



SIX WEEKS SUMMER TRAINING REPORT

on

DATA ANALYSIS AND VISUALIZATION USING PYTHON

Submitted by

SONPAL

Registration No. 12217639

Programme Name: Data Science

Under the Guidance of

**Center for Professional Enhancement
Lovely Professional University, Phagwara**

**School of Computer Application
Lovely Professional University, Phagwara
(June-July 2023)**

DECLARATION

I Sonpal a student of Master of Computer Application at Lovely Professional University, hereby solemnly declare that I have successfully completed a rigorous six-week summer training program at the Center for Professional Enhancement, Lovely Professional University. The training took place from 9th June 2019 to 17th July 2019, and throughout this period, I had the privilege to work under the expert guidance of Ms. Pooja Rana, whose insights and mentoring greatly contributed to my learning experience.

During these intensive six weeks of training, I wholeheartedly devoted myself to the pursuit of knowledge and skill enhancement. My dedication and enthusiasm were reflected in my commitment to the tasks, projects, and exercises presented during the training program. This period of training not only enriched my understanding but also enabled me to gain practical insights into the field of Data Analysis and Visualization using Python.

The culmination of my efforts over the course of this training program has resulted in the achievement of comprehensive learning outcomes that align perfectly with the prescribed requirements for the degree in Data Analysis and Visualization using Python, as established by Lovely Professional University, Phagwara.

I acknowledge that this training period has been an invaluable opportunity for hands-on learning, exposure to real-world applications, and the development of key analytical and programming skills. I am grateful to Lovely Professional University for providing me with this platform to enhance my knowledge and competencies.

I affirm that the work presented in this report is my original contribution and reflects my understanding of the concepts and techniques gained during the training program. Any external sources, references, or collaborative efforts have been duly acknowledged in accordance with academic integrity standards.

In witness thereof, I set my hand to this declaration.

Date: 10 Aug 2023

Place: Phagwara

SONPAL

Reg.NO. -12217639

ACKNOWLEDGEMENT

I am filled with a profound sense of gratitude as I acknowledge the invaluable contributions of a multitude of well-wishers, each playing a unique and significant role in the culmination of my successful Summer Training journey.

The path to the successful completion of any technological endeavor is illuminated by the support and efforts of numerous individuals. In preparing this report, I have been fortunate to receive assistance from various quarters, and it is with heartfelt appreciation that I take this opportunity to express my sincere gratitude to those who have extended their support.

First and foremost, I extend my deepest gratitude to our esteemed Training Mentor, Ms. Pooja Rana. I am genuinely thankful for her unwavering support, insightful guidance, and patient mentoring throughout the entire training period. Her expertise and willingness to share her knowledge played a pivotal role in shaping my understanding and navigating through the challenges that arose during the training. Without her consistent direction and profound guidance, the culmination of this endeavor would not have been as successful. Her supervisory role and continuous feedback have been instrumental in refining the outcomes of this training.

Moreover, I am indebted to the faculty members and staff of Lovely Professional University who provided the conducive learning environment and resources that enabled me to embark on this journey of knowledge acquisition and skill development.

Lastly, but certainly not least, I am thankful to my family and friends for their unwavering encouragement and patience during the training period. Their belief in my capabilities and constant motivation were essential factors in sustaining my dedication and commitment.

The collective efforts of all these individuals have played a pivotal role in my growth and achievement during this training program. I am profoundly grateful for their contributions, and I am humbled by their unwavering support.

With heartfelt thanks,

SONPAL

Reg. No. -12217639

CENTRE FOR PROFESSIONAL ENHANCEMENT

[Under the Aegis of Lovely Professional University, Jalandhar-Delhi G.T. Road, Phagwara (Punjab)]

Certificate No. 284340

Certificate of Merit

This is to certify that Mr./Ms. Sonpal S/O,D/O,W/O Hariom
student of School of Computer Application Registration No. 12217639
pursuing MCA participated in skill development
course named Data Analysis and Visualization using Python
organized by Centre for Professional Enhancement Lovely Professional University
w.e.f 09-06-2023 to 17-07-2023 and obtained A Grade.

Date of Issue : 11-08-2023
Place of Issue: Phagwara (India)


Prepared by
(Administrative Officer-Records)


Organizing Secretary


Head of School

Table of Contents

Week 1: Basics of Python Programming

1. Installation of Anaconda
2. Variables and Keywords
3. Operators
4. Conditional Statements
5. Looping Statements
6. Inbuilt and User-Defined Functions

Week 2: Basic Data Structures in Python

1. Lists
2. Strings
3. Tuples
4. Dictionaries
5. Sets

Week 3: Classes and Objects

1. Creating Classes and Objects
2. Accessing Attributes
3. Class Inheritance
4. Data Hiding, Overriding, and Overloading
5. File Handling
6. Exception Handling

Week 4: Numpy and Pandas

1. Numpy Arrays
 - Built-in Methods
 - Array Attributes and Methods
 - Array Indexing and Selection
 - Broadcasting
 - Fancy Indexing
 - Statistical Operations on Arrays
2. Introduction to Pandas
 - Series and DataFrames
 - Indexing of DataFrames
 - DataFrame Operations (count, unique, sum, describe, info, statistical operations)
 - Reading and Writing CSV Files Using DataFrames
 - Data Analysis Using DataFrame Operations

Week 5: Matplotlib and Seaborn

1. Introduction to Matplotlib
 - Plotting Graphs and Subplots
2. Data Visualization with Matplotlib
 - Bar Charts, Pie Charts, Histograms, and Scatterplots
3. Introduction to Seaborn
 - Plotting Using Seaborn
 - Distplot and Pairplot
 - Barplot, Boxplot, Countplot, and Whiskerplot Using CSV Files
 - Correlation Matrix, Heatmap, Pairgrids

Week 6: Project Submission Based on Concepts Learned in Python

Week 1: Basics of Python Programming

1. Installation of Anaconda

Anaconda is a powerful platform that simplifies the process of setting up a Python environment for data analysis, scientific computing, and machine learning. It includes the Python programming language, essential libraries, and tools in a convenient package. Follow these steps to install Anaconda:

Download Anaconda:

Visit the Anaconda website (<https://www.anaconda.com/products/distribution>) and download the appropriate version of Anaconda (Anaconda or Miniconda) for your operating system (Windows, macOS, or Linux).

Run Installer:

Once the download is complete, run the installer executable. Follow the on-screen instructions. You can choose to install Anaconda for the current user or all users on the system. You may also choose the installation path.

Agree to Terms:

Read and agree to the license terms and conditions.

Choose Installation Type:

During the installation, you might be prompted to choose whether to add Anaconda to your system's PATH environment variable. Adding it to the PATH makes it easier to use Anaconda from the command line.

Install:

Proceed with the installation. It may take a few minutes to complete.

Installation Complete:

Once the installation is finished, you should see a confirmation message. Anaconda Navigator, a graphical interface for managing environments and packages, will also be installed.

Test Installation:

Open a terminal (or Anaconda Prompt on Windows) and enter the command `conda --version` to check if Anaconda was installed correctly. You can also try `python --version` to verify the Python version.

2. Variable and Keywords

Variables and keywords are two of the most important concepts in Python programming. Variables are used to store data. They are named memory locations that can hold values. To create a variable, you assign a value to it using the equals (=) sign. For example: `name = "Bard" age = 20`

The variable `name` now stores the value "Bard" and the variable `age` stores the value 20.

Keywords are reserved words that have special meaning in Python. They cannot be used as variable names or as function names. Some examples of keywords are `if`, `else`, `while`, `for`, `def`, and `class`.

The following is a list of all the keywords in Python:

`and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `raise`, `return`, `try`, `while`, `with`, `yield`

It is important to remember that keywords cannot be used as variable names. If you try to do this, you will get an error.

3. Operators

Arithmetic operators are used to perform mathematical operations on numbers. Some examples of arithmetic operators are:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Exponentiation (**)
- Floor division (//)

Assignment operators are used to assign values to variables. Some examples of assignment operators are:

- Simple assignment (=)
- Add assignment (+=)
- Subtract assignment (-=)
- Multiply assignment (*=)
- Divide assignment (/=)
- Modulus assignment (%)=
- Exponentiation assignment (**=)
- Floor division assignment (//=)

Comparison operators are used to compare two values. The result of a comparison operator is either True or False. Some examples of comparison operators are:

- Equal to (==)
- Not equal to (!=)
- Greater than (>)
- Greater than or equal to (>=)
- Less than (<)
- Less than or equal to (<=)

Logical operators are used to combine logical expressions. The logical operators are and, or, and not.

- and returns True if both expressions are True.
- or returns True if either expression is True.
- not returns the opposite of the expression.

Identity operators are used to compare the identity of two objects. The identity operators are is and is not.

- is returns True if the two objects are the same object.
- is not returns True if the two objects are not the same object.

Membership operators are used to check if a value is a member of a sequence. The membership operators are in and not in.

- in returns True if the value is a member of the sequence.
- not in return True if the value is not a member of the sequence.

Bitwise operators are used to performing bitwise operations on integers. Bitwise operations are performed on the bits of the integers.

4. Conditional Statements

Conditional statements are used to control the flow of execution of a program. They allow you to execute different code depending on the value of a condition.

The most common conditional statement in Python is the if statement. The if statement has the following syntax:

```
if condition:
    # Do something
```

The condition is a Boolean expression that evaluates to either True or False. If the condition is True, the code inside the if statement is executed. If the condition is False, the code inside the if statement is skipped.

You can also use the elif and else statements to add more conditions to your code. The elif statement is used to check for an alternative condition. The else statement is used to execute code if none of the conditions are met.

```
if condition1:
    # Do something
elif condition2:
    # Do something else
else:
    # Do something else
```

5. Looping Statements

Looping statements are used to execute a block of code repeatedly. The most common looping statement in Python is the for loop. The for loop has the following syntax:

```
for variable in sequence:
    # Do something
```

The sequence can be a list, a string, or a range. The variable is a variable that will take on each value in the sequence. The code inside the for loop is executed for each value in the sequence.

