1**Code:-**

**A)**

import pandas as pd

mtcars = pd.read\_csv('prac1/mtcars.csv')

# Display basic information about the mtcars dataset

print("Summary for mtcars dataset:")

print(mtcars.info())

print("\n")

#Display summary statistics

print("Summary Statistics for mtcars dataset:")

print(mtcars.describe())

# Display structure information

print("\nStructure Information for mtcars dataset:")

print(mtcars.info())

#Display quartile information for numeric columns

numeric\_columns = mtcars.select\_dtypes(include=['number']).columns

print("\nQuartile Information for mtcars dataset:")

print(mtcars[numeric\_columns].quantile([0.25, 0.5, 0.75]))

cars = pd.read\_csv('prac1/cars.csv')

# Display basic information about the cars dataset

print("Summary for cars dataset:")

print(cars.info())

print("\n")

#Display summary statistics

print("Summary Statistics for cars dataset:")

print(cars.describe())

# Display structure information

print("\nStructure Information for cars dataset:")

print(cars.info())

#Display quartile information for numeric columns

numeric\_columns = cars.select\_dtypes(include=['number']).columns

print("\nQuartile Information for cars dataset:")

print(cars[numeric\_columns].quantile([0.25, 0.5, 0.75]))

**b)**

import pandas as pd

# Load the Iris dataset from CSV file

iris\_df = pd.read\_csv('prac1/iris.csv')

# Display a subset of the dataset using subset() function

subset\_condition = (iris\_df['sepal.length'] > 5) & (iris\_df['sepal.width'] > 3)

subset\_result = iris\_df[subset\_condition]

print("Subset of the Iris dataset using subset() function:")

print(subset\_result)

print("\n")

# Aggregate the dataset using aggregate() function

aggregation\_result = iris\_df.groupby('variety').aggregate({

'sepal.length': 'mean',

'sepal.width': 'std',

'petal.length': 'mean',

'petal.width': 'std'

})

aggregation\_result.rename(columns={

'sepal.length': 'mean\_sepal\_length',

'sepal.width': 'std\_sepal\_width',

'petal.length': 'mean\_petal\_length',

'petal.width': 'std\_petal\_width'

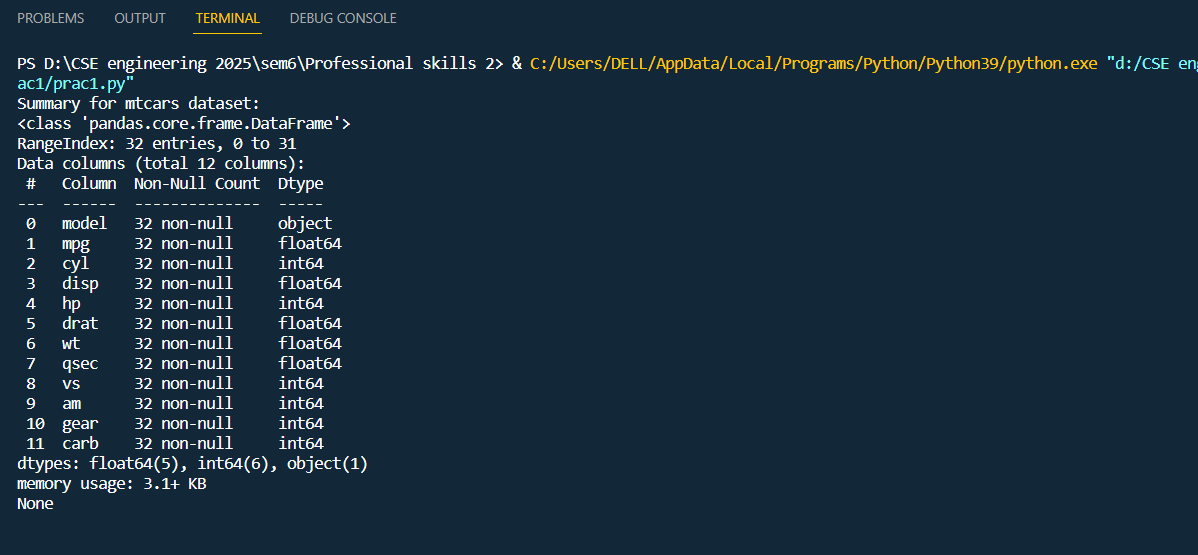
}, inplace=True)

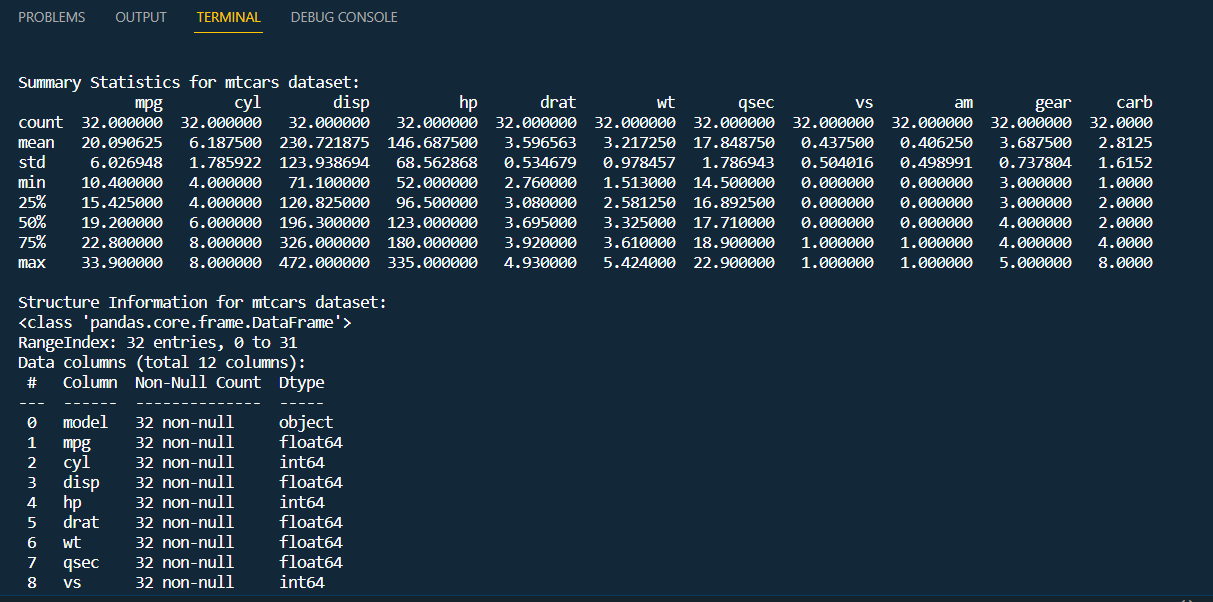
print("Aggregation of the Iris dataset using aggregate() function:")

