Practical File

April 15, 2025

Sudeep Kumar Singh

28021

BSc Computer Science Hons.

Unit:-1

1. Introduction to Python

Definition: Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used for web development, data science, automation, and machine learning.

[ ]:
```python
#Python Implementation:-
# Printing Hello, World!
print("Hello, World!")

print("Python is a powerful language capable of Artificial Intelligence and␣
  ↪Machine Learning.")
print("It is widely used for data science, automation, and web development.")
print("=" * 80)
```

```
Hello, World!
Python is a powerful language capable of Artificial Intelligence and Machine
Learning.
It is widely used for data science, automation, and web development.
================================================================================
```

2. Python vs. Excel

Definition: Python and Excel are both used for data analysis, but Python provides automation, scalability, and better handling of large datasets compared to Excel's manual operations.

- **Automation:** Python allows users to automate repetitive tasks and complex workflows, reducing manual effort and errors common in Excel.
- **Scalability:** Python can efficiently process and analyze large datasets that may slow down or crash Excel.
- **Advanced Analytics:** Python supports advanced statistical, machine learning, and visualization libraries, offering more powerful analysis than Excel's built-in tools.

[5]:
```
#Python Implementation: Using Pandas to read an Excel file.
# Reading the Excel file named 'data.xlsx' in the current directory
import pandas as pd
data = pd.read_excel("indian_data.xlsx")
print(data.head())
print("=" * 80)
```

```
      Name   Age  Salary       City
0  Karthik  30.0   50000  Bengaluru
1   Ananya  24.0   60000     Mumbai
2    Meena   NaN   45000      Delhi
3    Meena  24.0   45000      Delhi
4  Karthik  30.0   50000  Bengaluru
================================================================================
```

3. Anaconda and Jupyter Notebook: Interface Overview

Definition:

Anaconda is a popular Python distribution designed for scientific computing and data science. Jupyter Notebook is an interactive environment that allows you to write code, visualize results, and add markdown notes—all in one place. It supports live code execution, making it ideal for data analysis, visualization, and documentation.

[6]:
```
#Python Implementation: Running a simple Python cell in Jupyter Notebook.
# Simple arithmetic operation
print(5 + 9)
print(3 * 85)
print(67 - 70)
print(104 / 11)
print("=" * 80)
```

```
14
255
-3
9.454545454545455
================================================================================
```

4. Data Types in Python

Definition: Python supports various data types, including integers, floats, strings, lists, tuples, sets, and dictionaries.

[7]:
```
#Python Implementation:-
# Different data types in Python (Indian example)
integer_val = 75  # e.g., age of a person
float_val = 98.6  # e.g., average temperature in India
string_val = "Delhi"  # e.g., name of a city
list_val = ["Mumbai", "Chennai", "Kolkata", "Delhi"]  # list of Indian metro␣
 ↪cities
```

```
tuple_val = ("Karthik", 30, "Bengaluru")  # tuple with name, age, city
dict_val = {"name": "Ananya", "age": 24, "city": "Mumbai"}  # dictionary with␣
↪Indian context

print(type(integer_val), type(float_val), type(string_val), type(list_val),␣
↪type(tuple_val), type(dict_val))

print("=" * 80)
```

```
<class 'int'> <class 'float'> <class 'str'> <class 'list'> <class 'tuple'>
<class 'dict'>
================================================================================
```

5. Python Basic Syntax

Assignment Statements and Variables

[8]:
```
# Assigning values to variables
x = 10
y = 90
z = x + y
print("Sum:", z)

print("=" * 80)
```

```
Sum: 100
================================================================================
```

[9]:
```
# Assiging alphabetic values to variables
a = "Sudeep Kumar"
b = "Singh"
print("Concatenate two strings", a + " " + b)
print("Replicating a string", a*2)
print("=" * 80)
```

```
Concatenate two strings Sudeep Kumar Singh
Replicating a string Sudeep KumarSudeep Kumar
================================================================================
```

Indentation

[99]:
```
# Proper indentation in Python
print("Indentation is required for sub statement.")
print("It is four space or a tab.")
if True:
    print("This is indented correctly!")

print("=" * 80)
```

```
Indentation is required for sub statement.
It is four space or a tab.
This is indented correctly!
================================================================================
```

Conditionals

[100]:
```python
# If-Else Condition
num = 10
if num > 0:
    print("Positive number")
else:
    print("Negative number")

print("=" * 80)
```

```
Positive number
================================================================================
```

[101]:
```python
# If-Else Condition
age = int(input("Enter your age: "))
if age >= 18:
    print("You may drive any vehicle")
else:
    print("You cannot drive any vehicle")

print("=" * 80)
```

```
Enter your age:  45
```

```
You may drive any vehicle
================================================================================
```

[14]:
```python
# If-Else Ladder
marks = int(input("Enter your marks"))
if marks >= 90:
    print("You got grade \"O\"")
elif marks >= 80 and marks< 90:
    print("You got grade \"A+\"")
elif marks >= 70 and marks< 80:
    print("You got grade \"A\"")
elif marks >= 60 and marks< 70:
    print("You got grade \"B+\"")
elif marks >= 50 and marks< 60:
    print("You got grade \"B\"")
elif marks >= 40 and marks< 50:
    print("You got grade \"C+\"")
elif marks >= 33 and marks< 40:
    print("You got grade \"C\"")
```

```
else:
     print("You are Fail. Better luck next time!")

print("=" * 80)
```

```
You got grade "O"
================================================================================
```

Loops - For & While

[103]:
```
# For loop
for i in range(5):
    print(i)

# While loop
count = 0
while count < 5:
    print(count)
    count += 1

print("=" * 80)
```

```
0
1
2
3
4
0
1
2
3
4
================================================================================
```

[15]:
```
#For loop
num = int(input("Enter the number which table you want to print"))
for i in range(1, 11):
    print(f"{num} x {i} = {num * i}")

print("=" * 80)
```

```
99 x 1 = 99
99 x 2 = 198
99 x 3 = 297
99 x 4 = 396
99 x 5 = 495
99 x 6 = 594
99 x 7 = 693
```

```
99 x 8 = 792
99 x 9 = 891
99 x 10 = 990
================================================================================
```

[16]:

```
num = int(input("Enter the number which table you want to print"))
i = 1

while i <= 10:
    print(f"{num} x {i} = {num * i}")
    i += 1
```

```
13 x 1 = 13
13 x 2 = 26
13 x 3 = 39
13 x 4 = 52
13 x 5 = 65
13 x 6 = 78
13 x 7 = 91
13 x 8 = 104
13 x 9 = 117
13 x 10 = 130
```

User-Defined Functions

[18]:

```
#Basic Function (No Parameters, No Return Value)
def greet():
    print(f"Hello, {a}! Welcome to Python functions.")

# Calling the function
greet()
```

```
Hello, Sudeep Kumar! Welcome to Python functions.
```

[19]:

```
#Function with Parameters and Return Value
def add(a, b):
    return a + b

# Calling the function
result = add(10, 5)
print("Sum:", result)
```

```
Sum: 15
```

[21]:

```
#Function with Default Arguments
def greet_user(name="Programmer"):
    print(f"Hello, {name}!")
```

```python
greet_user("Sudeepwebdev")    # With argument
greet_user()           # Uses default value
```

```
Hello, Sudeepwebdev!
Hello, Programmer!
```

[23]:
```python
#Function with List Argument (Process Multiple Items)
def print_odd_numbers(numbers):
    print("Odd numbers:")
    for num in numbers:
        if num % 2 != 0:
            print(num, end=" ")

# Calling the function
nums = [1, 2, 3, 4, 5, 6]
print_odd_numbers(nums)
```

```
Odd numbers:
1 3 5
```

[27]:
```python
#Recursive Function (Factorial Example)
def factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

# Calling the recursive function
print("Factorial of 5 is:", factorial(5))
```

```
Factorial of 5 is: 120
```

[106]:
```python
# Function to calculate square of a number
def square(num):
    return num ** 2

print(square(4))

print("=" * 80)
```

```
16
================================================================================
```

6. Working with Libraries

Pandas

[24]:
```python
#Creating DataFrame from a List of Dictionaries
import pandas as pd
```

```
# Creating a DataFrame from list of dictionaries with Indian names and 'EName'
data = [
    {"EName": "Karthik", "Age": 30, "City": "Bengaluru"},
    {"EName": "Ananya", "Age": 24, "City": "Mumbai"}
]
df1 = pd.DataFrame(data)
print(df1)
print("=" * 80)
```

```
     EName  Age       City
0  Karthik   30  Bengaluru
1   Ananya   24     Mumbai
================================================================================
```

[26]:

```
#Creating DataFrame from a Dictionary of Series
import pandas as pd

# Creating Series for each column
names = pd.Series(["SUmit", "Farhan"])
ages = pd.Series([40, 22])
cities = pd.Series(["Chicago", "Delhi"])

# Creating DataFrame
data = {
    "Name": names,
    "Age": ages,
    "City": cities
}
df2 = pd.DataFrame(data)
print(df2)
print("=" * 80)
```

```
     Name  Age     City
0   SUmit   40  Chicago
1  Farhan   22    Delhi
================================================================================
```

[109]:

```
#Creating DataFrame from List of Lists with Custom Column Names
import pandas as pd

# List of lists
data = [
    ["George", 45, "Seattle"],
    ["Hannah", 32, "Boston"]
]

# Specifying column names
```

```
columns = ["Name", "Age", "City"]

# Creating DataFrame
df3 = pd.DataFrame(data, columns=columns)
print(df3)
print("=" * 80)
```

```
     Name  Age     City
0  George   45  Seattle
1  Hannah   32   Boston
================================================================================
```

[110]:

```
import pandas as pd

# Creating a DataFrame
data = {"Name": ["Alice", "Bob"], "Age": [25, 30]}
df = pd.DataFrame(data)
print(df)
print("=" * 80)
```

```
    Name  Age
0  Alice   25
1    Bob   30
================================================================================
```

NumPy

[111]:

```
#Creating an Array with np.arange()
import numpy as np

# Array with range of numbers
arr2 = np.arange(1, 6)  # [1, 2, 3, 4, 5]
print("Array from arange multiplied by 3:\n", arr2 * 3)
print("=" * 80)
```

```
Array from arange multiplied by 3:
 [ 3  6  9 12 15]
================================================================================
```

[112]:

```
#Creating an Array with np.zeros() and Broadcasting
import numpy as np

# Array filled with zeros
arr3 = np.zeros((2, 3))  # 2x3 matrix of zeros
print("Zeros array plus 5:\n", arr3 + 5)
print("=" * 80)
```

```
Zeros array plus 5:
```

```
 [[5. 5. 5.]
 [5. 5. 5.]]
================================================================================
```

[113]:

```
#Creating a 2D NumPy Array (Matrix)
import numpy as np

# 2D Array (Matrix)
arr1 = np.array([[1, 2], [3, 4]])
print("2D Array multiplied by 2:\n", arr1 * 2)
print("=" * 80)
```

```
2D Array multiplied by 2:
 [[2 4]
 [6 8]]
================================================================================
```

[114]:

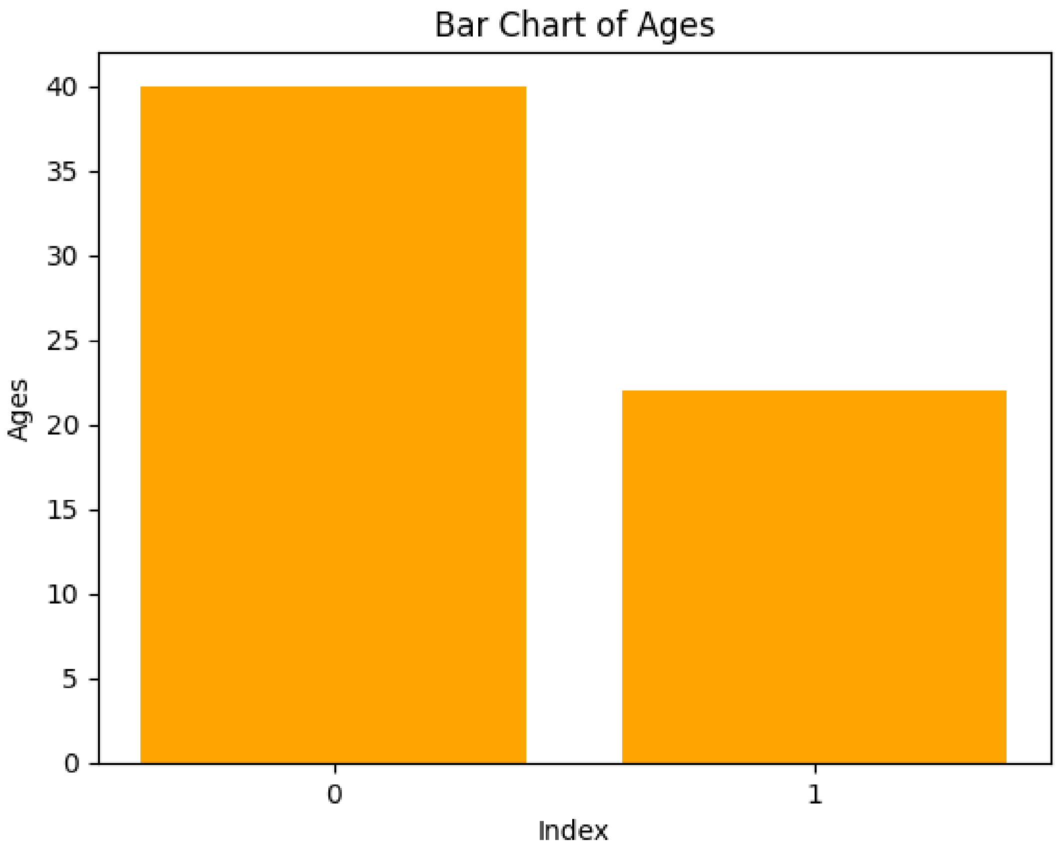
```
import numpy as np

# Creating a NumPy array
arr = np.array([1, 2, 3, 4])
print(arr * 2)

print("=" * 80)
```

```
[2 4 6 8]
================================================================================
```

Matplotlib

[27]:

```
import matplotlib.pyplot as plt

# Use the 'ages' Series for the bar chart
plt.bar(ages.index.astype(str), ages.values, color='orange')
plt.xlabel("Index")
plt.ylabel("Ages")
plt.title("Bar Chart of Ages")
plt.show()

print("=" * 80)
```



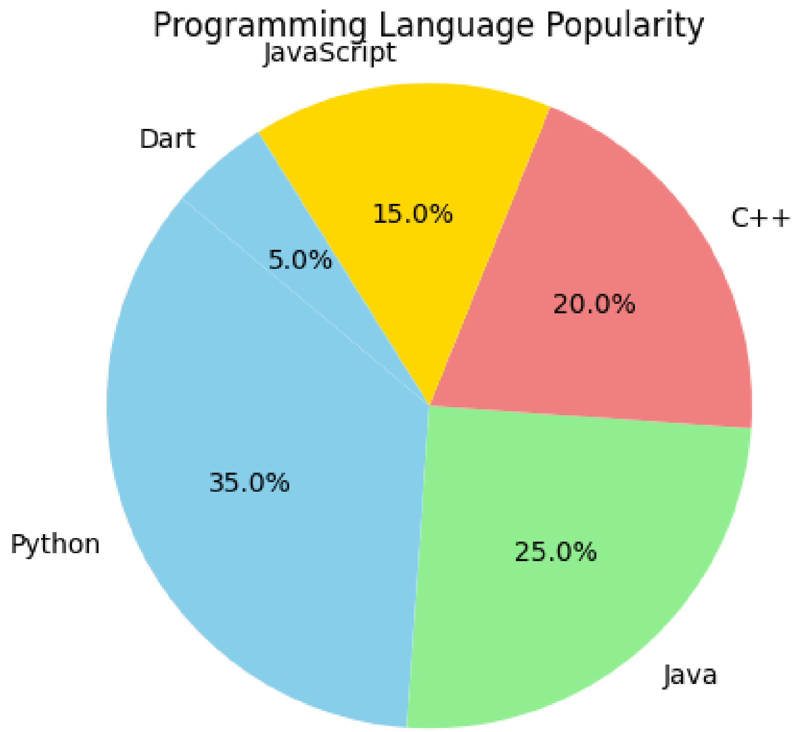
==

[ ]:

```

```

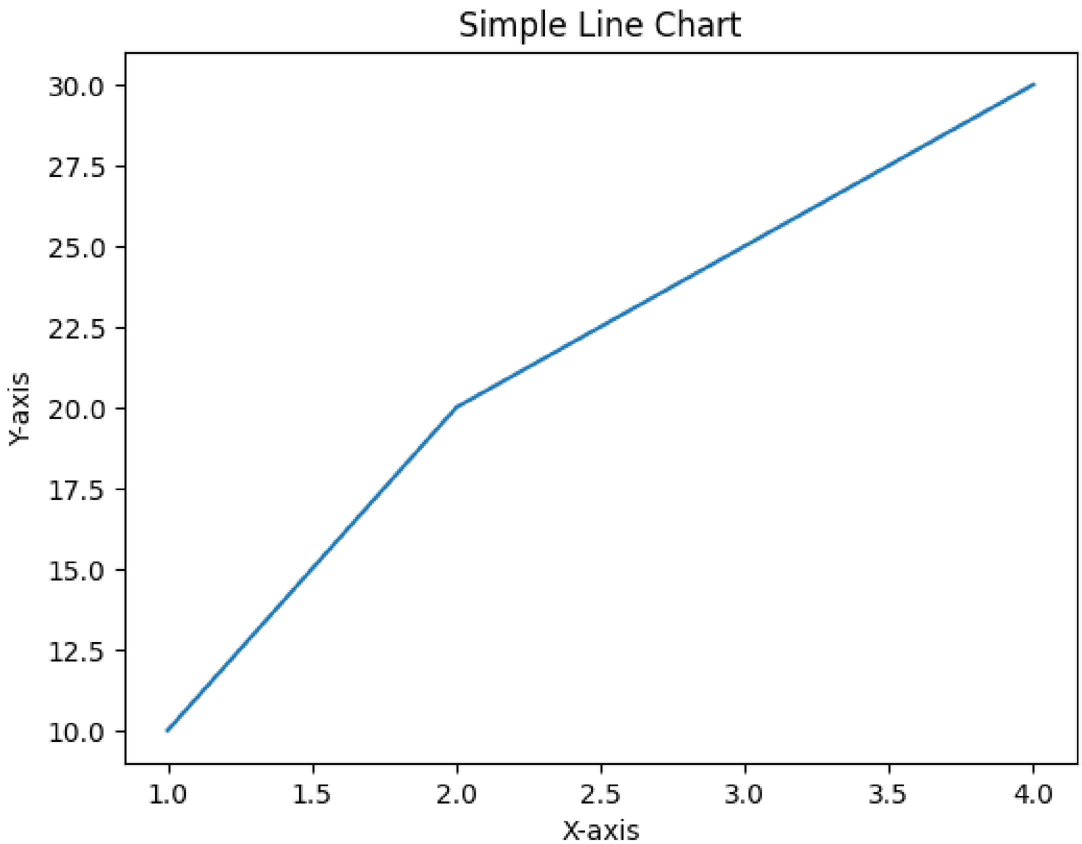
[ ]:

```
#Pie Chart
import matplotlib.pyplot as plt

# Data for pie chart
labels = ['Python', 'Java', 'C++', 'JavaScript', 'Dart']
sizes = [35, 25, 20, 15 ,5]
colors = ['skyblue', 'lightgreen', 'lightcoral', 'gold']

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title("Programming Language Popularity")
plt.axis('equal')
plt.show()

print("=" * 80)
```



