**Fall 2023 DATA 230 – 11**

**Data Visualization**

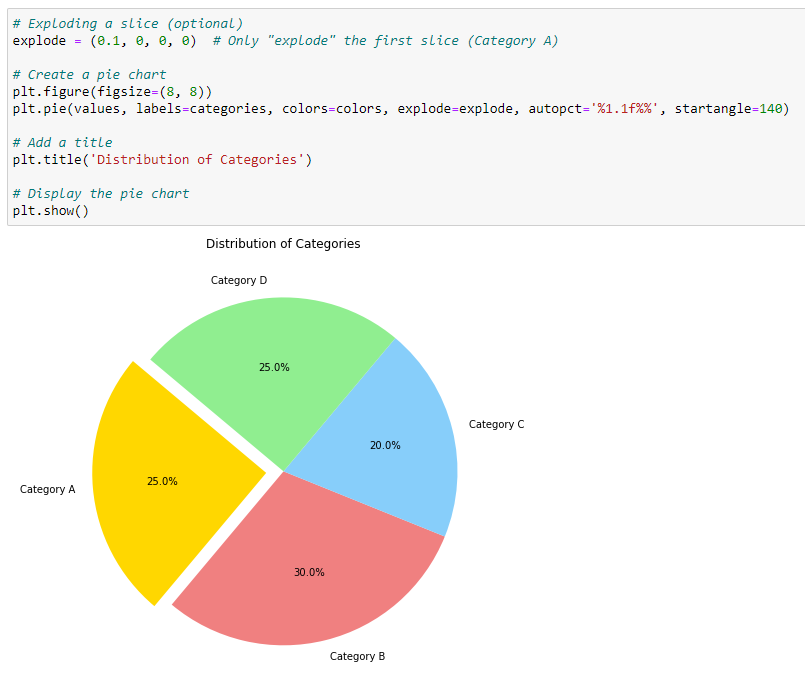
**Homework – 4**

**Name :- Prayag Nikul Purani**

**SJSU Id :- 017416737**

**Question 1:**

**Visualization 1: A case for pie charts code**



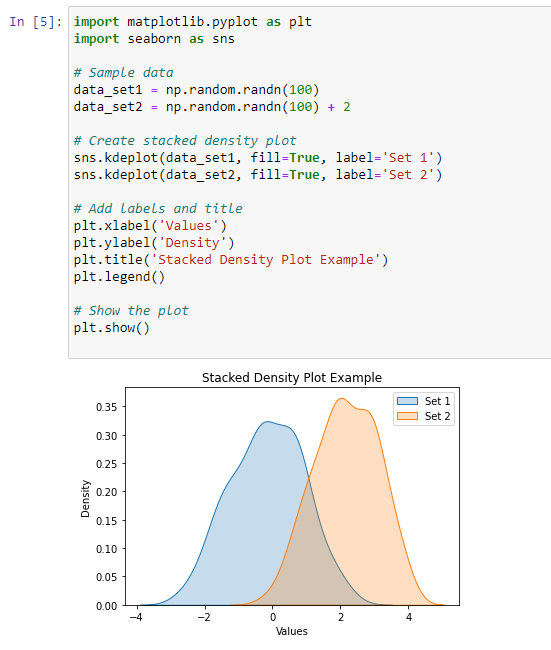
This is a basic example, and you can customize it further based on your specific needs. Pie charts are useful for showing the proportion of parts of a whole, but keep in mind that they can be misleading if not used appropriately, especially when comparing a large number of categories or when the differences between categories are small. Always consider the context and the nature of your data when choosing a visualization.

