|  |  |  |
| --- | --- | --- |
| ACAD-DI-86 | **Lab Manual** | Academic Year: 2024 - 25 |
| Rev : 01 | Semester: V |
| Date: 02.04.2021 |

**Subject:** Machine Learning Lab (BTAIL506)

**List of Experiments**

|  |  |
| --- | --- |
| **Exp. No.** | **Name of Experiments** |
|  | 1. Python Libraries for Data Science-  a. Pandas Library  b. Numpy Library  c. Scikit Learn Library  d. Matplotlib |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Experiment No: 01**

**Aim:**

Python Libraries for Data Science-

* 1. Pandas Library
  2. Numpy Library
  3. Matplotlib

**Objectives:** To demonstrate basic Python programming concepts

**Theory:**

In this lab exercise, we explore fundamental libraries used in data science - Pandas, NumPy, and Matplotlib - demonstrating their functionalities through practical examples.

#### 1. Pandas Library:

Pandas is a powerful and versatile library for data manipulation and analysis in Python. It provides high-level data structures like DataFrames and Series, making it efficient to handle and process complex datasets.

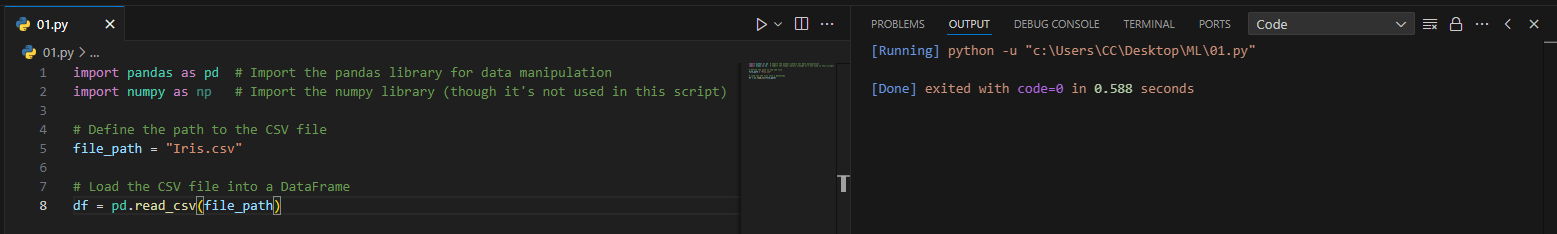
**Key Functionalities of Pandas:**

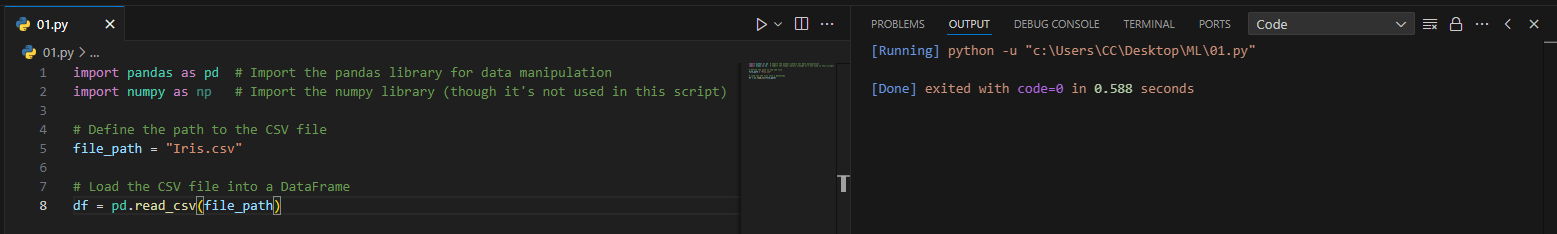
* **Data Loading:** Pandas allows loading data from various file formats like CSV, Excel, JSON, and databases into DataFrames for easy manipulation.
* **Data Cleaning and Transformation:** It offers tools for cleaning and preparing data, including handling missing values, filtering, and data type conversions.
* **Data Analysis and Exploration:** Pandas provides functions for summarizing data (mean, standard deviation), calculating statistics on groups of data, and performing exploratory data analysis.
* **Data Visualization:** While not its core functionality, Pandas interacts with visualization libraries like Matplotlib and Seaborn for creating informative plots and charts.

**Code :**

1. **Importing Libraries:**

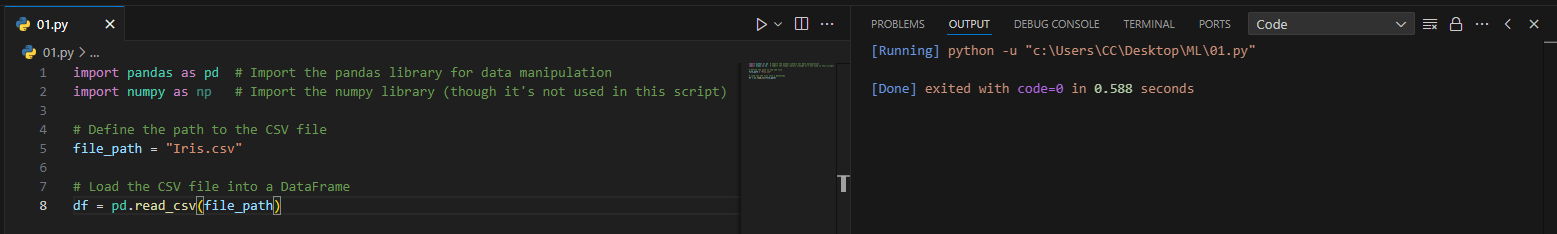
* import pandas as pd: We import Pandas as pd, which is a common alias used by the community for convenience.
* import numpy as np: Numpy is imported as np, though it's not used in this script, it's often used alongside Pandas for numerical operations.

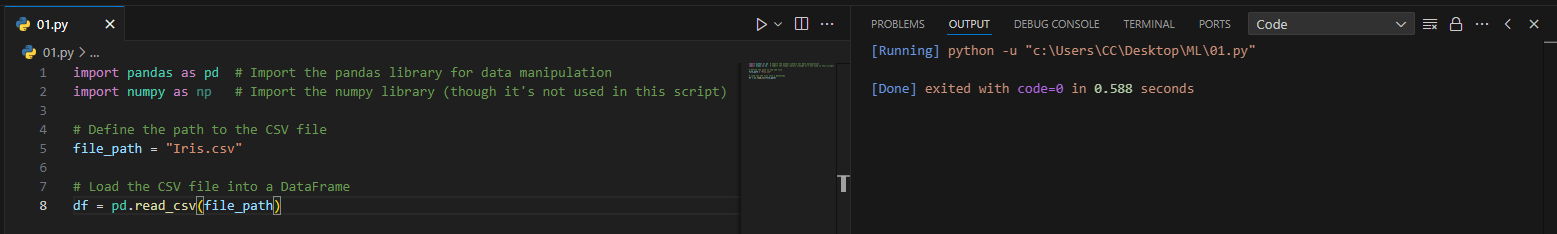




1. **Loading Data:**

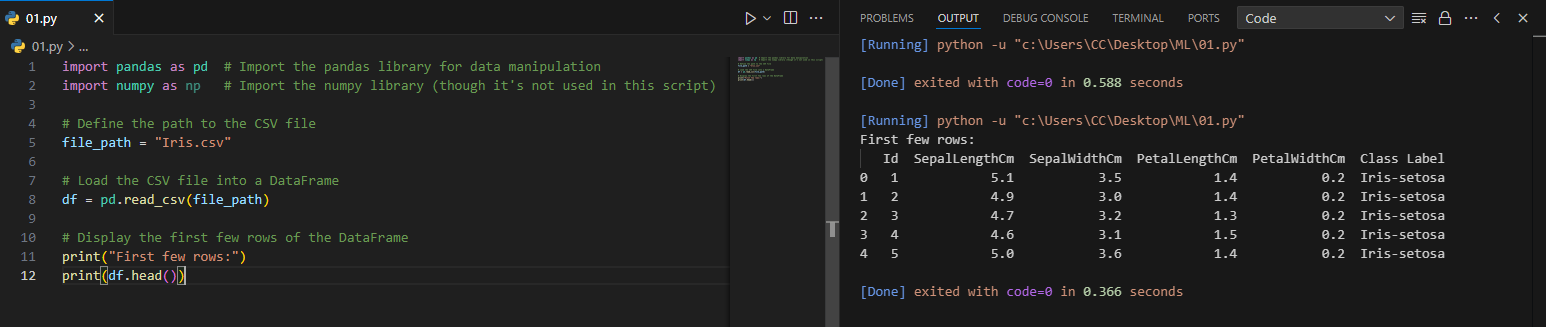
* The CSV file Iris.csv is loaded into a DataFrame using pd.read\_csv(file\_path), where file\_path is the path to the CSV file.

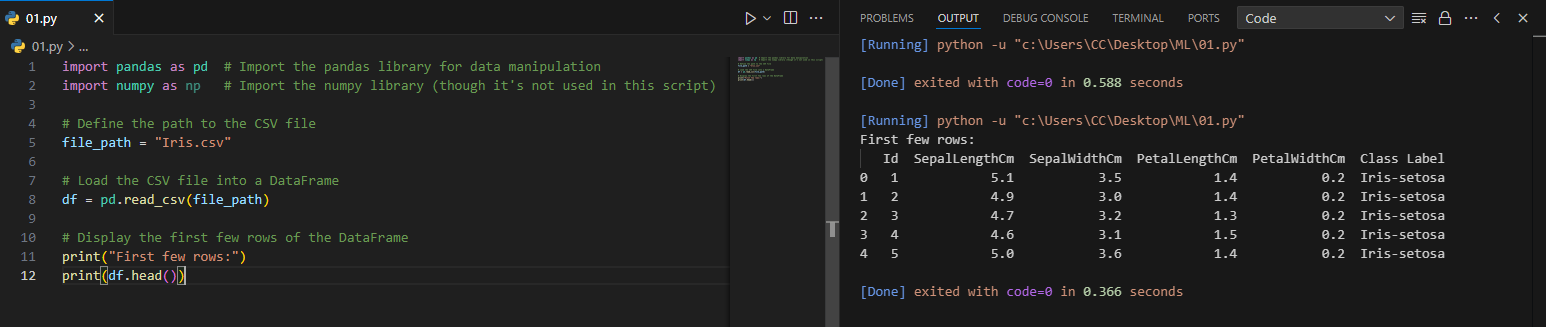




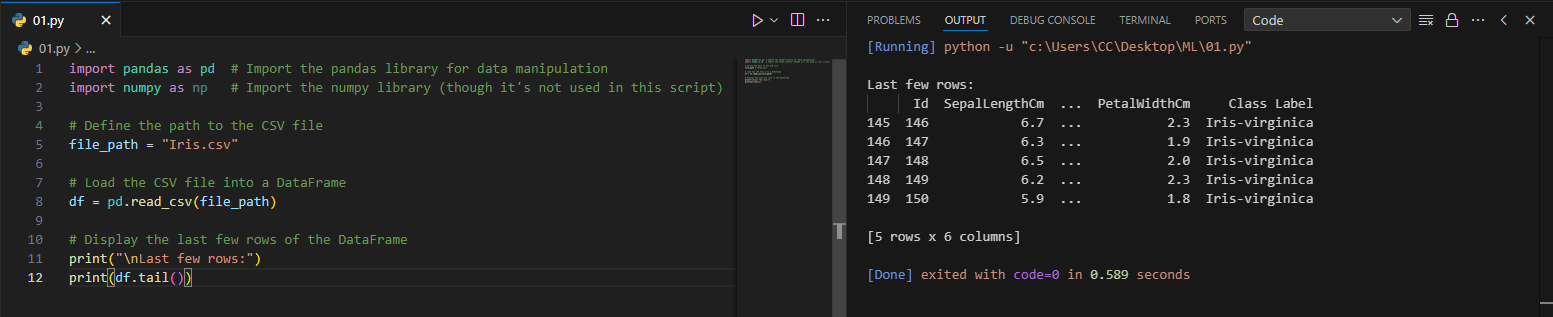
1. **Data Exploration:**

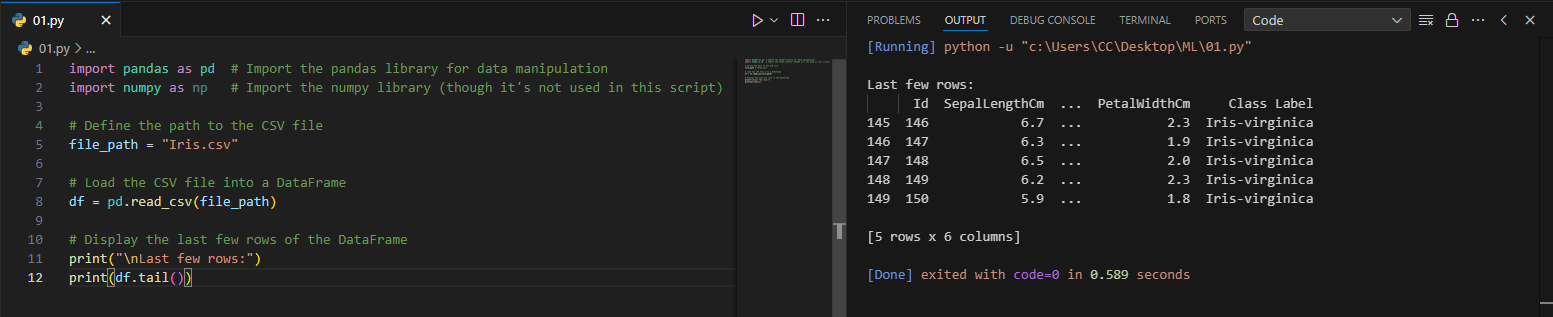
* df.head() displays the first few rows of the DataFrame, giving a quick glance at the data.





