3rd Sem - Data Visuatization with Python - Lab Manual - Dr. Shankar R

1. a) Write a python program to find the best of two test average marks out of three test's marks accepted from the user. 1. b) Develop a Python program to check whether a given number is palindrome or not and also count the number of occurrences of each digit in the input number.
2. a) Defined as a function F as Fn = Fn-1 + Fn-2. Write a Python program which accepts a value for N (where N >0) as input and pass this value to the function. Display suitable error message if the condition for input value is not followed. 2. b) Develop a python program to convert binary to decimal, octal to hexadecimal using functions.
3. a) Write a Python program that accepts a sentence and find the number of words, digits, uppercase letters and lowercase letters. 3. b) Write a Python program to find the string similarity between two given strings
4. a) Write a Python program to Demonstrate how to Draw a Bar Plot using Matplotlib. 4. b) Write a Python program to Demonstrate how to Draw a Scatter Plot using Matplotlib.
5. a) Write a Python program to Demonstrate how to Draw a Histogram Plot using Matplotlib. 5. b) Write a Python program to Demonstrate how to Draw a Pie Chart using Matplotlib.
6. a) Write a Python program to illustrate Linear Plotting using Matplotlib. 6. b) Write a Python program to illustrate liner plotting with line formatting using Matplotlib.
7. Write a Python program which explains uses of customizing seaborn plots with Aesthetic functions.
8. a) Write a Python program to explain working with bokeh line graph using Annotations and Legends. b) Write a Python program for plotting different types of plots using Bokeh.
9. Write a Python program to draw 3D Plots using Plotly Libraries.
10. a) Write a Python program to draw Time Series using Plotly Libraries. b) Write a Python program for creating Maps using Plotly Libraries

1a) Write a python program to find the best of two test average marks out of three test's marks accepted from the user.

```
# Input three test marks from the user
test1Marks = float(input("Enter marks for Test 1: "))
test2Marks = float(input("Enter marks for Test 2: "))
test3Marks = float(input("Enter marks for Test 3: "))

# Store the marks in a list and sort them
marks = [test1Marks, test2Marks, test3Marks]
marks.sort()

# Calculate the average of the best two marks
bestTwoAverage = (marks[1] + marks[2]) / 2

# Display the result
print("Average of the best two marks:", bestTwoAverage)
```

```
Enter marks for Test 1: 94
Enter marks for Test 2: 46
Enter marks for Test 3: 87
Average of the best two marks: 90.5
```

1b) Develop a Python program to check whetSher a given number is palindrome or not and also count the number of occurrences of each digit in the input number.

```
# Input the number from the user
originalNumber = input("Enter a number: ")
reversedNumber = originalNumber[::-1]

# Check if the number is a palindrome
if originalNumber == reversedNumber:
    print(f"{originalNumber} is a palindrome.")
else:
    print(f"{originalNumber} is not a palindrome.")

# Display digit occurrences
print("Digit occurrences:")
for digit in set(originalNumber):
    count = originalNumber.count(digit)
    print(f"{digit}: {count}")
```

```
Enter a number: 08096114528415248754332 3
0809611452841524875433 23 is not a palindrome.
Digit occurrences:
6: 1
7: 1
1: 3
0: 2
9: 1
8: 3
5: 3
4: 4
2: 3
3: 3
```

2a) Defined as a function F as Fn = Fn-1 + Fn-2. Write a Python program which accepts a value for N (where N >0) as input and pass this value to the function. Display suitable error message if the condition for input value is not followed.

```
def fibonacci(n):
    # Base case
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# Input the value for N
n = input("Enter a value for N (N > 0): ")

# Ensure the input is a positive integer
if n.isdigit() and int(n) > 0:
    n = int(n)
    print("Fibonacci sequence:")
    for i in range(1, n + 1):
        print(fibonacci(i), end=' ')  # Print each Fibonacci number
else:
    print("Error: Please enter a valid positive integer.")
```

```
Enter a value for N (N > 0): 15
Fibonacci sequence:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

2b) Develop a python program to convert binary to decimal, octal to hexadecimal using functions.

```
def binaryToDecimal(binaryStr):
    # Convert binary string to decimal integer
    return int(binaryStr, 2)

def octalToHexadecimal(octalStr):
    # Convert octal string to decimal first, then to hexadecimal
    decimalValue = int(octalStr, 8)
    return hex(decimalValue)[2:].upper()  # Convert to hexadecimal and remove '0x'

# Input binary string
binaryInput = input("Enter a binary number: ")
decimalOutput = binaryToDecimal(binaryInput)
print(f"Binary {binaryInput} to Decimal is: {decimalOutput}")

# Input octal string
octalInput = input("Enter an octal number: ")
hexadecimalOutput = octalToHexadecimal(octalInput)
print(f"Octal {octalInput} to Hexadecimal is: {hexadecimalOutput}")
```

```
Enter a binary number: 100101100111001
Binary 100101100111001 to Decimal is: 19257
Enter an octal number: 560064
Octal 560064 to Hexadecimal is: 2E034
```

Output Explanation:

$(100101100111001)_2 = (1 \times 2^{14}) + (0 \times 2^{13}) + (0 \times 2^{12}) + (1 \times 2^{11}) + (0 \times 2^{10}) + (1 \times 2^9) + (1 \times 2^8) + (0 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = (19257)_{10}$

So Binary 100101100111001 to Decimal is: 19257

Convert every octal digit to 3 binary digits, then convert every 4 binary digits to1 hex digit:

560064 = 5 6 0 0 6 4

= 101 - 110 - 000 - 000 - 110 - 100

= 10 - 1110 - 0000 - 0011 - 0100

= 2 E 0 3 4

= 2E034

So Octal 560064 to Hexadecimal is: 2E034

3a) Write a Python program that accepts a sentence and find the number of words, digits, uppercase letters and lowercase letters.

