SHADOWFOX INTERSHIP

Task Level (Beginner)

Visualization Library Documentation

Introduction to Data Visualization Libraries in Python:

Data visualization is an essential part of data analysis and machine learning. It provides a visual context that helps in understanding data patterns, trends, and insights, which are often difficult to comprehend through raw data alone. Python, being a popular language for data science, offers several powerful libraries for creating a wide variety of visualizations. Here's an introduction to some of the most commonly used data visualization libraries in Python:

1. Matplotlib:

Matplotlib is one of the oldest and most widely used data visualization libraries in Python. It provides a comprehensive set of tools for creating static, animated, and interactive plots. Matplotlib is highly customizable, allowing users to tweak almost every aspect of their plots.

Key Features:

- Extensive customization options.

- Large variety of plot types, including line plots, scatter plots, bar charts, histograms, and 3D plots.

- Supports interactive plots and animations.

- Strong integration with NumPy and pandas.

Typical Use Cases:

- Creating publication-quality figures.

- Developing detailed and highly customized visualizations.

- Plotting basic to advanced charts in scientific research.

2. Seaborn:

Seaborn is built on top of Matplotlib and is known for its high-level interface and attractive statistical graphics. It simplifies the process of creating complex visualizations and is designed to work seamlessly with pandas DataFrames.

Key Features:

- High-level interface for drawing attractive and informative statistical plots.

- Built-in themes for enhancing the aesthetics of plots.

- Integration with pandas for easy data manipulation and visualization.

- Supports complex visualizations like heatmaps, violin plots, and pair plots with minimal code.

Typical Use Cases:

- Quickly generating informative and aesthetically pleasing statistical plots.

- Visualizing distributions and relationships in datasets.

- Enhancing Matplotlib plots with better aesthetics and additional functionality.

3. Plotly:

Plotly is a library for creating interactive, web-based visualizations. It supports a wide range of chart types and can be used to create interactive plots that can be embedded in web applications.

Key Features:

- Interactive plots with zoom, pan, and hover functionalities.

- A wide variety of chart types, including 3D plots, geographical maps, and more.

- Ability to embed plots in web applications and dashboards.

- Integration with Dash, a framework for building analytical web applications.

Typical Use Cases:

- Creating interactive dashboards and web applications.

- Visualizing complex data in a web-based interface.

- Sharing interactive visualizations online.

4. Bokeh

Bokeh is another powerful library for creating interactive visualizations. It allows for the creation of complex, interactive visualizations that can be deployed in web applications.

Key Features:

- Interactive plots with tools for zooming, panning, and selecting data.

- Supports large and streaming datasets.

- Integration with web frameworks like Flask and Django.

- Ability to create custom, interactive web applications.

Typical Use Cases:

- Developing interactive data applications.

- Creating dashboards for real-time data analysis.

- Visualizing large datasets interactively.

5. Pandas

Pandas is a powerful data manipulation and analysis library for Python. It is widely used for data cleaning, preparation, and exploration. While Pandas is primarily a data manipulation library, it also includes built-in plotting capabilities that leverage Matplotlib to provide basic visualization functionalities directly from DataFrames.

Key Features:

-Built-in plotting methods for DataFrames and Series.

-Integration with Matplotlib for enhanced customization.

-Quick and easy generation of plots like line plots, bar charts, histograms, and scatter plots.

-Ideal for exploratory data analysis (EDA).

Typical Use Cases:

-Quickly visualizing data during the data exploration phase.

-Creating simple, quick plots directly from DataFrames without needing to switch to another library.

-Complementing more advanced visualizations with Matplotlib or Seaborn.

Python offers a rich ecosystem of libraries for data visualization, each with its strengths and suitable use cases. Matplotlib and Seaborn are excellent choices for static and publication-quality plots, while Plotly, Bokeh, and Altair provide robust options for creating interactive and web-based visualizations. Choosing the right library depends on your specific needs, the complexity of the visualization, and the level of interactivity required.

1.Guide to matplotlib:

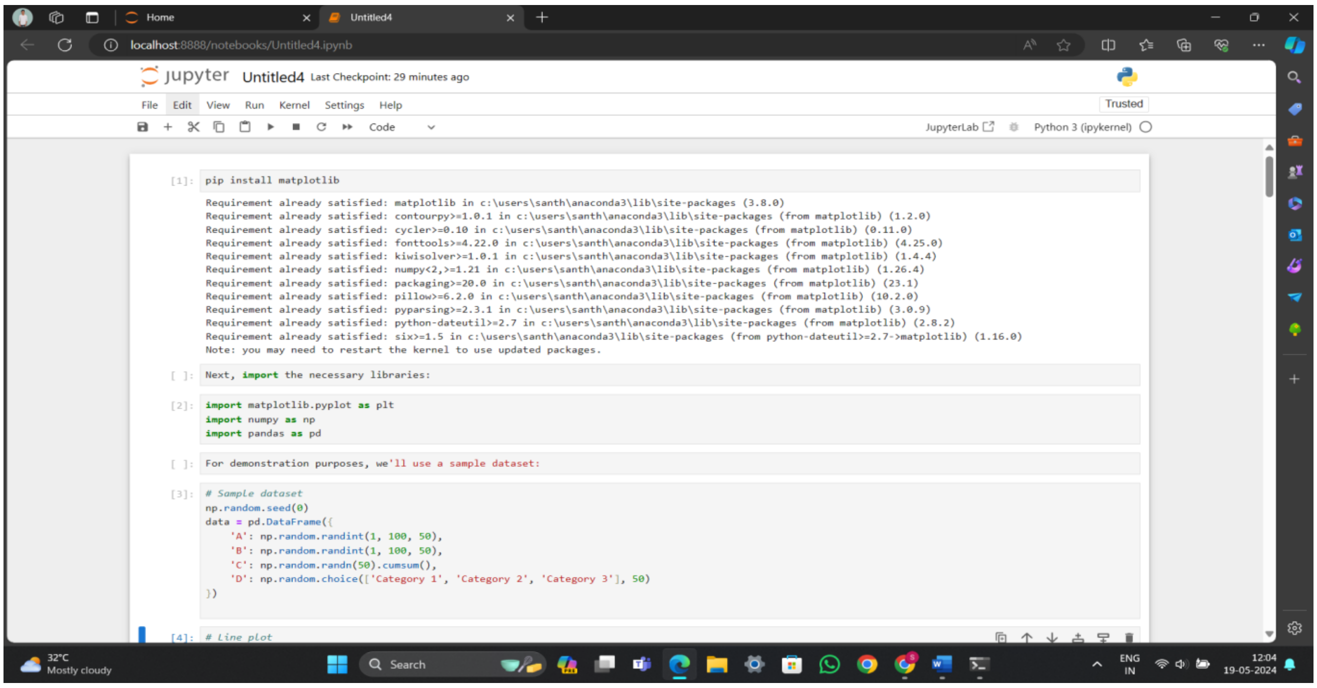
Matplotlib is a versatile library capable of creating a wide range of static, animated, and interactive plots. It is particularly known for its flexibility and extensive customization options. Matplotlib is a powerful Python library for creating static, animated, and interactive visualizations. This guide will provide an overview of the various types of charts and graphs available in Matplotlib, along with examples using a sample dataset.

Setting Up Matplotlib

To get started, you need to install Matplotlib if you haven't already:

pip install matplotlib

Next, import the necessary libraries:


Home
Untitled4
localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 29 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
32°C Mostly cloudy
Search
ENG IN
12:04 19-05-2024

```
[1]: pip install matplotlib

     Requirement already satisfied: matplotlib in c:\users\santh\anaconda3\lib\site-packages (3.8.0)
     Requirement already satisfied: contourpy>=1.0.1 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (1.2.0)
     Requirement already satisfied: cycler>=0.10 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
     Requirement already satisfied: fonttools>=4.22.0 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
     Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
     Requirement already satisfied: numpy<2,>=1.21 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (1.26.4)
     Requirement already satisfied: packaging>=20.0 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (23.1)
     Requirement already satisfied: pillow>=6.2.0 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (10.2.0)
     Requirement already satisfied: pyparsing>=2.3.1 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
     Requirement already satisfied: python-dateutil>=2.7 in c:\users\santh\anaconda3\lib\site-packages (from matplotlib) (2.8.2)
     Requirement already satisfied: six>=1.5 in c:\users\santh\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
     Note: you may need to restart the kernel to use updated packages.

[ ]: Next, import the necessary libraries:

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

[ ]: For demonstration purposes, we'll use a sample dataset:

[3]: # Sample dataset
     np.random.seed(0)
     data = pd.DataFrame({
         'A': np.random.randint(1, 100, 50),
         'B': np.random.randint(1, 100, 50),
         'C': np.random.randn(50).cumsum(),
         'D': np.random.choice(['Category 1', 'Category 2', 'Category 3'], 50)
     })

[4]: # Line plot
```

For demonstration purpose ,we'll use the sample dataset mentioned above in the cell

Basic plots in Matplotlib:

1.Line plot

A line plot is a type of chart used to show information that changes over time. It is one of the most commonly used plots in data visualization and is particularly useful for displaying trends, patterns, and relationships between two continuous variables.

