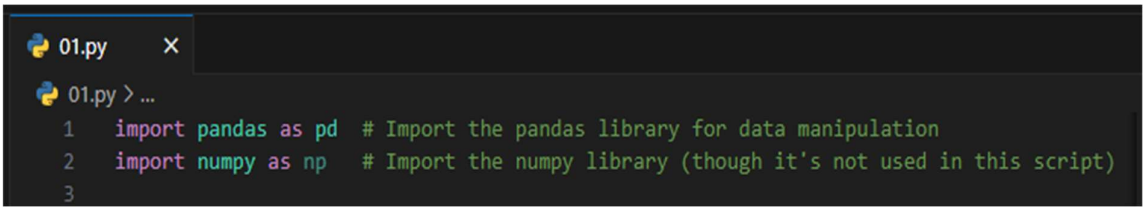
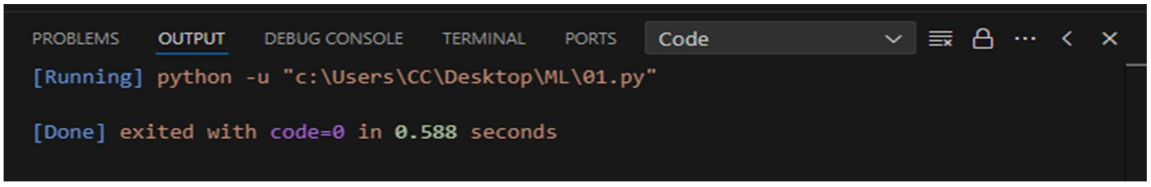

Experiment No: 01

1. Pandas Library:

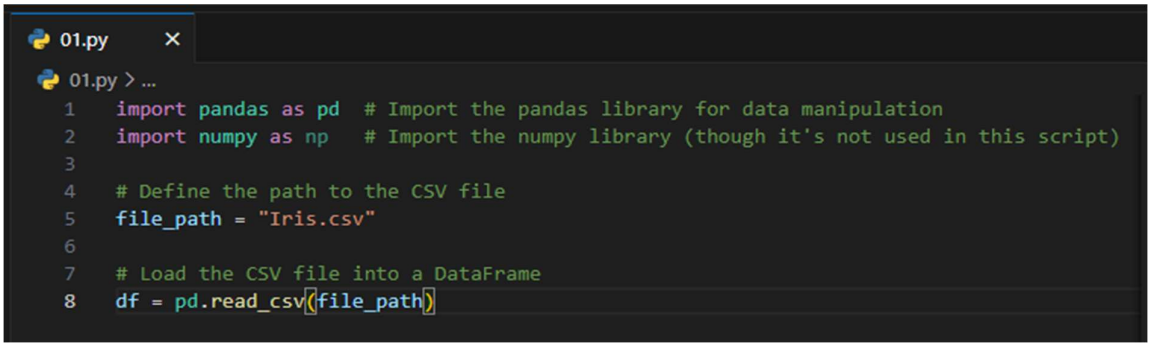
Code:

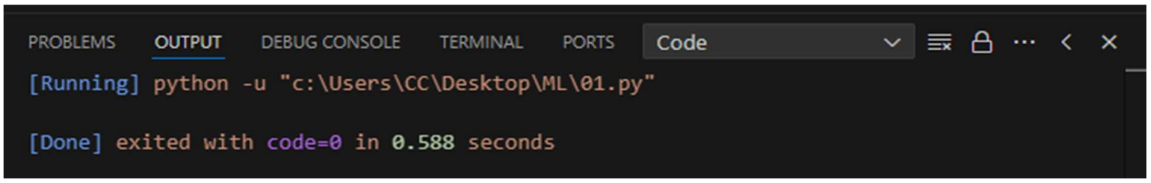
1. **Importing Libraries:**
   - import pandas as pd: We import Pandas as pd, which is a common alias used by the community for convenience.
   - import numpy as np: Numpy is imported as np, though it's not used in this script, it's often used alongside Pandas for numerical operations.





2. **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.