```
sentence = input("Enter a sentence: ")

words = len(sentence.split())
digits = 0
uppercase = 0
lowercase = 0

for c in sentence:
    if c.isdigit():
        digits += 1
    elif c.isupper():
```

```
        uppercase += 1
    elif c.islower():
        lowercase += 1

print("Words:", words)
print("Digits:", digits)
print("Uppercase:", uppercase)
print("Lowercase:", lowercase)
```

```
Enter a sentence: BMSIT has increased its 2024 batch intake. Total Strength is CSE 900 AIML 360 ECE 180 EEE 60 MECHANICAL 60 CIVIL 60
Words: 22
Digits: 19
Uppercase: 35
Lowercase: 39
```

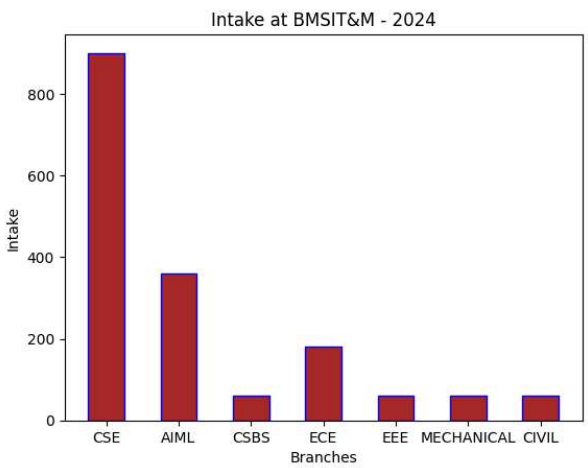
3b) Write a Python program to find the string similarity between two given strings

```
# Input two strings from the user
string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")

# Convert strings to sets to find common characters
set1 = set(string1)
set2 = set(string2)

# Calculate the number of common characters
common = set1.intersection(set2)

# Calculate the similarity ratio
similarity = len(common) / max(len(set1), len(set2))

# Print similarity percentage
print("Similarity:", similarity * 100, "%")
```

```
Enter the first string: bmsit
Enter the second string: bmsce
Similarity: 60.0 %
```

4a) Write a Python program to Demonstrate how to Draw a Bar Plot using Matplotlib.

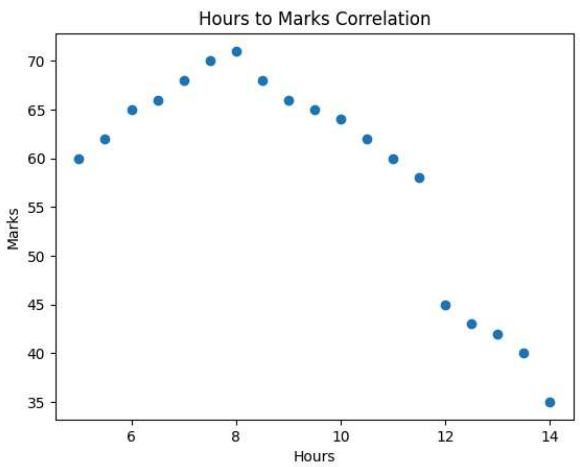
```
import matplotlib.pyplot as plt
branches = ["CSE","AIML","CSBS","ECE","EEE","MECHANICAL","CIVIL"]
intake = [900,360,60,180,60,60,60]

plt.bar(branches,intake,color="brown",edgecolor="blue",width = 0.5)

plt.title("Intake at BMSIT&M - 2024")
plt.xlabel("Branches")
plt.ylabel("Intake")

plt.show()
```



4b) Write a Python program to Demonstrate how to Draw a Scatter Plot using Matplotlib.

```
import matplotlib.pyplot as plt
hours = [5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10,10.5,11,11.5,12,12.5,13,13.5,14]
marks = [60,62,65,66,68,70,71,68,66,65,64,62,60,58,45,43,42,40,35]

plt.scatter(hours,marks)
```

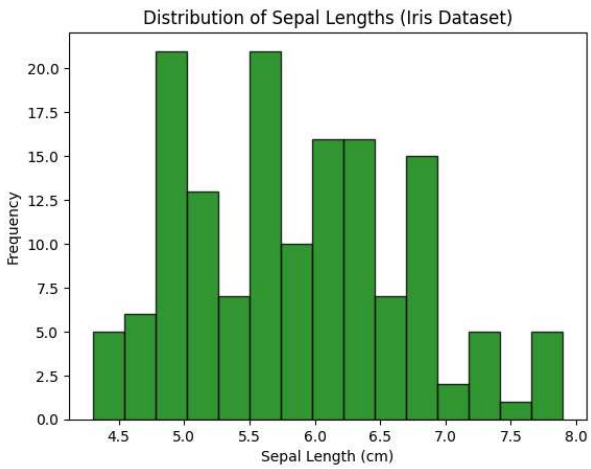
```
plt.title("Hours to Marks Correlation")
plt.xlabel("Hours")
plt.ylabel("Marks")

plt.show()
```



5a. Write a Python program to Demonstrate how to Draw a Histogram Plot using Matplotlib.

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()

# Extract the sepal length feature (first column of the data)
sepal_lengths = iris.data[:, 0]

print(sepal_lengths)

# Creating the histogram
plt.hist(sepal_lengths, bins=15, color='green', edgecolor='black', alpha=0.8)

# Adding titles and labels
plt.title('Distribution of Sepal Lengths (Iris Dataset)')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Frequency')

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

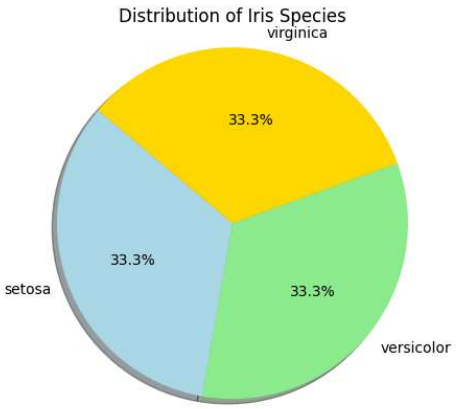
```
[5.1 4.9 4.7 4.6 5.  5.4 4.6 5.  4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.  5.  5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.
 5.5 4.9 4.4 5.1 5.  4.5 4.4 5.  5.1 4.8 5.1 4.6 5.3 5.  7.  6.4 6.9 5.5
 6.5 5.7 6.3 4.9 6.6 5.2 5.  5.9 6.  6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
 6.3 6.1 6.4 6.6 6.8 6.7 6.  5.7 5.5 5.5 5.8 6.  5.4 6.  6.7 6.3 5.6 5.5
 5.5 6.1 5.8 5.  5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.  6.9 5.6 7.7 6.3 6.7 7.2
 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.  6.9 6.7 6.9 5.8 6.8
 6.7 6.7 6.3 6.5 6.2 5.9]
```



5b) Write a Python program to Demonstrate how to Draw a Pie Chart using Matplotlib.

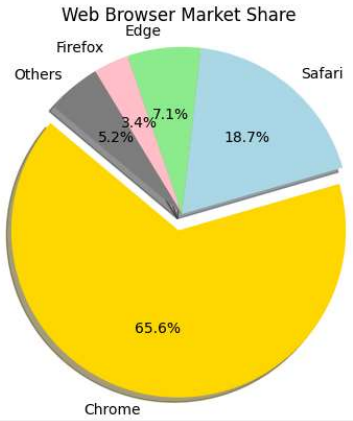
5b version 1 - with most commonly used machine learning dataset (iris dataset)

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()

# Count occurrences of each species
species_counts = np.bincount(iris.target)

# Labels and sizes
labels = iris.target_names  # Species names: 'setosa', 'versicolor', 'virginica'
sizes = species_counts      # Number of samples per species

# Colors for the pie chart
colors = ['lightblue', 'lightgreen', 'gold']

# Create the pie chart
plt.pie(
    sizes,
    labels=labels,
    colors=colors,
    autopct='%1.1f%%',
    shadow=True,
    startangle=140
)

# Add a title
plt.title('Distribution of Iris Species')

# Display the chart
plt.axis('equal')  # Ensures the pie is a circle
plt.show()
```