Key Components of a Line Plot

1. **X-Axis (Horizontal Axis)**: Represents the independent variable, which is often time.
2. **Y-Axis (Vertical Axis)**: Represents the dependent variable, which is the variable being measured.
3. **Data Points**: Points plotted on the graph that represent the values of the variables.
4. **Lines**: Lines that connect the data points, showing the trend or pattern in the data.

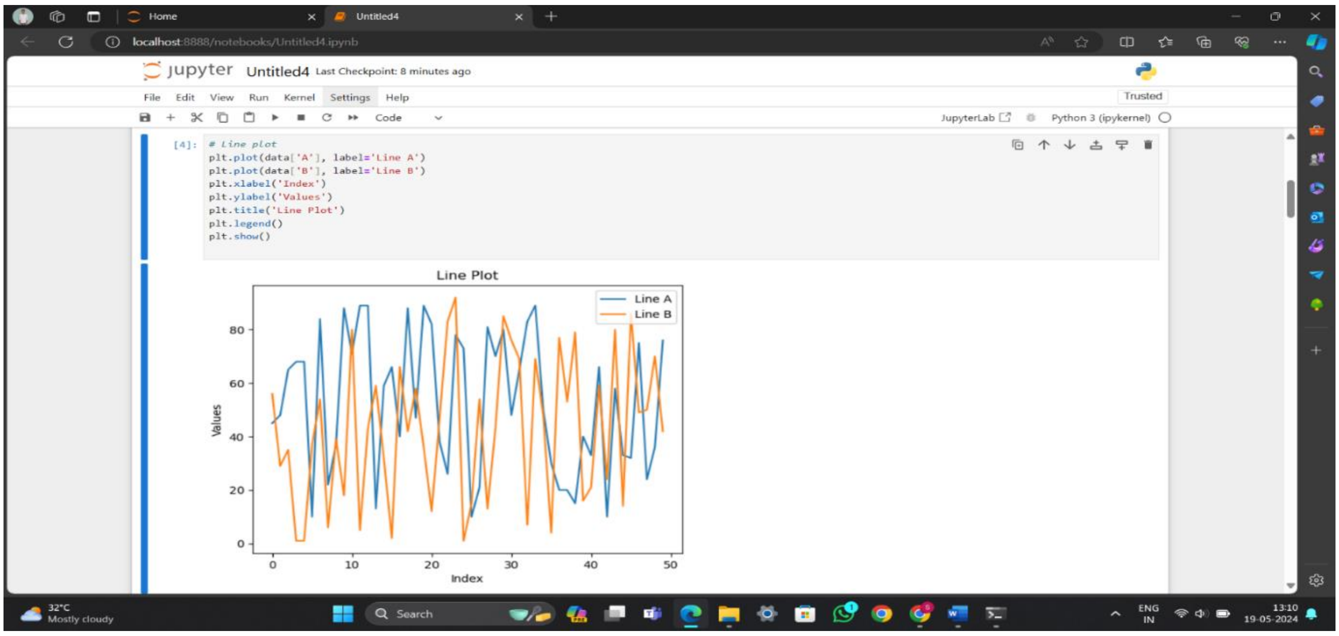
Use Cases

Line plots are ideal for:

- **Time Series Analysis**: Displaying data points in chronological order.
- **Trend Analysis**: Showing how data trends over a period.
- **Comparative Analysis**: Comparing different data sets on the same graph.

Creating a Line Plot in Matplotlib

Let's go through the steps to create and customize a line plot using Matplotlib.



```
[4]: # Line plot
plt.plot(data['A'], label='Line A')
plt.plot(data['B'], label='Line B')
plt.xlabel('Index')
plt.ylabel('Values')
plt.title('Line Plot')
plt.legend()
plt.show()
```

localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 8 minutes ago
File Edit View Run Kernel Settings Help
Code
Trusted
JupyterLab
Python 3 (ipykernel)
Line Plot
Line A
Line B
Values
Index
0
20
40
60
80
0
10
20
30
40
50

2.Scatter Plot:

A scatter plot is a type of data visualization that displays individual data points on a two-dimensional graph. It is particularly useful for identifying relationships, patterns, and trends between two continuous variables. Each point on the scatter plot represents a pair of values from the dataset.

Key Components of a Scatter Plot

1. **X-Axis (Horizontal Axis)**: Represents one of the variables being compared.
2. **Y-Axis (Vertical Axis)**: Represents the other variable being compared.
3. **Data Points**: Each point on the graph corresponds to a pair of values from the two variables.

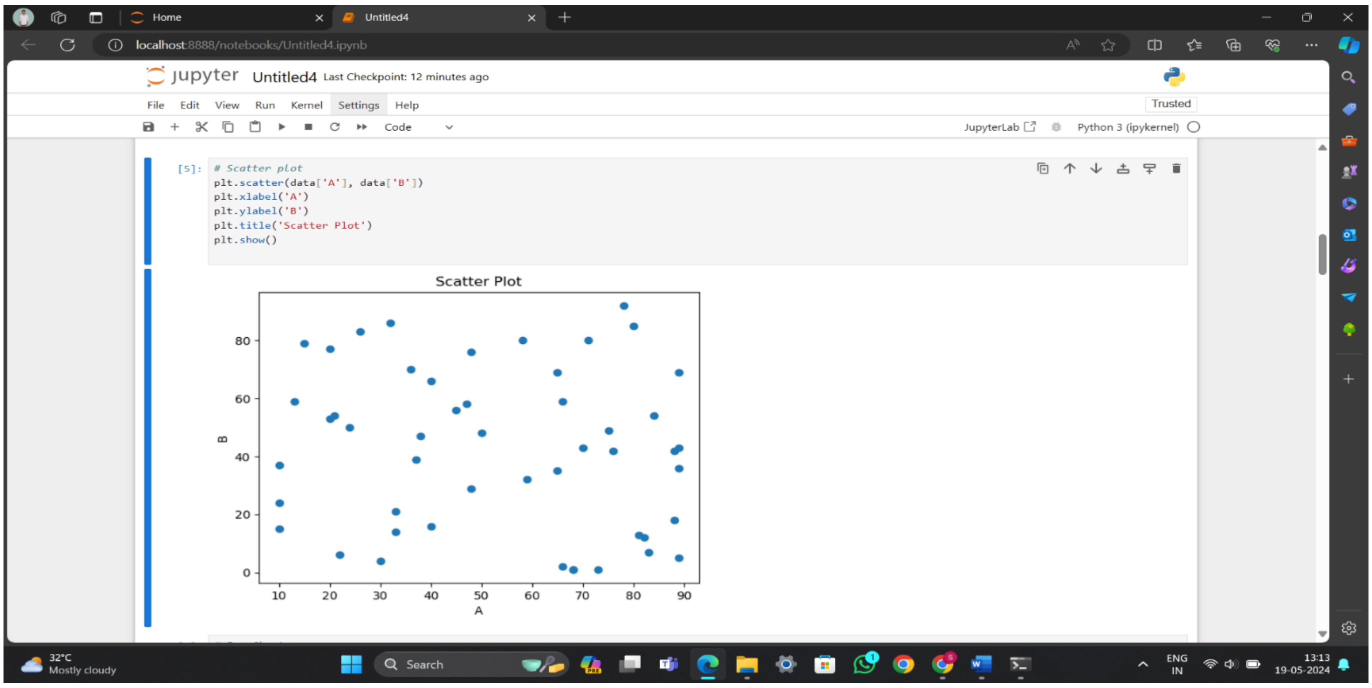
Use Cases

Scatter plots are ideal for:

- **Correlation Analysis**: Determining if there is a relationship between two variables.
- **Outlier Detection**: Identifying data points that deviate significantly from the pattern.
- **Distribution Examination**: Understanding the spread and clustering of data points.

Creating a Scatter Plot in Matplotlib

Let's go through the steps to create and customize a scatter plot using Matplotlib.


localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 12 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Scatter Plot
A
B

```
[5]: # Scatter plot
plt.scatter(data['A'], data['B'])
plt.xlabel('A')
plt.ylabel('B')
plt.title('Scatter Plot')
plt.show()
```

3.Bar Chart:

A bar chart is a type of data visualization that represents categorical data with rectangular bars. The length or height of each bar is proportional to the value it represents. Bar charts are particularly useful for comparing quantities across different categories.

Key Components of a Bar Chart

1. **X-Axis (Horizontal Axis)**: Represents the categories.
2. **Y-Axis (Vertical Axis)**: Represents the values associated with the categories.
3. **Bars**: Rectangular bars representing the values of each category. The height or length of each bar corresponds to the value it represents.