==

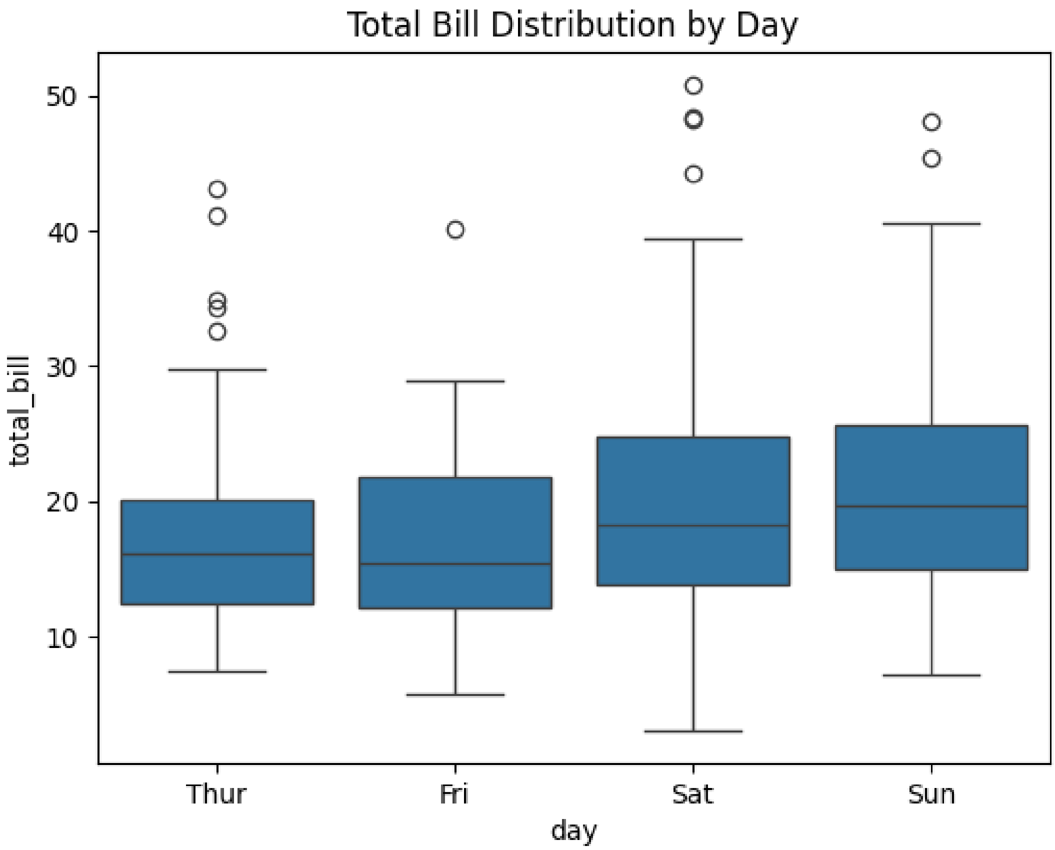
[117]:
```
import matplotlib.pyplot as plt

# Plotting a simple line chart
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Line Chart")
plt.show()

print("=" * 80)
```



==

Seaborn

[118]:
```
 #Box Plot
import seaborn as sns
import matplotlib.pyplot as plt

# Load sample dataset
data = sns.load_dataset("tips")

# Creating a box plot
sns.boxplot(x="day", y="total_bill", data=data)
plt.title("Total Bill Distribution by Day")
plt.show()

print("=" * 80)
```



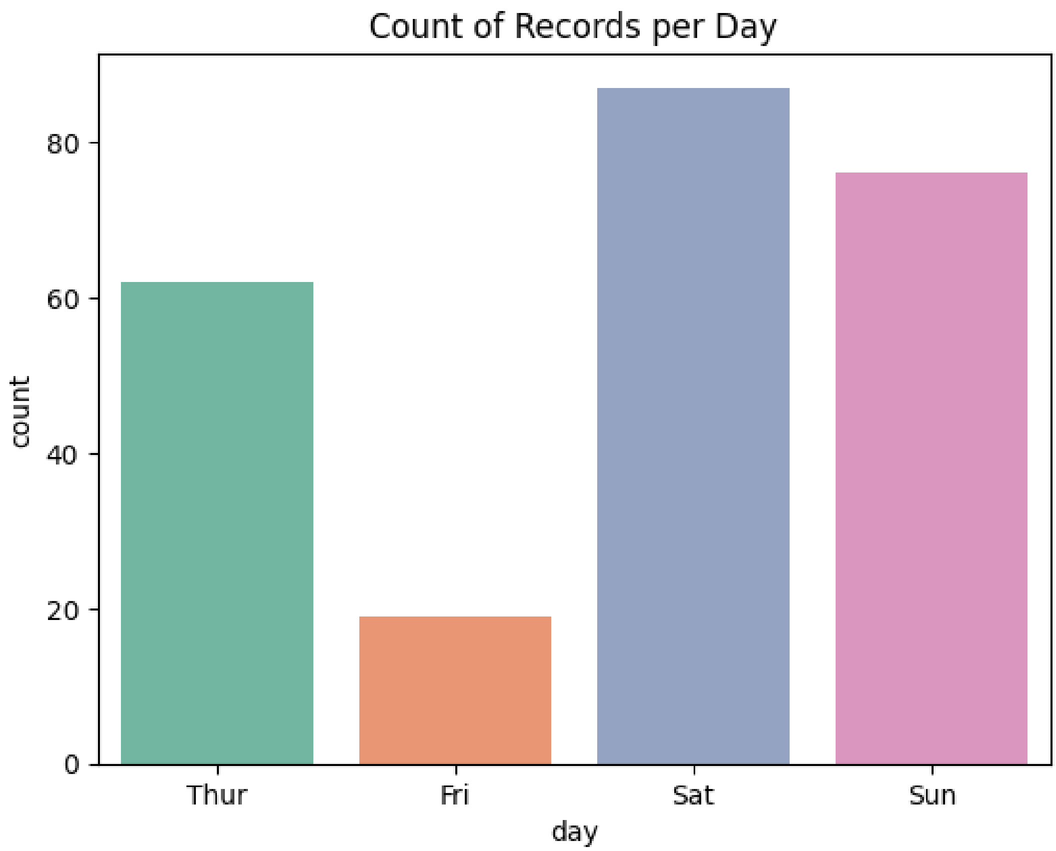

```
================================================================================
```

[119]:

```
#Count Plot
import seaborn as sns
import matplotlib.pyplot as plt

# Load sample dataset
data = sns.load_dataset("tips")

# Creating a count plot
sns.countplot(x="day", data=data, palette="Set2")
plt.title("Count of Records per Day")
plt.show()

print("=" * 80)
```

```
/var/folders/mz/ypdsq4nd3mqg289r8kwqv3_40000gn/T/ipykernel_24662/2938865471.py:9
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

```
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x="day", data=data, palette="Set2")
```

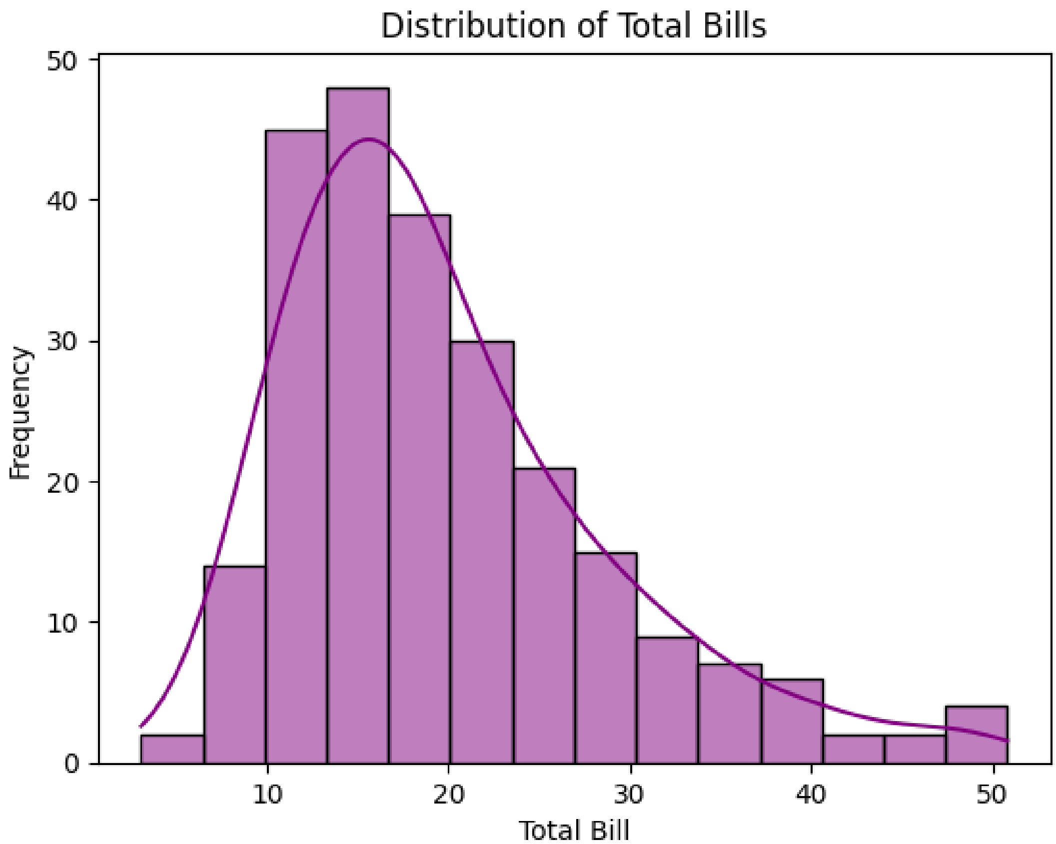


==

[ ]:

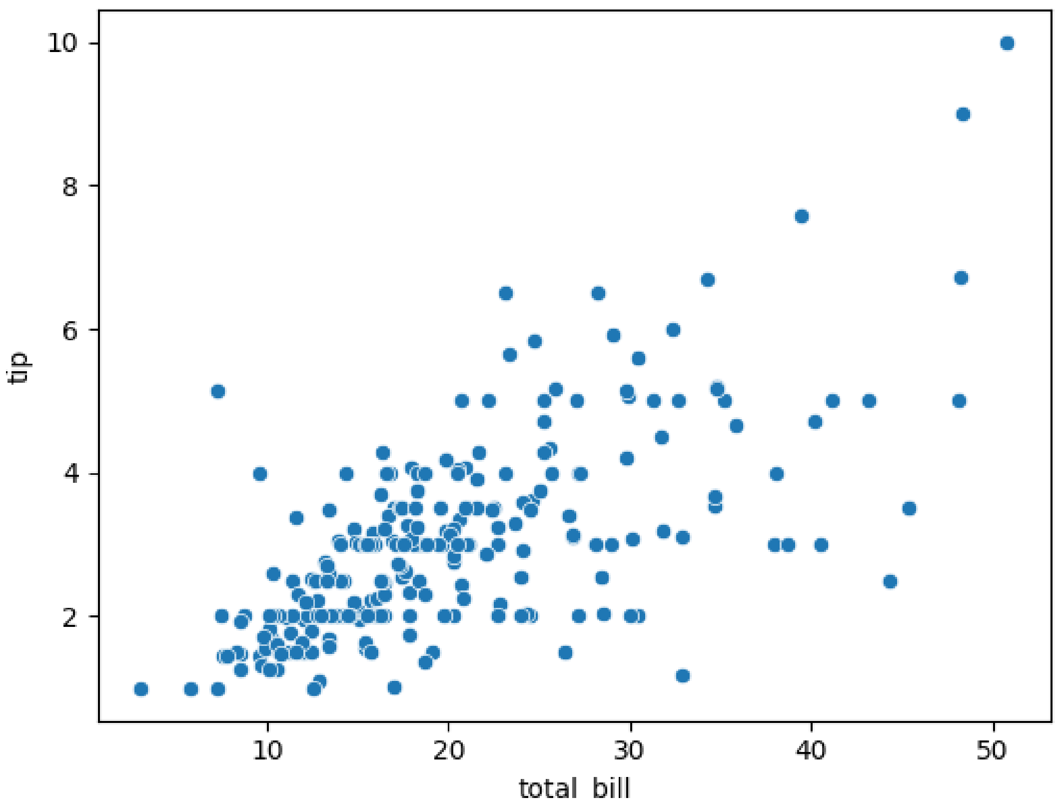
```python
#Histogram with KDE (Distribution Plot)
import seaborn as sns
import matplotlib.pyplot as plt

# Load sample dataset
data = sns.load_dataset("tips")

# Creating a histogram with KDE
sns.histplot(data["total_bill"], kde=True, color="purple")
plt.title("Distribution of  Bills")
plt.xlabel("Total Bill")
plt.ylabel("Frequency")
plt.show()
```

```
print("=" * 80)
```



==

[121]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset("tips")

# Creating a scatter plot
sns.scatterplot(x="total_bill", y="tip", data=data)
plt.show()

print("=" * 80)
```



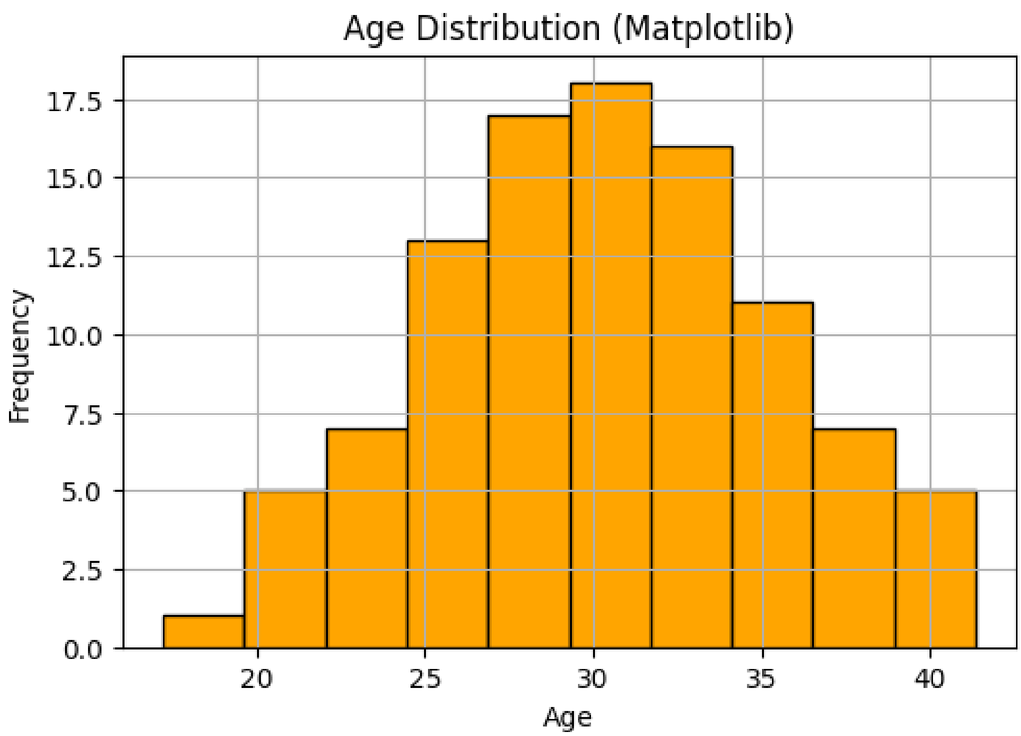
==

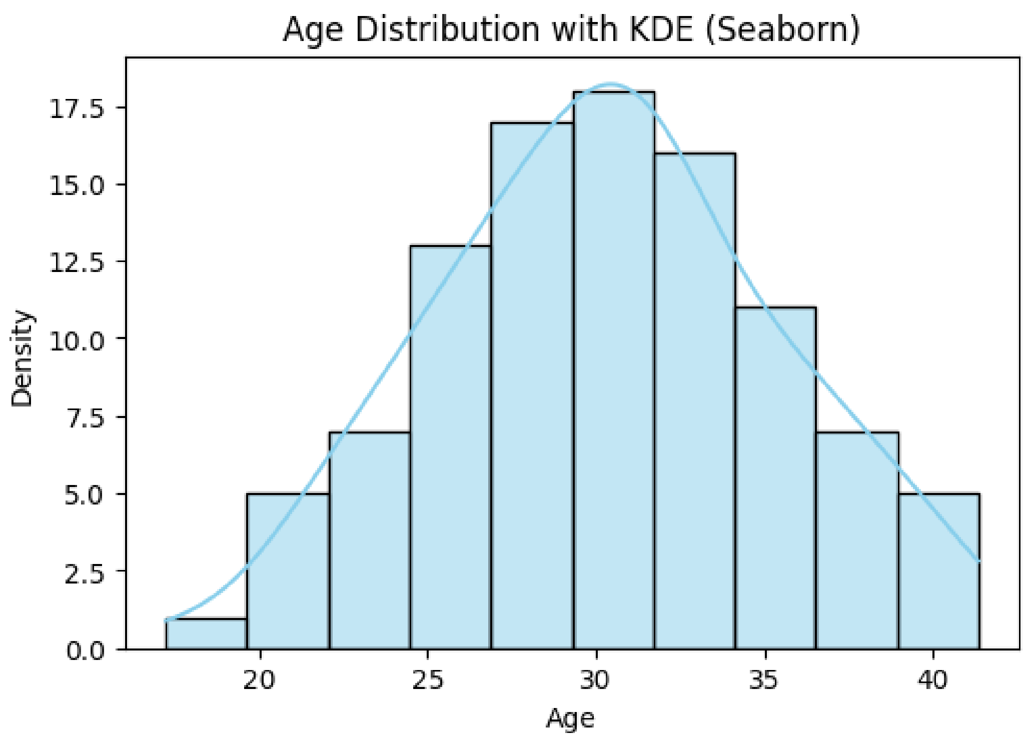
Comparison between Matplotlib and Seaborn

1. Matplotlib offers more control and is lower-level (you design everything).
2. Seaborn is built on top of Matplotlib and provides prettier, more statistically-informed plots with less code.

