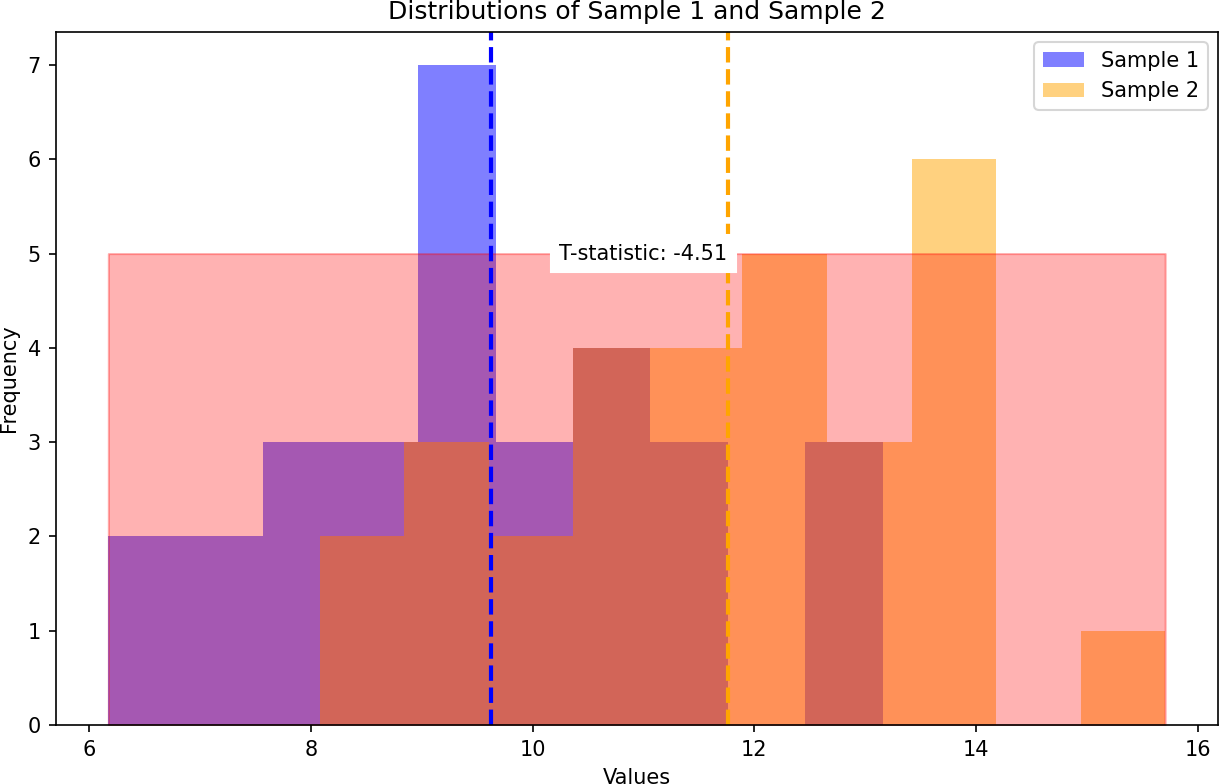
else:

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

significant difference between the means.")

Output:





#chi-test

import pandas as pd import numpy as np import matplotlib as plt import seaborn as sb import warnings

from scipy import stats warnings.filterwarnings('ignore') df=sb.load\_dataset('mpg') print(df) print(df['horsepower'].describe()) print(df['model\_year'].describe())

bins=[0,75,150,240]

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

c=df['horsepower\_new'] print(c) ybins=[69,72,74,84]