You can also use the while loop to loop over a block of code. The while loop has the following syntax:

```
while condition:
    # Do something
```

The condition is a Boolean expression that evaluates to either True or False. If the condition is True, the code inside the while loop is executed. If the condition is False, the loop terminates.

6. Inbuilt and User-Defined Functions

User-defined functions are functions that you create yourself. They allow you to reuse code and make your programs more modular.

To define a function, you use the def keyword. The def keyword has the following syntax:

```
def function_name(parameters):
    # Do something
```

The function_name is the name of the function. The parameters are the arguments that the function takes. The code inside the function_name() block is the body of the function.

To call a function, you use the function name followed by the arguments. For example:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```


Week 2: Basic Data Structures in Python

1. Lists

Lists are one of the most versatile data structures in Python. They are used to store a collection of data items. The data items in a list can be of any type, including numbers, strings, and other lists. Lists are created using square brackets []. The items in a list are separated by commas.

For example:

```
list1 = [1, 2, 3, 4, 5]
list2 = ["a", "b", "c", "d", "e"]
list3 = [1, "a", True, 3.14]
```

Lists can be indexed using numbers. The first item in a list has index 0, the second item has index 1, and so on. For example:

```
print(list1[0]) # prints 1
print(list2[2]) # prints c
print(list3[3]) # prints 3.14
```

Lists can be modified. You can add items to a list, remove items from a list, and change the order of the items in a list. For example:

```
list1.append(6) # adds 6 to the end of list 1
list2.remove("b") # removes "b" from list 2
list3.sort() # sorts list3 in ascending order
```

2. Strings

Strings are another important data structure in Python. They are used to store a sequence of characters. Strings are created using single quotes ' or double quotes ".

For example:

```
string1 = "This is a string."
string2 = 'Python is a programming language.'
```

Strings can be indexed using numbers. The first character in a string has index 0, the second character has index 1, and so on. For example:

```
print(string1[0]) # prints T
print(string2[6]) # prints o
```

Strings can be manipulated. You can change the case of the characters in a string, remove characters from a string, and split a string into a list of substrings.

For example:

```
string1 = string1.upper() # changes the case of string1 to uppercase
string2 = string2.replace("Python", "Java") # replaces "Python" with "Java" in string2
list1 = string3.split(" ") # splits string3 into a list of substrings
```

3. Tuples

Tuples are similar to lists, but they are immutable. This means that the items in a tuple cannot be changed once the tuple is created. Tuples are created using parentheses (). The items in a tuple are separated by commas. For example:

```
tuple1 = (1, 2, 3, 4, 5)
tuple2 = ("a", "b", "c", "d", "e")
tuple3 = (1, "a", True, 3.14)
```

4. Dictionaries

Dictionaries are a data structure that stores data in key-value pairs. The keys are unique, and the values can be of any type. Dictionaries are created using curly braces {}. The keys and values are separated by a colon :. For example:

```
dict1 = {"name": "Bard", "age": 20}
```

```
dict2 = {"fruits": ["apple", "banana", "orange"], "colors": ["red", "blue", "green"]}
```

You can access the value associated with a key using the square brackets [] notation.

For example:

```
print(dict1["name"]) # prints Bard
print(dict2["fruits"][0]) # prints apple
```

5. Sets

Sets are a data structure that stores unique elements. Sets are created using curly braces {}. The elements in a set are separated by commas. For example:

```
set1 = {1, 2, 3, 4, 5}
set2 = {"a", "b", "c", "d", "e"}
set3 = {1, "a", True, 3.14}
```

You can check if an element is in a set using the in keyword. For example:

```
print(1 in set1) # True
print("b" in set2) # True
print(3.14 in set3) # False
```

Week 3: Classes and Objects

1. Creating Classes and Objects

Classes and objects are the two fundamental concepts of object-oriented programming (OOP). A class is a blueprint for creating objects. It defines the attributes and methods of the objects.

An object is an instance of a class. It has the attributes and methods defined by the class. To create a class, you use the class keyword. The class keyword has the following syntax:

```
class Class_name:
    # Attributes and methods
```

The Class_name is the name of the class. The Attributes and methods are the attributes and methods of the class. To create an object, you use the object() function. The object() function has the following syntax:

```
object_name = Class_name()
```

The object_name is the name of the object. The Class_name is the name of the class.

2. Accessing Attributes

The attributes of an object can be accessed using the dot notation. For example:

```
class Person:
    name = "Bard"
    age = 20

person = Person()

print(person.name) # prints Bard
print(person.age) # prints 20
```

3. Class Inheritance

Inheritance is a way of creating new classes from existing classes. The new class inherits the attributes and methods of the existing class.

To inherit from a class, you use the extends keyword. The extends keyword has the following syntax:

```
class ChildClass(ParentClass):  
    # Attributes and methods
```

The ChildClass is the name of the new class. The ParentClass is the name of the existing class.

4. Data Hiding, Overriding, and Overloading

Data hiding is a way of preventing the direct access of attributes and methods from outside the class. This can be done by declaring the attributes and methods as private.

To declare an attribute or method as private, you use the `_` prefix. For example:

```
class Person:  
    _name = "Bard"  
    _age = 20  
  
    def getName(self):  
        return self._name  
  
    def getAge(self):  
  
        return self._age
```

The `_name` and `_age` attributes are declared as private. This means that they cannot be accessed directly from outside the class.

Overriding is a way of redefining a method in a subclass. This can be done by using the same method name in the subclass.

Overloading is a way of having multiple methods with the same name in a class. This can be done by using different parameters in the methods.

5. File Handling

File handling is the process of reading and writing files in Python. To read a file, you use the `open()` function. The `open()` function has the following syntax:

```
file = open("filename", "mode")
```

The filename is the name of the file. The mode is the mode in which the file is opened. The modes are "r" for reading, "w" for writing, and "a" for appending.

To write to a file, you use the `write()` method. The `write()` method has the following syntax:

```
file.write("This is a line of text.")
```

The file is the file object. The "This is a line of text." is the text that is written to the file.

To close a file, you use the `close()` method. The `close()` method has the following syntax:

```
file.close()
```

6. Exception Handling

Exception handling is the process of handling errors that occur during the execution of a program. To handle an exception, you use the try and except statements. The try and except statements have the following syntax:

try:

Code that might cause an exception

except Exception as e:

Code to handle the exception

The try block is the code that might cause an exception. The except block is the code that handles the exception. The Exception is the type of exception that is handled.

Week 4: Numpy and Pandas

1. Numpy Arrays

Numpy is a powerful library in Python for numerical computations. Numpy arrays are a fundamental data structure in Numpy, providing efficient storage and operations on arrays of homogeneous data. Let's delve into various aspects of Numpy arrays:

Built-in Methods: Numpy offers methods to create arrays quickly, such as `numpy.array()`, `numpy.zeros()`, `numpy.ones()`, and `numpy.arange()`.

Array Attributes and Methods: Arrays have attributes like `shape`, `dtype`, and `size`. Numpy also provides methods like `reshape()`, `flatten()`, and `transpose()` for array manipulation.

Array Indexing and Selection: You can access elements and subsets of arrays using indexing and slicing operations.

Broadcasting: Numpy allows operations on arrays of different shapes through broadcasting.

Fancy Indexing: You can use arrays of indices to access or modify specific elements of an array.

Statistical Operations on Arrays: Numpy provides functions to perform statistical calculations like `mean`, `median`, `standard deviation`, and more.

2. Introduction to Pandas

Pandas is a powerful library for data manipulation and analysis. It introduces two primary data structures: Series and DataFrames.

Series and DataFrames: A Series is a one-dimensional labeled array, similar to a column in a spreadsheet. A DataFrame is a two-dimensional table of data with rows and columns.

Indexing of DataFrames: Pandas provides flexible ways to index and select data from DataFrames using labels or positions.

DataFrame Operations:

count(), unique(): Count unique values in a column.

sum(), mean(), median(), std(): Compute statistics on columns.

describe(): Generate summary statistics of columns.

info(): Display information about the DataFrame.

Reading and Writing CSV Files Using DataFrames: Pandas makes it easy to read data from CSV files into DataFrames and save DataFrames to CSV files.

Data Analysis Using DataFrame Operations: Pandas allows for efficient data filtering, grouping, and aggregation. You can perform complex operations like merging, pivoting, and reshaping data.

Pandas simplifies data manipulation and analysis, making it an essential tool for any data scientist or analyst.

Both Numpy and Pandas are integral to data analysis workflows in Python. They provide the tools to efficiently work with large datasets, perform calculations, and gain insights from data.

Week 5: Matplotlib and Seaborn

1. Introduction to Matplotlib

Matplotlib is a robust and extensively employed Python library that empowers you to craft static, interactive, and even animated visualizations. It stands as an indispensable tool for generating a diverse array of plots and graphs, making data more comprehensible and insights more accessible.

Plotting Graphs and Subplots:

One of Matplotlib's key strengths is its capacity to create an assortment of graph types, each tailored to represent distinct data relationships:

Line Plots: These plots showcase the progression of data points using connected line segments. Line plots are commonly employed to track trends, patterns, or changes over time.

Scatter Plots: Scatter plots scatter individual data points across the graph, with each point denoting the values of two variables. They are effective in revealing correlations and clusters within data.

Bar Plots: Bar plots depict categorical data through vertical or horizontal bars. They provide an intuitive way to compare values between different categories.

Histograms: Histograms visualize the distribution of continuous data by partitioning it into bins. They help to understand the frequency distribution of data points.

Area Plots: These plots showcase the cumulative data magnitude over a specific range. They are frequently used to highlight the contribution of each category to the whole.

Pie Charts: Pie charts partition data into slices of a circle, each representing a different category's proportion. They are suitable for displaying parts of a whole.

