Data_Visualization_Using_Python

Ques 1 : Write code for waffle chart displaying the sales of the phone using below data. Also add title, legend and labels.
Data={'phone':['Xiaomi','Samasung','Apple','Nokia','Realme'],
'stock':[44,12,8,5,3] }

```
# Import neccessary libraries.
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pywaffle import Waffle

# Creation of a dataframe
data = {'phone' : ['Xiaomi','Samsung', 'Apple', 'Nokia', 'Realme'],
        'stock' : [44, 12, 8, 5, 3]}
df = pd.DataFrame(data)

# To plot the waffle chart
fig = plt.figure(
    FigureClass=Waffle,
    rows = 5,
    values = df['stock'],
    icons = 'phone', icon_size = 30,
    title={'label': 'Phone Sales Distribution', 'loc': 'center'},
    labels = list(df.phone)
)

plt.show()
```

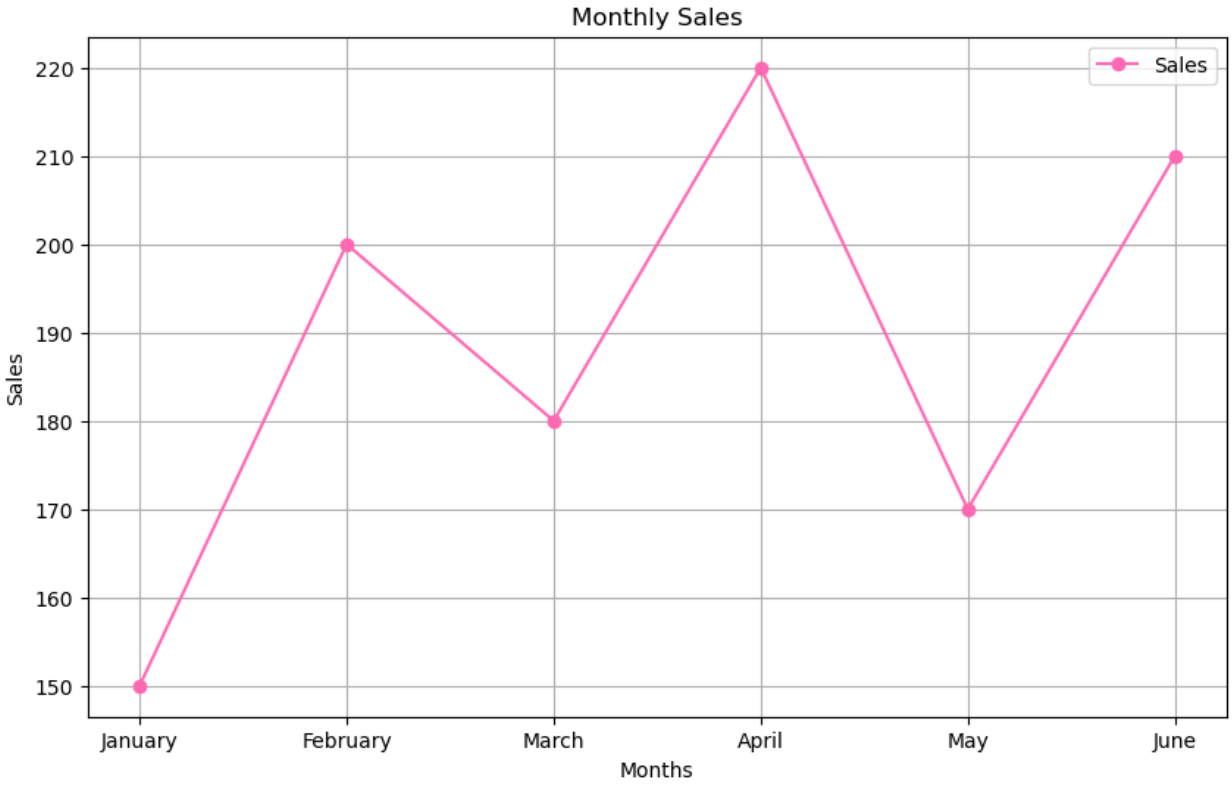
Explanation: 1) Import Libraries: We import the necessary libraries including matplotlib.pyplot, numpy, pandas, and pywaffle.Waffle. 2) Data Preparation: We create a dictionary for the phone sales data and convert it into a pandas DataFrame. 3) Plot the Waffle Chart:

- FigureClass=Waffle specifies the use of the Waffle chart.
- rows=5 defines the number of rows in the chart.
- values=df['stock'] takes the stock values for the chart.
- labels parameter creates labels for each phone with its stock value.
- legend adds a legend to the chart, positioned to the upper left outside the chart.
- title sets the chart title. 4) Display the Plot: plt.show() renders the plot. Make sure you have pywaffle installed. You can install it using pip install pywaffle.

Question 2 :

Create a line plot displaying the sales over the months using below data. Also add title, legend and lables.
Data:
Months=['January','February','March','April','May','June']
Sales= [150,200,180,220,170,210]

```
# Importing neccessary libraries
import matplotlib.pyplot as plt

# Data
months = ['January', 'February', 'March', 'April', 'May', 'June']
sales = [150, 200, 180, 220, 170, 210]

# Plot
plt.figure(figsize=(10, 6))
plt.plot(months, sales, marker='o', linestyle='-', color='hotpink',
label='Sales')
plt.title('Monthly Sales', loc='center')
plt.xlabel('Months')
plt.ylabel('Sales')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



Code Explanation

1. **Import Library**:
   - We import `matplotlib.pyplot` for plotting.
2. **Data**:
   - Define the months and corresponding sales data as two lists: `months` and `sales`.
3. **Plotting**:
   - `plt.figure(figsize=(10, 6))`:
     - Sets the figure size to 10 inches wide and 6 inches tall.
   - `plt.plot(months, sales, marker='o', linestyle='-', color='b', label='Sales')`:
     - Plots the line graph with the following customizations:
       - `marker='o'`: Specifies circular markers at data points.
       - `linestyle='-'`: Sets the line style to solid.
       - `color='b'`: Sets the line color to blue.
       - `label='Sales'`: Provides a label for the line which will appear in the legend.
   - `plt.title('Monthly Sales', loc='center')`:
     - Adds a centered title to the plot.
   - `plt.xlabel('Months')`:
     - Adds a label to the x-axis.
   - `plt.ylabel('Sales')`:
     - Adds a label to the y-axis.

- `plt.legend(loc='upper right')`:
  - Adds a legend to the plot, positioned at the upper right corner.
- `plt.grid(True)`:
  - Adds a grid to the plot for better readability of the data points.

4. **Display**:
   - `plt.show()`:
     - Renders the plot on the screen.

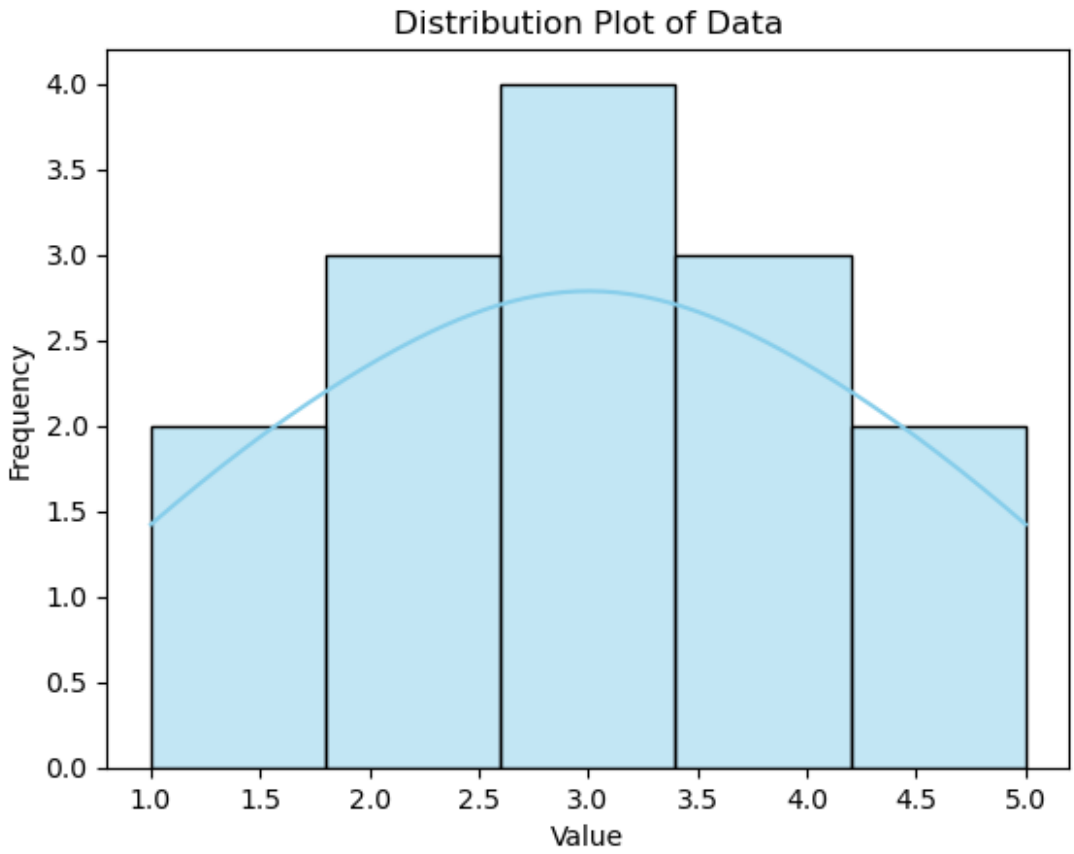
Usage

To run the code, ensure you have `matplotlib` installed. You can install it using pip:

```bash pip install matplotlib

# Question 3

Create a distribution plot for data = [1,1,2,2,2,3,3,3,3,4,4,4,5,5] using seaborn visualization.

```
# Importing neccessary libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Data
data = [1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5]

# Creating a distribution plot using Seaborn
sns.histplot(data, kde = True, color = "skyblue", label="kde")
plt.title('Distribution Plot of Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```
```



Code Explanation

1. **Import Libraries**:
   - We import `seaborn` for creating the distribution plot and `matplotlib.pyplot` for additional plot customizations and rendering.
2. **Data**:
   - The data is defined as a list of values.
3. **Plotting**:
   - `sns.histplot(data, kde=True)`:
     - Creates a histogram of the data with a Kernel Density Estimate (KDE) overlay.
     - `data`: The list of values to plot.
     - `kde=True`: Adds the KDE curve to the plot for visualizing the data's distribution.
   - `plt.title('Distribution Plot of Data')`:
     - Adds a title to the plot.
   - `plt.xlabel('Value')`:
     - Adds a label to the x-axis.
   - `plt.ylabel('Frequency')`:
     - Adds a label to the y-axis.
4. **Display**:

- `plt.show()`:
  - Renders the plot on the screen.

Usage

To run the code, ensure you have `seaborn` and `matplotlib` installed. You can install them using pip:

```bash pip install seaborn matplotlib

# Question 4

Write code for a word cloud using the text = “K.R. Mangalam University, the top university in Gurugram, is committed to advancing ambitious students educational journeys and igniting their passions.”

```
# Importing neccessary libraries
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# Data
text = "K.R. Mangalam University, the top university in Gurugram, is
committed to advancing ambitious students educational journeys and
igniting their passions."

# Generate Word Cloud
wordcloud = WordCloud(width = 800, height = 400, background_color =
'white').generate(text)

# Plot
plt.figure(figsize = (10, 5))
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.title("Word Cloud")
plt.show()
```
```

Code Explanation

1. **Import Libraries**:
   - We import `WordCloud` from the `wordcloud` library for generating the word cloud.
   - We import `matplotlib.pyplot` for rendering the word cloud.
2. **Text**:
   - The provided text is stored in a variable.
3. **Generate Word Cloud**:
   - `wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)`:
     - Generates the word cloud with the specified dimensions and background color.
     - `width=800, height=400`: Sets the size of the word cloud.
     - `background_color='white'`: Sets the background color of the word cloud to white.
     - `.generate(text)`: Generates the word cloud from the provided text.
4. **Plotting**:
   - `plt.figure(figsize=(10, 5))`:
     - Sets the figure size to 10 inches wide and 5 inches tall.
   - `plt.imshow(wordcloud, interpolation='bilinear')`:
     - Displays the word cloud with bilinear interpolation for smoothness.
   - `plt.axis('off')`:
     - Removes the axis from the plot for a cleaner look.
   - `plt.title('Word Cloud of Given Text')`:
     - Adds a title to the plot.

5. **Display**:
   - `plt.show()`:
     - Renders the plot on the screen.

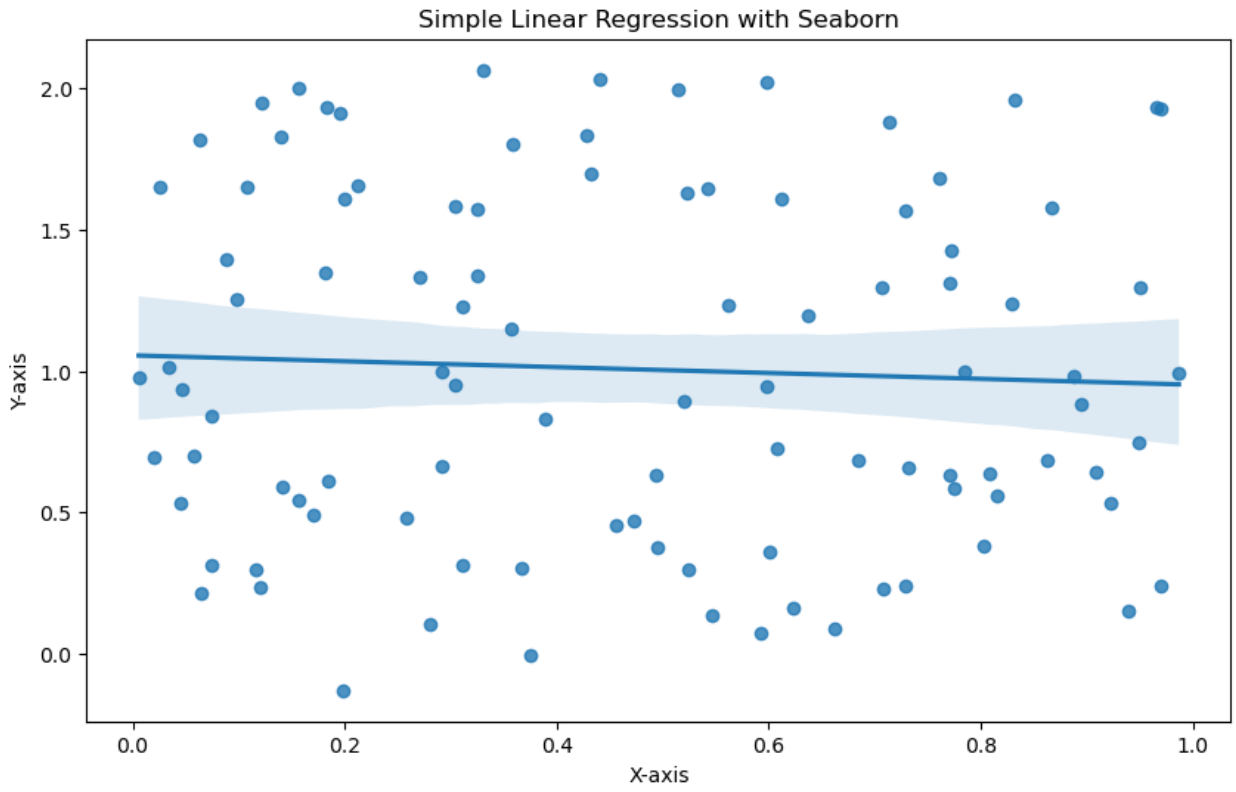
Usage

To run the code, ensure you have the `wordcloud` and `matplotlib` libraries installed. You can install them using pip:

```bash pip install wordcloud matplotlib

# Question 5

Construct a simple linear regression analysis and visualize it using Seaborn. Display the regression line and the confidence interval for the regression estimate.
Data:
df= pd.DataFrame({
"X": np,random.rand(100),
"Y": 2* np.random.rand(100) +
np.random.normal(0,0.1,100)
)}

```
# Importing neccessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Data
np.random.seed(42)
df = pd.DataFrame({
    "X": np.random.rand(100),
    "Y": 2 * np.random.rand(100) + np.random.normal(0, 0.1, 100)
})

# Plot
plt.figure(figsize = (10, 6))
sns.regplot(x = "X", y = "Y", data = df, ci = 95)
plt.title("Simple Linear Regression with Seaborn")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```
```



Code Explanation

1. **Import Libraries**:
   - `numpy` for generating random data.
   - `pandas` for creating the DataFrame.
   - `seaborn` for creating the regression plot.
   - `matplotlib.pyplot` for additional plot customizations and rendering.
2. **Generate Data**:
   - `np.random.seed(42)` sets the seed for reproducibility.
   - `X` is generated using `np.random.rand(100)` to create 100 random values between 0 and 1.
   - `Y` is generated using `2 * np.random.rand(100) + np.random.normal(0, 0.1, 100)` to create values based on a simple linear relationship with some added noise from a normal distribution with mean 0 and standard deviation 0.1.
   - The data is stored in a pandas DataFrame `df`.
3. **Plotting**:
   - `plt.figure(figsize=(10, 6))` sets the figure size to 10 inches wide and 6 inches tall.
   - `sns.regplot(x='X', y='Y', data=df, ci=95)`:
     - Creates a regression plot with `X` as the independent variable and `Y` as the dependent variable.

        - `ci=95` specifies the confidence interval for the regression estimate (95% confidence interval).
    - `plt.title('Simple Linear Regression with Seaborn')` adds a title to the plot.
    - `plt.xlabel('X')` and `plt.ylabel('Y')` add labels to the x-axis and y-axis, respectively.
4. **Display**:
    - `plt.show()` renders the plot on the screen.

Usage

To run the code, ensure you have the required libraries (`numpy`, `pandas`, `seaborn`, `matplotlib`) installed. You can install them using pip:

```bash pip install numpy pandas seaborn matplotlib

# Question 6

Develop a scatter chart with below x and y. Enhance the scatter chart by converting it into bubble chart. Customize the opacity and color of the bubbles . Add appropriate labels , titles, and a legend.
Data:
X= [5,7,8,7,2,17,2,9,4,11,12,9,6]
Y= [99,86,87,88,100,86,103,87,94,78,77,85,86]
Z = [210,320,180,260,300,230,120,160,300,410,200,220,190]

```
# Importing neccessary libraries
import matplotlib.pyplot as plt

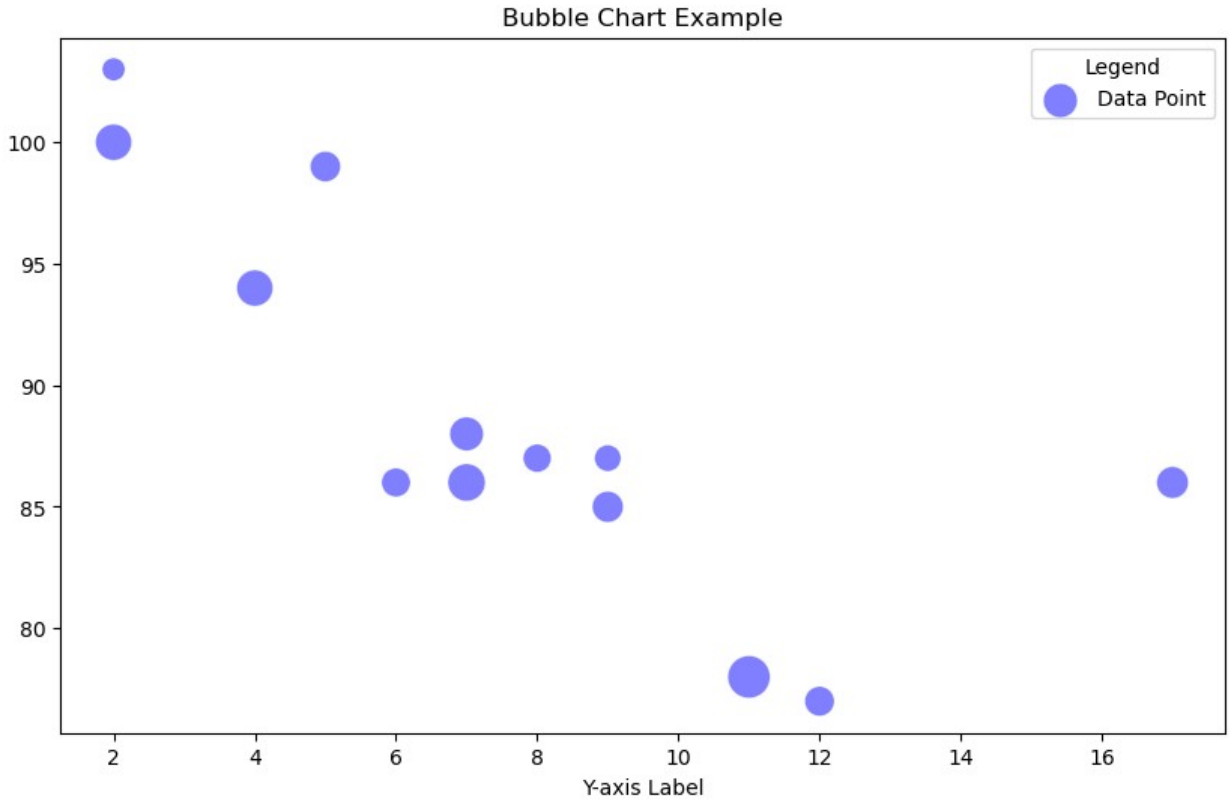
# Data
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
y = [99, 86, 87, 88, 100, 86, 103, 87, 94, 78, 77, 85, 86]
z = [210, 320, 180, 260, 300, 230, 120, 160, 300, 410, 200, 220, 190]

# Creating a scatter plot
plt.figure(figsize = (10, 6))
scatter = plt.scatter(x, y, s=z,label = "Data Point", c="blue", alpha
= 0.5, edgecolor = 'w', linewidth=0.5)

# Titles and lebels
plt.title('Bubble Chart Example')
plt.xlabel('X-axis Label')
plt.xlabel('Y-axis Label')

# Legend
plt.legend(title='Legend', loc='upper right')
```
```

```
# Show Plot
plt.show()
```



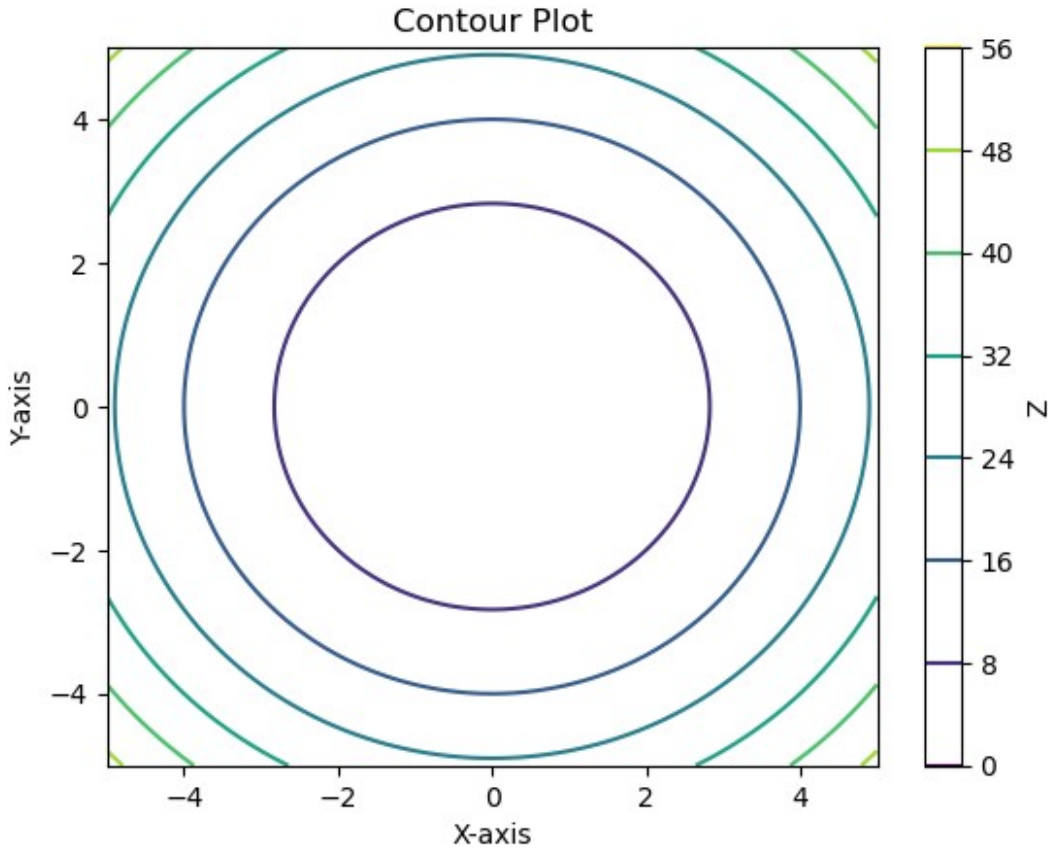
Question 7

Create a contour plot for the function Z = X ** 2 + Y ** 2 where X,Y are created using meshgrid(x,y) for the following points:
x= np.linspace(-5,5,100

y= np.linspace(-5,5,100) 0)

```
# Generate linearly spaced values for x and y
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)

# Create a mesh grid
X, Y = np.meshgrid(x, y)

# Define the function Z = np.sin(X) + np.cos(Y)
Z = X**2 + Y**2

# Create a contour plot
contour_plot = plt.contour(X, Y, Z)
```

```
# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Contour Plot")

# Add a color bar
plt.colorbar(contour_plot, label="Z")

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



Code Explanation

1. **Importing Necessary Libraries**:
   - `matplotlib.pyplot` is used for creating static, interactive, and animated visualizations in Python.
   - `numpy` is a fundamental package for scientific computing with Python, used here to generate arrays.
2. **Generating Linearly Spaced Values for x and y**:

- `np.linspace(start, stop, num)` generates `num` evenly spaced values between `start` and `stop`.
- Here, 100 values are generated between -5 and 5 for both `x` and `y`.