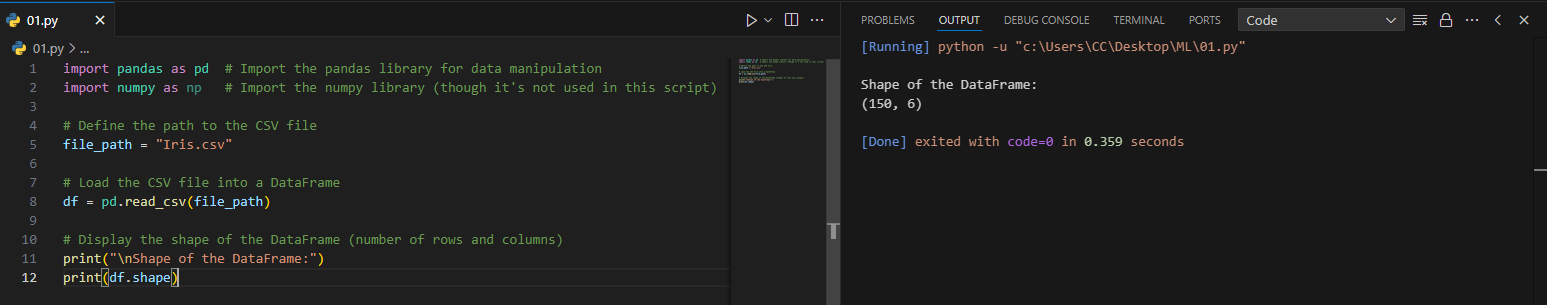
* df.tail() shows the last few rows, which is useful for checking the end of the data.

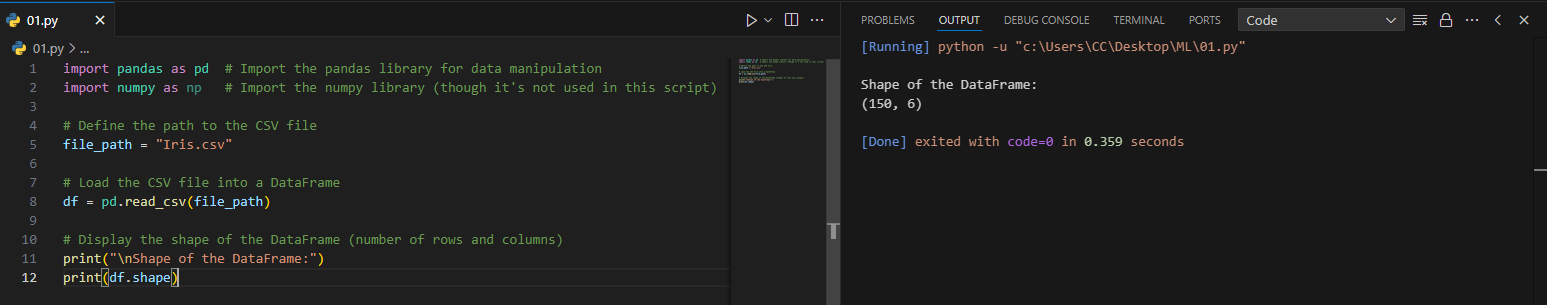




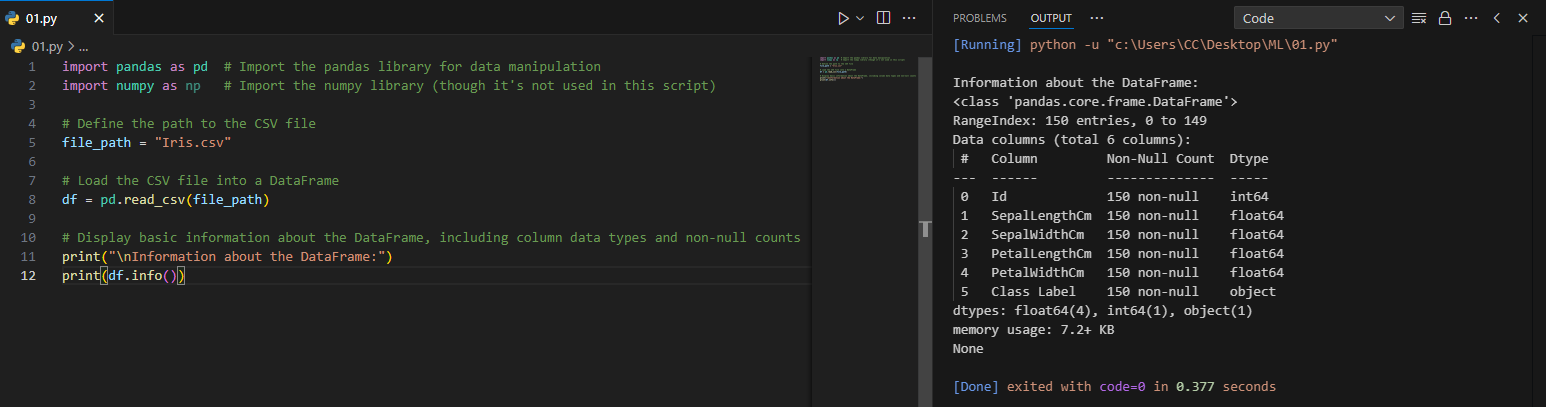
1. **Data Structure and Information:**

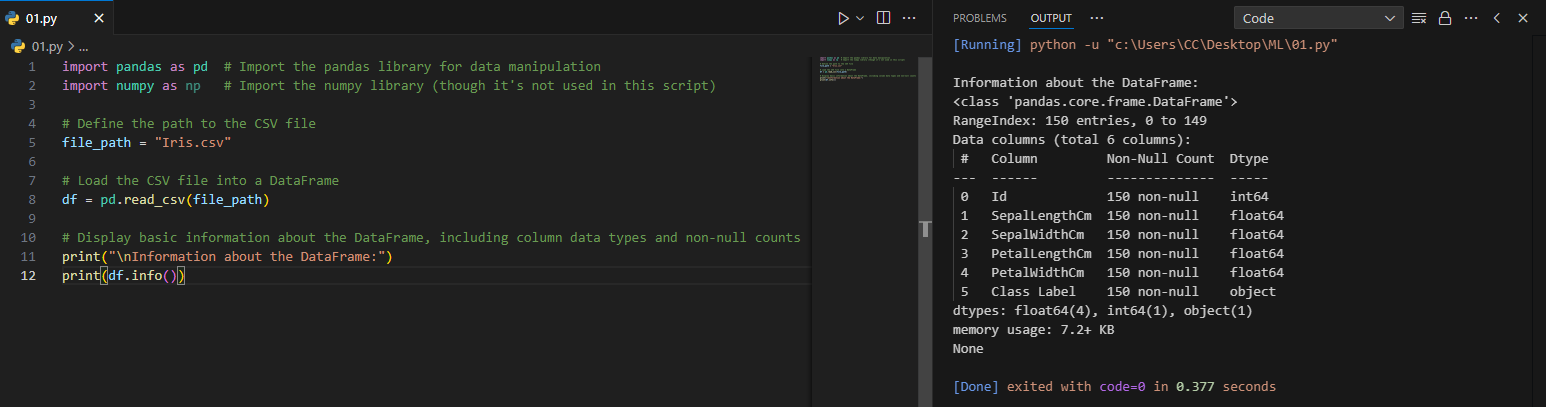
* df.shape gives the number of rows and columns, helping understand the size of the dataset.





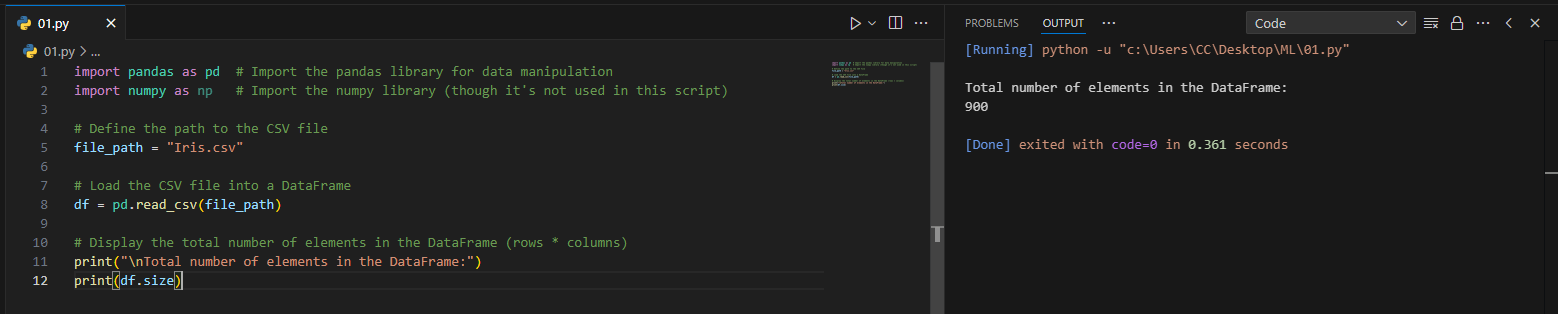
* df.info() provides a summary of the DataFrame, including data types and the count of non-null values for each column.

