DATA VISUALIZATION WITH PYTHON

LIBRARY OVERVIEW

MATPLOTLIB:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

USES:

Matplotlib is used for data exploration and analysis, presentation and reporting, scientific plotting, machine learning and data science visualizations, web-based visualization, education and learning, dashboard creation, quality control and monitoring, geospatial data visualization, and custom plotting for artistic visualization.

TYPES OF PLOTS:

- Line Plot
- Scatter Plot
- Histogram
- Bar Plot
- Pie Chart
- Box Plot (Box-and-Whisker Plot)
- Violin Plot
- Heatmap
- Contour Plot
- Hexbin Plot
- Polar Plot
- 3D Plotting (e.g., 3D Scatter Plot, Surface Plot)
- Stacked Bar Plot
- Error Bars
- Quiver Plot
- Stream Plot

SEABORN:

Seaborn is a library for making statistical graphics in Python. It builds on top on matplotlib and integrates closely with pandas data structures.

USES:

Seaborn helps you explore and understand your data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

TYPES OF PLOTS:

- Scatter plot
- Line plot
- Bar plot
- Count plot
- Box plot
- Violin plot
- Histogram
- Kernel Density Estimation (KDE) plot
- Distribution plot (deprecated)
- Heatmap
- Clustered heatmap
- Scatter plot with regression fit
- Facet Grid-based plotting for linear models
- Pairwise scatter plots
- Joint plot showing relationship between two variables
- Multi-plot grid for conditional relationships
- Line plot with datetime objects on x-axis
- Facet Grid-based plotting for time series data

COMMONLY USED PLOTS IN MATPLOTLIB AND SEABORN

SCATTER PLOT:

- Displays individual data points as markers, useful for visualizing relationships between two variables.
- Each point represents an observation in a two-dimensional space.

Real Time Scenario:

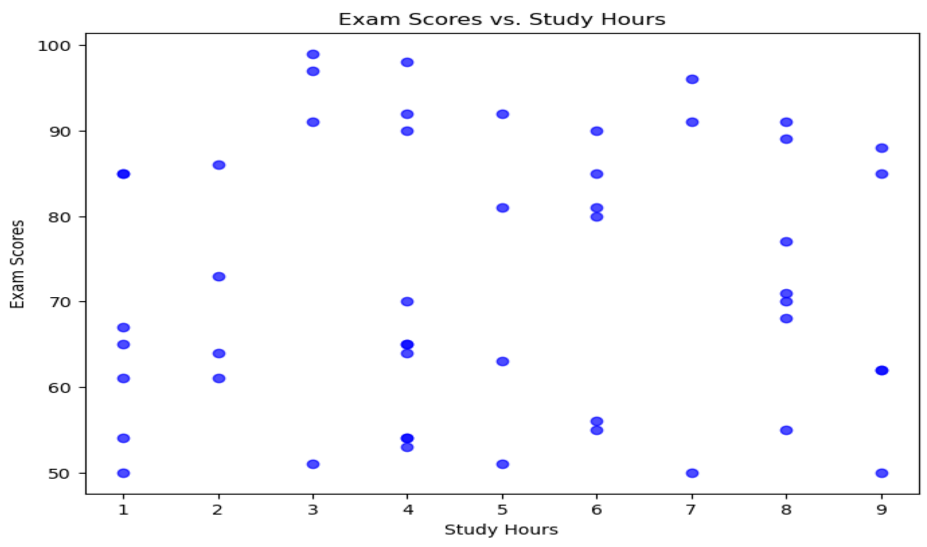
Suppose we have a dataset that represents the relationship between a student's study hours and their exam scores. We want to visualize how study hours affect exam scores using a scatter plot.

Using Matplotlib:

```
#scatter plot using matplotlib
import matplotlib.pyplot as plt

# Generate random data for study hours and exam scores
np.random.seed(0)
study_hours = np.random.randint(1, 10, size=50)  # Generate 50 random study hours (1-9 hours)
exam_scores = np.random.randint(50, 100, size=50)  # Generate corresponding random exam scores (50-99)

# Create a scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(study_hours, exam_scores, color='blue', alpha=0.7)  # Plot study hours vs. exam scores
plt.title('Exam Scores vs. Study Hours')
plt.xlabel('Study Hours')
plt.ylabel('Exam Scores')
plt.grid(False)
plt.show()
```

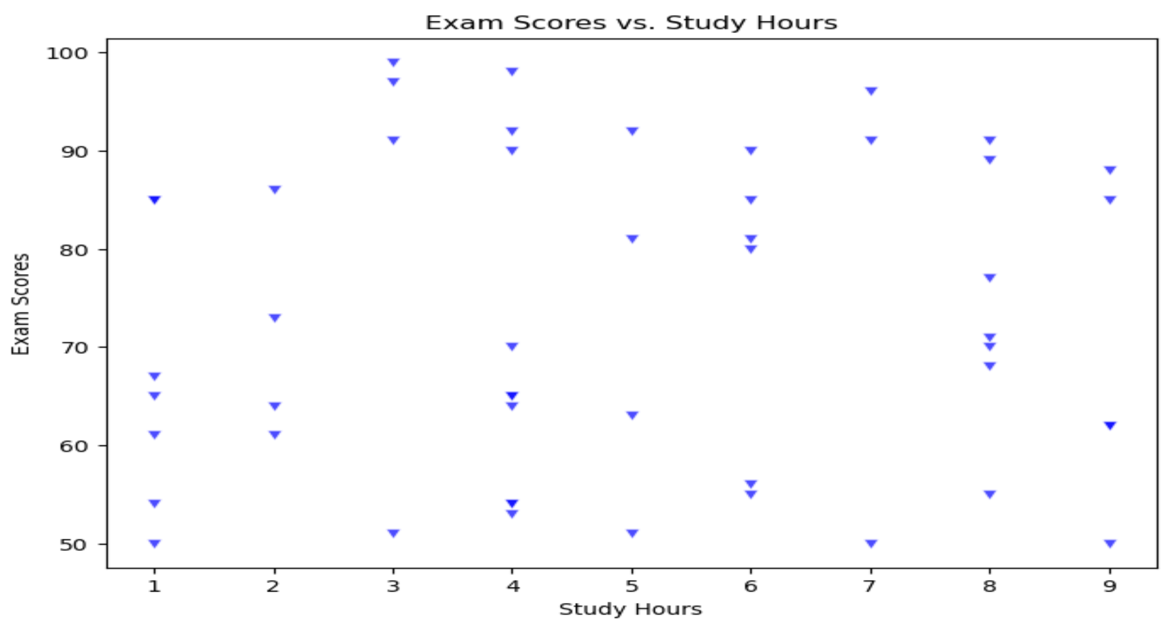


Using Seaborn:

```python
#scatter plot using seaborn
import seaborn as sns

# Generate random data for study hours and exam scores
np.random.seed(0)
study_hours = np.random.randint(1, 10, size=50)  # Generate 50 random study hours (1-9 hours)
exam_scores = np.random.randint(50, 100, size=50)  # Generate corresponding random exam scores (50-99)

# Create a scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=study_hours, y=exam_scores, color='blue', alpha=0.7,marker='v')# Plot study hours vs.exam scores
plt.title('Exam Scores vs. Study Hours')
plt.xlabel('Study Hours')
plt.ylabel('Exam Scores')
plt.grid(False)
plt.show()
```



LINE PLOT:

- Used to visualize data points connected by straight line segments.
- Typically used for time series data, trend lines, or continuous data.