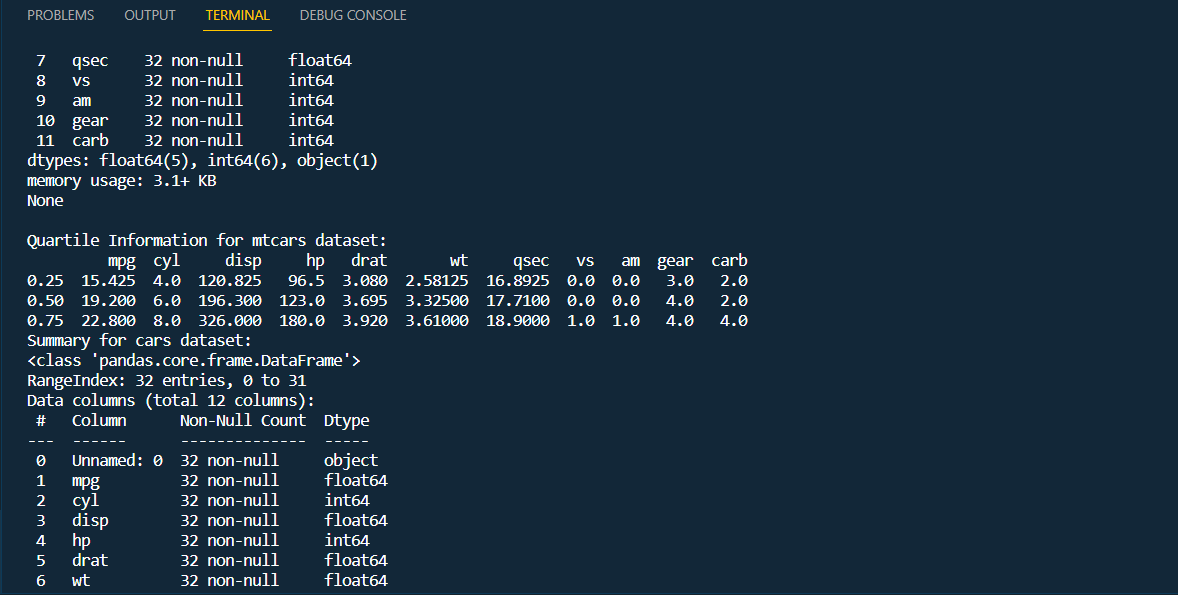
print(aggregation\_result)