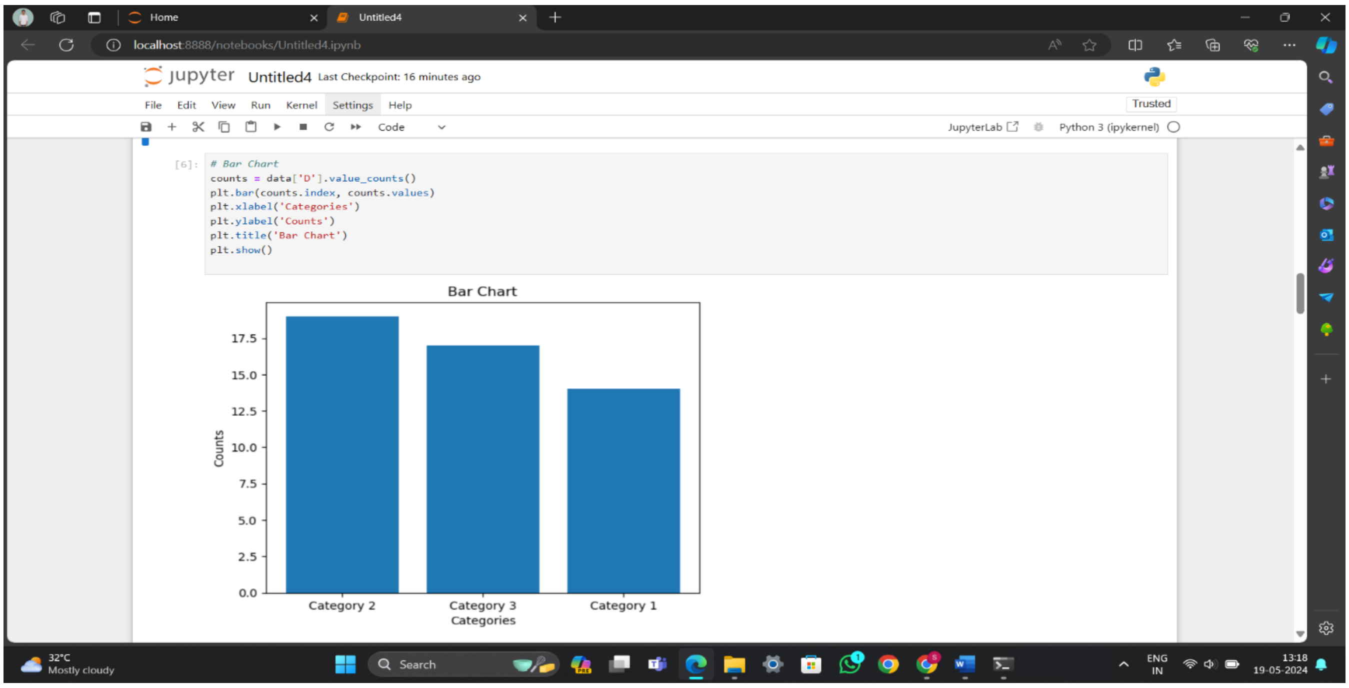
Use Cases

Bar charts are ideal for:

- **Comparing Quantities**: Comparing the size of different categories.
- **Distribution Analysis**: Analyzing the frequency distribution of categorical data.
- **Displaying Data Over Time**: When categories represent different time periods.

Creating a Bar Chart in Matplotlib

Let's go through the steps to create and customize a bar chart using Matplotlib.



```
[6]: # Bar Chart
counts = data['D'].value_counts()
plt.bar(counts.index, counts.values)
plt.xlabel('Categories')
plt.ylabel('Counts')
plt.title('Bar Chart')
plt.show()
```

4.Histogram:

A histogram is a type of data visualization that displays the distribution of a dataset. It divides the data into bins (or intervals) and shows the frequency of data points that fall within each bin. Histograms are useful for understanding the shape of the data distribution, detecting outliers, and observing central tendencies.

Key Components of a Histogram

1. **Bins**: Intervals into which the entire range of data is divided.
2. **X-Axis (Horizontal Axis)**: Represents the bins or intervals.
3. **Y-Axis (Vertical Axis)**: Represents the frequency or count of data points in each bin.
4. **Bars**: Rectangular bars where the height (or length) represents the frequency of data points in each bin.

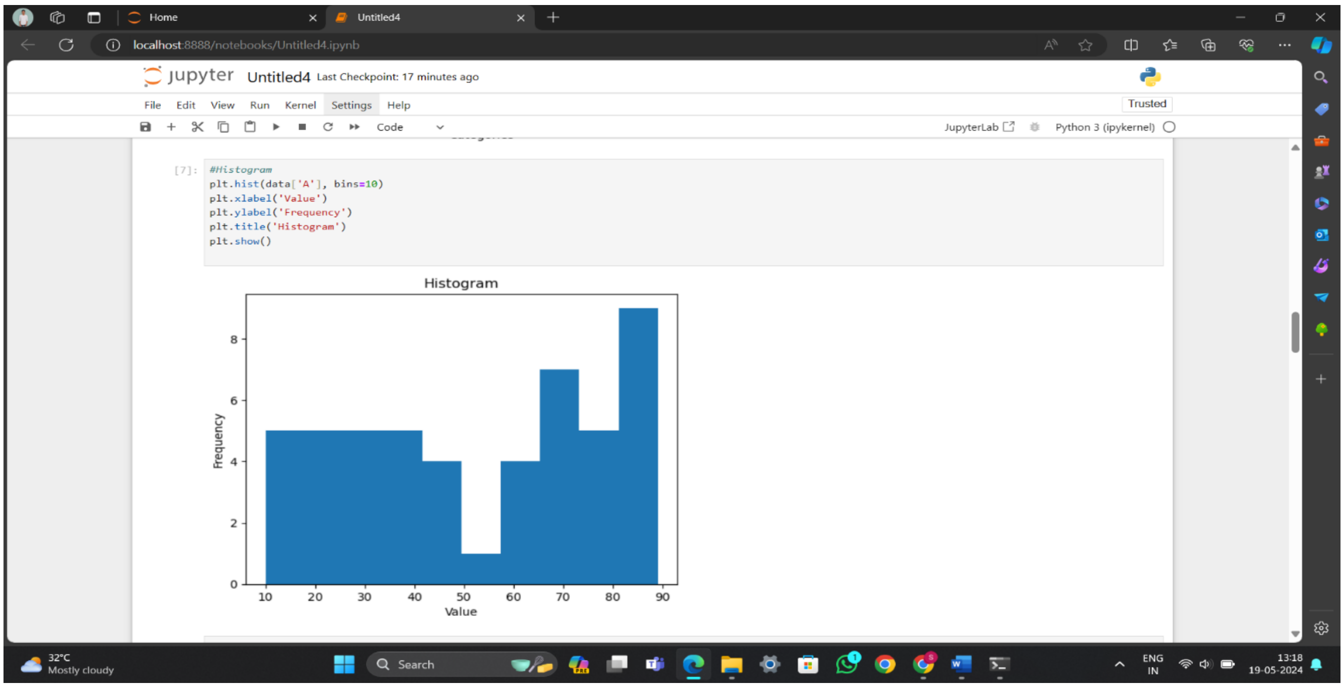
Use Cases

Histograms are ideal for:

- **Distribution Analysis**: Understanding the shape and spread of data.
- **Detecting Outliers**: Identifying data points that fall far from the majority.
- **Central Tendency**: Observing where the data is concentrated.

Creating a Histogram in Matplotlib

Let's go through the steps to create and customize a histogram using Matplotlib.



```
[7]: #Histogram
plt.hist(data['A'], bins=10)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```

5.Pie Chart:

A pie chart is a circular statistical graphic that is divided into slices to illustrate numerical proportions. Each slice of the pie represents a category's contribution to the whole. Pie charts are particularly useful for displaying data that adds up to a whole, such as percentages or proportions.

Key Components of a Pie Chart

1. **Slices**: The sectors of the pie, each representing a category.
2. **Labels**: Text annotations that describe each slice.
3. **Colors**: Different colors to distinguish between categories.

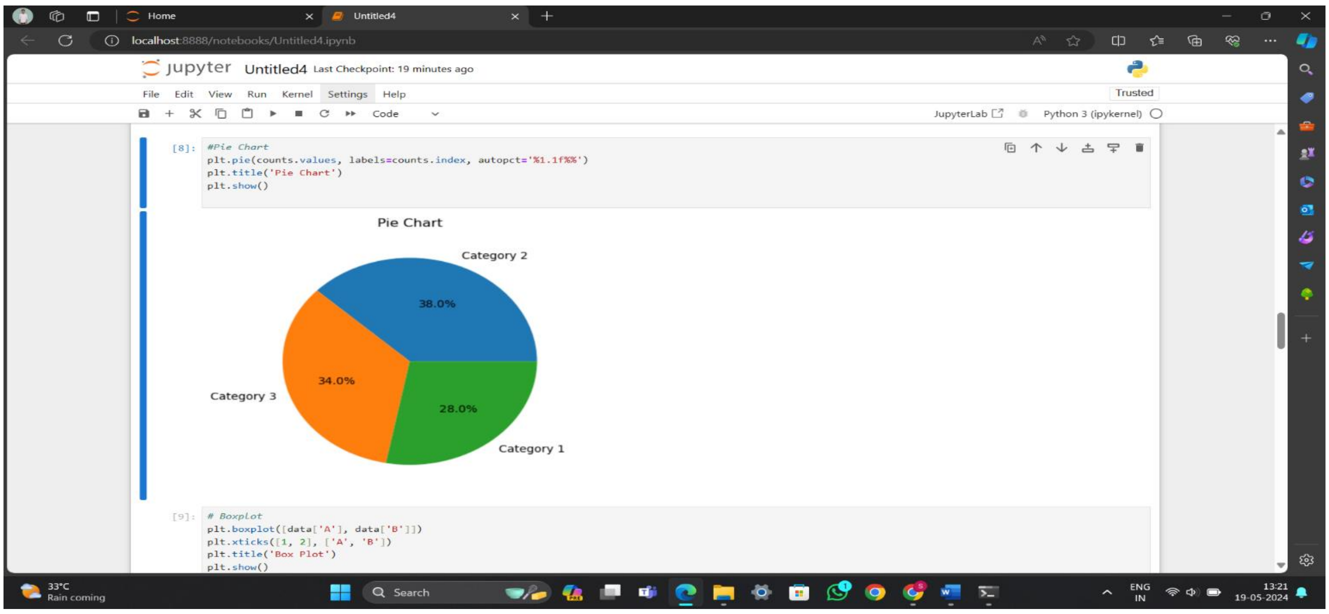
Use Cases

Pie charts are ideal for:

- **Proportion Analysis**: Showing how different parts make up a whole.
- **Comparing Categories**: Visually comparing the relative sizes of different categories.
- **Displaying Percentages**: Illustrating the percentage distribution of categories.

Creating a Pie Chart in Matplotlib

Let's go through the steps to create and customize a pie chart using Matplotlib.


localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 19 minutes ago
File Edit View Run Kernel Settings Help
Code
Trusted
JupyterLab
Python 3 (ipykernel)
Pie Chart
Category 2
38.0%
34.0%
Category 3
28.0%
Category 1