Creating Subplots further enhances your ability to visualize data effectively:

Subplots: A single figure divided into multiple smaller plots, or subplots, allows you to present several graphs side by side. Subplots prove valuable when comparing diverse datasets or when conveying different aspects of the same data in one cohesive layout.

```
import matplotlib.pyplot as plt

# Creating a simple line plot
x = [1, 2, 3, 4, 5]
y = [10, 7, 5, 3, 8]
plt.plot(x, y, marker='o', linestyle='--', color='b', label='Data')

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Sample Line Plot')
plt.legend()

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

Matplotlib's versatility and customization options make it an indispensable tool for crafting compelling visual representations of data, aiding in the exploration and communication of insights.

2. Data Visualization with Matplotlib

Matplotlib's extensive functionality empowers you to create a diverse range of both basic and advanced visualizations. Let's explore some of these visualization types:

Bar Charts:

Bar charts are a highly effective means of representing categorical data and facilitating comparisons between different categories. Matplotlib allows you to create customized bar charts, enhancing the clarity and communicative power of your visualizations.

```
import matplotlib.pyplot as plt
```

```
categories = ['A', 'B', 'C', 'D']
```

```
values = [25, 40, 30, 55]
```

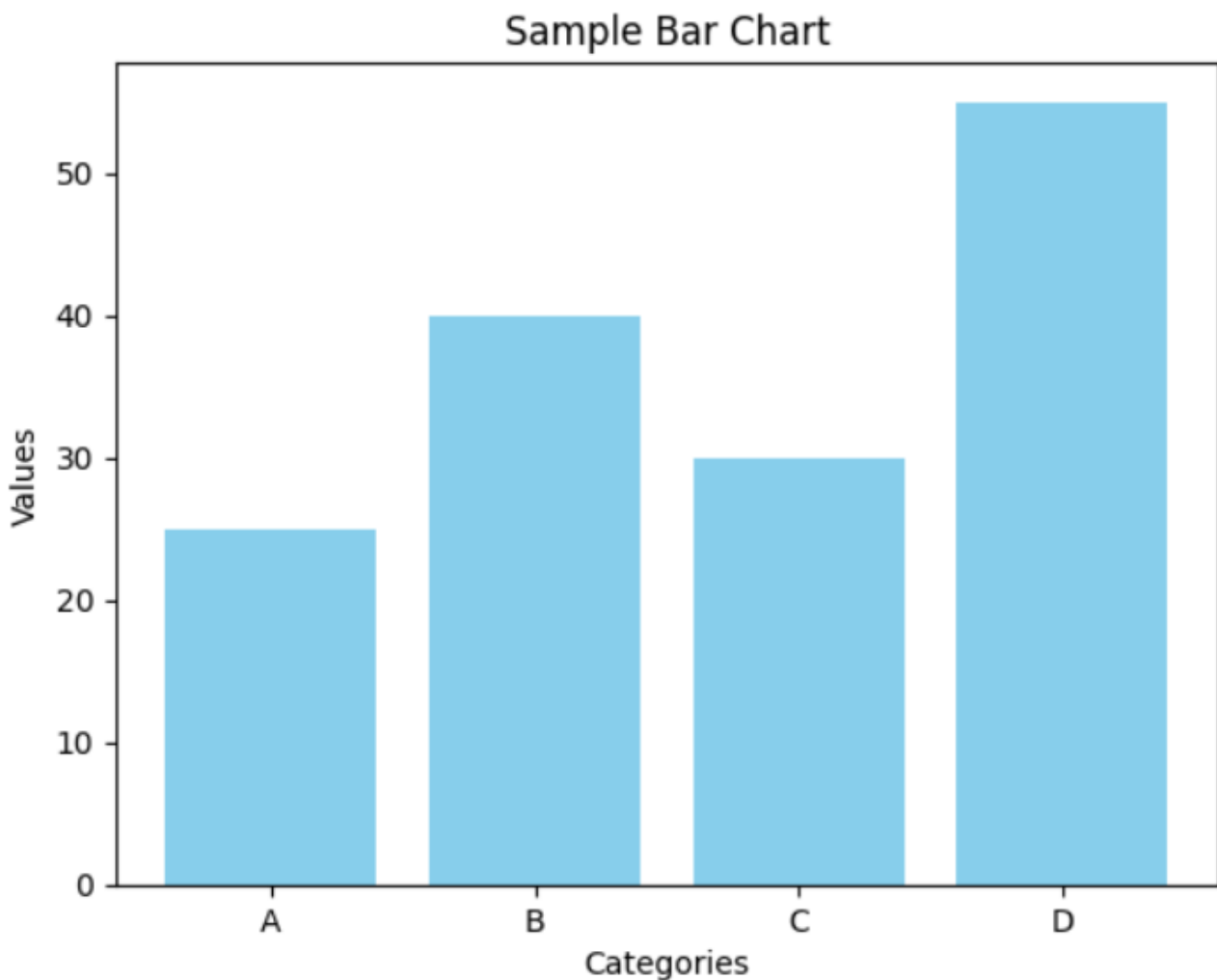
```
plt.bar(categories, values, color='skyblue')
```

```
plt.xlabel('Categories')
```

```
plt.ylabel('Values')
```

```
plt.title('Sample Bar Chart')
```

```
plt.show()
```



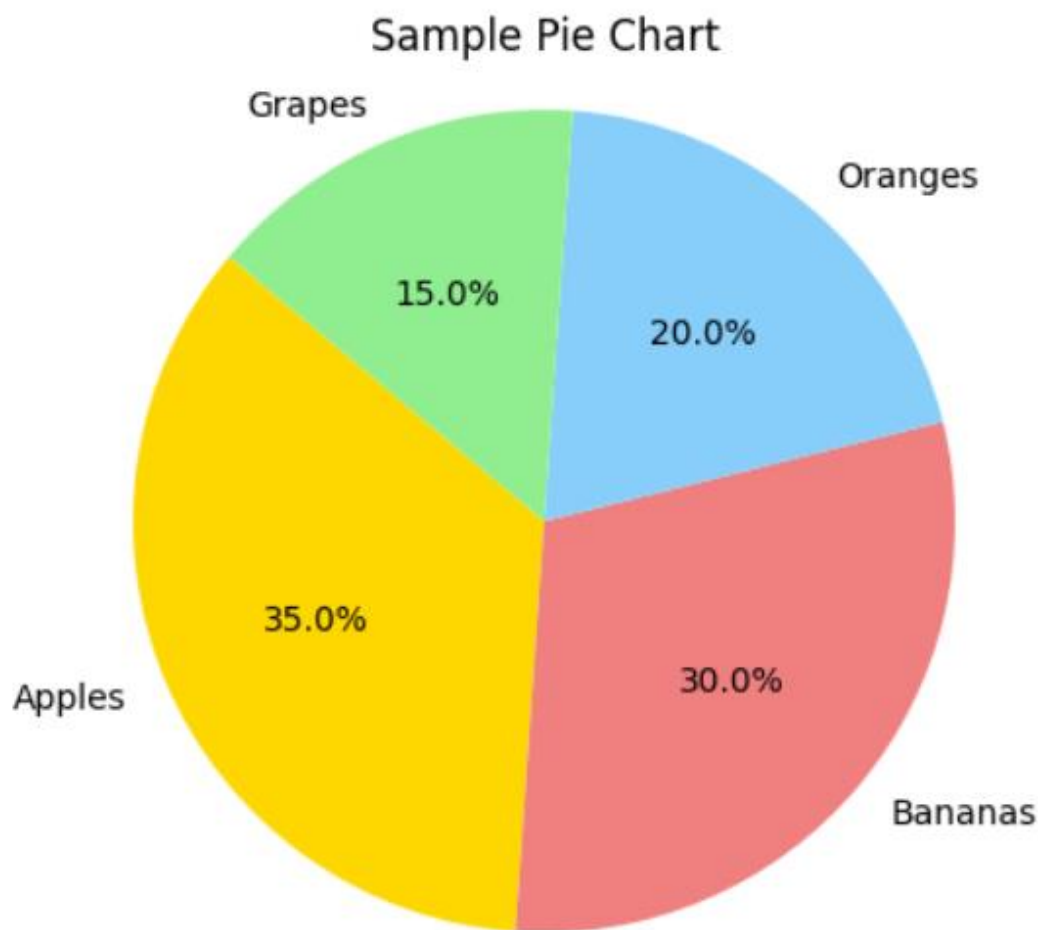
Pie Charts:

Pie charts elegantly portray the proportions of various categories as slices of a circular chart. This visualization type is ideal for illustrating components' share in a whole or comparing relative sizes of different segments.

```
import matplotlib.pyplot as plt

labels = ['Apples', 'Bananas', 'Oranges', 'Grapes']
sizes = [35, 30, 20, 15]
colors = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen']