3. **Data Exploration:**

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

```python
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np   # Import the numpy library (though it's not used in this script)

# Define the path to the CSV file
file_path = "Iris.csv"

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame
print("First few rows:")
print(df.head())
```

```
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

[Done] exited with code=0 in 0.588 seconds

[Running] python -u "c:\Users\CC\Desktop\ML\01.py"
First few rows:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Class Label
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa

[Done] exited with code=0 in 0.366 seconds
```

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

```python
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np   # Import the numpy library (though it's not used in this script)

# Define the path to the CSV file
file_path = "Iris.csv"

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display the last few rows of the DataFrame
print("\nLast few rows:")
print(df.tail())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

Last few rows:
     Id  SepalLengthCm  ...  PetalWidthCm     Class Label
145  146            6.7  ...           2.3  Iris-virginica
146  147            6.3  ...           1.9  Iris-virginica
147  148            6.5  ...           2.0  Iris-virginica
148  149            6.2  ...           2.3  Iris-virginica
149  150            5.9  ...           1.8  Iris-virginica

[5 rows x 6 columns]

[Done] exited with code=0 in 0.589 seconds
```

4. **Data Structure and Information:**

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

```
01.py
01.py > ...
1   import pandas as pd  # Import the pandas library for data manipulation
2   import numpy as np   # Import the numpy library (though it's not used in this script)
3
4   # Define the path to the CSV file
5   file_path = "Iris.csv"
6
7   # Load the CSV file into a DataFrame
8   df = pd.read_csv(file_path)
9
10  # Display the shape of the DataFrame (number of rows and columns)
11  print("\nShape of the DataFrame:")
12  print(df.shape)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

Shape of the DataFrame:
(150, 6)

[Done] exited with code=0 in 0.359 seconds
```

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

```python
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np   # Import the numpy library (though it's not used in this script)

# Define the path to the CSV file
file_path = "Iris.csv"

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display basic information about the DataFrame, including column data types and non-null counts
print("\nInformation about the DataFrame:")
print(df.info())
```

```
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

Information about the DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Id              150 non-null    int64
 1   SepalLengthCm   150 non-null    float64
 2   SepalWidthCm    150 non-null    float64
 3   PetalLengthCm   150 non-null    float64
 4   PetalWidthCm    150 non-null    float64
 5   Class Label     150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None

[Done] exited with code=0 in 0.377 seconds
```

5. **Data Size and Missing Values:**
   - df.size returns the total number of elements in the DataFrame, which is the product of rows and columns

```python
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np   # Import the numpy library (though it's not used in this script)

# Define the path to the CSV file
file_path = "Iris.csv"

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display the total number of elements in the DataFrame (rows * columns)
print("\nTotal number of elements in the DataFrame:")
print(df.size)
```

```
PROBLEMS  OUTPUT  ...                                   Code
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

Total number of elements in the DataFrame:
900

[Done] exited with code=0 in 0.361 seconds
```

- 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.

```
01.py
01.py > ...
1   import pandas as pd  # Import the pandas library for data manipulation
2   import numpy as np   # Import the numpy library (though it's not used in this script)
3
4   # Define the path to the CSV file
5   file_path = "Iris.csv"
6
7   # Load the CSV file into a DataFrame
8   df = pd.read_csv(file_path)
9
10  # Display information about missing values in the DataFrame (this method needs parentheses)
11  print("\nMissing values in the DataFrame:")
12  print(df.isna())
13  print(df.isna().sum())
```

```
PROBLEMS  OUTPUT  ...                                   Code
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

Missing values in the DataFrame:
        Id  SepalLengthCm  ...  PetalWidthCm  Class Label
0    False          False  ...         False        False
1    False          False  ...         False        False
2    False          False  ...         False        False
3    False          False  ...         False        False
4    False          False  ...         False        False
..     ...            ...  ...           ...          ...
145  False          False  ...         False        False
146  False          False  ...         False        False
147  False          False  ...         False        False
148  False          False  ...         False        False
149  False          False  ...         False        False

[150 rows x 6 columns]
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Class Label      0
dtype: int64

[Done] exited with code=0 in 0.371 seconds
```