[164]:
```
#Histogram with KDE (Distribution Plot)
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Sample data: Age distribution
np.random.seed(0)
ages = np.random.normal(loc=30, scale=5, size=100)

# ------------------ Matplotlib ------------------
plt.figure(figsize=(6, 4))
plt.hist(ages, bins=10, color='orange', edgecolor='black')
plt.title("Age Distribution (Matplotlib)")
```

```
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()

# ------------------ Seaborn ------------------
plt.figure(figsize=(6, 4))
sns.histplot(ages, bins=10, kde=True, color='skyblue')
plt.title("Age Distribution with KDE (Seaborn)")
plt.xlabel("Age")
plt.ylabel("Density")
plt.show()

print("=" * 80)
```



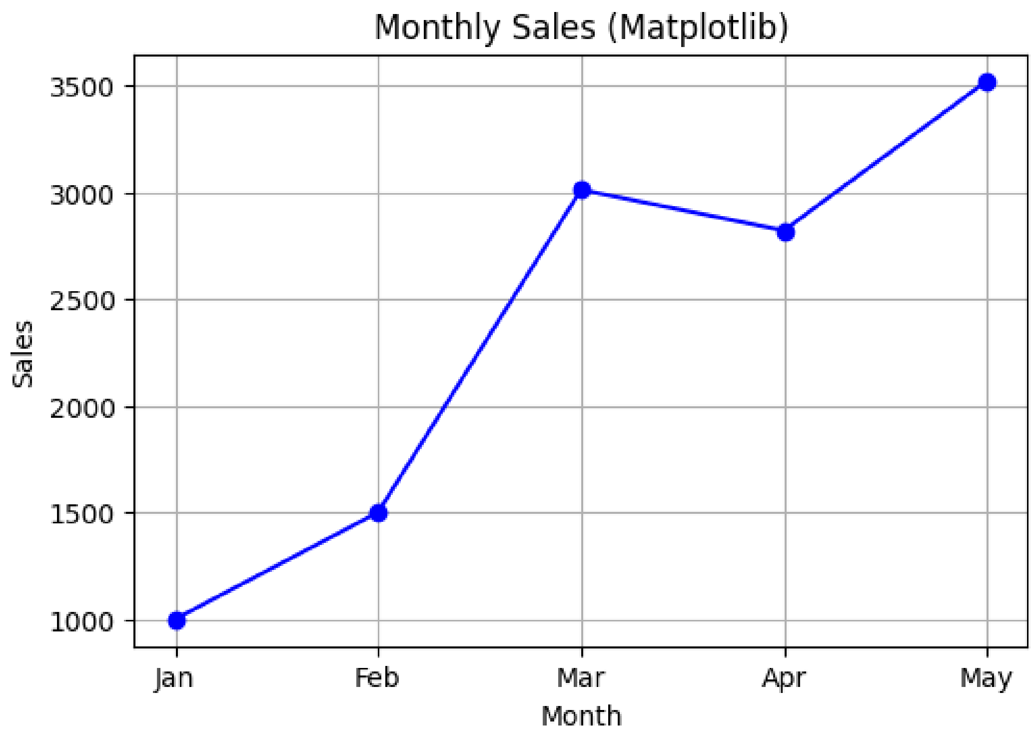


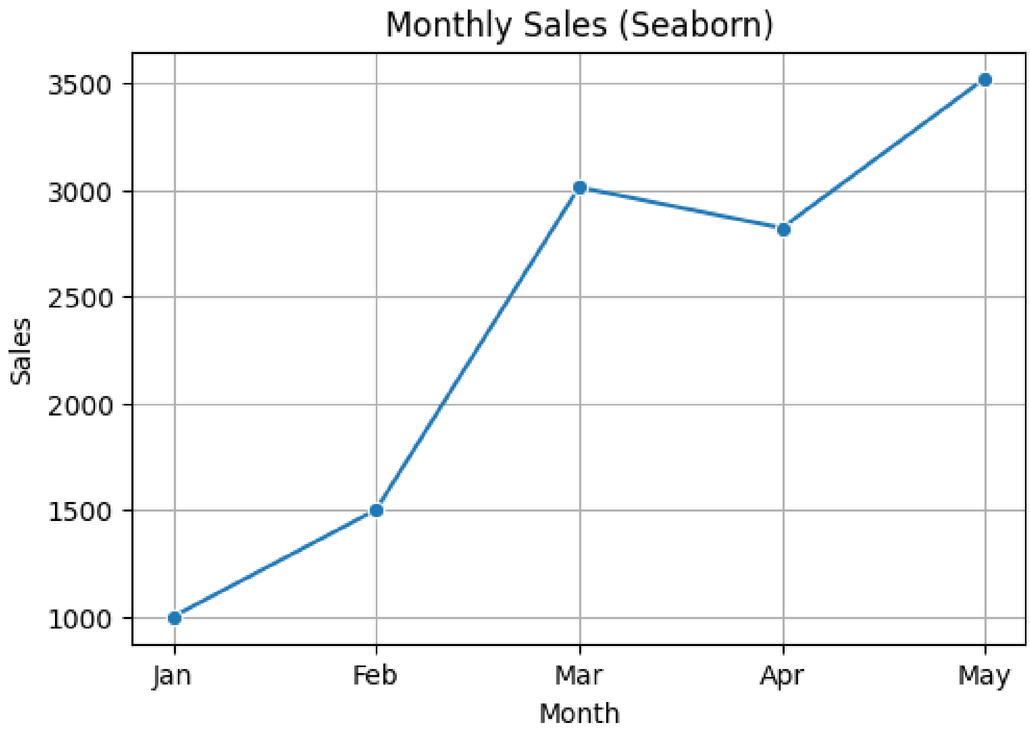
==

[29]:

```python
#Line Plot - Sales Over Months
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Sample dataset
data = {
    'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May'],
    'Sales': [1000, 1500, 3010, 2820, 3520]
}
df = pd.DataFrame(data)

# ------------------ Matplotlib ------------------
plt.figure(figsize=(6, 4))
plt.plot(df['Month'], df['Sales'], marker='o', color='blue')
plt.title("Monthly Sales (Matplotlib)")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.grid(True)
plt.show()
```

```python
# ------------------ Seaborn ------------------
plt.figure(figsize=(6, 4))
sns.lineplot(x='Month', y='Sales', data=df, marker='o')
plt.title("Monthly Sales (Seaborn)")
plt.xlabel("Month")
plt.ylabel("Sales")
plt.grid(True)
plt.show()

print("=" * 80)
```

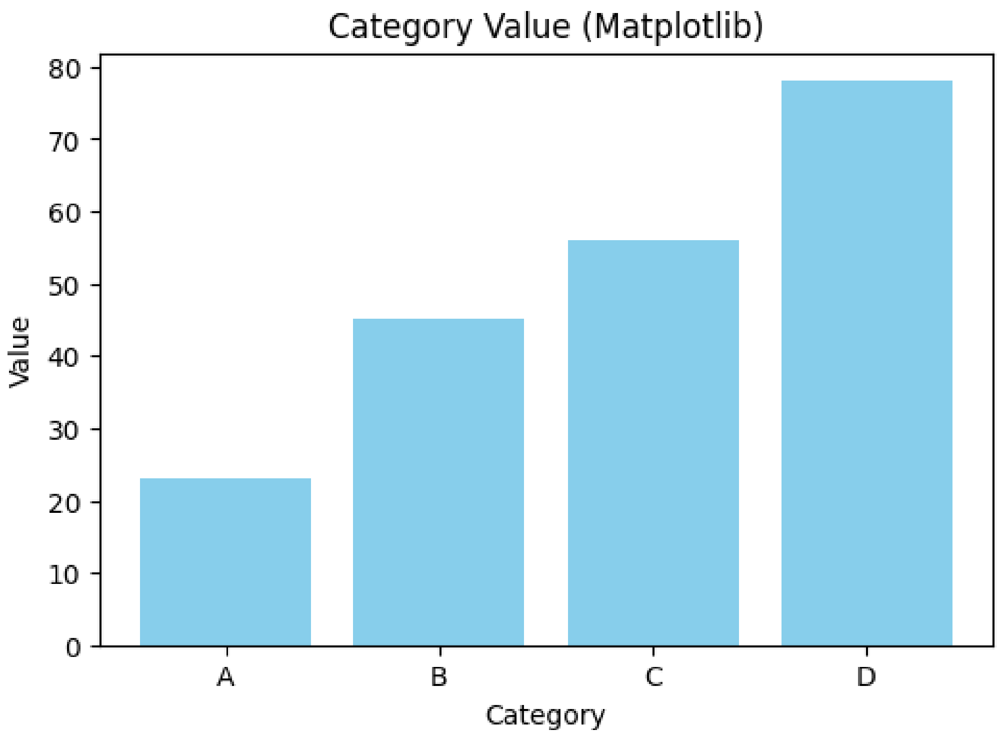




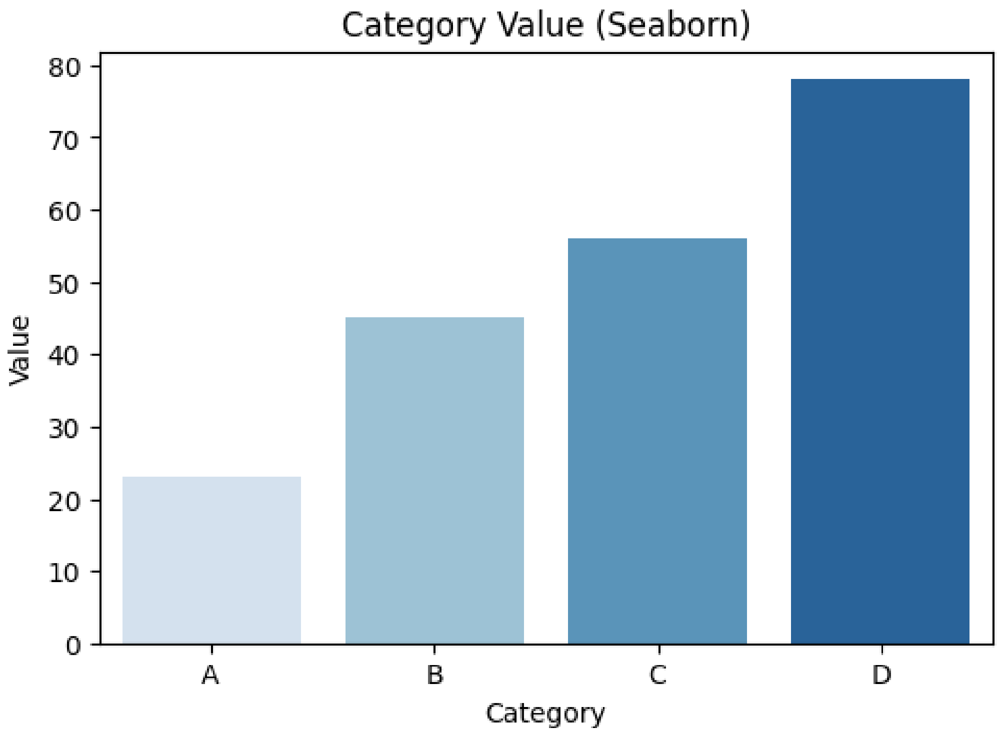
==

[161]:
```
#Bar Plot - Category vs Value
# Sample dataset
category_data = {
    'Category': ['A', 'B', 'C', 'D'],
    'Value': [23, 45, 56, 78]
}
df2 = pd.DataFrame(category_data)

# ------------------ Matplotlib ------------------
plt.figure(figsize=(6, 4))
plt.bar(df2['Category'], df2['Value'], color='skyblue')
plt.title("Category Value (Matplotlib)")
plt.xlabel("Category")
plt.ylabel("Value")
plt.show()

# ------------------ Seaborn ------------------
plt.figure(figsize=(6, 4))
sns.barplot(x='Category', y='Value', data=df2, palette='Blues')
plt.title("Category Value (Seaborn)")
plt.xlabel("Category")
```

```
plt.ylabel("Value")
plt.show()

print("=" * 80)
```




```
/var/folders/mz/ypdsq4nd3mqg289r8kwqv3_40000gn/T/ipykernel_24662/2134621543.py:1
9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Category', y='Value', data=df2, palette='Blues')
```



==

7. Python SQL Database Access

[31]:
```
import sqlite3
# Connecting to a database
conn = sqlite3.connect("example.db")
cursor = conn.cursor()
```

Creating a Table

[32]:
```
cursor.execute('''CREATE TABLE IF NOT EXISTS students (
                    id INTEGER PRIMARY KEY,
                    name TEXT,
                    age INTEGER)''')
conn.commit()
```

Inserting Data

[33]:
```
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Sudeep", 22))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Ravi", 23))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Amit", 21))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Priya", 22))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Neha", 20))
```

```python
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Rahul", 24))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Anjali", 23))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Vikram", 22))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Deepa", 21))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Twinkle",␣
↪25))
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Meena", 20))
conn.commit()
```

Fetching Data

[34]:
```python
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

```
(1, 'Sudeep', 22)
(2, 'Ravi', 23)
(3, 'Amit', 21)
(4, 'Priya', 22)
(5, 'Neha', 20)
(6, 'Rahul', 24)
(7, 'Anjali', 23)
(8, 'Vikram', 22)
(9, 'Deepa', 21)
(10, 'Twinkle', 25)
(11, 'Meena', 20)
```

Deleting Data

[35]:
```python
cursor.execute("DELETE FROM students WHERE name = ?", ("Ramesh",))
cursor.execute("DELETE FROM students WHERE name = ?", ("Neha",))
conn.commit()
```

[36]:
```python
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

```
(1, 'Sudeep', 22)
(2, 'Ravi', 23)
(3, 'Amit', 21)
(4, 'Priya', 22)
(6, 'Rahul', 24)
(7, 'Anjali', 23)
(8, 'Vikram', 22)
(9, 'Deepa', 21)
(10, 'Twinkle', 25)
```

```
(11, 'Meena', 20)
```

Closing Connection

[37]:
```python
conn.close()
```

Unit:2

1. Pandas

Definition: Pandas is an open-source Python library used for data manipulation and analysis. It provides data structures like Series (1D) and DataFrame (2D) that are powerful and flexible for handling structured data.

DataFarme

[122]:
```python
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)

print("=" * 80)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
================================================================================
```

Importing Data from Excel and CSV Files:

[123]:
```python
import pandas as pd

# Reading a CSV file
print("This is the CSV file read by the above code")
df_csv = pd.read_csv('/Users/shubhamkushwaha/Downloads/username.csv')
print(df_csv)


# Reading an Excel file
print("=" * 80)
print("This is the Excel file read by the above code")
df_excel = pd.read_excel("/Users/shubhamkushwaha/Downloads/file_example_XLSX_10.
 ↪xlsx")
```

```python
print(df_excel.head())
print("=" * 80)
```

```
This is the CSV file read by the above code
  Username; Identifier;First name;Last name
0              booker12;9012;Rachel;Booker
1                   grey07;2070;Laura;Grey
2             johnson81;4081;Craig;Johnson
3              jenkins46;9346;Mary;Jenkins
4                 smith79;5079;Jamie;Smith
================================================================================
This is the Excel file read by the above code
   0 First Name  Last Name  Gender        Country  Age        Date    Id
0  1      Dulce      Abril  Female  United States   32  15/10/2017  1562
1  2       Mara  Hashimoto  Female  Great Britain   25  16/08/2016  1582
2  3     Philip       Gent    Male         France   36  21/05/2015  2587
3  4   Kathleen     Hanner  Female  United States   25  15/10/2017  3549
4  5    Nereida    Magwood  Female  United States   58  16/08/2016  2468
================================================================================
```

Powerful Filters:

[124]:
```python
#Age Greater Than 25
filtered_df = df_excel[df_excel['Age'] > 25]
print(filtered_df)
print("=" * 80)
```

```
   0 First Name Last Name  Gender        Country  Age        Date    Id
0  1      Dulce     Abril  Female  United States   32  15/10/2017  1562
2  3     Philip      Gent    Male         France   36  21/05/2015  2587
4  5    Nereida   Magwood  Female  United States   58  16/08/2016  2468
6  7       Etta      Hurn  Female  Great Britain   56  15/10/2017  3598
7  8    Earlean    Melgar  Female  United States   27  16/08/2016  2456
8  9   Vincenza   Weiland  Female  United States   40  21/05/2015  6548
================================================================================
```

[125]:
```python
#Gender is Female
filtered_df = df_excel[df_excel['Gender'] == 'Female']
print(filtered_df)
print("=" * 80)
```

```
   0 First Name  Last Name  Gender        Country  Age        Date    Id
0  1      Dulce      Abril  Female  United States   32  15/10/2017  1562
1  2       Mara  Hashimoto  Female  Great Britain   25  16/08/2016  1582
3  4   Kathleen     Hanner  Female  United States   25  15/10/2017  3549
4  5    Nereida    Magwood  Female  United States   58  16/08/2016  2468
6  7       Etta       Hurn  Female  Great Britain   56  15/10/2017  3598
7  8    Earlean     Melgar  Female  United States   27  16/08/2016  2456
```

```
8  9   Vincenza    Weiland  Female  United States   40  21/05/2015  6548
================================================================================
```

[126]:
```python
#First Name Starts with 'A'
filtered_df = df_excel[df_excel['First Name'].str.startswith('E')]
print(filtered_df)
print("=" * 80)
```

```
   0 First Name Last Name  Gender        Country  Age        Date    Id
6  7       Etta      Hurn  Female  Great Britain   56  15/10/2017  3598
7  8    Earlean    Melgar  Female  United States   27  16/08/2016  2456
================================================================================
```

[127]:
```python
#Last Name Contains 'sh'
filtered_df = df_excel[df_excel['Last Name'].str.contains('sh', case=False)]
print(filtered_df)
print("=" * 80)
```

```
   0 First Name  Last Name  Gender        Country  Age        Date    Id
1  2       Mara  Hashimoto  Female  Great Britain   25  16/08/2016  1582
================================================================================
```

[128]:
```python
#Date After January 1, 2023
df_excel['Date'] = pd.to_datetime(df_excel['Date'])  # Ensure 'date' is datetime
filtered_df = df_excel[df_excel['Date'] > '2023-01-01']
print(filtered_df)
print("=" * 80)
```

```
Empty DataFrame
Columns: [0, First Name, Last Name, Gender, Country, Age, Date, Id]
Index: []
================================================================================

/var/folders/mz/ypdsq4nd3mqg289r8kwqv3_40000gn/T/ipykernel_24662/3133800990.py:2
: UserWarning: Parsing dates in %d/%m/%Y format when dayfirst=False (the
default) was specified. Pass `dayfirst=True` or specify a format to silence this
warning.
  df_excel['Date'] = pd.to_datetime(df_excel['Date'])  # Ensure 'date' is
datetime
```

[129]:
```python
#Males Under Age 25
filtered_df = df_excel[(df_excel['Gender'] == 'Male') & (df_excel['Age'] < 25)]
print(filtered_df)
print("=" * 80)
```

```
   0 First Name Last Name Gender        Country  Age       Date    Id
5  6     Gaston     Brumm   Male  United States   24 2015-05-21  2554
================================================================================
```

[130]:
```
#Rows with Missing Last Name
filtered_df = df_excel[df_excel['Last Name'].isnull()]
print(filtered_df)
print("=" * 80)
```

```
Empty DataFrame
Columns: [0, First Name, Last Name, Gender, Country, Age, Date, Id]
Index: []
================================================================================
```

Indexes:

[131]:
```
# Setting 'ID' as the index
df_indexed = df_excel.set_index('Id')
print(df_indexed)
print("=" * 80)
```

```
      0 First Name  Last Name  Gender        Country  Age       Date
Id
1562  1      Dulce      Abril  Female  United States   32 2017-10-15
1582  2       Mara  Hashimoto  Female  Great Britain   25 2016-08-16
2587  3     Philip       Gent    Male         France   36 2015-05-21
3549  4   Kathleen     Hanner  Female  United States   25 2017-10-15
2468  5    Nereida    Magwood  Female  United States   58 2016-08-16
2554  6     Gaston      Brumm    Male  United States   24 2015-05-21
3598  7       Etta       Hurn  Female  Great Britain   56 2017-10-15
2456  8    Earlean     Melgar  Female  United States   27 2016-08-16
6548  9   Vincenza    Weiland  Female  United States   40 2015-05-21
================================================================================
```

2. NumPy

Definition: NumPy (Numerical Python) is a core library for numerical computations. It provides support for large multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on them.

Selecting Data with loc and iloc (via Pandas):

[132]:
```
#Select Multiple Rows and Specific Columns using loc (Label-based)
# Selecting rows with index labels 1 to 3 and columns 'firstname' and 'age'
print(df_excel.loc[1:3, ['First Name', 'Age']])
print("=" * 80)
```

```
  First Name  Age
1       Mara   25
2     Philip   36
3   Kathleen   25
================================================================================
```

[133]:
```python
#Select Last 3 Rows and All Columns using iloc (Index-based)
# Selecting the last 3 rows (assuming you don't know total rows)
print(df_excel.iloc[-3:])
print("=" * 80)
```

```
   0 First Name Last Name  Gender        Country  Age       Date    Id
6  7       Etta      Hurn  Female  Great Britain   56 2017-10-15  3598
7  8    Earlean    Melgar  Female  United States   27 2016-08-16  2456
8  9   Vincenza   Weiland  Female  United States   40 2015-05-21  6548
================================================================================
```

[134]:
```python
# Using loc (label-based)
print(df_excel.loc[1])  # First row using label

# Using iloc (index-based)
print(df_excel.iloc[0])  # First row using index

print("=" * 80)
```

```
0                               2
First Name                   Mara
Last Name               Hashimoto
Gender                     Female
Country             Great Britain
Age                            25
Date          2016-08-16 00:00:00
Id                           1582
Name: 1, dtype: object
0                               1
First Name                  Dulce
Last Name                   Abril
Gender                     Female
Country             United States
Age                            32
Date          2017-10-15 00:00:00
Id                           1562
Name: 0, dtype: object
================================================================================
```

Using NumPy for Speed:

[135]:
```python
import numpy as np
import time

# Using list
list_data = list(range(1000000))
start = time.time()
sum_list = sum(list_data)
```

```
print("List sum:", sum_list, "Time:", time.time() - start)

# Using numpy array
array_data = np.array(list_data)
start = time.time()
sum_array = np.sum(array_data)
print("Array sum:", sum_array, "Time:", time.time() - start)

print("=" * 80)
```

```
List sum: 499999500000 Time: 0.007378339767456055
Array sum: 499999500000 Time: 0.0003471374511171875
================================================================================
```

Trade-offs Between Arrays and Lists: 1. NumPy Arrays: Faster, require less memory, better for numerical operations. 2. Python Lists: More flexible, can store multiple data types, but slower for numeric tasks.