plt.pie(sizes, labels=labels, colors=colors, autopct='% 1.1f%%', startangle=140)
plt.axis('equal') # Equal aspect ratio ensures the pie chart is circular.
plt.title('Sample Pie Chart')
plt.show()
```



Histograms:

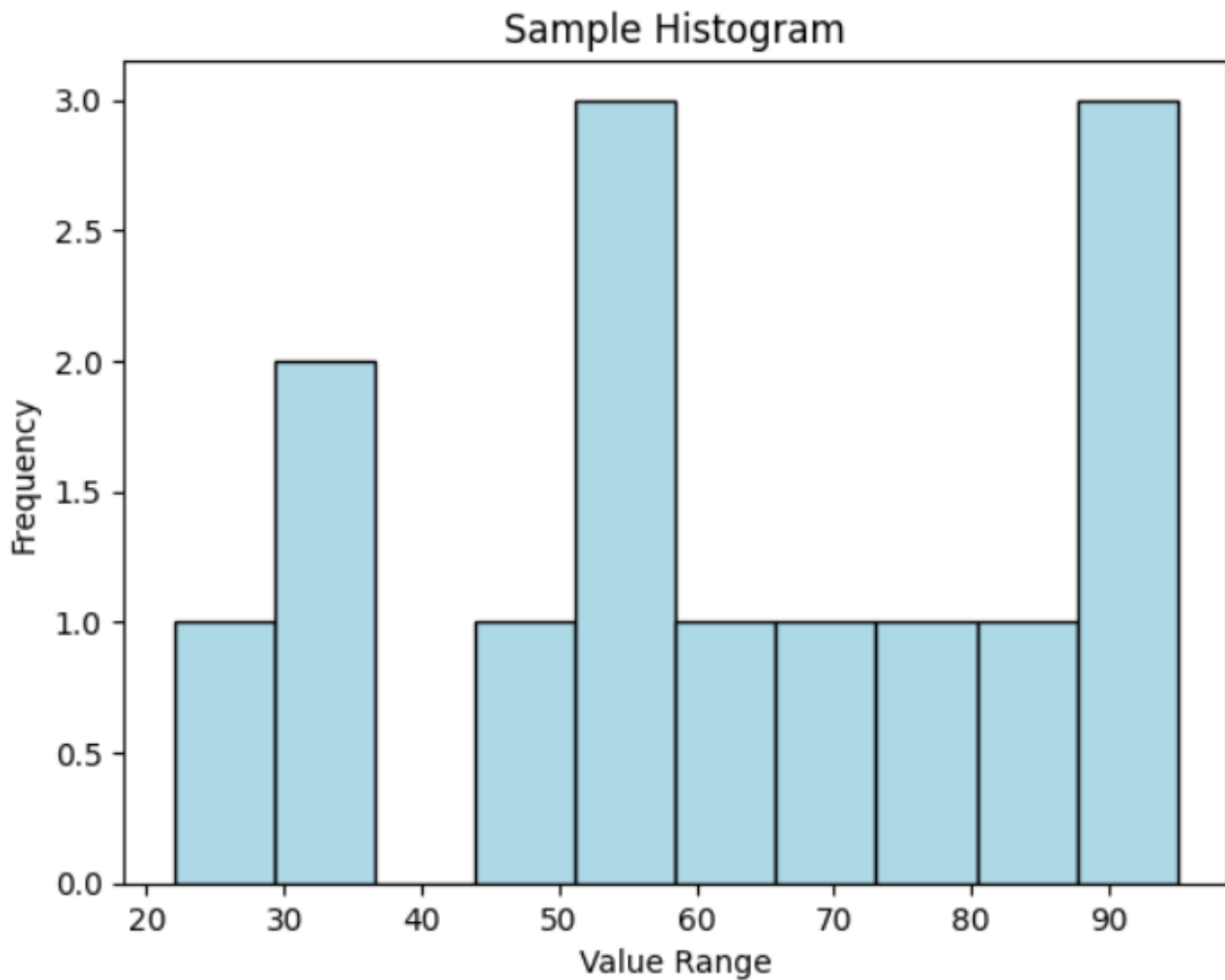
Histograms offer a valuable means of understanding the distribution of numerical data. By dividing data into intervals (bins) and portraying the frequency or density of data points within each bin, histograms enable you to discern patterns and central tendencies in the data.

```
import matplotlib.pyplot as plt
```



```
data = [22, 30, 33, 45, 55, 56, 58, 62, 70, 75, 85, 90, 92, 95]
```

```
plt.hist(data, bins=10, color='lightblue', edgecolor='black')  
plt.xlabel('Value Range')  
plt.ylabel('Frequency')  
plt.title('Sample Histogram')  
plt.show()
```



Scatterplots:

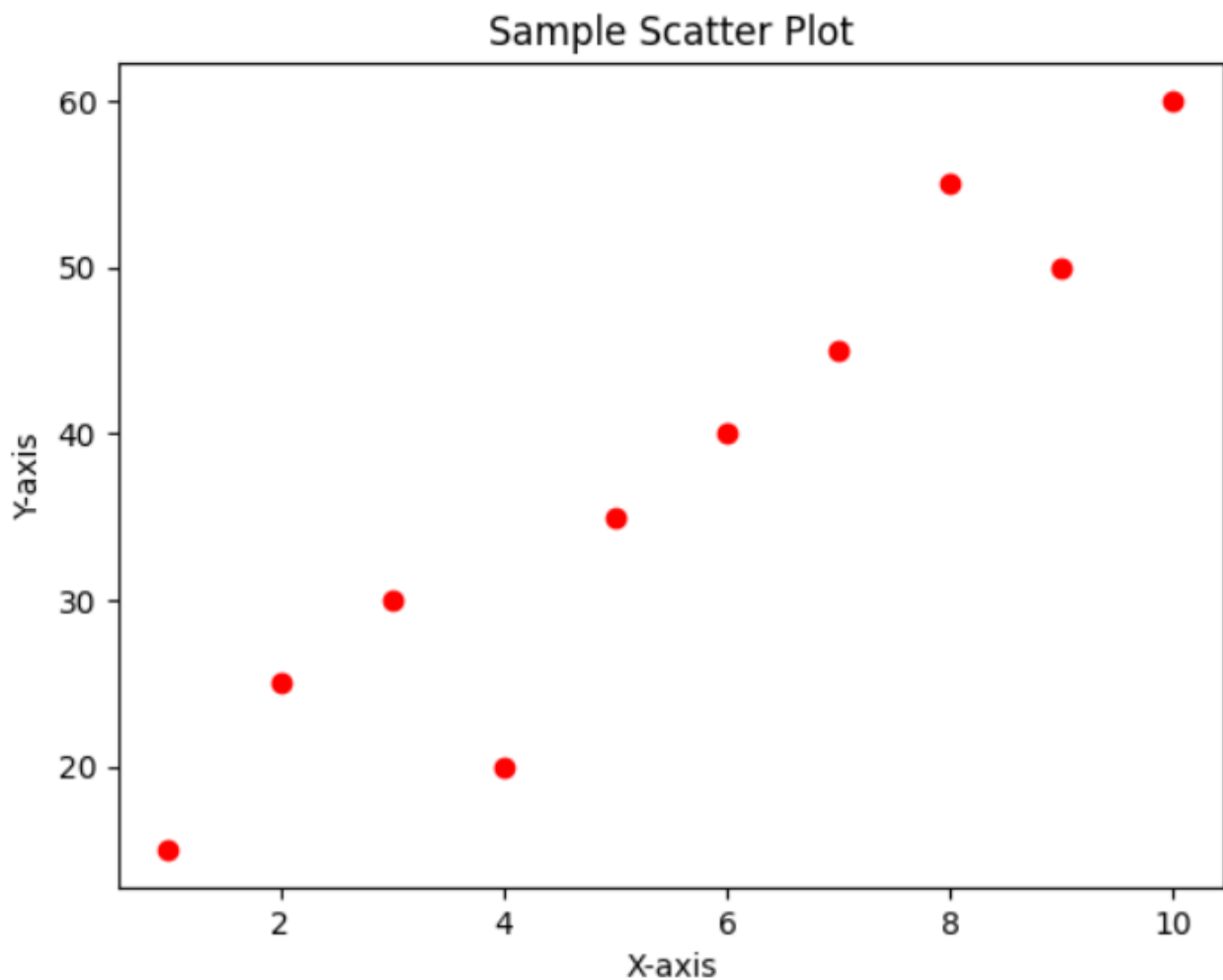
Scatterplots are instrumental in visualizing relationships between two continuous variables. Each data point is represented by a dot on the graph, allowing you to observe patterns, trends, and potential outliers.

```
import matplotlib.pyplot as plt
```

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
y = [15, 25, 30, 20, 35, 40, 45, 55, 50, 60]
```

```
plt.scatter(x, y, color='red', marker='o')  
plt.xlabel('X-axis')
```

```
plt.ylabel('Y-axis')
plt.title('Sample Scatter Plot')
plt.show()
```



Matplotlib's flexibility and versatility empower you to craft compelling visualizations that convey insights effectively. By mastering these visualization types, you'll be better equipped to analyze and communicate your data-driven findings.

3. Introduction to Seaborn

Seaborn is built on top of Matplotlib and designed specifically for statistical data visualization. Seaborn, a library built on top of Matplotlib, is tailored specifically for statistical data visualization. It streamlines the creation of intricate visualizations and offers a high-level interface that simplifies the customization of plots. It simplifies the creation of complex visualizations and provides a high-level interface for customizing plots:

Plotting Using Seaborn:

Seaborn provides functions that enhance the visual aesthetics of plots and simplify the creation of complex visualizations. It offers color palettes, grid styles, and themes that make plots more visually appealing and informative.

Enhanced Aesthetics:

Seaborn offers a collection of color palettes, themes, and grid styles that can be effortlessly applied to plots. This results in visually pleasing and coherent visualizations.

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
# Applying a Seaborn style and color palette
```