Real Time Scenario:

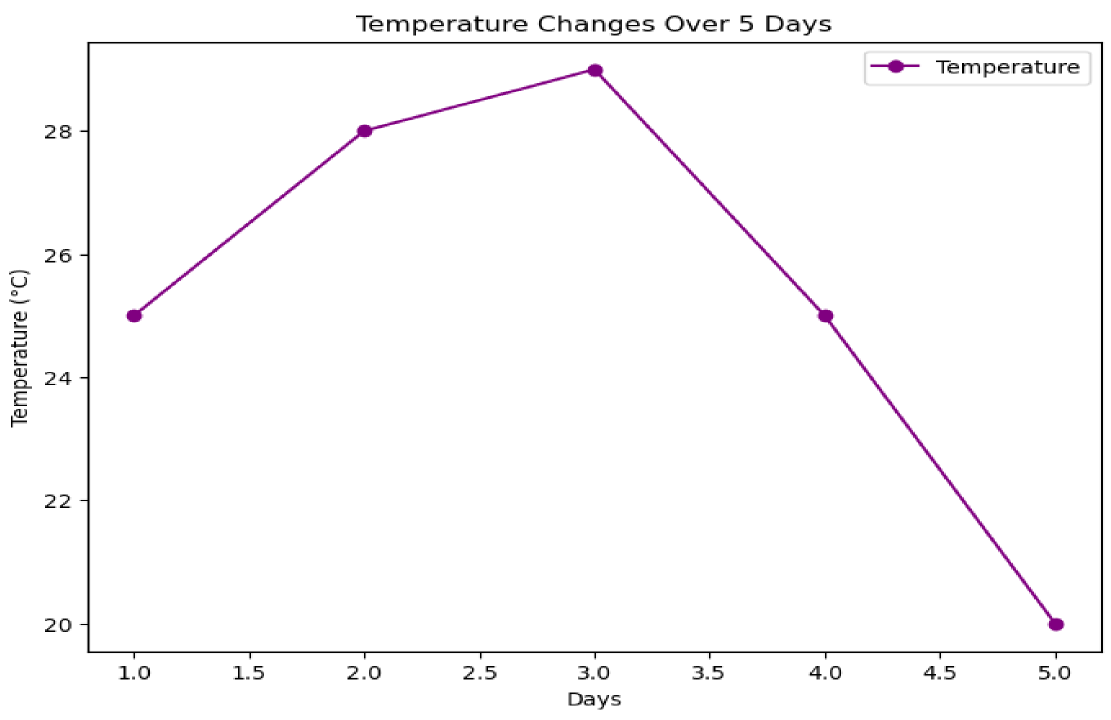
Suppose we want to visualize the temperature changes over 5 days. We can use line plot to visualize this data.

Using Matplotlib:

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

# Generate random data for time (days) and temperature (degrees Celsius)
np.random.seed(1)
days = np.arange(1, 6)  # 5 days
temperature = np.random.randint(20, 30, size=5)  # Random temperature values (20-29 degrees Celsius)

# Create a line plot using Matplotlib
plt.figure(figsize=(8, 6))
plt.plot(days, temperature, marker='o', linestyle='-', color='purple', label='Temperature')
plt.xlabel('Days')
plt.ylabel('Temperature (°C)')
plt.title('Temperature Changes Over 5 Days')
plt.legend()
plt.grid(False)
plt.show()
```

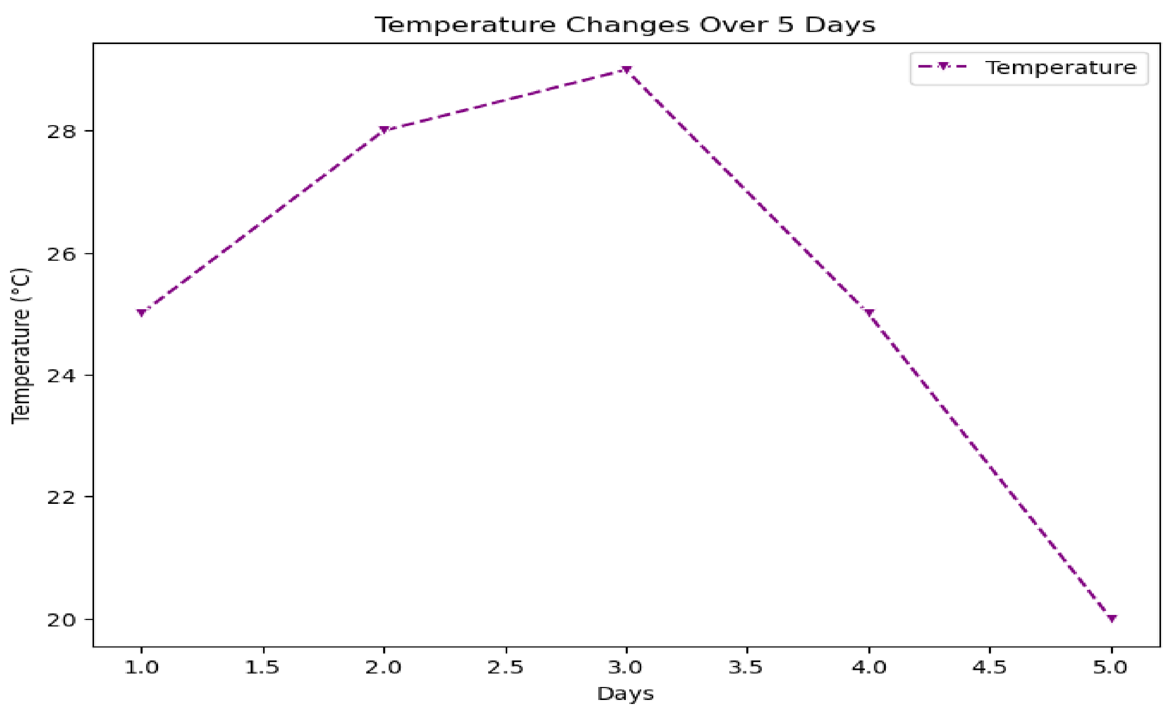


Using Seaborn:

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

# Generate random data for time (days) and temperature (degrees Celsius)
np.random.seed(1)
days = np.arange(1, 6)  # 5 days
temperature = np.random.randint(20, 30, size=5)  # Random temperature values (20-29 degrees Celsius)

# Create a line plot using Seaborn
plt.figure(figsize=(8, 6))
sns.lineplot(x=days, y=temperature, marker='v', linestyle='--', color='purple', label='Temperature')
plt.xlabel('Days')
plt.ylabel('Temperature (°C)')
plt.title('Temperature Changes Over 5 Days')
plt.legend()
plt.grid(False)
plt.show()
```



HISTOGRAM:

- Represents the distribution of numerical data by dividing data into bins and displaying the frequency of observations in each bin.
- Useful for understanding data distributions and identifying patterns.

Real Time Scenario:

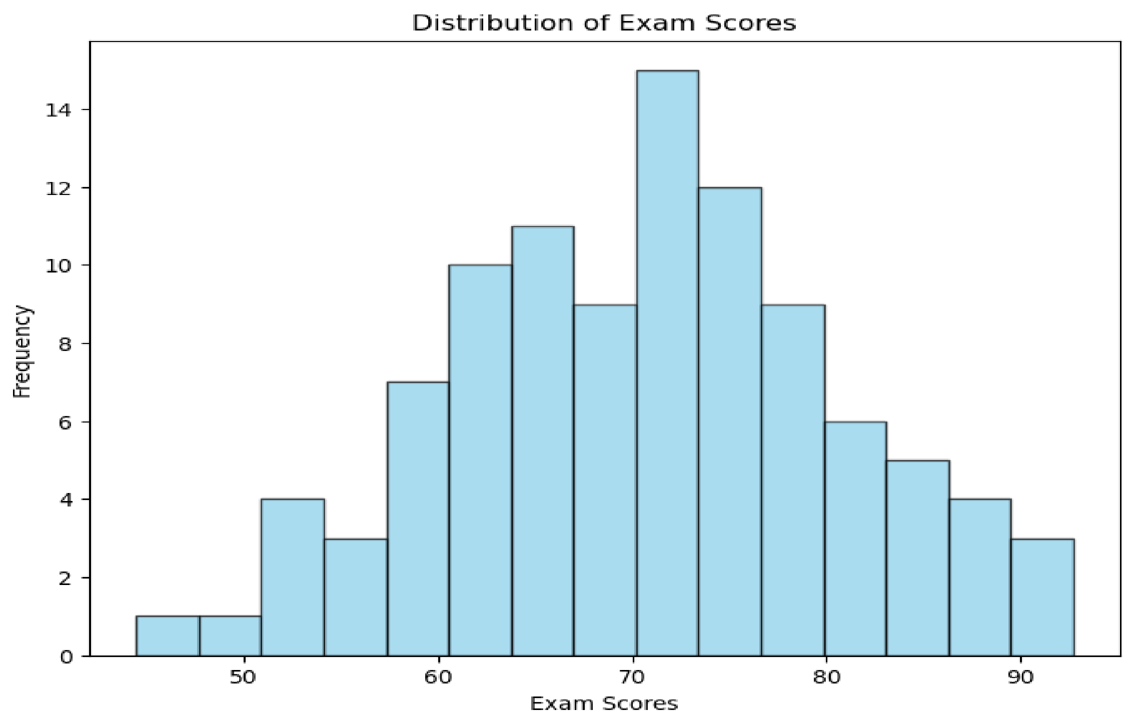
Suppose we have a dataset representing the exam scores of students, and we want to visualize the distribution of these scores using histograms.

Using Matplotlib:

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

# Generate random exam scores (normally distributed) for 100 students
np.random.seed(0)
exam_scores = np.random.normal(loc=70, scale=10, size=100)  # Mean=70, Standard Deviation=10

# Create a histogram using Matplotlib
plt.figure(figsize=(8, 6))
plt.hist(exam_scores, bins=15, color='skyblue', edgecolor='black', alpha=0.7)
plt.xlabel('Exam Scores')
plt.ylabel('Frequency')
plt.title('Distribution of Exam Scores')
plt.show()
```

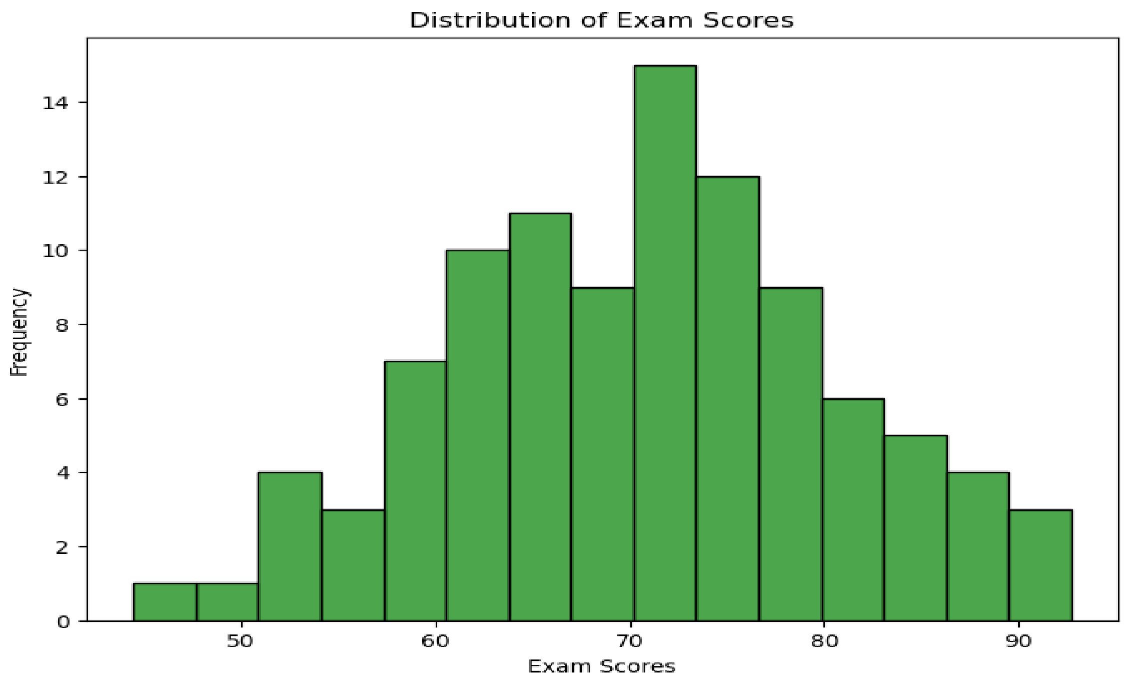


Using Seaborn:

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

# Generate random exam scores (normally distributed) for 100 students
np.random.seed(0)
exam_scores = np.random.normal(loc=70, scale=10, size=100)  # Mean=70, Standard Deviation=10

# Create a histogram using Seaborn
plt.figure(figsize=(8, 6))
sns.histplot(exam_scores, bins=15, color='green', edgecolor='black', alpha=0.7)
plt.xlabel('Exam Scores')
plt.ylabel('Frequency')
plt.title('Distribution of Exam Scores')
plt.show()
```



BAR PLOT:

- Displays categorical data with rectangular bars, where the height of each bar represents the value of a variable.
- Suitable for comparing quantities across different categories.

Real Time Scenario:

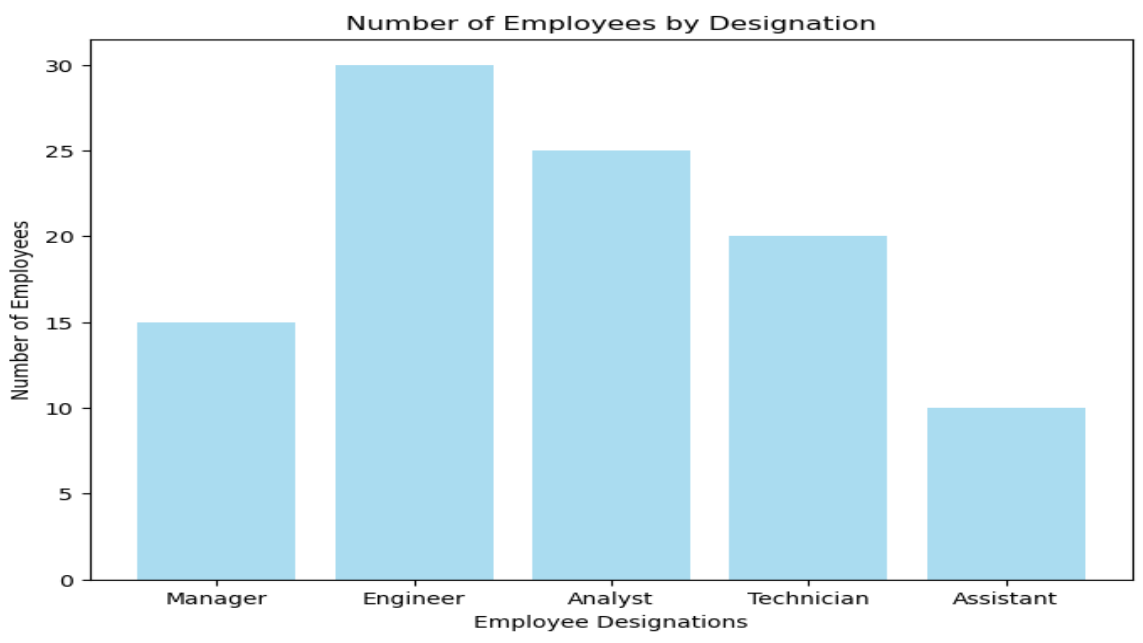
Suppose we want to visualize the number of employees under different designation using a bar plot.