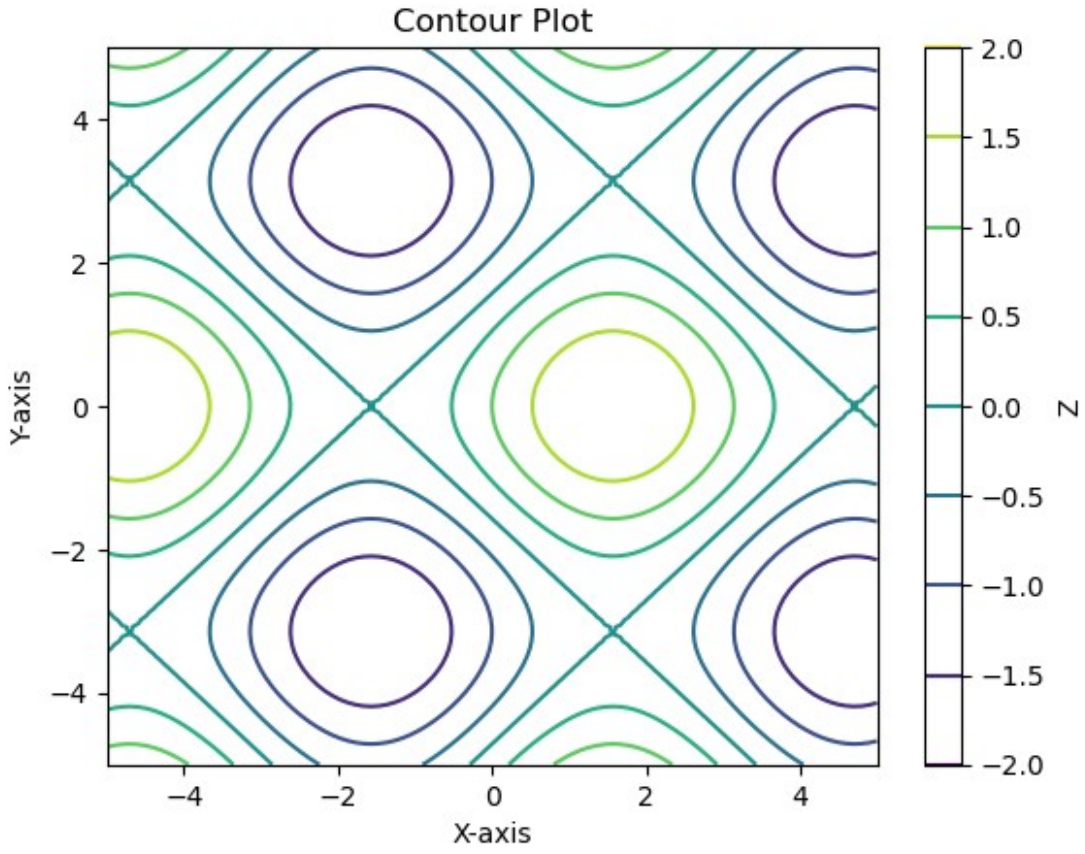
3. **Creating a Mesh Grid**:
   - `np.meshgrid` creates a rectangular grid out of two given one-dimensional arrays representing the Cartesian coordinates `x` and `y`.
   - `X` and `Y` are the coordinate matrices returned by `np.meshgrid`, which are used to evaluate the function on a grid.
4. **Defining the Function**:
   - `Z = X**2 + Y**2` computes the value of the function for each point on the grid.
5. **Creating a Contour Plot**:
   - `plt.contour(X, Y, Z)` creates a contour plot of the function values `Z` on the grid defined by `X` and `Y`.
6. **Adding Labels and Title**:
   - `plt.xlabel("X-axis")` labels the x-axis.
   - `plt.ylabel("Y-axis")` labels the y-axis.
   - `plt.title("Contour Plot")` adds a title to the plot.
7. **Adding a Color Bar**:
   - `plt.colorbar(contour_plot, label="Z")` adds a color bar to the plot, which indicates the values of `Z`.
8. **Displaying the Plot**:
   - `plt.show()` renders the plot in a window.

Question 8

Create a contour plot for the function **Z = np.sin(X) + np.cos(Y)** where X,Y are created using meshgrid(x,y) for the following points:

```
x= np.linspace(-5,5,100)
y= np.linspace(-5,5,100)

import matplotlib.pyplot as plt
import numpy as np

# Generate linearly spaced values for x and y
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)

# Create a mesh grid
X, Y = np.meshgrid(x, y)

# Define the function Z = np.sin(X) + np.cos(Y)
Z = np.sin(X) + np.cos(Y)
```

```
# Create a contour plot
contour_plot = plt.contour(X, Y, Z)

# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Contour Plot")

# Add a color bar
plt.colorbar(contour_plot, label="Z")

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



Explanation

1. **Importing Necessary Libraries**:

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

   - `matplotlib.pyplot` is used for creating visualizations.
   - `numpy` is used for numerical operations and array handling.

2. **Generating Linearly Spaced Values for x and y**:

```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
```

   - `np.linspace` generates 100 evenly spaced values between -5 and 5 for both `x` and `y`.

3. **Creating a Mesh Grid**:

```
X, Y = np.meshgrid(x, y)
```

   - `np.meshgrid` creates two 2D arrays, `X` and `Y`, which contain the x and y coordinates for each point in the grid.

4. **Defining the Function**:

```
Z = np.sin(X) + np.cos(Y)
```

   - Calculates the value of the function (Z = \sin(X) + \cos(Y)) for each point on the grid.

5. **Creating a Contour Plot**:

```
contour_plot = plt.contour(X, Y, Z)
```

   - `plt.contour` generates the contour plot for the function `Z` over the grid defined by `X` and `Y`.

6. **Adding Labels and Title**:

```
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Contour Plot")
```

   - Adds labels to the x-axis and y-axis, and a title to the plot.

7. **Adding a Color Bar**:

```
plt.colorbar(contour_plot, label="Z")
```

   - Adds a color bar to the plot to represent the values of `Z`.

8. **Displaying the Plot**:

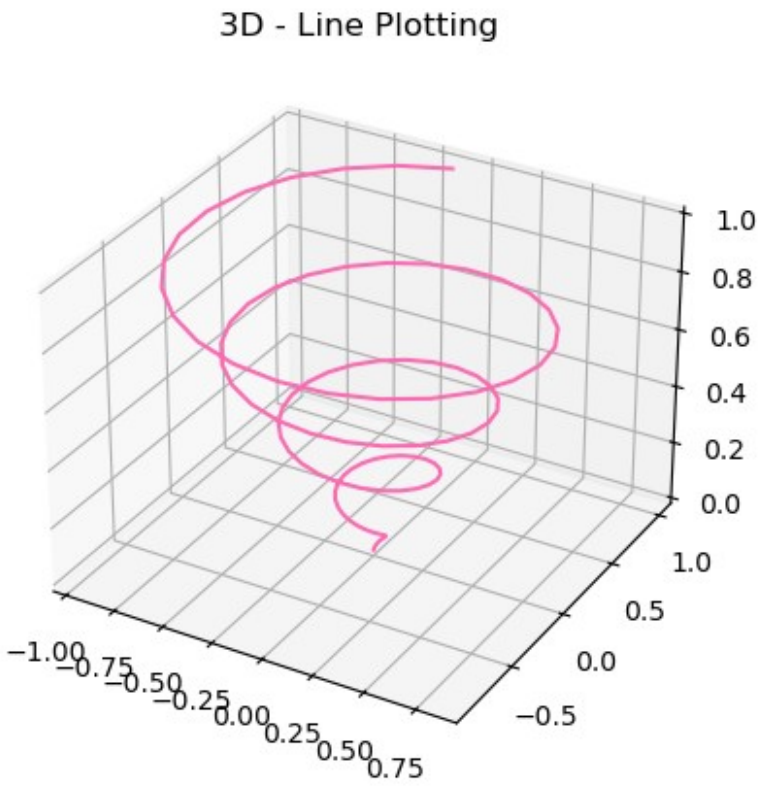
```
plt.show()
```

   - Renders and displays the plot.

Question 9

Create visulaization and write code for 3D line plotting using ```bash data:
z= np.linspace(0, 1, 100)
x= z*np.sin(25*z)
y= z*np.cos(25*z)

```
# Importing neccessary libraries.
import matplotlib.pyplot as plt
import numpy as np

# Create a 3D axis
ax = plt.axes(projection="3d")

# Generate data
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)

# Plot the 3D line
ax.plot3D(x, y, z, "hotpink")

# Add a title
ax.set_title("3D - Line Plotting")

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



Explanation

1. **Importing Necessary Libraries**:

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

   - `matplotlib.pyplot` is used for creating visualizations.
   - `numpy` is used for numerical operations and array handling.

2. **Creating a 3D Axis**:

```
ax = plt.axes(projection="3d")
```

   - `plt.axes(projection="3d")` creates a 3D axis for the plot.

3. **Generating Data**:

```
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
```

   - `z = np.linspace(0, 1, 100)` generates 100 evenly spaced values between 0 and 1.
   - `x = z * np.sin(25 * z)` calculates `z` times the sine of 25 times each value in `z`.

- `y = z * np.cos(25 * z)` calculates `z` times the cosine of 25 times each value in `z`.

4. **Plotting the 3D Line**:

```
ax.plot3D(x, y, z, "hotpink")
```

   - `ax.plot3D(x, y, z, "hotpink")` plots the 3D line using the `x`, `y`, and `z` data with the color "hotpink".

5. **Adding a Title**:

```
ax.set_title("3D - Line Plotting")
```

   - `ax.set_title("3D - Line Plotting")` adds a title to the plot.

6. **Displaying the Plot**:

```
plt.show()
```

   - `plt.show()` renders and displays the plot.

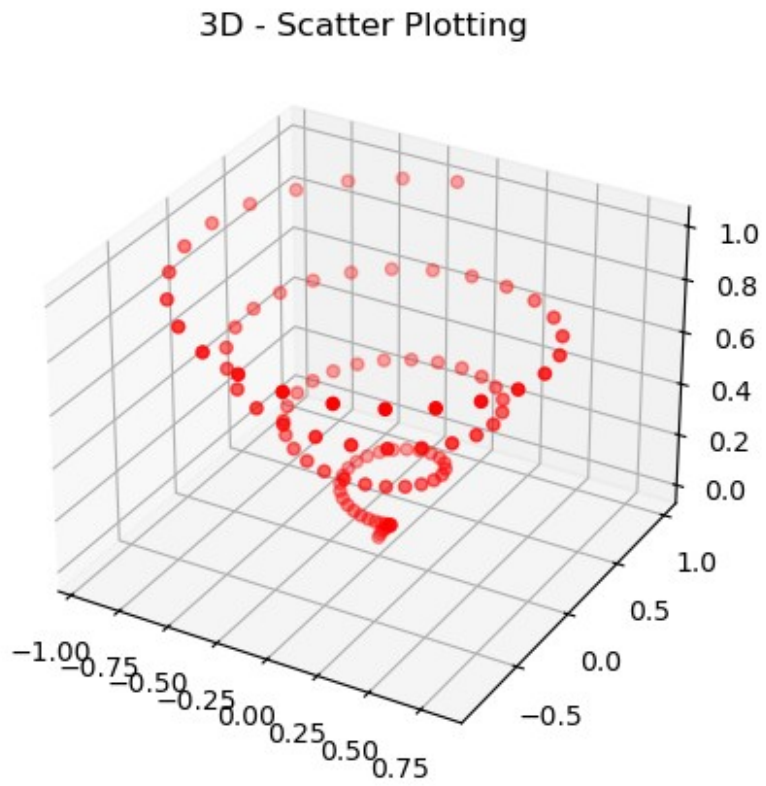
This code effectively creates a 3D line plot with the specified data and settings.