```
[8]: #Pie Chart
plt.pie(counts.values, labels=counts.index, autopct='%1.1f%%')
plt.title('Pie Chart')
plt.show()
```

```
[9]: # Boxplot
plt.boxplot([data['A'], data['B']])
plt.xticks([1, 2], ['A', 'B'])
plt.title('Box Plot')
plt.show()
```

Advanced Plots in Matplotlib:

1.Box Plot:

A box plot, also known as a box-and-whisker plot, is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum. It is useful for identifying outliers and understanding the spread and skewness of the data.

Key Components of a Box Plot

1. **Box**: Represents the interquartile range (IQR), which contains the middle 50% of the data.
2. **Whiskers**: Extend from the box to the minimum and maximum values within 1.5 * IQR from the quartiles.
3. **Median Line**: A line inside the box that represents the median (Q2).
4. **Outliers**: Data points that fall outside the range of the whiskers.

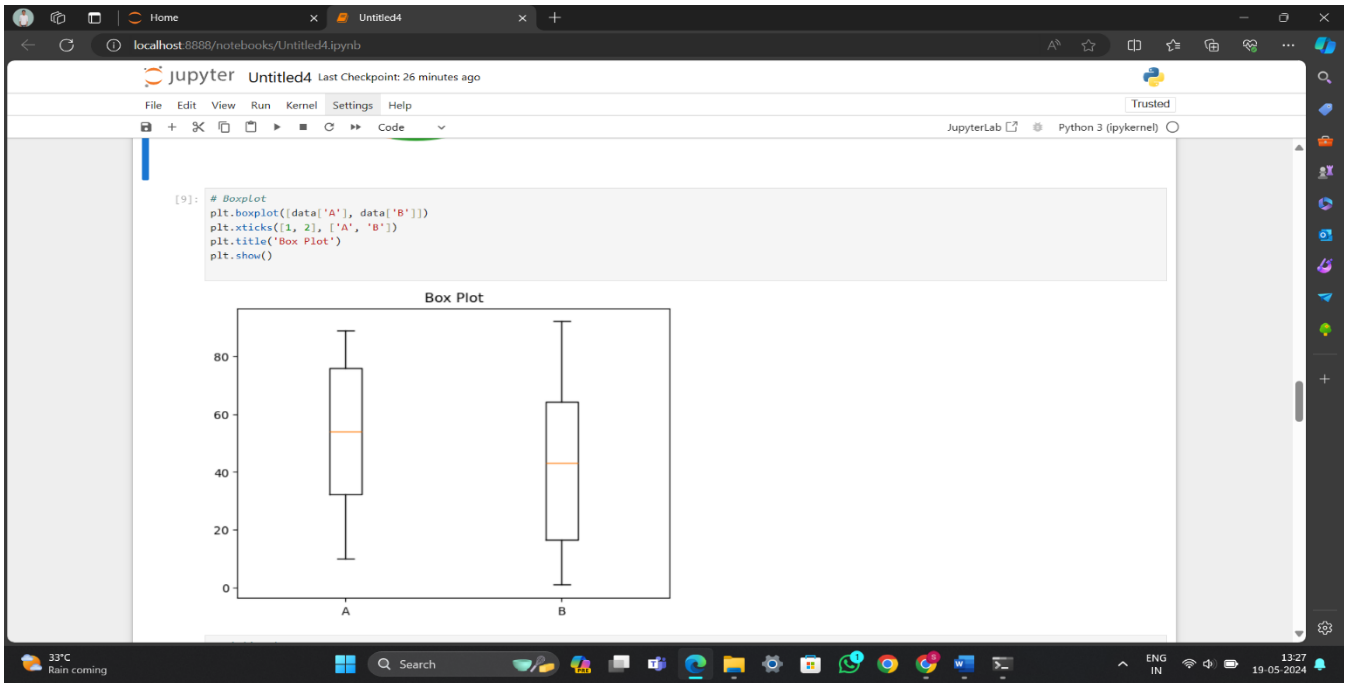
Use Cases

Box plots are ideal for:

- **Comparing Distributions**: Comparing the distribution of data across different categories.
- **Identifying Outliers**: Detecting outliers in the data.
- **Understanding Spread and Skewness**: Visualizing the spread and central tendency of the data.

Creating a Box Plot in Matplotlib

Let's go through the steps to create and customize a box plot using Matplotlib.



```
[9]: # Boxplot
plt.boxplot([data['A'], data['B']])
plt.xticks([1, 2], ['A', 'B'])
plt.title('Box Plot')
plt.show()
```

2.Violin Plot:

A violin plot is a method of plotting numeric data and can be understood as a combination of a box plot and a kernel density plot. It shows the distribution of the data across different categories, similar to a box plot, but also includes a rotated density plot on each side. This provides a deeper understanding of the distribution shape, especially for multimodal data.

Key Components of a Violin Plot

1. **Violin Body**: Represents the kernel density estimate of the data distribution.
2. **Inner Box Plot**: Shows the summary statistics (median, quartiles) within the violin.
3. **Bandwidth**: Controls the smoothness of the density estimate.

Use Cases

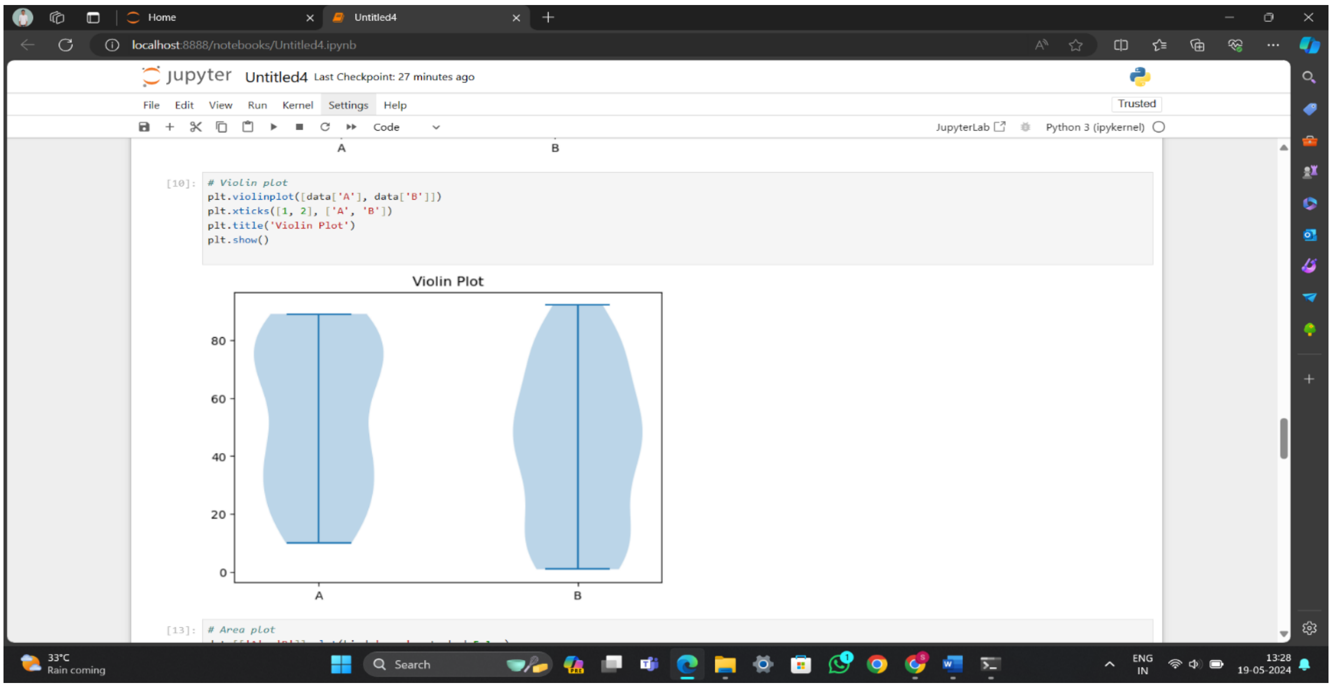
Violin plots are ideal for:

- **Comparing Distributions**: Comparing the distribution of data across multiple categories.

- **Identifying Data Shape**: Understanding the underlying distribution shape, including multimodal distributions.
- **Displaying Density**: Showing how the density of data varies.

Creating a Violin Plot in Matplotlib

Let's go through the steps to create and customize a violin plot using Matplotlib.