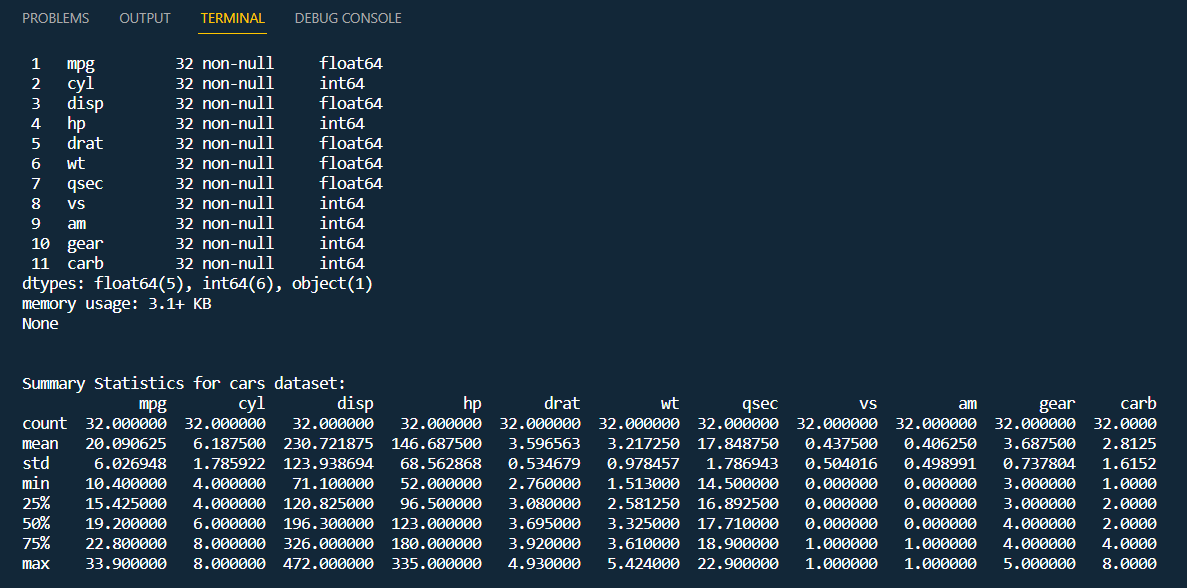
1**Output:-**

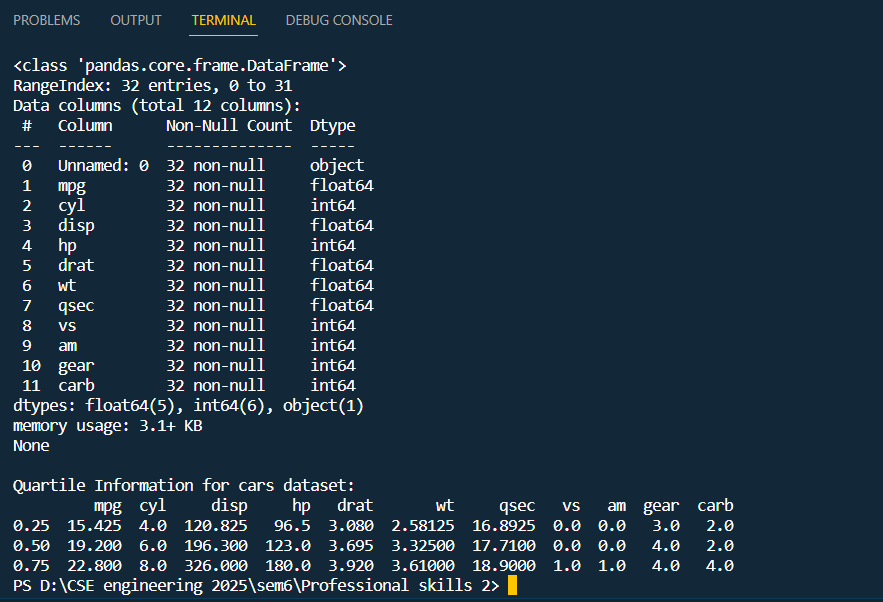
**a)**



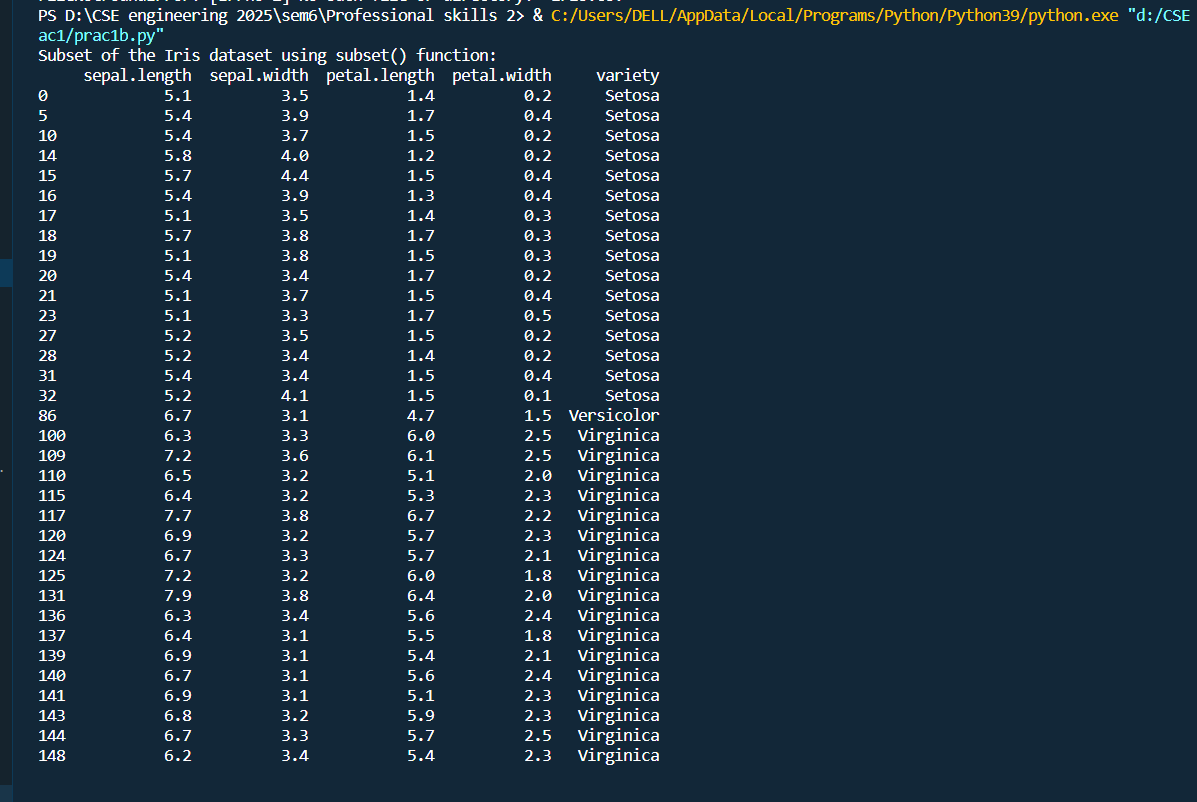


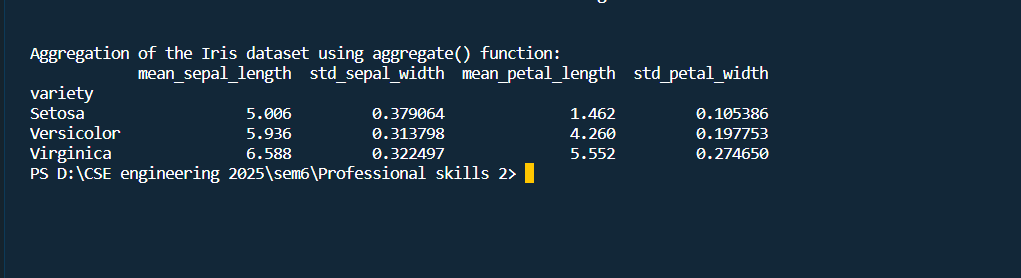






**b)**

****

****

2**Code:-**

import pandas as pd

# Read a CSV file from the web

url\_csv = 'https://gist.githubusercontent.com/rnirmal/e01acfdaf54a6f9b24e91ba4cae63518/raw/6b589a5c5a851711e20c5eb28f9d54742d1fe2dc/datasets.csv'

df\_csv\_web = pd.read\_csv(url\_csv)

# Read a TXT file from the web

url\_txt = 'https://raw.githubusercontent.com/selva86/datasets/master/sample.txt'

df\_txt\_web = pd.read\_csv(url\_txt, delimiter='\t') # Assuming tab-separated data

# Read a CSV file from disk

file\_csv\_disk = 'prac2/iris.csv'

df\_csv\_disk = pd.read\_csv(file\_csv\_disk)

# Read a TXT file from disk

file\_txt\_disk = 'prac2/sample.txt'

df\_txt\_disk = pd.read\_csv(file\_txt\_disk, delimiter='\t') # Assuming tab-separated data

# Concatenate the dataframes

combined\_df = pd.concat([df\_csv\_web, df\_txt\_web, df\_csv\_disk, df\_txt\_disk], ignore\_index=True)

# Display the shape (number of rows and columns) of the combined dataframe

print("Shape of the combined dataframe:", combined\_df.shape)

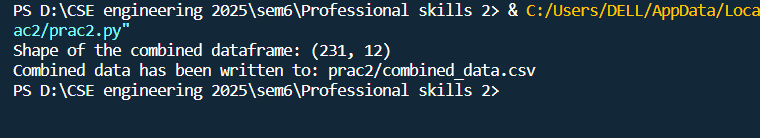
# Write the combined dataframe to a file on disk

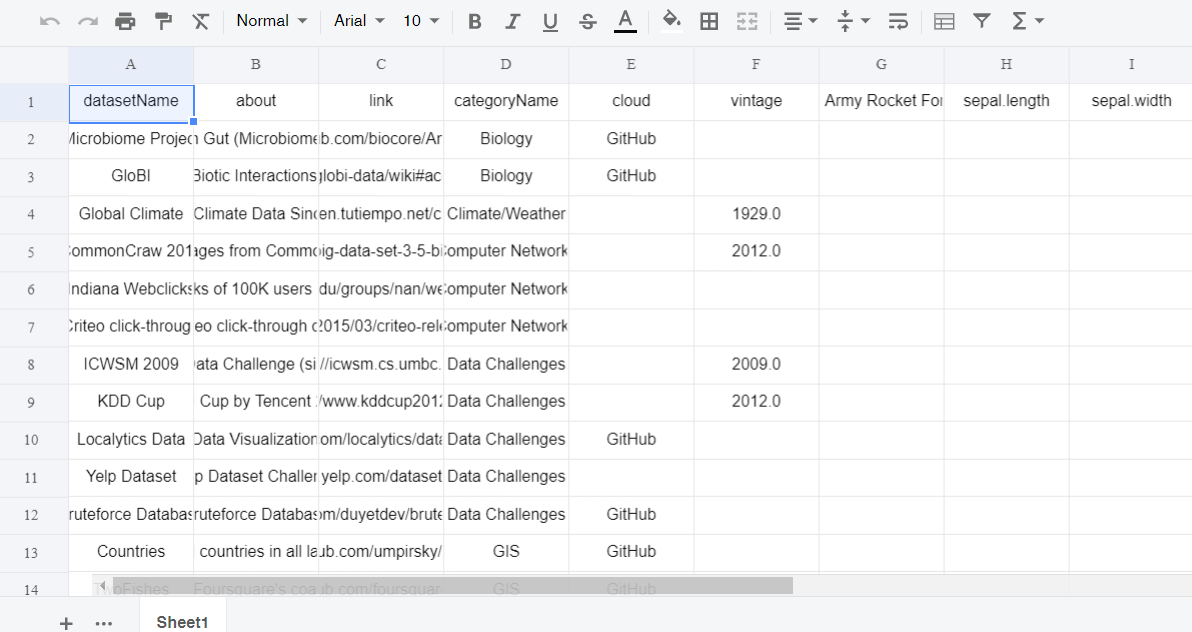
output\_file = 'prac2/combined\_data.csv'

combined\_df.to\_csv(output\_file, index=False)

print("Combined data has been written to:", output\_file)