Question 10

create visulaization and write code for 3D Scatter plotting using

```
data:
z= np.linspace(0, 1, 100)
x= z*np.sin(25*z)
y= z*np.cos(25*z)

# Importing neccessary libraries
import matplotlib.pyplot as plt
import numpy as np

# Create a 3D axis
ax = plt.axes(projection='3d')

# Generate data
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)

# Plot the 3D scatter
ax.scatter(x, y, z, c='red')

# Add a title
ax.set_title("3D - Scatter Plotting")
```

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



Explanation

1. **Importing Necessary Libraries**:

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

   - `matplotlib.pyplot` is used for creating visualizations.
   - `numpy` is used for numerical operations and array handling.

2. **Creating a 3D Axis**:

```
ax = plt.axes(projection='3d')
```

   - `plt.axes(projection='3d')` creates a 3D axis for the plot.

3. **Generating Data**:

```
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
```

- `z = np.linspace(0, 1, 100)` generates 100 evenly spaced values between 0 and 1.
- `x = z * np.sin(25 * z)` calculates `z` times the sine of 25 times each value in `z`.
- `y = z * np.cos(25 * z)` calculates `z` times the cosine of 25 times each value in `z`.

4. **Plotting the 3D Scatter**:

```
ax.scatter(x, y, z, c='red')
```

   - `ax.scatter(x, y, z, c='red')` plots the 3D scatter using the `x`, `y`, and `z` data with the color "red".

5. **Adding a Title**:

```
ax.set_title("3D - Scatter Plotting")
```

   - `ax.set_title("3D - Scatter Plotting")` adds a title to the plot.

6. **Displaying the Plot**:

```
plt.show()
```

   - `plt.show()` renders and displays the plot.

This code effectively creates a 3D scatter plot with the specified data and settings.

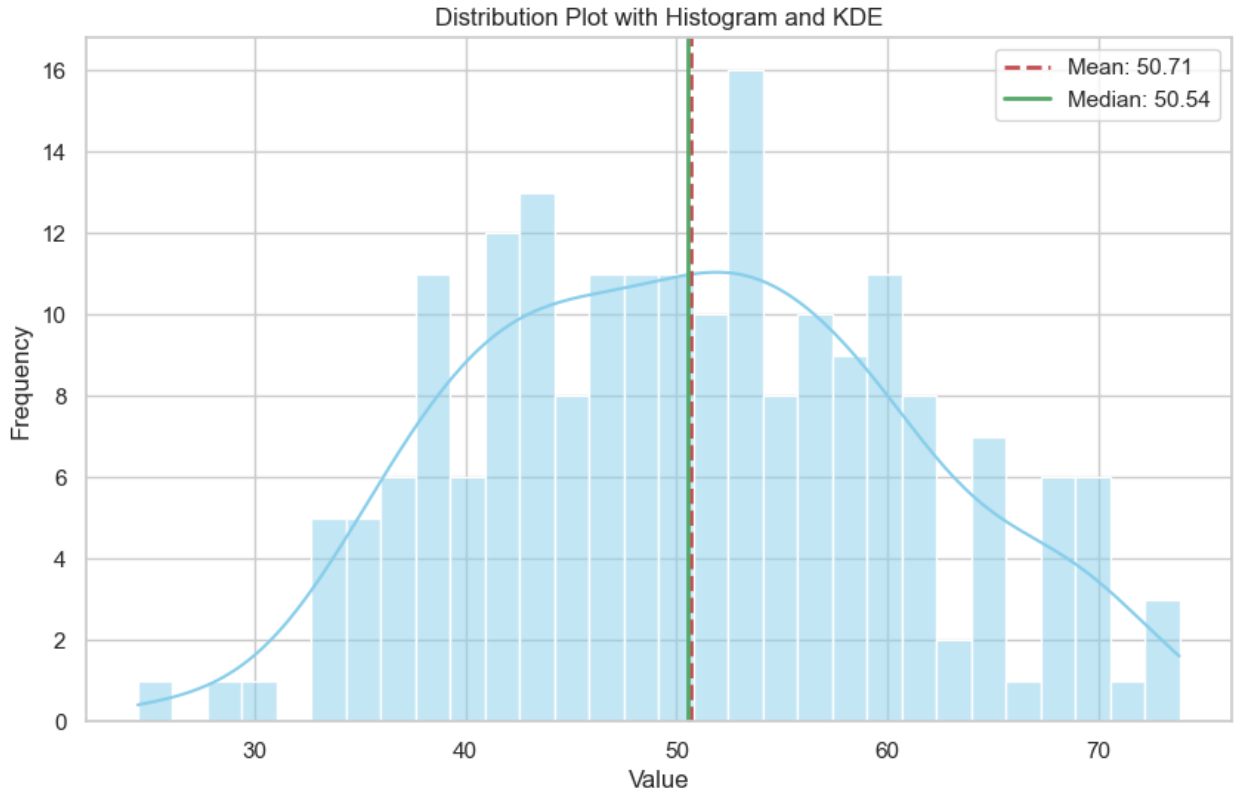
Question 11

Create and write code for distribution plot showing the histogram and KDE (kernal density estimation) using below data. Overlay the mean and median of the dataset on the plot. Customize the plot with seaborn style and colors. Briefly descibe what the plot reveals about the dataset distribution.

```
Data:
import numpy as np
data= np.random.normal(loc=50, scale=10, size=200)

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

# Generate data
np.random.seed(0)  # For reproducibility
data = np.random.normal(loc=50, scale=10, size=200)

# Set seaborn style
sns.set(style="whitegrid")
```

```
# Create the plot
plt.figure(figsize=(10, 6))
sns.histplot(data, kde=True, color="skyblue", bins=30)

# Calculate and plot mean and median
mean = np.mean(data)
median = np.median(data)
plt.axvline(mean, color='r', linestyle='--', linewidth=2,
label=f'Mean: {mean:.2f}')
plt.axvline(median, color='g', linestyle='-', linewidth=2,
label=f'Median: {median:.2f}')

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Distribution Plot with Histogram and KDE')
plt.legend()

# Show plot
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Explanation

1. **Importing Necessary Libraries**:

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

   - `numpy` is used for numerical operations and generating the dataset.
   - `matplotlib.pyplot` is used for creating visualizations.
   - `seaborn` is used for creating attractive and informative statistical graphics.

2. **Generating Data**:

```
np.random.seed(0)  # For reproducibility
data = np.random.normal(loc=50, scale=10, size=200)
```

   - `np.random.seed(0)` ensures reproducibility of the random data.
   - `data = np.random.normal(loc=50, scale=10, size=200)` generates 200 random values from a normal distribution with a mean of 50 and a standard deviation of 10.

3. **Setting Seaborn Style**:

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

- `sns.set(style="whitegrid")` sets the style of the plot to "whitegrid" for better aesthetics.

4. **Creating the Plot**:

```
plt.figure(figsize=(10, 6))
sns.histplot(data, kde=True, color="skyblue", bins=30)
```

- `plt.figure(figsize=(10, 6))` creates a new figure with specified size.
- `sns.histplot(data, kde=True, color="skyblue", bins=30)` creates a histogram and KDE plot with 30 bins and a sky blue color.

5. **Calculating and Plotting Mean and Median**:

```
mean = np.mean(data)
median = np.median(data)
plt.axvline(mean, color='r', linestyle='--', linewidth=2,
label=f'Mean: {mean:.2f}')
plt.axvline(median, color='g', linestyle='-', linewidth=2,
label=f'Median: {median:.2f}')
```

- `mean = np.mean(data)` calculates the mean of the data.
- `median = np.median(data)` calculates the median of the data.
- `plt.axvline` plots vertical lines for the mean (in red, dashed) and median (in green, solid) with labels.

6. **Adding Labels and Title**:

```
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Distribution Plot with Histogram and KDE')
plt.legend()
```

- `plt.xlabel('Value')` labels the x-axis.
- `plt.ylabel('Frequency')` labels the y-axis.
- `plt.title('Distribution Plot with Histogram and KDE')` adds a title to the plot.
- `plt.legend()` adds a legend to the plot.

7. **Displaying the Plot**:

```
plt.show()
```

- `plt.show()` renders and displays the plot.

Plot Description

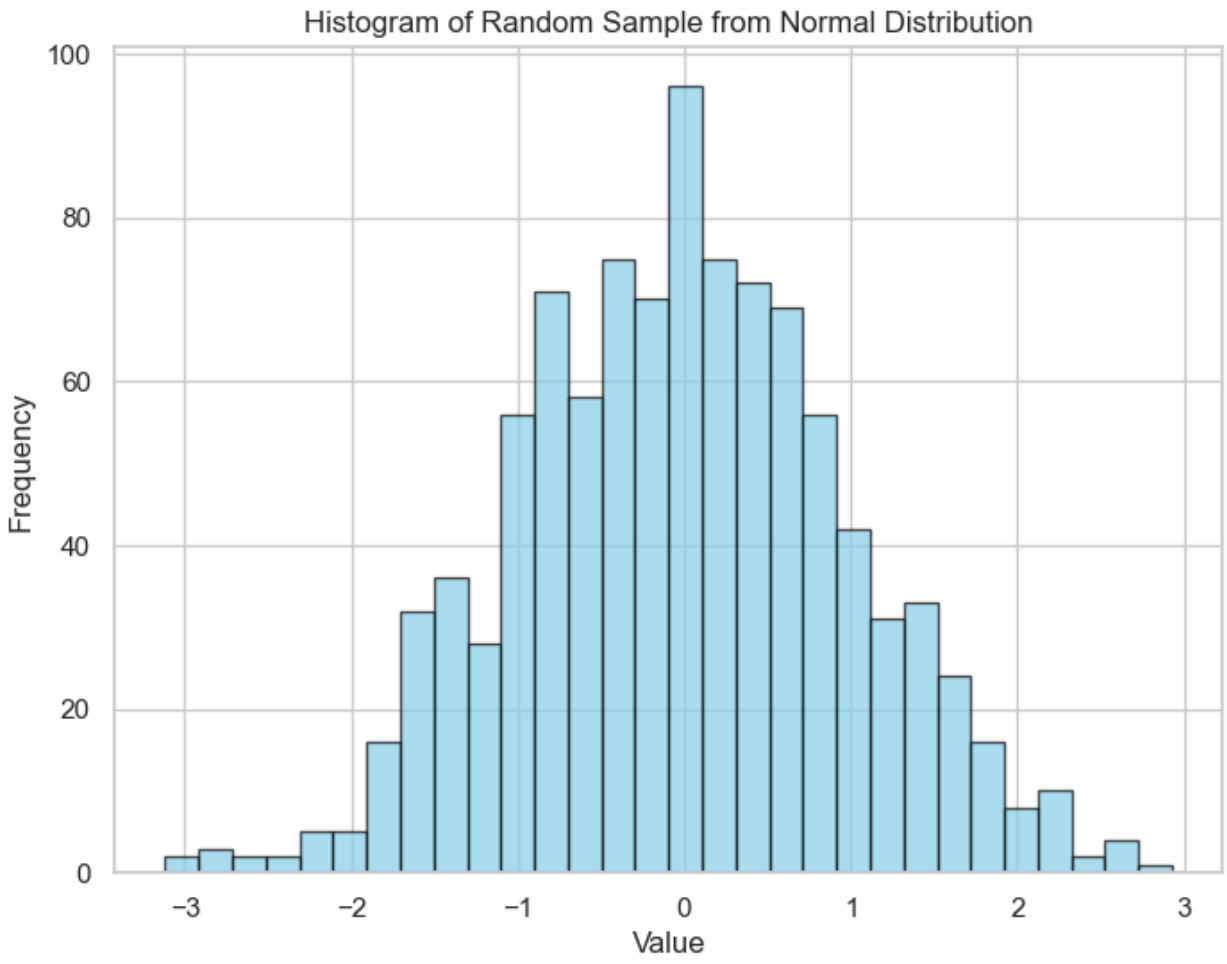
The distribution plot with both histogram and KDE reveals the following about the dataset:

- The dataset is approximately normally distributed, centered around the mean value of 50.
- The histogram shows the frequency distribution of the data values, while the KDE provides a smooth estimate of the probability density function.
- The mean and median are very close to each other, indicating a symmetric distribution with no significant skewness.

Question 12

Develop code and create histogram for random sample from normal distribution using

```
data: m, sigma= 0, 1
data= np.random.normal(m, sigma, 1000)

import matplotlib.pyplot as plt
import numpy as np

# Generate data
m, sigma = 0, 1
data = np.random.normal(m, sigma, 1000)

# Create the histogram
plt.figure(figsize=(8, 6))
plt.hist(data, bins=30, color='skyblue', edgecolor='black', alpha=0.7)

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Random Sample from Normal Distribution')

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



Explanation

1. **Importing Necessary Libraries**:

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

   - `matplotlib.pyplot` is used for creating visualizations.
   - `numpy` is used for numerical operations and generating the dataset.

2. **Generating Data**:

```
m, sigma = 0, 1
data = np.random.normal(m, sigma, 1000)
```

   - `m, sigma = 0, 1` sets the mean (`m`) and standard deviation (`sigma`) for the normal distribution.
   - `np.random.normal(m, sigma, 1000)` generates 1000 random values from a normal distribution with mean `m` and standard deviation `sigma`.

3. **Creating the Histogram**:

```
plt.figure(figsize=(8, 6))
plt.hist(data, bins=30, color='skyblue', edgecolor='black',
alpha=0.7)
```

- `plt.figure(figsize=(8, 6))` creates a new figure with specified size.
- `plt.hist(data, bins=30, color='skyblue', edgecolor='black', alpha=0.7)` plots a histogram of the `data` with 30 bins, sky blue color for bars, black edges, and 70% transparency (`alpha=0.7`).

4. **Adding Labels and Title**:

```
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Random Sample from Normal Distribution')
```

- `plt.xlabel('Value')` labels the x-axis.
- `plt.ylabel('Frequency')` labels the y-axis.
- `plt.title('Histogram of Random Sample from Normal Distribution')` adds a title to the plot.

5. **Displaying the Plot**:

```
plt.show()
```

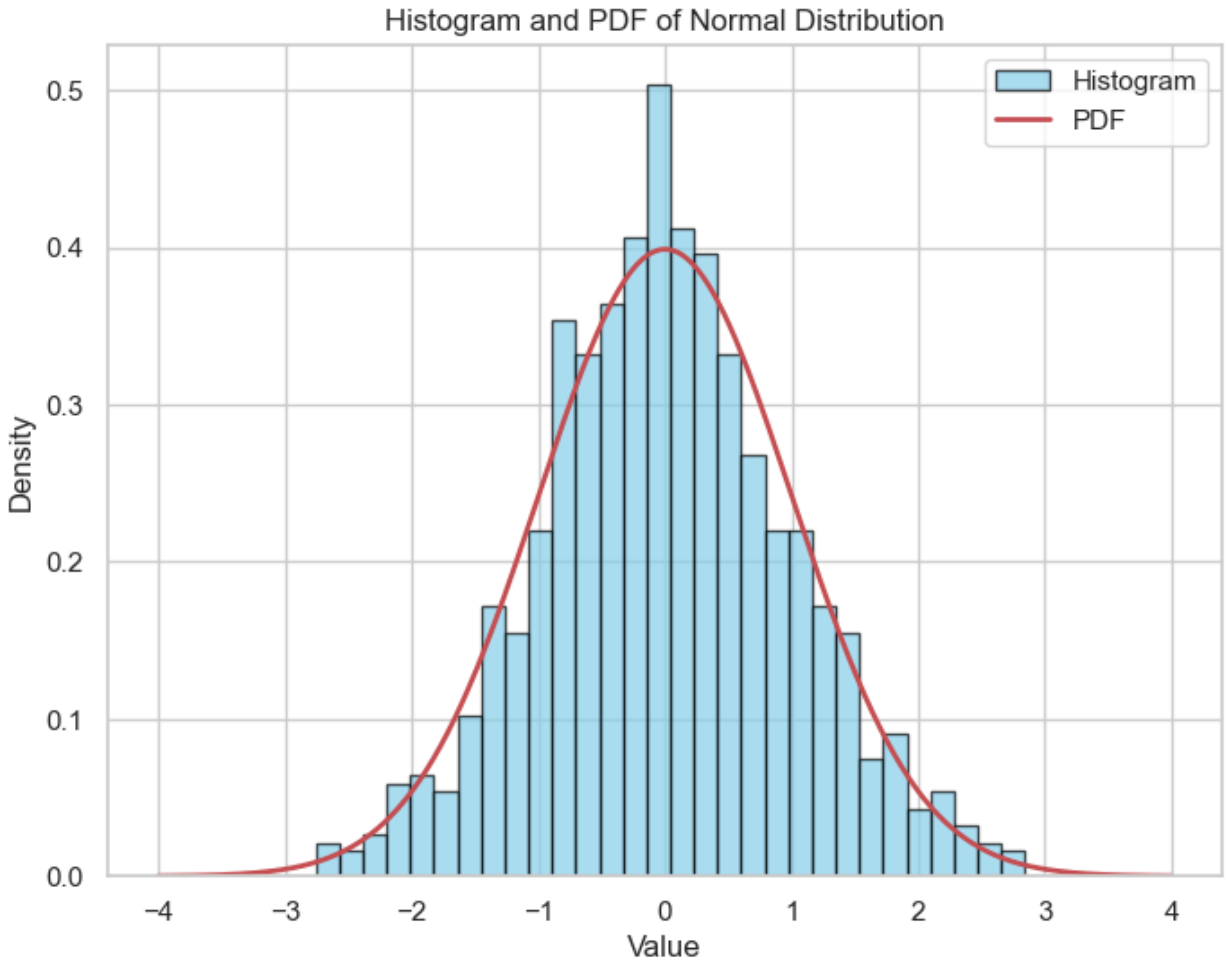
- `plt.show()` renders and displays the plot.

