1. Create a NumPy array with the values [10, 20, 30, 40, 50]. Add your name as a string (e.g., "John") at the end of the array and print the resulting array.

2. Create a Pandas DataFrame with columns "Name", "Age", and "Class". Include your name, your age, and a random class assignment (e.g., "A", "B", or "C") as the first row. Print the DataFrame.

3. Create a line plot using Matplotlib where the x-axis represents the letters of your name (e.g., ['J', 'O', 'H', 'N']) and the y-axis represents the number of times each letter appears in your name.

4. Generate a NumPy array of 5 random numbers. Rename the array variable to include your name as a suffix (e.g., `random_numbers_John`). Find the sum of the array elements.

5. Create a DataFrame with columns "Product", "Price", and "Seller". Add at least three products and prices. For the "Seller" column, include your name for each row. Display only the rows where the price is greater than 50.

6. Create a bar chart showing sales data for 5 products. Label the title of the chart with your name (e.g., "John's Sales Data").

7. Generate a histogram with 50 random numbers using Seaborn. Use your name as part of the figure's title (e.g., "Distribution of Random Numbers by John").

8. Create a 3x3 NumPy matrix with random integers between 1 and 10. Name the matrix variable as `matrix_<YourName>` (e.g., `matrix_John`). Calculate the sum of all elements in the matrix.

9. Load a CSV file into a Pandas DataFrame. Add a column labeled "Reviewer" and fill it with your name for all rows. Display the first 5 rows of the modified DataFrame.

10. Create a scatter plot with a regression line using Seaborn. Use random `x` values and calculate `y` values using the formula `y = 2x + noise`. Title the chart with your name (e.g., "John's Scatter Plot with Regression Line").

In [31]:
```
# Creating a NumPy array
import numpy as np
arr = np.array([10, 20, 30, 40, 50,"Siddharth"])
print("Array:", arr)
```

```
Array: ['10' '20' '30' '40' '50' 'Siddharth']
```

The provided code creates a NumPy array containing a mix of numeric values and a string ("Siddharth"). Since NumPy arrays must have elements of the same data type, the array will automatically cast all elements to the string data type. As a result, even numeric values will be treated as strings.

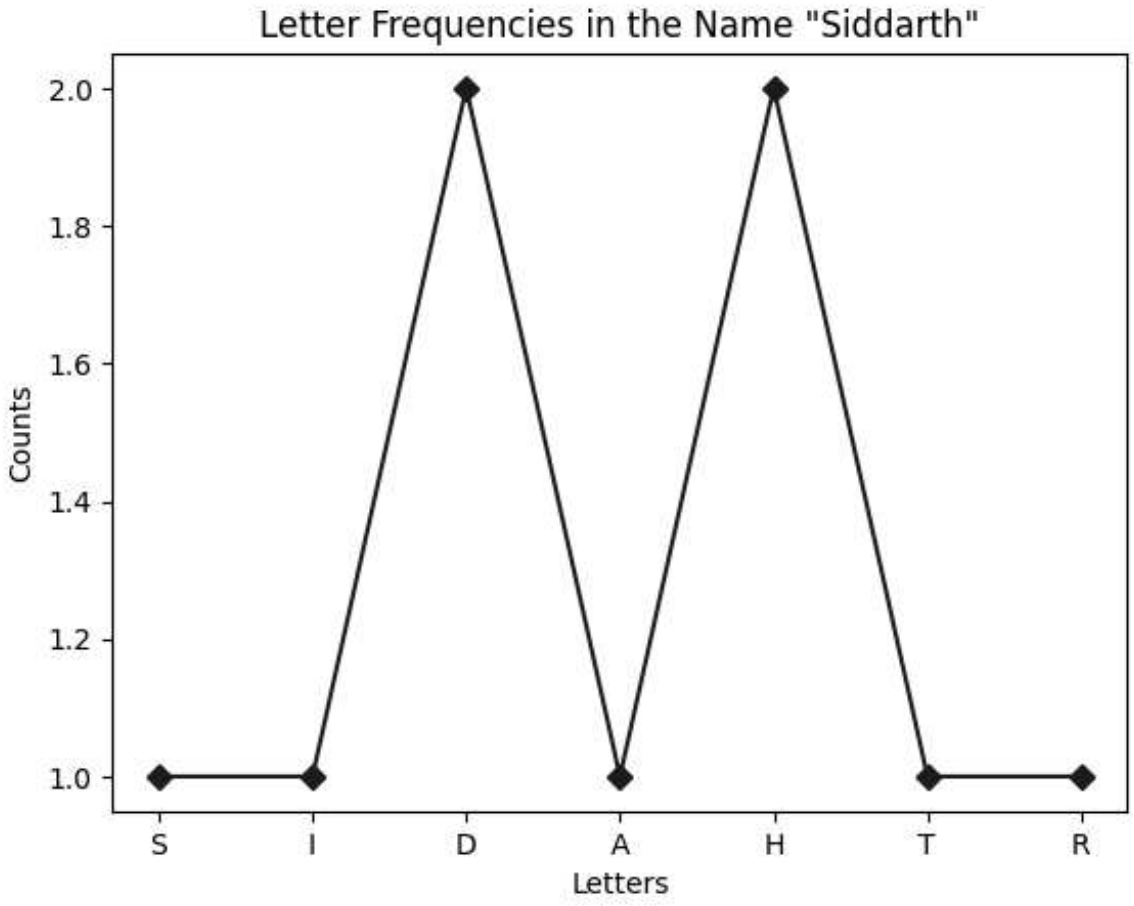
In [32]:
```
import pandas as pd
# Creating a DataFrame
data = {'Name': ['Siddharth'], 'Age': [19], 'Class': ["A"]}
df = pd.DataFrame(data)
print("DataFrame:\n", df)
```

```
DataFrame:
         Name  Age Class
0  Siddharth   19     A
```