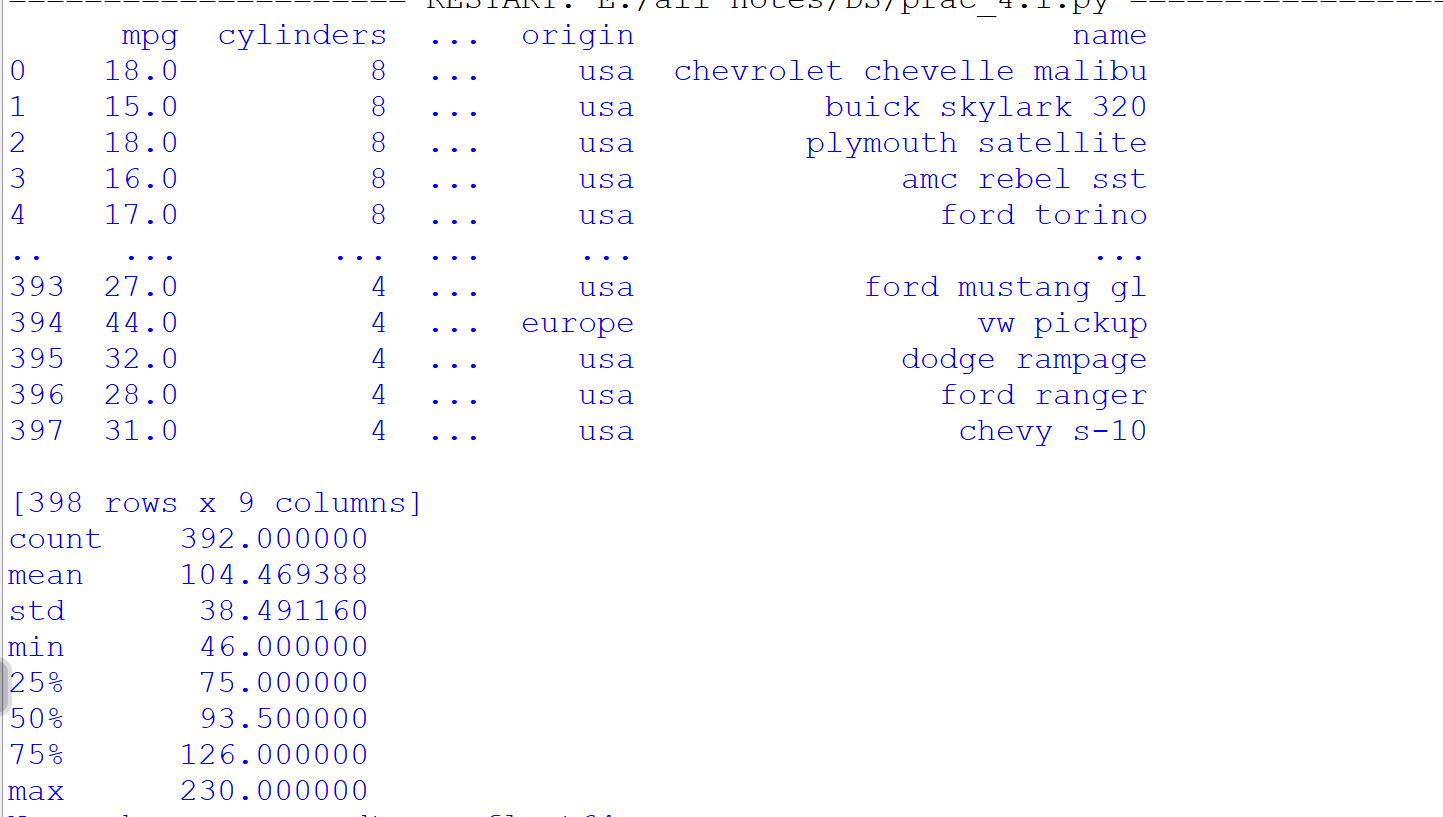
label=['t1','t2','t3'] df['modelyear\_new']=pd.cut(df['model\_year'],bins=ybins,labels=lab el)