```
# Violin plot
plt.violinplot([data['A'], data['B']])
plt.xticks([1, 2], ['A', 'B'])
plt.title('Violin Plot')
plt.show()
```

localhost:8888/notebooks/Untitled4.ipynb
Untitled4 Last Checkpoint: 27 minutes ago
Trusted
Python 3 (ipykernel)
Violin Plot
80
60
40
20
0
A
B
# Area plot

3.Area plot:

An area plot, also known as a filled line plot or a stacked line plot, is a type of data visualization that displays quantitative data over a continuous interval or time period. It is similar to a line plot but with the area between the line and the x-axis filled with color. Area plots are useful for showing the trend of multiple variables and their cumulative contribution to the whole.

Key Components of an Area Plot

1. **X-Axis (Horizontal Axis)**: Represents the independent variable, typically time or a continuous interval.
2. **Y-Axis (Vertical Axis)**: Represents the dependent variable, which can be a single variable or a combination of variables.

3. **Filled Area**: The space between the line and the x-axis is filled with color, representing the cumulative contribution of the variables.

Use Cases

Area plots are ideal for:

- **Trend Analysis**: Showing how variables change over time or across different intervals.
- **Comparing Contributions**: Comparing the contribution of different variables to the whole.
- **Displaying Cumulative Data**: Visualizing cumulative data trends.

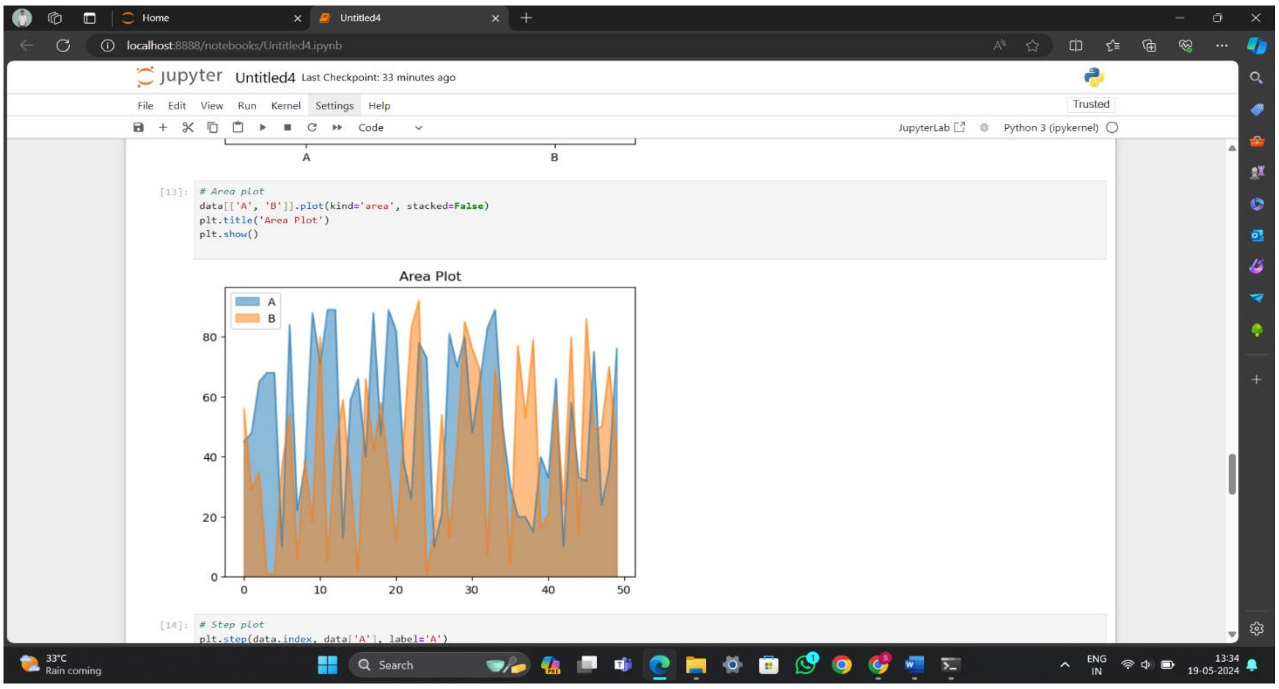
Creating an Area Plot in Matplotlib

Let's go through the steps to create and customize an area plot using Matplotlib.

```
data[['A', 'B']].plot(kind='area', stacked=False)

plt.title('Area Plot')

plt.show()
```



```
[13]: # Area plot
      data[['A', 'B']].plot(kind='area', stacked=False)
      plt.title('Area Plot')
      plt.show()
```

Home
Untitled4
localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 33 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
A
B
Area Plot
A
B
80
60
40
20
0
0
10
20
30
40
50
33°C
Rain coming
Search
ENG
IN
13:34
19-05-2024

```
[14]: # Step plot
      plt.step(data.index, data['A'], label='A')
```

4.Step plot:

A step plot, also known as a step function or a stair-step plot, is a type of data visualization that is used to represent discrete data points with a series of horizontal or vertical lines. Step plots are commonly used to visualize data that changes abruptly at discrete intervals or to display cumulative data.

Key Components of a Step Plot

1. **Data Points**: Discrete data points represented by markers.
2. **Step Lines**: Horizontal or vertical lines connecting the data points.
3. **Axes**: Horizontal and vertical axes representing the independent and dependent variables.

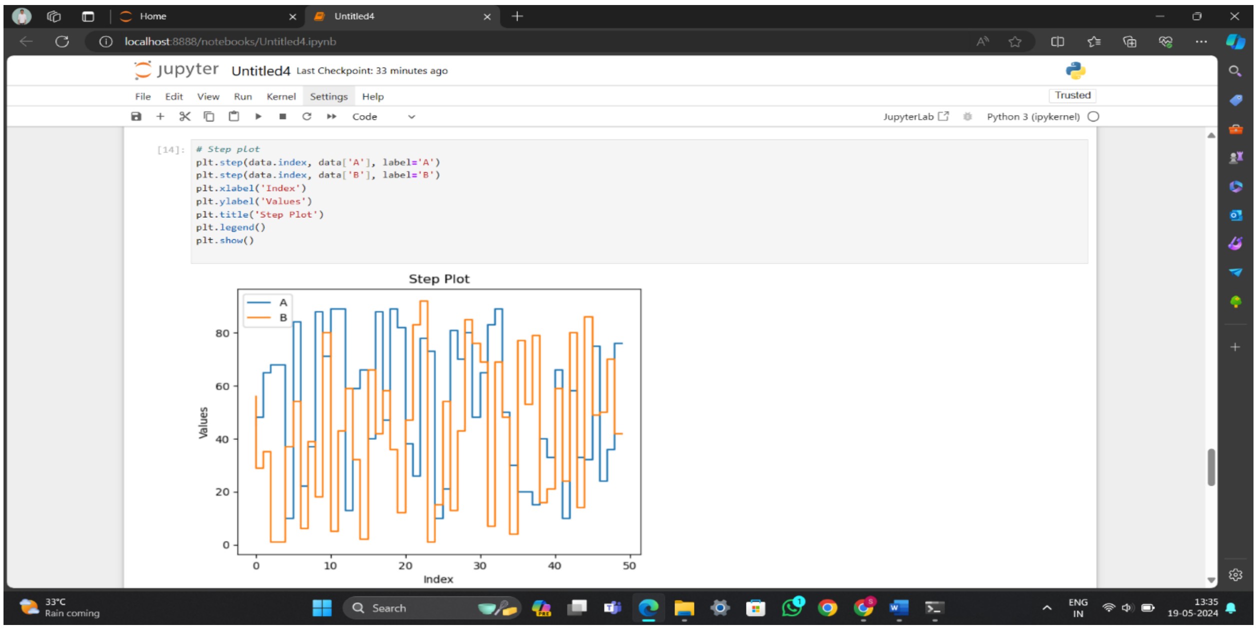
Use Cases

Step plots are ideal for:

- **Visualizing Discrete Data**: Representing data that changes abruptly at specific intervals.
- **Cumulative Data**: Showing cumulative trends or distributions.
- **Time Series Data**: Displaying time series data with discrete intervals.

Creating a Step Plot in Matplotlib

Let's go through the steps to create and customize a step plot using Matplotlib.