5b version 2 - without dataset

```
import matplotlib.pyplot as plt

# Data for web browser market share
labels = ['Chrome', 'Safari', 'Edge', 'Firefox', 'Others']
sizes = [65.6, 18.7, 7.1, 3.4, 5.2]  # Market share percentages
colors = ['gold', 'lightblue', 'lightgreen', 'pink', 'gray']  # Colors for slices
explode = (0.1, 0, 0, 0, 0)  # Highlight the largest slice (Chrome)

# Create the pie chart
plt.pie(
    sizes,
    explode=explode,
    labels=labels,
    colors=colors,
    autopct='%1.1f%%',
    shadow=True,
    startangle=140
)

# Add a title
plt.title('Web Browser Market Share')

# Display the chart
plt.axis('equal')  # Ensures the pie is a circle
plt.show()
```



6a) Write a Python program to illustrate Linear Plotting using Matplotlib.

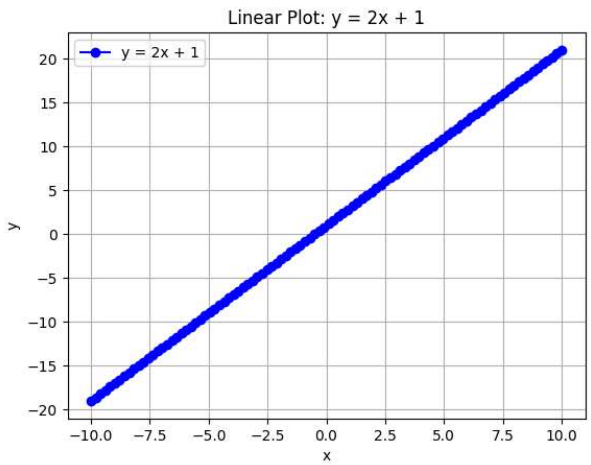
6a version 1 - less data

```
import matplotlib.pyplot as plt
import numpy as np

# Realistic data: Years of experience vs. Salary (in thousands)
years_of_experience = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
salary = np.array([30, 35, 40, 45, 50, 55, 60, 65, 70, 75])

# Create the linear plot
plt.plot(years_of_experience, salary, label='Salary vs. Experience', color='green', linestyle='-', marker='o')

# Adding title and labels
plt.title('Salary vs. Years of Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary (in thousands)')

# Displaying the legend
plt.legend()

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

6a version 2 - more data

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(-10, 10, 100)  # Generate 100 values from -10 to 10
y = 2 * x + 1  # Linear equation y = 2x + 1

# Create the plot
plt.plot(x, y, label='y = 2x + 1', color='blue', linestyle='-', marker='o')

# Adding title and labels
plt.title('Linear Plot: y = 2x + 1')
plt.xlabel('x')
```

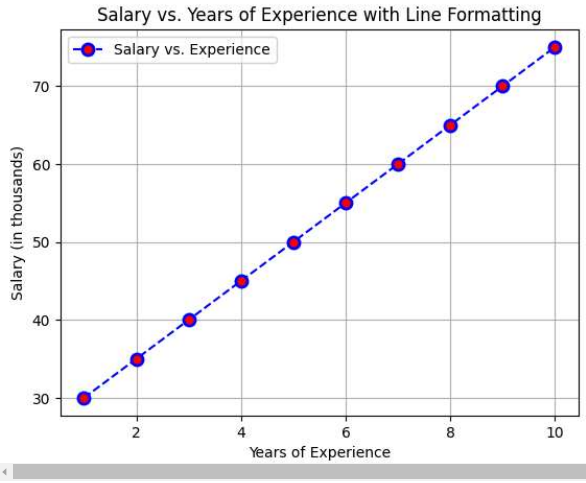
```
plt.ylabel('y')

# Displaying the legend
plt.legend()

# Display the plot
plt.grid(True)  # Add grid for better readability
plt.show()
```



6b) Write a Python program to illustrate linear plotting with line formatting using Matplotlib.

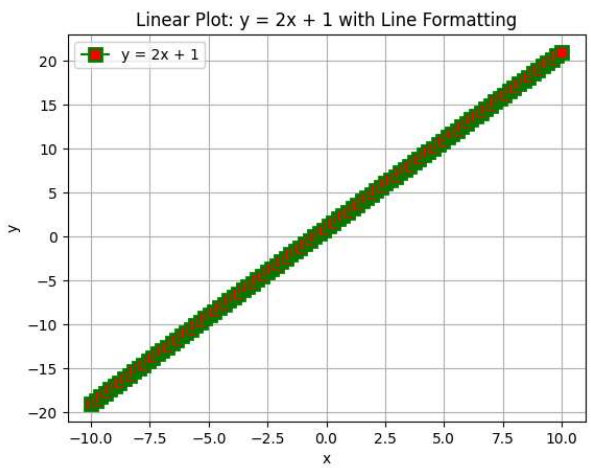
6b version 1 - less data

```
import matplotlib.pyplot as plt
import numpy as np

# Realistic data: Years of experience vs. Salary (in thousands)
years_of_experience = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
salary = np.array([30, 35, 40, 45, 50, 55, 60, 65, 70, 75])

# Create the linear plot with different line formatting
plt.plot(years_of_experience, salary,
         label='Salary vs. Experience',
         color='blue',
         linestyle='--',  # Dashed line
         marker='o',      # Circle markers
         markersize=8,    # Marker size
         markerfacecolor='red',  # Marker color
         markeredgewidth=2)  # Edge width of markers

# Adding title and labels
plt.title('Salary vs. Years of Experience with Line Formatting')
plt.xlabel('Years of Experience')
plt.ylabel('Salary (in thousands)')

# Displaying the legend
plt.legend()

# Display the plot with grid
plt.grid(True)

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



6b version 2 - more data