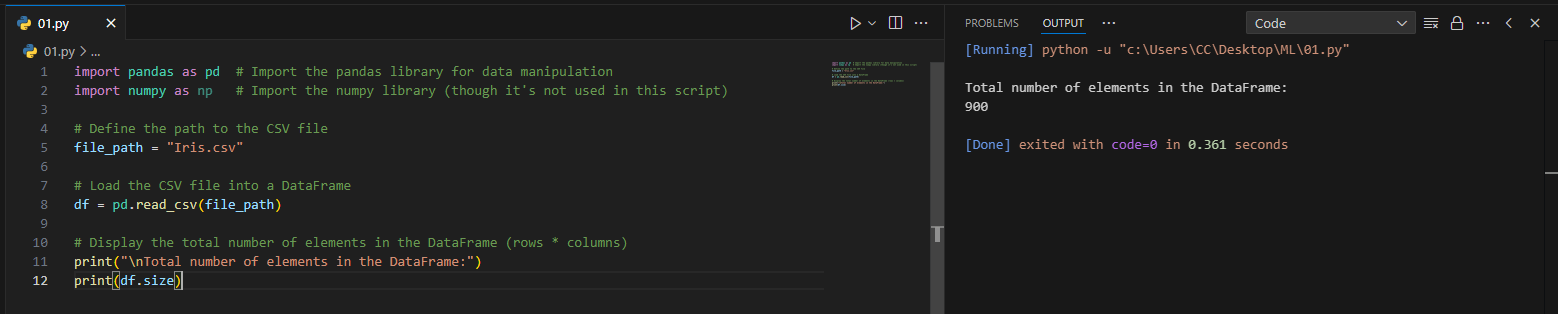




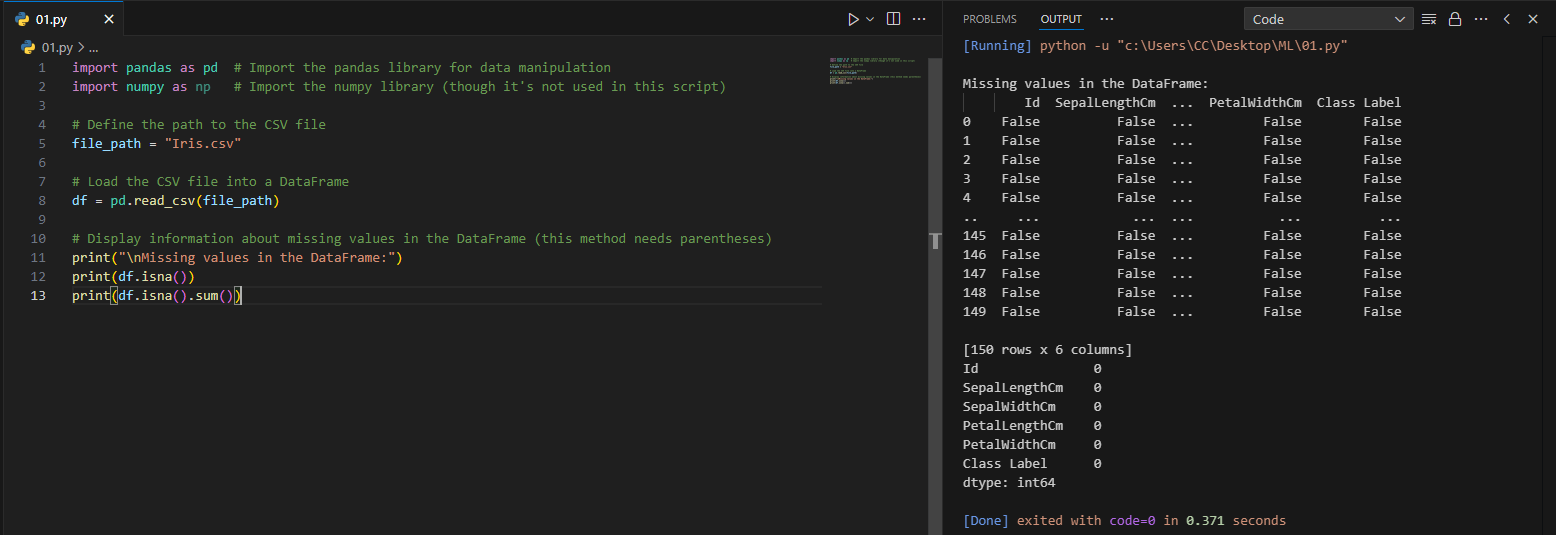
1. **Data Size and Missing Values:**

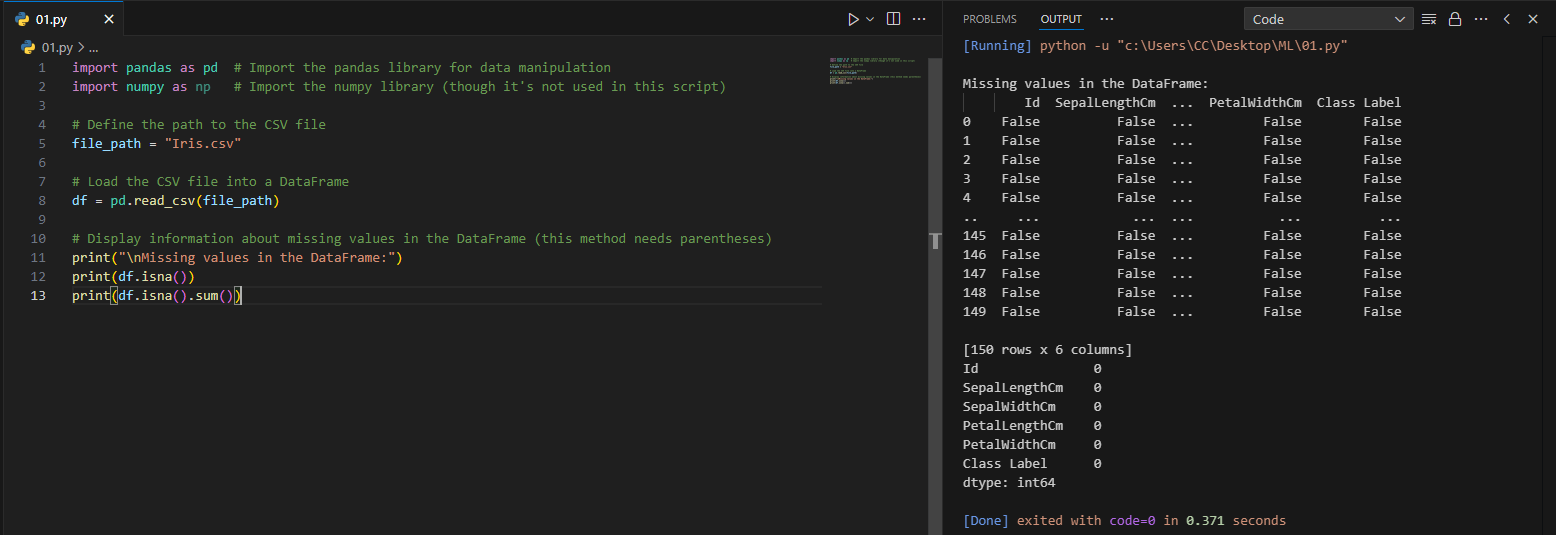
* df.size returns the total number of elements in the DataFrame, which is the product of rows and columns





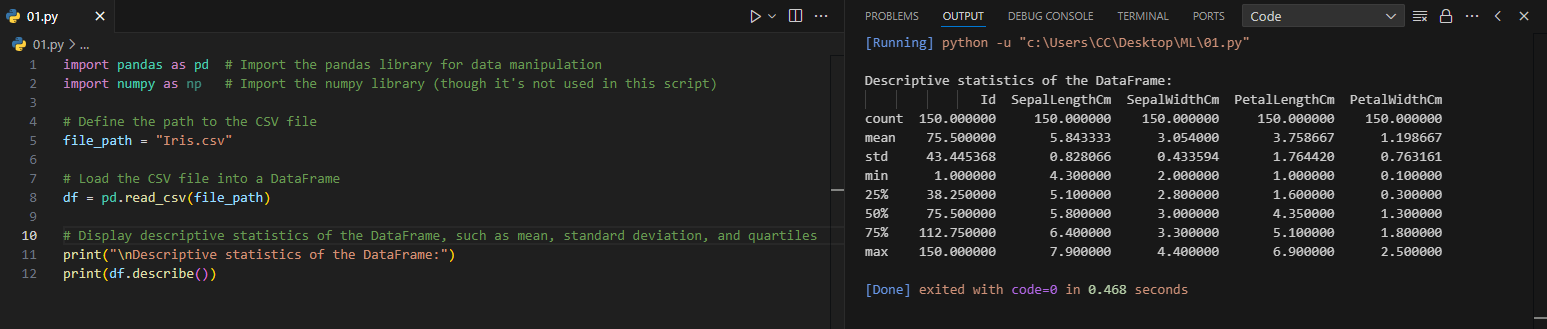
* df.isna() checks for missing values in the DataFrame and returns a DataFrame of the same shape with boolean values (True for missing values). df.isna().sum() gives the total count of missing values for each column.

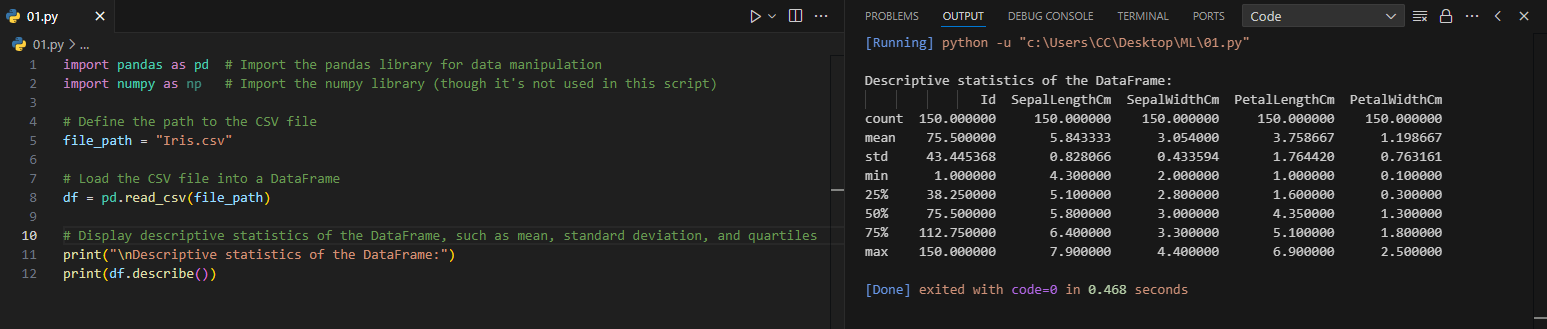




1. **Descriptive Statistics:**

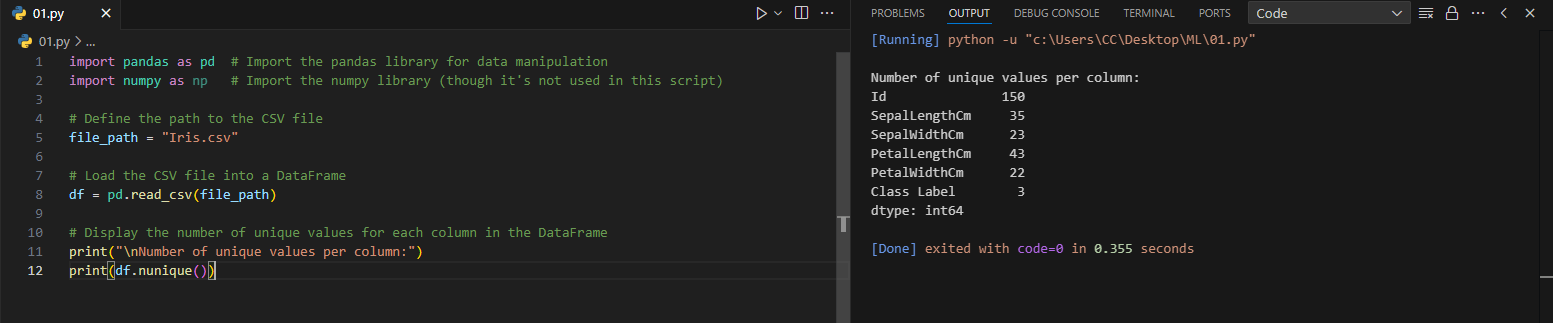
* df.describe() generates descriptive statistics like mean, standard deviation, min, max, and percentiles for each numerical column.

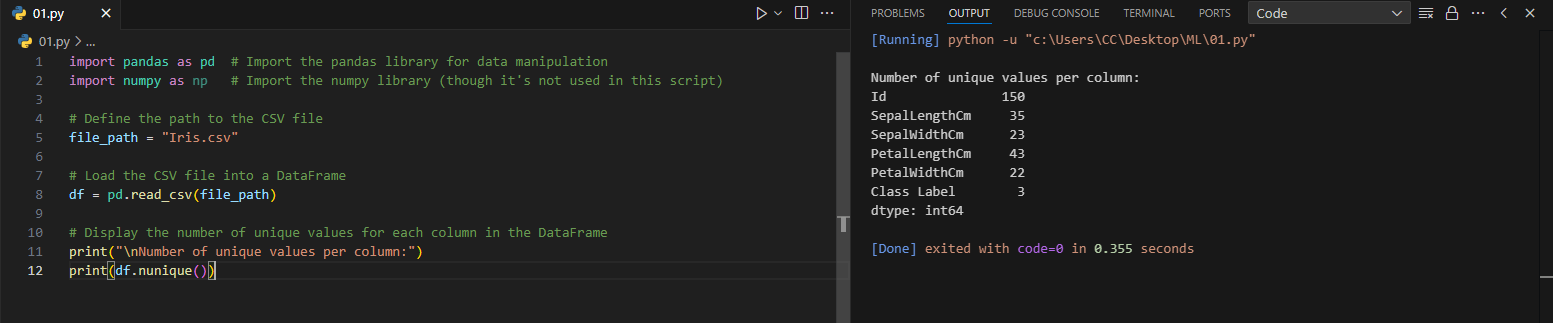




1. **Unique Values:**

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





#### 2. NumPy Library:

NumPy (Numerical Python) is a foundational library for scientific computing in Python. It provides powerful tools for working with multidimensional arrays, linear algebra operations, and high-performance numerical computations.

**Key Functionalities of NumPy:**

* **Array Creation and Manipulation:** NumPy offers efficient ways to create, manipulate, and perform operations on multidimensional arrays. These arrays are fundamental building blocks for data analysis tasks.
* **Linear Algebra Operations:** NumPy provides functions for linear algebra operations like matrix multiplication, solving linear equations, and finding eigenvalues/eigenvectors.
* **Random Number Generation:** It offers functions for generating random numbers with various distributions (e.g., normal, uniform) crucial for data simulation and machine learning.
* **Integration with Pandas:** Pandas heavily relies on NumPy for its underlying array operations.

**Code: -**

**1. Creating a NumPy Array**

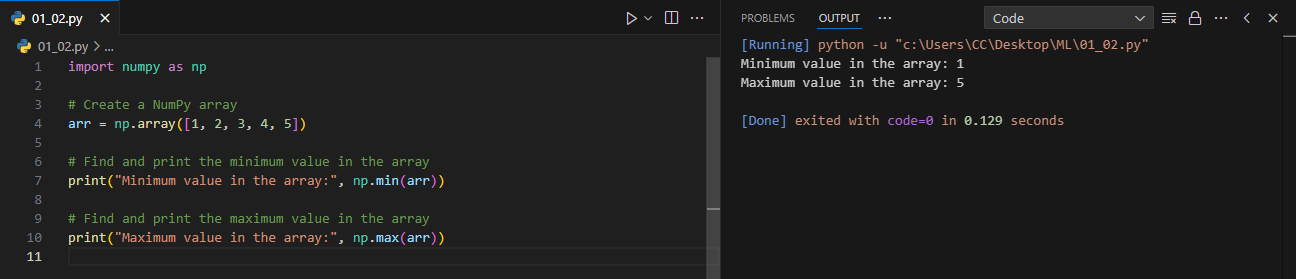
* np.array(): Creates a NumPy array from a list or other iterable, enabling array operations.