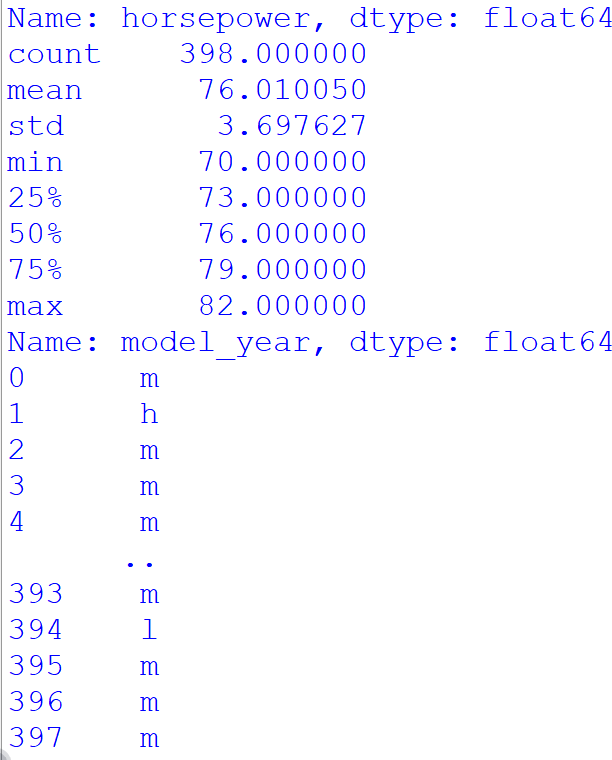
newyear=df['modelyear\_new'] print(newyear)

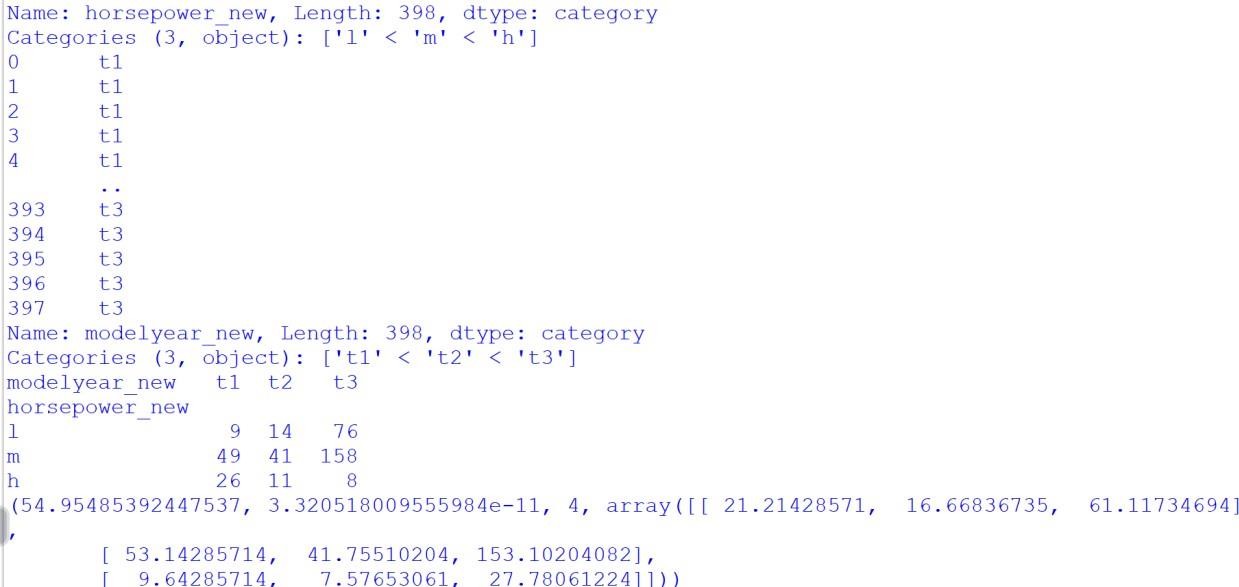
df\_chi=pd.crosstab(df['horsepower\_new'],df['modelyear\_new']) print(df\_chi)

print(stats.chi2\_contingency(df\_chi)

Output:







Conclusion: There is sufficient evidence to reject the null hypothesis, indicating that there is a significant association between 'horsepower\_new' and 'modelyear\_new' categories.

# PRACTICAL 5

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.

import pandas as pd import scipy.stats as stats

from statsmodels.stats.multicomp import pairwise\_tukeyhsd

group1 = [23, 25, 29, 34, 30]