Using Matplotlib:

```
import matplotlib.pyplot as plt

# Categorical data: Employee designations and corresponding number of employees
designations = ['Manager', 'Engineer', 'Analyst', 'Technician', 'Assistant']
num_employees = [15, 30, 25, 20, 10]  # Number of employees in each designation

# Create a bar plot using Matplotlib
plt.figure(figsize=(8, 6))
plt.bar(designations, num_employees, color='skyblue', alpha=0.7)
plt.xlabel('Employee Designations')
plt.ylabel('Number of Employees')
plt.title('Number of Employees by Designation')
plt.show()
```

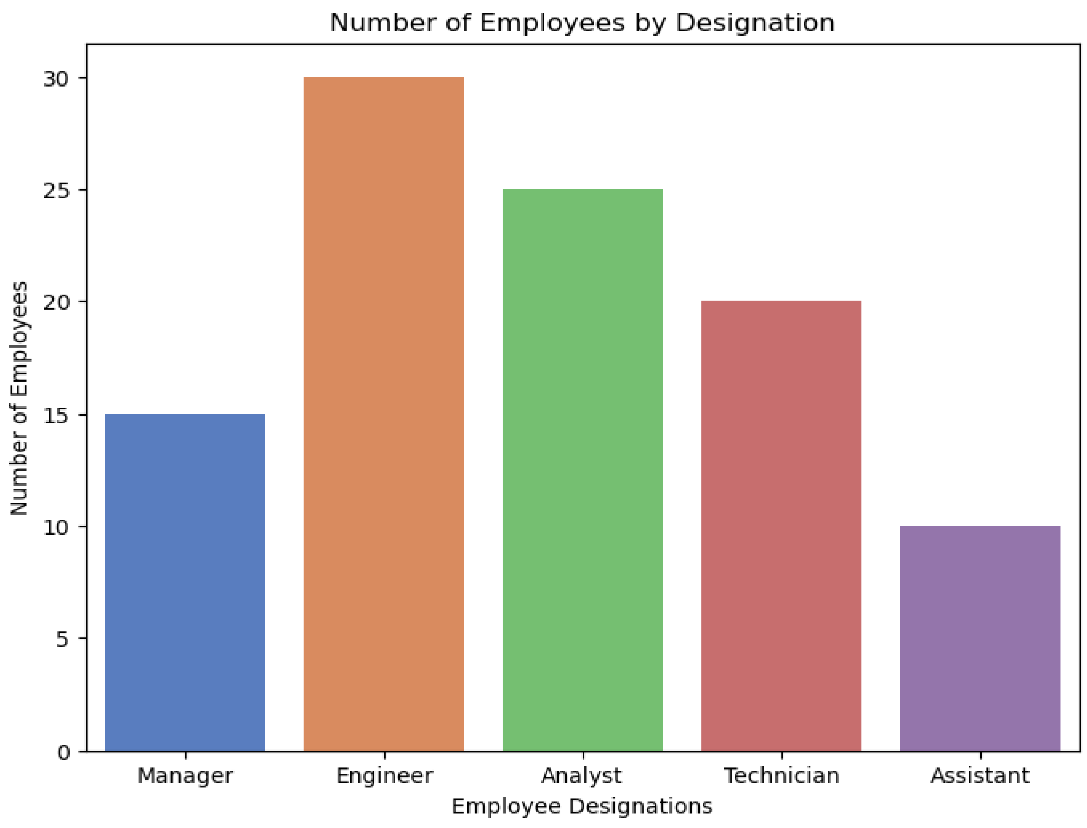


Using Seaborn:

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

# Categorical data: Employee designations and corresponding number of employees
designations = ['Manager', 'Engineer', 'Analyst', 'Technician', 'Assistant']
num_employees = [15, 30, 25, 20, 10]  # Number of employees in each designation

# Create a bar plot using Seaborn
plt.figure(figsize=(8, 6))
sns.barplot(x=designations, y=num_employees, palette='muted')
plt.xlabel('Employee Designations')
plt.ylabel('Number of Employees')
plt.title('Number of Employees by Designation')
plt.show()
```



BOX PLOT:

- Summarizes the distribution of numerical data using quartiles (median, first quartile, third quartile) and outliers.
- Useful for detecting outliers and comparing distributions.

Real Time Scenario:

Suppose we have sales data from a company operating in different regions, and we want to analyse and compare the sales performance across these regions using box plots.

Using Matplotlib:

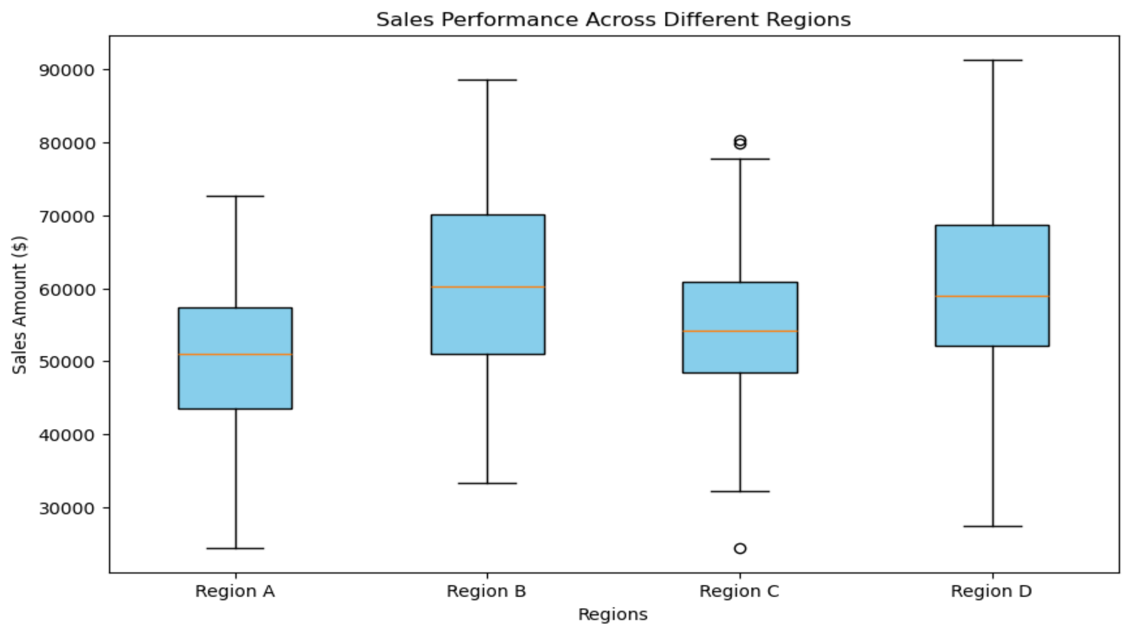
```python
import matplotlib.pyplot as plt
import numpy as np

# Generate random sales data for different regions
np.random.seed(0)

# Sales data for four regions (normally distributed)
region_a_sales = np.random.normal(loc=50000, scale=10000, size=100)
region_b_sales = np.random.normal(loc=60000, scale=12000, size=100)
region_c_sales = np.random.normal(loc=55000, scale=11000, size=100)
region_d_sales = np.random.normal(loc=62000, scale=13000, size=100)

# Combine sales data into a list of lists
all_sales = [region_a_sales, region_b_sales, region_c_sales, region_d_sales]

# Create a box plot using Matplotlib
plt.figure(figsize=(10, 6))
plt.boxplot(all_sales, labels=['Region A', 'Region B', 'Region C', 'Region D'],
            patch_artist=True, boxprops=dict(facecolor='skyblue'))
plt.xlabel('Regions')
plt.ylabel('Sales Amount ($)')
plt.title('Sales Performance Across Different Regions')
plt.show()
```



Using Seaborn:

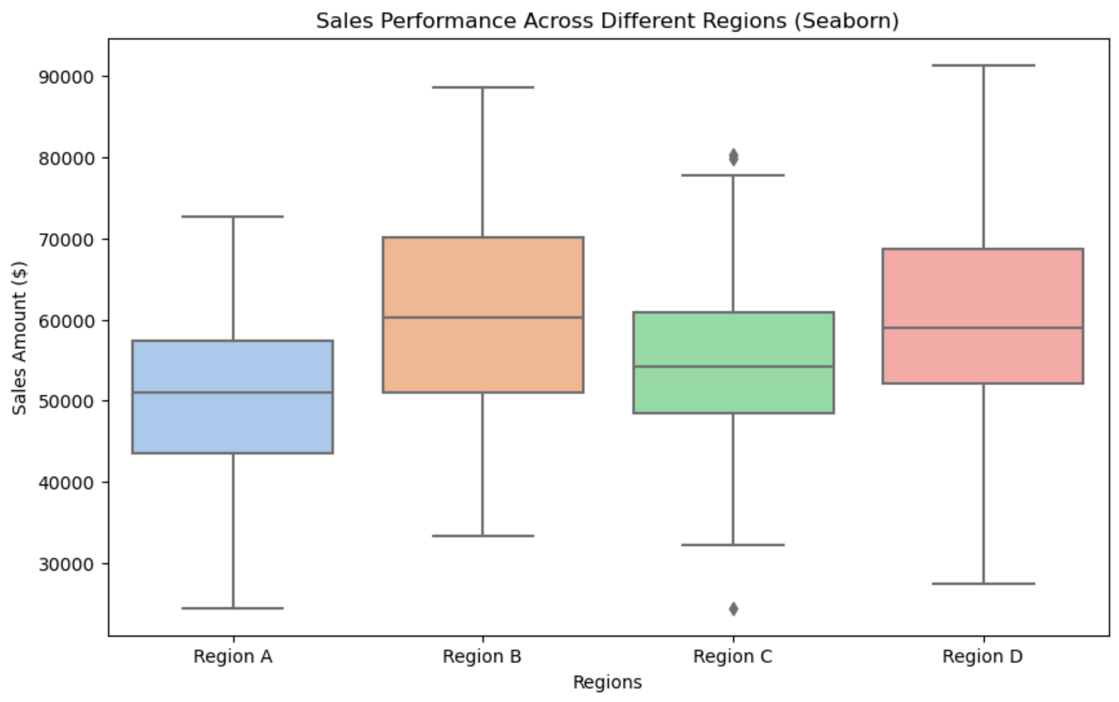
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Generate random sales data for different regions
np.random.seed(0)

# Sales data for four regions (normally distributed)
region_a_sales = np.random.normal(loc=50000, scale=10000, size=100)
region_b_sales = np.random.normal(loc=60000, scale=12000, size=100)
region_c_sales = np.random.normal(loc=55000, scale=11000, size=100)
region_d_sales = np.random.normal(loc=62000, scale=13000, size=100)

# Create a DataFrame to store the sales data with region labels
data = pd.DataFrame({
    'Region': ['Region A']*100 + ['Region B']*100 + ['Region C']*100 + ['Region D']*100,
    'Sales Amount ($)': np.concatenate([region_a_sales, region_b_sales,
                                        region_c_sales, region_d_sales])})

# Create a box plot using Seaborn
plt.figure(figsize=(10, 6))
sns.boxplot(x='Region', y='Sales Amount ($)', data=data, palette='pastel')
plt.xlabel('Regions')
plt.ylabel('Sales Amount ($)')
plt.title('Sales Performance Across Different Regions (Seaborn)')
plt.show()
```



VIOLIN PLOT:

- Combines a box plot with a kernel density plot, providing a more detailed view of the data's distribution.
- Useful for comparing distributions across different categories.

Real Time Scenario:

Suppose we have exam scores of students from different classes, and we want to visualize and compare the distribution of these scores using violin plots.

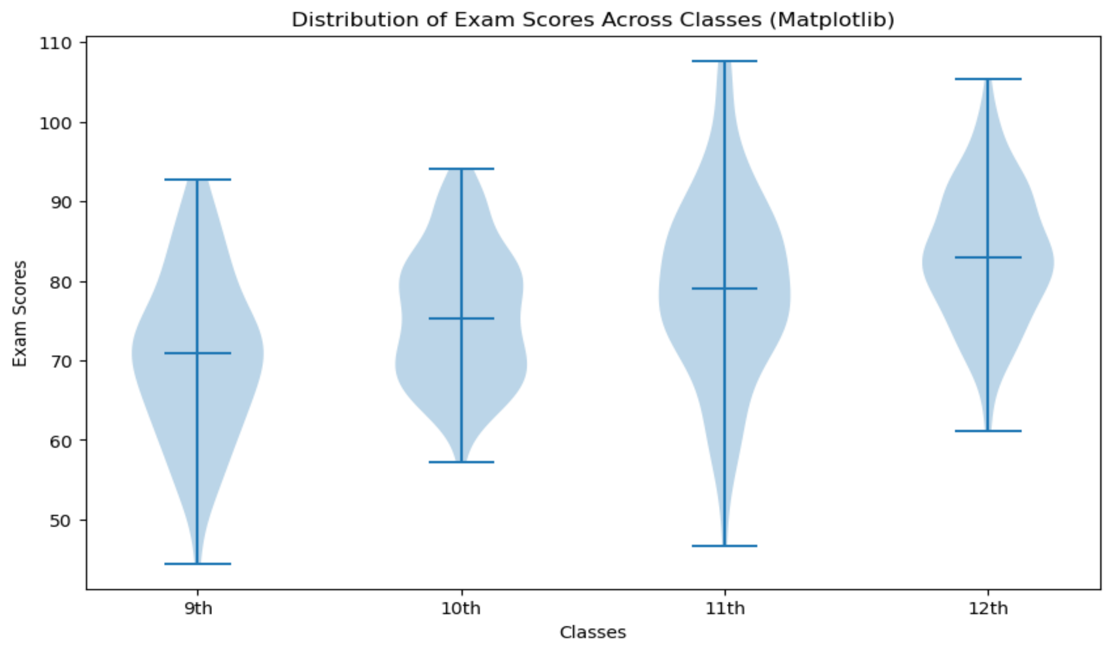
Using Matplotlib:

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

# Generate random exam scores for four classes
np.random.seed(0)
class_9th_scores = np.random.normal(loc=70, scale=10, size=100)
class_10th_scores = np.random.normal(loc=75, scale=8, size=100)
class_11th_scores = np.random.normal(loc=80, scale=12, size=100)
class_12th_scores = np.random.normal(loc=85, scale=9, size=100)

# Combine scores into a list of lists
all_scores = [class_9th_scores, class_10th_scores, class_11th_scores, class_12th_scores]

# Create a violin plot using Matplotlib
plt.figure(figsize=(10, 6))
plt.violinplot(all_scores, showmeans=False, showmedians=True)
plt.xticks([1, 2, 3, 4], ['9th', '10th', '11th', '12th'])
plt.xlabel('Classes')
plt.ylabel('Exam Scores')
plt.title('Distribution of Exam Scores Across Classes (Matplotlib)')
plt.show()
```



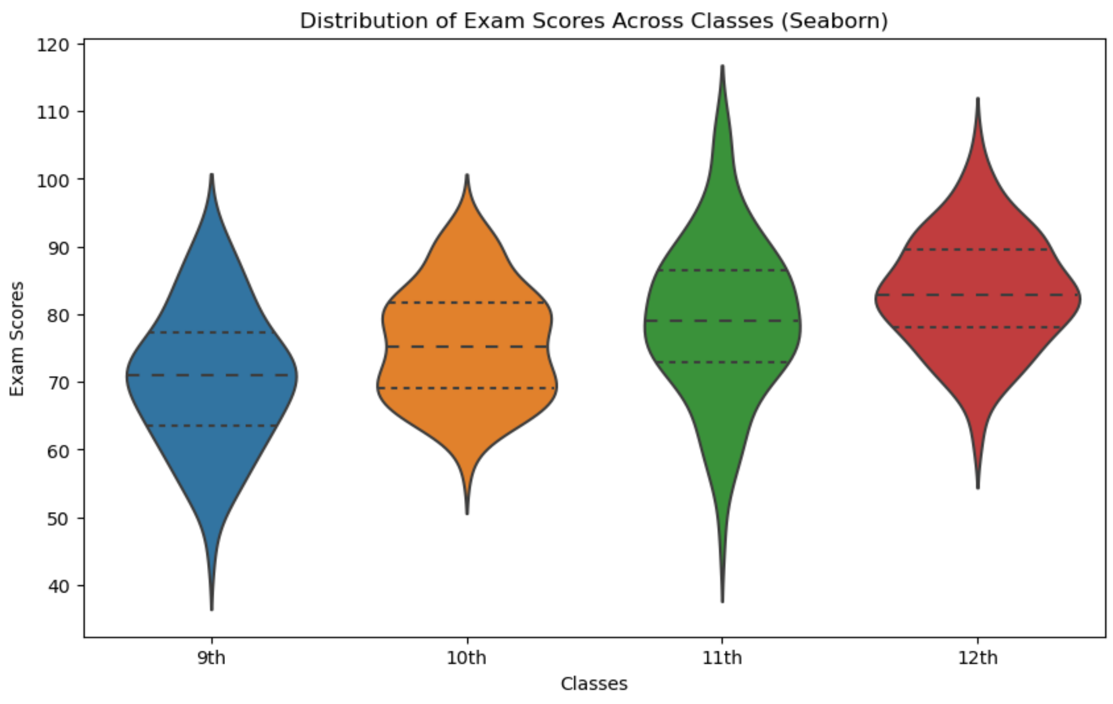
Using Seaborn:

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

# Generate random exam scores for four classes
np.random.seed(0)
class_names = ['9th', '10th', '11th', '12th']
scores = np.concatenate([
    np.random.normal(loc=70, scale=10, size=100),
    np.random.normal(loc=75, scale=8, size=100),
    np.random.normal(loc=80, scale=12, size=100),
    np.random.normal(loc=85, scale=9, size=100)
])
classes = np.repeat(class_names, 100)

# Create a DataFrame to store the data
data = pd.DataFrame({'Class': classes, 'Exam Scores': scores})

# Create a violin plot using Seaborn
plt.figure(figsize=(10, 6))
sns.violinplot(x='Class', y='Exam Scores', data=data, inner='quartile')
plt.xlabel('Classes')
plt.ylabel('Exam Scores')
plt.title('Distribution of Exam Scores Across Classes (Seaborn)')
plt.show()
```



UNIQUE PLOTS IN MATPLOTLIB

PIE CHART:

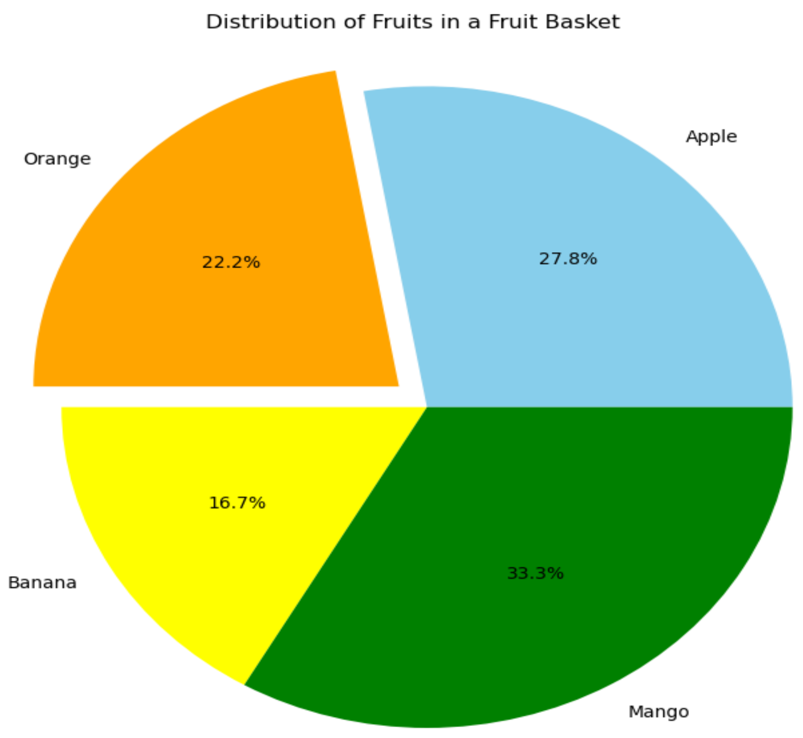
- Represents data as sectors of a circle, where each sector's area is proportional to the data's value.
- Effective for showing proportions or percentages of a whole.

Real Time Scenario:

Suppose we want to visualize the distribution of different types of fruits in a fruit basket using a pie chart.

```python
import matplotlib.pyplot as plt

# Categorical data: Types of fruits and corresponding quantities
fruits = ['Apple', 'Orange', 'Banana', 'Mango']
quantities = [25, 20, 15, 30]  # Quantities of each fruit
explode = (0,0.1,0,0)
# Create a pie chart using Matplotlib
plt.figure(figsize=(8, 8))
plt.pie(quantities, labels=fruits,explode=explode, autopct='%1.1f%%',
        colors=['skyblue', 'orange', 'yellow', 'green'])
plt.title('Distribution of Fruits in a Fruit Basket')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```



3D PLOTTING (E.G., 3D SCATTER PLOT, SURFACE PLOT):

- Visualizes three-dimensional data using three axes (x, y, z) to represent data points.
- Suitable for exploring complex relationships and spatial data.

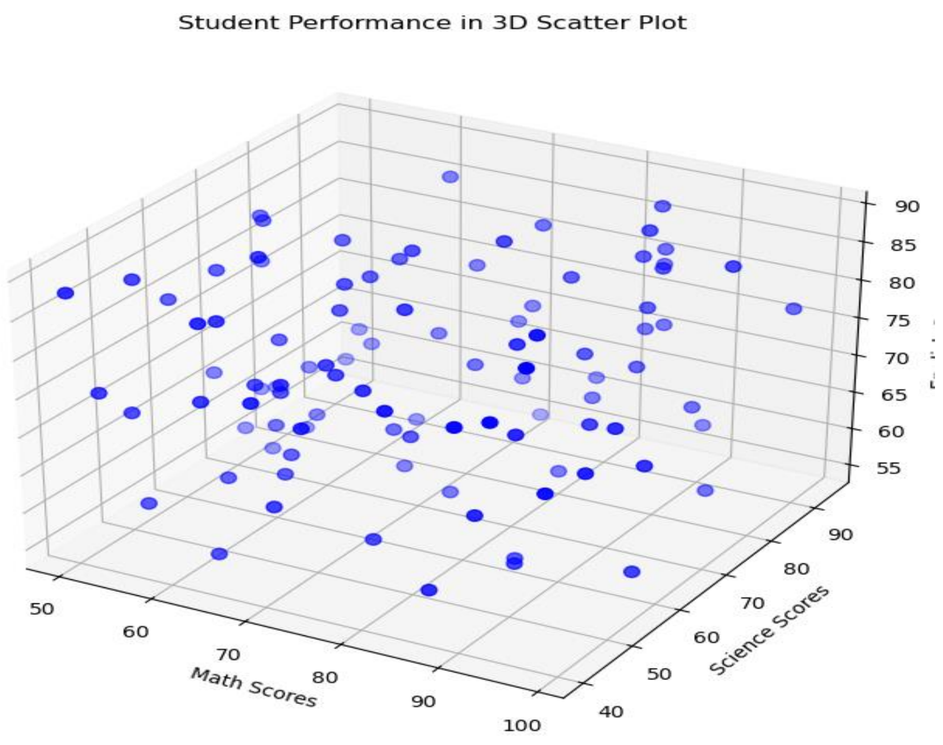
Real Time Scenario:

Suppose we have a dataset representing the performance of students in three subjects (Math, Science, English), and we want to visualize these students' scores in a 3D scatter plot.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Generate synthetic student performance data (example)
np.random.seed(0)
num_students = 100
math_scores = np.random.randint(50, 100, size=num_students)
science_scores = np.random.randint(40, 95, size=num_students)
english_scores = np.random.randint(55, 90, size=num_students)
# Create a 3D scatter plot of student scores using Matplotlib
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
# Scatter plot with three dimensions: Math, Science, English scores
ax.scatter(math_scores, science_scores, english_scores, c='blue', marker='o', s=50)

# Set labels for each axis
ax.set_xlabel('Math Scores')
ax.set_ylabel('Science Scores')
ax.set_zlabel('English Scores')
ax.set_title('Student Performance in 3D Scatter Plot')
plt.show()
```



ERROR BARS:

- Represents uncertainty or variability in data by displaying error bars around data points or bars.
- Useful for showing the precision of measurements or model predictions.

Real Time Scenarios:

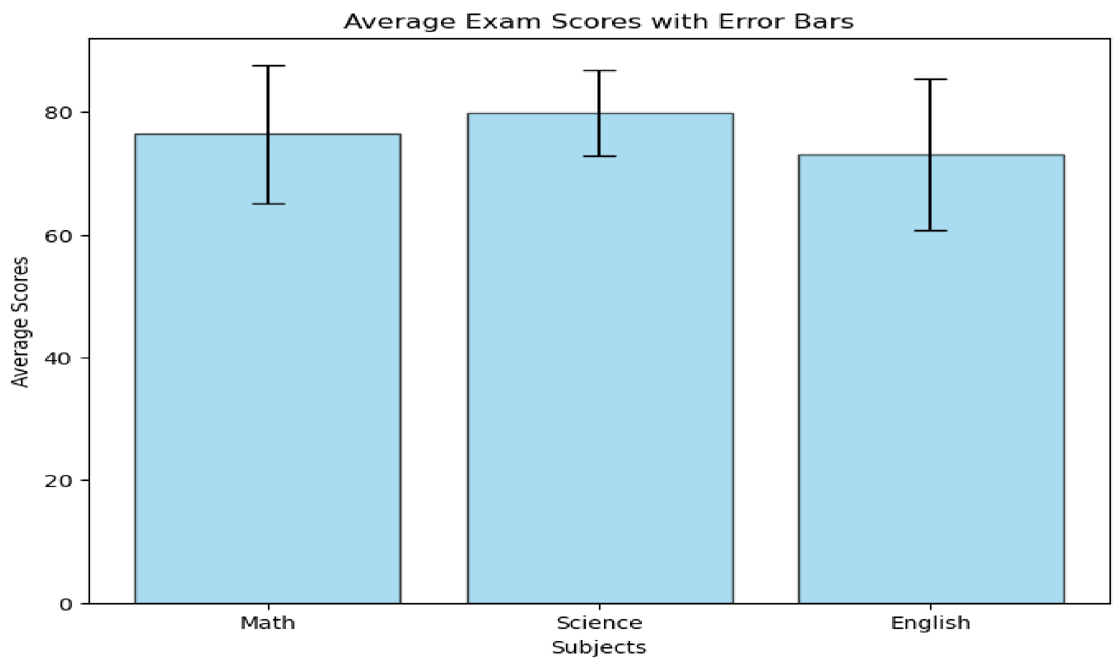
Suppose we want to visualize the average exam scores of students along with the standard deviation as error bars.

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

# Generate synthetic data for exam scores
np.random.seed(0)
scores = {
    'Math': np.random.normal(loc=75, scale=10, size=50),
    'Science': np.random.normal(loc=80, scale=8, size=50),
    'English': np.random.normal(loc=70, scale=12, size=50)
}

# Calculate mean scores and standard deviations
mean_scores = {subject: np.mean(data) for subject, data in scores.items()}
std_scores = {subject: np.std(data) for subject, data in scores.items()}

# Plot error bars for average exam scores
plt.figure(figsize=(8, 6))
plt.bar(mean_scores.keys(), mean_scores.values(), yerr=std_scores.values(),
        alpha=0.7, color='skyblue', edgecolor='black', capsize=7)
plt.xlabel('Subjects')
plt.ylabel('Average Scores')
plt.title('Average Exam Scores with Error Bars')
plt.show()
```



UNIQUE PLOTS IN SEABORN

KERNAL DENSITY ESTIMATION PLOT:

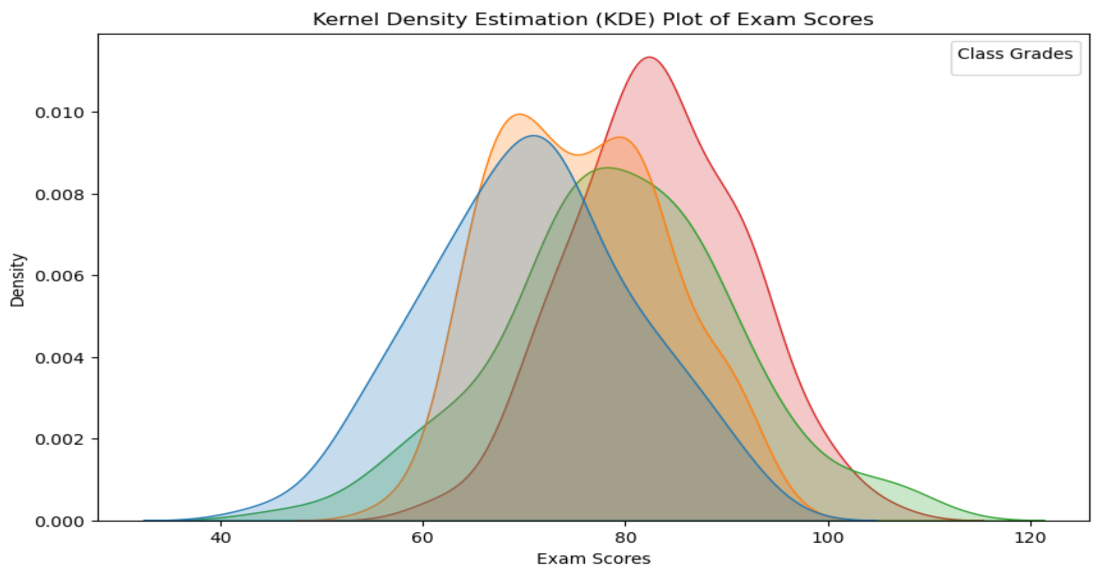
- KDE plots provide a smooth and continuous estimate of the underlying probability density function (PDF) of a dataset.
- They are particularly useful for visualizing the distribution of data, especially when the data may not be well-represented by traditional histogram bins.

Real Time Scenario:

Suppose we have exam scores of students from different classes, and we want to visualize the distribution of these scores using a KDE plot.

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

# Generate synthetic exam scores for different classes
np.random.seed(0)
scores = {
    '9th Grade': np.random.normal(loc=70, scale=10, size=100),
    '10th Grade': np.random.normal(loc=75, scale=8, size=100),
    '11th Grade': np.random.normal(loc=80, scale=12, size=100),
    '12th Grade': np.random.normal(loc=85, scale=9, size=100)}
# Plot KDE plot using Seaborn
plt.figure(figsize=(10, 6))
sns.kdeplot(data=scores, shade=True)
plt.title('Kernel Density Estimation (KDE) Plot of Exam Scores')
plt.xlabel('Exam Scores')
plt.ylabel('Density')
plt.legend(title='Class Grades')
plt.show()
```



PAIR PLOT:

- Pair plots provide a comprehensive visualization of pairwise relationships between multiple variables in a dataset.
- Scatter plots are used to display numeric vs. numeric relationships, while histograms show the distribution of each variable along the diagonal.