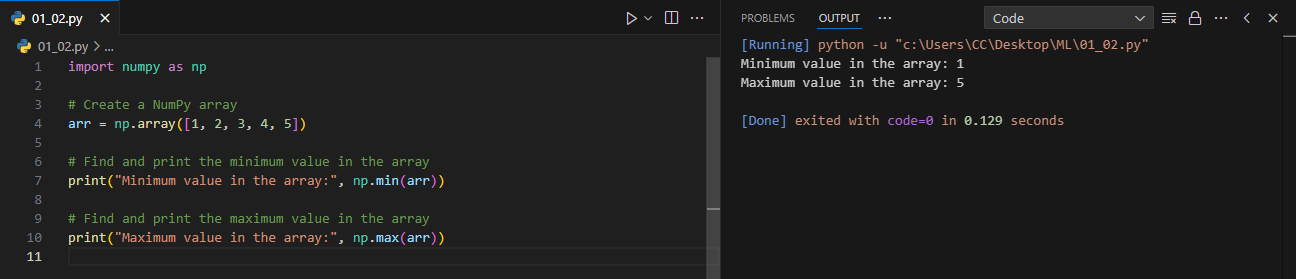
**2. Finding Minimum Value**

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

**3. Finding Maximum Value**

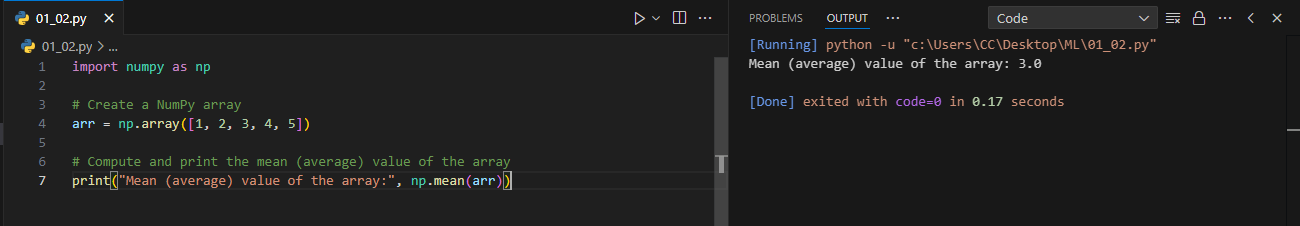
* np.max(): Returns the maximum value from the array.

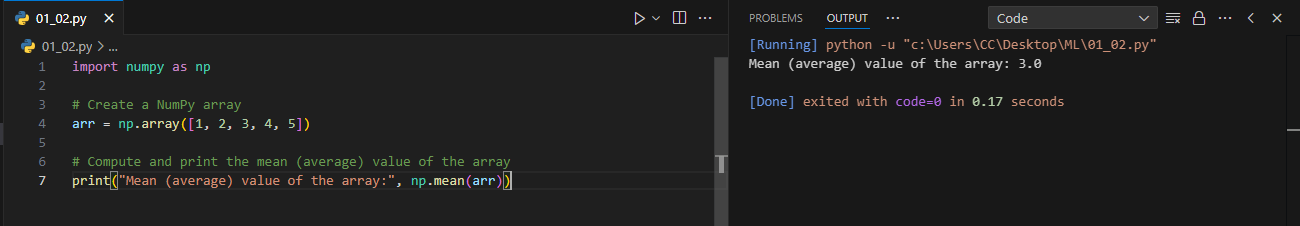




**4. Computing Mean**

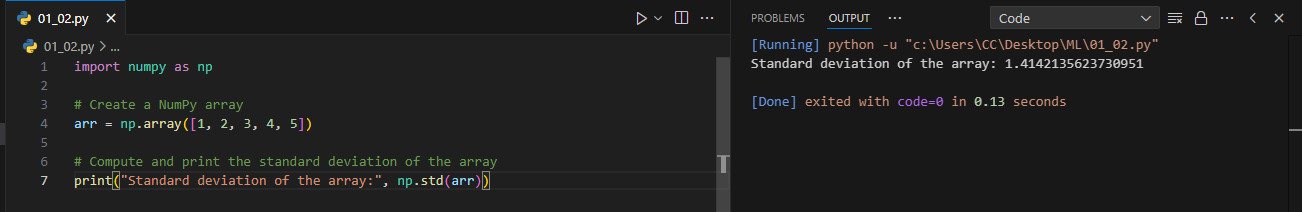
* np.mean(): Computes the arithmetic mean (average) of the elements in the array.

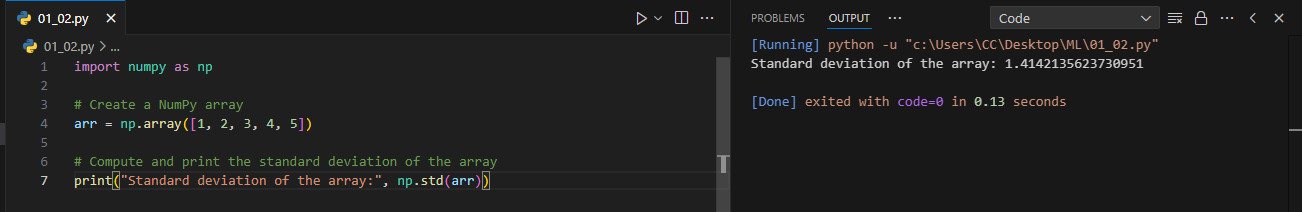




**5. Computing Standard Deviation**

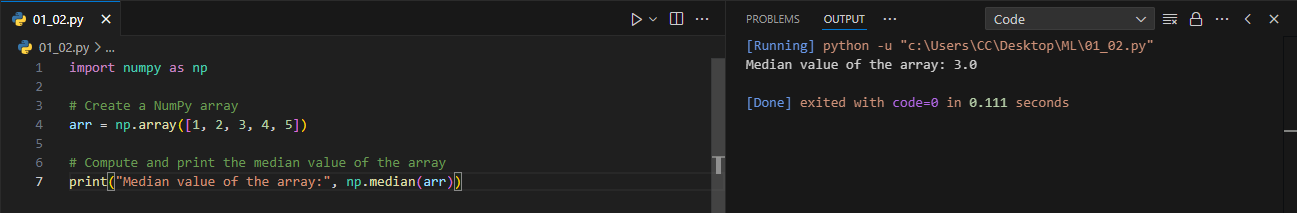
* np.std(): Calculates the standard deviation, which measures the amount of variation or dispersion of a set of values.

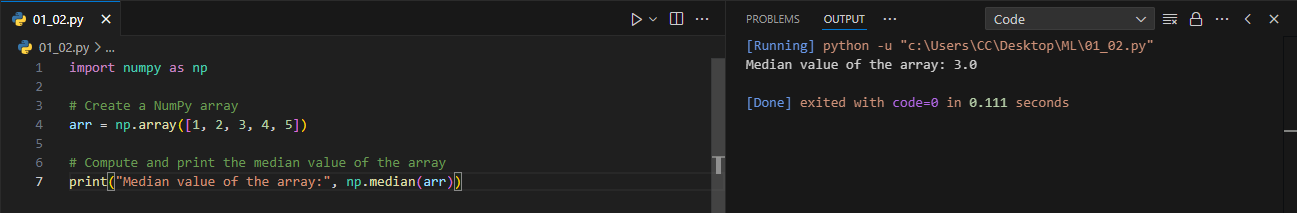




**6. Computing Median**

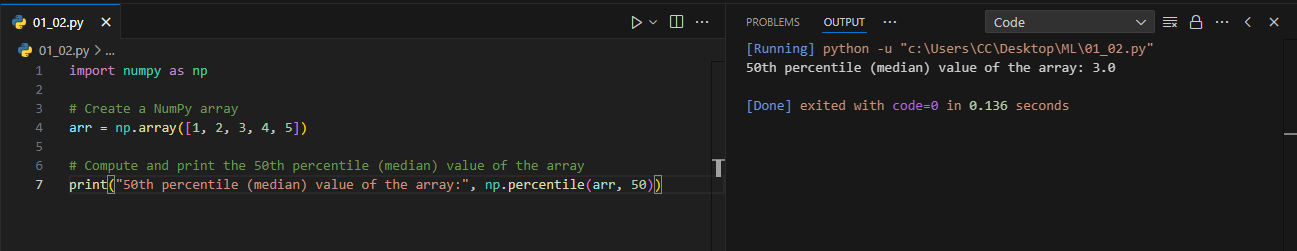
* np.median(): Finds the median value, which is the middle value when the array is sorted.

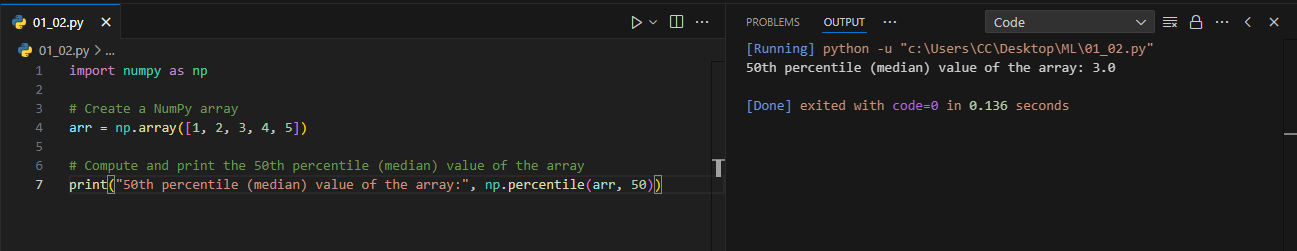




**7. Computing Percentile**

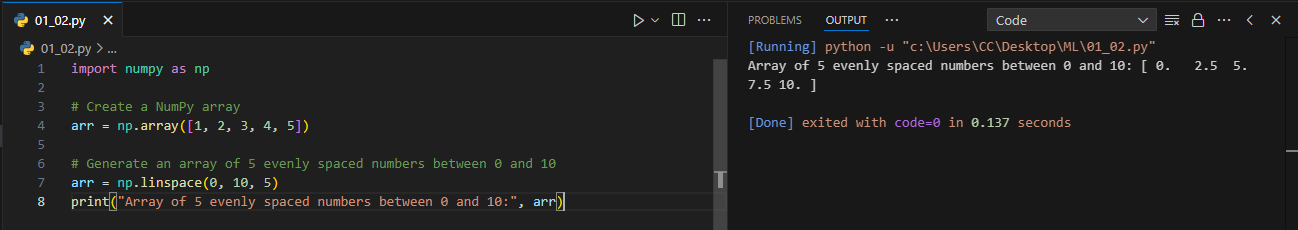
* np.percentile(): Returns the nth percentile of the array. The 50th percentile is the median.

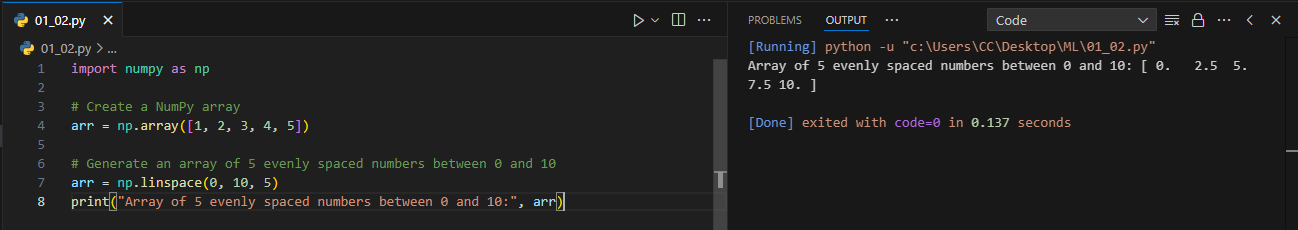




**8. Generating Linearly Spaced Numbers**

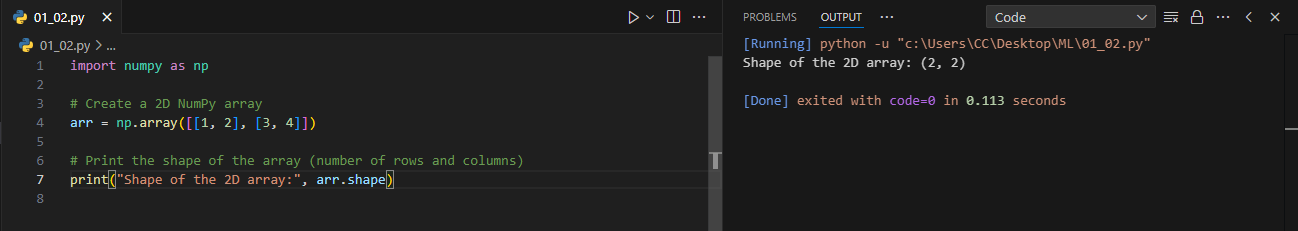
* np.linspace(): Generates an array of evenly spaced numbers over a specified range.