**Visualization 2: A case for side-by-side bars**

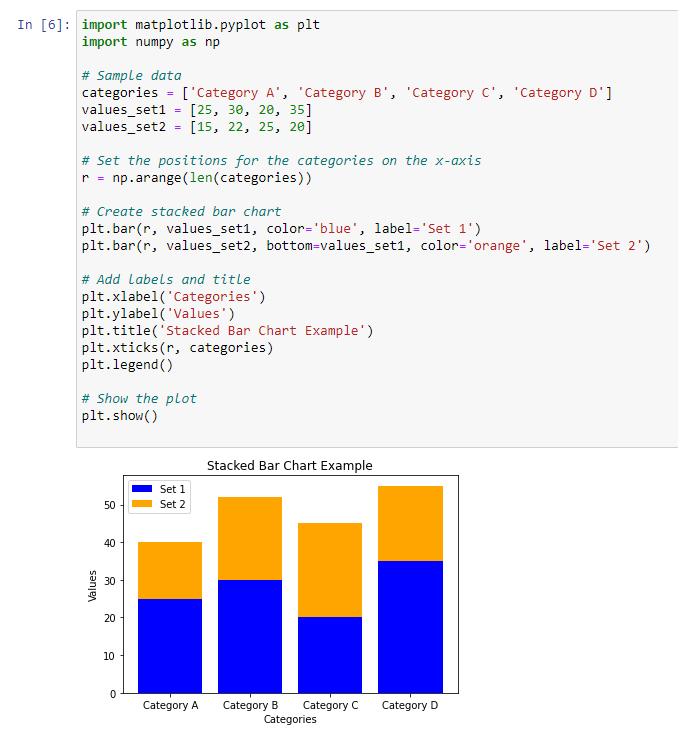


Import the necessary libraries (**matplotlib.pyplot** and **numpy** for creating arrays).Define your sample data - categories and values for two sets.Set the bar width and positions for the categories on the x-axis.Create side-by-side bar charts using **plt.bar()** for each set of values.Customize the chart with labels, title, and x-axis ticks.Add a legend to distinguish between the sets.Display the chart using **plt.show()**.This example illustrates how side-by-side bar charts can be used to compare the values of different sets within the same categories. Adjust the data and customization options based on your specific requirements.

**Visualization 3: A case for stacked bars and stacked densities**



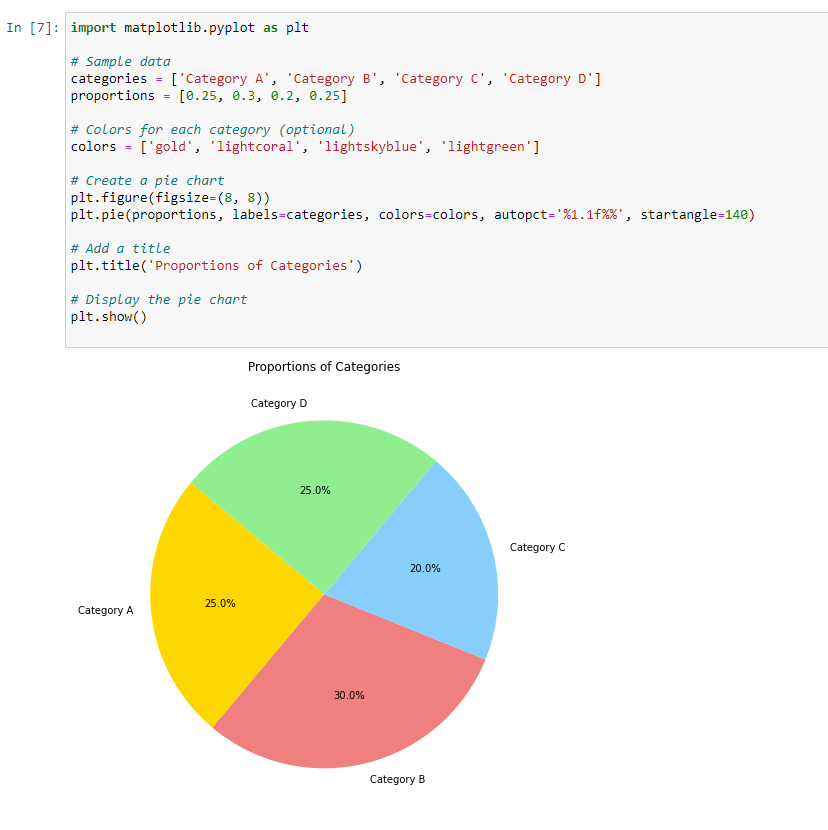
Use **seaborn.kdeplot()** to create kernel density estimates for each dataset.The **fill=True** parameter is used to fill the area under the density curve.Add labels, title, and a legend to the plot.These examples demonstrate how to create stacked bar charts and stacked density plots to visualize the distribution of data across different sets or categories. Adjust the data and customization options based on your specific needs.