```
sns.set(style="whitegrid", palette="pastel")
```

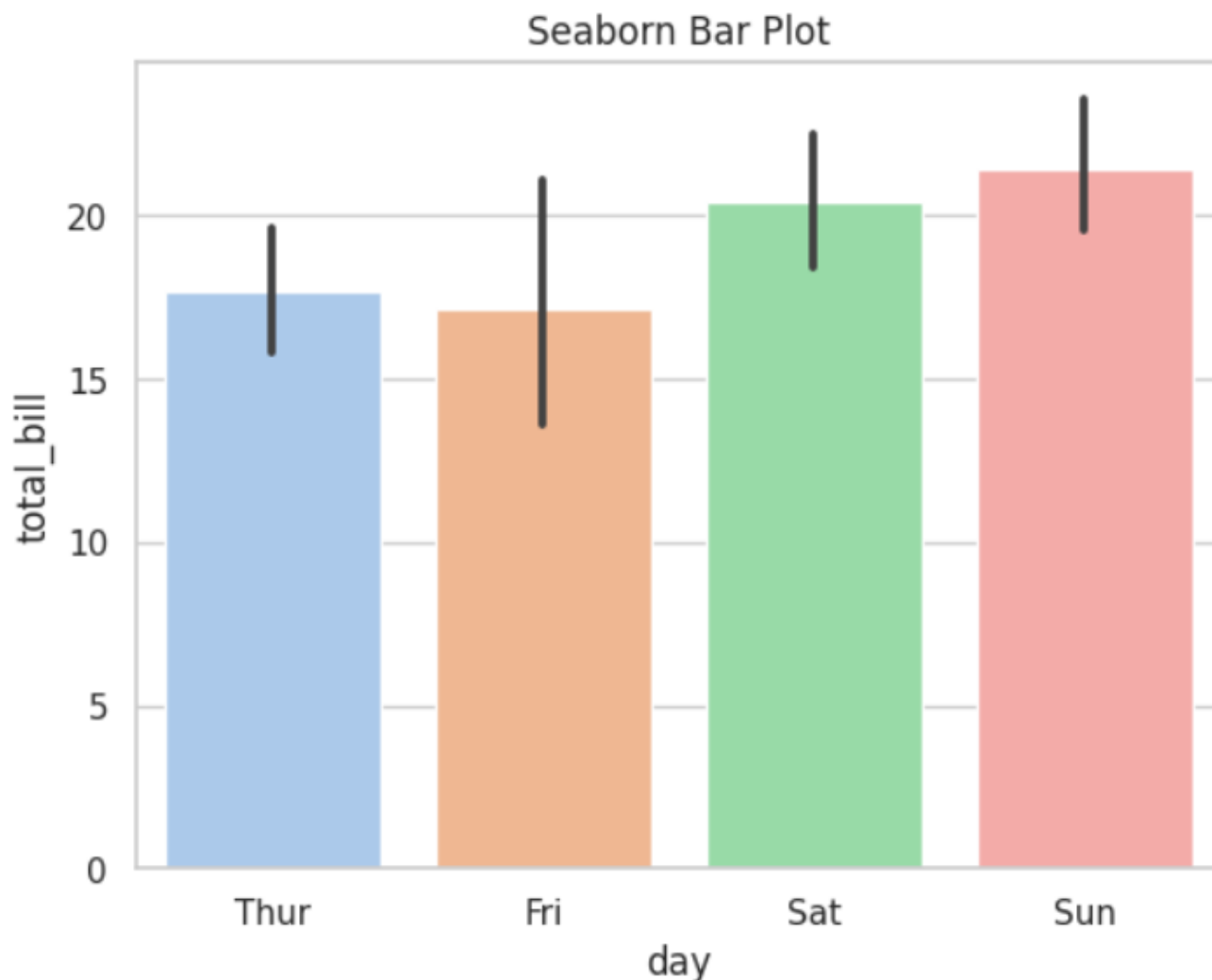
```
# Creating a bar plot with Seaborn
```

```
data = sns.load_dataset("tips")
```

```
sns.barplot(x="day", y="total_bill", data=data)
```

```
plt.title('Seaborn Bar Plot')
```

```
plt.show()
```



Distplot and Pairplot:

The `distplot()` function creates a histogram along with a kernel density estimate, providing a comprehensive view of the distribution of a variable. The `pairplot()` function generates scatter plots for pairs of variables, allowing you to quickly identify relationships and correlations within a dataset.

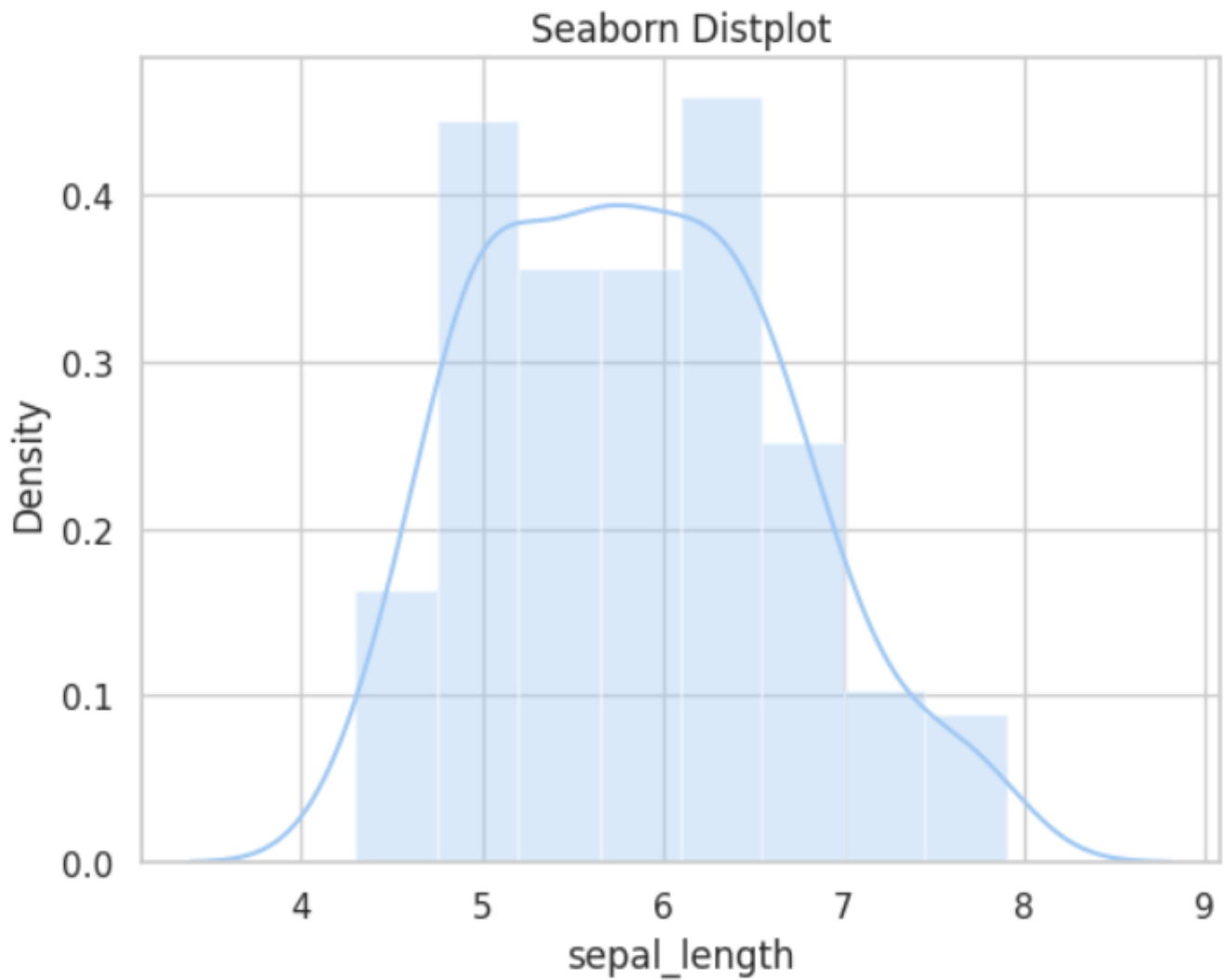
`distplot()`: This function combines a histogram with a kernel density estimate to offer a comprehensive view of the distribution of a variable. It aids in understanding the shape, spread, and central tendencies of data.

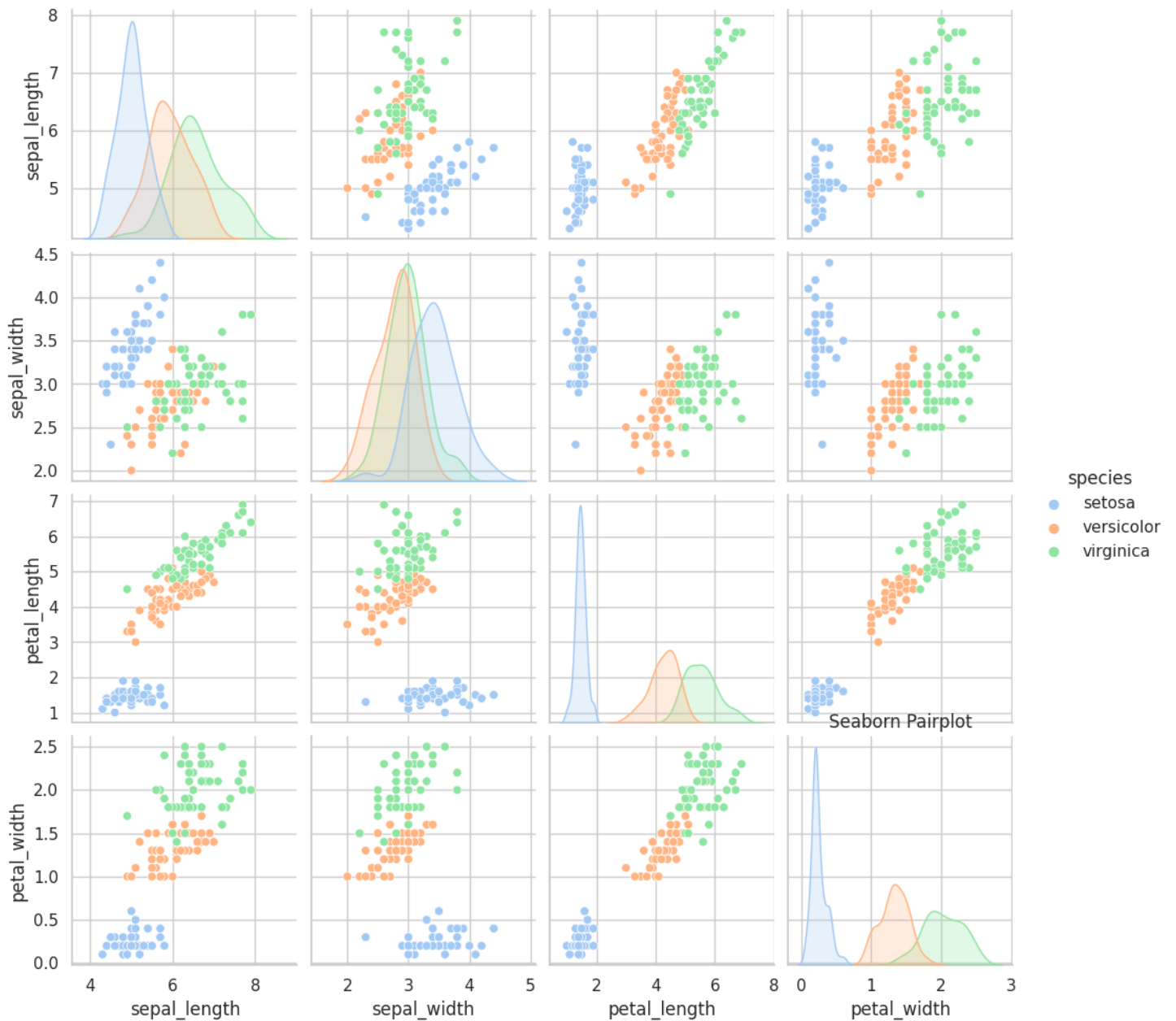
`pairplot()`: The `pairplot` function generates a matrix of scatter plots for pairs of variables. This provides a quick overview of relationships and correlations within a dataset.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Creating a distplot with Seaborn
data = sns.load_dataset("iris")
sns.distplot(data['sepal_length'], kde=True)
plt.title('Seaborn Distplot')
plt.show()

# Creating a pairplot with Seaborn
sns.pairplot(data, hue='species')
plt.title('Seaborn Pairplot')
plt.show()
```