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

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

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

all\_data = group1 + group2 + group3 + group4

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

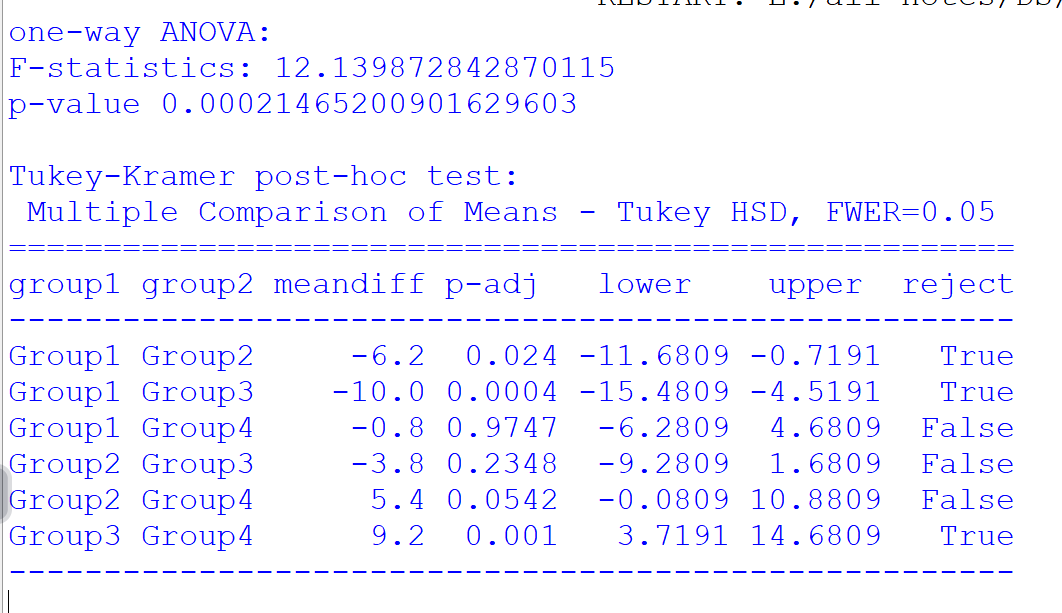
+ ['Group3'] \* len(group3) + ['Group4'] \* len(group4)

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

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

tukey\_results = pairwise\_tukeyhsd(all\_data, group\_labels) print("\nTukey-Kramer post-hoc test:") print(tukey\_results)

Output:



# PRACTICAL 6

Regression and its Types.

import numpy as np import pandas as pd

from sklearn.datasets import fetch\_california\_housing from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

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

housing = fetch\_california\_housing() housing\_df =

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

housing\_df['PRICE'] = housing.target

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

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

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

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\_)

#########################################

#Multiple Liner Regression

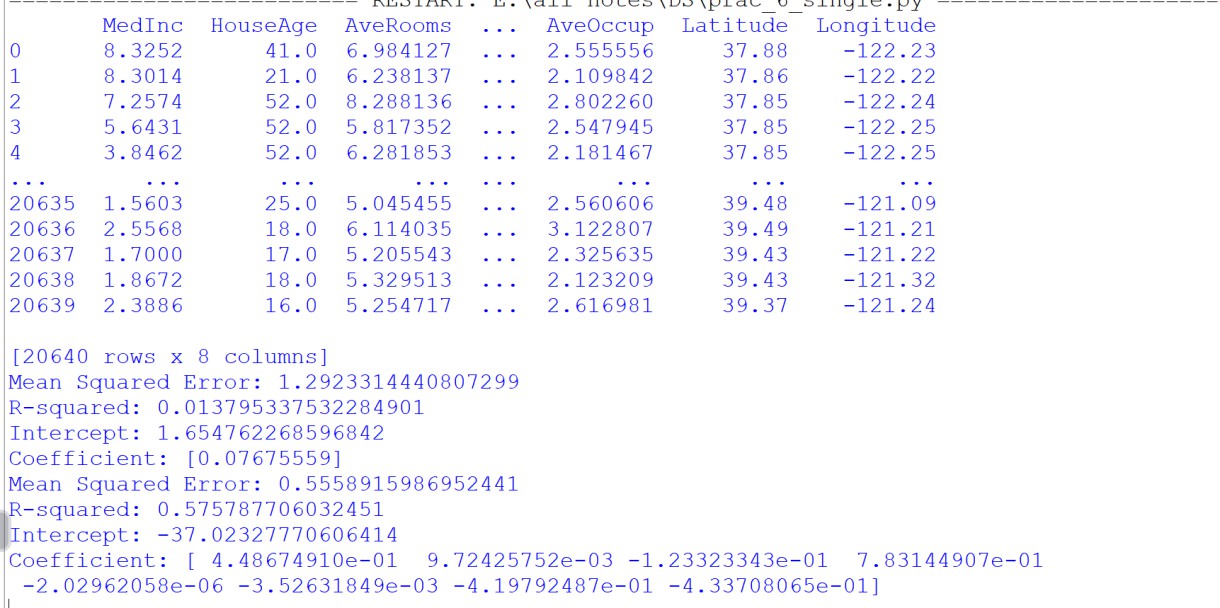
X = housing\_df.drop('PRICE',axis=1) y = housing\_df['PRICE']