Explanation:

Similar to the side-by-side bar chart, set positions for the categories on the x-axis.Use **plt.bar()** to create stacked bars, where **bottom** parameter is used to specify the bottom value for the second set.

**Visualization 4: Visualizing proportions separately as parts of the total**

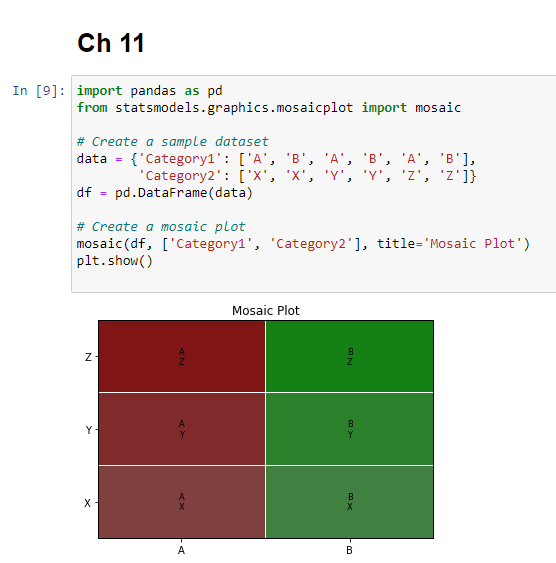


proportions represent the proportion of each category in the total.The plt.pie() function is used to create the pie chart.The autopct='%1.1f%%' parameter adds percentage labels to each slice.The startangle parameter determines the starting angle of the pie chart.

Adjust the categories and proportions lists according to your specific data. This pie chart visualizes each category as a part of the total, making it easy to compare the proportions of different categories.