```
[14]: # Step plot
plt.step(data.index, data['A'], label='A')
plt.step(data.index, data['B'], label='B')
plt.xlabel('Index')
plt.ylabel('Values')
plt.title('Step Plot')
plt.legend()
plt.show()
```

5.3D Scatter Plot:

A 3D scatter plot is a type of data visualization that represents three-dimensional data points in a Cartesian coordinate system. It is used to visualize relationships between three variables and is particularly useful for exploring complex datasets with multiple dimensions.

Key Components of a 3D Scatter Plot

1. **X-Axis, Y-Axis, Z-Axis**: Represents the three dimensions of the data.
2. **Data Points**: Individual points in the three-dimensional space, each representing a combination of the three variables.
3. **Markers**: Symbols or shapes used to represent data points.

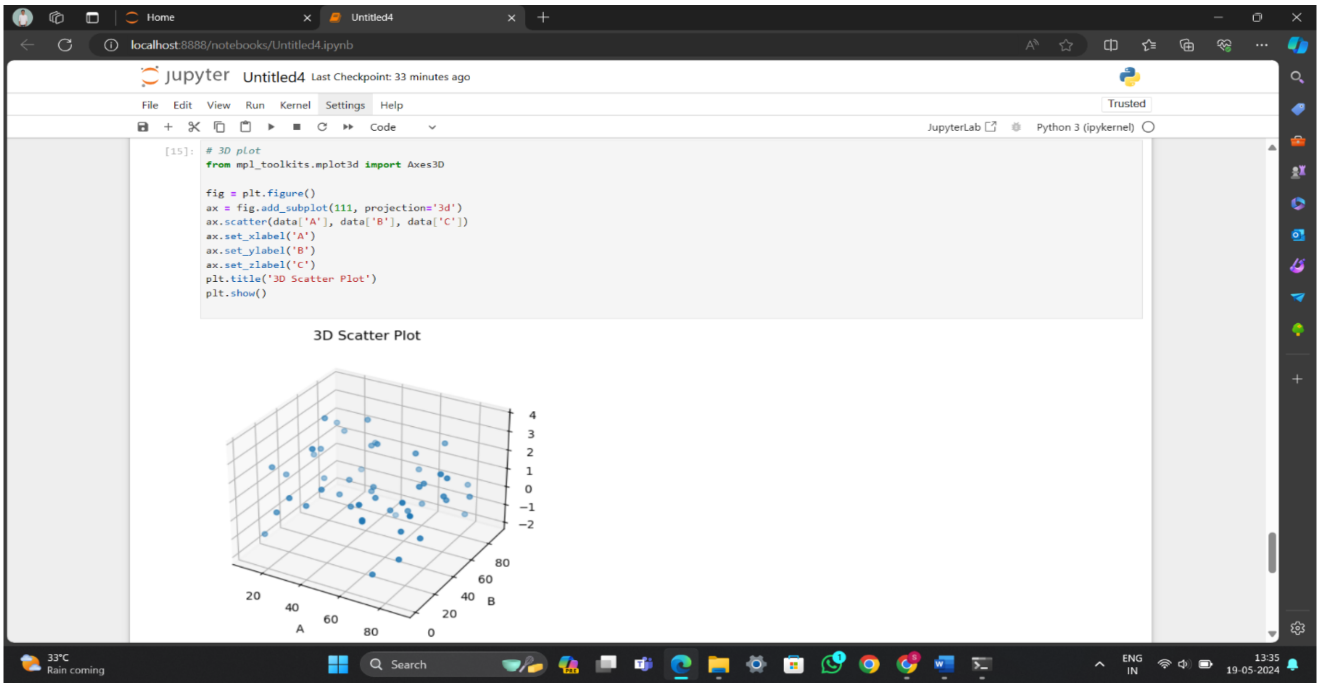
Use Cases

3D scatter plots are ideal for:

- **Exploring Multivariate Relationships**: Visualizing relationships between three variables simultaneously.
- **Identifying Clusters**: Detecting clusters or patterns in multidimensional datasets.
- **Understanding Spatial Data**: Analyzing spatial relationships between variables.

Creating a 3D Scatter Plot in Matplotlib

Let's go through the steps to create and customize a 3D scatter plot using Matplotlib.



```
[15]: # 3D plot
      from mpl_toolkits.mplot3d import Axes3D

      fig = plt.figure()
      ax = fig.add_subplot(111, projection='3d')
      ax.scatter(data['A'], data['B'], data['C'])
      ax.set_xlabel('A')
      ax.set_ylabel('B')
      ax.set_zlabel('C')
      plt.title('3D Scatter Plot')
      plt.show()
```

2.Guide to Seaborn:

Seaborn is a Python data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. It simplifies the process of creating complex visualizations by providing easy-to-use functions with built-in themes and color palettes. Seaborn is particularly well-suited for exploring and visualizing relationships in datasets, making it an essential tool for data analysis and exploration.

Key Features of Seaborn:

1. **High-Level Interface**: Seaborn provides a concise and intuitive API for creating a wide range of statistical plots with minimal code.
2. **Built-in Themes and Styles**: Seaborn comes with built-in themes and styles that enhance the aesthetics of plots, making them visually appealing.
3. **Statistical Estimation**: Seaborn provides functions for statistical estimation and aggregation, allowing users to visualize data distributions and relationships.
4. **Color Palettes**: Seaborn offers a variety of color palettes for customizing the appearance of plots and enhancing their interpretability.
5. **Integration with Pandas**: Seaborn seamlessly integrates with Pandas DataFrames, making it easy to work with structured datasets.

Graph Types in Seaborn

Seaborn provides a wide range of statistical plots for visualizing data distributions, relationships, and patterns. Some of the commonly used graph types in Seaborn include:

1. **Distribution Plots**: Visualize the distribution of univariate or bivariate data.
2. **Categorical Plots**: Compare distributions across categorical variables.
3. **Relational Plots**: Explore relationships between variables.
4. **Regression Plots**: Visualize linear regression models and explore relationships between variables.
5. **Matrix Plots**: Display matrices and heatmaps to explore correlations and patterns in data.
6. **Time Series Plots**: Analyze time series data and trends over time.

1.Distribution Plots:

A distribution plot, also known as a histogram, is a type of data visualization that displays the distribution of a univariate or bivariate dataset. It provides a graphical representation of the frequency or probability distribution of numerical data. Distribution plots are particularly useful for understanding the shape, spread, and central tendency of the data.

Key Components of a Distribution Plot

1. **Bins**: Intervals into which the entire range of data is divided.
2. **X-Axis (Horizontal Axis)**: Represents the bins or intervals.
3. **Y-Axis (Vertical Axis)**: Represents the frequency, probability density, or cumulative density of data points in each bin.
4. **Bars**: Rectangular bars where the height (or length) represents the frequency, probability density, or cumulative density of data points in each bin.

Use Cases

Distribution plots are ideal for:

- **Understanding Data Distribution**: Visualizing how data is distributed across different intervals.
- **Identifying Central Tendency**: Observing where the data is concentrated and identifying outliers.
- **Comparing Distributions**: Comparing the distribution of multiple datasets.

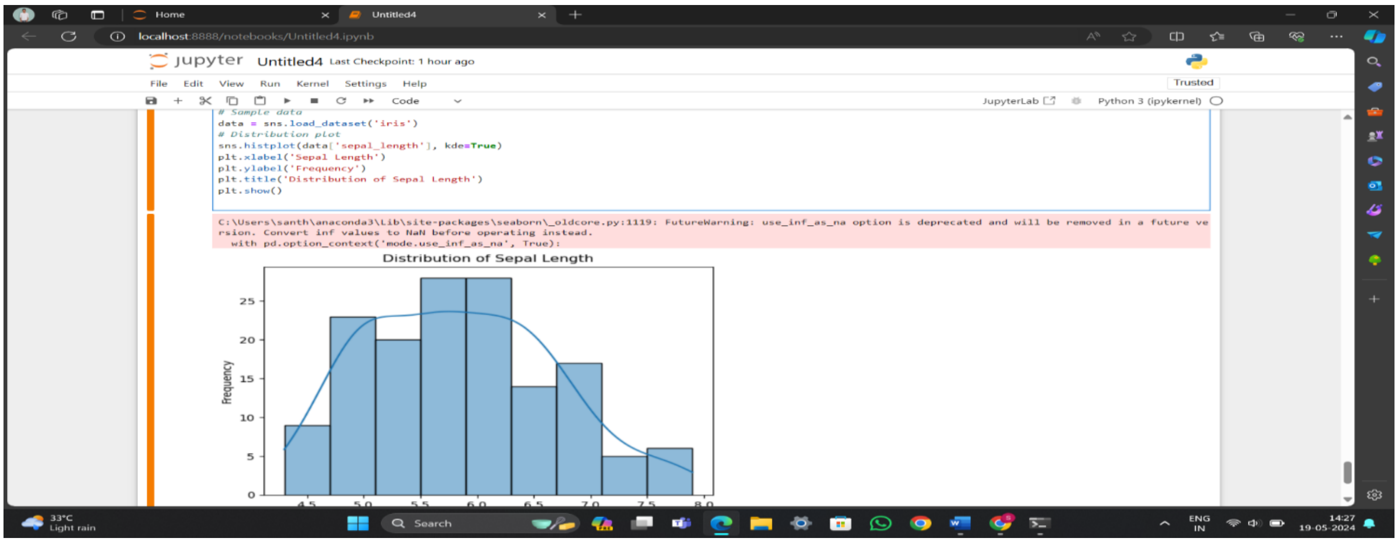
Creating a Distribution Plot in Seaborn

Let's go through the steps to create and customize a distribution plot using Seaborn.

import the necessary libraries:

import seaborn as sns

import matplotlib.pyplot as plt