2**Output:-**

****

****

2**Code:-**

import pandas as pd

# Specify the path to your Excel file

excel\_file\_path = 'prac2/airline.xlsx'

# Read the Excel file into a DataFrame

df\_excel = pd.read\_excel(excel\_file\_path)

# Display the DataFrame

print(df\_excel)

2**Output:-**

****

3**Code:-**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the Iris dataset

iris\_df = pd.read\_csv("prac3/iris.csv")

# Display the first few rows of the dataset

print(iris\_df.head())

# a. Box and scatter plots for data distributions

# Box plot

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

sns.boxplot(data=iris\_df)

plt.title("Box Plot of Iris Dataset")

plt.xlabel("Features")

plt.ylabel("Centimeters")

plt.xticks(rotation=45)

plt.show()

# Scatter plot

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

sns.scatterplot(data=iris\_df, x='sepal.length', y='sepal.width', hue='variety')

plt.title("Scatter Plot of Sepal Length vs Sepal Width")

plt.xlabel("Sepal Length (cm)")

plt.ylabel("Sepal Width (cm)")

plt.show()

# b. Finding outliers using box plot

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

sns.boxplot(data=iris\_df)

plt.title("Box Plot to Identify Outliers")

plt.xlabel("Features")

plt.ylabel("Centimeters")

plt.xticks(rotation=45)

plt.show()

# c. Histogram and pie chart

# Histogram

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

plt.hist(iris\_df['petal.length'], bins=20, edgecolor='black')

plt.title("Histogram of Petal Length")

plt.xlabel("Petal Length (cm)")

plt.ylabel("Frequency")

plt.show()

# Pie chart

variety\_counts = iris\_df['variety'].value\_counts()

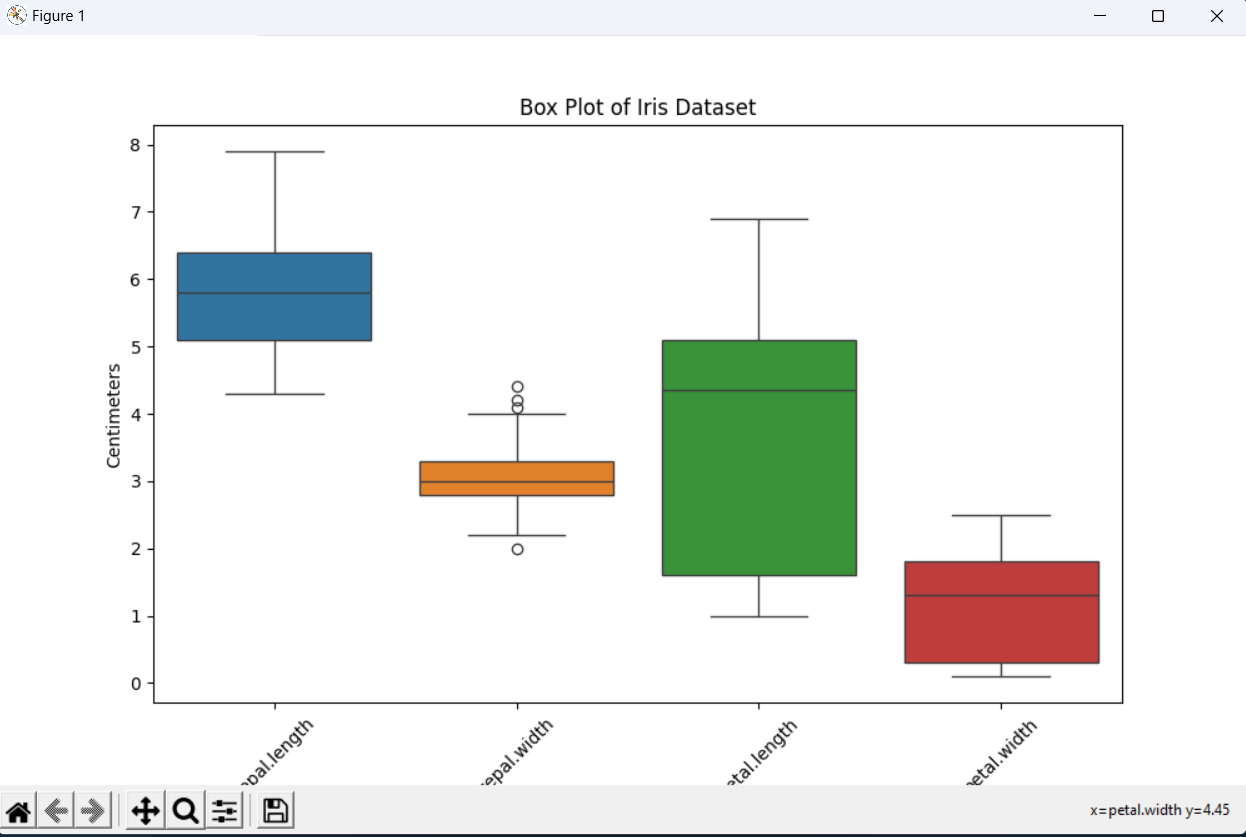
plt.figure(figsize=(8, 8))