6. **Descriptive Statistics:**

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

```python
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np   # Import the numpy library (though it's not used in this script)

# Define the path to the CSV file
file_path = "Iris.csv"

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display descriptive statistics of the DataFrame, such as mean, standard deviation, and quartiles
print("\nDescriptive statistics of the DataFrame:")
print(df.describe())
```

```
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

Descriptive statistics of the DataFrame:
                Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   150.000000     150.000000    150.000000     150.000000    150.000000
mean     75.500000       5.843333      3.054000       3.758667      1.198667
std      43.445368       0.828066      0.433594       1.764420      0.763161
min       1.000000       4.300000      2.000000       1.000000      0.100000
25%      38.250000       5.100000      2.800000       1.600000      0.300000
50%      75.500000       5.800000      3.000000       4.350000      1.300000
75%     112.750000       6.400000      3.300000       5.100000      1.800000
max     150.000000       7.900000      4.400000       6.900000      2.500000

[Done] exited with code=0 in 0.468 seconds
```

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

```python
import pandas as pd  # Import the pandas library for data manipulation
import numpy as np   # Import the numpy library (though it's not used in this script)

# Define the path to the CSV file
file_path = "Iris.csv"

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

# Display the number of unique values for each column in the DataFrame
print("\nNumber of unique values per column:")
print(df.nunique())
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01.py"

Number of unique values per column:
Id               150
SepalLengthCm     35
SepalWidthCm      23
PetalLengthCm     43
PetalWidthCm      22
Class Label        3
dtype: int64

[Done] exited with code=0 in 0.355 seconds
```

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.

01_02.py

```python
import numpy as np

# Create a NumPy array
arr = np.array([1, 2, 3, 4, 5])

# Find and print the minimum value in the array
print("Minimum value in the array:", np.min(arr))

# Find and print the maximum value in the array
print("Maximum value in the array:", np.max(arr))

```

```
PROBLEMS  OUTPUT  ...                                  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Minimum value in the array: 1
Maximum value in the array: 5

[Done] exited with code=0 in 0.129 seconds
```

4. Computing Mean

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

```
01_02.py
01_02.py > ...
1  import numpy as np
2
3  # Create a NumPy array
4  arr = np.array([1, 2, 3, 4, 5])
5
6  # Compute and print the mean (average) value of the array
7  print("Mean (average) value of the array:", np.mean(arr))
```

```
PROBLEMS  OUTPUT  ...                                  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Mean (average) value of the array: 3.0

[Done] exited with code=0 in 0.17 seconds
```

5. Computing Standard Deviation

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

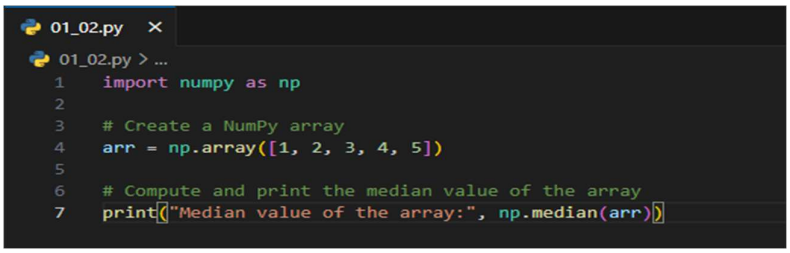
```
01_02.py
01_02.py > ...
1  import numpy as np
2
3  # Create a NumPy array
4  arr = np.array([1, 2, 3, 4, 5])
5
6  # Compute and print the standard deviation of the array
7  print("Standard deviation of the array:", np.std(arr))
```

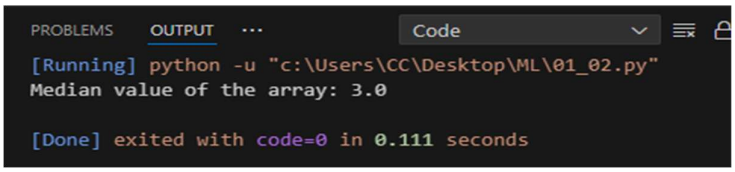
```
PROBLEMS  OUTPUT  ...                                  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Standard deviation of the array: 1.4142135623730951

[Done] exited with code=0 in 0.13 seconds
```

---

6. Computing Median

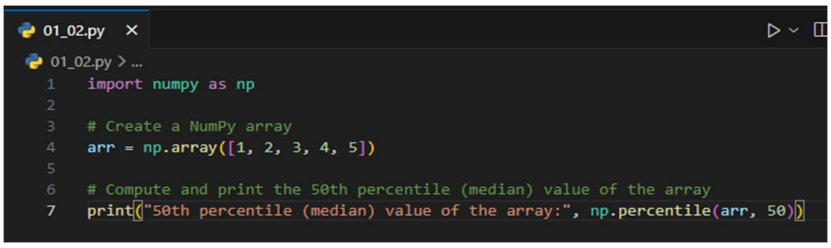
- np.median(): Finds the median value, which is the middle value when the array is sorted.