Barplot, Boxplot, Countplot, and Whiskerplot Using CSV Files:

Seaborn's specialized functions, such as `barplot()`, `boxplot()`, and `countplot()`, enable you to create informative categorical plots. These plots are valuable for comparing data across categories and understanding the distribution of values.

barplot(): This function generates bar plots for comparing data across categories, revealing trends and variations in the data.

boxplot(): Seaborn's boxplot function enables you to visualize the distribution of data using quartiles, identifying potential outliers and central tendencies.

countplot(): Countplots tally the occurrences of categorical variables, making it easy to discern the frequency of each category.

whiskerplot(): A whisker plot, often referred to as a violin plot, combines elements of box plots and kernel density plots to provide an informative depiction of data distribution.

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Loading a CSV file into a DataFrame
```

```
data = sns.load_dataset("titanic")
```

```
# Creating a barplot with Seaborn
```

```
sns.barplot(x="class", y="age", data=data)
```

```
plt.title('Seaborn Barplot')
```

```
plt.show()
```

```
# Creating a boxplot with Seaborn
```

```
sns.boxplot(x="class", y="age", data=data)
```

```
plt.title('Seaborn Boxplot')
```

```
plt.show()
```

```
# Creating a countplot with Seaborn
```

```
sns.countplot(x="class", data=data)
```

```
plt.title('Seaborn Countplot')
```

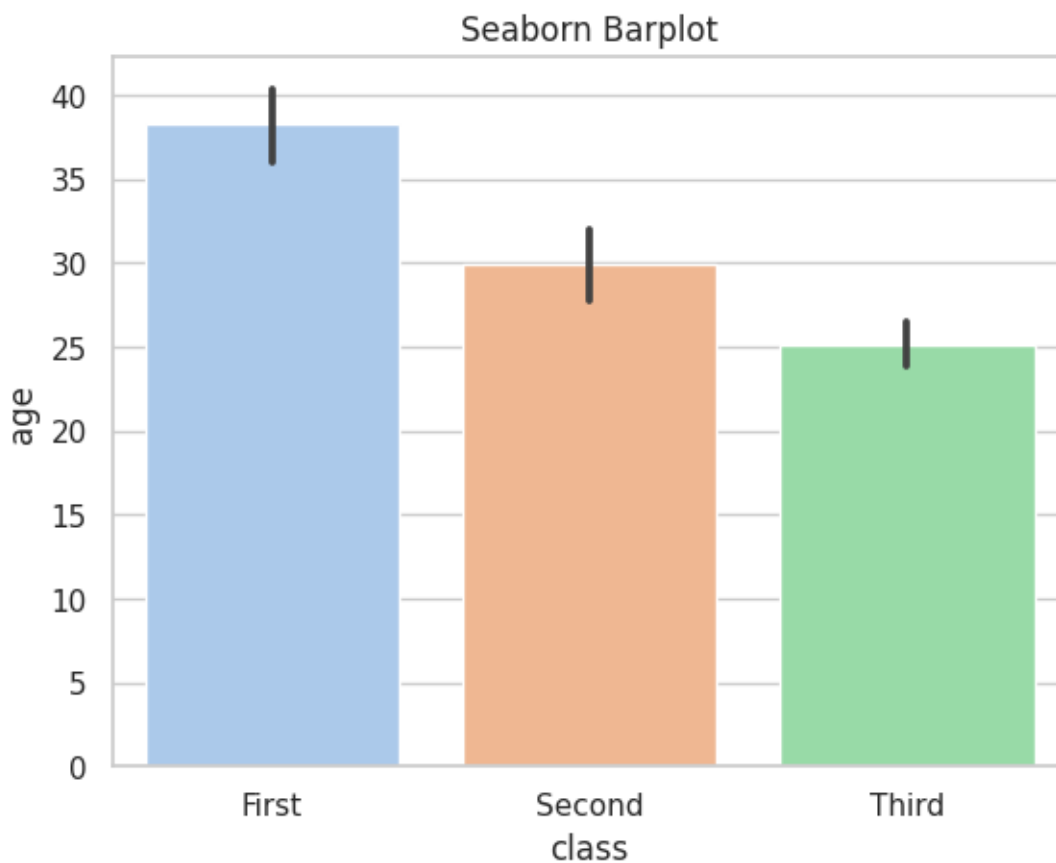
```
plt.show()
```

```
# Creating a whiskerplot with Seaborn
```

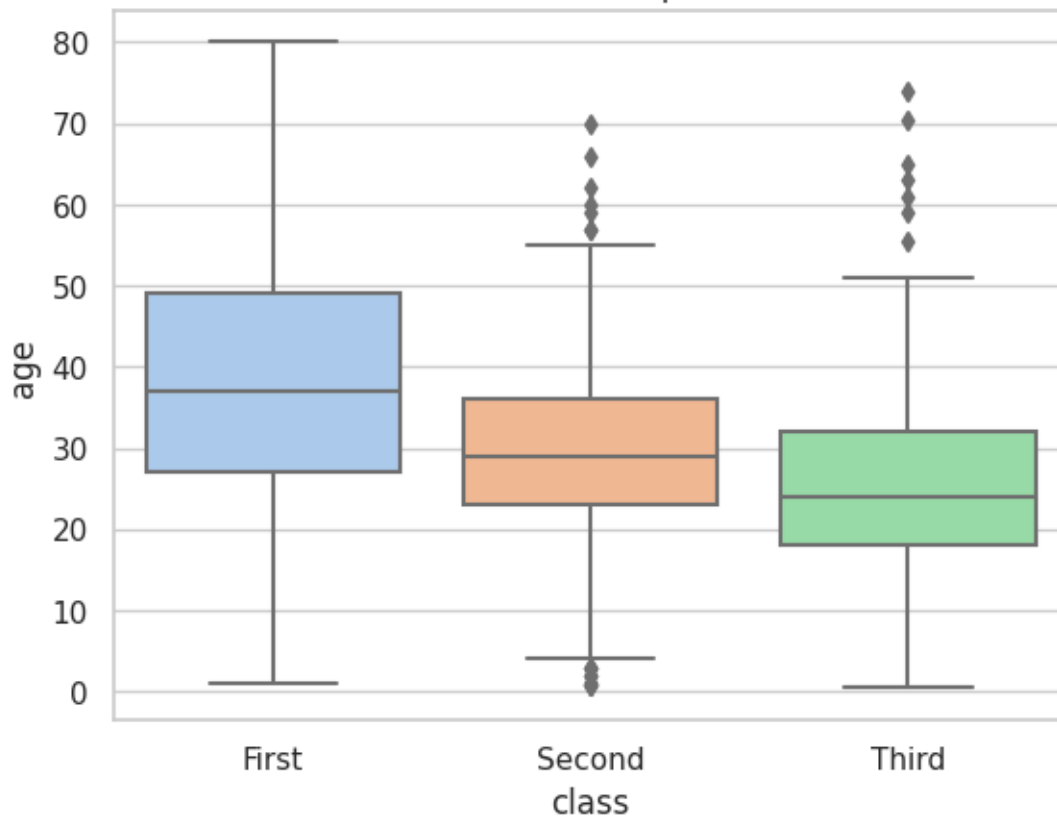
```
sns.violinplot(x="class", y="age", data=data)
```