```
import matplotlib.pyplot as plt
import numpy as np

# Data for the plot
x = np.linspace(-10, 10, 100)  # Generate 100 values from -10 to 10
y = 2 * x + 1  # Linear equation y = 2x + 1

# Create the plot with line formatting
plt.plot(x, y,
         label='y = 2x + 1',    # Label for the legend
         color='green',         # Line color
         linestyle='-.',        # Dash-dot line style
         marker='s',            # Square markers
         markersize=8,          # Size of the markers
         markerfacecolor='red', # Marker fill color
         markeredgewidth=2)     # Border width of the markers

# Adding title and labels
plt.title('Linear Plot: y = 2x + 1 with Line Formatting')
plt.xlabel('x')
plt.ylabel('y')

# Displaying the legend
plt.legend()

# Display the plot with grid
plt.grid(True)  # Add grid for better readability
plt.show()
```



7) Write a Python program which explains uses of customizing seaborn plots with Aesthetic functions.

lets understand seaborn

Seaborn: A Python Visualization Library

Seaborn is a **Python visualization library** built on top of **Matplotlib** that provides a high-level interface for drawing attractive and informative statistical graphics. It simplifies the process of creating complex visualizations, especially for exploring relationships between variables in datasets, by offering a more convenient API and additional functionalities.

Key Features of Seaborn:

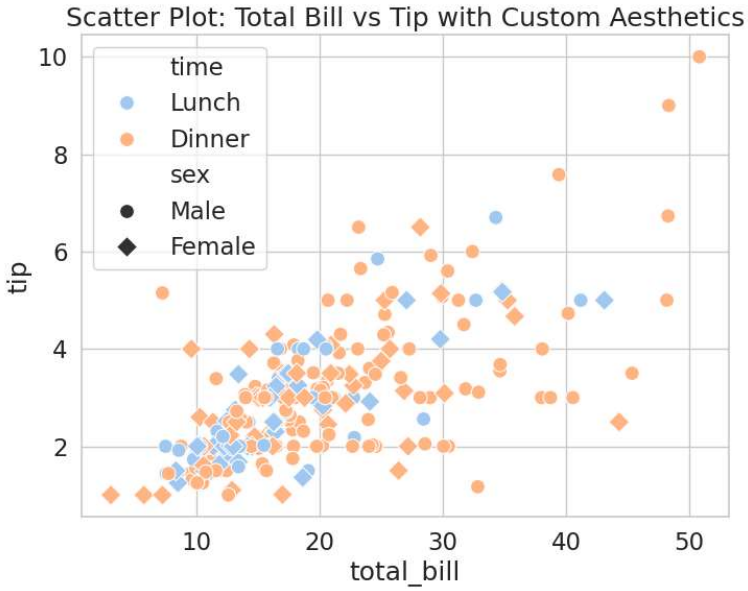
1. **High-Level Interface**:
   Seaborn offers a more concise and easy-to-use interface for creating common types of plots such as scatter plots, bar plots, histograms, and heatmaps, compared to Matplotlib.
2. **Statistical Plots**:
   Seaborn makes it easier to create complex statistical plots like regression plots, box plots, violin plots, pair plots, etc., which are helpful for data analysis.
3. **Built-in Datasets**:
   Seaborn includes a number of **example datasets** (like `tips`, `iris`, `penguins`, etc.) that make it easier to practice and test visualizations without needing external data.
4. **Better Aesthetics**:
   The library improves upon **Matplotlib's default appearance** by using attractive color palettes, automatic styles, and formatting for plot elements.
5. **Integration with Pandas**:
   Seaborn integrates well with **Pandas DataFrames** (which are common in data analysis), making it easy to plot directly from data in a DataFrame.
6. **Customization**:
   Seaborn provides several functions for controlling **plot appearance**, colors, and style to make plots more polished and readable.

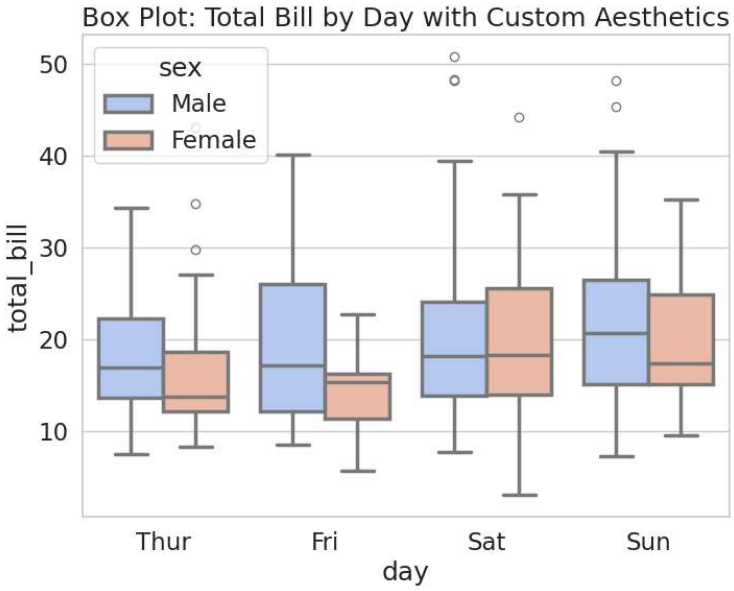
Commonly Used Seaborn Functions:

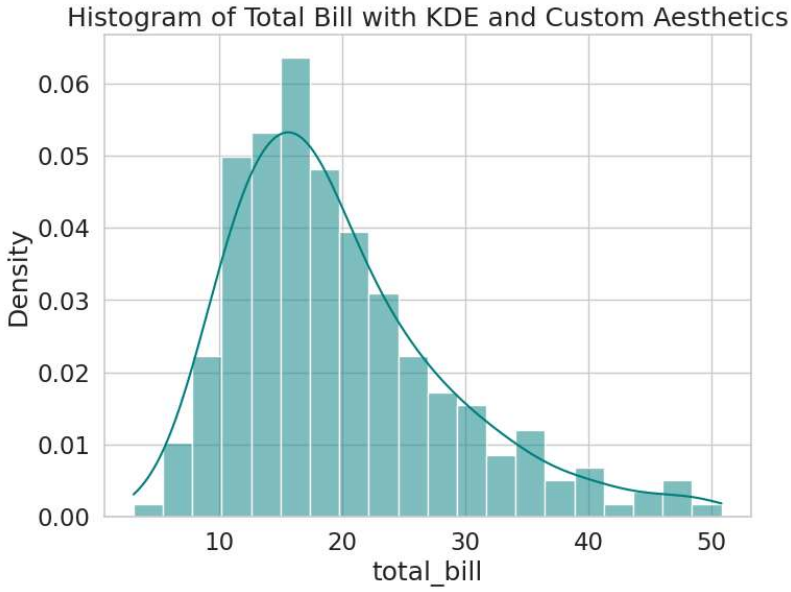
- `sns.scatterplot()` : Creates a **scatter plot**.
- `sns.lineplot()` : Creates a **line plot**.
- `sns.barplot()` : Creates a **bar plot**.
- `sns.histplot()` : Creates a **histogram**.
- `sns.heatmap()` : Creates a **heatmap**.
- `sns.boxplot()` : Creates a **box plot**.
- `sns.violinplot()` : Creates a **violin plot**.
- `sns.pairplot()` : Creates **pairwise plots** for numerical variables in a dataset.

Now lets do 7th program - Write a Python program which explains uses of customizing seaborn plots with Aesthetic functions

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

# Load the example dataset from seaborn
tips = sns.load_dataset("tips")

# Set Seaborn aesthetic functions with default font
sns.set_theme(style="whitegrid", palette="pastel", font_scale=1.5)

# Plot 1: Scatter plot with customized aesthetics
plt.figure(figsize=(8, 6))
sns.scatterplot(x="total_bill", y="tip", data=tips, hue="time", style="sex", s=100, markers=["o", "D"])
plt.title('Scatter Plot: Total Bill vs Tip with Custom Aesthetics', fontsize=18)
plt.show()

# Plot 2: Box plot with customized aesthetics and hue
plt.figure(figsize=(8, 6))
sns.boxplot(x="day", y="total_bill", data=tips, hue="sex", palette="coolwarm", linewidth=2.5)
plt.title('Box Plot: Total Bill by Day with Custom Aesthetics', fontsize=18)
plt.show()

# Plot 3: Histogram with customized aesthetics
plt.figure(figsize=(8, 6))
sns.histplot(tips["total_bill"], kde=True, color="teal", bins=20, stat="density")
plt.title('Histogram of Total Bill with KDE and Custom Aesthetics', fontsize=18)
plt.show()

# Plot 4: Heatmap with customized aesthetics
# Select only numeric columns for correlation
numeric_cols = tips.select_dtypes(include=[np.number]).columns
correlation = tips[numeric_cols].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation, annot=True, cmap="YlGnBu", linewidths=1.5, linecolor="gray", fmt=".2f", cbar_kws={"shrink": 0.8})
plt.title('Heatmap: Correlation Matrix with Custom Aesthetics', fontsize=18)
plt.show()
```