Question 13

Develop code and create histogram for pdf for the normal distribution using

```
data :
m, sigma=0, 1
data=np.random.normal(m, sigma, 1000)

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm

# Generate data
m, sigma = 0, 1
data = np.random.normal(m, sigma, 1000)

# Create an array of values for x-axis based on the distribution of
data
x = np.linspace(-4, 4, 1000)  # Adjust range as needed for your
distribution

# Calculate the PDF values for the normal distribution
```

```
pdf_values = norm.pdf(x, m, sigma)

# Create the histogram for the PDF
plt.figure(figsize=(8, 6))
plt.hist(data, bins=30, density=True, color='skyblue',
edgecolor='black', alpha=0.7, label='Histogram')

# Plot the PDF curve
plt.plot(x, pdf_values, 'r-', lw=2, label='PDF')

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Histogram and PDF of Normal Distribution')
plt.legend()

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



Explanation

1. **Importing Necessary Libraries**:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
```

   - `matplotlib.pyplot` is used for creating visualizations.
   - `numpy` is used for numerical operations and array handling.
   - `scipy.stats.norm` is used to access the normal distribution and its probability density function (`pdf`).

2. **Generating Data**:

```
m, sigma = 0, 1
data = np.random.normal(m, sigma, 1000)
```

   - `m, sigma = 0, 1` sets the mean (`m`) and standard deviation (`sigma`) for the normal distribution.
   - `np.random.normal(m, sigma, 1000)` generates 1000 random values from a normal distribution with mean `m` and standard deviation `sigma`.

3. **Creating an Array for x-axis Values**:

```
x = np.linspace(-4, 4, 1000)
```

   - `np.linspace(-4, 4, 1000)` generates 1000 evenly spaced values from -4 to 4. Adjust the range (`-4, 4`) and the number of points (`1000`) based on the distribution and visualization requirements.

4. **Calculating PDF Values**:

```
pdf_values = norm.pdf(x, m, sigma)
```

   - `norm.pdf(x, m, sigma)` calculates the probability density function (PDF) values for the normal distribution with mean `m` and standard deviation `sigma` at each value of `x`.

5. **Creating the Histogram for PDF**:

```
plt.hist(data, bins=30, density=True, color='skyblue',
edgecolor='black', alpha=0.7, label='Histogram')
```

   - `plt.hist(data, bins=30, density=True, color='skyblue', edgecolor='black', alpha=0.7, label='Histogram')` plots a histogram of `data` with 30 bins, normalized (`density=True`) to show density rather than frequency, using sky blue bars with black edges and 70% transparency (`alpha=0.7`).

6. **Plotting the PDF Curve**:

```
    plt.plot(x, pdf_values, 'r-', lw=2, label='PDF')
```

- `plt.plot(x, pdf_values, 'r-', lw=2, label='PDF')` plots the PDF curve (`pdf_values`) against `x` with a red solid line (`'r-'`) and a line width of 2 (`lw=2`).

7. **Adding Labels and Title**:

```
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.title('Histogram and PDF of Normal Distribution')
    plt.legend()
```

- `plt.xlabel('Value')` labels the x-axis.
- `plt.ylabel('Density')` labels the y-axis.
- `plt.title('Histogram and PDF of Normal Distribution')` adds a title to the plot.
- `plt.legend()` adds a legend to the plot to differentiate between the histogram and the PDF curve.

8. **Displaying the Plot**:

```
    plt.show()
```

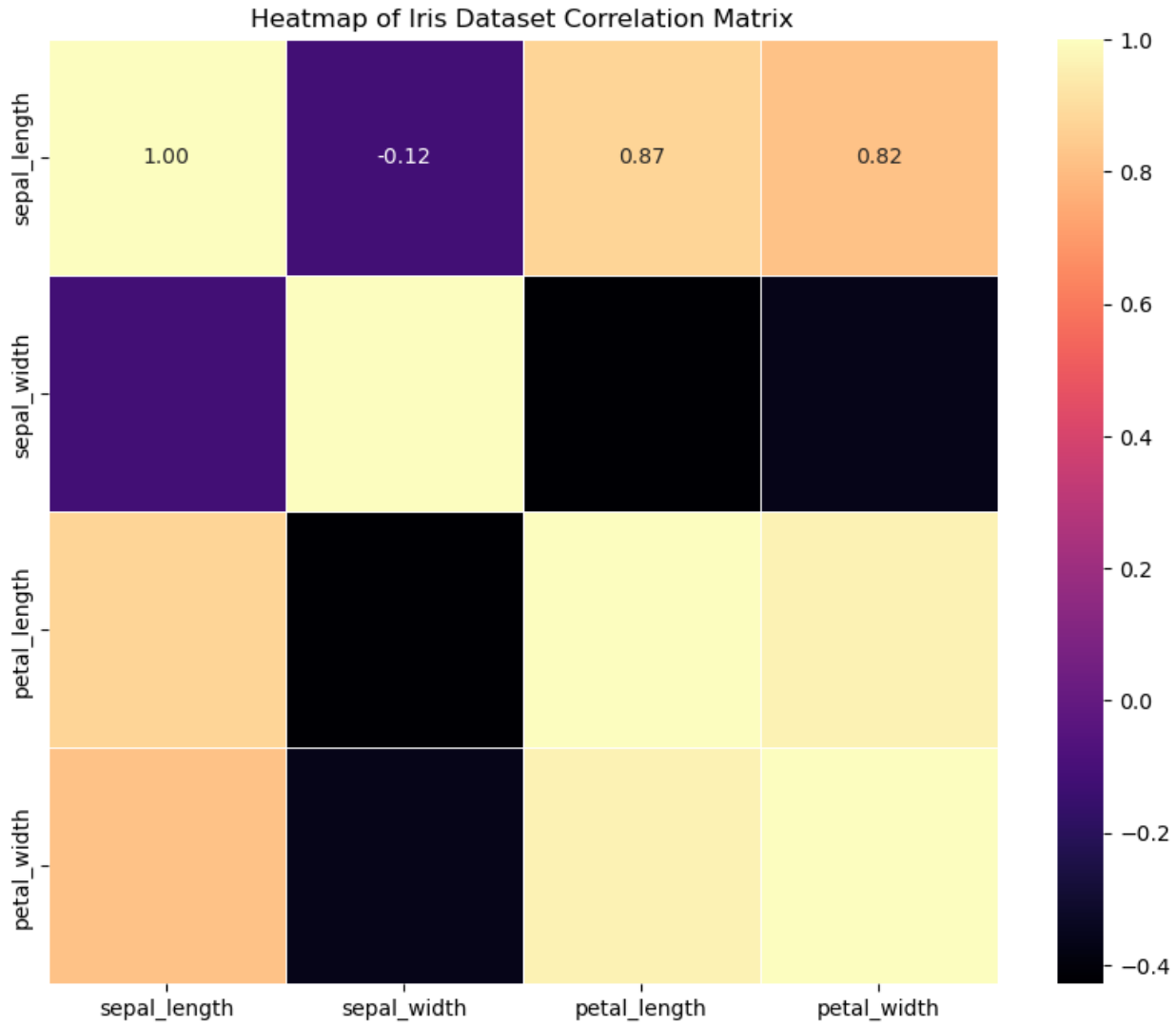
- `plt.show()` renders and displays the plot.

Question 14

Develop code and create heatmap for iris dataset using seaborn. Also add annotation and cmap.

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

# Load the Iris dataset
data = sns.load_dataset('iris')

# Compute the correlation matrix, excluding the non-numeric 'species'
column
c = data.drop(columns=['species']).corr()

# Create the heatmap with annotations and a color map
plt.figure(figsize=(10, 8))
sns.heatmap(c, annot=True, cmap='magma', fmt='.2f', linewidths=.5)

# Add title
plt.title('Heatmap of Iris Dataset Correlation Matrix')
```

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



Explanation

1. **Importing Necessary Libraries**:

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

   - `matplotlib.pyplot` is used for creating visualizations.
   - `seaborn` is used for creating statistical graphics.
   - `pandas` is used for data manipulation and analysis.

2. **Loading the Iris Dataset**:

```
data = sns.load_dataset('iris')
```

- `sns.load_dataset('iris')` loads the Iris dataset directly from Seaborn's dataset library. This dataset includes 150 samples of iris flowers, with features such as sepal length, sepal width, petal length, petal width, and the species of the flower.

3. **Computing the Correlation Matrix**:

```
c = data.drop(columns=['species']).corr()
```

- `data.drop(columns=['species'])` removes the `species` column from the DataFrame because it contains non-numeric data (strings) that cannot be used in the correlation calculation.
- `.corr()` computes the correlation matrix of the remaining numeric columns. The correlation matrix shows the pairwise correlation coefficients between the features.

4. **Creating the Heatmap**:

```
plt.figure(figsize=(10, 8))
sns.heatmap(c, annot=True, cmap='magma', fmt='.2f',
linewidths=.5)
```

- `plt.figure(figsize=(10, 8))` creates a new figure with specified size (10 inches by 8 inches).
- `sns.heatmap(c, annot=True, cmap='magma', fmt='.2f', linewidths=.5)` creates a heatmap of the correlation matrix `c` with:
  - `annot=True` to show the correlation values on the heatmap.
  - `cmap='magma'` to use the "magma" colormap for better visualization.
  - `fmt='.2f'` to format the annotations to two decimal places.
  - `linewidths=.5` to add lines between the cells for better readability.

5. **Adding Title**:

```
plt.title('Heatmap of Iris Dataset Correlation Matrix')
```

- `plt.title('Heatmap of Iris Dataset Correlation Matrix')` adds a title to the plot to describe what the heatmap represents.

6. **Displaying the Plot**:

```
plt.show()
```

- `plt.show()` renders and displays the plot.

Question 15

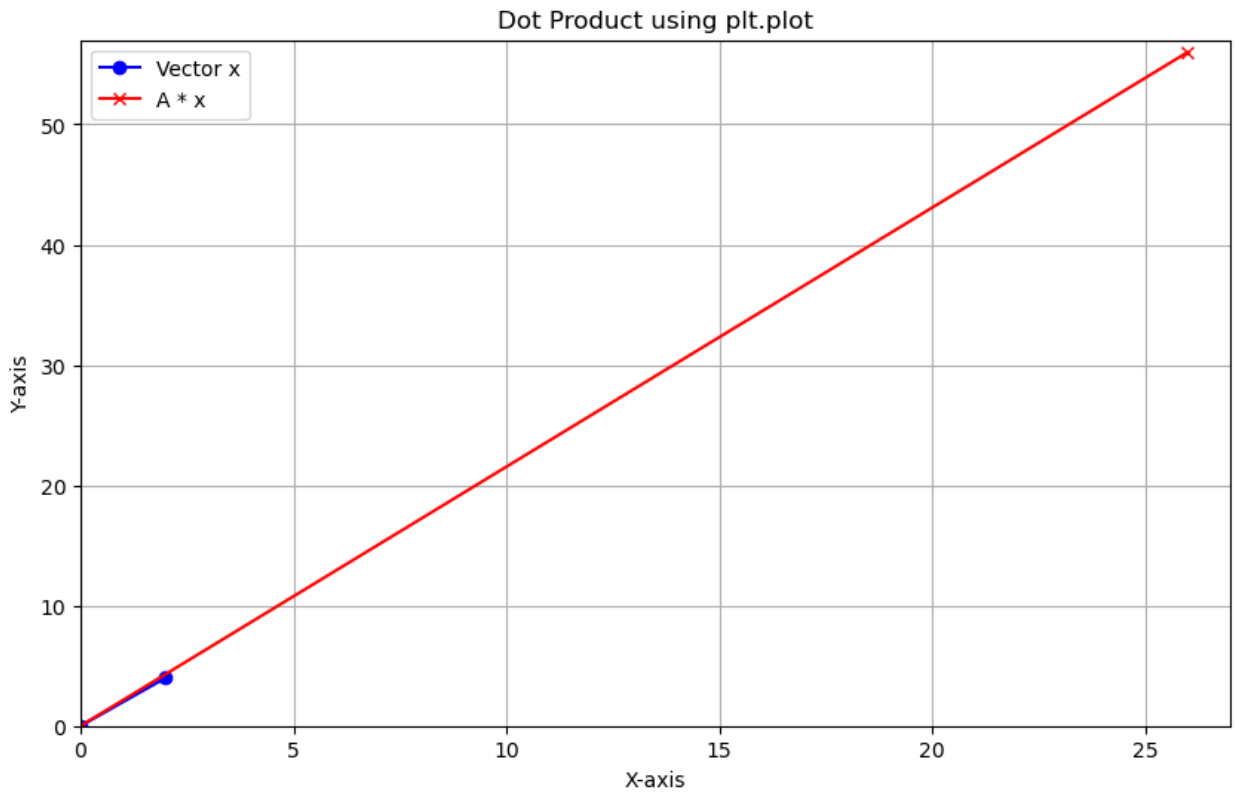
Demonstrate dot product (matrix vector multiplication) using both methods i.e. plt.plot as well as plt.quiver for below data. Also add xlim, ylim, label, xlabel, ylabel, color, title, lagend, grid, marker. Write code and visulaize for both the method.

```
A= np.array([[3, 5],
             [8, 10]])
x= np.array([2, 4])
```

Method 1: Using 'plt.plot'

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

# Define the matrix A and vector x
A = np.array([[3, 5], [8, 10]])
x = np.array([2, 4])

# Perform matrix-vector multiplication
result = A.dot(x)

# Plotting
plt.figure(figsize=(10, 6))

# Plotting the original vector x
plt.plot([0, x[0]], [0, x[1]], marker='o', color='b', label='Vector
x')

# Plotting the resulting vector A*x
plt.plot([0, result[0]], [0, result[1]], marker='x', color='r',
label='A * x')

# Setting the limits
plt.xlim(0, result[0] + 1)
plt.ylim(0, result[1] + 1)

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Dot Product using plt.plot')

# Adding legend
plt.legend()

# Adding grid
plt.grid(True)
```

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



Method 2: Using 'plt.quiver'

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

# Define the matrix A and vector x
A = np.array([[3, 5], [8, 10]])
x = np.array([2, 4])

# Perform matrix-vector multiplication
result = A.dot(x)

# Plotting
plt.figure(figsize=(10, 6))

# Plotting the original vector x using quiver
plt.quiver(0, 0, x[0], x[1], angles='xy', scale_units='xy', scale=1,
color='b', label='Vector x')

# Plotting the resulting vector A*x using quiver
plt.quiver(0, 0, result[0], result[1], angles='xy', scale_units='xy',
scale=1, color='r', label='A * x')
```

```
# Setting the limits
plt.xlim(0, result[0] + 1)
plt.ylim(0, result[1] + 1)

# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Dot Product using plt.quiver')

# Adding legend
plt.legend()

# Adding grid
plt.grid(True)

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

Explanation

1. **Defining the Matrix and Vector**:

```
A = np.array([[3, 5], [8, 10]])
x = np.array([2, 4])
```

2. **Performing Matrix-Vector Multiplication**:

```
result = A.dot(x)
```

   - `A.dot(x)` computes the dot product of matrix `A` and vector `x`.

3. **Method 1: Using `plt.plot`**:
   - We plot the original vector `x` and the resulting vector `A*x` using `plt.plot`.
   - Customizations such as `marker`, `color`, `label`, `xlabel`, `ylabel`, `title`, `legend`, and `grid` are added.

4. **Method 2: Using `plt.quiver`**:
   - We plot the original vector `x` and the resulting vector `A*x` using `plt.quiver`, which is suitable for plotting vectors.
   - Customizations such as `color`, `label`, `xlabel`, `ylabel`, `title`, `legend`, and `grid` are added.

5. **Displaying the Plot**:
   - `plt.show()` renders and displays the plot for both methods.

This code will visualize the dot product of the matrix-vector multiplication using both plotting methods, with all requested customizations.

Question 16

Create and interactive map using folium showing specific location using below data. Import foliumn and create a base map centered around the USA. Add markers for each city in the location dictionary. Also include popups with the city names for each markers. Descibe how this could be use in a real world application.

```
Data :
location = {
    "New York": (40.7128, -74.0060),
    "San Francisco": (37.7749, -122.4194),
    "Chicago": (41.8781, -87.6298)
    }

import folium

# Data: locations of cities
location = {
    "New York": (40.7128, -74.0060),
    "San Francisco": (37.7749, -122.4194),
    "Chicago": (41.8781, -87.6298)
}

# Create a base map centered around the USA
```

```
m = folium.Map(location=[39.8283, -98.5795], zoom_start=4)

# Add markers for each city in the location dictionary
for city, coordinates in location.items():
    folium.Marker(
        location=coordinates,
        popup=city,
        icon=folium.Icon(color='blue', icon='info-sign')
    ).add_to(m)

# Save the map as an HTML file
# m.save('us_cities_map.html')

# If you are running this in a Jupyter notebook, use the following to
display the map:
m
```

```
<folium.folium.Map at 0x1d94e9e0ad0>
```

Description

1. **Importing the Required Library**:

```
import folium
```

   - `folium` is used to create interactive maps in Python.

2. **Data: Locations of Cities**:

```
location = {
    "New York": (40.7128, -74.0060),
    "San Francisco": (37.7749, -122.4194),
    "Chicago": (41.8781, -87.6298)
}
```

   - A dictionary `location` holds the coordinates of each city.

3. **Creating a Base Map**:

```
m = folium.Map(location=[39.8283, -98.5795], zoom_start=4)
```

   - `folium.Map` initializes a base map centered around the geographic center of the USA with a zoom level of 4.

4. **Adding Markers**:

```
for city, coordinates in location.items():
    folium.Marker(
        location=coordinates,
        popup=city,
```

```
            icon=folium.Icon(color='blue', icon='info-sign')
        ).add_to(m)
```

- A loop iterates over the `location` dictionary.
- `folium.Marker` adds a marker for each city with its coordinates, a popup displaying the city name, and a blue icon.

5. **Saving the Map**:

```
m.save('us_cities_map.html')
```

- `m.save('us_cities_map.html')` saves the created map as an HTML file.

6. **Displaying the Map in a Jupyter Notebook**:

```
# m
```

- Uncomment `# m` if running in a Jupyter notebook to display the map directly.

Real-World Applications

1. **Tourism and Travel Planning**:
   - An interactive map can help tourists visualize and plan their trips by displaying popular destinations, landmarks, and routes.
2. **Logistics and Transportation**:
   - Companies can use interactive maps to optimize routes, track deliveries, and manage fleets efficiently by visualizing key locations and real-time positions of vehicles.
3. **Real Estate**:
   - Real estate agents and buyers can use interactive maps to locate properties, view nearby amenities, and analyze neighborhood statistics.
4. **Public Services**:
   - Governments and organizations can display locations of public services like hospitals, schools, and emergency centers to inform and assist residents.
5. **Marketing and Sales**:
   - Businesses can visualize the geographic distribution of customers, sales territories, and market trends to strategize marketing efforts and sales operations.