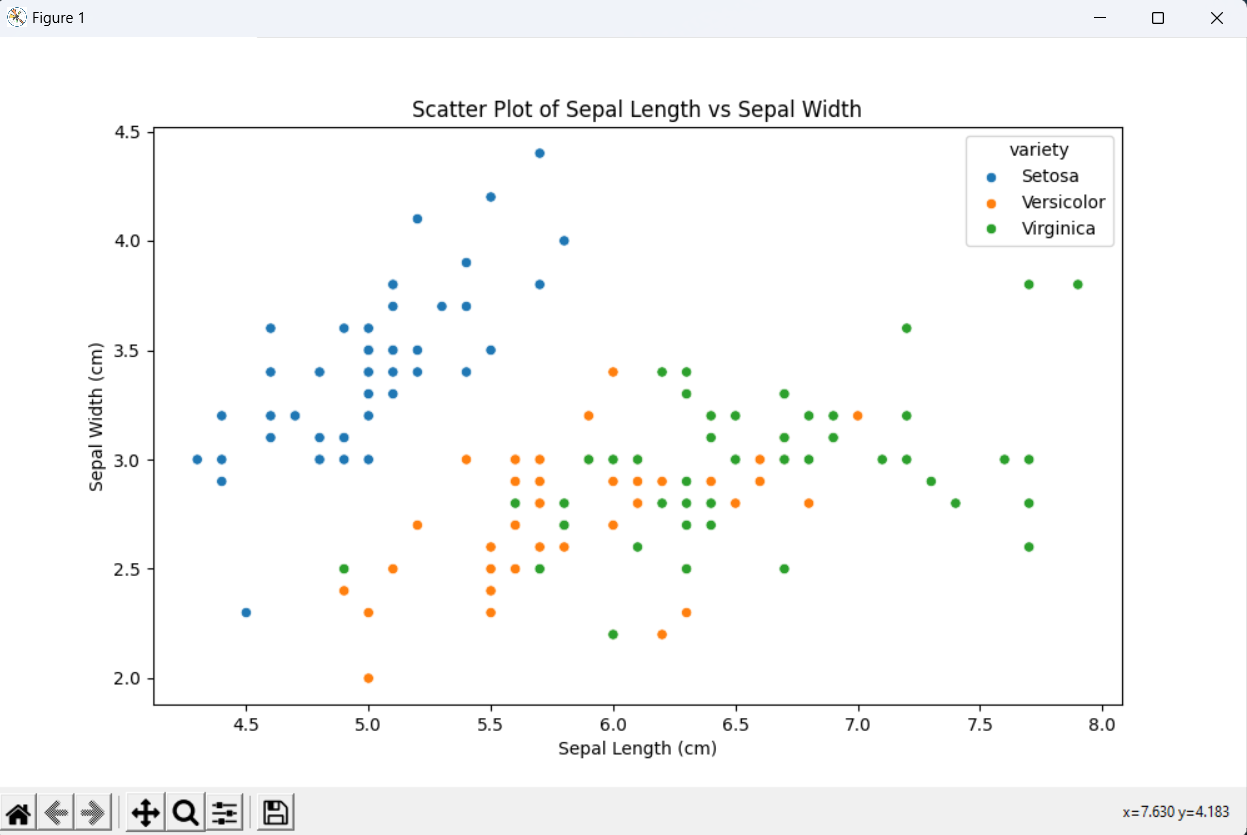
plt.pie(variety\_counts, labels=variety\_counts.index, autopct='%1.1f%%', startangle=140)

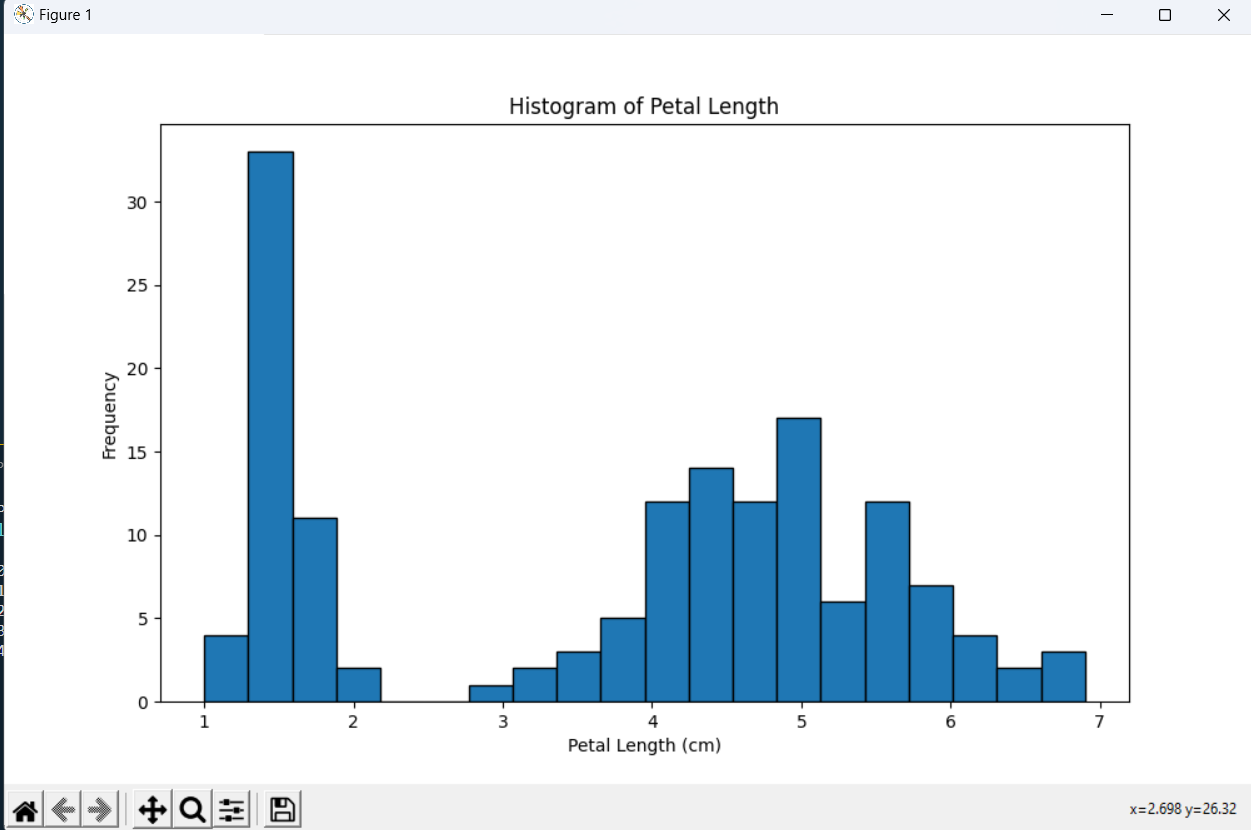
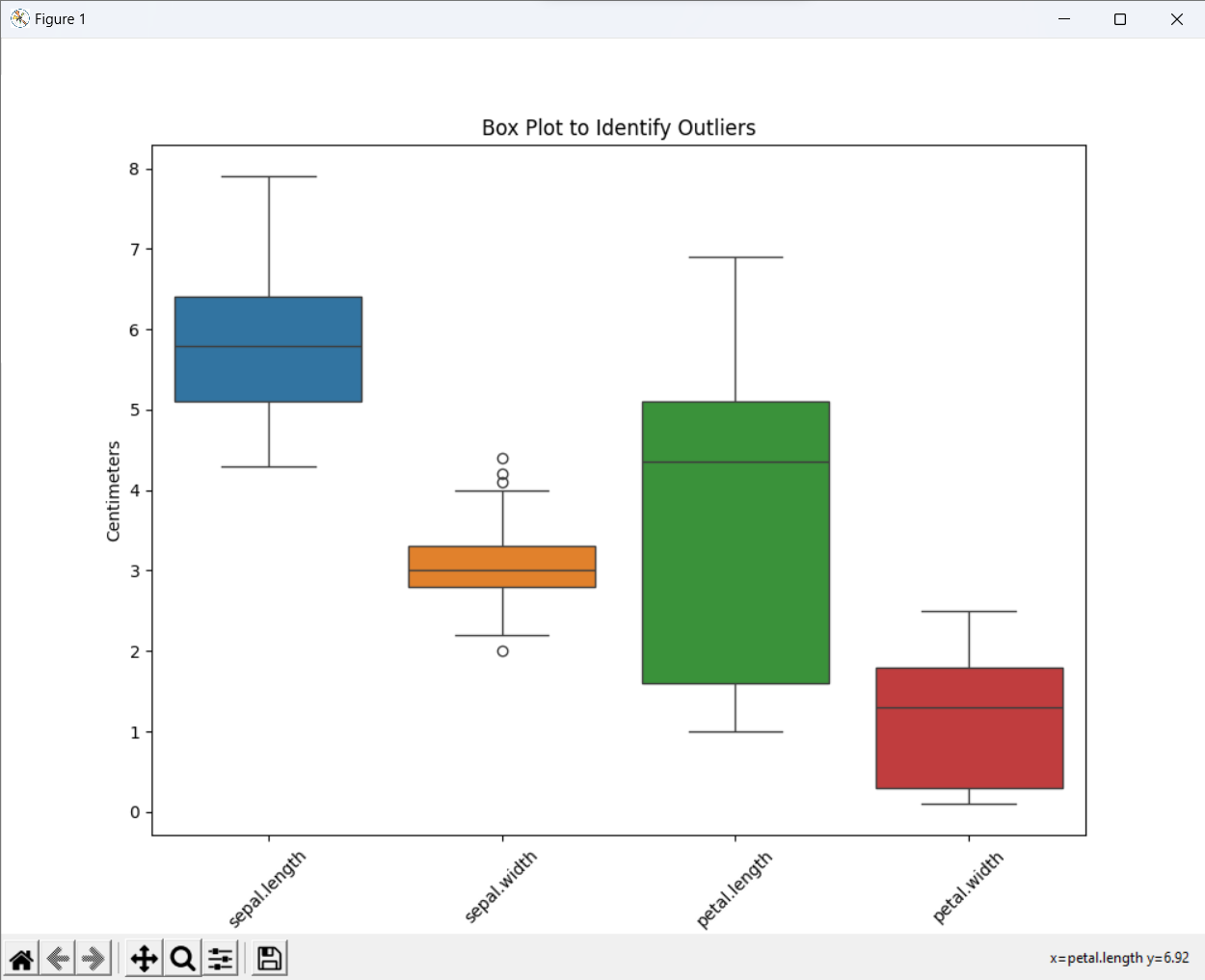
plt.title("Distribution of Iris variety")

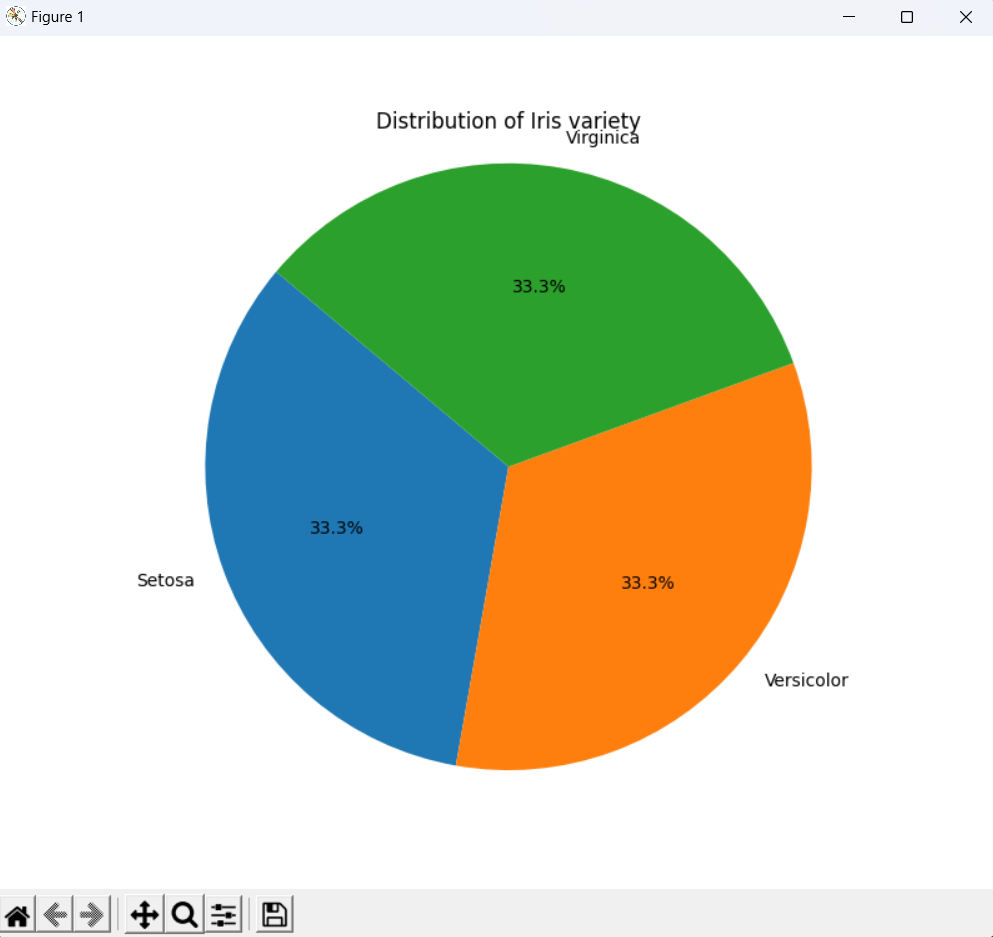
plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

****3**Output:-**

****

****

****

4**Code:-**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.stats import f\_oneway

# Load the Iris dataset

iris\_df = pd.read\_csv("prac4/iris.csv")

# Display the first few rows of the dataset

print(iris\_df.head())

# a. Correlation matrix (excluding the categorical column 'variety')

correlation\_matrix = iris\_df.drop('variety', axis=1).corr()

print("Correlation Matrix:")

print(correlation\_matrix)

# b. Correlation plot

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

sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.title("Correlation Plot of Iris Dataset")

plt.show()

# c. Analysis of covariance (ANOVA)

# Perform ANOVA on petal lengths for each variety

setosa\_petal\_length = iris\_df[iris\_df['variety'] == 'Setosa']['petal.length']

versicolor\_petal\_length = iris\_df[iris\_df['variety'] == 'Versicolor']['petal.length']

virginica\_petal\_length = iris\_df[iris\_df['variety'] == 'Virginica']['petal.length']

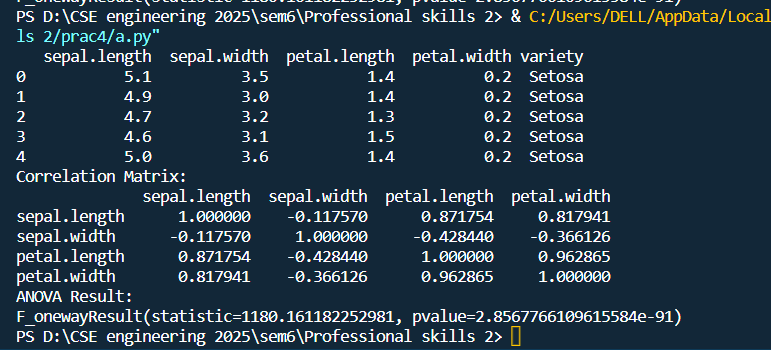
# Perform ANOVA

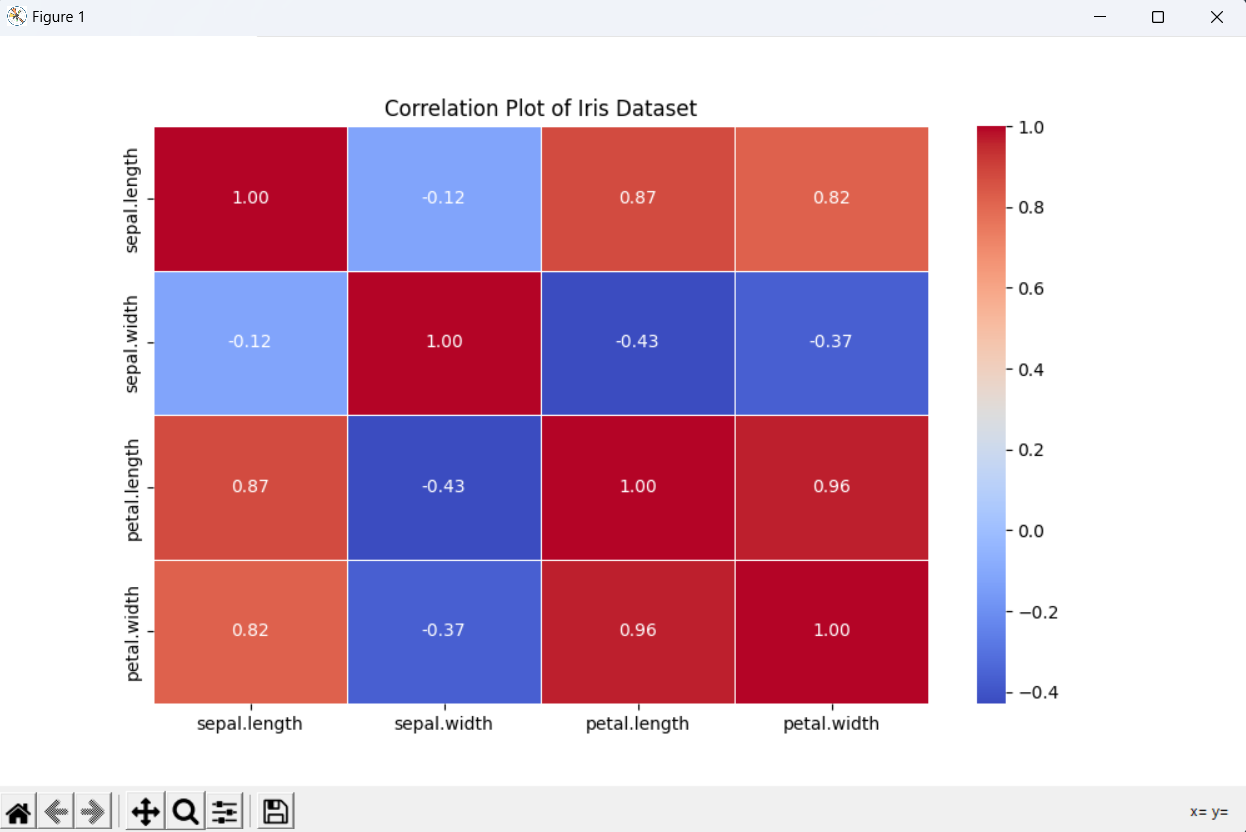
anova\_result = f\_oneway(setosa\_petal\_length, versicolor\_petal\_length, virginica\_petal\_length)

print("ANOVA Result:")

print(anova\_result)

4**Output:-**

****

****

5**Code:-**

# Importing required libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

import statsmodels.api as sm

# Load your dataset

df = pd.read\_csv('https://raw.githubusercontent.com/Opensourcefordatascience/Data-sets/master/admission.csv')

# Name your dataset

df.name = "Admissions\_Dataset"

# Checking the first few rows of the dataset

print(df.head())

# Separating features and target variable

X = df[['gre', 'gpa', 'rank']]

y = df['admit']

# Adding constant term for logistic regression

X = sm.add\_constant(X)

# Splitting the dataset into training and testing sets

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

# Fitting logistic regression model

log\_reg = sm.Logit(y\_train, X\_train)

result = log\_reg.fit()

# Checking model summary

print(result.summary())

# Predicting on the test set

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

# Converting probabilities to binary predictions

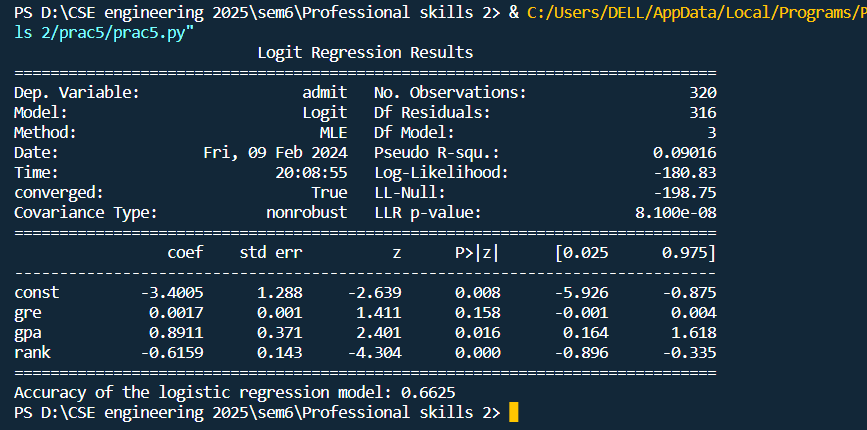
y\_pred\_binary = [1 if p >= 0.5 else 0 for p in y\_pred]

# Calculating accuracy

accuracy = accuracy\_score(y\_test, y\_pred\_binary)

print("Accuracy of the logistic regression model:", accuracy)

5**Output:-**



6**Code:-**

# P6: MULTIPLE REGRESSION MODEL

# Apply multiple regressions, if data have a continuous Independent variable. Apply on above dataset.

import pandas as pd

import statsmodels.api as sm

# Load the dataset

df = pd.read\_csv('prac6/admission.csv')

# Name the dataset

df.name = "Admissions\_Dataset"

# Define independent variables (X) and dependent variable (y)

X = df[['gre', 'gpa', 'rank']] # Independent variables

y = df['admit'] # Dependent variable

# Adding constant term for intercept

X = sm.add\_constant(X)

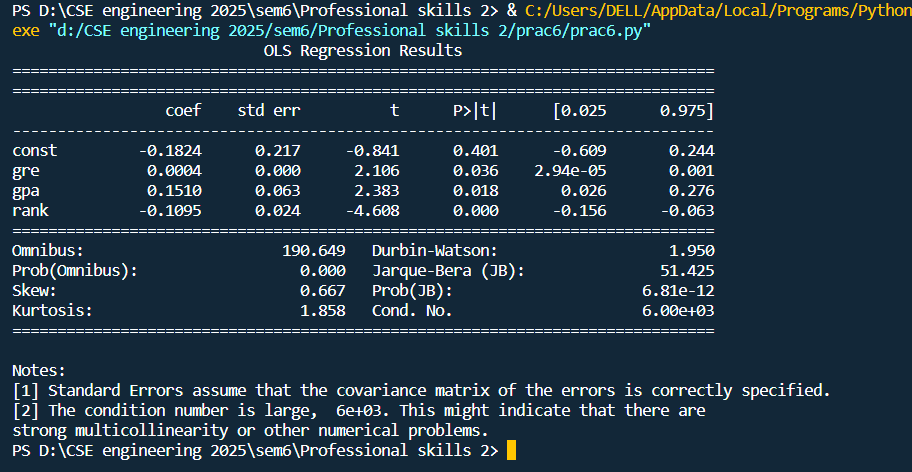
# Fit the multiple regression model

model = sm.OLS(y, X).fit()

# Print model summary

print(model.summary())

6**Output:-**

****

7**Code:-**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report

# b. Choose classifier for classification problem

classifier = LogisticRegression()

df = pd.read\_csv('prac7/iris.csv')

# Name the dataset

df.name = "Classification\_Dataset"

# Define features (X) and target variable (y)

X = df.drop('variety', axis=1) # Features

y = df['variety'] # Target variable

# Split the dataset into training and testing sets

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

classifier = LogisticRegression(solver='saga')

# Fit the classifier on the training data

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

# Predict on the testing data

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

# Evaluate the performance of the classifier

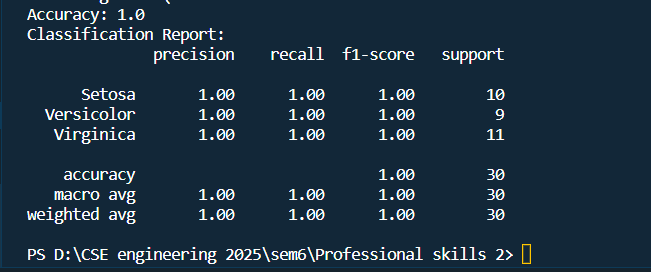
accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

7**Output:-**



8**Code:-**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

df = pd.read\_csv('prac8/iris.csv')

# Name the dataset

df.name = "Clustering\_Dataset"

# Define features (X)

X = df.drop(columns=['variety'])

# Choose the number of clusters (K)

num\_clusters = 3

# Initialize the KMeans model

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)

# Fit the model to the data

kmeans.fit(X)

# Get the cluster labels

cluster\_labels = kmeans.labels\_

# Add cluster labels to the dataframe

df['cluster'] = cluster\_labels

# Plot the clustered data

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

# Scatter plot for each cluster

for cluster in range(num\_clusters):

cluster\_data = df[df['cluster'] == cluster]

plt.scatter(cluster\_data['petal.length'], cluster\_data['petal.width'], label=f'Cluster {cluster}')

# Plot centroids

centroids = kmeans.cluster\_centers\_

plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='black', s=100, label='Centroids')

plt.xlabel('petal.length')

plt.ylabel('petal.width')

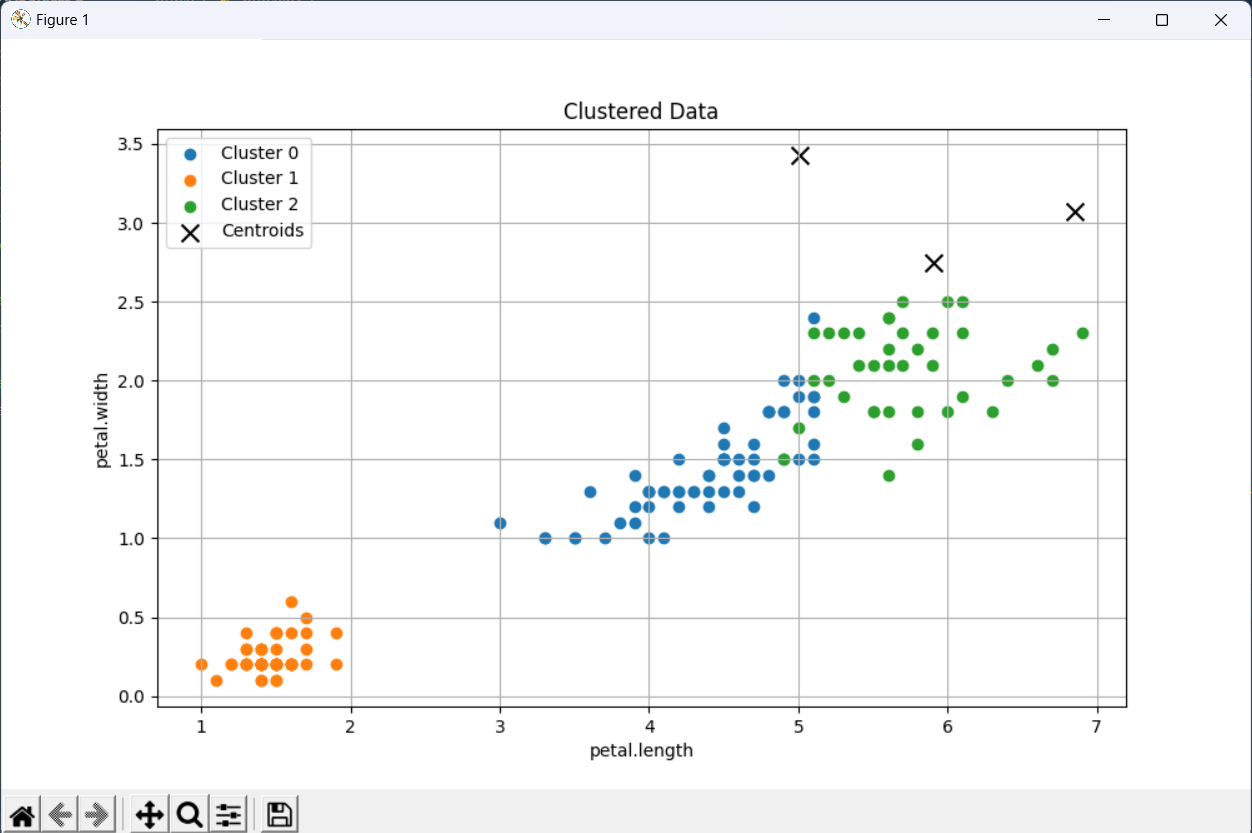
plt.title('Clustered Data')

plt.legend()

plt.grid(True)

plt.show()

8**Output:-**



9**Code:-**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset

df = pd.read\_csv('prac9/iris.csv')

# Define features (X) and target variable (y)

X = df.drop('variety', axis=1) # Features

y = df['variety'] # Target variable

# Split the dataset into training and testing sets

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

# Initialize k-Nearest Neighbors classifier

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the classifier on the training data

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

# Predict on the testing data

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

# Evaluate the performance of the classifier

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Print correct and incorrect predictions

correct\_predictions = X\_test[y\_test == y\_pred]

incorrect\_predictions = X\_test[y\_test != y\_pred]

print("\nCorrect Predictions:")

print(correct\_predictions)

print("\nIncorrect Predictions:")

print(incorrect\_predictions)

9**Output:-**