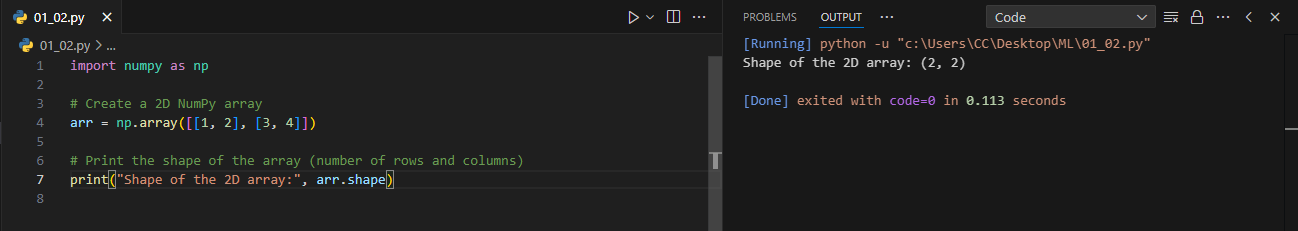




**9. Creating a 2D Array and Getting Array Shape**

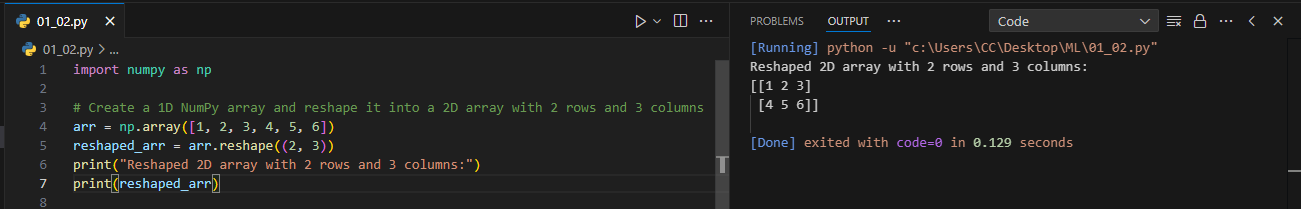
* np.array(): Used again to create a 2D NumPy array.
* .shape: Returns a tuple representing the dimensions (number of rows and columns) of the array.

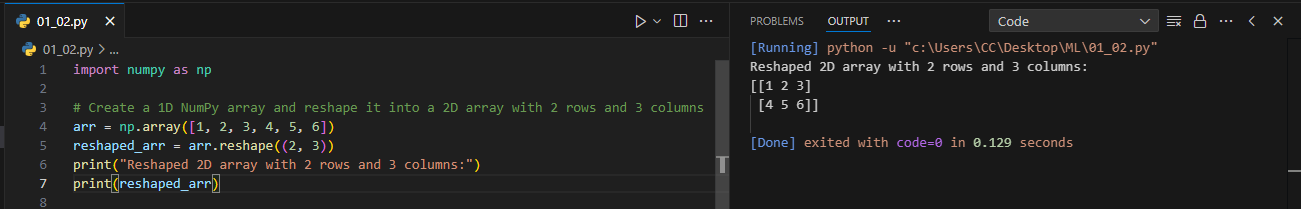




**10. Reshaping an Array**

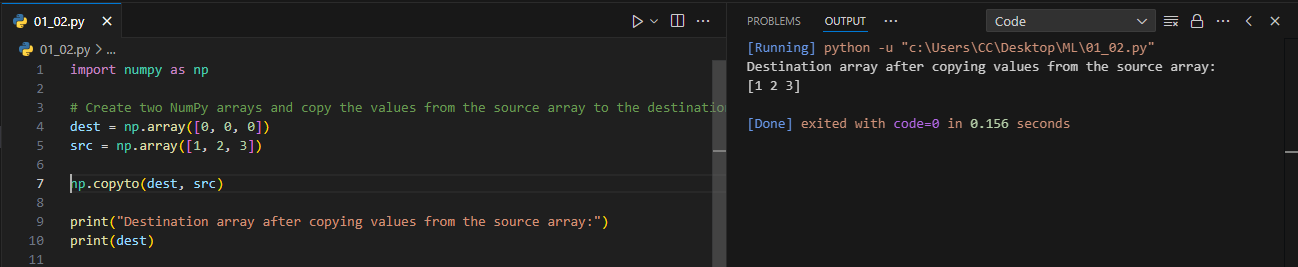
* .reshape(): Changes the shape of an array without changing its data, here reshaping a 1D array into a 2D array.

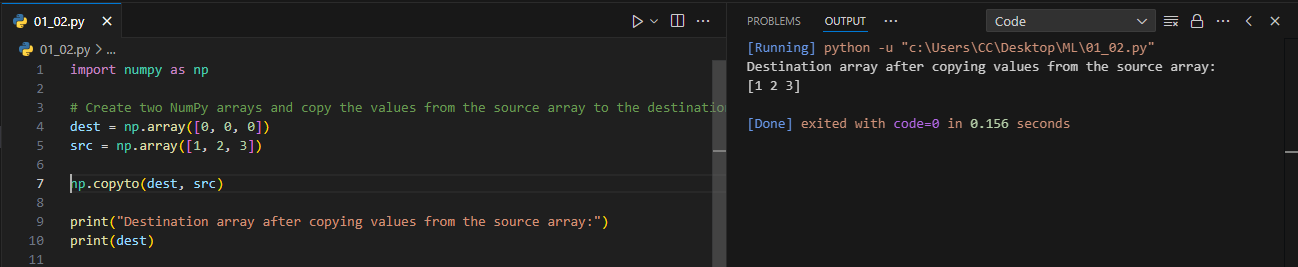




**11. Copying Values Between Arrays**

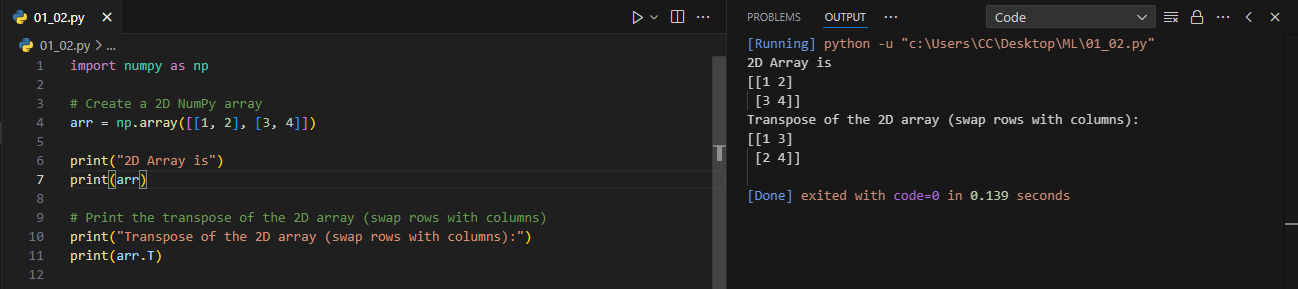
* np.copyto(): Copies values from one array (source) to another (destination), modifying the destination array in place.

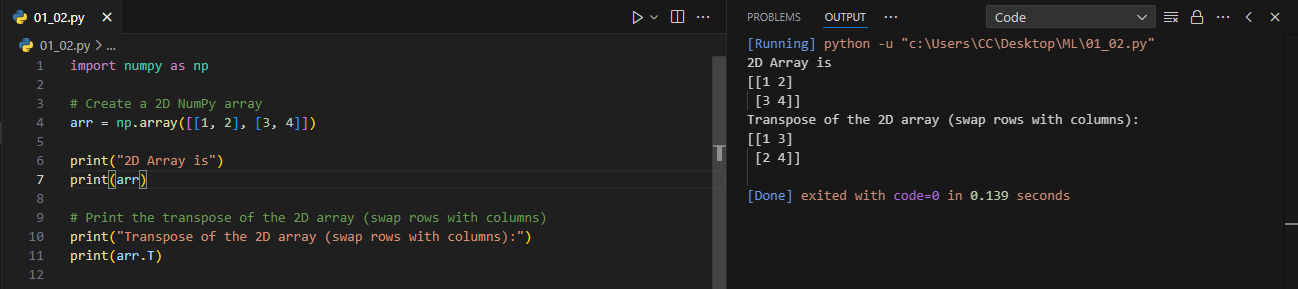




**12. Transposing an Array**

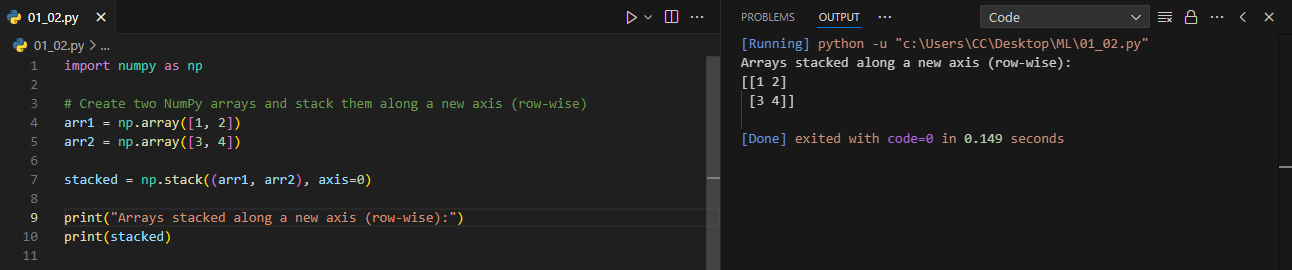
* .T: Transposes the array, swapping rows with columns.

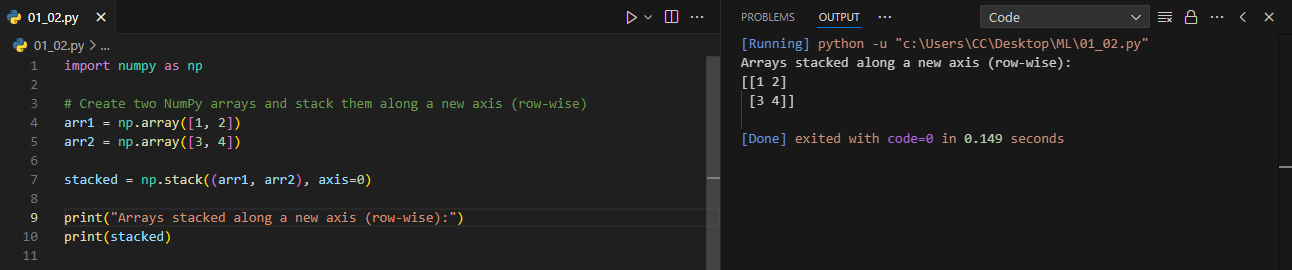




**13. Stacking Arrays**

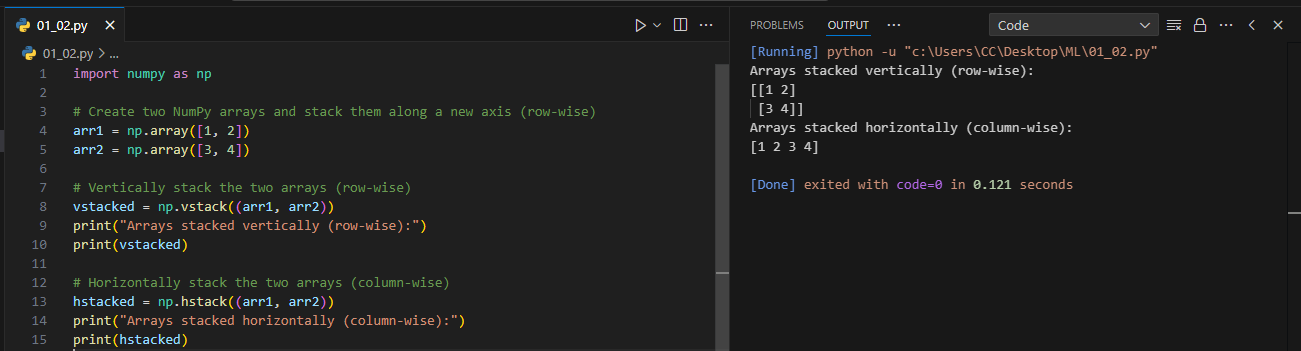
* np.stack(): Combines arrays along a new axis, here stacking them row-wise.

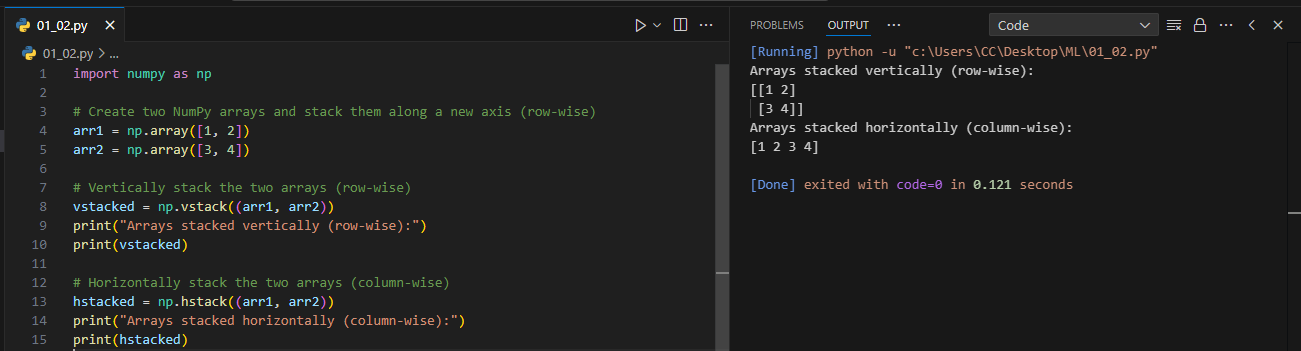




**14. Vertical Stacking and Horizontal Stacking**

* 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*

Matplotlib is a popular plotting library for Python that simplifies the creation of static, animated, and interactive visualizations. It is widely used in scientific computing, data analysis, and machine learning.