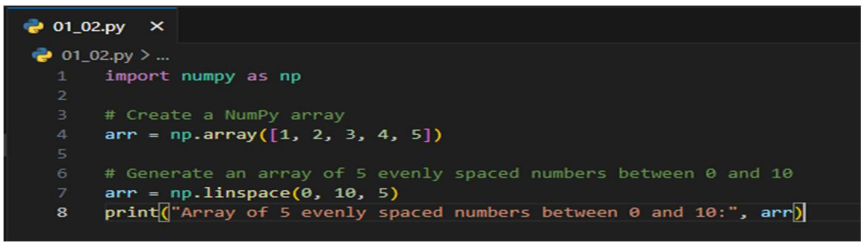
7. Computing Percentile

- 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.



```
PROBLEMS  OUTPUT  ...                                  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Array of 5 evenly spaced numbers between 0 and 10: [ 0.   2.5  5.
7.5 10. ]

[Done] exited with code=0 in 0.137 seconds
```

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.

```
01_02.py
01_02.py > ...
1   import numpy as np
2
3   # Create a 2D NumPy array
4   arr = np.array([[1, 2], [3, 4]])
5
6   # Print the shape of the array (number of rows and columns)
7   print("Shape of the 2D array:", arr.shape)
8
```

```
PROBLEMS  OUTPUT  ...                                  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Shape of the 2D array: (2, 2)

[Done] exited with code=0 in 0.113 seconds
```

10. Reshaping an Array

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

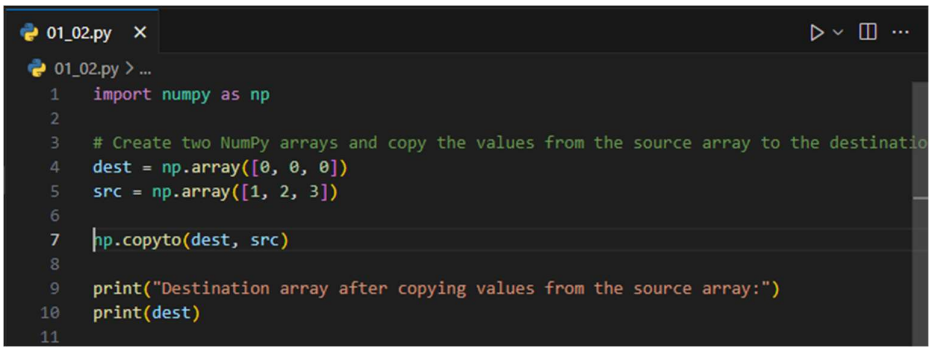
```
01_02.py
01_02.py > ...
1   import numpy as np
2
3   # Create a 1D NumPy array and reshape it into a 2D array with 2 rows and 3 columns
4   arr = np.array([1, 2, 3, 4, 5, 6])
5   reshaped_arr = arr.reshape((2, 3))
6   print("Reshaped 2D array with 2 rows and 3 columns:")
7   print(reshaped_arr)
8
```

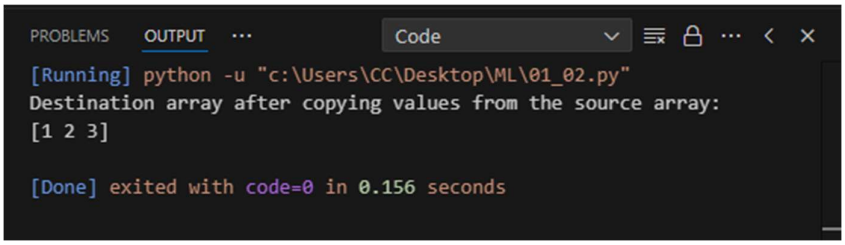
```
PROBLEMS  OUTPUT  ...                                  Code
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Reshaped 2D array with 2 rows and 3 columns:
[[1 2 3]
 [4 5 6]]

[Done] exited with code=0 in 0.129 seconds
```