Common NumPy Array Functions:

[136]:
```
arr = np.array([10, 20, 30, 40, 50])

print("Mean:", np.mean(arr))
print("Standard Deviation:", np.std(arr))
print("Maximum:", np.max(arr))
print("Minimum:", np.min(arr))

print("=" * 80)
```

```
Mean: 30.0
Standard Deviation: 14.142135623730951
Maximum: 50
Minimum: 10
================================================================================
```

[137]:
```
#Sum and Product of Array Elements
arr = np.array([10, 20, 30, 40, 50])

print("Sum:", np.sum(arr))
print("Product:", np.prod(arr))
print("=" * 80)
```

```
Sum: 150
Product: 12000000
================================================================================
```

[138]:
```
#Median and Percentile
print("Median:", np.median(arr))
print("25th Percentile:", np.percentile(arr, 25))
```

```
print("75th Percentile:", np.percentile(arr, 75))
print("=" * 80)
```

```
Median: 30.0
25th Percentile: 20.0
75th Percentile: 40.0
================================================================================
```

[139]:
```
#Sorting and Reversing
print("Sorted Array:", np.sort(arr))
print("Reversed Array:", arr[::-1])
print("=" * 80)
```

```
Sorted Array: [10 20 30 40 50]
Reversed Array: [50 40 30 20 10]
================================================================================
```

3. Data Cleansing and Normalization

Definition: Data cleansing refers to the process of detecting and correcting (or removing) corrupt or inaccurate records from a dataset. Normalization involves scaling data into a standard range (e.g., 0 to 1) for better performance in machine learning.

Handling Missing Values:

[140]:
```
# Checking for missing values
print(df_excel.isnull().sum())

# Filling missing values with mean
df_excel['Age'].fillna(df_excel['Age'].mean(), inplace=True)

print("=" * 80)
```

```
0             0
First Name    0
Last Name     0
Gender        0
Country       0
Age           0
Date          0
Id            0
dtype: int64
================================================================================
```

```
/var/folders/mz/ypdsq4nd3mqg289r8kwqv3_40000gn/T/ipykernel_24662/490198935.py:5:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df_excel['Age'].fillna(df_excel['Age'].mean(), inplace=True)
```

Removing Duplicates:

[29]:
```python
df_csv = df_csv.drop_duplicates()
```

Normalization using scikit-learn:

[141]:
```python
from sklearn.preprocessing import MinMaxScaler

# Assuming df has numeric values
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(df_excel[['Age']])
print(normalized_data)

print("=" * 80)
```

```
[[0.23529412]
 [0.02941176]
 [0.35294118]
 [0.02941176]
 [1.        ]
 [0.        ]
 [0.94117647]
 [0.08823529]
 [0.47058824]]
================================================================================
```

4. Data Visualization

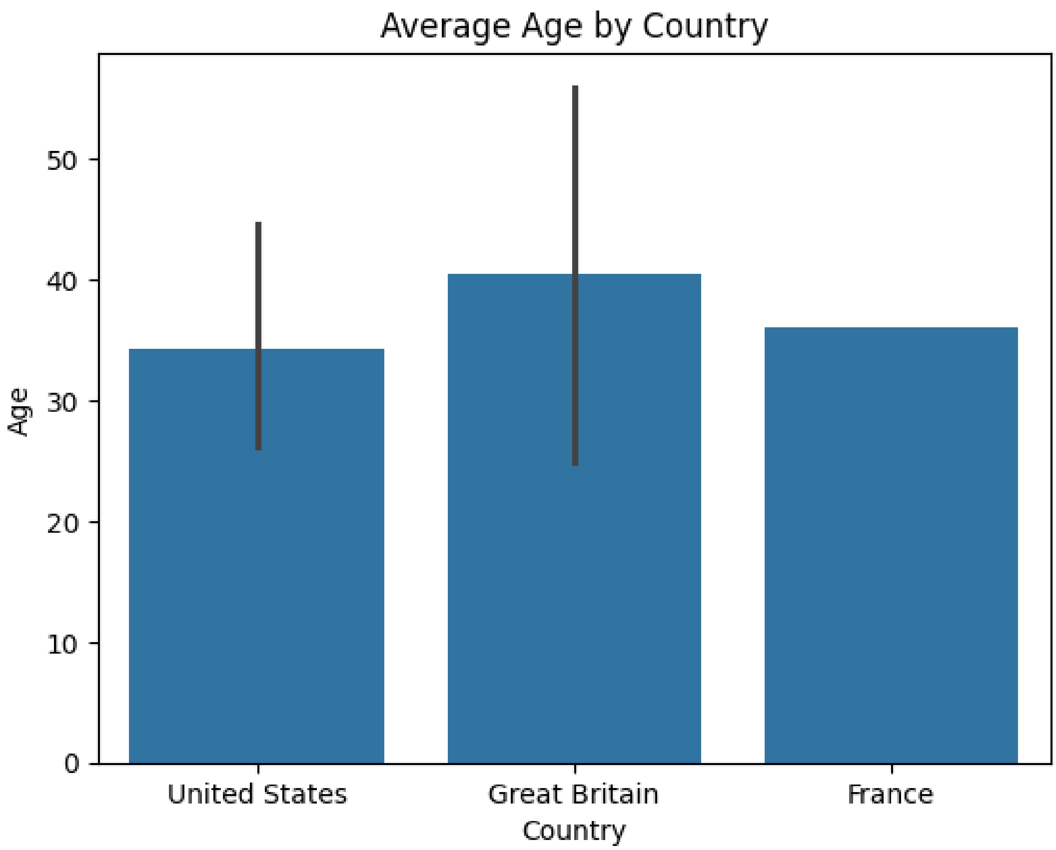
Definition: Data visualization is the graphical representation of information and data. It helps in understanding trends, outliers, and patterns in data.

Popular Libraries: 1. Matplotlib – basic plotting 2. Seaborn – statistical plots built on top of matplotlib

Bar Chart:

[142]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Bar chart of average Age by Country
sns.barplot(x='Country', y='Age', data=df_excel)
plt.title("Average Age by Country")
plt.show()
```

```
print("=" * 80)
```



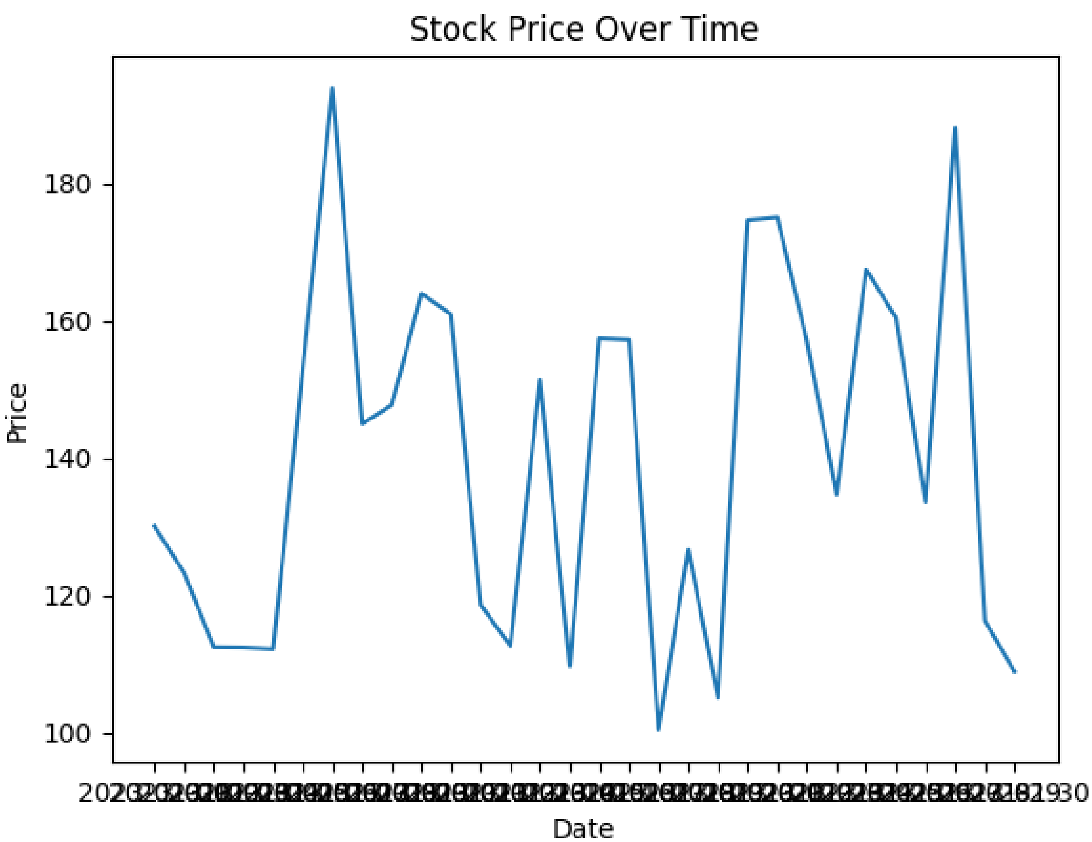

```
================================================================================
```

Line Plot:

[4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read the CSV file back into a new DataFrame
df_new = pd.read_csv("my_stock_data.csv")
print(df_new.head())
print("="*80)

# Plotting stock prices over time using the new DataFrame
plt.plot(df_new['Date'], df_new['Close'])
plt.title("Stock Price Over Time")
plt.xlabel("Date")
plt.ylabel("Price")
```

```
plt.show()
print("=" * 80)
```

```
         Date       Close
0  2023-01-01  130.049681
1  2023-01-02  123.297252
2  2023-01-03  112.454919
3  2023-01-04  112.410107
4  2023-01-05  112.187185
================================================================================
```

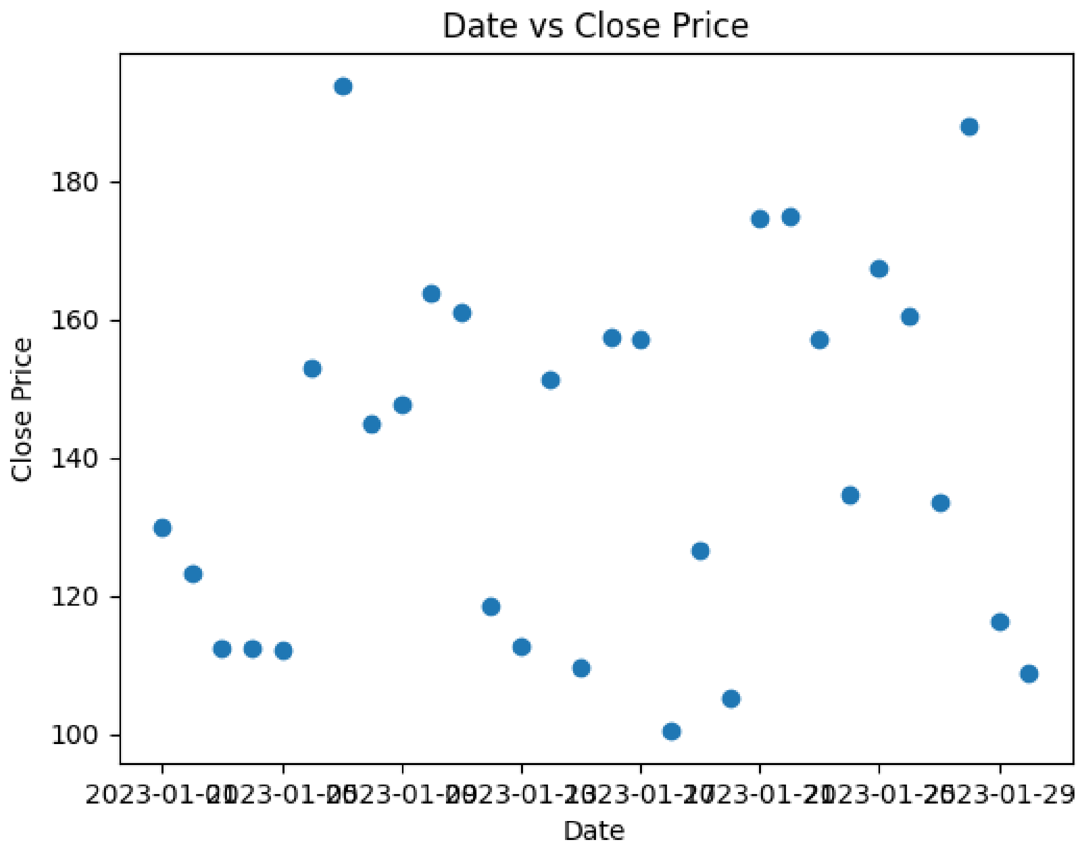



```
================================================================================
```

Scatter Plot:

[6]:
```
# Scatter plot of Date vs Close price
plt.scatter(df['Date'], df['Close'])
plt.title("Date vs Close Price")
plt.xlabel("Date")
plt.ylabel("Close Price")
plt.show()
```

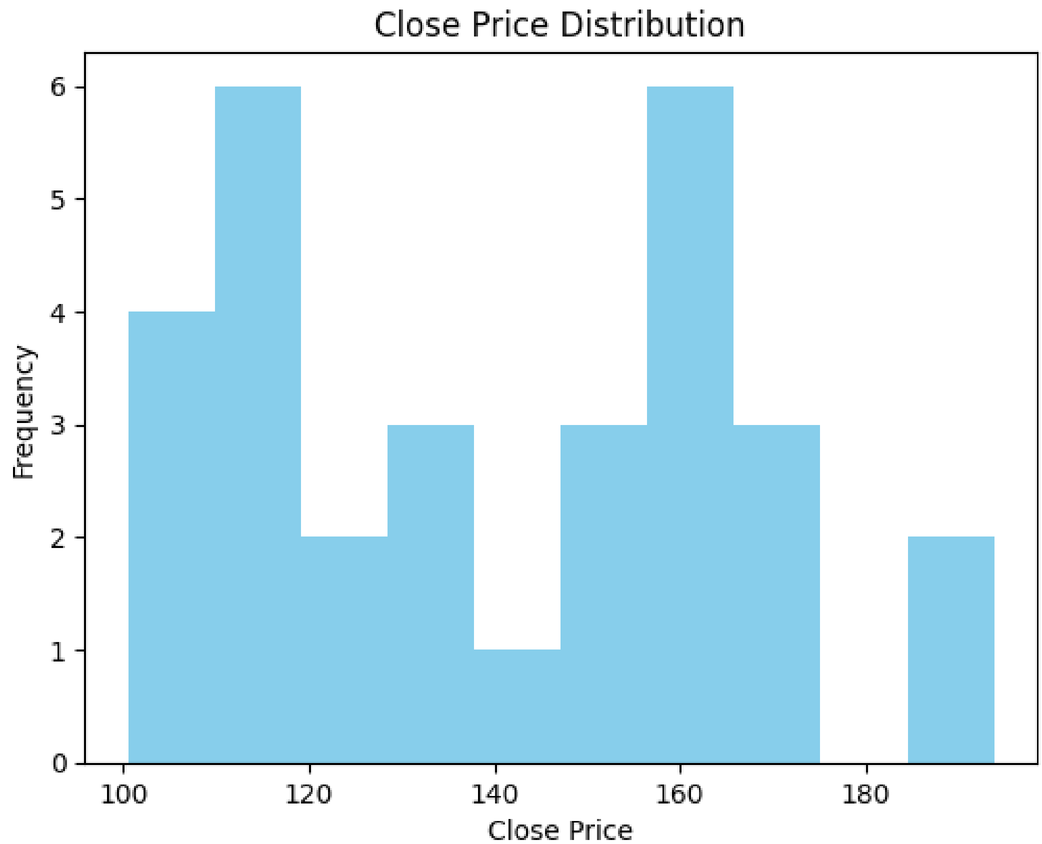
```
print("=" * 80)
```



==

Histogram:

[8]:
```
# Distribution of closing prices
plt.hist(df['Close'], bins=10, color='skyblue')
plt.title("Close Price Distribution")
plt.xlabel("Close Price")
plt.ylabel("Frequency")
plt.show()

print("=" * 80)
```



==

Unit:-3

1. Introduction to Machine Learning

Definition:
Machine Learning (ML) is a subset of Artificial Intelligence (AI) that enables computers to learn patterns from data and make predictions or decisions without being explicitly programmed for each task.

2. Types of Machine Learning

A. Supervised Learning 1. The model is trained on labeled data (i.e., data with input-output pairs). 2. Used for tasks like classification and regression.

Examples: Spam detection, price prediction, disease diagnosis.

B. Unsupervised Learning 1. The model is trained on unlabeled data and discovers hidden patterns or groupings. 2. Used for clustering and dimensionality reduction.

Examples: Customer segmentation, market basket analysis.

===

3. Python Libraries for Machine Learning

Popular Libraries: **1. scikit-learn** – core ML algorithms (classification, regression, clustering)

2. pandas – data manipulation

3. numpy – numerical operations

4. matplotlib, seaborn – data visualization

==

4. Regression Models

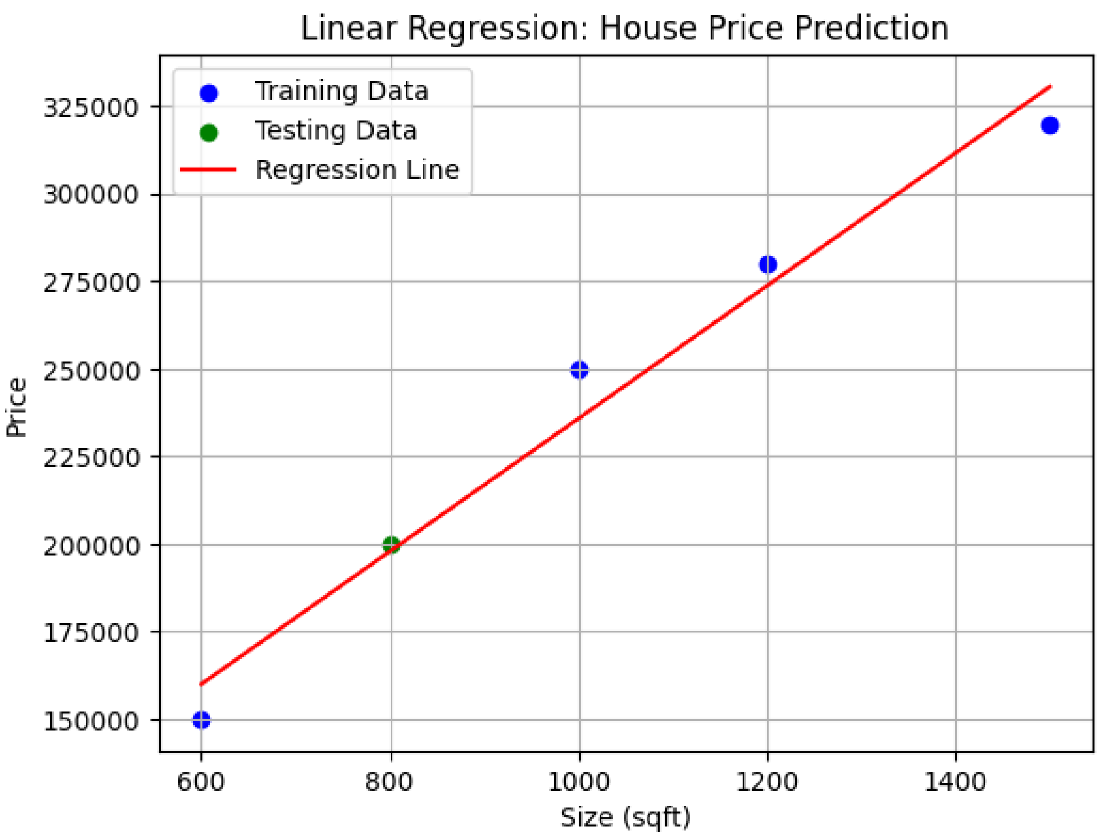
A. Linear Regression

Definition: Linear Regression predicts a continuous value based on the linear relationship between input variables (X) and the output (Y).