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

model = LinearRegression() model.fit(X\_train,y\_train) y\_pred = model.predict(X\_test)

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

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

Output:

# PRACTICAL 7

**Logistic Regression and Decision Tree**

import numpy as np import pandas as pd

from sklearn.datasets import load\_iris

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

# Load the Iris dataset and create a binary 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("Logistic 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))

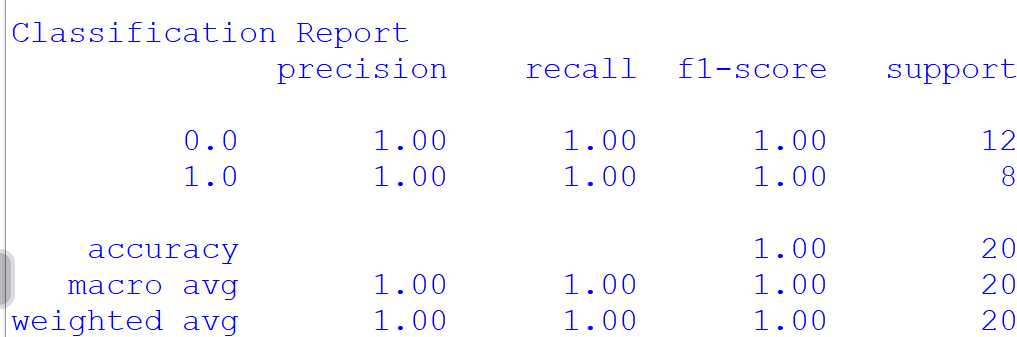
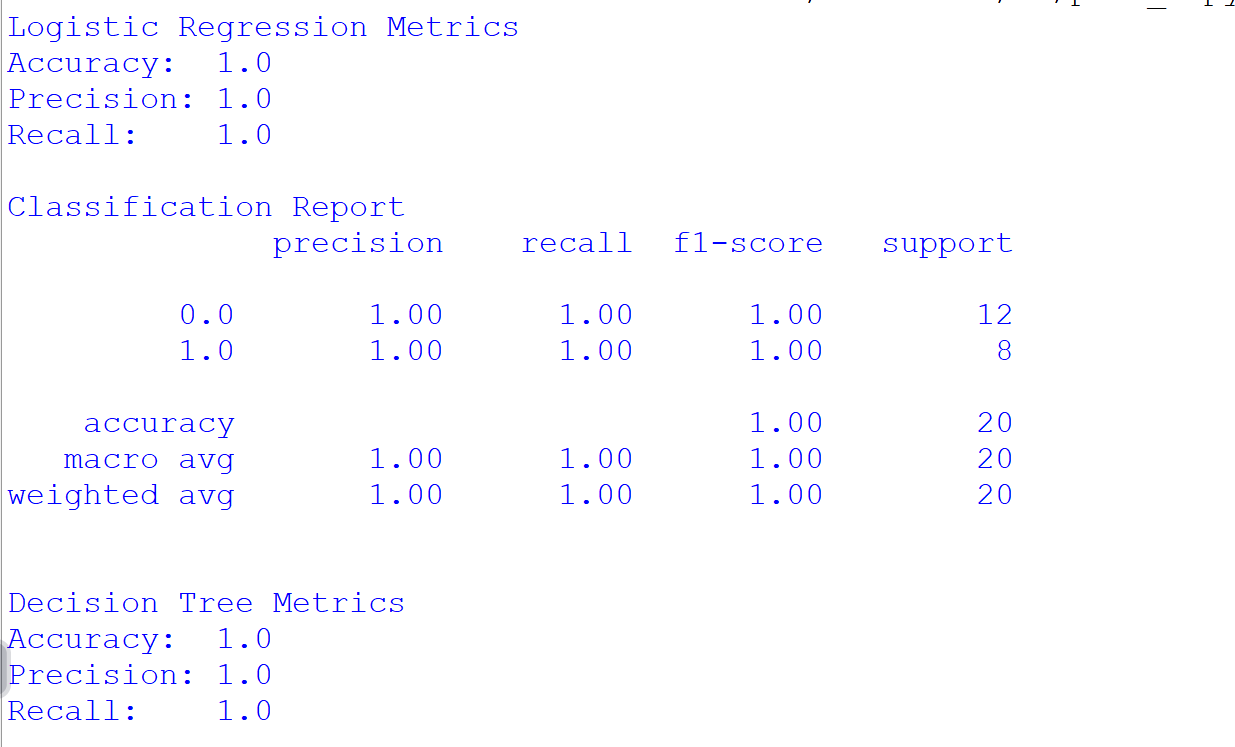
print("\nClassification Report") print(classification\_report(y\_test, y\_pred\_logistic))