---

11. Copying Values Between Arrays

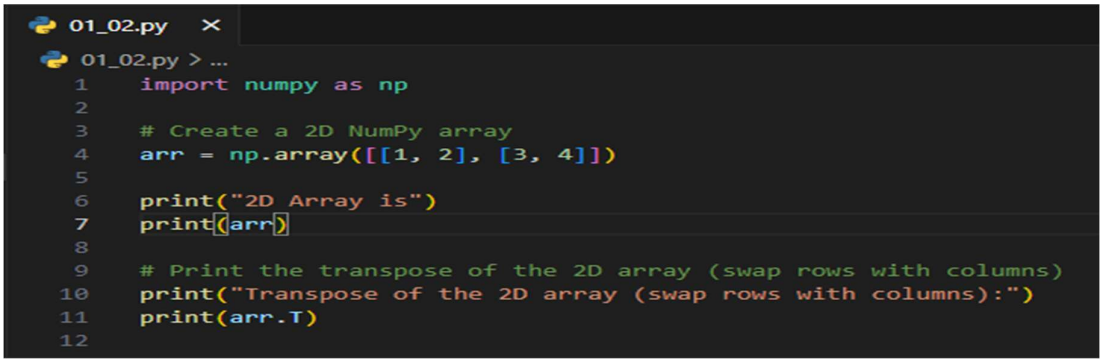
- np.copyto(): Copies values from one array (source) to another (destination), modifying the destination array in place.

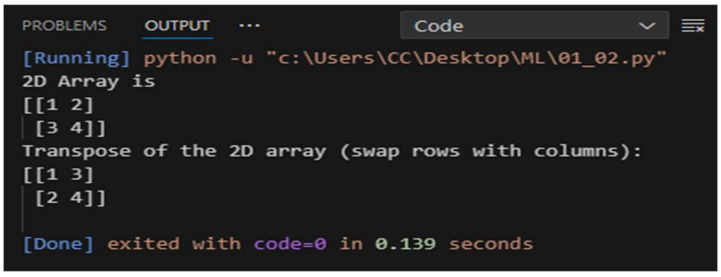




12. Transposing an Array

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





---

13. Stacking Arrays

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

```python
import numpy as np

# Create two NumPy arrays and stack them along a new axis (row-wise)
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])

stacked = np.stack((arr1, arr2), axis=0)

print("Arrays stacked along a new axis (row-wise):")
print(stacked)
```

```
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Arrays stacked along a new axis (row-wise):
[[1 2]
 [3 4]]

[Done] exited with code=0 in 0.149 seconds
```

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.

```python
import numpy as np

# Create two NumPy arrays and stack them along a new axis (row-wise)
arr1 = np.array([1, 2])
arr2 = np.array([3, 4])

# Vertically stack the two arrays (row-wise)
vstacked = np.vstack((arr1, arr2))
print("Arrays stacked vertically (row-wise):")
print(vstacked)

# Horizontally stack the two arrays (column-wise)
hstacked = np.hstack((arr1, arr2))
print("Arrays stacked horizontally (column-wise):")
print(hstacked)
```

```
[Running] python -u "c:\Users\CC\Desktop\ML\01_02.py"
Arrays stacked vertically (row-wise):
[[1 2]
 [3 4]]
Arrays stacked horizontally (column-wise):
[1 2 3 4]

[Done] exited with code=0 in 0.121 seconds
```

Matplotlib Library

1. Creating Graph

1. Creating a Scatter Plot
    a. .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
    a. .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
    a. .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