Scatter Plot: Total Bill vs Tip with Custom Aesthetics
time
Lunch
Dinner
sex
Male
Female
tip
total_bill


Box Plot: Total Bill by Day with Custom Aesthetics
sex
Male
Female
total_bill
Thur
Fri
Sat
Sun
day


Histogram of Total Bill with KDE and Custom Aesthetics
Density
total_bill

Heatmap: Correlation Matrix with Custom Aesthetics
total_bill
tip
1.00
0.68
0.60
0.68
1.00
0.49

8a) Write a Python program to explain working with bokeh line graph using Annotations and Legends.

What is Bokeh?

Bokeh is an interactive visualization library for Python that enables the creation of rich, interactive plots and dashboards for modern web browsers. It's designed to provide easy-to-use interfaces for creating interactive visualizations while also allowing for powerful customizations. Bokeh's focus is on providing interactive plots, which are highly suitable for data exploration and presentation on web platforms.

Key Features of Bokeh:

1. **Interactive Plots**:
   Bokeh makes it simple to create highly interactive plots that can respond to user actions such as zooming, panning, hovering, and clicking. This interactivity is often used in data visualization applications to explore large datasets and highlight important trends.
2. **Web-Based Output**:
   Bokeh generates plots that are easy to embed in web applications or dashboards. You can output visualizations as HTML files, which makes it simple to integrate them into web applications and websites.
3. **Versatile Plot Types**:
   Bokeh supports a wide variety of plot types, including:
   - **Scatter plots**
   - **Line plots**
   - **Bar plots**
   - **Heatmaps**
   - **Box plots**
   - **Histograms**
   - **Geospatial maps**, etc.
4. **Customizable**:
   Bokeh allows for fine control over plot appearance and behavior, letting users customize everything from plot dimensions and styles to axes, grids, tooltips, and interactivity.
5. **Integration with Other Libraries**:
   Bokeh can be used alongside other Python data science libraries like **Pandas**, **NumPy**, **SciPy**, and **Matplotlib**. It also integrates well with web frameworks such as **Flask** and **Django**.
6. **Streaming and Real-Time Data**:
   Bokeh supports real-time data streaming, allowing users to create live dashboards and plots that update automatically as new data becomes available.
7. **Server-Backed Visualizations**:
   Bokeh provides a powerful server application (`bokeh.server`) that can be used to serve dynamic, interactive visualizations that are updated in real-time.

Advantages of Using Bokeh:

- **Highly Interactive**: It allows users to interact with visualizations in a variety of ways (zoom, pan, hover, click).
- **Web-Based**: Bokeh outputs in a web-friendly format, which makes it suitable for web development.
- **Responsive**: Plots are responsive to screen size and can adjust according to the container's dimensions.
- **Integrates with other Python Libraries**: Easily integrates with other data manipulation libraries like Pandas and NumPy.