#### Key Functionalities:

* **Data Visualization**: Matplotlib allows users to create a variety of visualizations, including line charts, bar charts, scatter plots, and histograms, making it easier to identify and understand data patterns.
* **Customization**: The library offers extensive options for customizing plots, such as modifying colors, labels, legends, titles, and line styles, enabling users to create professional and tailored graphics.
* **Integration with NumPy and Pandas**: Matplotlib seamlessly integrates with NumPy arrays and Pandas DataFrames, facilitating efficient visualization of data stored in these formats.
* **Subplots**: Users can create multiple plots within a single figure using the subplots() function, which is useful for comparing different datasets or presenting related information together.
* **Interactive Features**: The library supports interactive plotting, allowing users to zoom, pan, and save figures, enhancing the experience of data exploration and analysis.

**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**

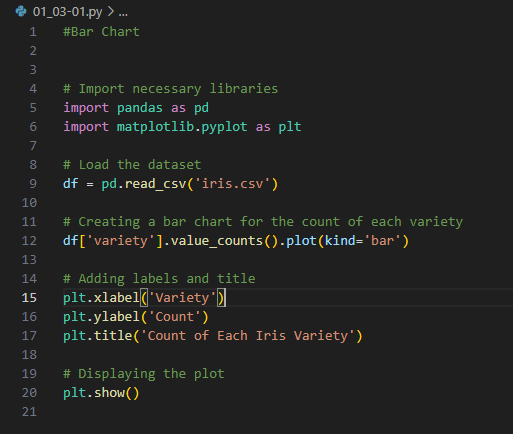
* plt.title(): Assigns a title to the plot, summarizing the content or purpose of the visualization.

**5. Displaying the Plot**

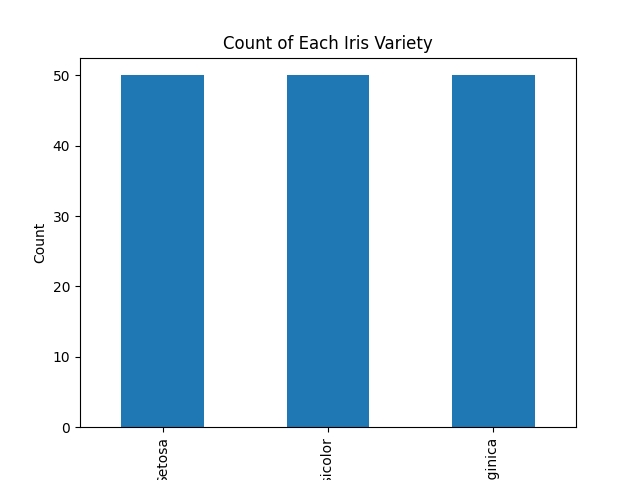
* plt.show(): Renders and displays the current figure or plot. This function is essential for visualizing the created plot in a standalone window or inline, depending on the environment.

**Code: -**

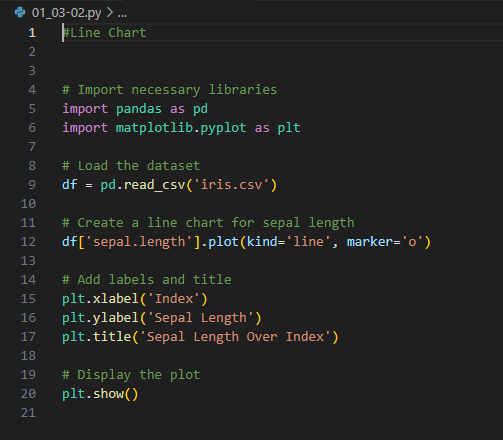
* 1. **Bar Chart**



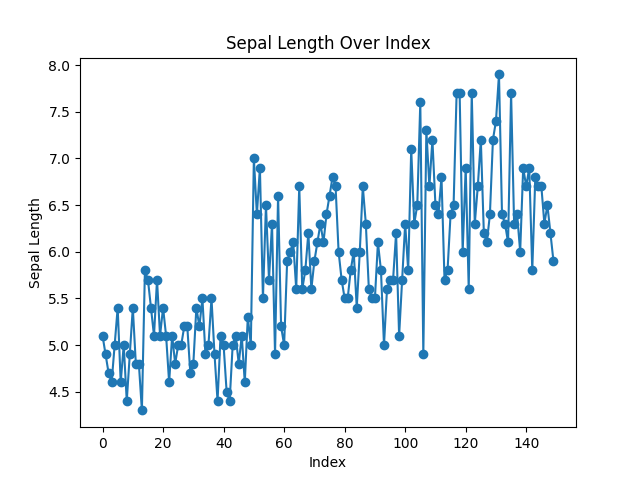
**Output: -**

****

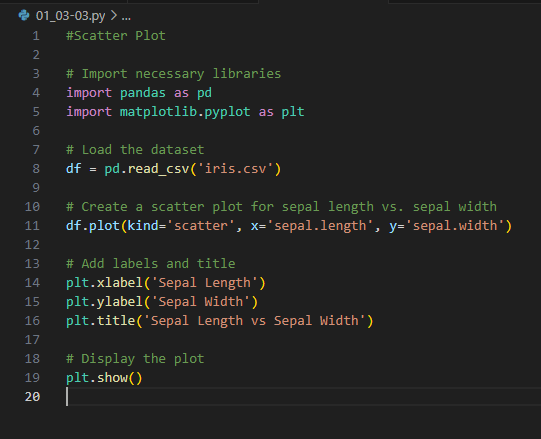
* 1. **Line Chart**



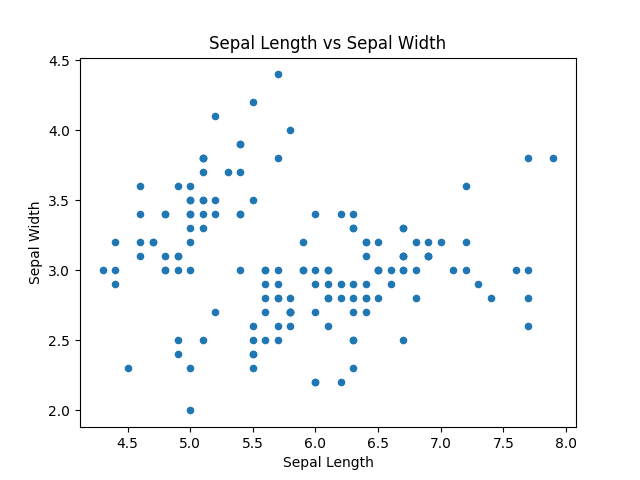
**Output: -**



* 1. **Scatter Plot**

****

**Output:-**



**Experiment No: 02**

**Aim:**

**Objectives:**

**Theory:**

1. **Introduction**  
    This experiment aims To evaluate the performance of a machine learning model using a Random Forest classifier on the Breast Cancer dataset. We will analyze the model's predictions through confusion matrices and accuracy metrics.
2. **Dataset**  
    Breast Cancer Dataset: The dataset contains features of breast cancer tumors, labeled as malignant (1) or benign (0). It is used for binary classification.
3. **Data Preparation:**
   1. Independent Variables (X): Tumor characteristics (e.g., radius, texture, perimeter).
   2. Dependent Variable (y): Class labels indicating tumor type (malignant or benign).
4. **Data Splitting**
   1. The dataset is split into training and testing sets using an 80-20 ratio. The training set is used to train the model, while the testing set evaluates its performance.
   2. Function Used: train\_test\_split() from sklearn.model\_selection.
5. **Model Training**
   1. Algorithm: Random Forest Classifier is employed due to its robustness and ability to handle high-dimensional data.
   2. Training: The classifier is fitted to the training data using the fit() method.
6. **Predictions**
   1. After training, the model predicts the labels for the testing data using the predict() method.
7. **Performance Evaluation**
   1. Accuracy Score: The accuracy of the classifier is calculated to assess how many predictions were correct. This is done using accuracy\_score() from sklearn.metrics.
   2. Confusion Matrix:  
       A confusion matrix is generated to visualize the performance of the model. It displays the true positive, true negative, false positive, and false negative values.
   3. Function Used: confusion\_matrix() from sklearn.metrics and ConfusionMatrixDisplay for visualization.
8. **Results**  
   The output includes:
   * The shape of the training and testing datasets.
   * The accuracy of the classifier.
   * A plotted confusion matrix illustrating the classification results.

**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)

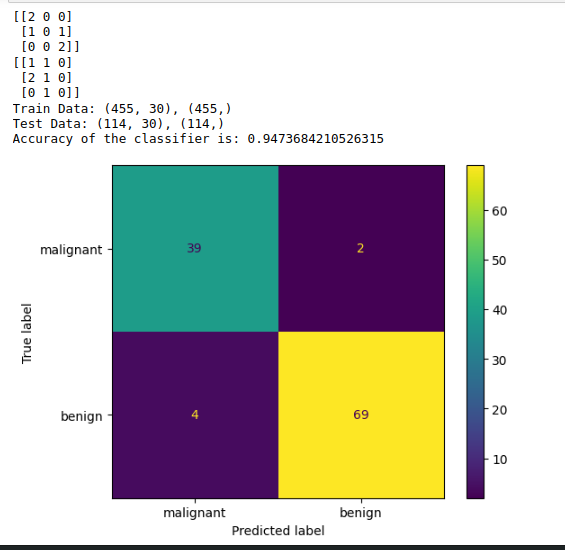
# Plot the confusion matrix

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

disp.plot()

plt.show()

**Output :-**

****

**Conclusion**

This experiment demonstrates how to implement a Random Forest classifier on the Breast Cancer dataset, providing insights into the model's effectiveness through accuracy and confusion matrix visualization.

**Experiment No: 03**

**Aim:**

**Objectives:**

**Theory:**

**Code 01:**

1. **Importing Libraries**
   1. pandas: For data manipulation and analysis.
   2. train\_test\_split: To split the dataset into training and testing sets.
   3. accuracy\_score: To evaluate the model's performance.
   4. KNeighborsClassifier: The algorithm used for classification.
2. **Loading the Dataset**
   1. The dataset is loaded from a CSV file named "diabetes.csv" using pd.read\_csv().
   2. The dataset contains various health-related features and a target variable indicating diabetes presence.
3. **Separating Features and Target Variable**
   1. The data is converted to a NumPy array, and features (X) are separated from the target variable (y).
   2. The shape of features and target variables is printed for verification.
4. **Splitting the Dataset**
   1. The dataset is split into training (70%) and testing (30%) sets using train\_test\_split().
   2. Random state is set to ensure reproducibility.
   3. The shapes of training and testing sets are printed to confirm the split.
5. **Initializing and Fitting the KNN Model**
   1. An instance of KNeighborsClassifier is created.
   2. The model is trained using the fit() method on the training data.
6. **Making Predictions**
   1. Predictions are made on the test set using the predict() method.
7. **Evaluating the Model**
   1. The accuracy of the model is calculated using accuracy\_score() by comparing predicted values with actual values.
   2. The accuracy is printed as a percentage to assess model performance.
8. **Dataframe Preview**
   1. The first few rows of the original dataset are printed to provide an overview of the data structure.

**Code 02:**

1. **Importing Libraries**
   1. numpy: For numerical operations and array manipulation.
   2. pandas: For creating and managing DataFrames.
2. **Defining Synthetic Data**
   1. A NumPy array X is created to represent features.
   2. A corresponding target variable y is defined as a one-dimensional array.
3. **Creating a DataFrame**
   1. The features are converted into a Pandas DataFrame for better handling.
   2. A new column 'Target' is added to the DataFrame, containing the target variable.
4. **Separating Features and Target Variable**
   1. Features (X) are selected by excluding the last column of the DataFrame.
   2. The target variable (y) is extracted as the last column.
5. **Splitting the Dataset**
   1. The dataset is split into training (70%) and testing (30%) sets using train\_test\_split().
   2. Random state is set for reproducibility.
6. **Printing the Datasets**
   1. The training and testing sets for features and targets are printed to verify the split and ensure the data is correctly prepared.

**Code 01 Functions :-**

1. **from sklearn.model\_selection import train\_test\_split**
   1. **Usage**: Imports the function to split datasets into training and testing sets.
   2. **Explanation**: This function helps us separate our data so we can train our model and test its performance on unseen data.
2. **from sklearn.metrics import accuracy\_score**
   1. **Usage**: Imports the function to calculate the accuracy of the model.
   2. **Explanation:** This function measures how many predictions made by the model are correct.
3. **from sklearn.neighbors import KNeighborsClassifier**
   1. **Usage**: Imports the KNN classifier model.
   2. **Explanation:** This initializes the K-Nearest Neighbors algorithm, which will be used for classification tasks.
4. **df.values**
   1. **Usage:** Converts the DataFrame into a NumPy array.
   2. **Explanation:** This allows us to access the underlying data as an array for easier slicing and manipulation.
5. **train\_test\_split(X, y, test\_size=0.3, random\_state=1)**
   1. **Usage**: Splits the data into training and testing sets.
   2. **Explanation**: This function divides our features (X) and target variable (y) into two parts: one for training the model and one for testing it, ensuring 30% of the data is used for testing.
6. **KNeighborsClassifier()**
   1. **Usage**: Initializes the KNN classifier.
   2. **Explanation**: This creates an instance of the KNN model, which we will train on our data.
7. **model.fit(X\_train, y\_train**)
   1. **Usage**: Trains the KNN model on the training data.
   2. **Explanation:** This function allows the model to learn from the training features and their corresponding labels.
8. **model.predict(X\_test)**
   1. **Usage**: Makes predictions on the testing data.
   2. **Explanation**: This function uses the trained model to predict the target variable for the unseen testing set.
9. **accuracy\_score(y\_test, y\_predict)**
   1. **Usage:** Calculates the accuracy of the model's predictions.
   2. **Explanation**: This function compares the actual labels of the testing set with the model's predictions to determine how many were correct.