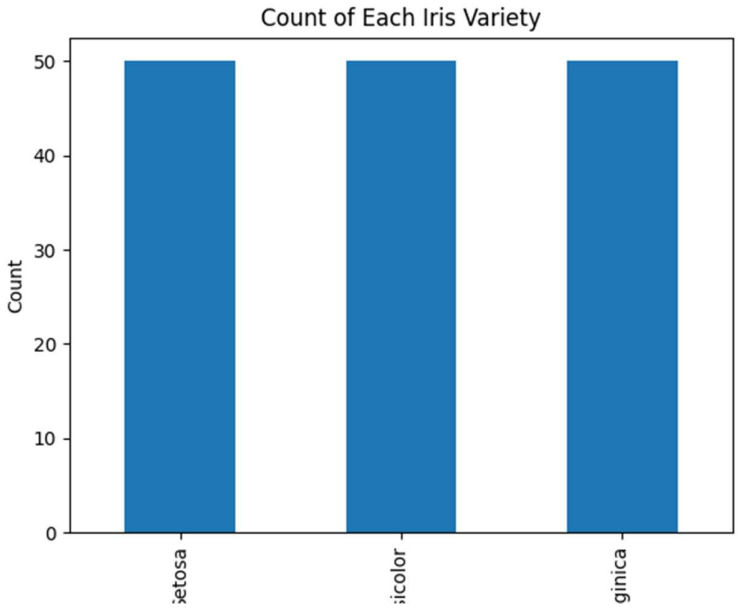
- plt.show(): Renders and displays the current figure or plot. This function is essential for visualizing the created plot in a standalone window or inline, depending on the environment.

Code: -

1. **Bar Chart**

```python
#Bar Chart


# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('iris.csv')

# Creating a bar chart for the count of each variety
df['variety'].value_counts().plot(kind='bar')

# Adding labels and title
plt.xlabel('Variety')
plt.ylabel('Count')
plt.title('Count of Each Iris Variety')

# Displaying the plot
plt.show()

```

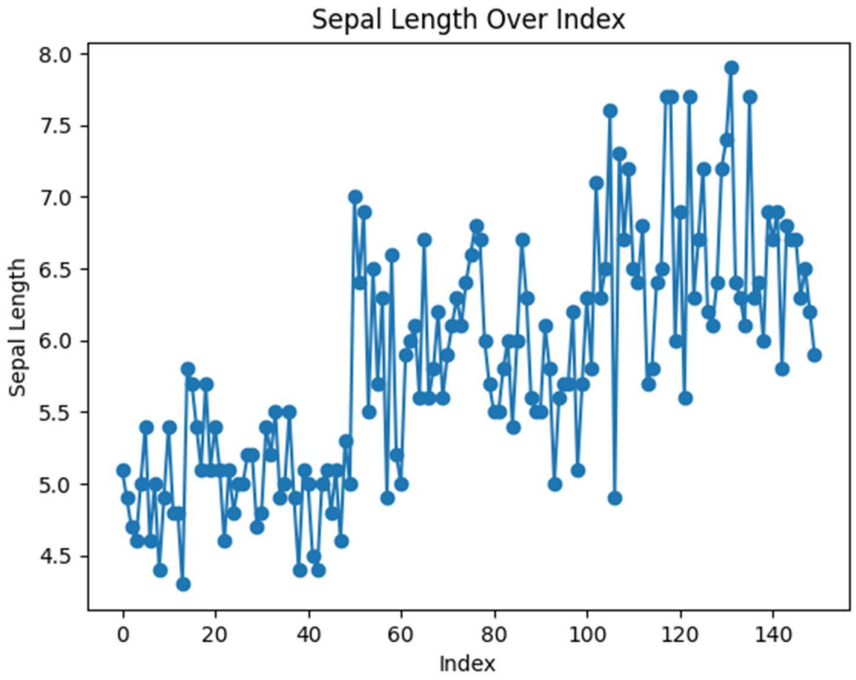
Output: -



2. Line Chart

```python
#Line Chart


# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('iris.csv')

# Create a line chart for sepal length
df['sepal.length'].plot(kind='line', marker='o')

# Add labels and title
plt.xlabel('Index')
plt.ylabel('Sepal Length')
plt.title('Sepal Length Over Index')

# Display the plot
plt.show()

```

Output: -



---

3. Scatter Plot

```python
#Scatter Plot

# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('iris.csv')

# Create a scatter plot for sepal length vs. sepal width
df.plot(kind='scatter', x='sepal.length', y='sepal.width')

# Add labels and title
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Sepal Length vs Sepal Width')

# Display the plot
plt.show()
```

Output:-