[9]:
```python
#Linear Regression: Predicting House Price Based on Size with Plot
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Dataset
data = {'Size_sqft': [600, 800, 1000, 1200, 1500],
        'Price': [150000, 200000, 250000, 280000, 320000]}
df = pd.DataFrame(data)

X = df[['Size_sqft']]
y = df['Price']

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Model
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
pred = model.predict(X_test)
print("Predicted Prices:", pred)
print("MSE:", mean_squared_error(y_test, pred))
print("=" * 80)

# ------------------ Plotting ------------------
# Plot training data
plt.scatter(X_train, y_train, color='blue', label='Training Data')
```

```python
# Plot testing data
plt.scatter(X_test, y_test, color='green', label='Testing Data')

# Plot regression line
line = model.predict(X)  # Predict on the full X to draw the line
plt.plot(X, line, color='red', label='Regression Line')

# Labels and legend
plt.xlabel("Size (sqft)")
plt.ylabel("Price")
plt.title("Linear Regression: House Price Prediction")
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicted Prices: [197894.73684211]
MSE: 4432132.963988964
================================================================================
```

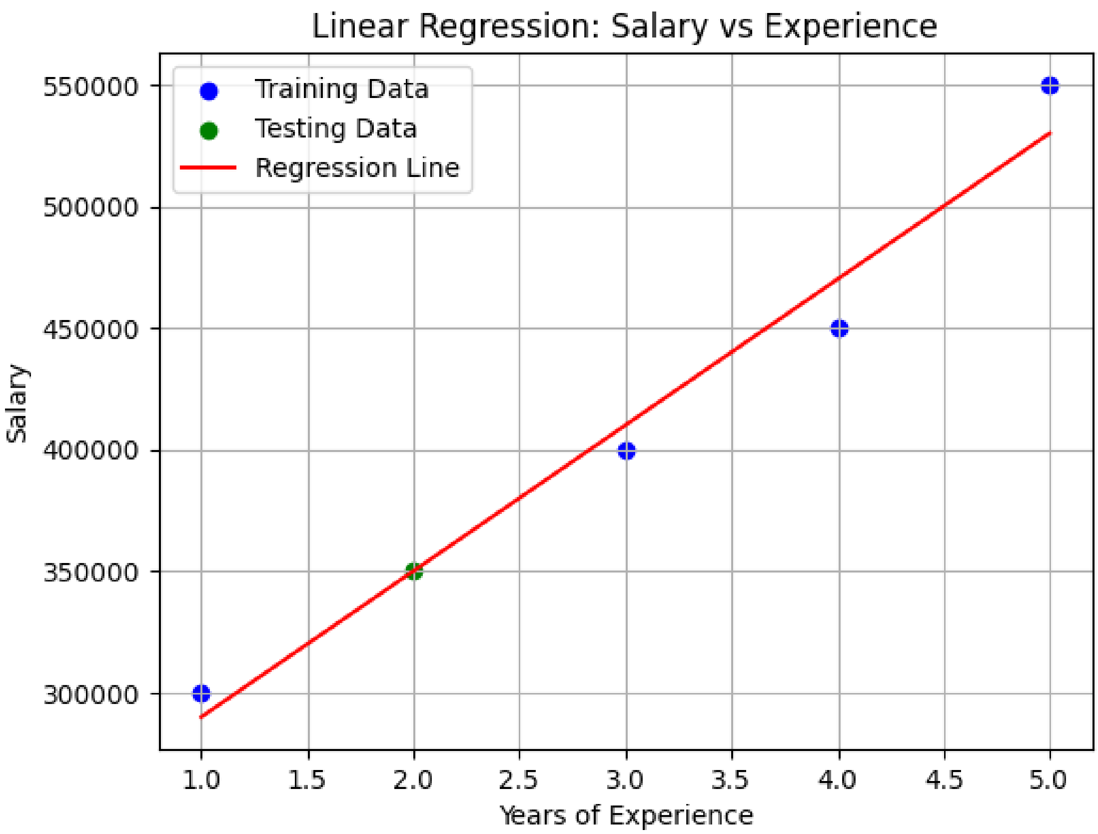


[12]:

```python
#Python implementation
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Sample dataset
data = {'Experience': [1, 2, 3, 4, 5],
        'Salary': [300000, 350000, 400000, 450050, 550000]}
df = pd.DataFrame(data)

# Splitting data
X = df[['Experience']]  # Input feature
y = df['Salary']        # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Model training
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
predictions = model.predict(X_test)
print("Predicted Salaries:", predictions)

# Evaluation
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
print("=" * 80)

# ------------------ Plotting ------------------
# Plot training data
plt.scatter(X_train, y_train, color='blue', label='Training Data')

# Plot testing data
plt.scatter(X_test, y_test, color='green', label='Testing Data')

# Regression line
line = model.predict(X)  # Use all X values to draw the full line
plt.plot(X, line, color='red', label='Regression Line')

# Labels and legend
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Linear Regression: Salary vs Experience")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

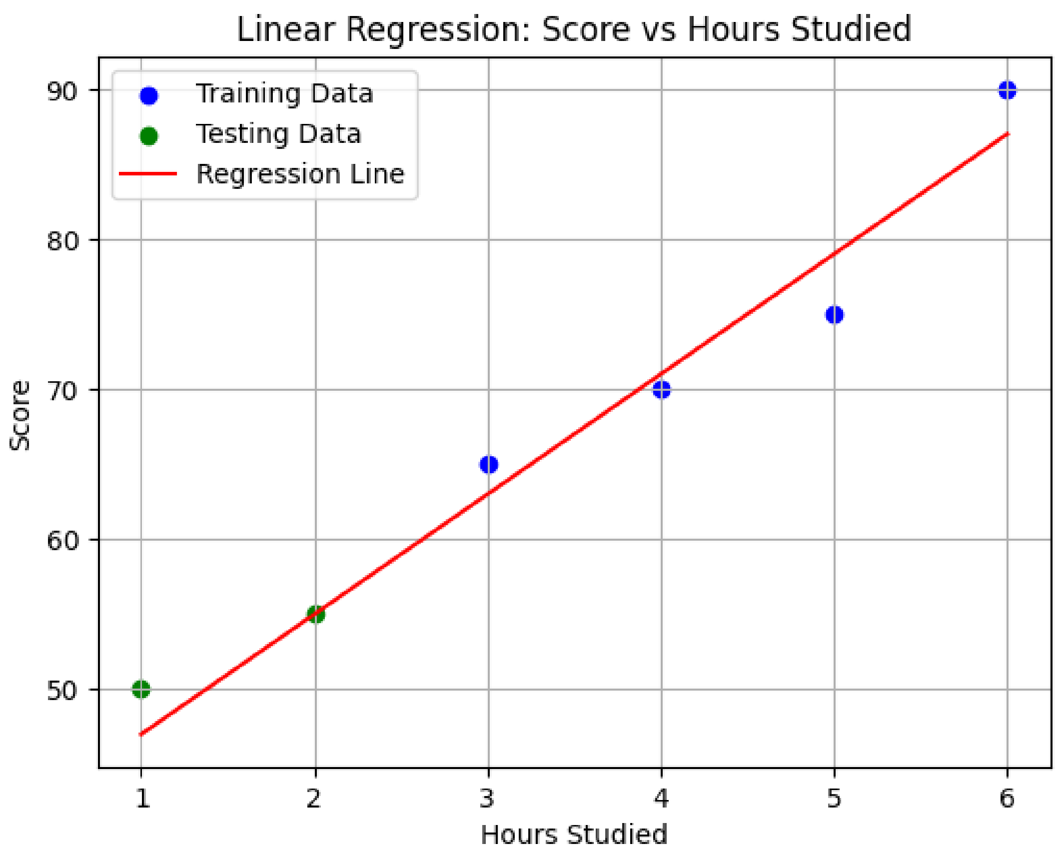
```
Predicted Salaries: [350007.14285714]
Mean Squared Error: 51.020408163027724
================================================================================
```



[13]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Dataset
data = {'Hours_Studied': [1, 2, 3, 4, 5,6],
        'Score': [50, 55, 65, 70, 75,90]}
df = pd.DataFrame(data)

X = df[['Hours_Studied']]
```

```
y = df['Score']

# Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Model
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
pred = model.predict(X_test)
print("Predicted Scores:", pred)
print("MSE:", mean_squared_error(y_test, pred))
print("=" * 80)

# ------------------ Plotting ------------------
# Training points
plt.scatter(X_train, y_train, color='blue', label='Training Data')

# Test points
plt.scatter(X_test, y_test, color='green', label='Testing Data')

# Regression line
line = model.predict(X)
plt.plot(X, line, color='red', label='Regression Line')

# Labels and formatting
plt.xlabel("Hours Studied")
plt.ylabel("Score")
plt.title("Linear Regression: Score vs Hours Studied")
plt.legend()
plt.grid(True)
plt.show()
```

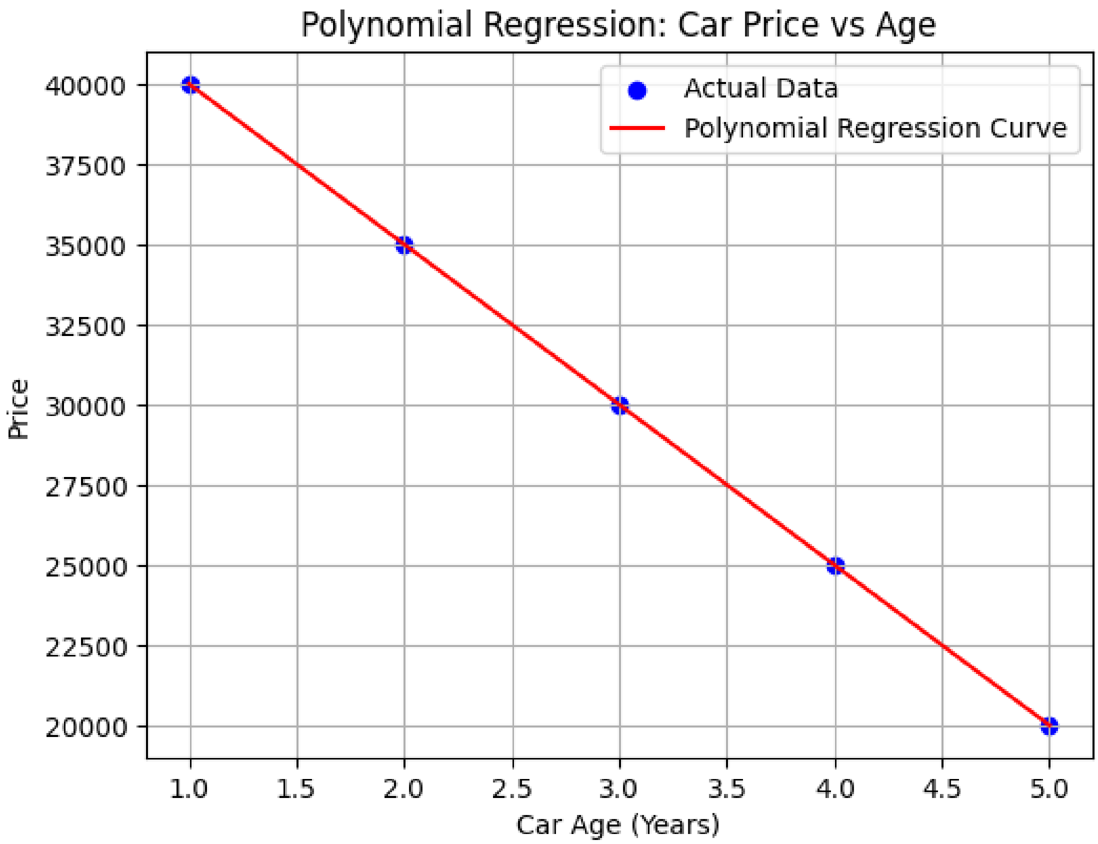
```
Predicted Scores: [47. 55.]
MSE: 4.499999999999957
================================================================================
```



[14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Dataset
data = {'Age': [1, 2, 3, 4, 5],
        'Price': [40000, 35000, 30000, 25000, 20000]}
df = pd.DataFrame(data)

X = df[['Age']]
y = df['Price']

# Polynomial transformation
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
```

```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.2,␣
↪random_state=42)

# Model training
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
pred = model.predict(X_test)
print("Predicted Car Prices:", pred)
print("MSE:", mean_squared_error(y_test, pred))
print("=" * 80)

# ------------------ Plotting ------------------
# Generate smoother curve for plotting
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_range_poly = poly.transform(X_range)
y_range_pred = model.predict(X_range_poly)

# Plot actual data
plt.scatter(X, y, color='blue', label='Actual Data')

# Plot regression curve
plt.plot(X_range, y_range_pred, color='red', label='Polynomial Regression␣
↪Curve')

# Labels and formatting
plt.xlabel("Car Age (Years)")
plt.ylabel("Price")
plt.title("Polynomial Regression: Car Price vs Age")
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicted Car Prices: [35000.]
MSE: 2.117582368135751e-22
================================================================================

/home/codespace/.local/lib/python3.12/site-
packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
```

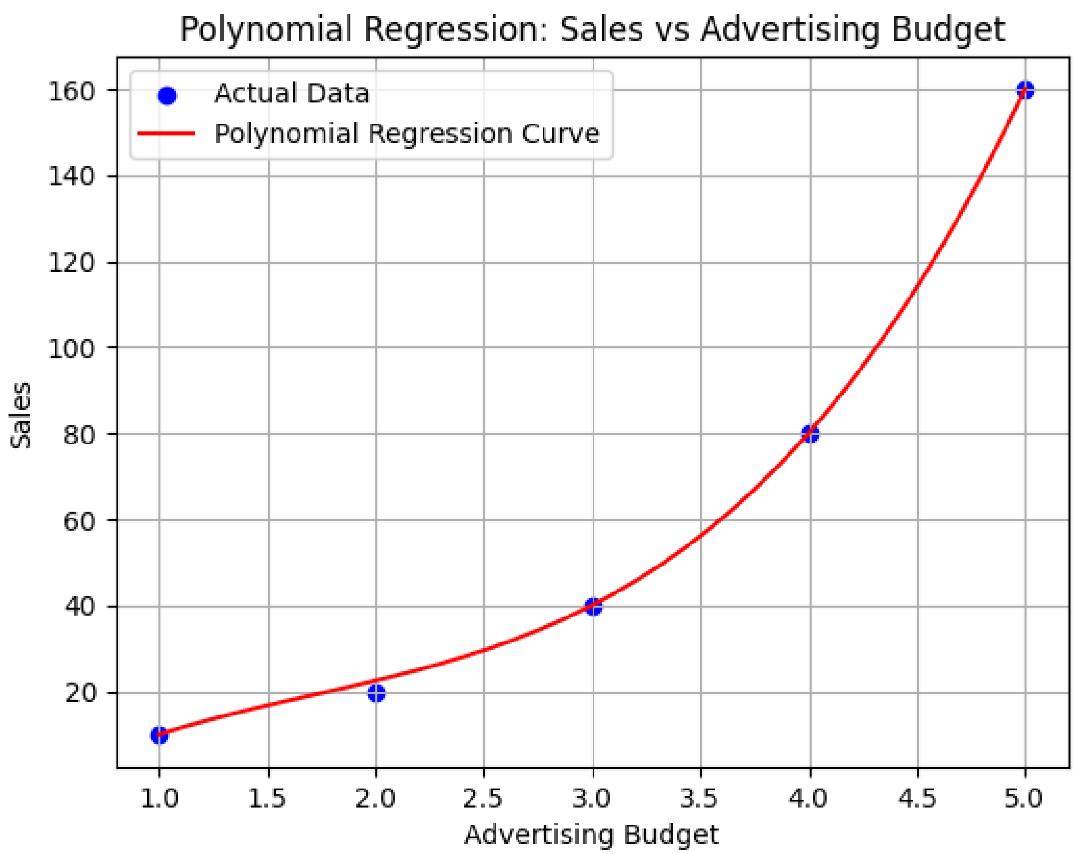


[15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Dataset
data = {'Ad_Budget': [1, 2, 3, 4, 5],
        'Sales': [10, 20, 40, 80, 160]}
df = pd.DataFrame(data)

X = df[['Ad_Budget']]
y = df['Sales']

# Polynomial transformation (degree 3)
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.2,␣
↪random_state=42)

# Model training
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
pred = model.predict(X_test)
print("Predicted Sales:", pred)
print("MSE:", mean_squared_error(y_test, pred))
print("=" * 80)

# ------------------ Plotting ------------------
# Generate smooth curve
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_range_poly = poly.transform(X_range)
y_range_pred = model.predict(X_range_poly)

# Plot actual data
plt.scatter(X, y, color='blue', label='Actual Data')

# Plot regression curve
plt.plot(X_range, y_range_pred, color='red', label='Polynomial Regression␣
↪Curve')

# Labels and formatting
plt.xlabel("Advertising Budget")
plt.ylabel("Sales")
plt.title("Polynomial Regression: Sales vs Advertising Budget")
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicted Sales: [22.5]
MSE: 6.249999999999254
================================================================================
```

```
/home/codespace/.local/lib/python3.12/site-
packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
```

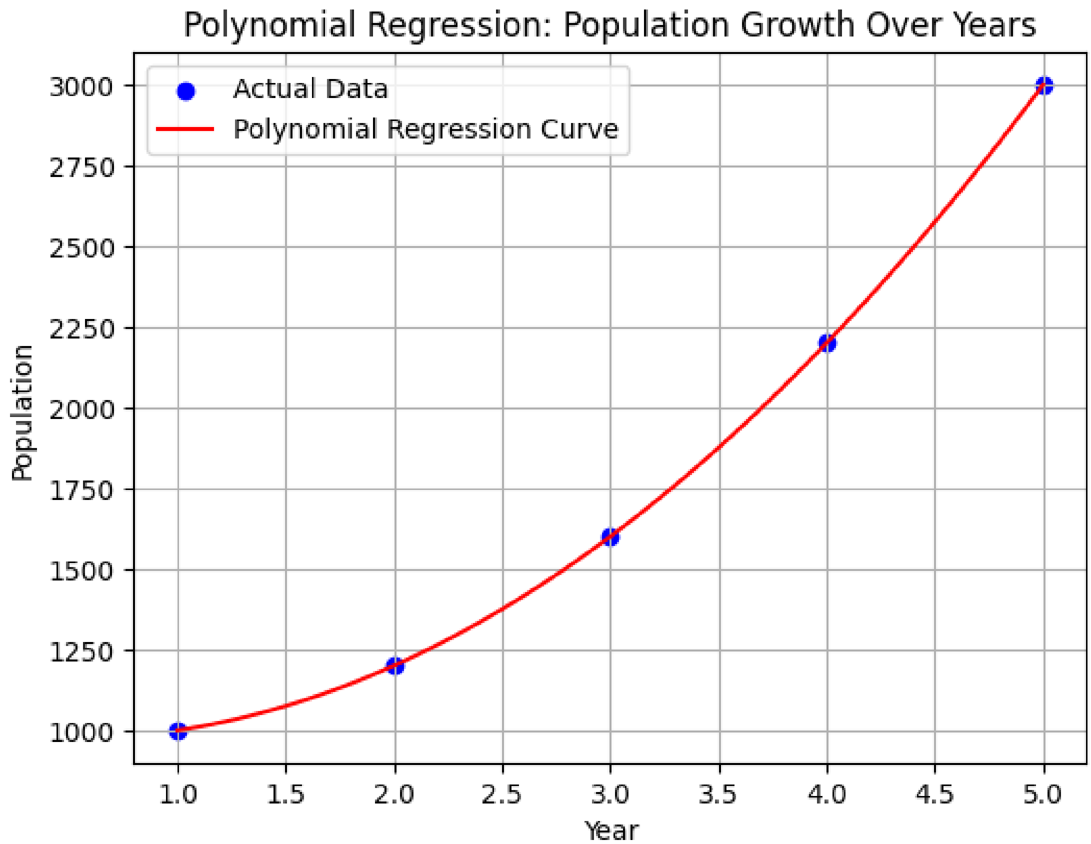


[16]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Dataset
data = {'Year': [1, 2, 3, 4, 5],
        'Population': [1000, 1200, 1600, 2200, 3000]}
df = pd.DataFrame(data)