```
# Sample data
data = sns.load_dataset('iris')
# Distribution plot
sns.histplot(data['sepal_length'], kde=True)
plt.xlabel('Sepal Length')
plt.ylabel('Frequency')
plt.title('Distribution of Sepal Length')
plt.show()
```

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

Untitled4
localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Distribution of Sepal Length
Frequency
25
20
15
10
5
0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
33°C Light rain
Search
ENG IN
14:27 19-05-2024

2.Categorical Plots:

A categorical plot in Seaborn is a type of data visualization used to compare the distribution of data across different categories. It provides a visual summary of categorical data by displaying summary statistics such as the mean, median, or count for each category. Categorical plots are particularly useful for exploring relationships between categorical variables or comparing distributions across different groups.

Key Components of a Categorical Plot

1. **Categories**: The distinct groups or levels of the categorical variable being analyzed.
2. **X-Axis (Horizontal Axis)**: Represents the categories or groups being compared.
3. **Y-Axis (Vertical Axis)**: Represents the summary statistic being visualized, such as mean, median, or count.
4. **Bars, Points, or Boxes**: Visual elements used to represent the summary statistic for each category.

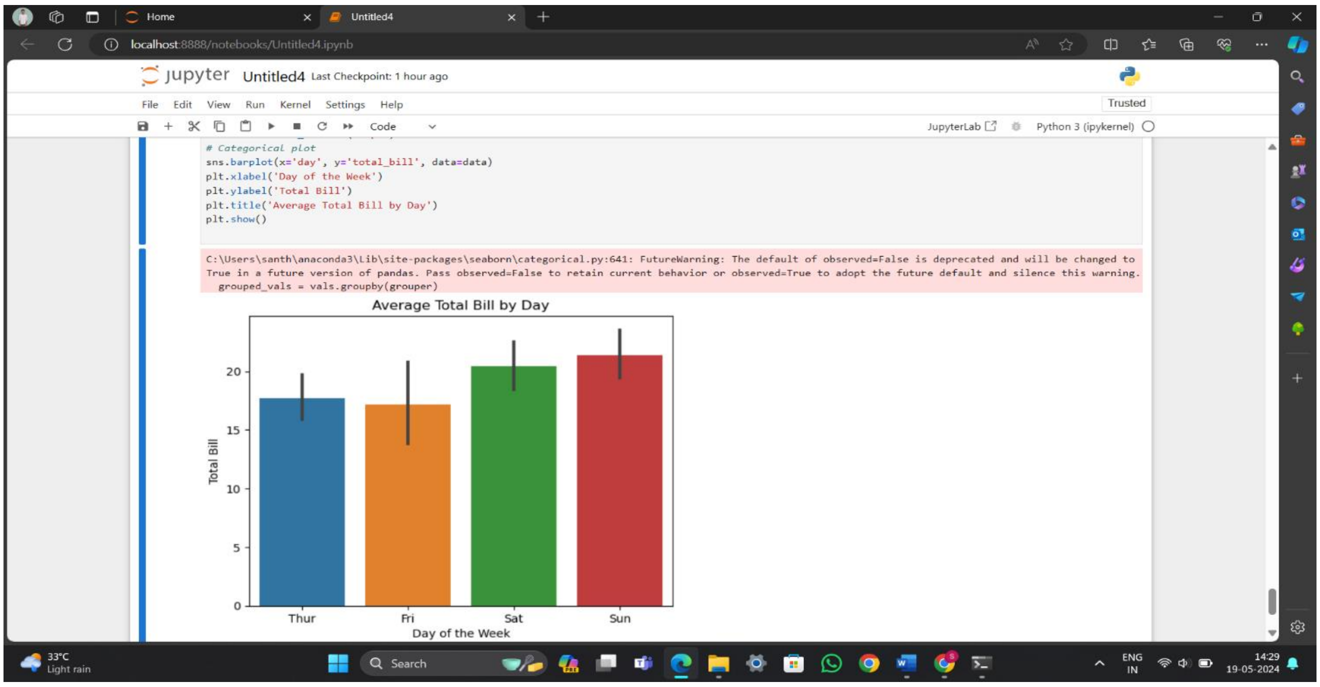
Use Cases

Categorical plots are ideal for:

- **Comparing Distributions**: Comparing the distribution of a numerical variable across different categories.
- **Identifying Patterns**: Observing patterns or trends in categorical data.
- **Highlighting Differences**: Highlighting differences or similarities between groups.

Creating a Categorical Plot in Seaborn

Let's go through the steps to create and customize a categorical plot using Seaborn.


```
# Categorical plot
sns.barplot(x='day', y='total_bill', data=data)
plt.xlabel('Day of the Week')
plt.ylabel('Total Bill')
plt.title('Average Total Bill by Day')
plt.show()
```

```
C:\Users\santh\anaconda3\Lib\site-packages\seaborn\categorical.py:641: FutureWarning: The default of observed=False is deprecated and will be changed to
True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
```
localhost:8888/notebooks/Untitled4.ipynb
Jupyter Untitled4 Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Average Total Bill by Day
Total Bill
0
5
10
15
20
Thur
Fri
Sat
Sun
Day of the Week

3.Relational Plots:

A relational plot in Seaborn is a type of data visualization used to explore relationships between two or more variables. It provides a visual summary of the relationship between variables, helping to identify patterns, trends, correlations, and outliers in the data. Relational plots are particularly useful for analyzing bivariate relationships and understanding how one variable changes with respect to another.

Key Components of a Relational Plot

1. **X-Axis (Horizontal Axis)**: Represents one variable in the relationship.
2. **Y-Axis (Vertical Axis)**: Represents another variable in the relationship.
3. **Markers or Lines**: Visual elements used to represent individual data points or trends.
4. **Color or Size Mapping**: Additional visual encoding used to represent a third variable.

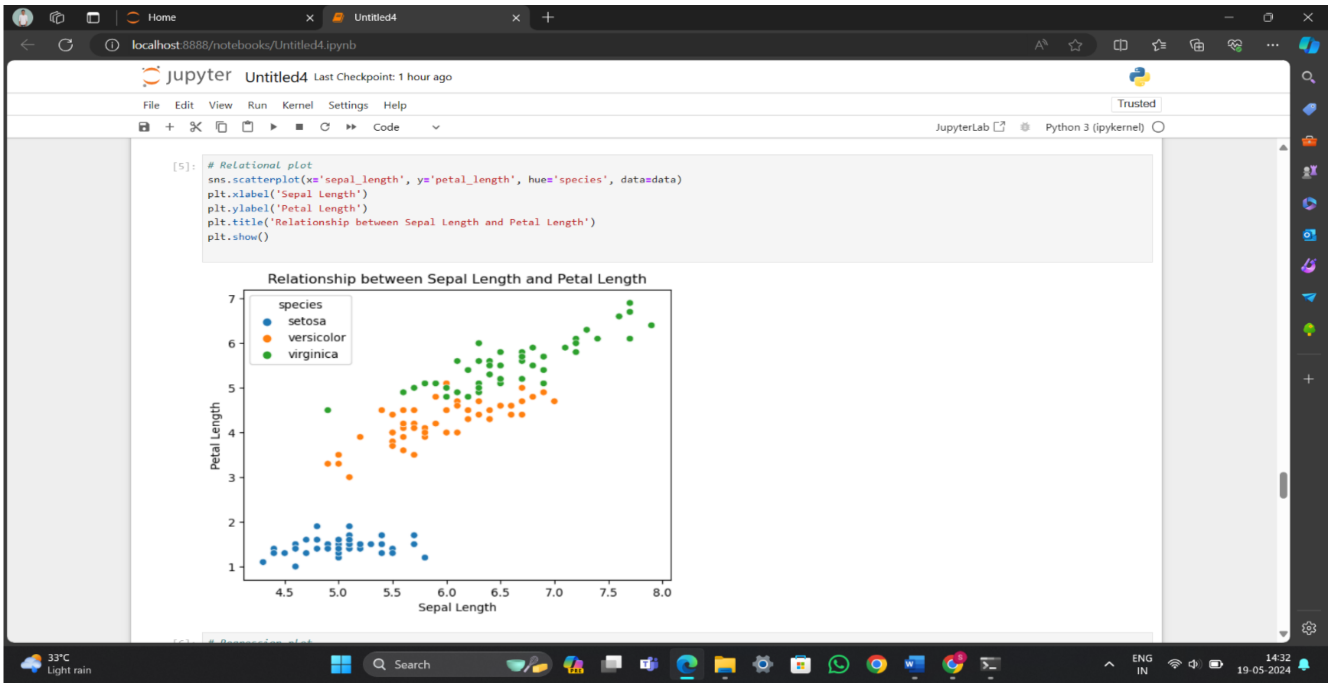
Use Cases

Relational plots are ideal for:

- **Exploring Bivariate Relationships**: Visualizing the relationship between two variables.
- **Identifying Trends and Patterns**: Observing trends, patterns, or correlations in the data.
- **Detecting Outliers**: Identifying outliers or unusual data points.

Creating a Relational Plot in Seaborn

Let's go through the steps to create and customize a relational plot using Seaborn.