The provided code creates a simple Pandas DataFrame with one row of data. The DataFrame contains three columns: `"Name"`, `"Age"`, and `"Class"`, with respective values `"Siddharth"`, `19`, and `"A"`. This demonstrates how to use a dictionary to initialize a DataFrame in Pandas.

In [33]:
```
import matplotlib.pyplot as plt
from collections import Counter

# List representing the letters of your name
list_a = ['S', 'I', 'D', 'D', 'H', 'A', 'R', 'T', 'H']

print(Counter("Sidhartha"))

# Counting the occurrences of each letter
s = list_a.count('S')
i = list_a.count('I')
d = list_a.count('D')
h = list_a.count('H')
a = list_a.count('A')
t = list_a.count('T')
r = list_a.count('R')

# Lists for x-axis (letters) and y-axis (counts)
letters = ['S', 'I', 'D', 'A', 'H', 'T', 'R']
counts = [s, i, d, a, h, t, r]

# Creating the line plot
plt.plot(letters, counts, marker='D', linestyle='-', color='b')

# Adding labels and title
plt.xlabel('Letters')
plt.ylabel('Counts')
plt.title('Letter Frequencies in the Name "Siddarth"')

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

```
Counter({'h': 2, 'a': 2, 'S': 1, 'i': 1, 'd': 1, 'r': 1, 't': 1})
```



This code calculates the frequency of each letter in the name "Siddarth" using both a Counter object and manual counting with the .count() method. A line plot is created with Matplotlib to visualize the frequencies of the letters in the name. The x-axis represents the letters, and the y-axis shows the corresponding counts, with markers and a blue line connecting the points. The plot is labeled and titled to clearly represent the data.

In [34]:
```
import numpy as np

# Generating a NumPy array of 5 random numbers
random_numbers_Siddharth = np.random.rand(5)

# Calculating the sum of the array elements
sum_of_elements = np.sum(random_numbers_Siddharth)

# Printing the array and its sum
print("Random numbers array:", random_numbers_Siddharth)
print("Sum of array elements:", sum_of_elements)
```