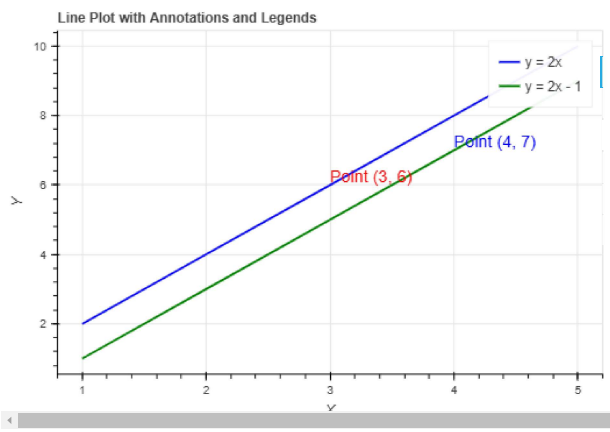
Commonly Used Bokeh Functions:

- `figure()` : The main function used to create a plot.
- `line()` : Creates a line plot.
- `scatter()` : Creates a scatter plot.
- `show()` : Renders and displays the plot in a web browser or Jupyter notebook.
- `output_file()` : Specifies the file path to save the plot as an HTML file.
- `output_notebook()` : Displays the plot inline within a Jupyter notebook.

Now lets do - 8a) Write a Python program to explain working with bokeh line graph using Annotations and Legends.

```
from bokeh.plotting import figure, show
from bokeh.models import Label

# Create a new plot with a title, axis labels, and dimensions
p = figure(title="Line Plot with Annotations and Legends",
           x_axis_label="X", y_axis_label="Y", width=600, height=400)
```

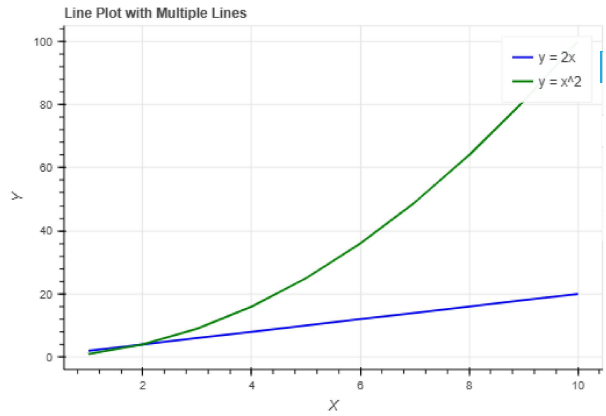
```
# Data for the lines
x = [1, 2, 3, 4, 5]
y1 = [2, 4, 6, 8, 10]
y2 = [1, 3, 5, 7, 9]

# Add first line with blue color and a label in the legend
p.line(x, y1, legend_label="y = 2x", line_width=2, color="blue")

# Add second line with green color and a label in the legend
p.line(x, y2, legend_label="y = 2x - 1", line_width=2, color="green")

# Add annotations to the plot
annotation1 = Label(x=3, y=6, text="Point (3, 6)", text_font_size="12pt", text_color="red")
annotation2 = Label(x=4, y=7, text="Point (4, 7)", text_font_size="12pt", text_color="blue")

# Add the annotations to the plot
p.add_layout(annotation1)
p.add_layout(annotation2)

# Show the plot
show(p)
```

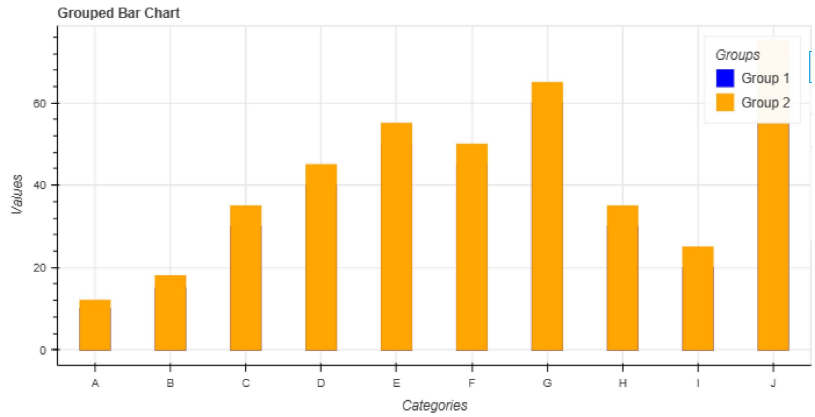


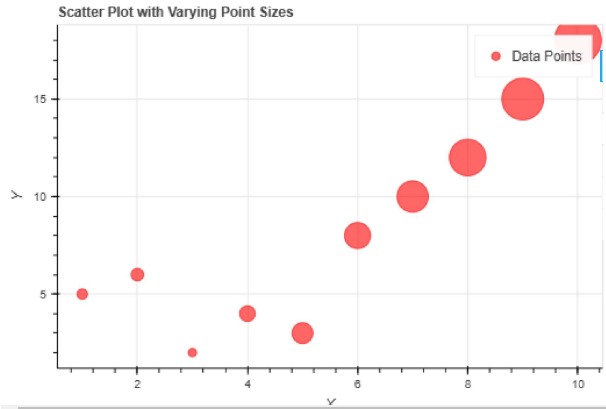
8b) Write a Python program for plotting different types of plots using Bokeh.

```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

# Enable output in the notebook
output_notebook()

# 1. Line Plot
def create_line_plot():
    x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    y1 = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]  # y = 2x
    y2 = [i ** 2 for i in x]  # y = x^2
    p = figure(title="Line Plot with Multiple Lines", x_axis_label='X', y_axis_label='Y', width=600, height=400)
    p.line(x, y1, line_width=2, color="blue", legend_label="y = 2x")
    p.line(x, y2, line_width=2, color="green", legend_label="y = x^2")
    return p

# 2. Bar Chart
def create_bar_chart():
    categories = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
    values1 = [10, 15, 30, 40, 50, 45, 60, 30, 20, 70]
    values2 = [12, 18, 35, 45, 55, 50, 65, 35, 25, 75]
    p = figure(x_range=categories, title="Grouped Bar Chart", width=800, height=400)
    p.vbar(x=categories, top=values1, width=0.4, color="blue", legend_label="Group 1")
    p.vbar(x=categories, top=values2, width=0.4, color="orange", legend_label="Group 2", bottom=0)
    p.xaxis.axis_label = "Categories"
    p.yaxis.axis_label = "Values"
    p.legend.title = "Groups"
    return p

# 3. Scatter Plot
def create_scatter_plot():
    x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    y = [5, 6, 2, 4, 3, 8, 10, 12, 15, 18]
    size = [10, 12, 8, 15, 20, 25, 30, 35, 40, 45]  # Size of each point
    p = figure(title="Scatter Plot with Varying Point Sizes", x_axis_label='X', y_axis_label='Y', width=600, height=400)
    p.scatter(x, y, size=size, color="red", alpha=0.6, legend_label="Data Points")
    return p

# Create the plots
line_plot = create_line_plot()
bar_chart = create_bar_chart()
```

```
scatter_plot = create_scatter_plot()

# Show all the plots
show(line_plot)
show(bar_chart)
show(scatter_plot)
```