X = df[['Year']]
y = df['Population']

# Polynomial transformation (degree 2)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.2,␣
↪random_state=42)

# Model training
model = LinearRegression()
model.fit(X_train, y_train)

# Prediction
pred = model.predict(X_test)
print("Predicted Population:", pred)
print("MSE:", mean_squared_error(y_test, pred))
print("=" * 80)

# ------------------ Plotting ------------------
# Generate smoother curve for plotting
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_range_poly = poly.transform(X_range)
y_range_pred = model.predict(X_range_poly)

# Plot actual data
plt.scatter(X, y, color='blue', label='Actual Data')

# Plot regression curve
plt.plot(X_range, y_range_pred, color='red', label='Polynomial Regression␣
↪Curve')

# Labels and formatting
plt.xlabel("Year")
plt.ylabel("Population")
plt.title("Polynomial Regression: Population Growth Over Years")
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicted Population: [1200.]
MSE: 0.0
================================================================================

/home/codespace/.local/lib/python3.12/site-
packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
```



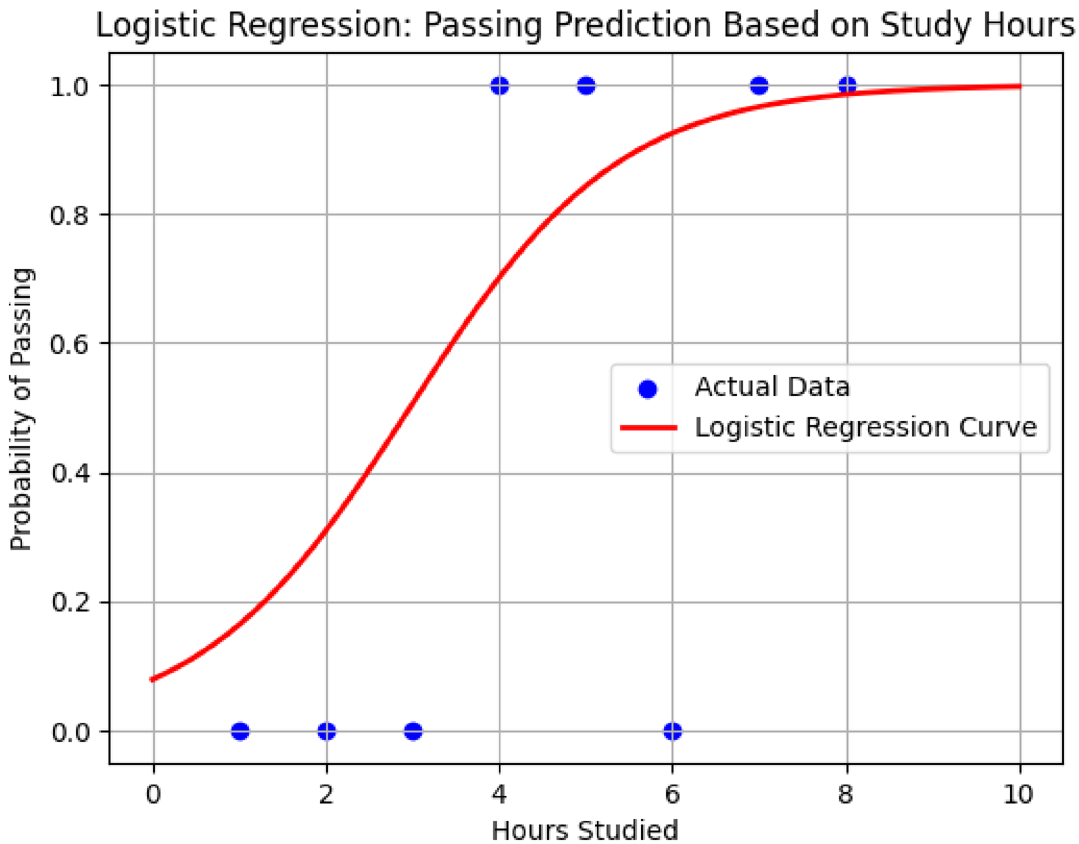
B. Logistic Regression

Definition: Logistic Regression is used for classification problems. It predicts the probability of a binary outcome (0 or 1).

[17]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Dataset
data = {'Hours_Studied': [1, 2, 3, 4, 5, 6, 7, 8],
        'Passed': [0, 0, 0, 1, 1, 0, 1, 1]}
df = pd.DataFrame(data)

X = df[['Hours_Studied']]
y = df['Passed']

# Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Model training
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction & accuracy
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("=" * 80)

# ------------------ Plotting ------------------
# Plotting the logistic regression curve
x_range = np.linspace(0, 10, 200).reshape(-1, 1)
y_prob = model.predict_proba(x_range)[:, 1]

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(x_range, y_prob, color='red', linewidth=2, label='Logistic Regression␣
 ↪Curve')

# Formatting
plt.xlabel("Hours Studied")
plt.ylabel("Probability of Passing")
plt.title("Logistic Regression: Passing Prediction Based on Study Hours")
plt.legend()
plt.grid(True)
plt.show()
```

```
Accuracy: 0.6666666666666666
================================================================================

/home/codespace/.local/lib/python3.12/site-
packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid
feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

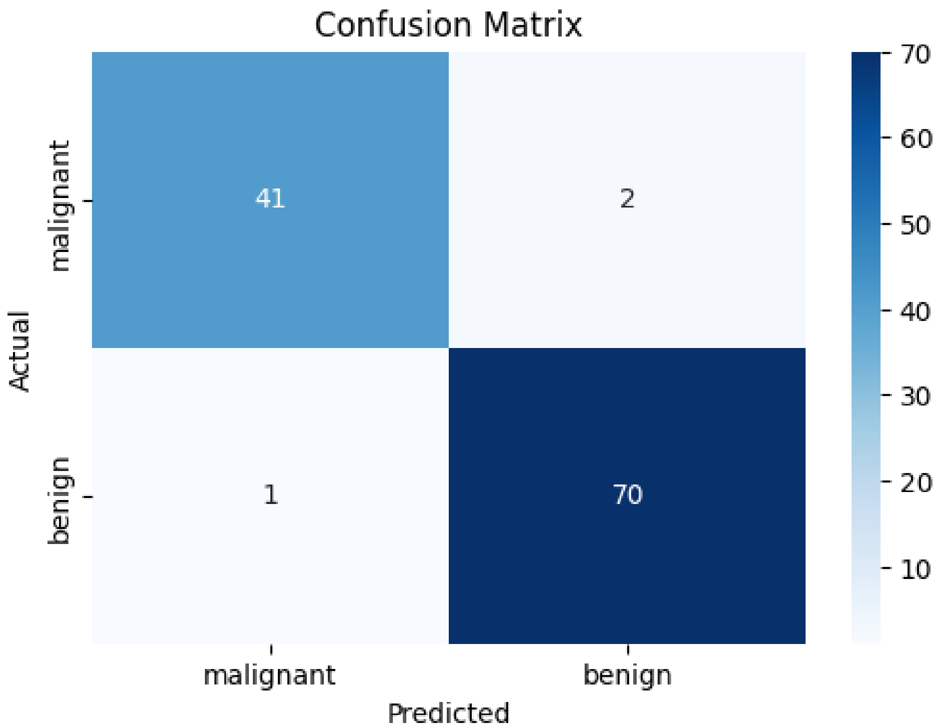


[18]:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,␣
    ↪classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```python
# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

# Model training
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("=" * 80)
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=data.
 ↪target_names, yticklabels=data.target_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Accuracy: 0.9736842105263158
================================================================================
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

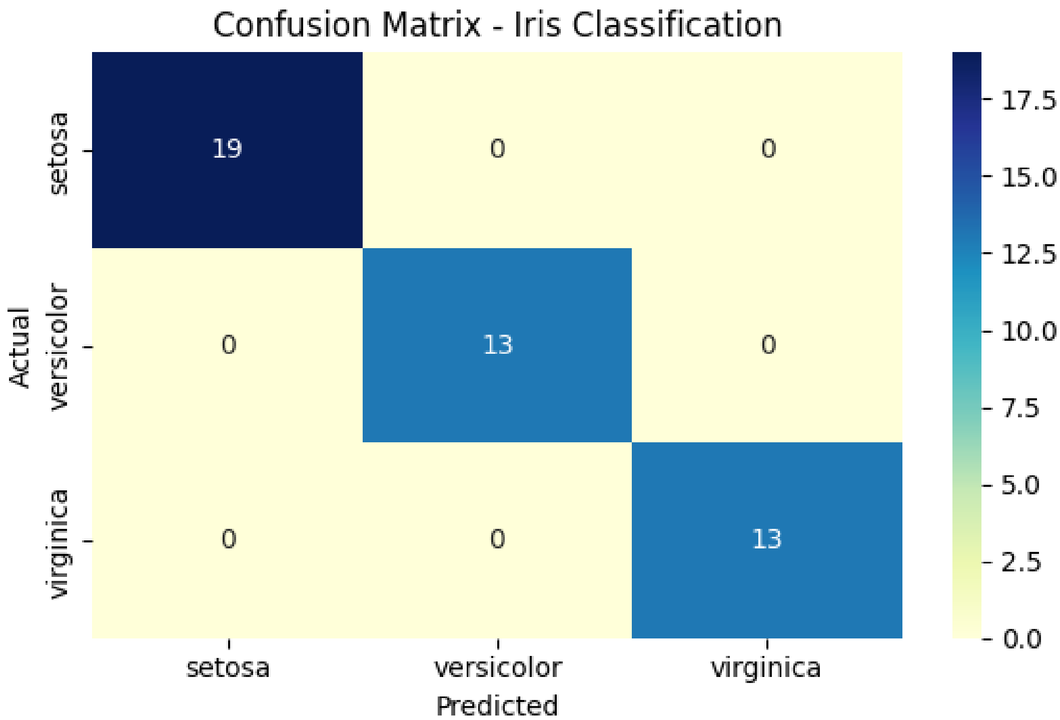


[19]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
```

```python
# Predict
y_pred = model.predict(X_test)

# Accuracy
print("Accuracy (Multi-class):", accuracy_score(y_test, y_pred))
print("=" * 80)

# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred,
    target_names=iris.target_names))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu', xticklabels=iris.
    target_names, yticklabels=iris.target_names)
plt.title("Confusion Matrix - Iris Classification")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

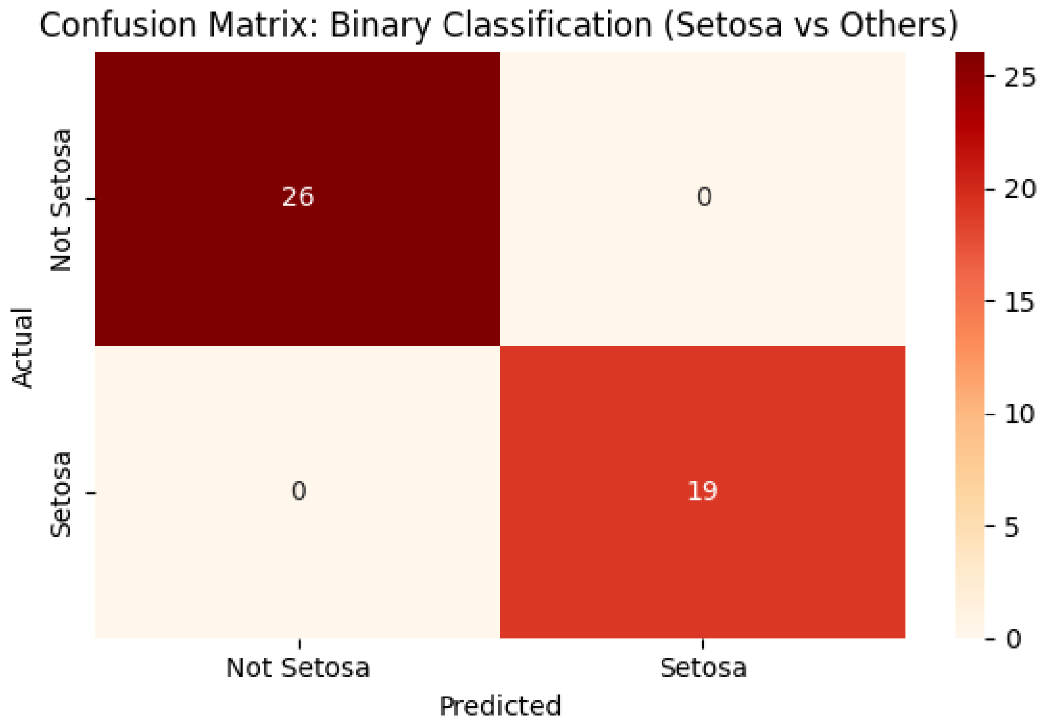
```
Accuracy (Multi-class): 1.0
================================================================================
Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```



[20]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report

# Load dataset
iris = load_iris()
X = iris.data
y = (iris.target == 0).astype(int)  # Binary classification: Setosa = 1, others␣
↪= 0

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Train model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```python
# Predict
y_pred = model.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))
print("=" * 80)

# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='OrRd', xticklabels=['Not Setosa',␣
↪'Setosa'], yticklabels=['Not Setosa', 'Setosa'])
plt.title("Confusion Matrix: Binary Classification (Setosa vs Others)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
Accuracy: 1.0
================================================================================
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       1.00      1.00      1.00        19

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```



5. Overfitting and Regularization

Overfitting: 1. Occurs when the model learns too much from training data (including noise). 2. t performs well on training data but poorly on new (test) data.

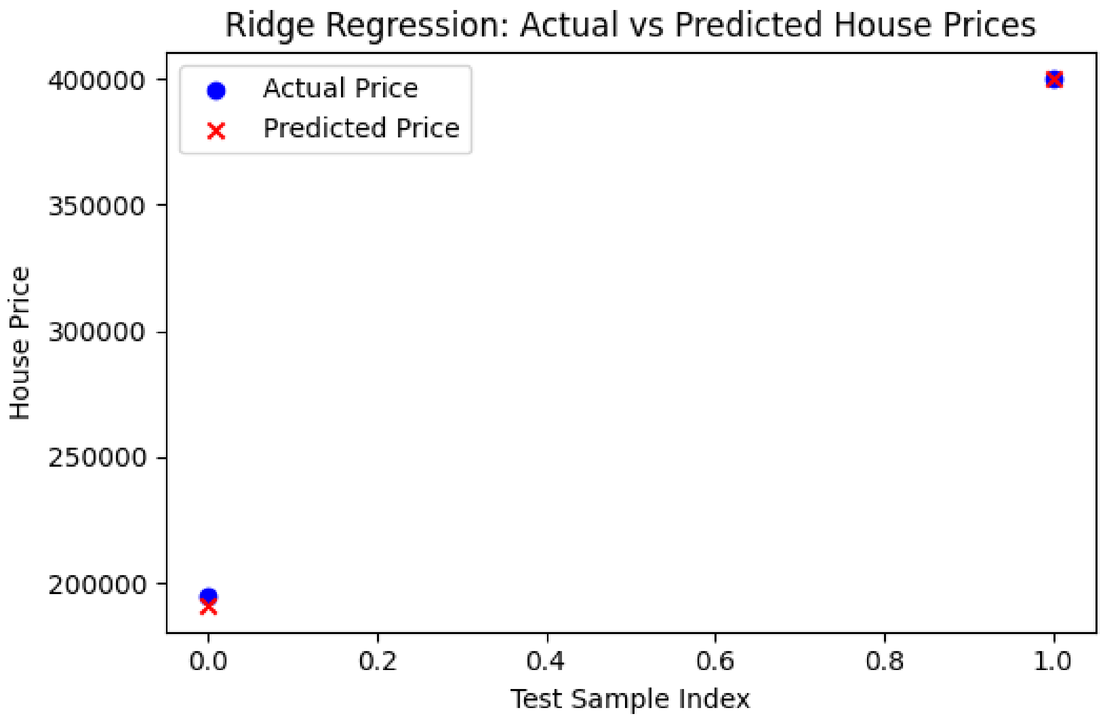
Regularization: 1. A technique to prevent overfitting by penalizing complex models. 2. Helps simplify the model to generalize better on unseen data.

Types of Regularization: 1. L1 Regularization (Lasso): Adds absolute value of coefficients. 2. L2 Regularization (Ridge): Adds square of coefficients.