```
Random numbers array: [0.892559   0.53934224 0.80744016 0.8960913  0.3180
0347]
Sum of array elements: 3.4534361704650482
```

This code generates a NumPy array with 5 random numbers between 0 and 1 using `np.random.rand(5)` and calculates the sum of its elements using `np.sum()`. The array values and their sum are then printed. This demonstrates how to use NumPy for random number generation and basic array operations.

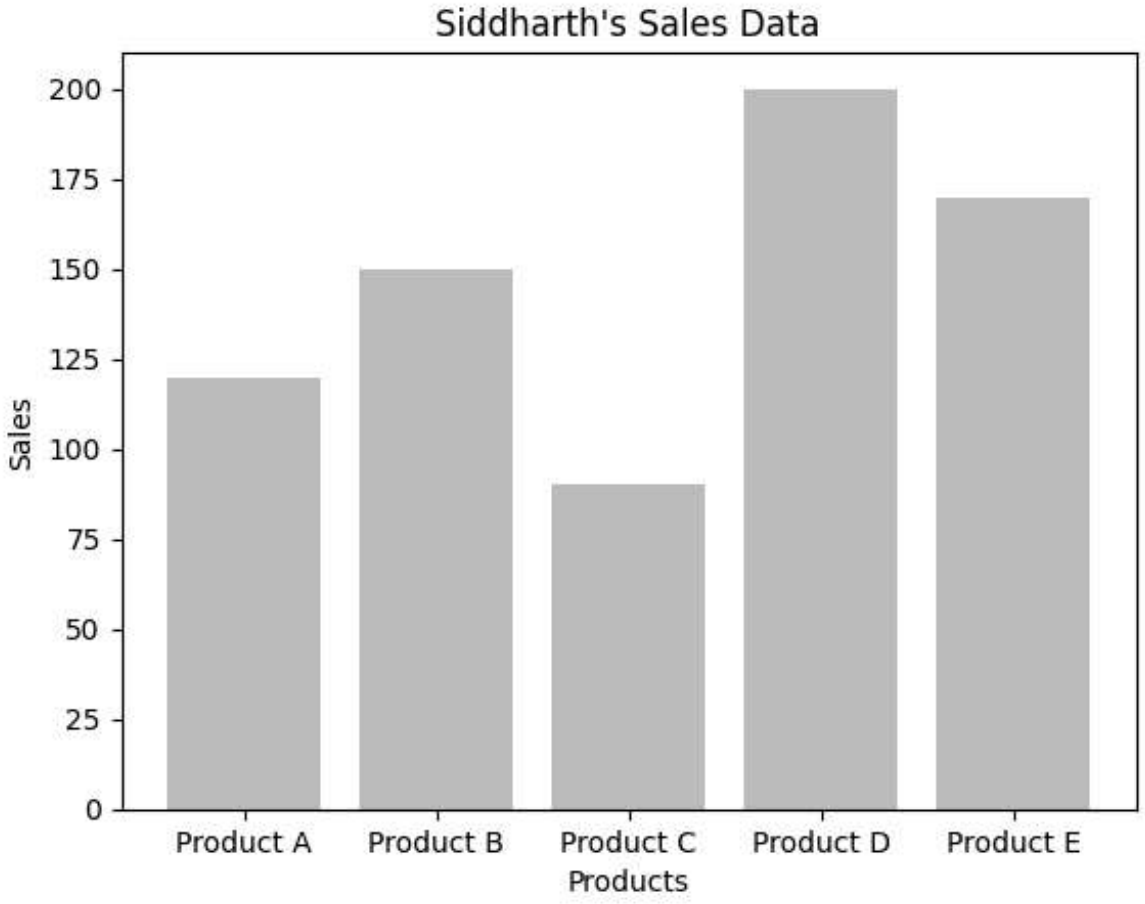
In [35]:
```
import pandas as pd

# Creating the DataFrame
data = {
    'Product': ['Book', 'Copy', 'Pen'],
    'Price': [975,98,28],
    'Seller': ["Siddharth","Siddharth","Siddharth"]
}
df = pd.DataFrame(data)

# Calculating the average marks
average_price = df['Price'].mean()

print("DataFrame:\n", df)
print("Average Price:", average_price)
print()

# Filtering rows where marks are greater than 70
filtered_df = df[df['Price'] > 70]

print("\n\nFiltered DataFrame:\n", filtered_df.to_string(index=False)) #
```

```
DataFrame:
   Product  Price     Seller
0    Book    975  Siddharth
1    Copy     98  Siddharth
2     Pen     28  Siddharth
Average Price: 367.0



Filtered DataFrame:
 Product  Price    Seller
   Book    975 Siddharth
   Copy     98 Siddharth
```

This code creates a Pandas DataFrame with details of three products (`'Book'` , `'Copy'` , `'Pen'`) and their prices, all sold by `"Siddharth"` . It calculates the average price of the products using the `mean()` method and filters the rows where the price is greater than 70. The filtered DataFrame and average price are then printed, demonstrating basic data manipulation operations like filtering and aggregation in Pandas. Disabling the index in the final output makes it more user-friendly.

In [36]:
```
# Create a bar chart for data of 5 products:

import matplotlib.pyplot as plt

# Products data
products = ['Product A', 'Product B', 'Product C', 'Product D', 'Product
sales = [120, 150, 90, 200, 170]