```
plt.title('Seaborn Whiskerplot')
```

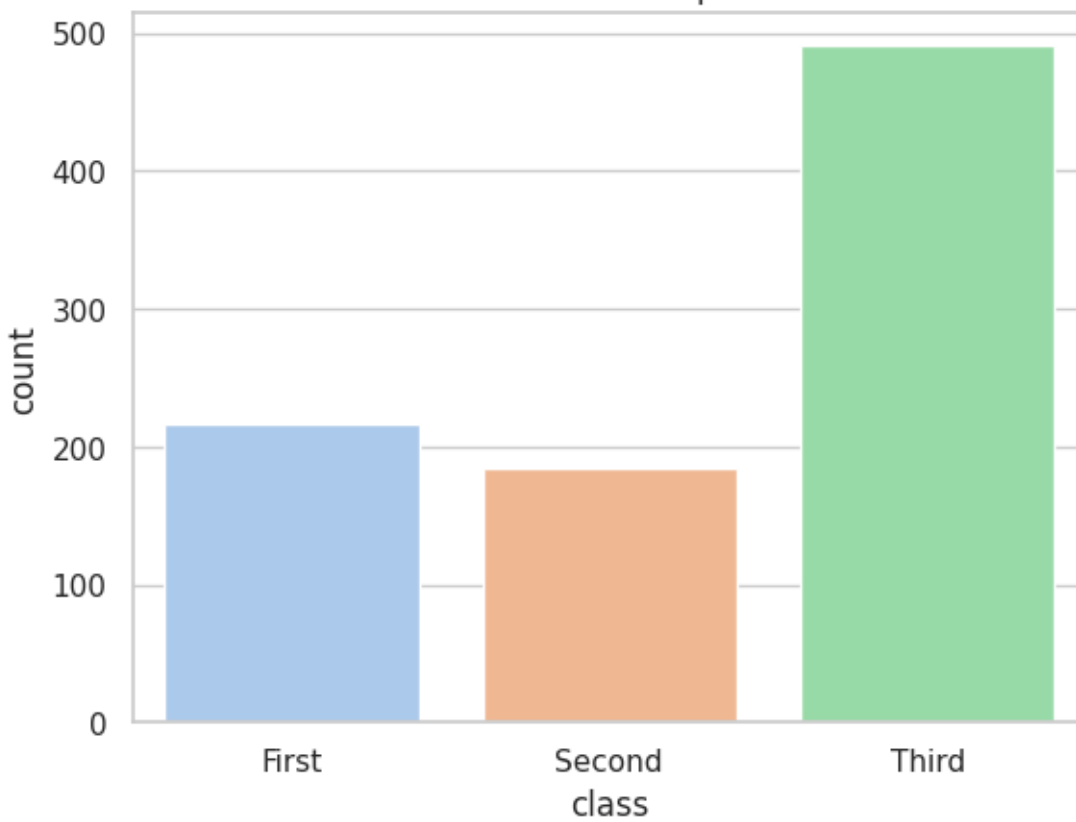
```
plt.show()
```



Seaborn Boxplot



Seaborn Countplot





Correlation Matrix, Heatmap, Pairgrids:

Seaborn's heatmap() function allows you to visualize a correlation matrix, making it easier to identify strong and weak correlations between variables. Pairgrids extend the functionality of pairplots, allowing you to create more comprehensive matrix plots for multiple variables.

heatmap(): The heatmap function generates a color-coded matrix that allows you to visualize the correlations between variables, helping identify strong and weak relationships.

pairgrid(): Pairgrids extend the capabilities of pairplots by enabling more comprehensive matrix plots for multiple variables.

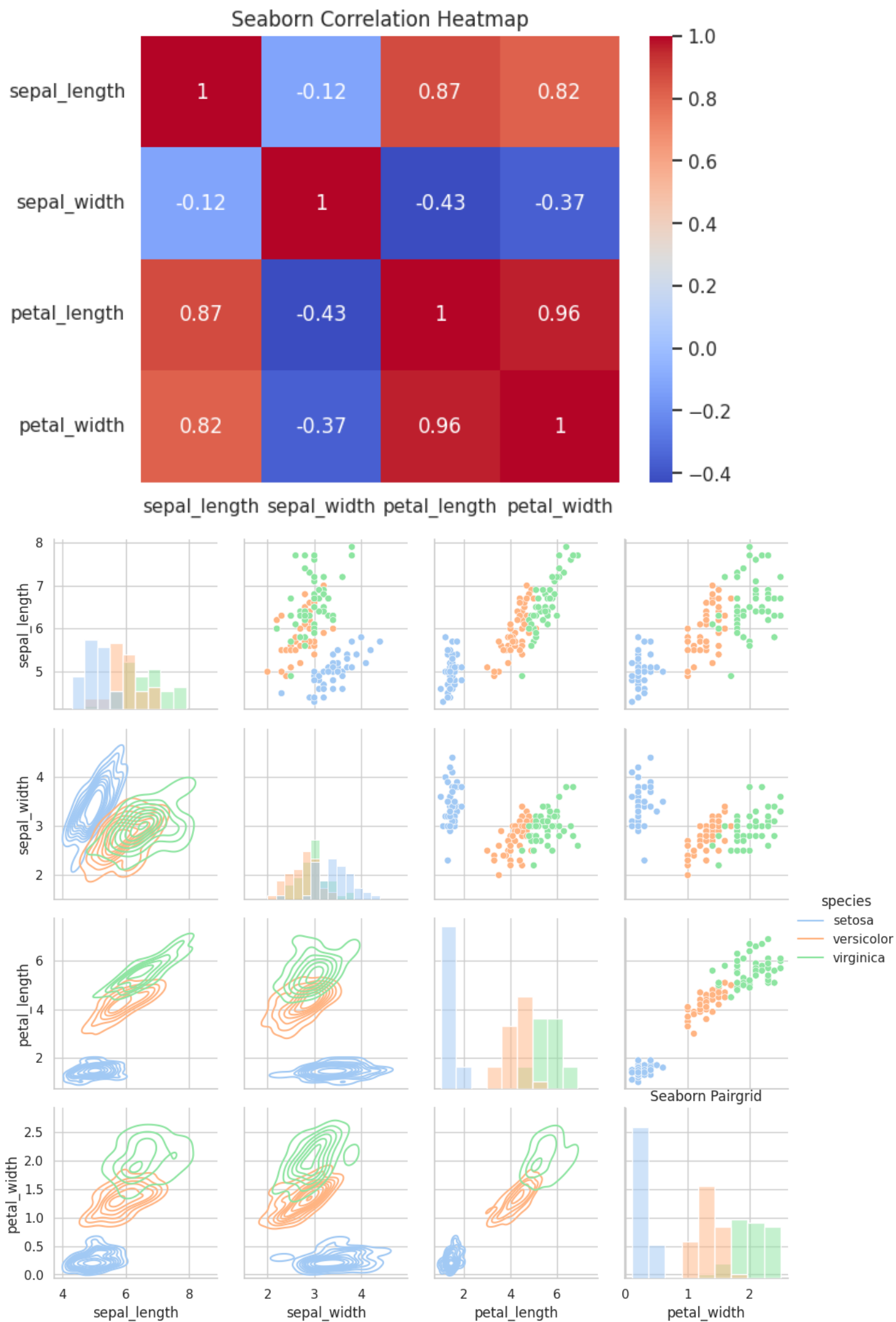
```
import seaborn as sns
import matplotlib.pyplot as plt
```

Creating a correlation heatmap with Seaborn

```
data = sns.load_dataset("iris")
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Seaborn Correlation Heatmap')
plt.show()
```

Creating a pairgrid with Seaborn

```
g = sns.PairGrid(data, hue='species')
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.histplot)
g.add_legend()
plt.title('Seaborn Pairgrid')
plt.show()
```

Seaborn's integration with Matplotlib and its specialized functions make it a powerful tool for statistical data visualization. It simplifies the creation of intricate plots and offers insights into complex relationships within datasets.

Week 6: Project Submission Based on Concepts Learned in Python

Project: Make a modal using machine learning take passengers data those travel in flights and check your modal how much percentage is give correct answer.

Step1. : - open Jupiter notebook and add passengers' data set.

```
In [43]: df=pd.read_csv('Passanger_booking_data.csv')
```

```
In [44]: df
```

```
Out[44]:
```

| | num_passengers | sales_channel | trip_type | purchase_lead | length_of_stay | flight_hour | flight_day | route | booking_origin | wants_extra_baggage | wants |
|-------|----------------|---------------|-----------|---------------|----------------|-------------|------------|--------|----------------|---------------------|-------|
| 0 | 1 | Internet | RoundTrip | 21 | 12 | 6 | Tue | AKLHGH | Australia | 0 | |
| 1 | 2 | Internet | RoundTrip | 262 | 19 | 7 | Sat | AKLDEL | New Zealand | 1 | |
| 2 | 1 | Internet | RoundTrip | 112 | 20 | 3 | Sat | AKLDEL | New Zealand | 0 | |
| 3 | 2 | Internet | RoundTrip | 243 | 22 | 17 | Wed | AKLDEL | India | 1 | |
| 4 | 1 | Internet | RoundTrip | 96 | 31 | 4 | Sat | AKLDEL | New Zealand | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 49997 | 2 | Internet | RoundTrip | 27 | 6 | 9 | Sat | PERPNH | Australia | 1 | |
| 49998 | 1 | Internet | RoundTrip | 111 | 6 | 4 | Sun | PERPNH | Australia | 0 | |
| 49999 | 1 | Internet | RoundTrip | 24 | 6 | 22 | Sat | PERPNH | Australia | 0 | |
| 50000 | 1 | Internet | RoundTrip | 15 | 6 | 11 | Mon | PERPNH | Australia | 1 | |
| 50001 | 1 | Internet | RoundTrip | 19 | 6 | 10 | Thu | PERPNH | Australia | 0 | |

50002 rows x 14 columns

```
In [40]: df.isna().sum().sum()
```

```
Out[40]: 0
```

```
In [45]: df.drop(['sales_channel','trip_type','flight_day','route','booking_origin'],axis= 1,inplace=True)
```

In [46]: df

Out[46]:

| | num_passengers | purchase_lead | length_of_stay | flight_hour | wants_extra_baggage | wants_preferred_seat | wants_in_flight_meals | flight_duration | booking_ |
|-------|----------------|---------------|----------------|-------------|---------------------|----------------------|-----------------------|-----------------|----------|
| 0 | 1 | 21 | 12 | 6 | 0 | 0 | 0 | 7.21 | |
| 1 | 2 | 262 | 19 | 7 | 1 | 0 | 0 | 5.52 | |
| 2 | 1 | 112 | 20 | 3 | 0 | 0 | 0 | 5.52 | |
| 3 | 2 | 243 | 22 | 17 | 1 | 1 | 0 | 5.52 | |
| 4 | 1 | 96 | 31 | 4 | 0 | 0 | 1 | 5.52 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 49997 | 1 | 27 | 6 | 9 | 1 | 0 | 1 | 5.62 | |
| 49998 | 1 | 111 | 6 | 4 | 0 | 0 | 0 | 5.62 | |
| 49999 | 1 | 24 | 6 | 22 | 0 | 0 | 1 | 5.62 | |
| 50000 | 1 | 15 | 6 | 11 | 1 | 0 | 1 | 5.62 | |
| 50001 | 1 | 19 | 6 | 10 | 0 | 1 | 0 | 5.62 | |

50002 rows × 9 columns

In [63]:

```
new = {
    'num_passengers':0,
    'purchase_lead':1,
    'length_of_stay':2,
    'flight_hour':3,
    'wants_extra_baggage':4,
    'wants_preferred_seat':5,
    'wants_in_flight_meals':6,
    'flight_duration':7,
    'booking_complete':8
}
```

In [64]: df.rename(columns=new, inplace=True)

In [65]: df.describe()

Out[65]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| count | 50002.000000 | 50002.000000 | 50002.000000 | 50002.000000 | 50002.000000 | 50002.000000 | 50002.000000 | 50002.000000 | 50002.000000 |
| mean | 1.591256 | 84.940582 | 23.044778 | 9.066277 | 0.668773 | 0.296968 | 0.427143 | 7.277524 | 0.149574 |
| std | 1.020167 | 90.450548 | 33.887171 | 5.412569 | 0.470659 | 0.456927 | 0.494668 | 1.496854 | 0.356657 |
| min | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 4.670000 | 0.000000 |
| 25% | 1.000000 | 21.000000 | 5.000000 | 5.000000 | 0.000000 | 0.000000 | 0.000000 | 5.620000 | 0.000000 |
| 50% | 1.000000 | 51.000000 | 17.000000 | 9.000000 | 1.000000 | 0.000000 | 0.000000 | 7.570000 | 0.000000 |
| 75% | 2.000000 | 115.000000 | 28.000000 | 13.000000 | 1.000000 | 1.000000 | 1.000000 | 8.830000 | 0.000000 |
| max | 9.000000 | 867.000000 | 778.000000 | 23.000000 | 1.000000 | 1.000000 | 1.000000 | 9.500000 | 1.000000 |

In [74]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
col=['0','1','2','3','4','5','6','7']
df_sc=pd.DataFrame(sc.fit_transform(df[col]),columns=col)
```

```
In [75]: df_sc.describe()
```

```
Out[75]:
```

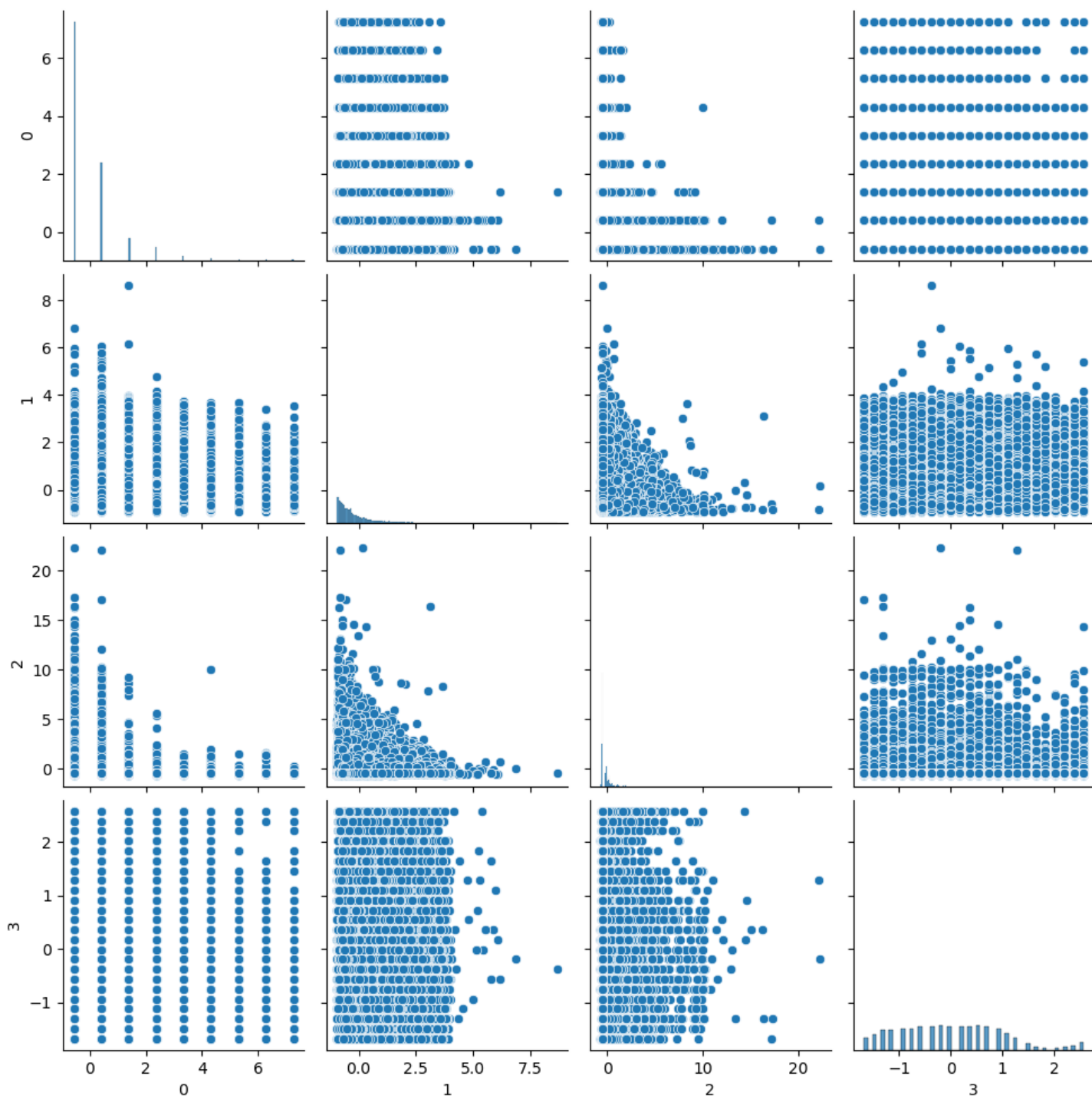
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| count | 5.000200e+04 | 5.000200e+04 | 5.000200e+04 | 5.000200e+04 | 5.000200e+04 | 5.000200e+04 | 5.000200e+04 | 5.000200e+04 |
| mean | -3.637833e-17 | -5.129913e-17 | 1.818917e-17 | 1.156717e-16 | -9.961411e-17 | 1.524764e-16 | -4.973600e-19 | -8.594381e-16 |
| std | 1.000010e+00 | 1.000010e+00 | 1.000010e+00 | 1.000010e+00 | 1.000010e+00 | 1.000010e+00 | 1.000010e+00 | 1.000010e+00 |
| min | -5.795739e-01 | -9.390925e-01 | -6.800511e-01 | -1.675058e+00 | -1.420943e+00 | -6.499312e-01 | -8.635022e-01 | -1.742021e+00 |
| 25% | -5.795739e-01 | -7.069191e-01 | -5.325012e-01 | -7.512732e-01 | -1.420943e+00 | -6.499312e-01 | -8.635022e-01 | -1.107350e+00 |
| 50% | -5.795739e-01 | -3.752429e-01 | -1.783813e-01 | -1.224520e-02 | 7.037579e-01 | -6.499312e-01 | -8.635022e-01 | 1.953956e-01 |
| 75% | 4.006674e-01 | 3.323332e-01 | 1.462285e-01 | 7.267828e-01 | 7.037579e-01 | 1.538624e+00 | 1.158075e+00 | 1.037170e+00 |
| max | 7.262356e+00 | 8.646352e+00 | 2.227872e+01 | 2.574353e+00 | 7.037579e-01 | 1.538624e+00 | 1.158075e+00 | 1.484780e+00 |

```
In [76]: df_sc.shape
```

```
Out[76]: (50002, 8)
```

```
In [77]: import seaborn as sns  
from matplotlib import pyplot as plt
```


```
In [79]: sns.pairplot(df_sc[['0','1','2','3']])
```



```
In [81]: from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(df_sc, df['8'], test_size=0.3)
print(xtrain.shape, xtest.shape)

(35001, 8) (15001, 8)
```

```
In [82]: from sklearn.linear_model import Perceptron
p=Perceptron()
p.fit(xtrain,ytrain)
```

Out[82]: Perceptron()
 Snipping Tool
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
In [83]: prediction=p.predict(xtest)
```

```
In [84]: from sklearn.metrics import accuracy_score
print("Testing accuracy is ",accuracy_score(ytest,prediction))

Testing accuracy is  0.667355096326912
```

```
In [85]: pred_train=p.predict(xtrain)
print("Training accuracy is: ",accuracy_score(ytrain,pred_train))

Training accuracy is:  0.6666095254421303
```

Reason for choosing data analysis and visualization using python.

Choosing to pursue data analysis and visualization using Python offers a multitude of compelling reasons, each contributing to its popularity and effectiveness in the field. Here are several key reasons why individuals often opt for Python for data analysis and visualization:

Rich Ecosystem of Libraries: Python boasts a vast collection of powerful libraries specifically designed for data analysis and visualization, such as Pandas, NumPy, Matplotlib, Seaborn, and more. These libraries provide ready-to-use tools for data manipulation, exploration, and representation.

Ease of Learning and Use: Python's intuitive syntax is easy to learn, making it an ideal language for both beginners and experienced programmers. Its readability and straightforwardness facilitate efficient code development, even for complex tasks.

Wide Adoption: Python is extensively used across various industries, from tech and finance to healthcare and academia. This widespread adoption has led to a rich community of practitioners, online resources, and continuous development.

Data Handling Capabilities: Pandas, a core library for data analysis, excels at handling structured data, whether in tabular formats or databases. It simplifies data cleaning, transformation, and aggregation.

CONCLUSION

Practical knowledge means the visualization of the knowledge, which we read in our books. For this, we perform experiments and get observations. Practical knowledge is very important in every field. One must be familiar with the problems related to that field so that he may solve them and become a successful person.

After achieving the proper goal in life, an engineer must enter in professional life.

According to this life, he has to serve an industry, may be public or private sector or self own.

For the efficient work in the field, he must be well aware of the practical knowledge. as well as theoretical knowledge.

Due to all above reasons and to bridge the gap between theory and practical, us

Engineering curriculum provides a practical training of 45 days. During this period a student work in the industry and get well all type of experience and knowledge about the working of companies and hardware and software tools.

I have undergone my 45 days summer training.

This report is based on the knowledge, which I acquired during my 45 days of summer training.