```
[5]: # Relational plot
sns.scatterplot(x='sepal_length', y='petal_length', hue='species', data=data)
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.title('Relationship between Sepal Length and Petal Length')
plt.show()
```

4.Regression Plots:

A regression plot in Seaborn is a type of data visualization used to visualize the relationship between two variables and fit a regression model to the data. It provides a visual summary of the linear relationship between variables, helping to identify trends, patterns, and correlations in the data. Regression plots are particularly useful for analyzing the strength and direction of the relationship between variables.

Key Components of a Regression Plot:

1. **X-Axis (Horizontal Axis)**: Represents the independent variable or predictor variable.
2. **Y-Axis (Vertical Axis)**: Represents the dependent variable or response variable.
3. **Markers or Lines**: Visual elements used to represent individual data points or the regression line.
4. **Regression Line**: A line that best fits the observed data points, indicating the relationship between variables.
5. **Confidence Interval**: Shaded area around the regression line representing the uncertainty.

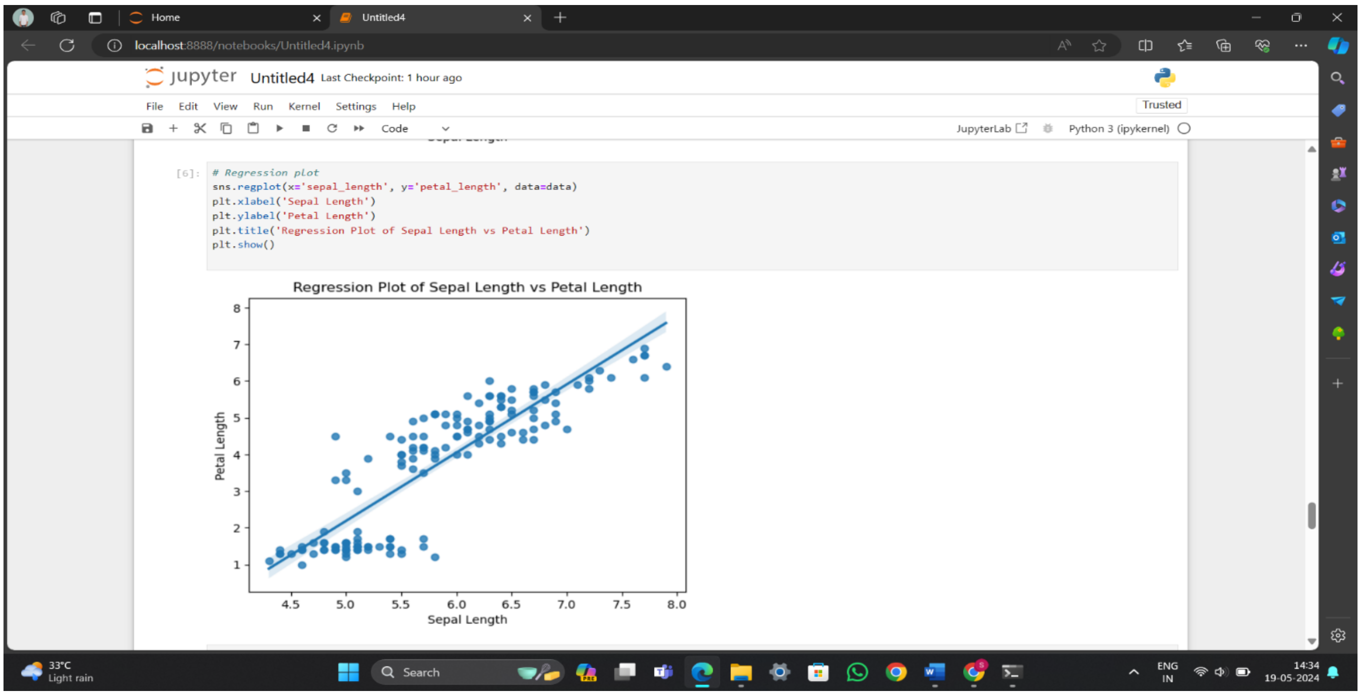
Use Cases:

Regression plots are ideal for:

- **Visualizing Linear Relationships**: Analyzing the linear relationship between two variables.
- **Modeling Predictive Relationships**: Fitting a regression model to the data and making predictions.
- **Identifying Outliers**: Detecting outliers or unusual data points that deviate from the regression line.

Creating a Regression Plot in Seaborn

Let's go through the steps to create and customize a regression plot using Seaborn.


Home
Untitled4
localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
Regression Plot of Sepal Length vs Petal Length
Sepal Length
Petal Length
33°C Light rain
Search
ENG IN
14:34 19-05-2024

```
[6]: # Regression plot
sns.regplot(x='sepal_length', y='petal_length', data=data)
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.title('Regression Plot of Sepal Length vs Petal Length')
plt.show()
```

5.Time Series Plots:

A time series plot is a type of data visualization used to display the variation of a variable over time. It is particularly useful for analyzing time-dependent data and identifying trends, patterns, and seasonality in the data. Time series plots represent data points as a series of points or lines connected by straight lines, with time represented on the x-axis and the variable of interest on the y-axis.

Key Components of a Time Series Plot

1. **X-Axis (Horizontal Axis)**: Represents time, typically in chronological order.
2. **Y-Axis (Vertical Axis)**: Represents the variable of interest, such as stock prices, temperature, or sales.
3. **Data Points or Lines**: Individual data points or lines connecting data points to visualize the trend over time.
4. **Labels and Titles**: Labels for the axes and a title to provide context and interpretation.

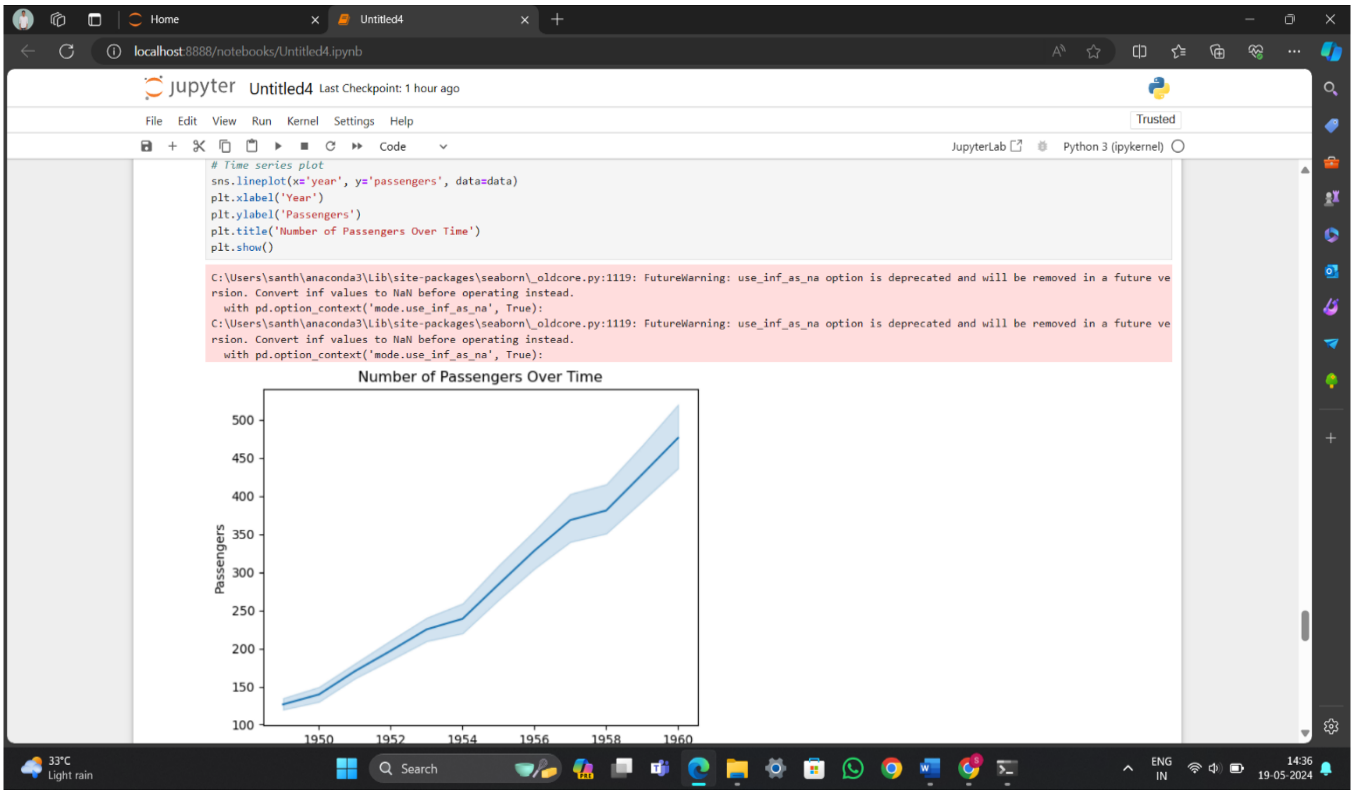
Use Cases

Time series plots are ideal for:

- **Analyzing Trends**: Observing trends or patterns in time-dependent data.
- **Forecasting**: Making predictions about future values based on historical data.
- **Detecting Seasonality**: Identifying seasonal patterns or cycles in the data.

Creating a Time Series Plot in Seaborn

Let's go through the steps to create and customize a time series plot using Seaborn.


Home
Untitled4
localhost:8888/notebooks/Untitled4.ipynb
jupyter Untitled4 Last Checkpoint: 1 hour ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```python
# Time series plot
sns.lineplot(x='year', y='passengers', data=data)
plt.xlabel('Year')
plt.ylabel('Passengers')
plt.title('Number of Passengers Over Time')
plt.show()
```

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