Real Time Scenario:

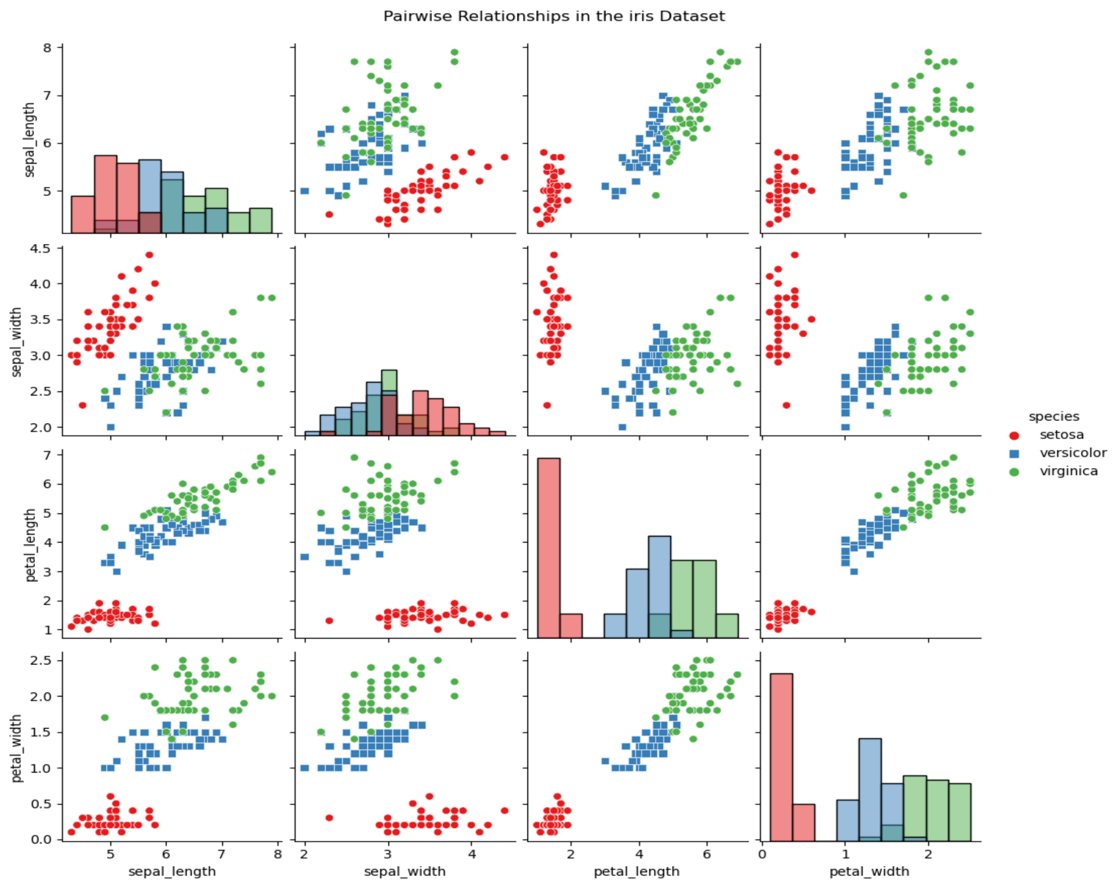
Suppose we want to visualize the pair wise relationship between iris dataset we can use pair plot.

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

# Load the tips dataset from Seaborn
iris = sns.load_dataset('iris')

# Display the first few rows of the dataset
print(iris.head())

# Create a pair plot to visualize pairwise relationships
sns.pairplot(iris, hue='species', palette='Set1', markers=['o', 's'], diag_kind='hist')
plt.suptitle('Pairwise Relationships in the iris Dataset', y=1.02)
plt.show()
```



JOINT PLOT:

- Joint plots in Seaborn are used to visualize the relationship between two numeric variables by combining scatter plots, histograms, and kernel density estimates (KDE) in a single plot.
- They provide insights into the joint distribution and correlation between variables, along with individual variable distributions.

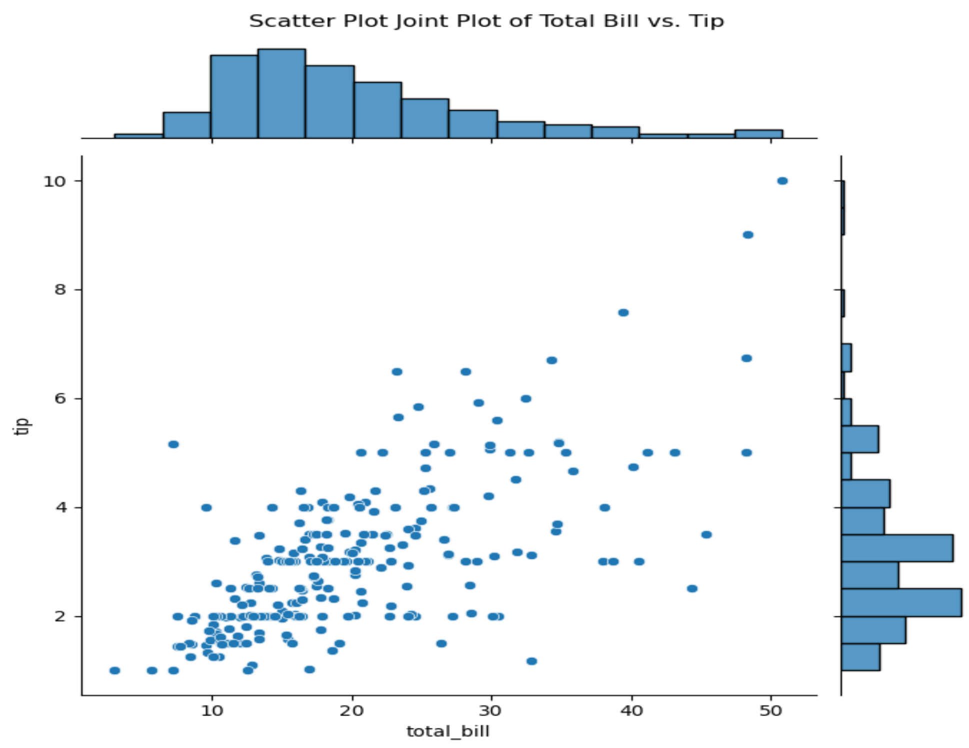
Real Time Scenario:

Suppose we want to visualize the relationship between tips dataset we can use joint plot.

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

# Load the tips dataset from Seaborn
tips = sns.load_dataset('tips')

# Create a scatter plot joint plot to visualize relationship between 'total_bill' and 'tip'
sns.jointplot(data=tips, x='total_bill', y='tip', kind='scatter', height=7)
plt.suptitle('Scatter Plot Joint Plot of Total Bill vs. Tip', y=1.02)
plt.show()
```



COMPARISON OF MATPLOTLIB WITH SEABORN

ASPECTS	MATPLOTLIB	SEABORN
Ease of use	Offers granular control but has a steeper learning curve	Designed for simplicity with high-level plotting functions
Customization	Extensive customization options with low-level APIs	Convenient wrappers for common tasks
Interactivity	Limited native interactivity, requires external libraries	Basic interactivity supported, integrates with Matplotlib
Performance	Efficient rendering, can handle large datasets with care	Optimal for medium-sized datasets, integrates with Pandas

CONCLUSION:

- **Ease of Use:** Seaborn is designed for simplicity, whereas Matplotlib offers more control but requires more code.
- **Customization:** Seaborn provides easy customization for common tasks, while Matplotlib allows extensive customization at a lower level.
- **Interactivity:** Seaborn supports basic interactivity, while Matplotlib requires additional libraries for advanced interactivity.
- **Performance:** Seaborn is optimized for medium-sized datasets, whereas Matplotlib can handle large datasets efficiently with proper optimization.