# Creating the bar chart
plt.bar(products, sales, color='skyblue')
```

```
plt.title("Siddharth's Sales Data")
plt.xlabel('Products')
plt.ylabel('Sales')
plt.show()
```

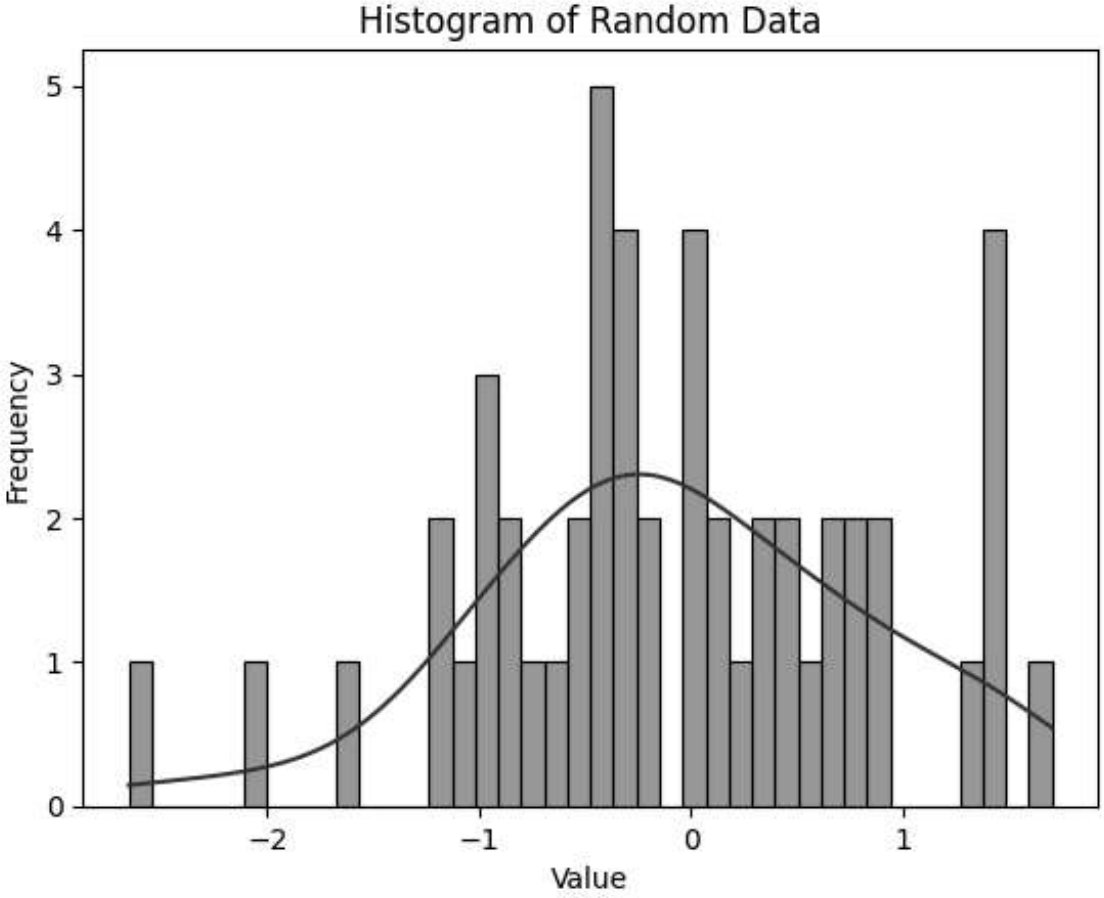


This code creates a bar chart to represent the sales data of five products (`'Product A'` to `'Product E'`). The sales figures are displayed on the y-axis, while the product names are on the x-axis. The bars are styled with a light blue (`'skyblue'`) color, and the chart is labeled with a title (`"Siddharth's Sales Data"`) and axis labels. This visual representation effectively compares the sales performance of the products.

In [37]:
```
# Generate a histogram for random data of size 50 using Seaborn:

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

# Generating random data
random_data = np.random.randn(50)
print("Random Data : ", random_data)

# Creating the histogram
sns.histplot(random_data, bins=40, kde=True, color='purple')
# In the context of Seaborn's histplot function, the kde parameter stands
# When you set kde=True, it adds a smooth curve over the histogram, which