# 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:



import pandas as pd

# PRACTICAL 8

K-Means clustering

from sklearn.preprocessing

import MinMaxScaler from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

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

categorical\_features = ['Channel', 'Region']

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

data[continuous\_features].describe()

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()

mms = MinMaxScaler() mms.fit(data)

data\_transformed = mms.transform(data)

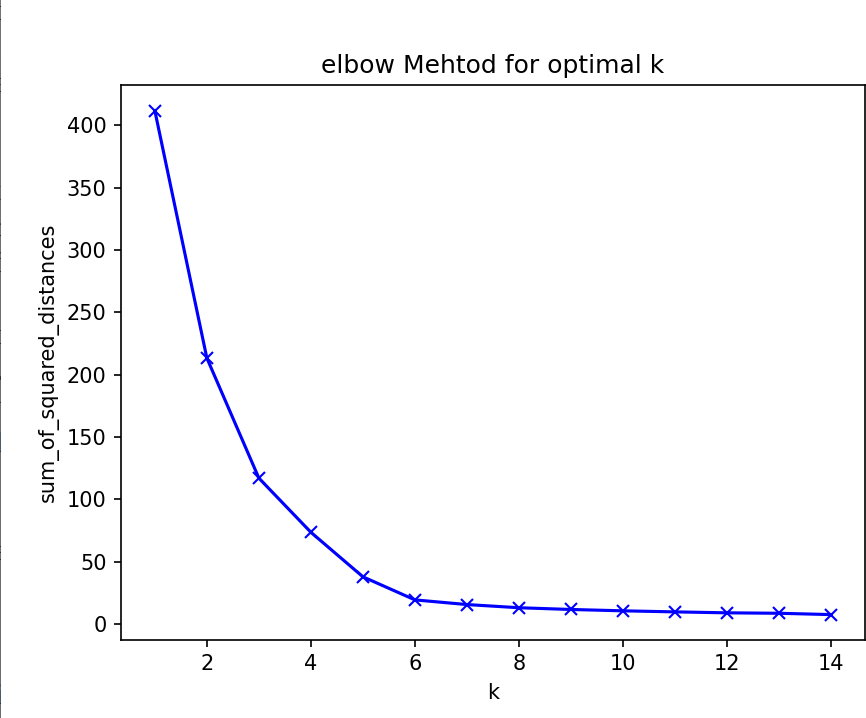
sum\_of\_squared\_distances = [] K = range(1, 15)

for k in K:

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

sum\_of\_squared\_distances.append(km.inertia\_) plt.plot(K, sum\_of\_squared\_distances, 'bx-') plt.xlabel('k') plt.ylabel('sum\_of\_squared\_distances') plt.title('elbow Mehtod for optimal k') plt.show()

Output:



# PRACTICAL 9

Principal Component Analysis (PCA)

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

iris = load\_iris()

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

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

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

pca = PCA()

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

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()

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}")

pca = PCA(n\_components=n\_components) X\_reduced = pca.fit\_transform(X\_scaled)

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:

