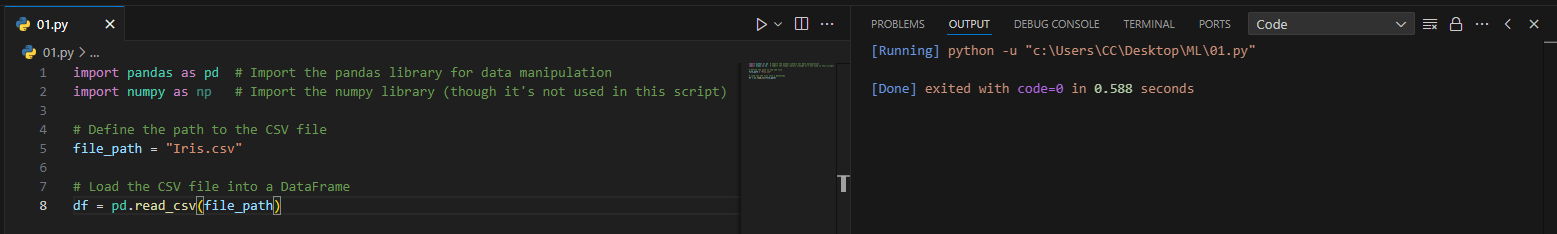
**Experiment No: 01**

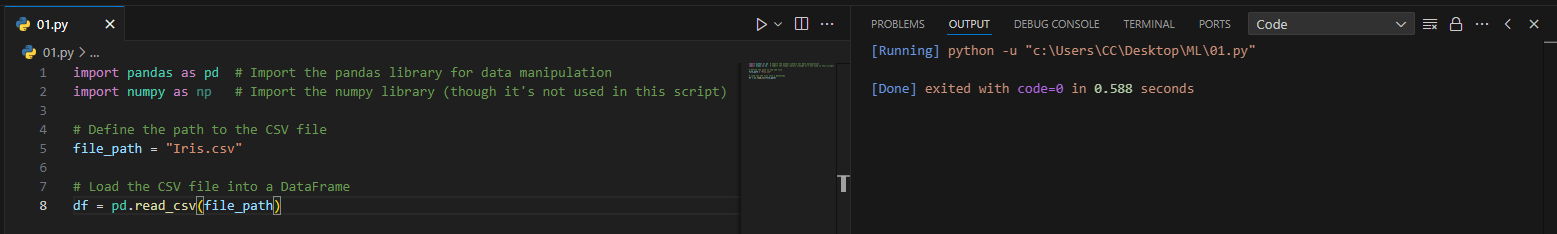
#### 1. Pandas Library:

**Code:**

1. **Importing Libraries:**

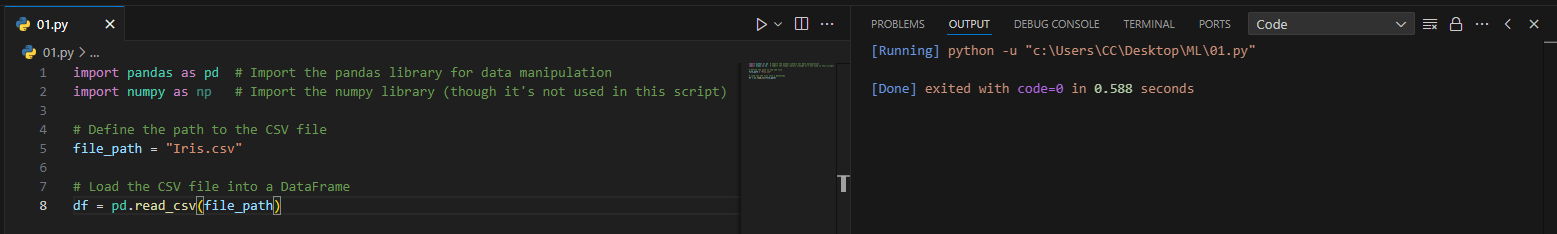
* import pandas as pd: We import Pandas as pd, which is a common alias used by the community for convenience.
* import numpy as np: Numpy is imported as np, though it's not used in this script, it's often used alongside Pandas for numerical operations.

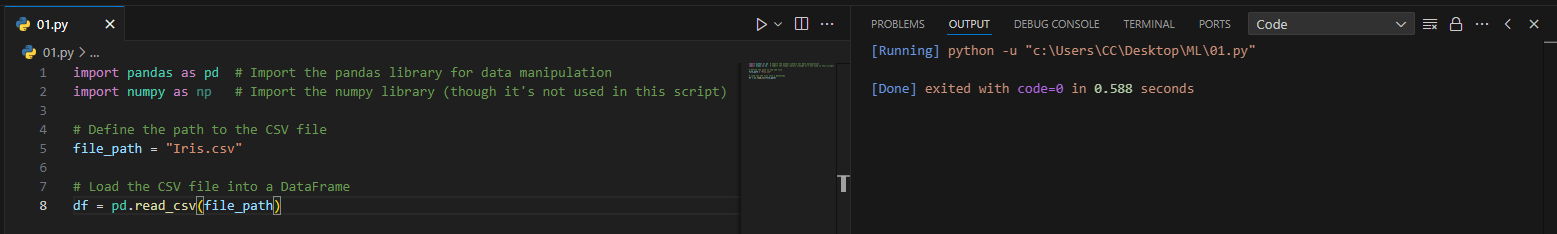




1. **Loading Data:**

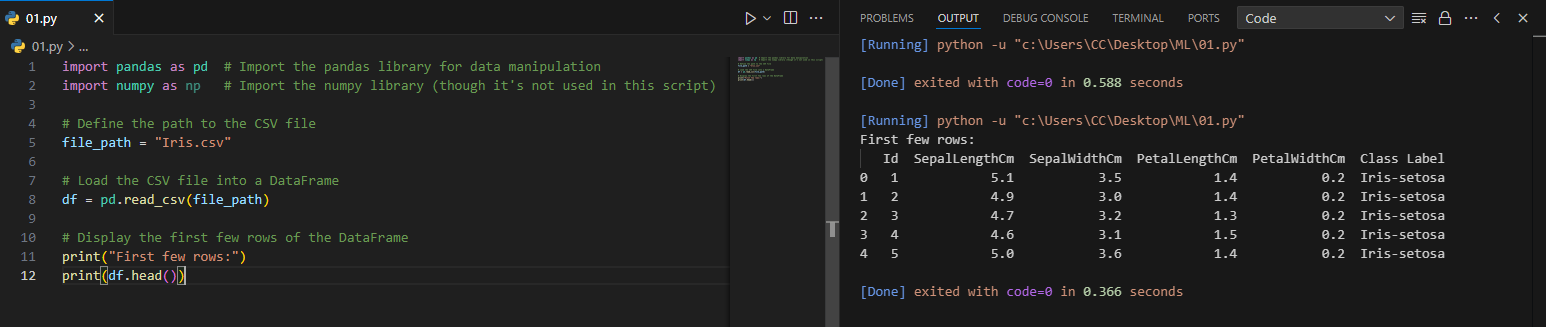
* The CSV file Iris.csv is loaded into a DataFrame using pd.read\_csv(file\_path), where file\_path is the path to the CSV file.

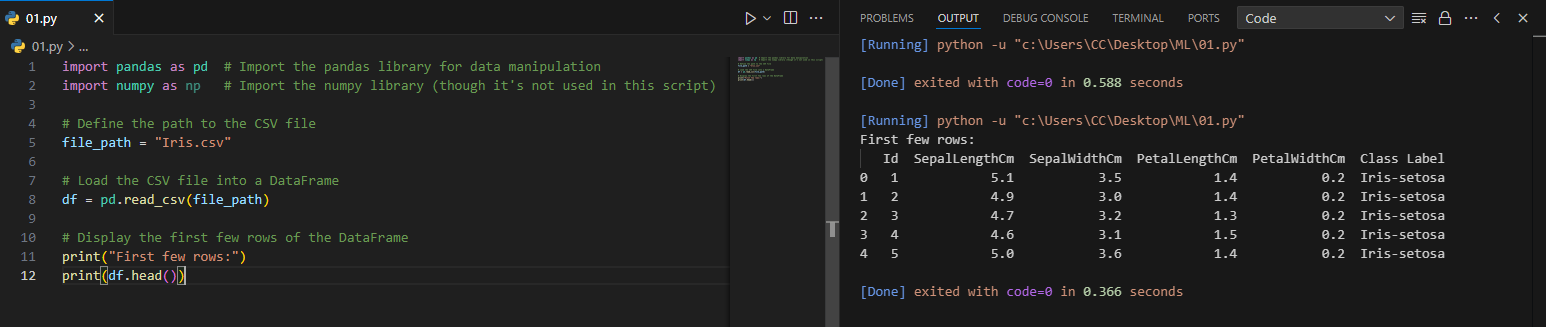




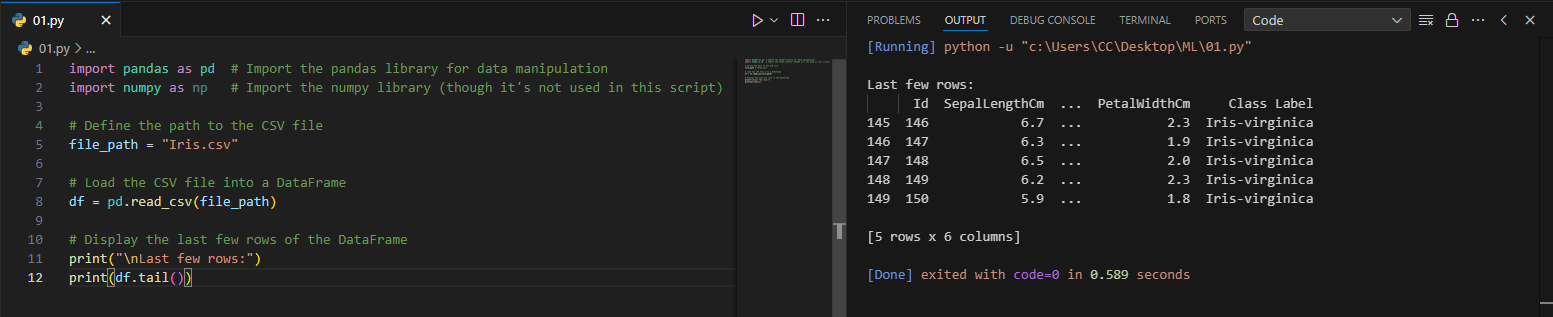
1. **Data Exploration:**

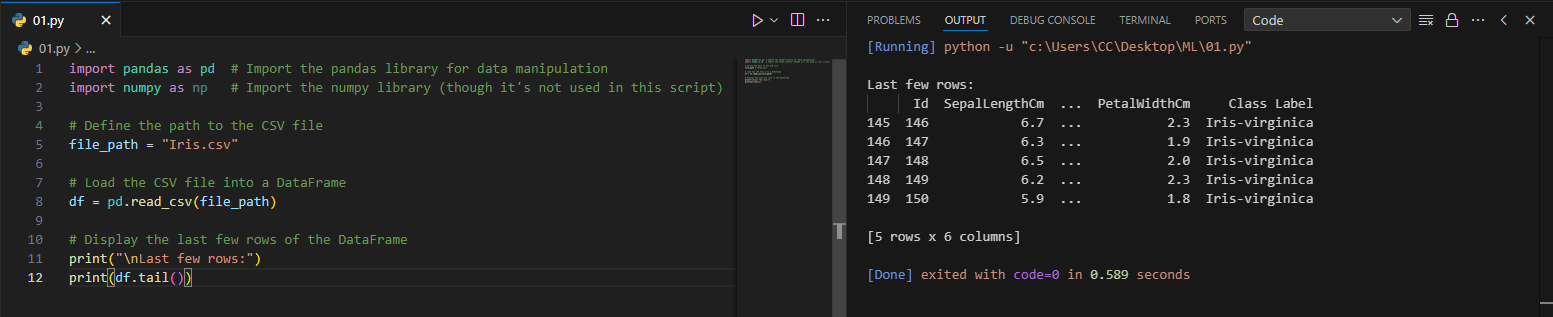
* df.head() displays the first few rows of the DataFrame, giving a quick glance at the data.





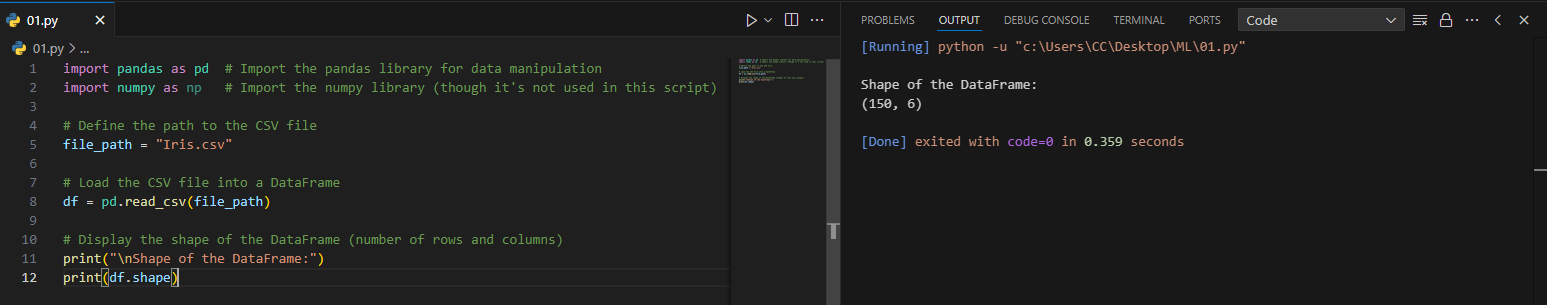
* df.tail() shows the last few rows, which is useful for checking the end of the data.

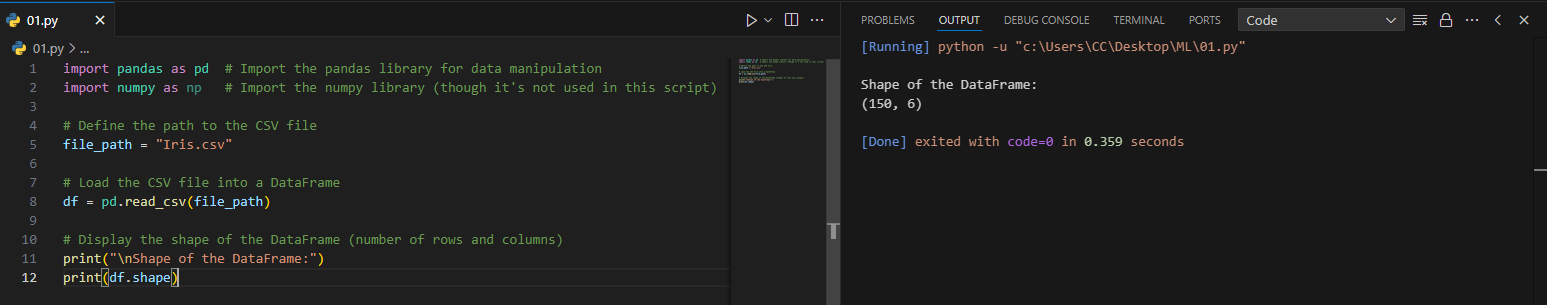




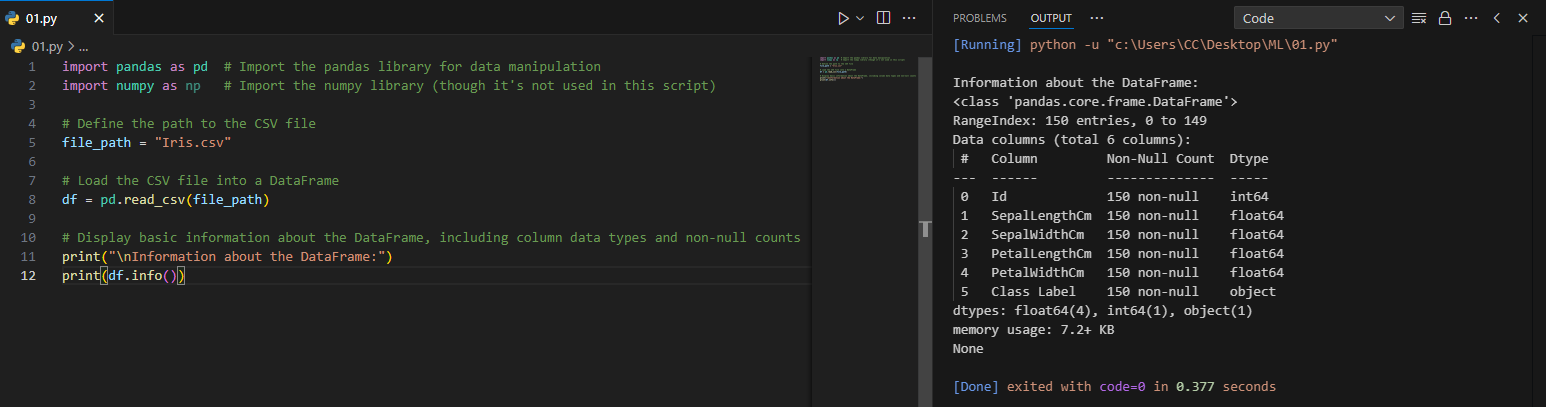
1. **Data Structure and Information:**

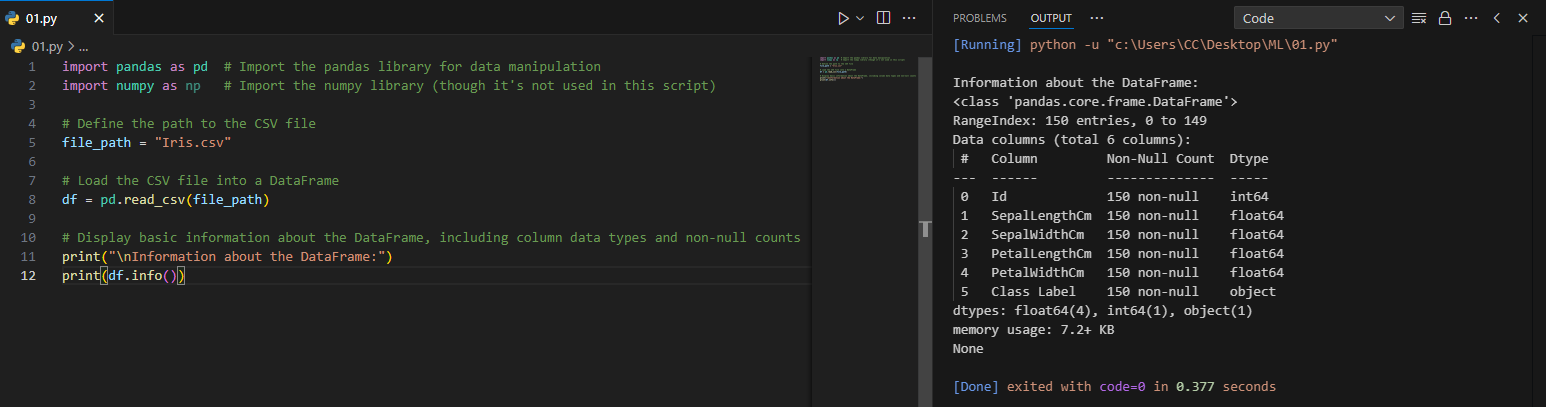
* df.shape gives the number of rows and columns, helping understand the size of the dataset.





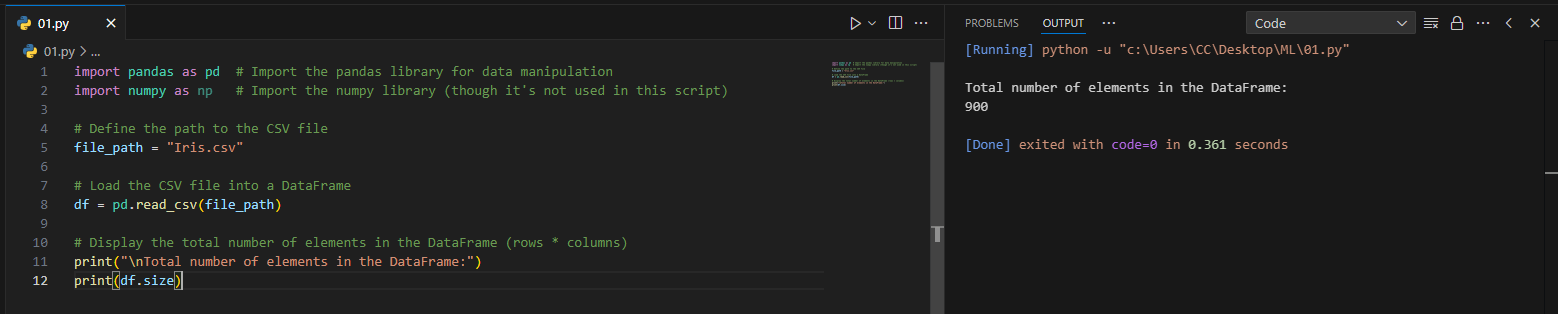
* df.info() provides a summary of the DataFrame, including data types and the count of non-null values for each column.

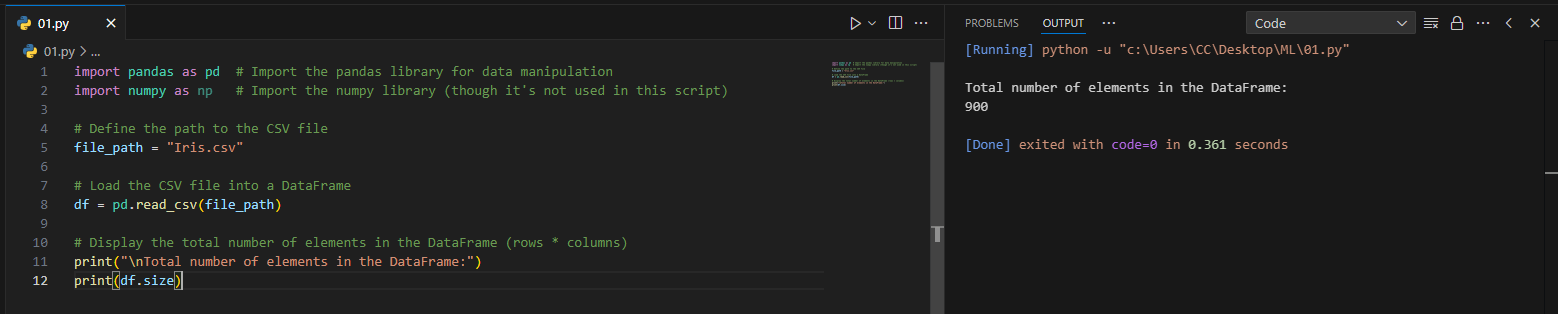




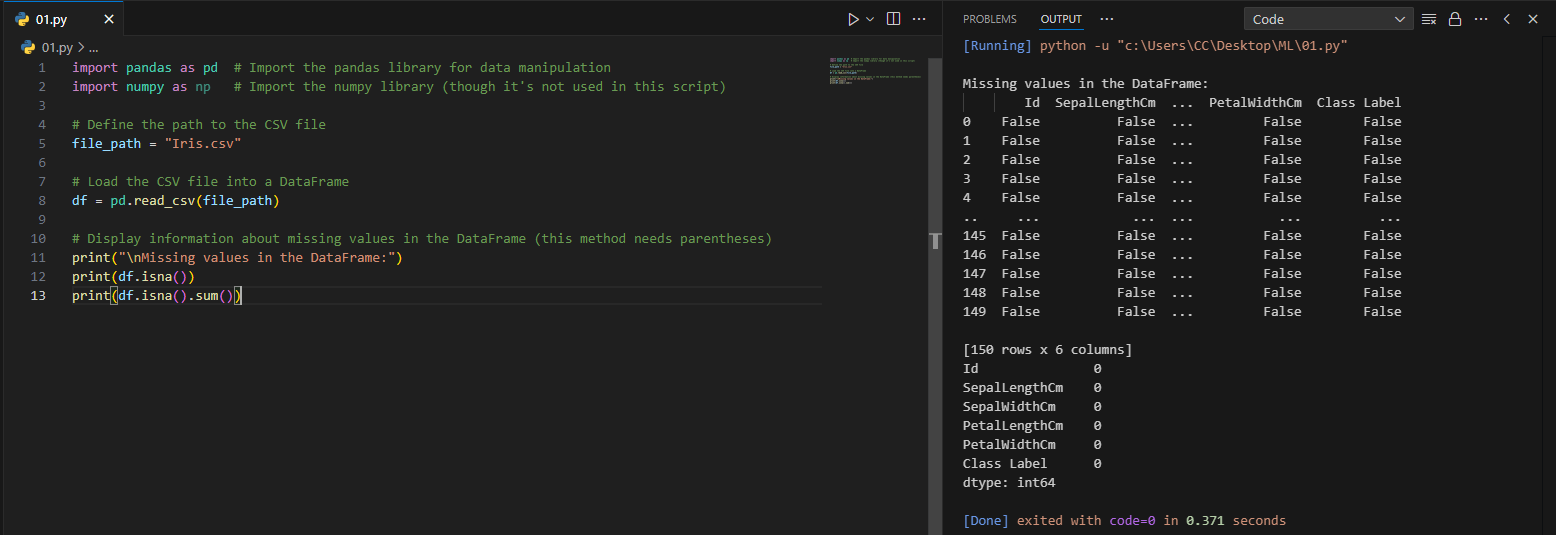
1. **Data Size and Missing Values:**

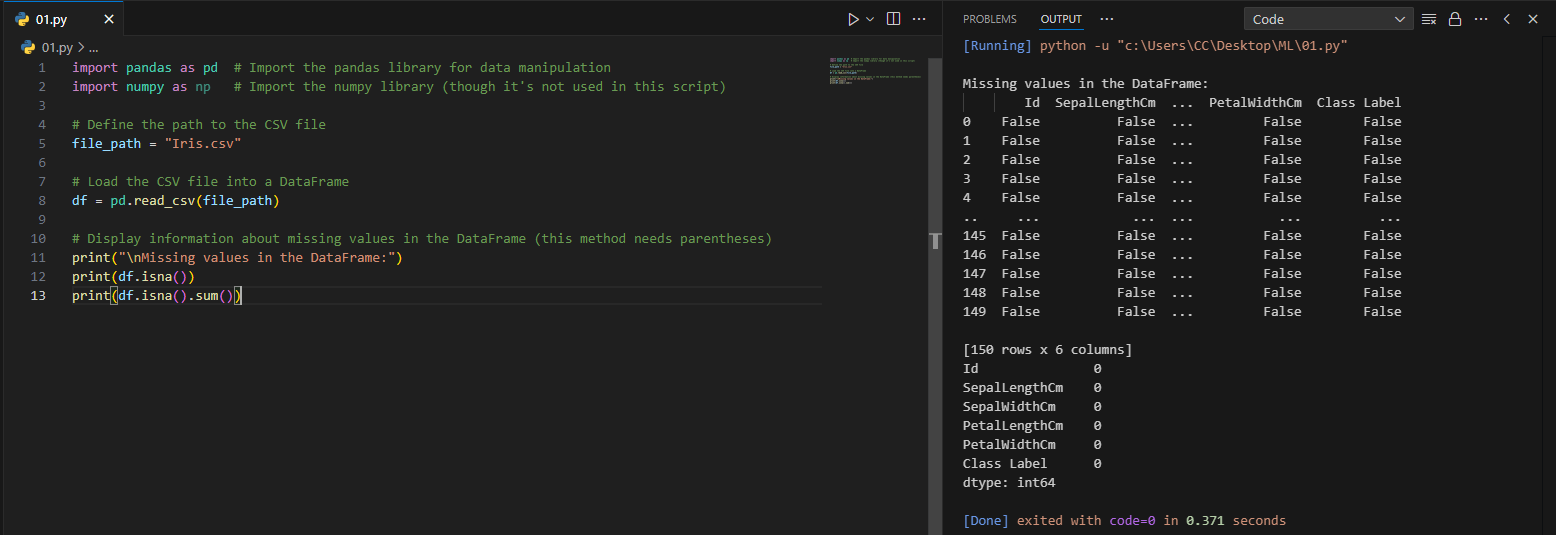
* df.size returns the total number of elements in the DataFrame, which is the product of rows and columns





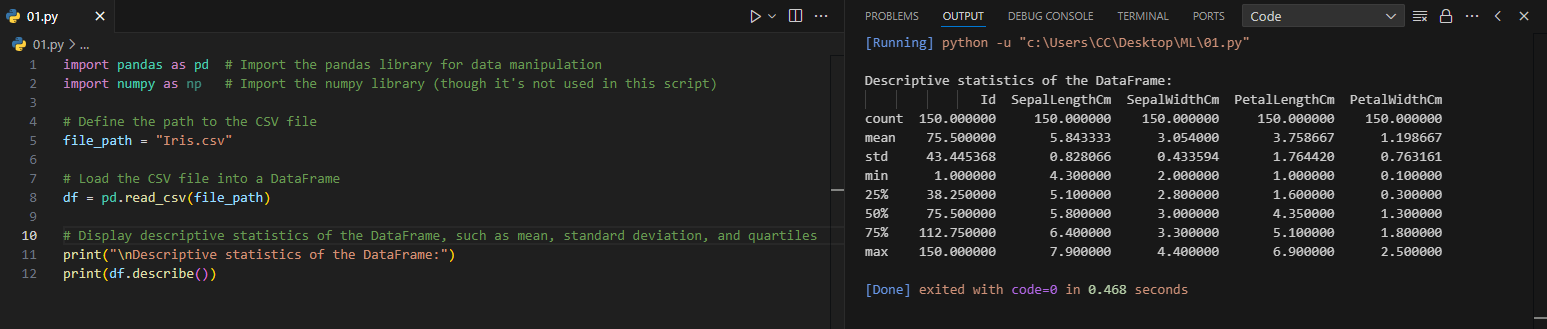
* df.isna() checks for missing values in the DataFrame and returns a DataFrame of the same shape with boolean values (True for missing values). df.isna().sum() gives the total count of missing values for each column.

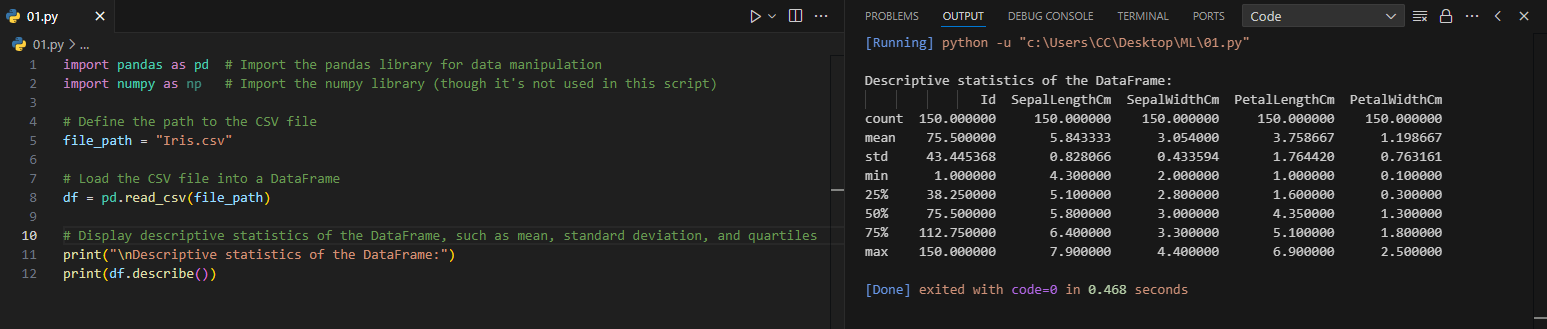




1. **Descriptive Statistics:**

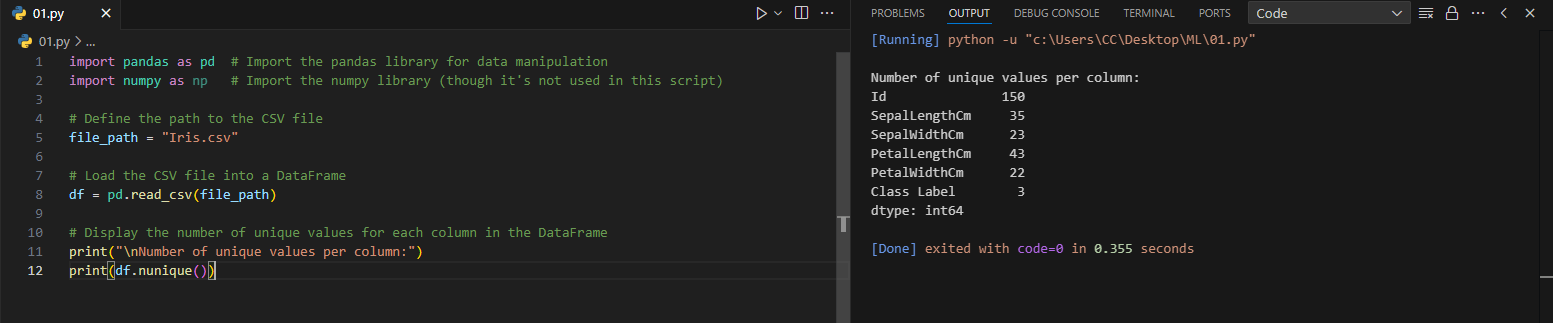
* df.describe() generates descriptive statistics like mean, standard deviation, min, max, and percentiles for each numerical column.

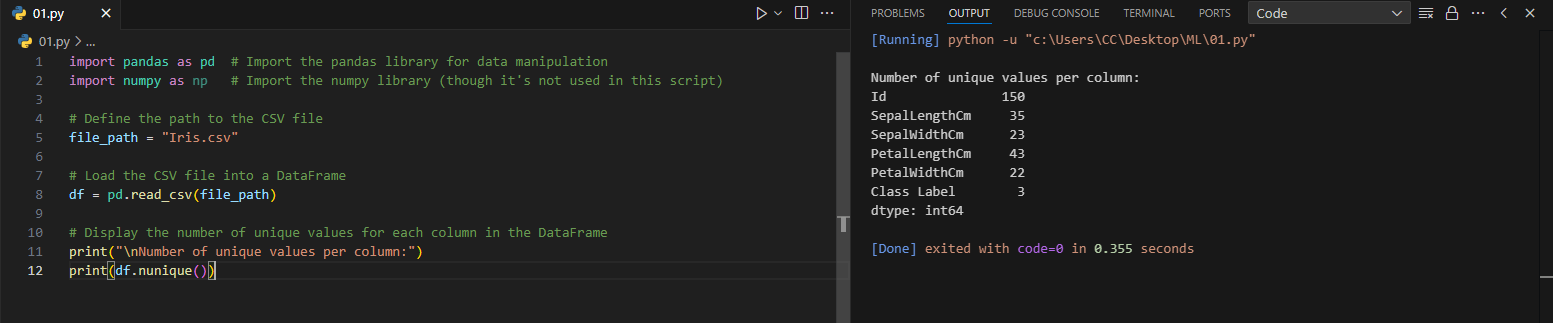




1. **Unique Values:**

* df.nunique() counts the number of unique values in each column, which is helpful in understanding the variability of the data.





#### 2. NumPy Library:

**Code: -**

**1. Creating a NumPy Array**

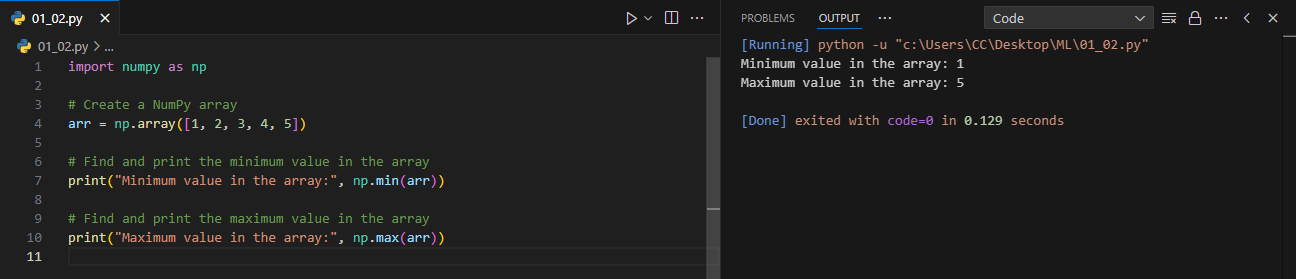
* np.array(): Creates a NumPy array from a list or other iterable, enabling array operations.

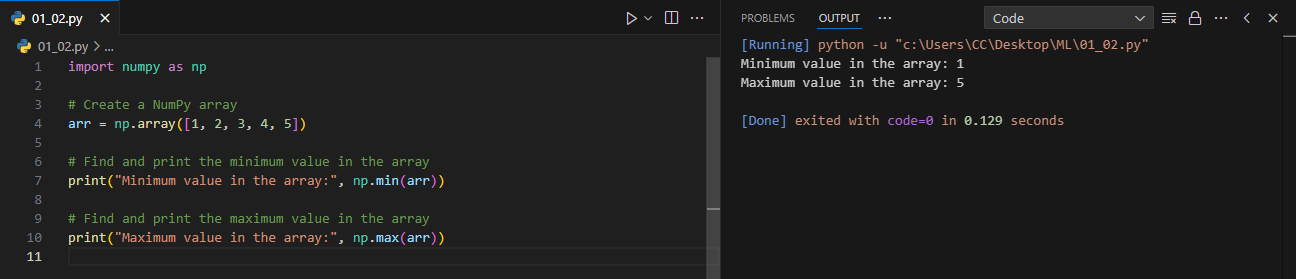
**2. Finding Minimum Value**

* np.min(): Returns the minimum value from the array.

**3. Finding Maximum Value**

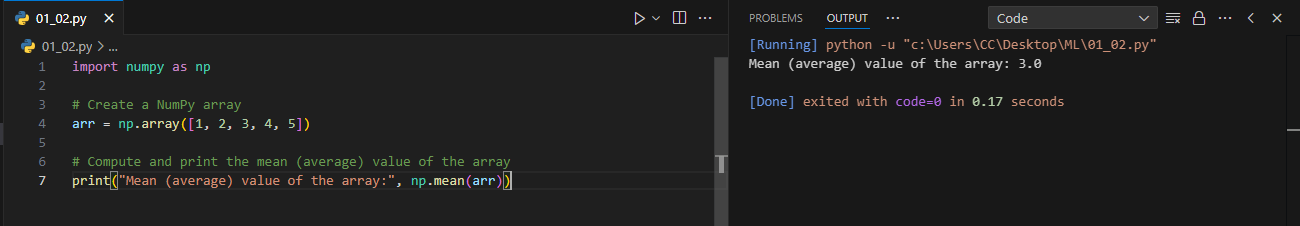
* np.max(): Returns the maximum value from the array.

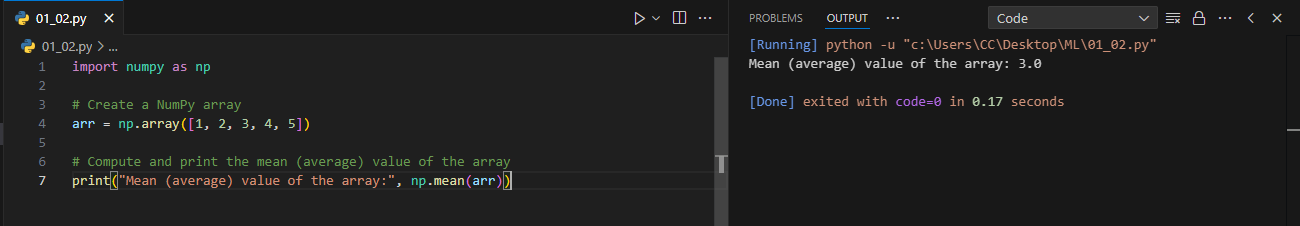




**4. Computing Mean**

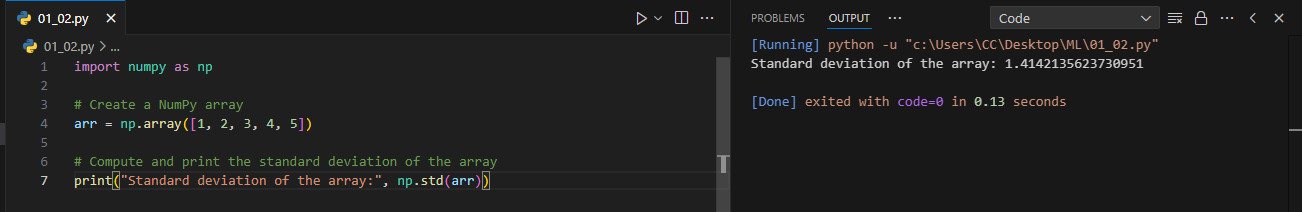
* np.mean(): Computes the arithmetic mean (average) of the elements in the array.

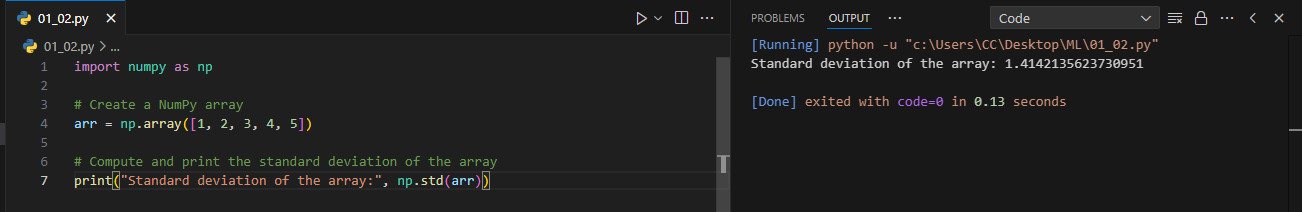




**5. Computing Standard Deviation**

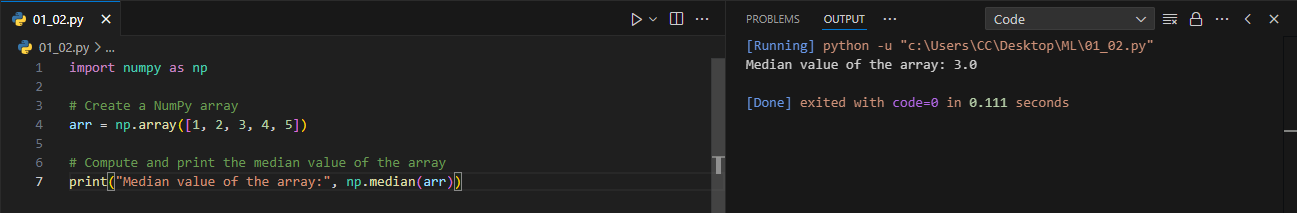
* np.std(): Calculates the standard deviation, which measures the amount of variation or dispersion of a set of values.

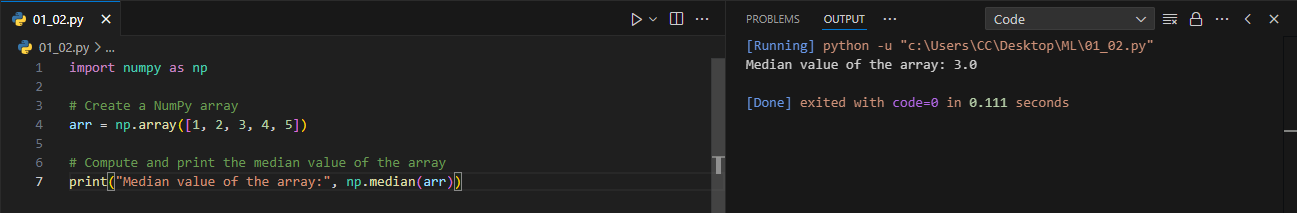




**6. Computing Median**

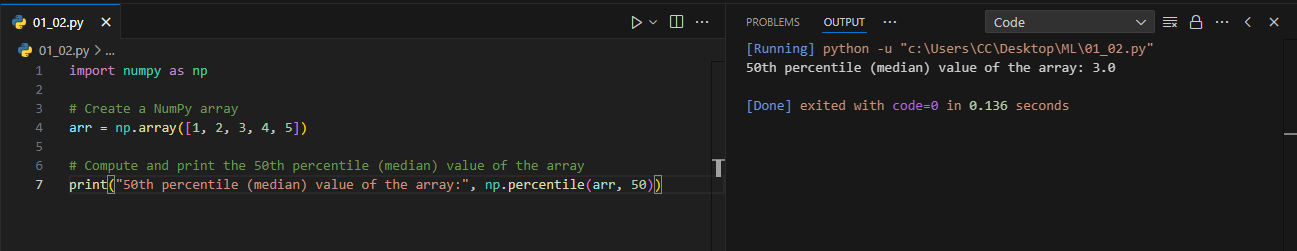
* np.median(): Finds the median value, which is the middle value when the array is sorted.

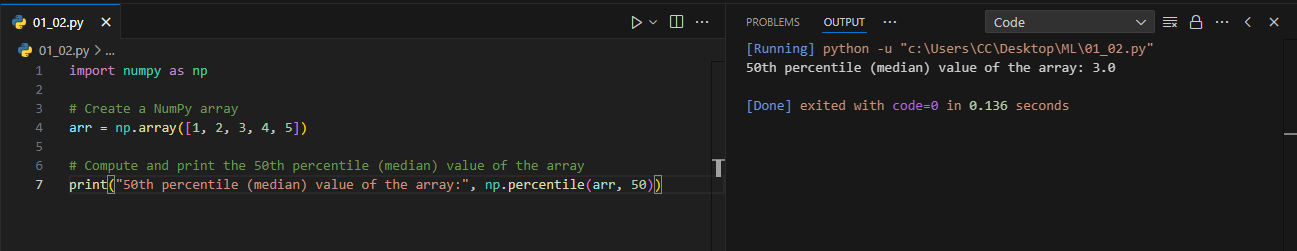




**7. Computing Percentile**

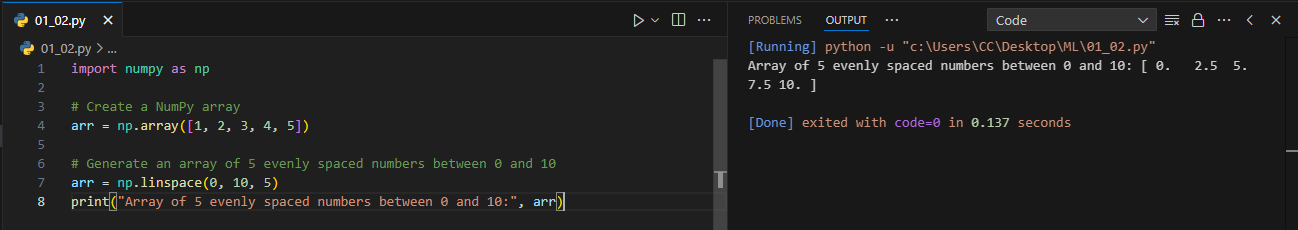
* np.percentile(): Returns the nth percentile of the array. The 50th percentile is the median.

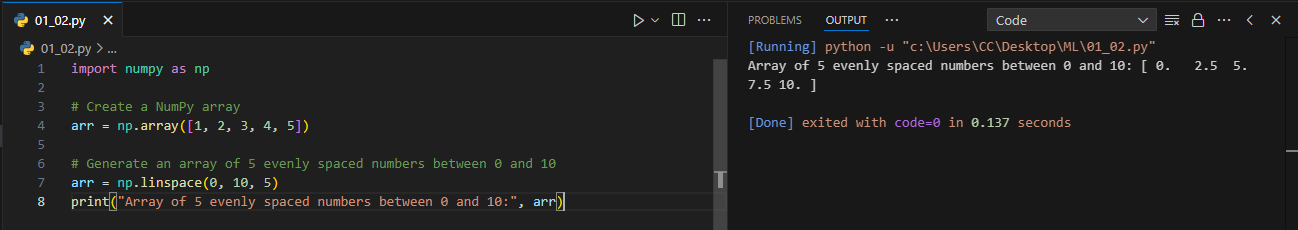




**8. Generating Linearly Spaced Numbers**

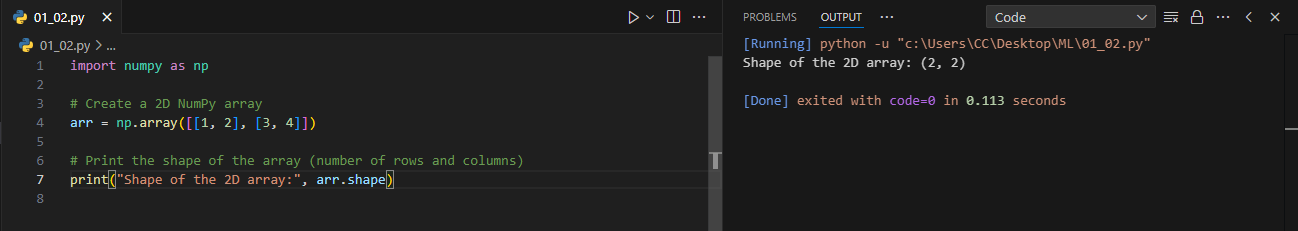
* np.linspace(): Generates an array of evenly spaced numbers over a specified range.

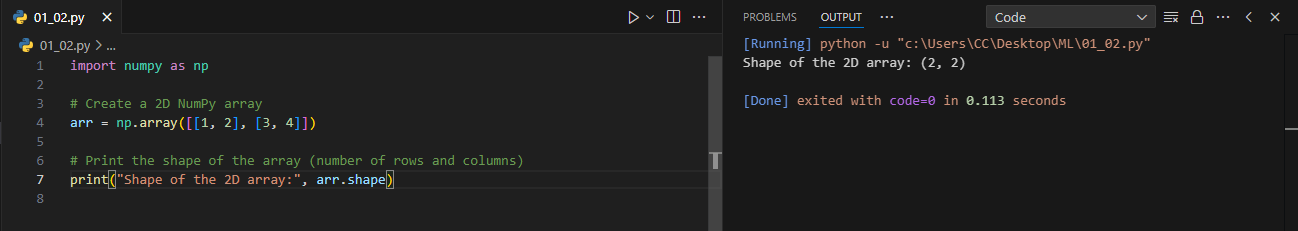




**9. Creating a 2D Array and Getting Array Shape**

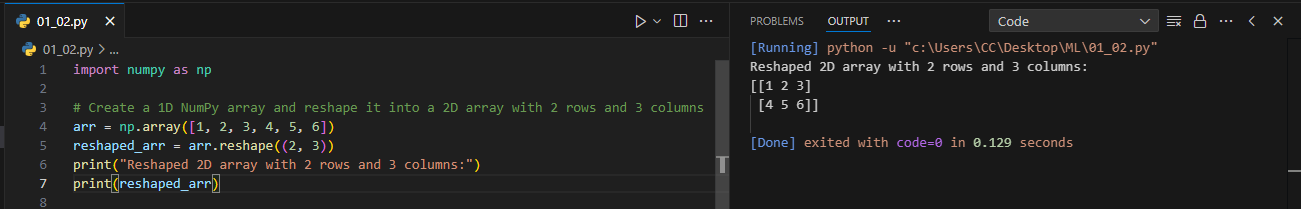
* np.array(): Used again to create a 2D NumPy array.
* .shape: Returns a tuple representing the dimensions (number of rows and columns) of the array.

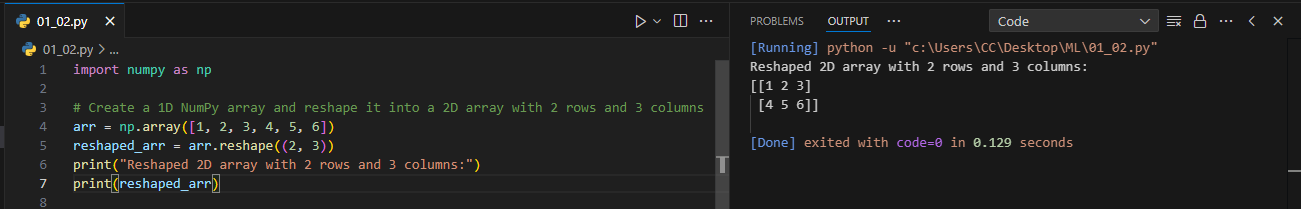




**10. Reshaping an Array**

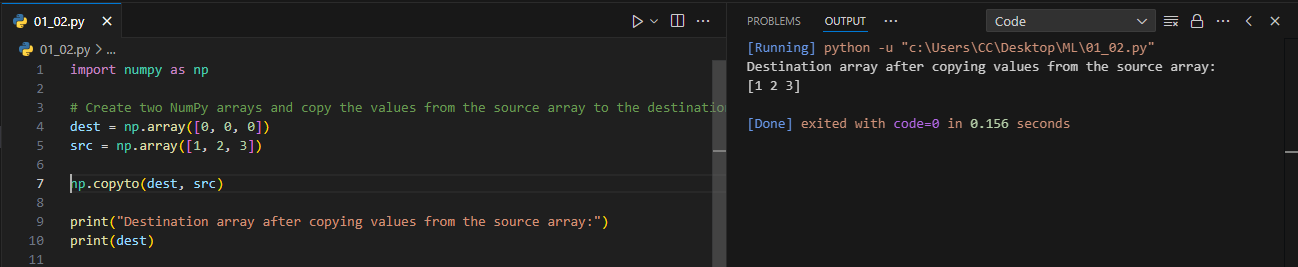
* .reshape(): Changes the shape of an array without changing its data, here reshaping a 1D array into a 2D array.

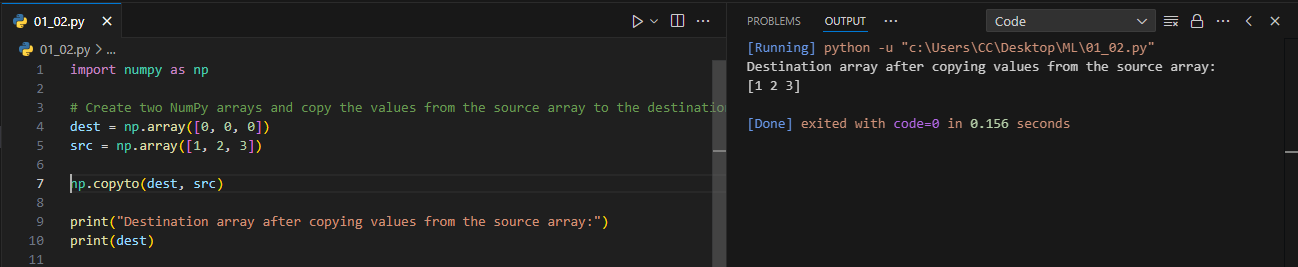




**11. Copying Values Between Arrays**

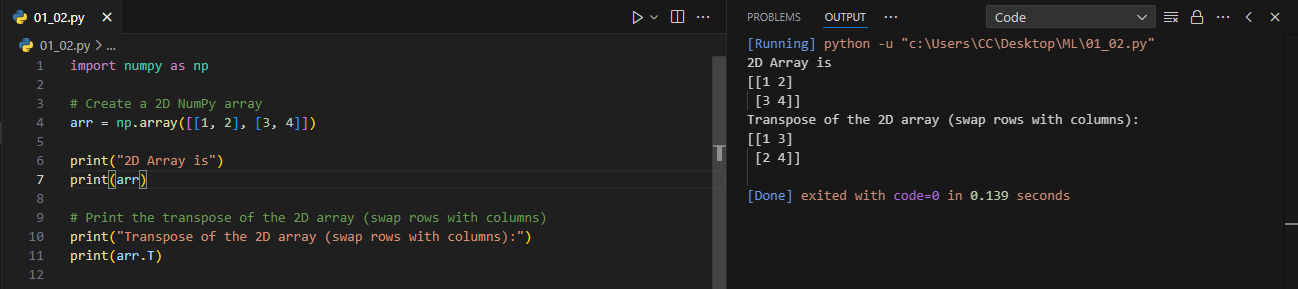
* np.copyto(): Copies values from one array (source) to another (destination), modifying the destination array in place.

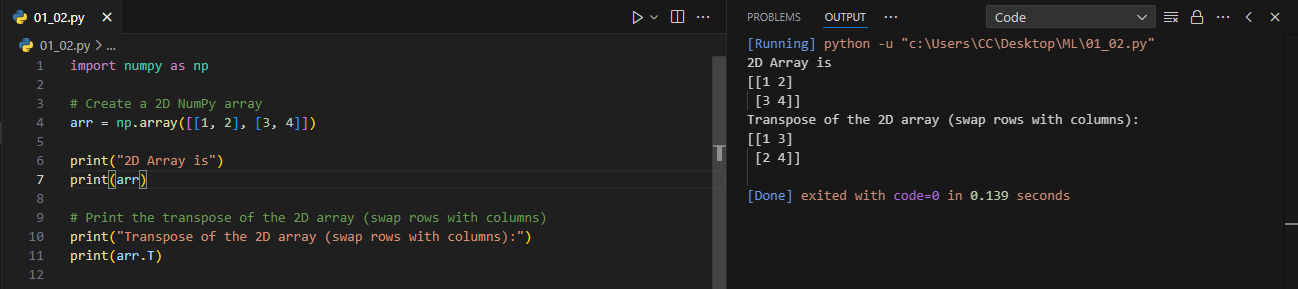




**12. Transposing an Array**

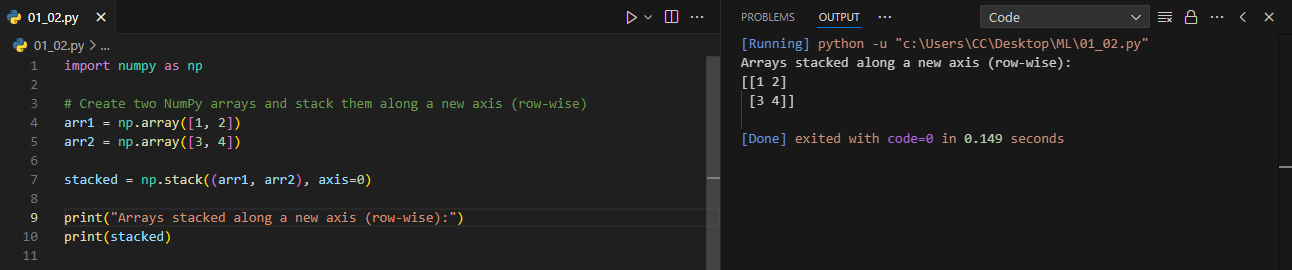
* .T: Transposes the array, swapping rows with columns.

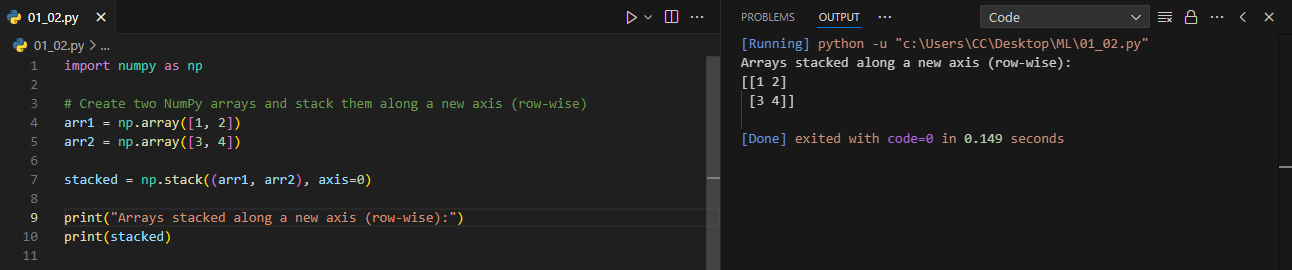




**13. Stacking Arrays**

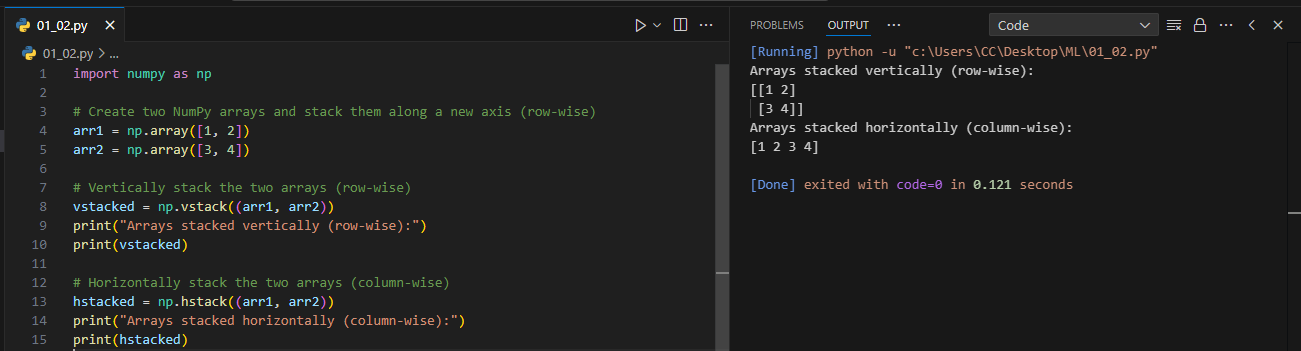
* np.stack(): Combines arrays along a new axis, here stacking them row-wise.

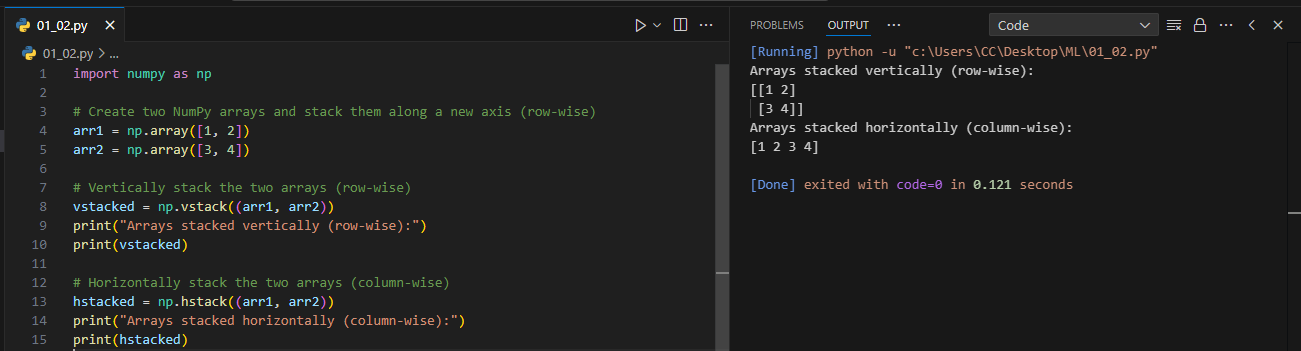




**14. Vertical Stacking and Horizontal Stacking**

* np.vstack(): Vertically stacks arrays, adding rows to form a new array.
* np.hstack(): Horizontally stacks arrays, adding columns to form a new array.





### *Matplotlib Library*

**1. Creating Graph**

1. Creating a Scatter Plot
   1. .plot(): A method from a pandas DataFrame that allows you to create various types of plots. Here, `kind='scatter'` specifies a scatter plot, and the `x` and `y` parameters specify the columns used for the x-axis and y-axis, respectively.
2. Creating a Bar Chart
   1. .plot(): A method from a pandas DataFrame that generates plots. Here, `kind='bar'` specifies that a bar chart should be created, displaying the counts of each variety.
3. Creating a Line Chart
   1. .plot()`: A pandas method that generates plots from DataFrame or Series data. Here, `kind='line'` specifies a line chart, and `marker='o'` adds circular markers at each data point for visibility.

**2. Adding X-axis Label**

* plt.xlabel(): Sets the label for the x-axis in the plot, enhancing readability and providing context for the data.

**3. Adding Y-axis Label**

* plt.ylabel(): Sets the label for the y-axis in the plot, similar to `plt.xlabel()`, ensuring clarity for the viewer.

**4. Adding Title to the Plot**

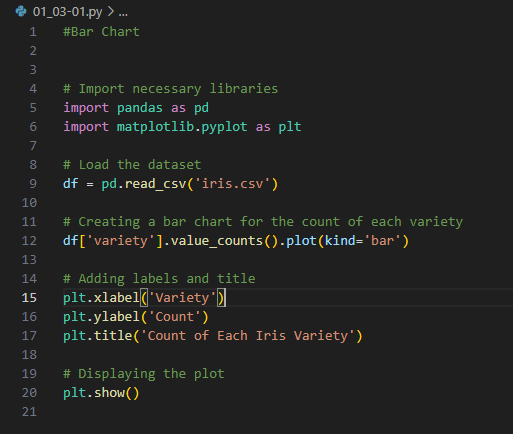
* plt.title(): Assigns a title to the plot, summarizing the content or purpose of the visualization.

**5. Displaying the Plot**

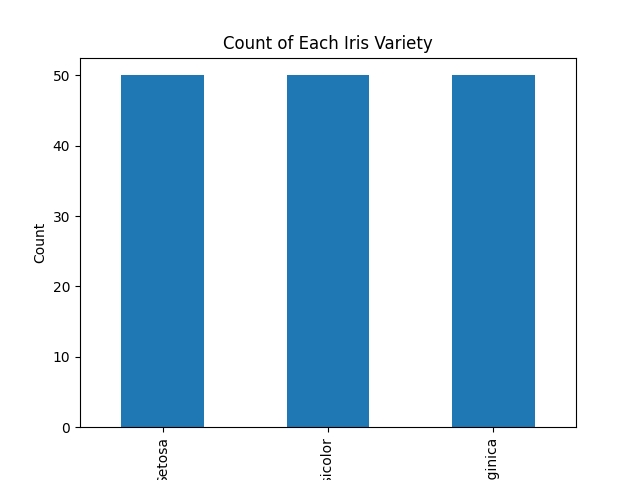
* plt.show(): Renders and displays the current figure or plot. This function is essential for visualizing the created plot in a standalone window or inline, depending on the environment.

**Code: -**

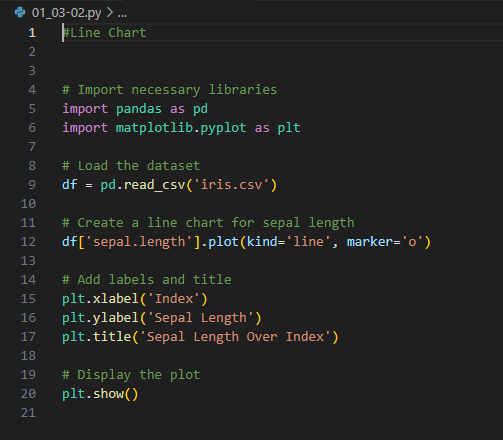
* 1. **Bar Chart**



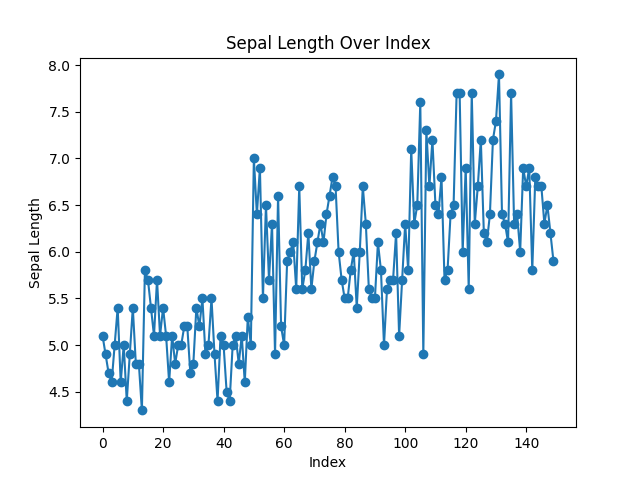
**Output: -**

****

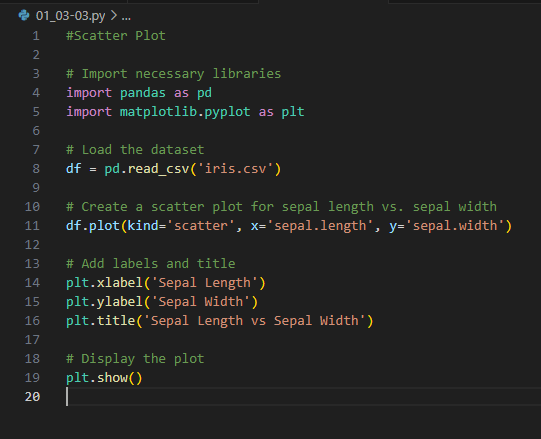
* 1. **Line Chart**



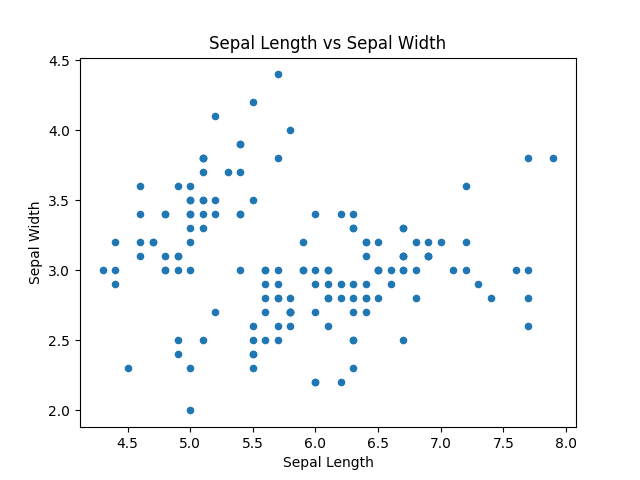
**Output: -**



* 1. **Scatter Plot**

****

**Output:-**



**Experiment No: 02**

**Code :-**

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

x\_true = [1, 0, 2, 2, 1, 0]

x\_pred = [2, 0, 2, 2, 0, 0]

print(confusion\_matrix(x\_true, x\_pred))

y\_true = ["cat", "ant", "cat", "ant", "cat", "bird"]

y\_pred = ["cat", "ant", "ant", "cat", "ant", "cat"]

print(confusion\_matrix(y\_true, y\_pred, labels = ["ant", "cat", "bird"]))

from sklearn.datasets import load\_breast\_cancer

from sklearn.ensemble import RandomForestClassifier

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

# Loading the breast cancer data set

diabetes\_data = load\_breast\_cancer()

# Creating independent and dependent variables

X = diabetes\_data.data

y = diabetes\_data.target

# Splitting the data into training and testing set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2,

random\_state=24)

print(f"Train Data: {X\_train.shape}, {y\_train.shape}")

print(f"Test Data: {X\_test.shape}, {y\_test.shape}")

# Training a binary classifier using Random Forest Algorithm with default hyperparameters

classifier = RandomForestClassifier(random\_state=18)

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

# Here X\_test, y\_test are the test data points

predictions = classifier.predict(X\_test)

#Importing all necessary libraries

from sklearn.metrics import accuracy\_score

# Calculating the accuracy of classifier

print(f"Accuracy of the classifier is: {accuracy\_score(y\_test, predictions)}")

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

# Compute and print the confusion matrix

cm = confusion\_matrix(y\_test, predictions)

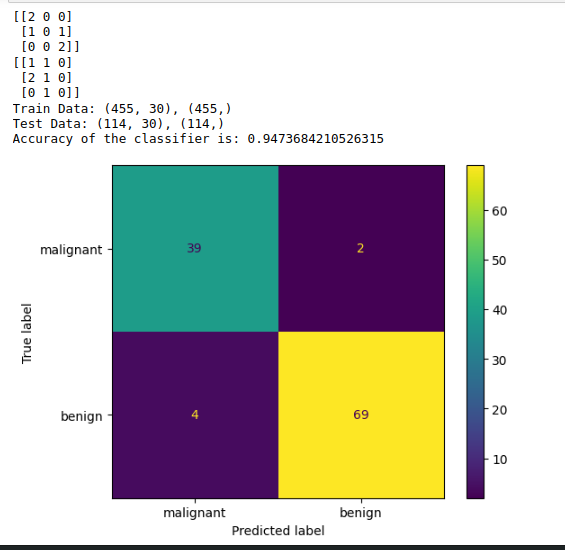
# Plot the confusion matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=diabetes\_data.target\_names)

disp.plot()

plt.show()

**Output :-**

****

**Experiment No: 03**

**Code 01 :-**

import pandas as pd

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

from sklearn.metrics import accuracy\_score

from sklearn.neighbors import KNeighborsClassifier

# Load the dataset

file = "diabetes.csv"

df = pd.read\_csv(file)

# Separate features and target variable

data = df.values

X, y = data[:, :-1], data[:, -1]

print(f"Feature shape: {X.shape}, Target shape: {y.shape}")

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

print(f"Training feature shape: {X\_train.shape}, Testing feature shape: {X\_test.shape}")

print(f"Training target shape: {y\_train.shape}, Testing target shape: {y\_test.shape}")

# Initialize and fit the KNN model

model = KNeighborsClassifier()

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

# Make predictions

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

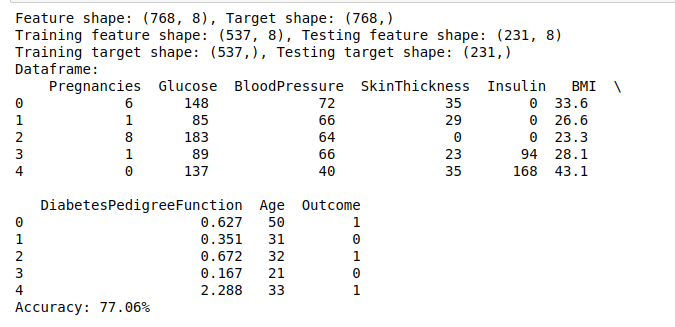
# Calculate and print accuracy

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

print("Dataframe:\n", df.head()) # Print only the first few rows for clarity

print("Accuracy: {:.2f}%".format(accuracy \* 100))

**Code 01 Output :-**



**Code 02 :-**

import numpy as np

import pandas as pd

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

# Define arrays for X and y

X = np.array([

[1, 2, 3],

[4, 5, 6],

[7, 8, 9],

[10, 11, 12],

[13, 14, 15],

[16, 17, 18],

[19, 20, 21],

[22, 23, 24],

[25, 26, 27],

[28, 29, 30]

])

y = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Converting to pandas DataFrame for consistency

df = pd.DataFrame(X, columns=['Feature1', 'Feature2', 'Feature3'])

df['Target'] = y

# Separating features and target variable

X = df.iloc[:, :-1] # Features

y = df.iloc[:, -1] # Target variable

# Spliting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

# Print the training and testing sets

print("X\_train:")

print(X\_train)

print("\nX\_test:")

print(X\_test)

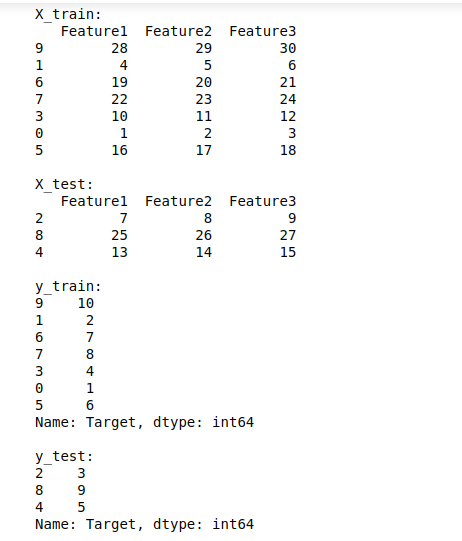
print("\ny\_train:")

print(y\_train)

print("\ny\_test:")

print(y\_test)

**Code 02 Output :-**



**Experiment No: 04**

**Code :-**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

# Importing the dataset

dataset = pd.read\_csv('Salary\_Dataset.csv')

X = dataset.iloc[:, 0:1].values # Features (Years of Experience)

y = dataset.iloc[:, 1].values # Target variable (Salary)

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

# Fitting the Linear Regression model

regressor = LinearRegression()

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

# Print shapes for debugging

print("X\_train shape:", X\_train.shape)

print("y\_train shape:", y\_train.shape)

print("X shape:", X.shape)

print("y shape:", y.shape)

# Visualizing the Training set results

plt.scatter(X\_train, y\_train, color='red')

plt.plot(X\_train, regressor.predict(X\_train), color='blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

# Visualizing the Test set results

plt.scatter(X\_test, y\_test, color='red')

plt.plot(X\_train, regressor.predict(X\_train), color='blue') # Using training data for the line

plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

# Predicting the Test set results

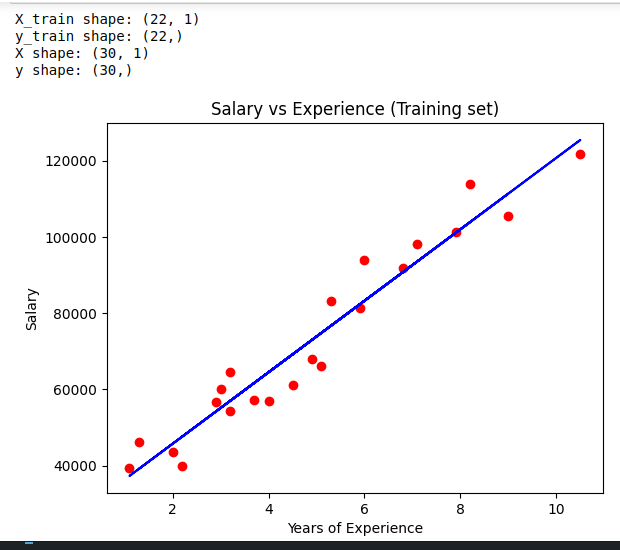
y\_pred = regressor.predict(X\_test)

# Displaying predictions

for pred in y\_pred:

print("Predicted Salary:", pred)

**Code Output :-**





**Experiment No: 05**

**Code 01 :-**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Import warnings

import warnings

warnings.filterwarnings("ignore")

# We will use some methods from the sklearn module

from sklearn import linear\_model

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

# Load the dataset

df = pd.read\_csv("Student\_Performance.csv")

print(df.corr(numeric\_only=True)) # Use numeric\_only=True to only get numeric correlations

print(df.describe())

# Setting the value for X and y

X = df[['Hours\_Studied', 'Previous\_Scores']] # Adjusted features

y = df['Performance\_Index'] # Adjusted target variable

# Visualizing the features

fig, axs = plt.subplots(2, figsize=(5, 10)) # Adjusted height for better spacing

plt1 = sns.boxplot(x=df['Hours\_Studied'], ax=axs[0]) # Adjusted for the new feature

plt2 = sns.boxplot(x=df['Previous\_Scores'], ax=axs[1]) # Adjusted for the new feature

plt.tight\_layout()

sns.histplot(df['Performance\_Index'], kde=True) # Changed to histplot with kde for better visualization

sns.pairplot(df, x\_vars=['Hours\_Studied', 'Previous\_Scores'], y\_vars='Performance\_Index', height=4, aspect=1, kind='scatter')

plt.show()

# Create the correlation matrix and represent it as a heatmap.

sns.heatmap(df.corr(numeric\_only=True), annot=True, cmap='coolwarm') # Use numeric\_only=True for heatmap

plt.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=100)

# Fitting the Multiple Linear Regression model

reg\_model = LinearRegression().fit(X\_train, y\_train)

# Printing the model coefficients

print('Intercept: ', reg\_model.intercept\_)

# Pair the feature names with the coefficients

print(list(zip(X.columns, reg\_model.coef\_))) # Corrected to use .columns for feature names

# Predicting the Test and Train set result

y\_pred = reg\_model.predict(X\_test)

x\_pred = reg\_model.predict(X\_train) # This is actually predicting on the train set

print("Prediction for test set: {}".format(y\_pred))

# Actual value and the predicted value

reg\_model\_diff = pd.DataFrame({'Actual value': y\_test, 'Predicted value': y\_pred})

print(reg\_model\_diff) # Added print to display the DataFrame

mae = metrics.mean\_absolute\_error(y\_test, y\_pred)

mse = metrics.mean\_squared\_error(y\_test, y\_pred)

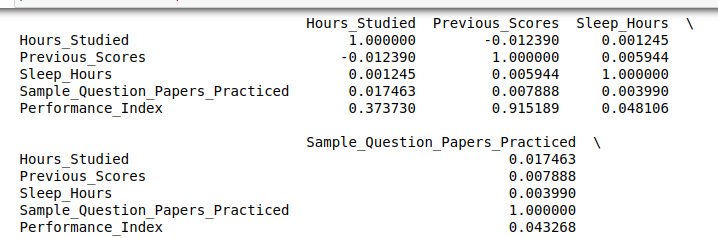
r2 = np.sqrt(mse) # Changed to use mse instead of mean\_squared\_error directly

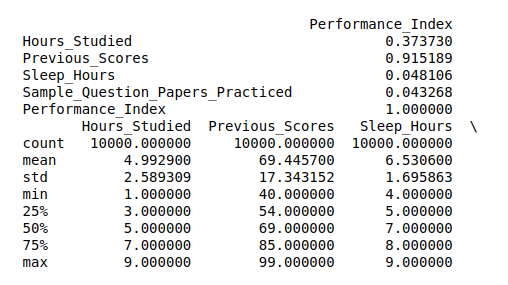
print('Mean Absolute Error:', mae)

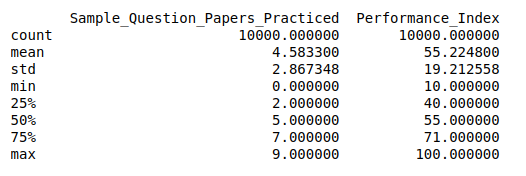
print('Mean Square Error:', mse)

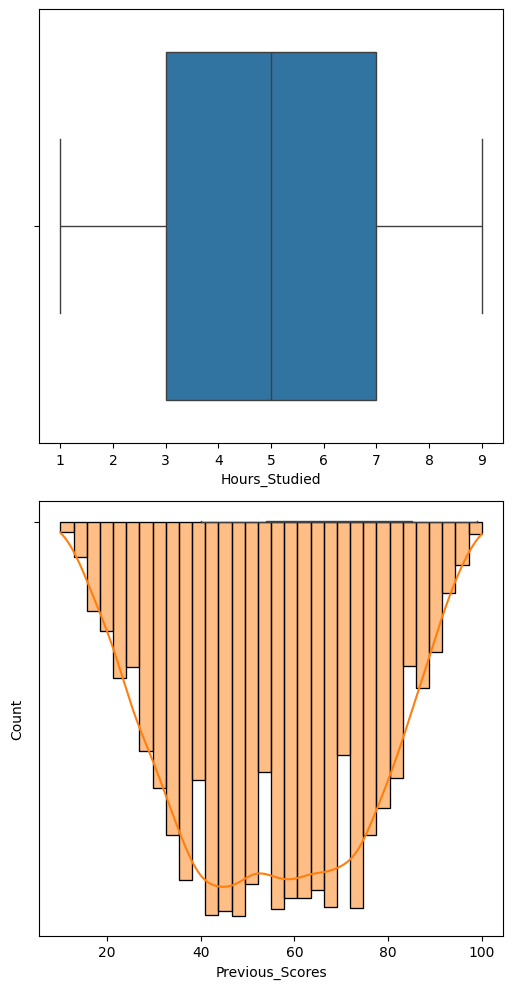
print('Root Mean Square Error:', r2)

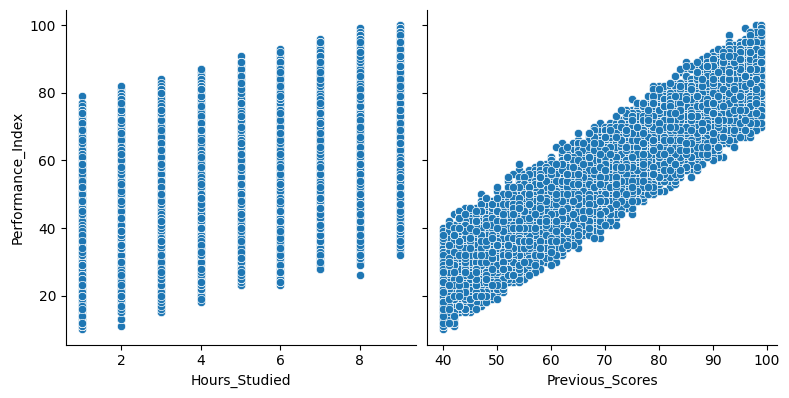
**Code 01 Output :-**

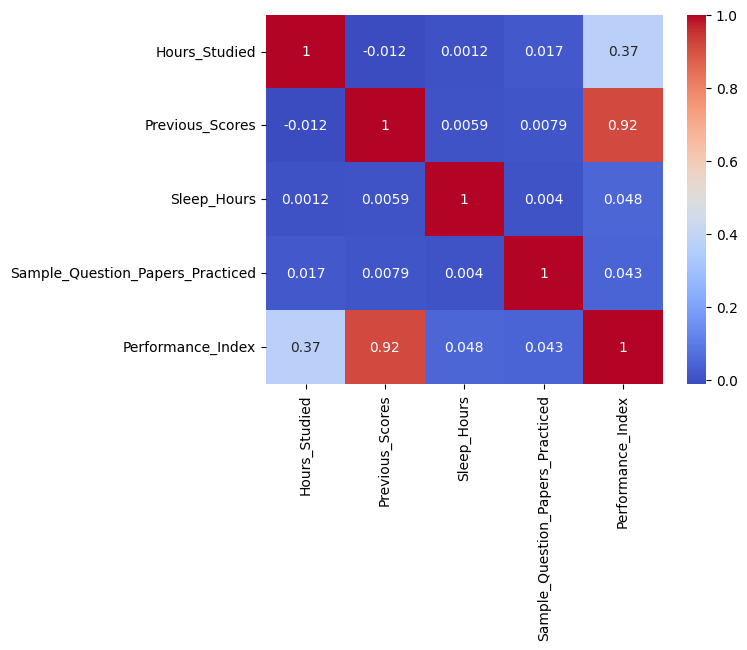


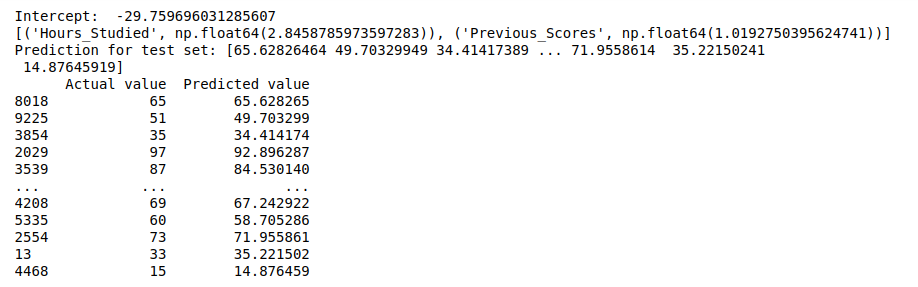


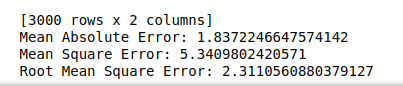








****

****

**Code 02 :-**

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import linear\_model

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

from sklearn import metrics

# Load the dataset

df = pd.read\_csv("Student\_Performance.csv")

# Prepare the features and target variable

X = df[['Hours\_Studied', 'Previous\_Scores']] # Adjusted features

y = df['Performance\_Index'] # Adjusted target variable

# Perform exploratory data analysis

print(df.corr(numeric\_only=True)) # Only get numeric correlations

sns.pairplot(df, x\_vars=['Hours\_Studied', 'Previous\_Scores'], y\_vars='Performance\_Index', height=4, aspect=1, kind='scatter')

plt.show()

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create and fit the multivariable regression model

reg\_model = linear\_model.LinearRegression()

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

# Print model coefficients

print('Intercept:', reg\_model.intercept\_)

print('Coefficients:', reg\_model.coef\_)

# Make predictions

y\_pred = reg\_model.predict(X\_test)

# Evaluate the model

mae = metrics.mean\_absolute\_error(y\_test, y\_pred)

mse = metrics.mean\_squared\_error(y\_test, y\_pred)

r2 = metrics.r2\_score(y\_test, y\_pred)

print('Mean Absolute Error:', mae)

print('Mean Squared Error:', mse)

print('R-squared:', r2)

# Visualize actual vs predicted values

plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual Performance Index')

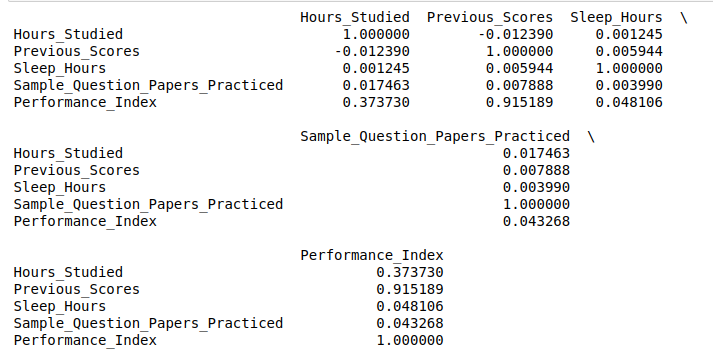
plt.ylabel('Predicted Performance Index')

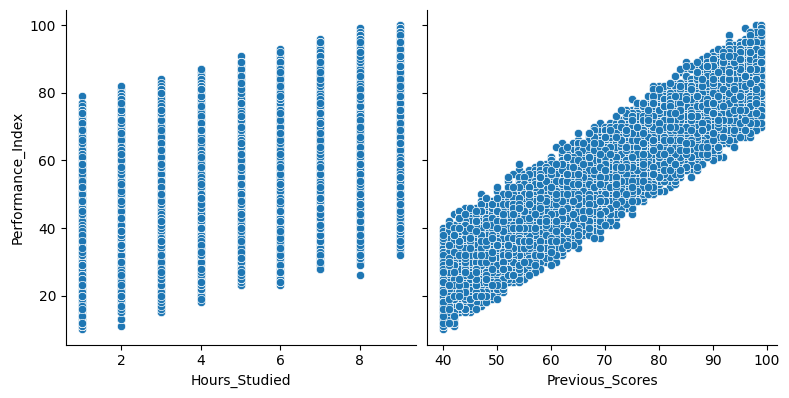
plt.title('Actual vs Predicted Performance Index')

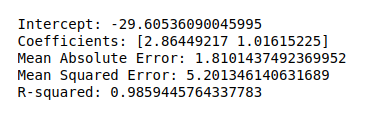
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--') # 45-degree line

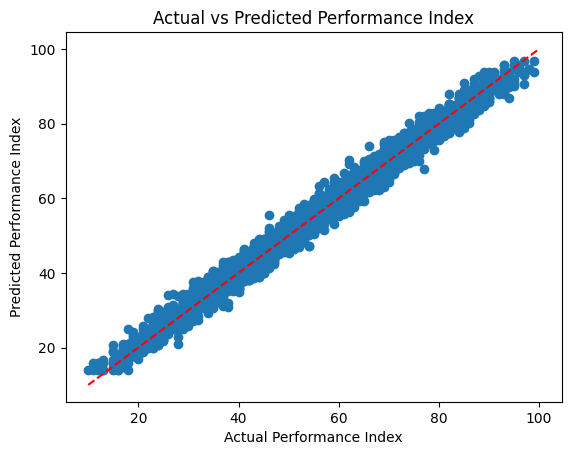
plt.show()

**Code 02 Output: -**









**Experiment No: 06**

**Code:**

# Import libraries

import sys

import matplotlib

matplotlib.use('Agg')

import pandas as pd

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

# Load the data

df = pd.read\_csv(r"D:\[Current\_Learning]\TY\_NOTES\ML\Practical\Exp\_No\_06\Experiment6.csv")

# Map categorical variables to numerical values

nationality\_map = {'UK': 1, 'USA': 0, 'N': 2}

df['Nationality'] = df['Nationality'].map(nationality\_map)

go\_map = {'YES': 1, 'NO': 0}

df['Go'] = df['Go'].map(go\_map)

# Define all possible feature pairs to explore

feature\_combinations = [

    ['Age', 'Experience'],

    ['Age', 'Rank'],

    ['Age', 'Nationality'],

    ['Experience', 'Rank'],

    ['Experience', 'Nationality'],

    ['Rank', 'Nationality']

]

# Iterate through each feature combination and plot the tree

for i, features in enumerate(feature\_combinations):

    X = df[features]

    y = df['Go']

    # Initialize and fit the Decision Tree Classifier

    dtree = DecisionTreeClassifier()

    dtree.fit(X, y)

    # Plot the decision tree

    plt.figure(figsize=(10, 8))

    tree.plot\_tree(dtree, feature\_names=features, class\_names=['NO', 'YES'], filled=True)

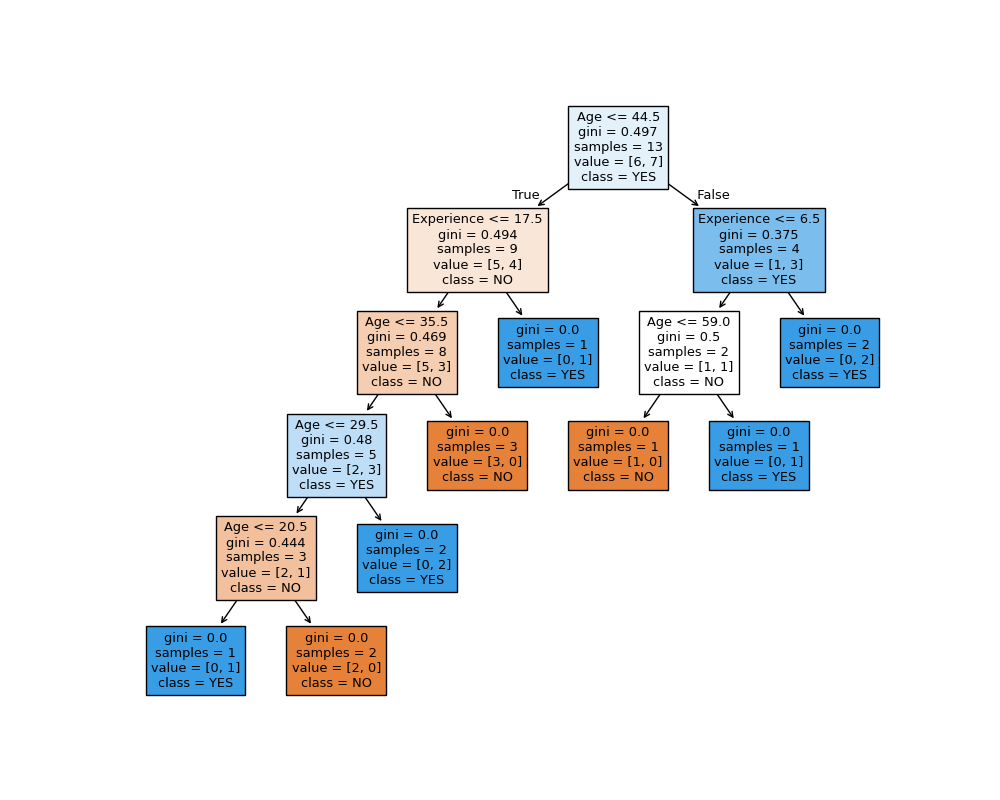
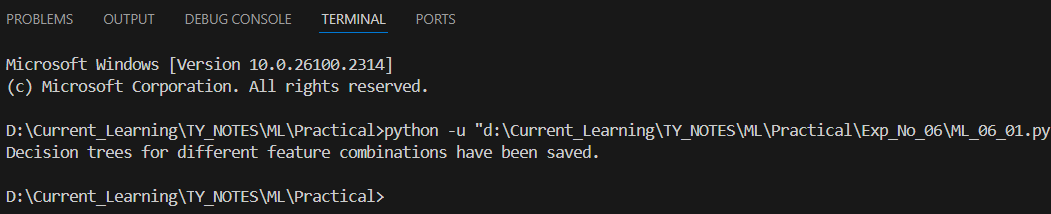
    # Save the plot with a unique name for each feature combination

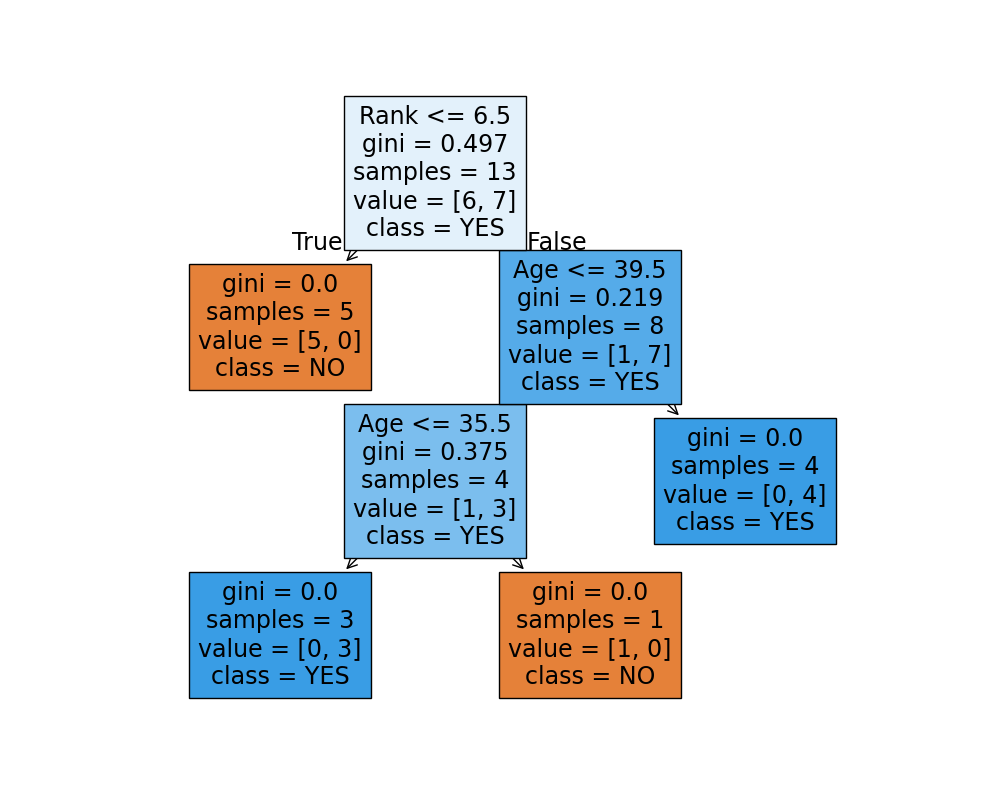
    plt.savefig(f"decision\_tree\_{i+1}.png")

    plt.close()

print("Decision trees for different feature combinations have been saved.")

**Output:**

****

****

**Experiment No: 07**

**Code:**

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

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

# Load the data

data = pd.read\_csv(r'D:\Current\_Learning\TY\_NOTES\ML\Practical\Exp\_No\_07\Experiment7.csv')

# Check for missing values in categorical columns

data.fillna({'Sex': 'unknown', 'Embarked': 'unknown'}, inplace=True)

# data['Sex'] = data['Sex'].fillna('unknown')

# data['Embarked'] = data['Embarked'].fillna('unknown')

# Encode categorical variables

label\_encoder = LabelEncoder()

data['Sex'] = label\_encoder.fit\_transform(data['Sex'])

data['Embarked'] = label\_encoder.fit\_transform(data['Embarked'])

# Define features and target variable

X = data.drop(columns=['Survived'])

y = data['Survived']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Initialize and train the Random Forest Classifier

rf\_model = RandomForestClassifier(random\_state=42, n\_estimators=100)

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

# Make predictions

y\_pred = rf\_model.predict(X\_test)

# Evaluate the model

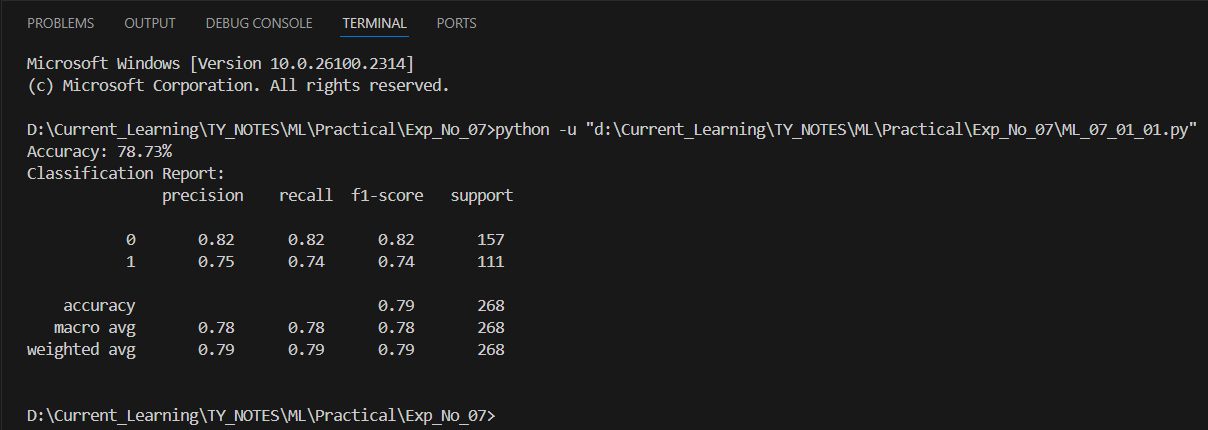
accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")

print("Classification Report:\n", report)

**Output:**



**Experiment No: 08**

**Code:**

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import warnings

# Ignore warnings for cleaner output

warnings.filterwarnings('ignore')

# Load the dataset

glass = pd.read\_csv(r"D:\Current\_Learning\TY\_NOTES\ML\Practical\Exp\_No\_08\Experiment8.csv")

# Print dataset information

print("Shape of dataset:", glass.shape)

print("Null values per column:\n", glass.isnull().sum())

# Plot distribution of Glass Types

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

sns.countplot(x='Type', data=glass, color='red')

plt.title("Distribution of Glass Types")

plt.xlabel("Type")

plt.ylabel("Count")

plt.show()

# Initialize Gaussian Naive Bayes model

nb = GaussianNB()

# Separate features and target variable

X = glass.drop(columns=['Type'])

y = glass['Type']

# Split 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=4)

# Train the model on the training data

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

# Make predictions on the test set

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

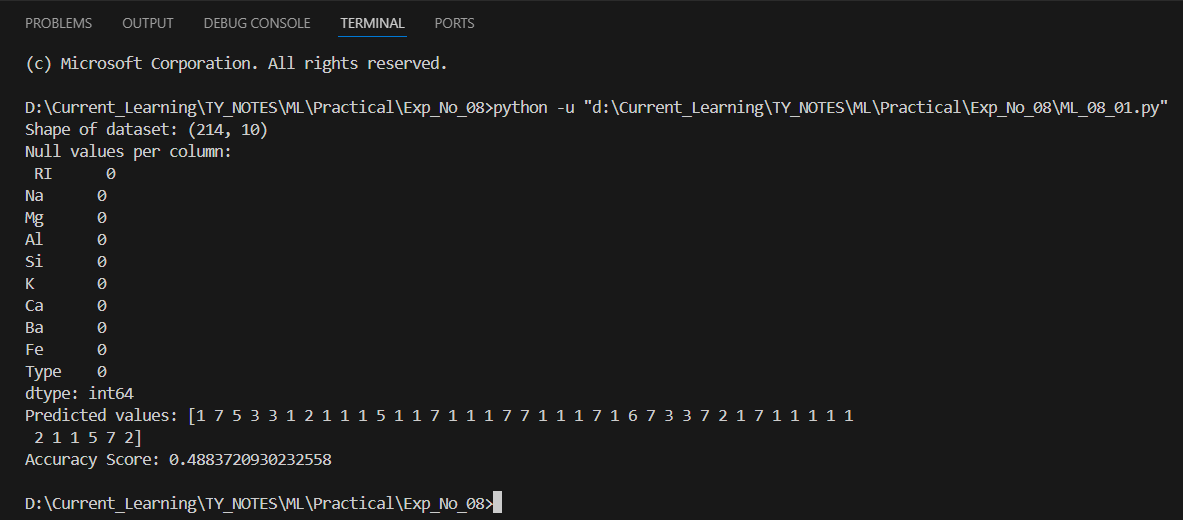
print("Predicted values:", y\_pred)

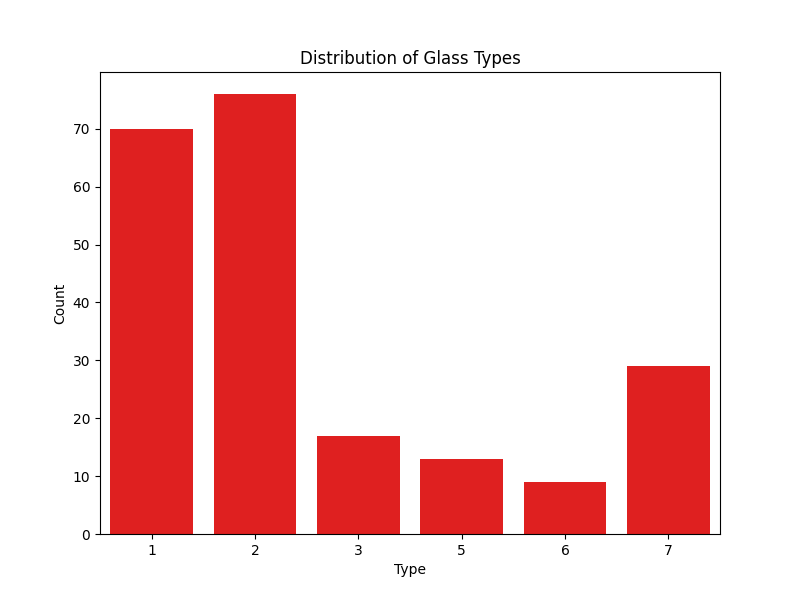
# Calculate and print accuracy

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

print("Accuracy Score:", accuracy)

**Output:**





**Experiment No: 09**

**Code:**

import pandas as pd

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

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

import matplotlib

matplotlib.use('Agg')  # Use non-interactive backend

import matplotlib.pyplot as plt

import seaborn as sns

# Load the data

data = pd.read\_csv(r'D:\Current\_Learning\TY\_NOTES\ML\Practical\Exp\_No\_09\Experiment9.csv')

# Print columns to verify

print("Columns in dataset:", data.columns)

# Encode categorical variables

label\_encoder = LabelEncoder()

if 'Gender' in data.columns:

    data['Gender'] = label\_encoder.fit\_transform(data['Gender'])

# Select features and the target variable for KNN

X\_knn = data[['Age', 'Gender']]

y\_knn = data['Purchased']

# Split data into training and testing sets for KNN

X\_train\_knn, X\_test\_knn, y\_train\_knn, y\_test\_knn = train\_test\_split(X\_knn, y\_knn, test\_size=0.3, random\_state=42)

# Initialize and train the KNN classifier

knn\_classifier = KNeighborsClassifier(n\_neighbors=5)

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

# Make predictions on the test set

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

# Generate and display the confusion matrix

conf\_matrix\_knn = confusion\_matrix(y\_test\_knn, y\_pred\_knn)

plt.figure(figsize=(8, 5))

sns.heatmap(conf\_matrix\_knn, annot=True, fmt="d", cmap="Blues", cbar=False,

            xticklabels=['Not Purchased', 'Purchased'], yticklabels=['Not Purchased', 'Purchased'])

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('KNN Classification Confusion Matrix')

# Save the figure instead of showing it

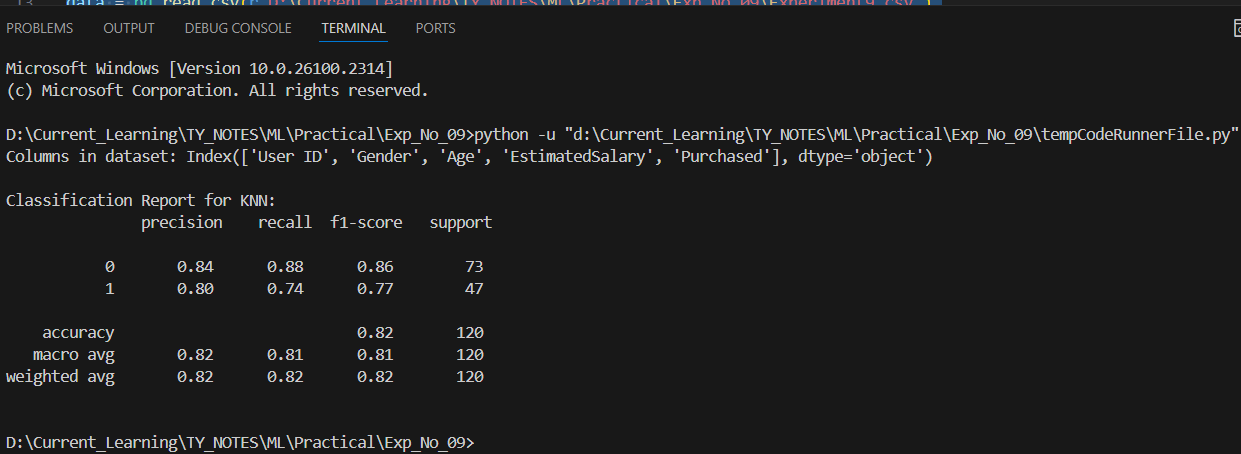
plt.savefig('knn\_confusion\_matrix.png')

plt.close()

# Print classification report for additional performance metrics

print("\nClassification Report for KNN:\n", classification\_report(y\_test\_knn, y\_pred\_knn))

**Output:**





**Experiment No: 10**

**Code:**

import pandas as pd

from sklearn import svm

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import accuracy\_score

# Load your dataset

data = pd.read\_csv(r'D:\Current\_Learning\TY\_NOTES\ML\Practical\Exp\_No\_10\Experiment10.csv')

# Encode 'species' if it's categorical

if data['Species'].dtype == 'object':

    encoder = LabelEncoder()

    data['Species'] = encoder.fit\_transform(data['Species'])

# Define features and target

X = data[['Weight', 'Width']]

y = data['Species']

# 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)

scaler = StandardScaler()

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

X\_test = scaler.transform(X\_test)

# Create and train the SVM classification model

model = svm.SVC()

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

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

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

print(f"Model Accuracy: {accuracy:.2f}")

y\_pred\_labels = encoder.inverse\_transform(y\_pred)

print("SVM Predictions (Species):", y\_pred\_labels)

**Output:**