# This can be particularly useful for visualizing the distribution of dat
# as opposed to the discrete bins of a histogram. The KDE curve helps to
```

```
# patterns and trends in the data that might not be immediately apparent
plt.title('Histogram of Random Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

```
Random Data :  [-0.25663018 -0.36782572  1.27373362 -0.29195267 -2.655176
05  0.34551794
 -0.39551645 -0.28913686  0.45293633 -0.16606091  0.21493883 -2.02231493
 -0.94305681  1.40395874 -0.0185508  -1.67350462 -1.07253183 -0.99258618
  0.10234768 -0.43260928 -0.6591823   0.0039373   0.4777541  -0.25902864
 -0.57470921 -0.42149822  0.33982096 -0.00738015  0.76729684 -1.14979258
 -0.77533611  0.77314086 -0.80182784  1.38401572  1.40520531  1.39232575
 -0.88064082  0.07689495 -0.49343248  0.92316261  1.70660495  0.87358942
  0.00914435 -0.3655393   0.64908673 -1.22287354  0.53633603 -0.91469093
  0.62054822 -0.16093738]
```



This code generates a histogram for a dataset of 50 random numbers drawn from a standard normal distribution (`np.random.randn(50)`) using Seaborn's `histplot` function. The histogram displays the frequency of data points across 40 bins (`bins=40`) with a kernel density estimate (KDE) overlay for a smooth representation of the data distribution. The histogram uses a purple color (`color='purple'`), and the chart is appropriately labeled with a title, x-axis, and y-axis. This visualization effectively showcases the distribution and density of the random dataset.

In [38]:
```
import numpy as np

# Creating a 3x3 matrix with random numbers
```

```
matrix_siddharth = np.random.randint(1,3,(3, 3))


# Printing the original matrix
print("Original Matrix:\n", matrix_siddharth)

# Calculating the sum of the elements
matrix_sum = np.sum(matrix_siddharth)
print("Sum of all elements in the matrix:", matrix_sum)
```

```
Original Matrix:
 [[1 1 2]
 [1 1 2]
 [1 1 2]]
Sum of all elements in the matrix: 12
```

This code generates a 3x3 matrix (`matrix_siddharth`) with random integer values between 1 and 2 using `np.random.randint(1, 3, (3, 3))` . The `np.sum()` function is then used to calculate the sum of all the elements in the matrix.

Here's the breakdown:

- **Matrix Creation**: `np.random.randint(1, 3, (3, 3))` creates a 3x3 matrix where each element is either 1 or 2.
- **Sum Calculation**: The sum of all elements in the matrix is calculated using `np.sum()` and displayed.

The code will print the matrix and its sum, providing an understanding of both the data and its aggregate value.

In [39]:
```
import pandas as pd

# Load the CSV file into a Pandas DataFrame
df = pd.read_csv("students.csv")

# Add a new column named "Reviewer" and fill it with your name
df['Reviewer'] = "Siddharth"

# Display the first 5 rows of the modified DataFrame
print(df.head())
```

```
      Name  Marks Grade   Reviewer
0    Alice     85     A  Siddharth
1      Bob     72     B  Siddharth
2  Charlie     90    A+  Siddharth
3    David     68     C  Siddharth
4     John     90    A+  Siddharth
```