[22]:
```
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Sample dataset
data = {
    'Size': [850, 900, 1200, 1500, 1750, 2000, 2200, 2500],
    'Bedrooms': [2, 2, 3, 3, 3, 4, 4, 5],
    'Price': [185000, 195000, 240000, 310000, 355000, 400000, 430000, 500000]
}
df = pd.DataFrame(data)
```

```
X = df[['Size', 'Bedrooms']]
y = df['Price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Ridge Regression model
model = Ridge(alpha=0.5)
model.fit(X_train, y_train)

# Prediction
predictions = model.predict(X_test)

# Evaluation
print("Ridge Predictions:", predictions)
print("MSE:", mean_squared_error(y_test, predictions))
print("=" * 80)

# Plot actual vs predicted prices
plt.figure(figsize=(6, 4))
plt.scatter(range(len(y_test)), y_test, color='blue', label='Actual Price')
plt.scatter(range(len(predictions)), predictions, color='red', label='Predicted␣
↪Price', marker='x')
plt.title("Ridge Regression: Actual vs Predicted House Prices")
plt.xlabel("Test Sample Index")
plt.ylabel("House Price")
plt.legend()
plt.tight_layout()
plt.show()
```

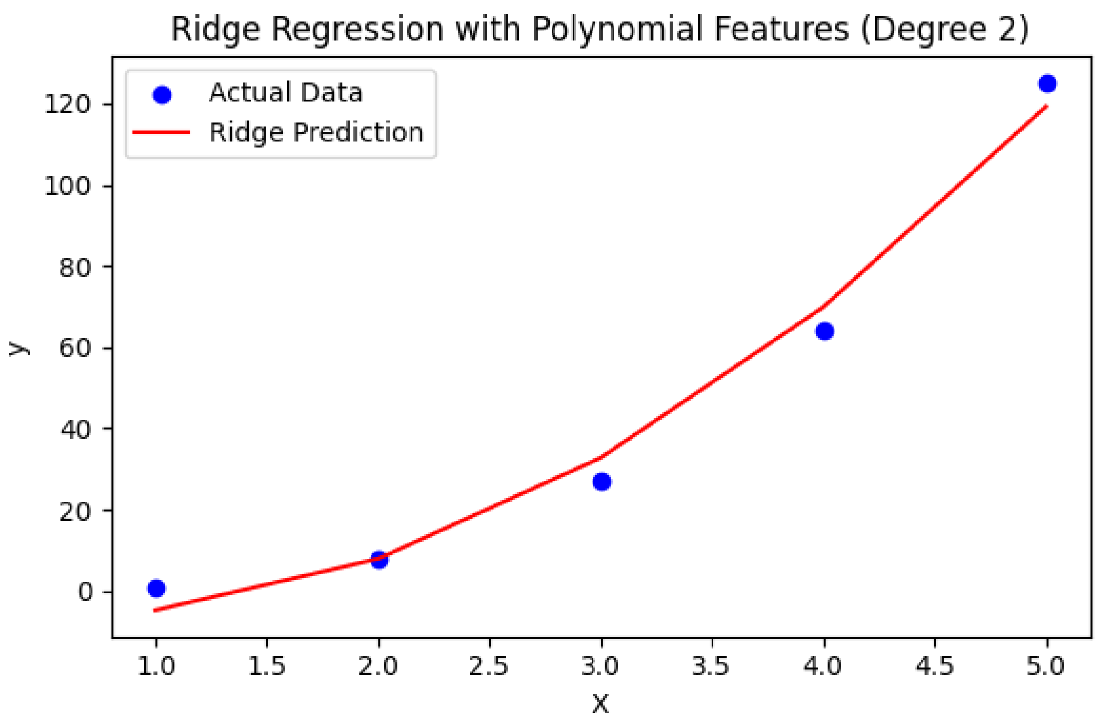
```
Ridge Predictions: [191131.77873259 399797.30055298]
MSE: 7502111.419731108
================================================================================
```



[23]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Sample dataset
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([1, 8, 27, 64, 125])  # y = x^3

# Ridge Regression with Polynomial Features
model = make_pipeline(PolynomialFeatures(degree=2), Ridge(alpha=1.0))
model.fit(X, y)

# Predictions
predictions = model.predict(X)
print("Ridge Polynomial Predictions:", predictions)
print("=" * 80)

# Plotting
plt.figure(figsize=(6, 4))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, predictions, color='red', label='Ridge Prediction')
```

```
plt.title("Ridge Regression with Polynomial Features (Degree 2)")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.tight_layout()
plt.show()
```

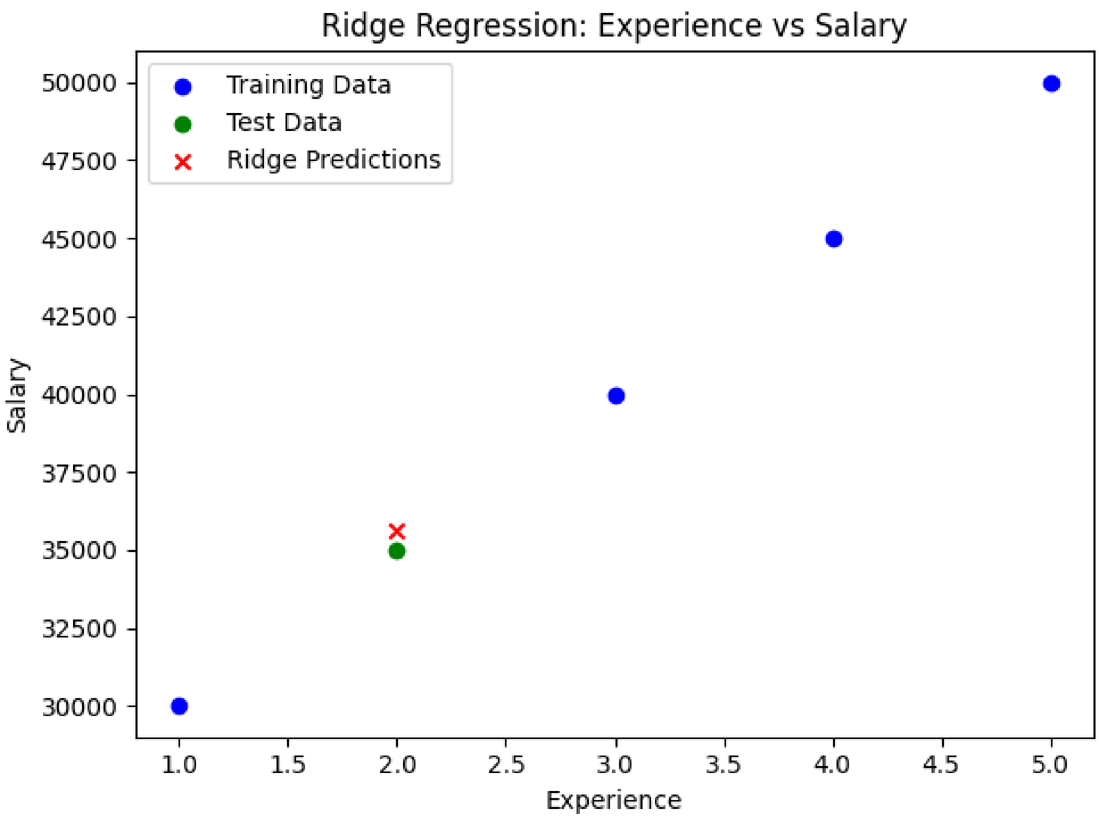
```
Ridge Polynomial Predictions: [ -4.71428571   7.91428571  32.77142857
 69.85714286 119.17142857]
================================================================================
```



[25]:
```
# Ridge Regression (Experience vs Salary)
model = Ridge(alpha=1.0)
model.fit(X_train, y_train)
ridge_predictions = model.predict(X_test)

print("Ridge Predictions:", ridge_predictions)
print("MSE:", mean_squared_error(y_test, ridge_predictions))
print("=" * 80)

# Plotting actual vs predicted
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='green', label='Test Data')
```

```
plt.scatter(X_test, ridge_predictions, color='red', marker='x', label='Ridge␣
  ↪Predictions')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.title('Ridge Regression: Experience vs Salary')
plt.legend()
plt.tight_layout()
plt.show()
```

```
Ridge Predictions: [35641.02564103]
MSE: 410913.8724523335
================================================================================
```



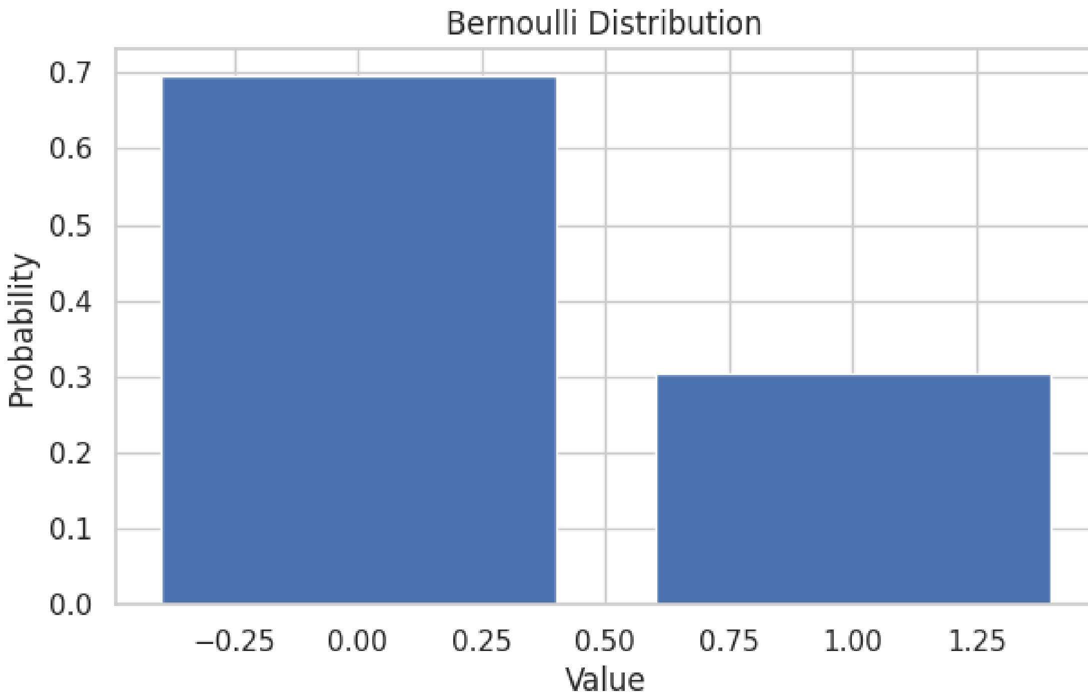
Distributions

Discrete Distributions

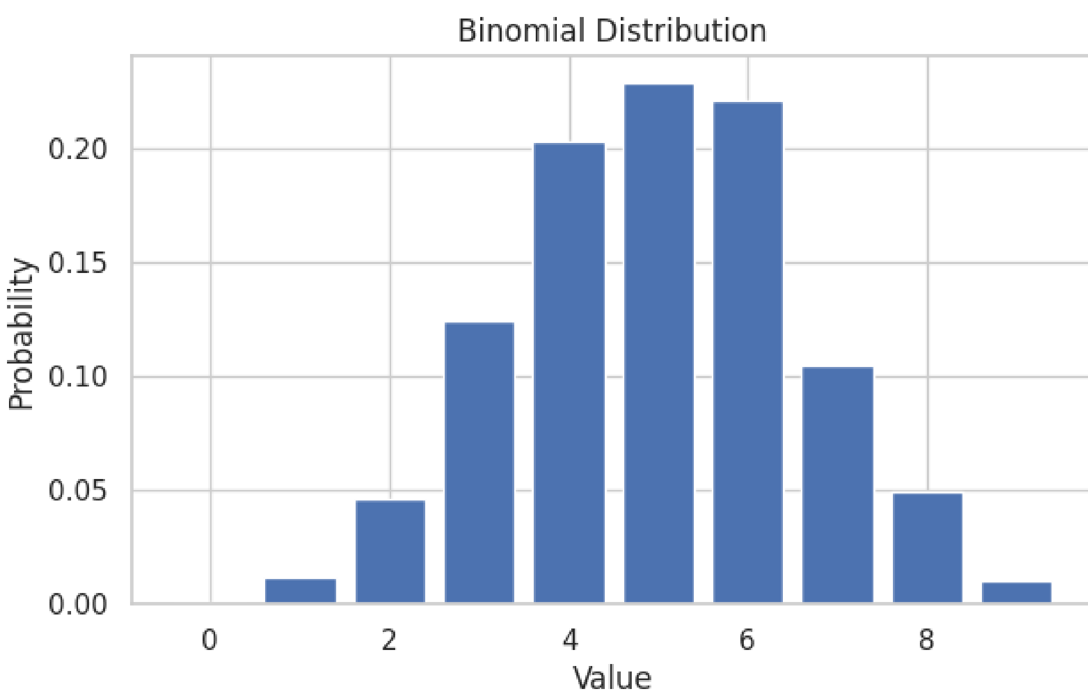
[26]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import bernoulli, binom, nbinom, geom, poisson
```

```
sns.set(style='whitegrid')

def plot_discrete(data, title):
    values, counts = np.unique(data, return_counts=True)
    plt.figure(figsize=(7, 4))
    plt.bar(values, counts / len(data))
    plt.title(title)
    plt.xlabel("Value")
    plt.ylabel("Probability")
    plt.show()

# 1. Uniform (Discrete)
data_uniform_discrete = np.random.randint(1, 7, size=1000)
plot_discrete(data_uniform_discrete, "Uniform Distribution (Discrete)")
```

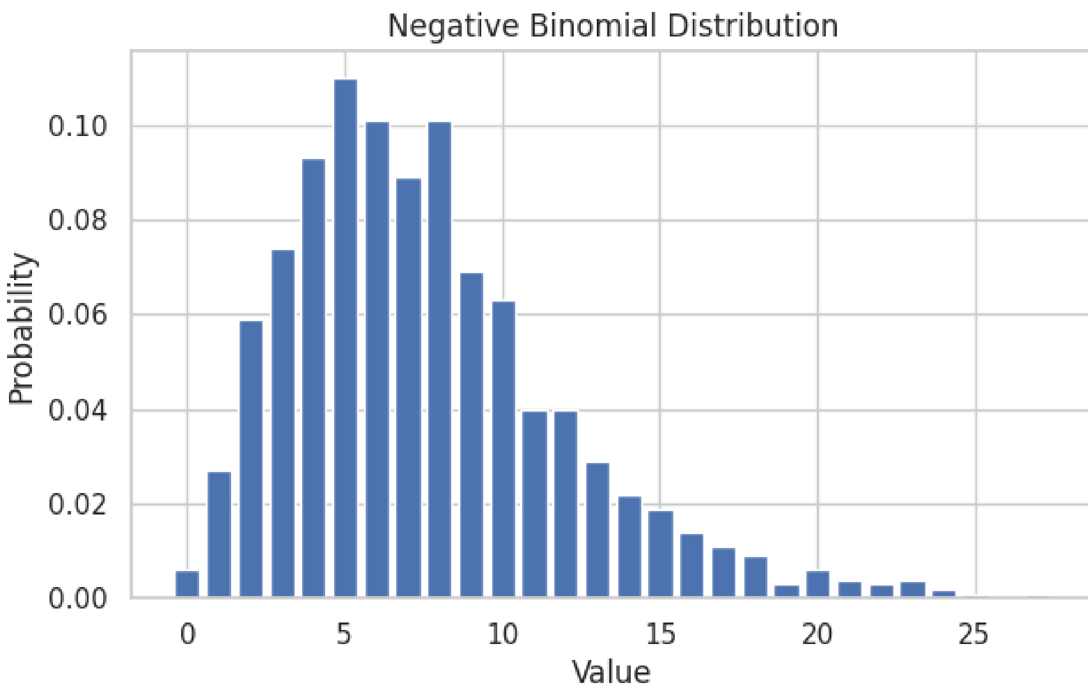
[27]:
```
# 2. Bernoulli
data_bernoulli = bernoulli.rvs(p=0.3, size=1000)
plot_discrete(data_bernoulli, "Bernoulli Distribution")
```



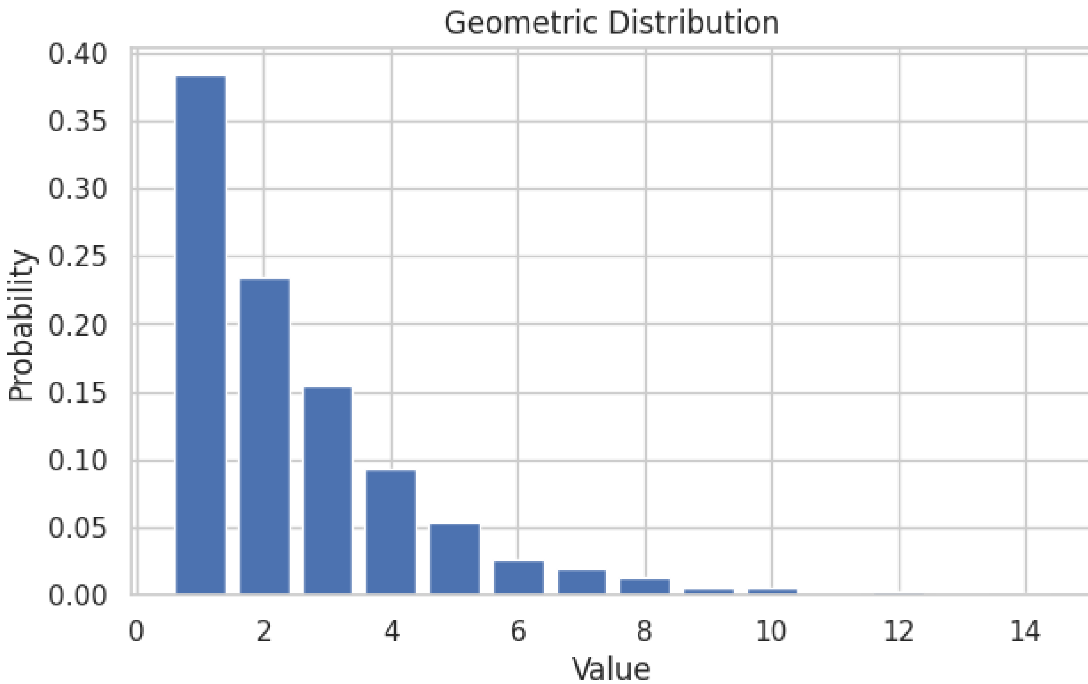
[28]:
```
# 3. Binomial
data_binomial = binom.rvs(n=10, p=0.5, size=1000)
plot_discrete(data_binomial, "Binomial Distribution")
```



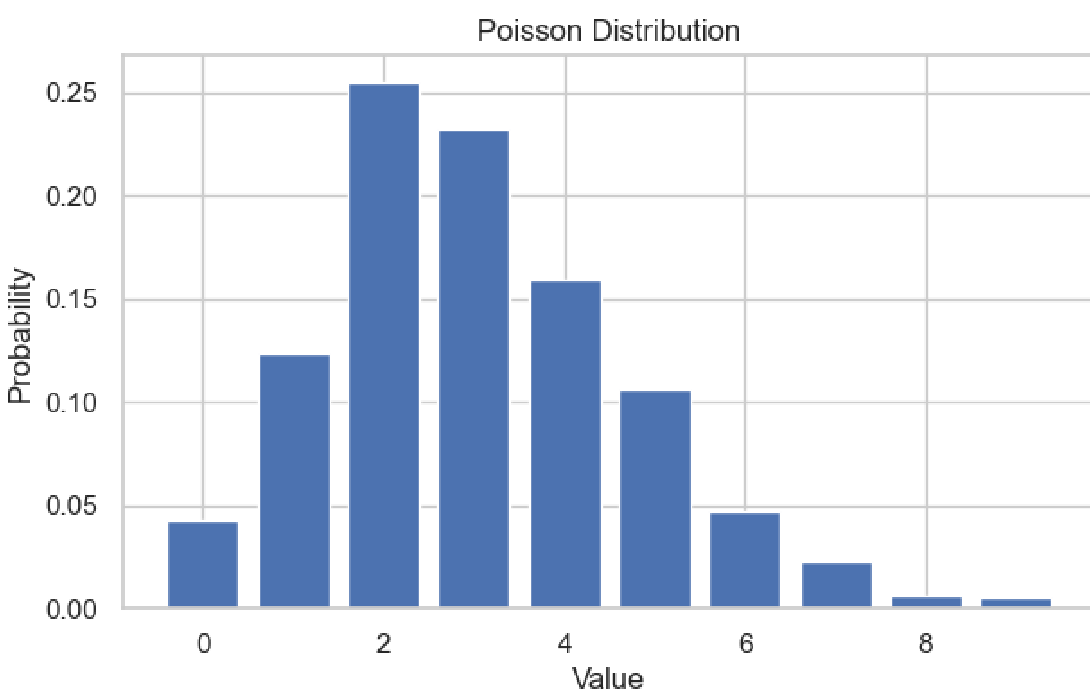
[29]:
```
# 4. Negative Binomial
data_neg_binomial = nbinom.rvs(n=5, p=0.4, size=1000)
plot_discrete(data_neg_binomial, "Negative Binomial Distribution")
```



[30]:
```
# 4b. Geometric
data_geometric = geom.rvs(p=0.4, size=1000)
plot_discrete(data_geometric, "Geometric Distribution")
```