9. Program about plotly... lets understand plotly first

Plotly is a powerful and interactive data visualization library for Python (and other programming languages like R, Julia, and JavaScript). It allows you to create a wide variety of interactive, visually appealing, and web-ready charts and plots, including 2D and 3D graphs, and maps.

Key Features of Plotly

- **Interactive Plots**: Plotly plots are interactive by default. You can zoom, pan, hover to view data points, and more, which is particularly useful for data exploration and presentation.
- **Wide Range of Plots**: It supports various types of charts such as:
  - Line plots
  - Bar charts
  - Scatter plots
  - 3D plots (scatter, surface, line, etc.)
  - Heatmaps
  - Geographic maps
  - Candlestick charts
  - And many more.
- **Web-based**: Plotly's interactive plots can be displayed directly in Jupyter Notebooks, web applications, and even saved as HTML files for sharing.
- **Easy to Use**: Plotly integrates seamlessly with Python libraries like Pandas and NumPy, making it easy to create plots directly from data structures like DataFrames or arrays.

- **Customization**: It offers extensive customization options for modifying plot layouts, themes, colors, labels, and more, giving users full control over the appearance of the charts.

9. Write a Python program to draw 3D Plots using Plotly Libraries.

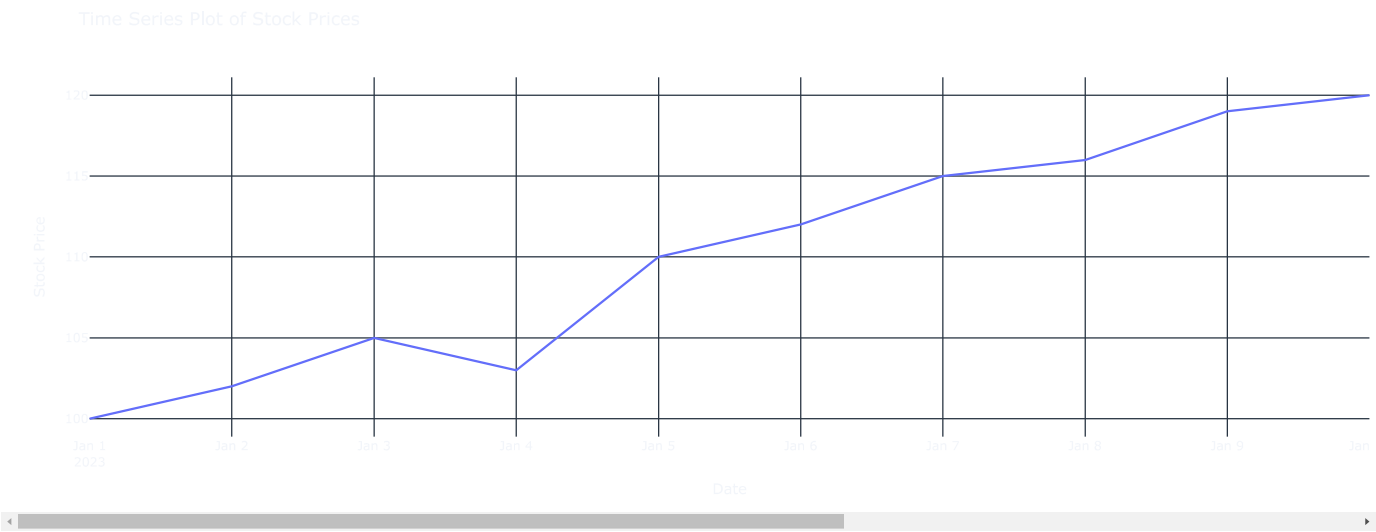
```
import plotly.graph_objects as go
import numpy as np

# 1. 3D Scatter Plot
def create_3d_scatter_plot():
    x = np.random.rand(100)
    y = np.random.rand(100)
    z = np.random.rand(100)

    scatter_plot = go.Figure(data=[go.Scatter3d(x=x, y=y, z=z, mode='markers')])
    scatter_plot.show()

# 2. 3D Surface Plot
def create_3d_surface_plot():
    x = np.linspace(-5, 5, 50)
    y = np.linspace(-5, 5, 50)
    x, y = np.meshgrid(x, y)
    z = np.sin(np.sqrt(x**2 + y**2))

    surface_plot = go.Figure(data=[go.Surface(z=z, x=x, y=y)])
    surface_plot.show()

# 3. 3D Line Plot
def create_3d_line_plot():
    t = np.linspace(0, 10, 100)
    x = np.sin(t)
    y = np.cos(t)
    z = t

    line_plot = go.Figure(data=[go.Scatter3d(x=x, y=y, z=z, mode='lines')])
    line_plot.show()

# Calling the functions to create the plots
create_3d_scatter_plot()
create_3d_surface_plot()
create_3d_line_plot()
```

10. a) Write a Python program to draw Time Series using Plotly Libraries.