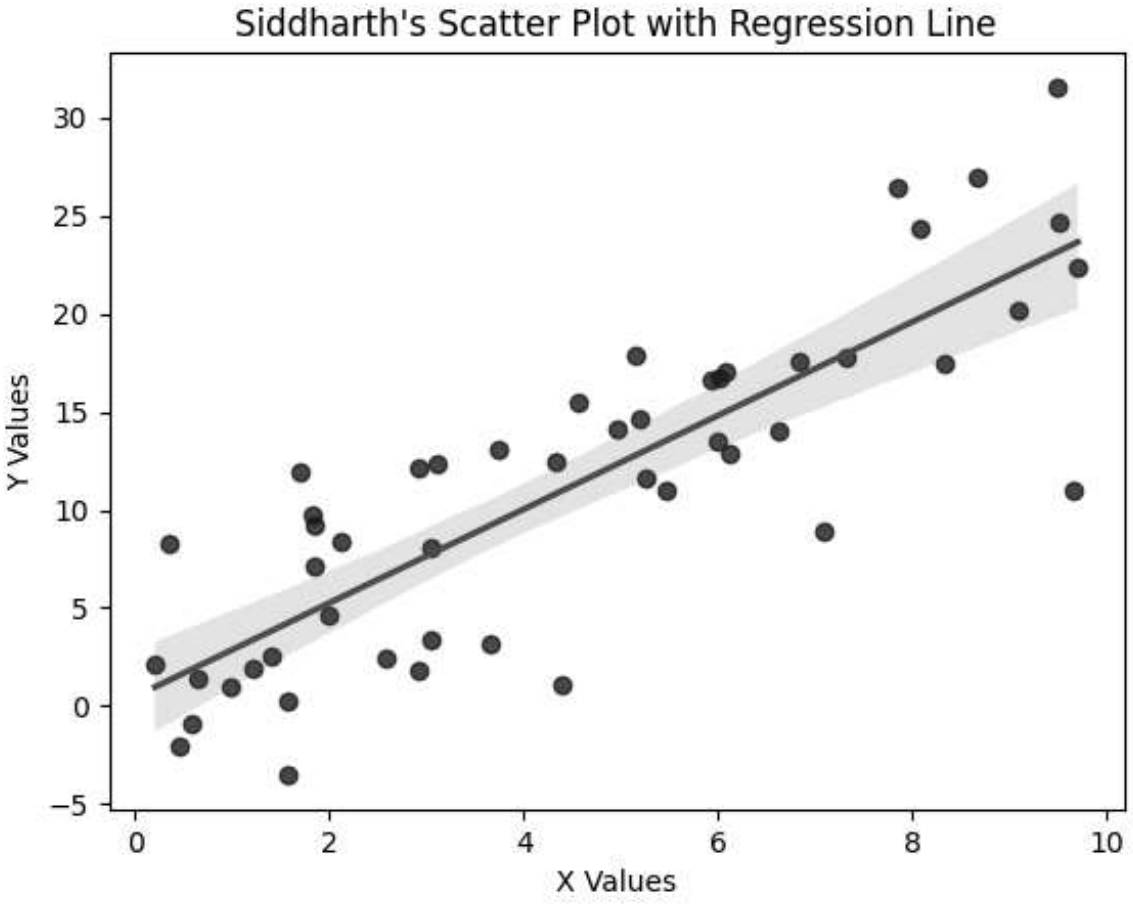
This code loads a CSV file (`students.csv`) into a Pandas DataFrame, adds a new column called `"Reviewer"` , and fills it with the name `"Siddharth"` for all rows. The `.head()` method is then used to display the first 5 rows of the modified DataFrame.

Key Steps:

1. **Loading CSV**: The `pd.read_csv("students.csv")` function reads the `students.csv` file into a DataFrame.
2. **Adding a Column**: The new column `"Reviewer"` is added, and the entire column is filled with the value `"Siddharth"`.
3. **Displaying Data**: `df.head()` shows the first 5 rows of the DataFrame, allowing you to quickly inspect the result.

Make sure the file `students.csv` is in the correct directory or specify its full path for successful loading.

In [40]:
```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# Generating sample data
np.random.seed(42)
x = np.random.rand(50) * 10  # Random x values
y = 2.5 * x + np.random.randn(50) * 5  # Linear relation with noise

# Creating the scatter plot with regression line
sns.regplot(x=x, y=y, scatter_kws={'color': 'blue'}, line_kws={'color':
plt.title("Siddharth's Scatter Plot with Regression Line")
plt.xlabel('X Values')
plt.ylabel('Y Values')
plt.show()
```



This code generates a scatter plot with a regression line using Seaborn. It creates random `x` values between 0 and 10, and `y` values are calculated with a linear relationship (`y = 2.5 * x`) and added noise (`np.random.randn(50) * 5`).

Here's the breakdown:

- **Random Data**: `x` values are randomly generated, and `y` values are based on a linear function of `x` with some noise to simulate real-world data.
- **Scatter Plot and Regression Line**: The `sns.regplot()` function creates the scatter plot and overlays a regression line. The scatter points are colored blue (`scatter_kws={'color': 'blue'}`) and the regression line is red (`line_kws={'color': 'red'}`).
- **Labels and Title**: The plot is titled `"Siddharth's Scatter Plot with Regression Line"` , and the axes are labeled accordingly.

This visualization allows for the analysis of the linear relationship between `x` and `y` , along with the regression line that best fits the data.