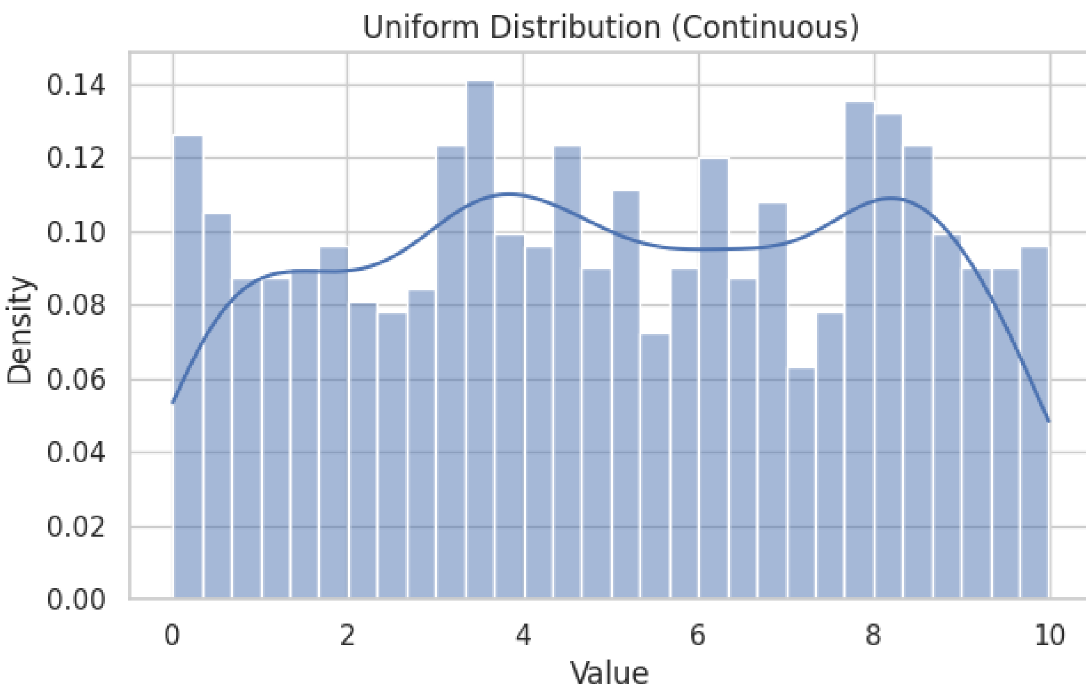


[184]:
```
# 5. Poisson
data_poisson = poisson.rvs(mu=3, size=1000)
plot_discrete(data_poisson, "Poisson Distribution")
```

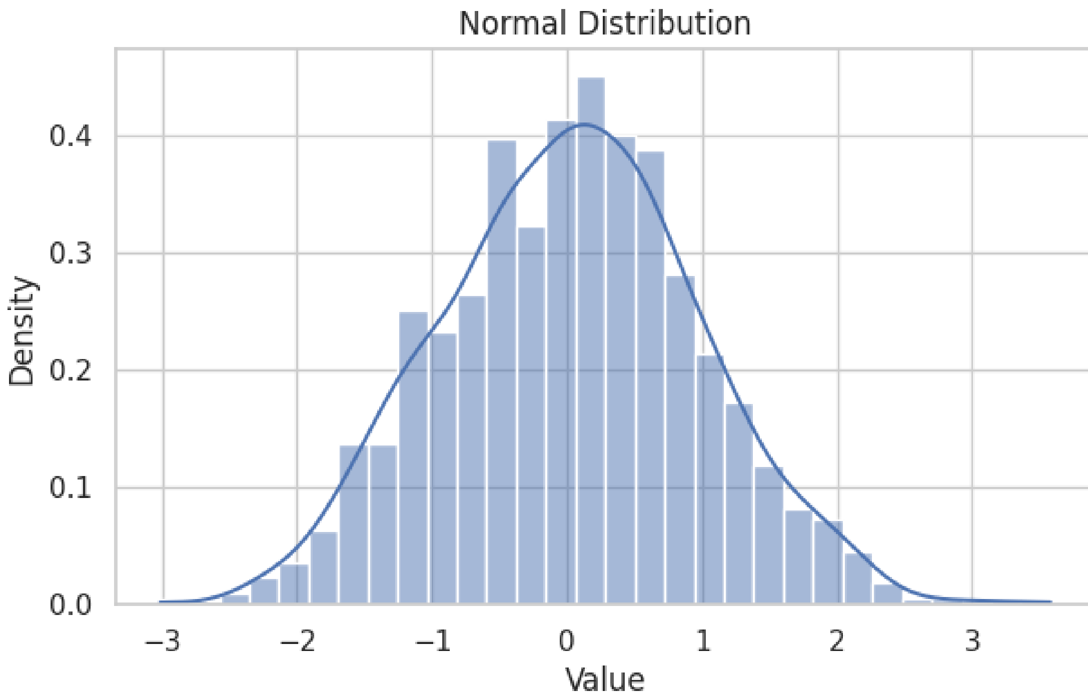


Continuous Distributions

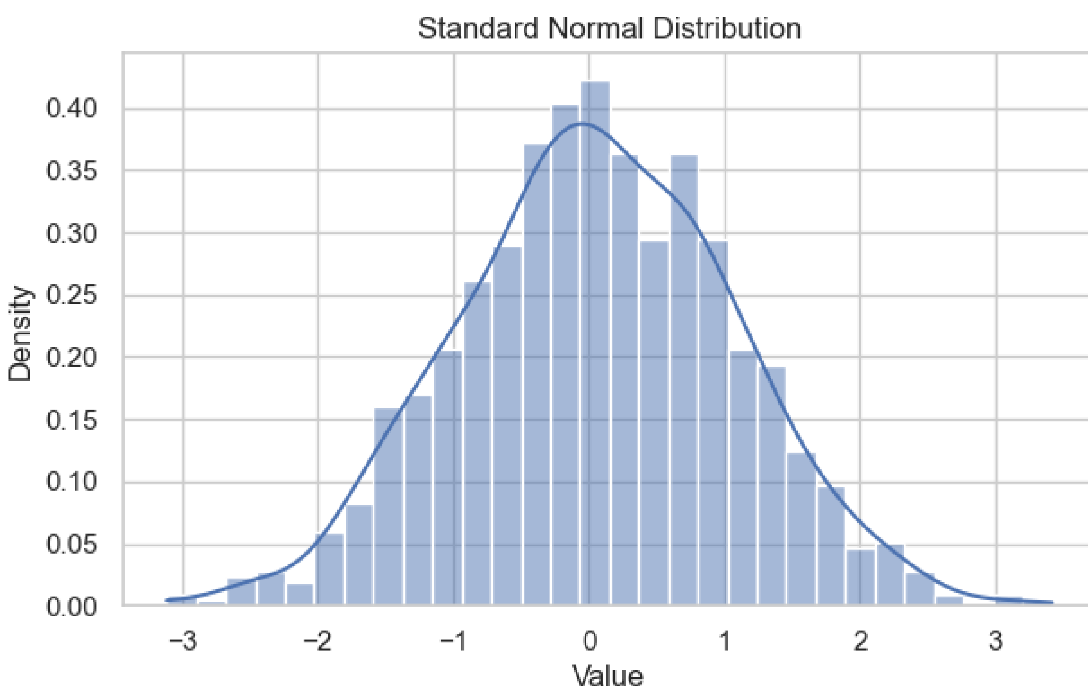
[31]:
```
from scipy.stats import uniform, norm, expon, gamma, beta, chi2

def plot_continuous(data, title, bins=30):
    plt.figure(figsize=(7, 4))
    sns.histplot(data, bins=bins, kde=True, stat="density")
    plt.title(title)
    plt.xlabel("Value")
    plt.ylabel("Density")
    plt.show()

# 1. Uniform (Continuous)
data_uniform_cont = uniform.rvs(loc=0, scale=10, size=1000)
plot_continuous(data_uniform_cont, "Uniform Distribution (Continuous)")
```



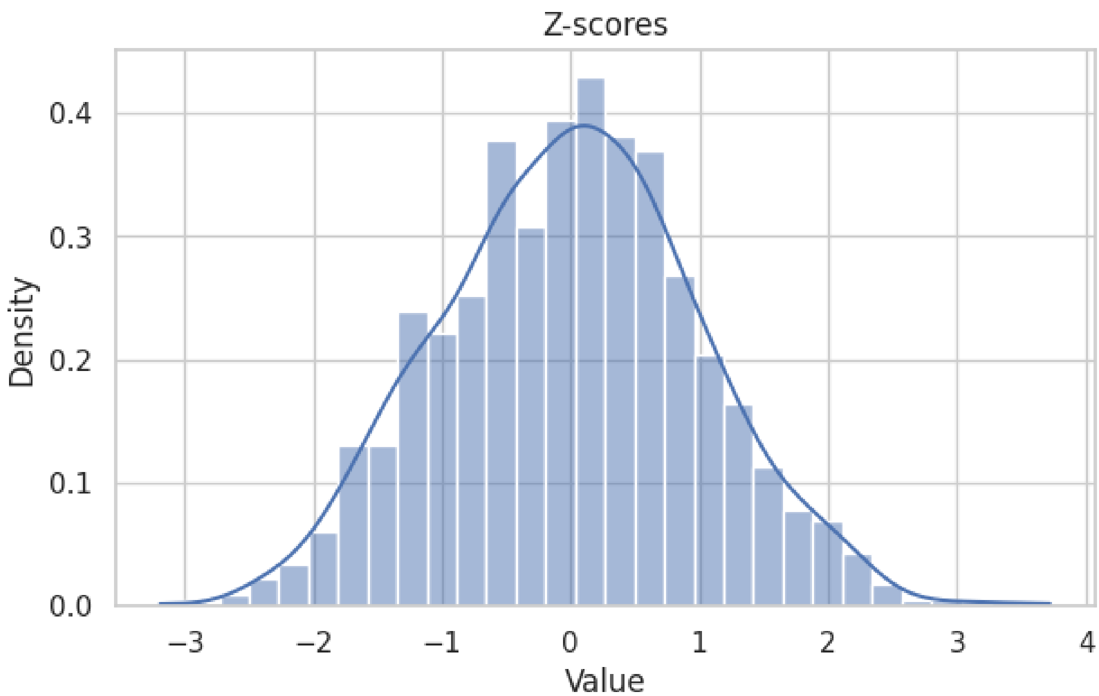
[32]:
```
# 2. Normal Distribution
data_normal = norm.rvs(loc=0, scale=1, size=1000)
plot_continuous(data_normal, "Normal Distribution")
```



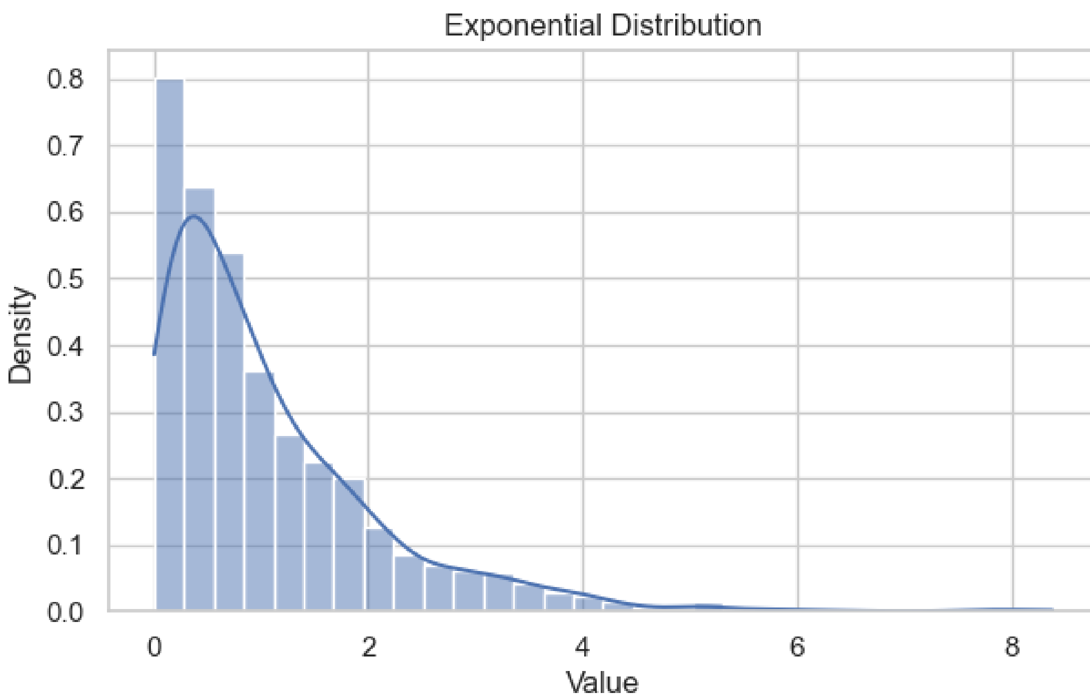
[188]:
```
# 3. Standard Normal
data_std_normal = norm.rvs(size=1000)
plot_continuous(data_std_normal, "Standard Normal Distribution")
```



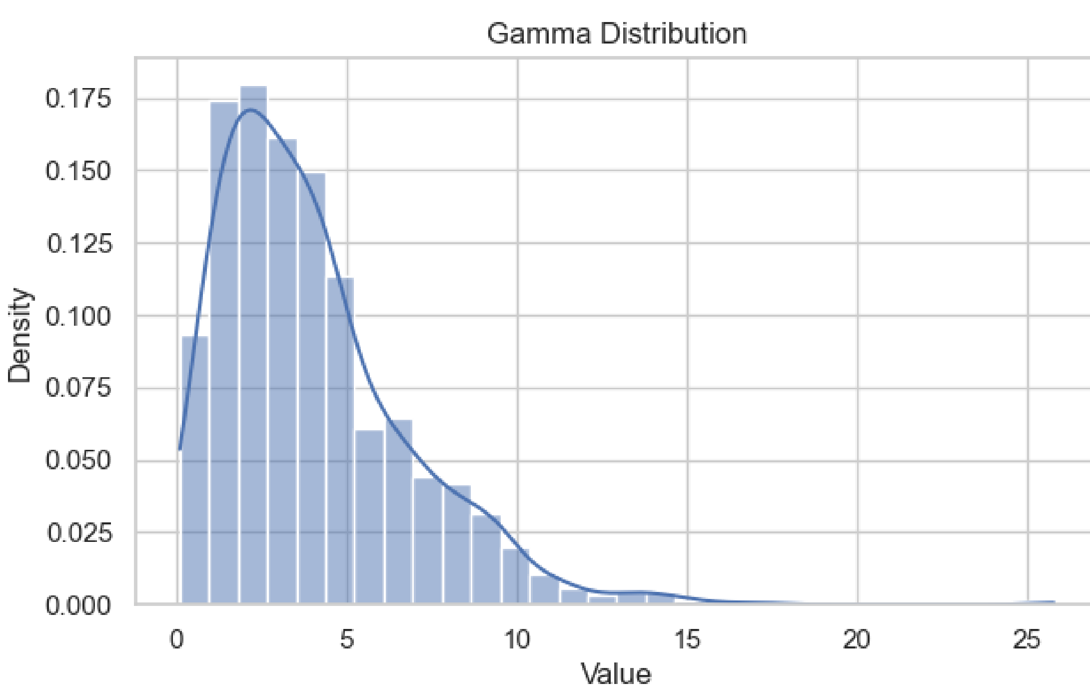
[33]:
```python
# 4. Z-scores
z_scores = (data_normal - np.mean(data_normal)) / np.std(data_normal)
plot_continuous(z_scores, "Z-scores")
```



[190]:
```python
# 5. Exponential
data_exponential = expon.rvs(scale=1, size=1000)
plot_continuous(data_exponential, "Exponential Distribution")
```

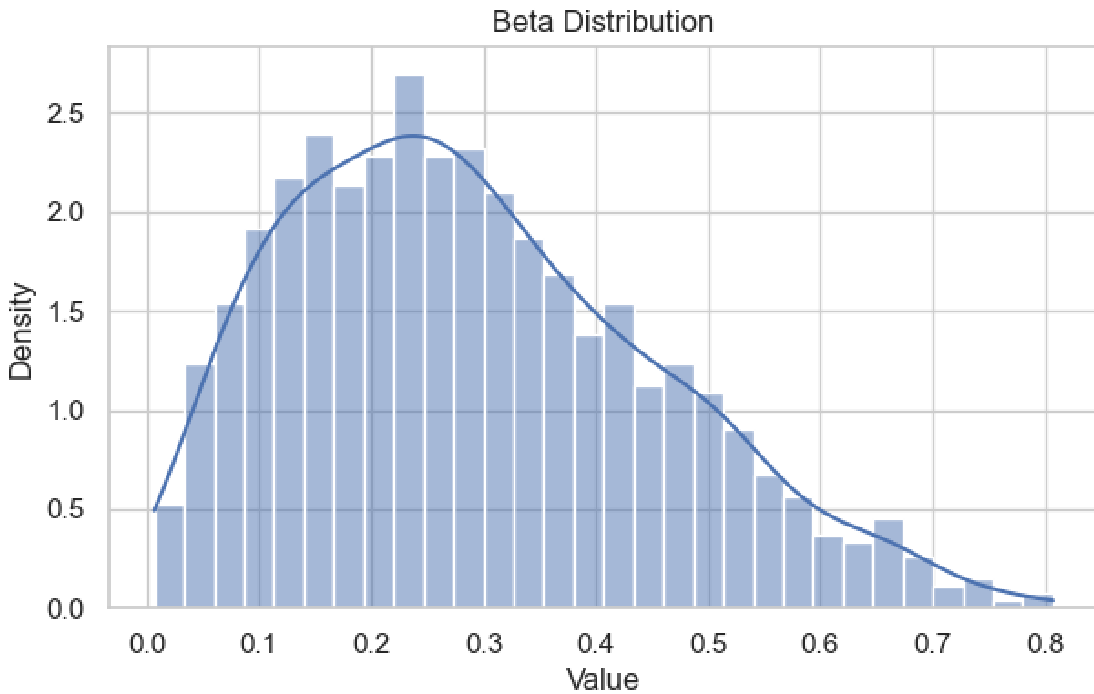



```
[191]: # 6. Gamma
data_gamma = gamma.rvs(a=2, scale=2, size=1000)
plot_continuous(data_gamma, "Gamma Distribution")
```



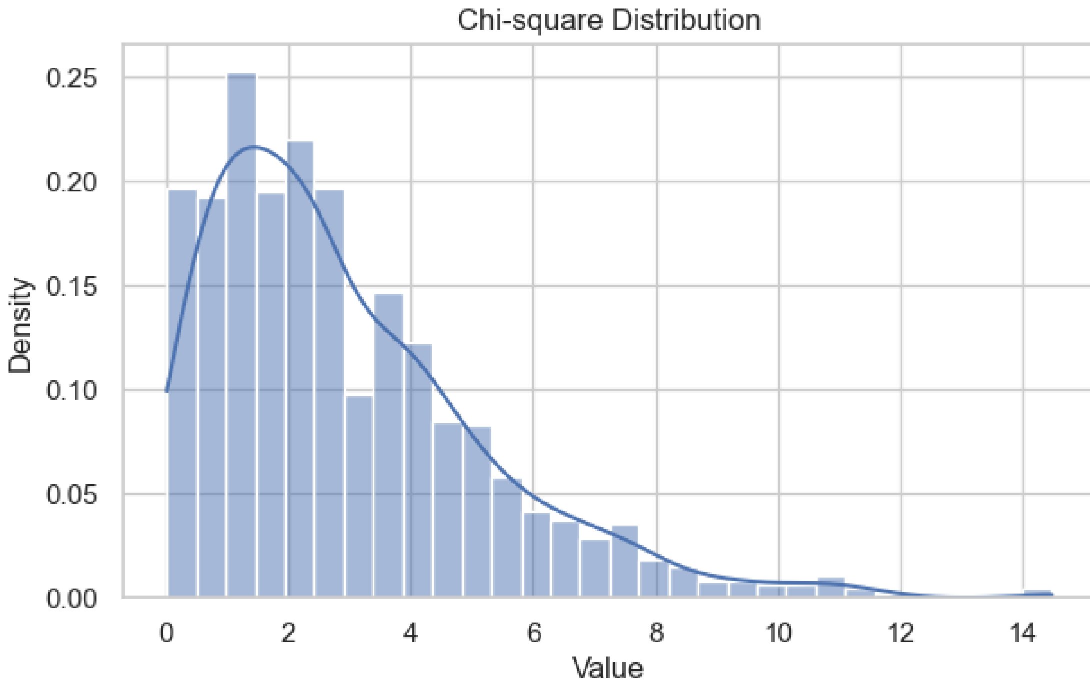
[192]:

```
# 7. Beta
data_beta = beta.rvs(a=2, b=5, size=1000)
plot_continuous(data_beta, "Beta Distribution")
```



[193]:

```
# 8. Chi-square
data_chi2 = chi2.rvs(df=3, size=1000)
plot_continuous(data_chi2, "Chi-square Distribution")
```


Chi-square Distribution
Density
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
12
14
Value