**Code 02 Functions :-**

1. **np.array():** Creates a NumPy array from a list of lists, allowing for efficient numerical operations.
2. **pd.DataFrame()**: Converts the NumPy array into a Pandas DataFrame for easier data manipulation and analysis.
3. **df.iloc[]:** Selects rows and columns by integer-location based indexing; used here to separate features and target variables.
4. **train\_test\_split():** Splits the dataset into training and testing sets based on a specified proportion, ensuring randomness for model evaluation.

**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)

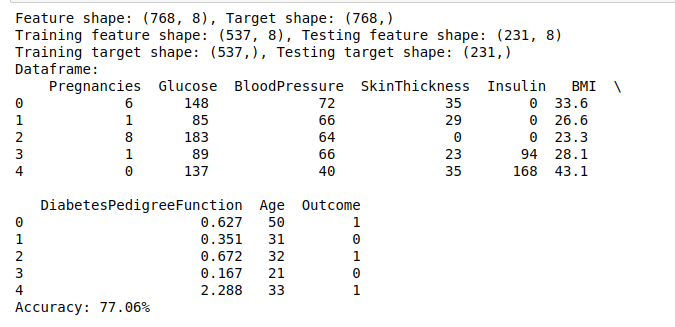
# Calculate and print accuracy

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)

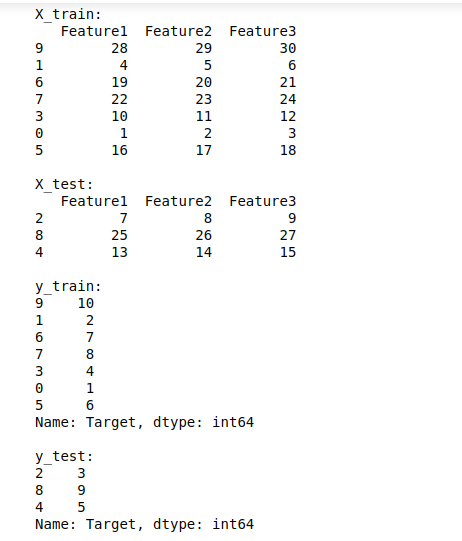
print("\ny\_train:")

print(y\_train)

print("\ny\_test:")

print(y\_test)

**Code 02 Output :-**



**Conclusion**

Both code segments illustrate fundamental steps in a machine learning workflow, including data loading, preprocessing, model training, and evaluation. The first code uses a real-world dataset for classification, while the second demonstrates data preparation using synthetic data.

**Experiment No: 04**

**Aim:**

**Objectives:**

**Theory:**

1. **Importing Libraries**
   1. **numpy**: For numerical operations and array handling.
   2. **matplotlib.pyplot**: For data visualization and plotting.
   3. **pandas:** For data manipulation and analysis.
   4. **train\_test\_split:** For splitting the dataset into training and testing sets.
   5. **LinearRegression**: The algorithm used for regression analysis.
2. **Importing the Dataset**
   1. **pd.read\_csv('Salary\_Dataset.csv')**: Loads the dataset from a CSV file into a Pandas DataFrame.
3. **Separating Features and Target Variable**
   1. **dataset.iloc[:, 0:1].values**: Selects the first column (Years of Experience) as features (X).
   2. **dataset.iloc[:, 1].values:** Selects the second column (Salary) as the target variable (y).
4. **Splitting the Dataset**
   1. train\_test\_split(X, y, test\_size=0.25, random\_state=0): Splits the dataset into training (75%) and testing (25%) sets, ensuring reproducibility.
5. **Fitting the Linear Regression Model**
   1. LinearRegression(): Creates an instance of the linear regression model.
   2. regressor.fit(X\_train, y\_train): Trains the model using the training data.
6. **Print Shapes for Debugging**
   1. **print("X\_train shape:", X\_train.shape)**: Displays the shape of the training feature set.
   2. **print("y\_train shape:", y\_train.shape)**: Displays the shape of the training target set.
   3. **print("X shape:", X.shape):** Displays the shape of the features.
   4. **print("y shape:", y.shape):** Displays the shape of the target variable.
7. **Visualizing the Training Set Results**
   1. **plt.scatter(X\_train, y\_train, color='red'):** Plots the training data points in red.
   2. **plt.plot(X\_train, regressor.predict(X\_train), color='blue'):** Plots the regression line using the training data predictions in blue.
   3. **plt.title(), plt.xlabel(), plt.ylabel()**: Set the title and axis labels for the plot.
   4. **plt.show():** Displays the plot.
8. **Visualizing the Test Set Results**
   1. **plt.scatter(X\_test, y\_test, color='red')**: Plots the test data points in red.
   2. **plt.plot(X\_train, regressor.predict(X\_train), color='blue')**: Plots the regression line using the training data predictions.
   3. **plt.title(), plt.xlabel(), plt.ylabel():** Set the title and axis labels for the plot.
   4. **plt.show()**: Displays the plot.
9. **Predicting the Test Set Results**
   1. **y\_pred = regressor.predict(X\_test):** Makes predictions on the test set using the trained model.
10. **Displaying Predictions**
    1. **for pred in y\_pred:**: Iterates through predicted salary values.
    2. **print("Predicted Salary:", pred):** Outputs each predicted salary to the console.

**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

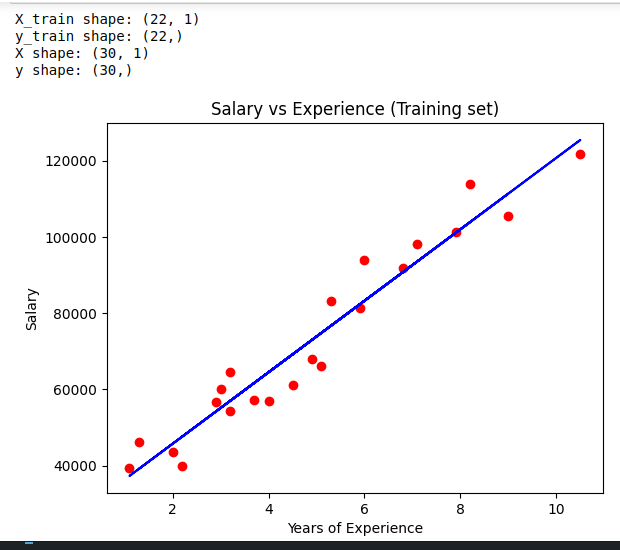
y\_pred = regressor.predict(X\_test)

# Displaying predictions

for pred in y\_pred:

print("Predicted Salary:", pred)

**Code Output :-**





**Experiment No: 05**

**Aim:**

**Objectives:**

**Theory:**

**Code 01 Theory**

1. **Library Imports:**
   1. Imports necessary libraries such as pandas, numpy, matplotlib.pyplot, and seaborn for data manipulation and visualization.
   2. Imports LinearRegression and metrics from sklearn for building and evaluating the regression model.
2. **Suppressing Warnings:**
   1. Uses warnings.filterwarnings("ignore") to suppress any warnings during execution, allowing for cleaner output.
3. **Loading the Dataset:**
   1. Loads the Student\_Performance.csv dataset into a DataFrame named df.
4. **Correlation and Description:**
   1. Computes and prints the correlation matrix using df.corr(numeric\_only=True) to analyze relationships between numeric variables.
   2. Prints summary statistics of the dataset using df.describe().
5. **Feature and Target Variable Selection:**
   1. Sets X to contain the features: Hours\_Studied and Previous\_Scores.
   2. Sets y to be the target variable: Performance\_Index.
6. **Data Visualization:**
   1. Creates boxplots for Hours\_Studied and Previous\_Scores to visualize their distributions.
   2. Uses sns.histplot to visualize the distribution of the Performance\_Index with a Kernel Density Estimate (KDE).
   3. Generates a pairplot to visualize relationships between features and the target variable.
7. Heatmap Visualization:
   1. Creates a heatmap to visualize the correlation matrix, with annotations and a color gradient.
8. **Train-Test Split:**
   1. Splits the dataset into training and testing sets (70%-30% split) using train\_test\_split().
   2. Model Fitting:
   3. Initializes and fits a multiple linear regression model using the training data.
9. **Model Coefficients:**
   1. Prints the intercept and coefficients of the regression model, helping understand the impact of each feature.
10. **Prediction:**
    1. Uses the fitted model to make predictions on the test set.
11. **Performance Evaluation:**
    1. Calculates and prints Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE) to evaluate model performance.
12. **Results Comparison:**
    1. Creates a DataFrame to compare actual vs. predicted values of the target variable and prints it.

**Code 02 Theory**

1. **Library Imports:**
   1. Similar to Code 01, imports essential libraries for data handling and visualization, along with regression modeling tools from sklearn.
2. **Loading the Dataset:**
   1. Loads the Student\_Performance.csv dataset into a DataFrame named df.
3. **Feature and Target Variable Preparation:**
   1. Defines X with features: Hours\_Studied and Previous\_Scores.
   2. Defines y as the target variable: Performance\_Index.
4. **Exploratory Data Analysis:**
   1. Prints the correlation matrix to explore relationships among numeric variables.
   2. Generates a pairplot to visualize the relationship between features and the target variable.
5. **Train-Test Split:**
   1. Splits the dataset into training and testing sets (70%-30% split) for model validation.
6. **Model Creation and Fitting:**
   1. Initializes a linear regression model and fits it using the training data.
7. **Model Coefficients:**
   1. Prints the intercept and coefficients, providing insights into the influence of each feature.
8. **Making Predictions:**
   1. Predicts target values for the test set using the fitted regression model.
9. **Model Evaluation:**
   1. Calculates Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R²) to assess the model's accuracy and performance.
10. **Results Visualization:**
    1. Creates a scatter plot to compare actual vs. predicted values of the target variable, including a 45-degree reference line to visualize prediction accuracy.

**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)

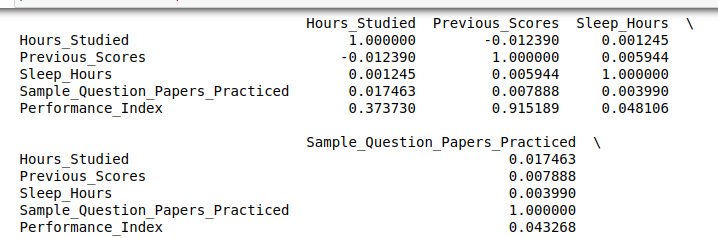
r2 = np.sqrt(mse) # Changed to use mse instead of mean\_squared\_error directly

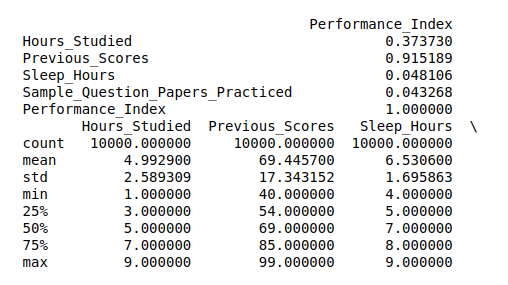
print('Mean Absolute Error:', mae)

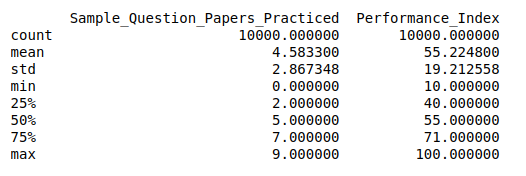
print('Mean Square Error:', mse)

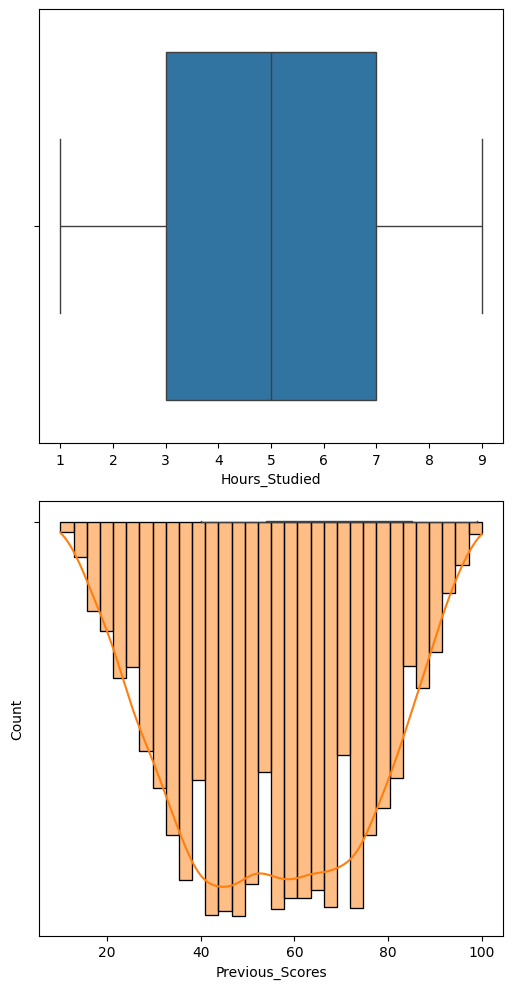
print('Root Mean Square Error:', r2)