10a. version 1 - less data

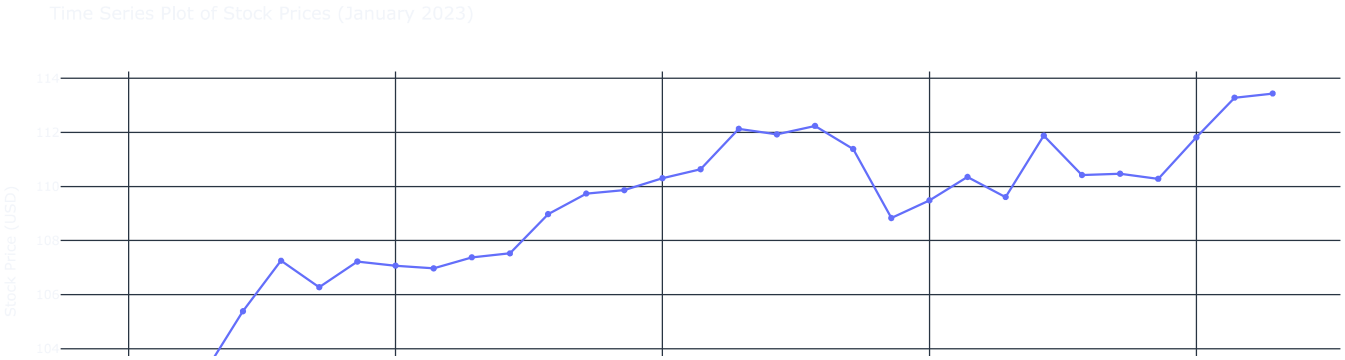
```
# Importing necessary libraries
import plotly.graph_objects as go
import pandas as pd

# Sample time series data
# Create a date range for the x-axis (date)
dates = pd.date_range('2023-01-01', periods=10, freq='D')

# Sample data for the y-axis (some random values, for example, stock prices)
values = [100, 102, 105, 103, 110, 112, 115, 116, 119, 120]
```

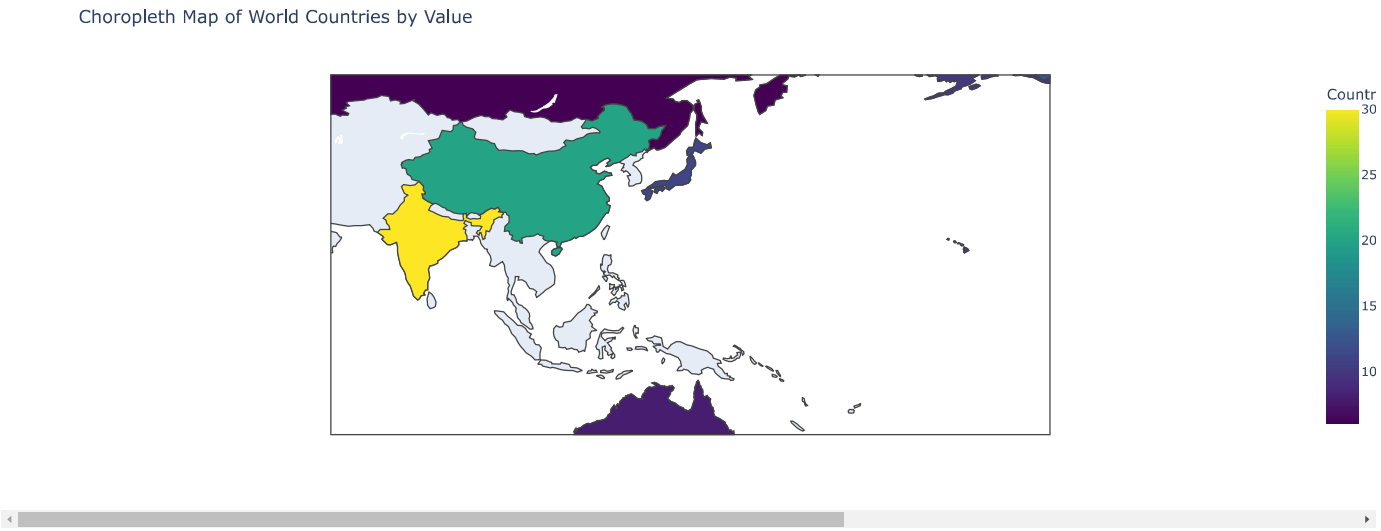
```
# Create a time series plot
fig = go.Figure()

# Add a line plot for the time series data
fig.add_trace(go.Scatter(x=dates, y=values, mode='lines', name='Stock Price'))

# Customize the layout of the plot
fig.update_layout(
    title="Time Series Plot of Stock Prices",
    xaxis_title="Date",
    yaxis_title="Stock Price",
    template="plotly_dark",  # Optional: Use a dark theme
)

# Show the plot
fig.show()
```



10a. version 2 - more data

```
# Importing necessary libraries
import plotly.graph_objects as go
import pandas as pd
import numpy as np

# Generate a larger date range for the x-axis (from Jan 1, 2023 to Jan 31, 2023)
dates = pd.date_range('2023-01-01', periods=31, freq='D')

# Generate more realistic stock price data (random walk with some trend)
np.random.seed(0)
price_changes = np.random.randn(31)  # Random daily changes in stock price
initial_price = 100  # Starting price
prices = initial_price + np.cumsum(price_changes)  # Cumulative sum to simulate random walk

# Create a time series plot
fig = go.Figure()

# Add a line plot for the time series data
fig.add_trace(go.Scatter(x=dates, y=prices, mode='lines+markers', name='Stock Price'))

# Customize the layout of the plot
fig.update_layout(
    title="Time Series Plot of Stock Prices (January 2023)",
    xaxis_title="Date",
    yaxis_title="Stock Price (USD)",
    template="plotly_dark",  # Optional: Use a dark theme
    xaxis=dict(showgrid=True),  # Show grid on x-axis
    yaxis=dict(showgrid=True),  # Show grid on y-axis
)

# Show the plot
fig.show()
```



10 b) Write a Python program for creating Maps using Plotly Libraries

```
import plotly.express as px
import pandas as pd

# Sample Data: Country names and corresponding values
data = {
    'Country': ['United States', 'China', 'India', 'Brazil', 'Australia', 'Germany', 'United Kingdom', 'Russia', 'Canada', 'Japan'],
    'Value': [100, 200, 300, 150, 80, 120, 90, 60, 130, 110]
}

# Convert the data to a DataFrame
df = pd.DataFrame(data)

# Create the Choropleth Map
fig = px.choropleth(df, locations='Country', locationmode='country names', color='Value',
                    color_continuous_scale='Viridis', labels={'Value': 'Country Value'},
                    title='Choropleth Map of World Countries by Value')

# Show the plot
fig.show()
```