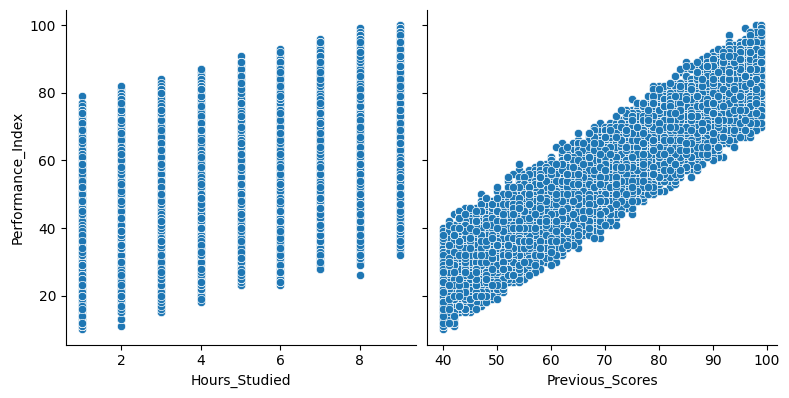
**Code 01 Output :-**

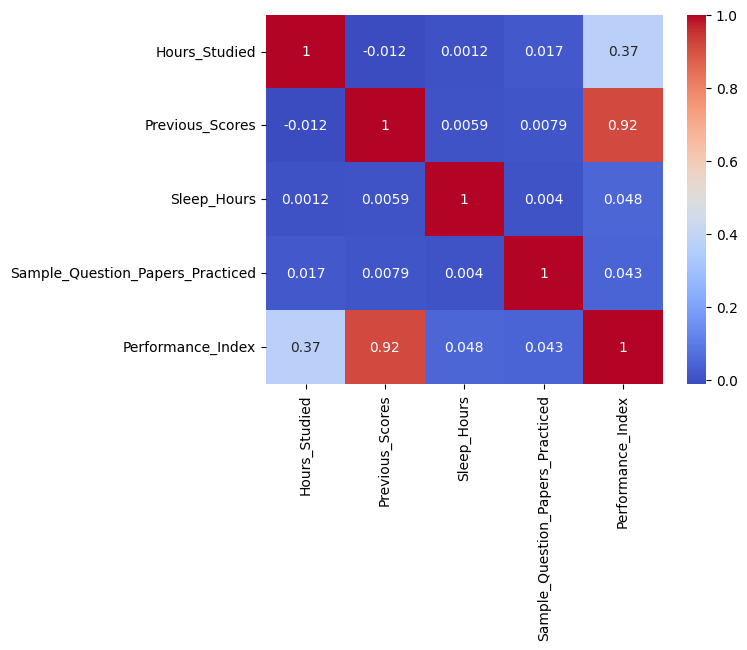


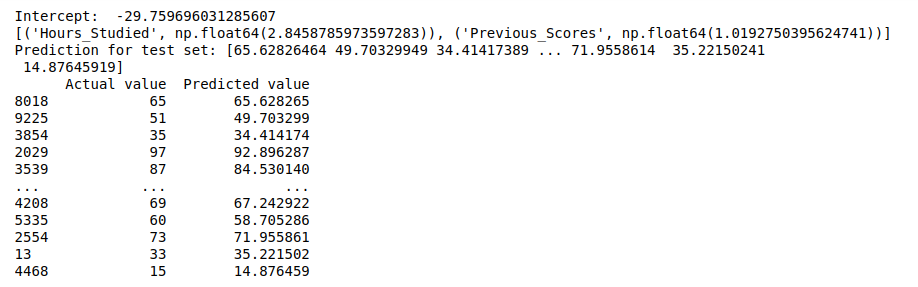


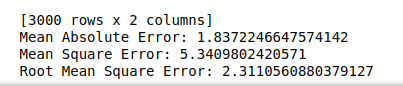








****

****

**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')

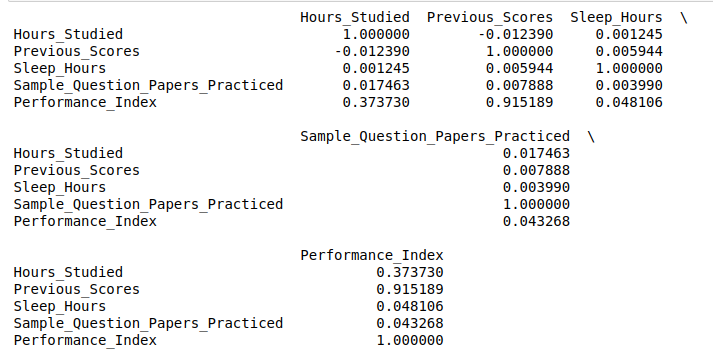
plt.ylabel('Predicted Performance Index')

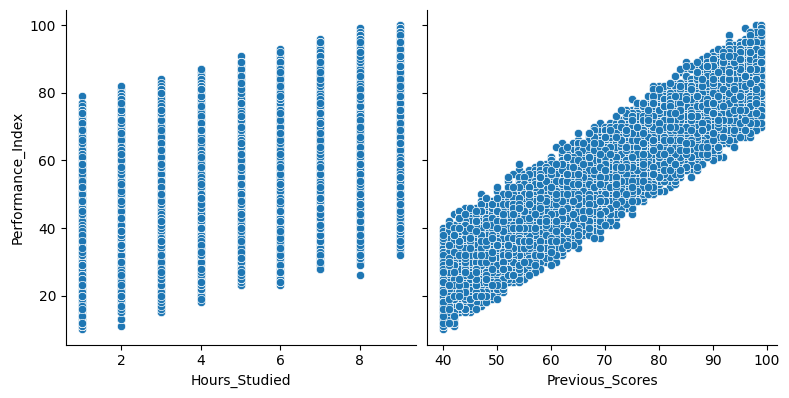
plt.title('Actual vs Predicted Performance Index')

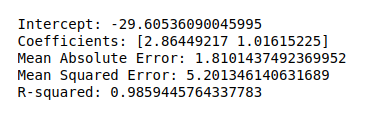
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--') # 45-degree line

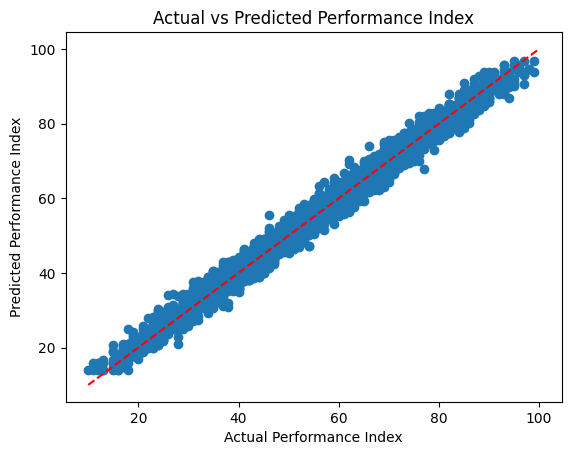
plt.show()

**Code 02 Output :-**









**Experiment No: 06**

**Aim:**

**Objectives:**

**Theory:**

**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)

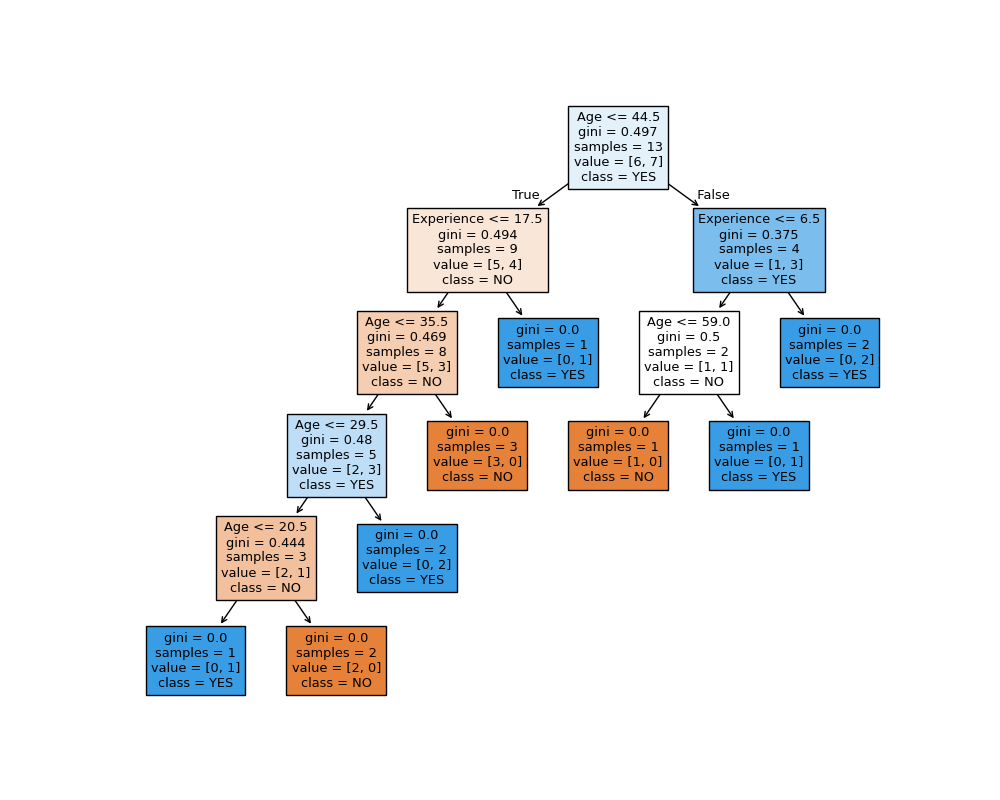
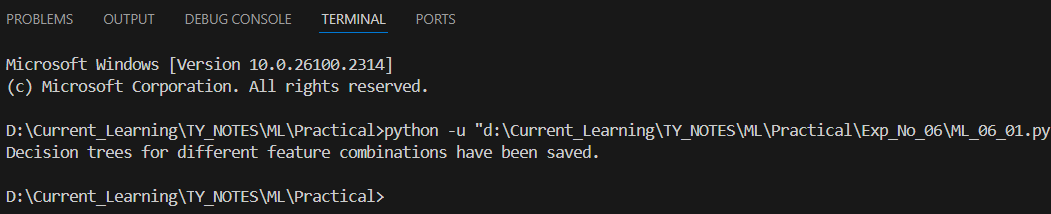
    # Save the plot with a unique name for each feature combination

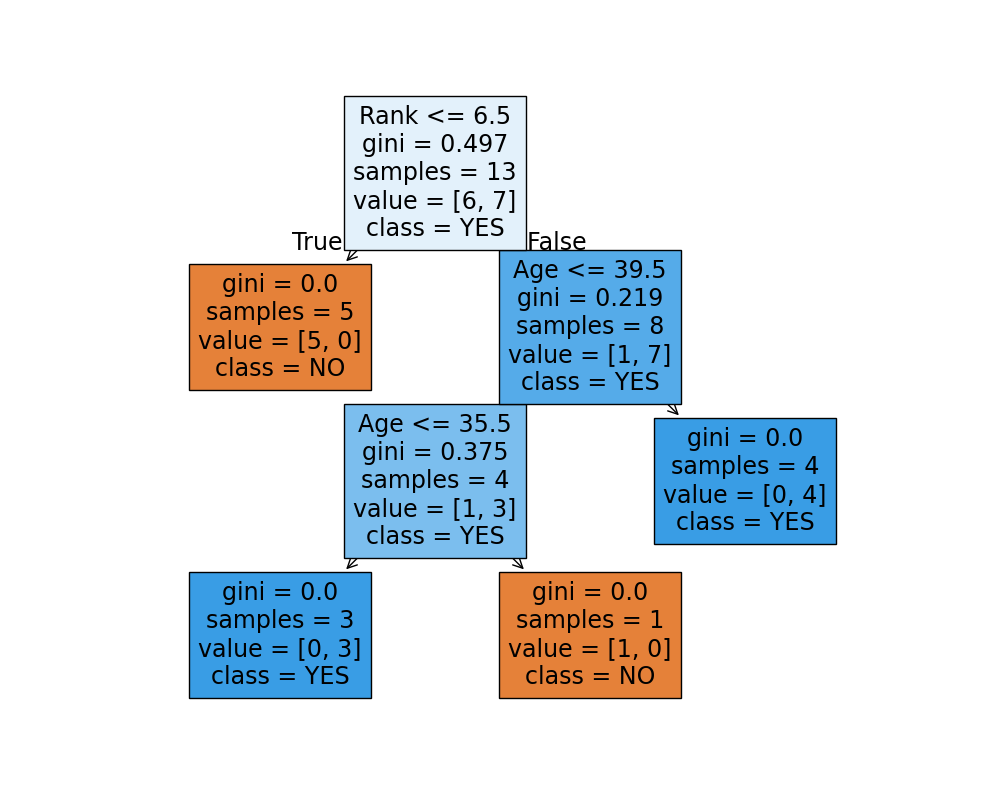
    plt.savefig(f"decision\_tree\_{i+1}.png")

    plt.close()

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

**Output:**

****

****

**Experiment No: 07**

**Aim:**

**Objectives:**

**Theory:**

**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['Sex'].fillna('unknown', inplace=True)

data['Embarked'].fillna('unknown', inplace=True)

# 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)