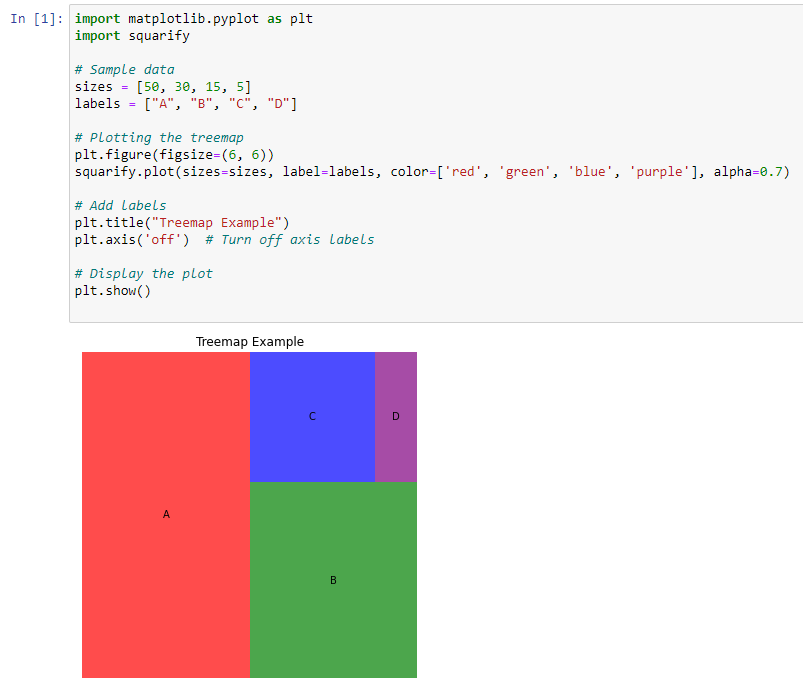
**Question 2:**

**Visualization 1: Mosai plots**



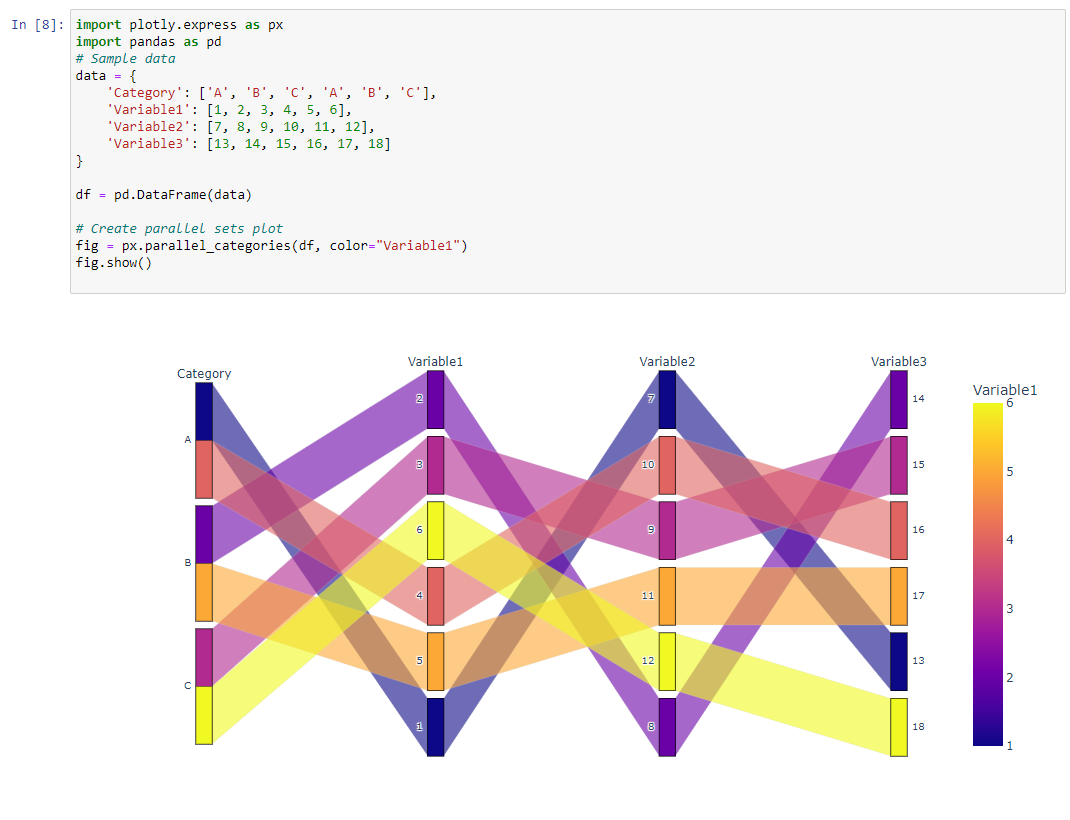
A mosaic plot displays the joint distribution of two or more categorical variables as rectangles (mosaics) within a rectangle. The area of each rectangle corresponds to the proportion of observations in each category combination.Here's a basic example using the **statsmodels** library in Python:This example creates a mosaic plot for two categorical variables, 'Category1' and 'Category2'. You can customize this code based on your specific data.If you have a different concept in mind related to nested proportions or exponentiation, please provide more details or clarify, and I'll be happy to assist you further.

**Visualization 2: Tree maps**



A treemap is another type of visualization that displays hierarchical data as nested rectangles. Each branch of the hierarchy is given a colored rectangle, and its sub-branches are represented as smaller rectangles within it. The size of each rectangle is usually proportional to a specific metric, such as the number of observations or some other numerical value. In this example, **sizes** represents the size of each rectangle, and **labels** provides labels for each rectangle. The **squarify.plot** function is used to create the treemap, and you can customize it further based on your specific needs.

**Visualization 3: Parallel sets**



Parallel sets (also known as parallel coordinates or parallel coordinates plot) is a method of visualizing high-dimensional data. It is particularly useful for exploring relationships and patterns in multivariate datasets. Each variable is represented by a vertical axis, and lines are drawn to connect points with the same values across these axes.

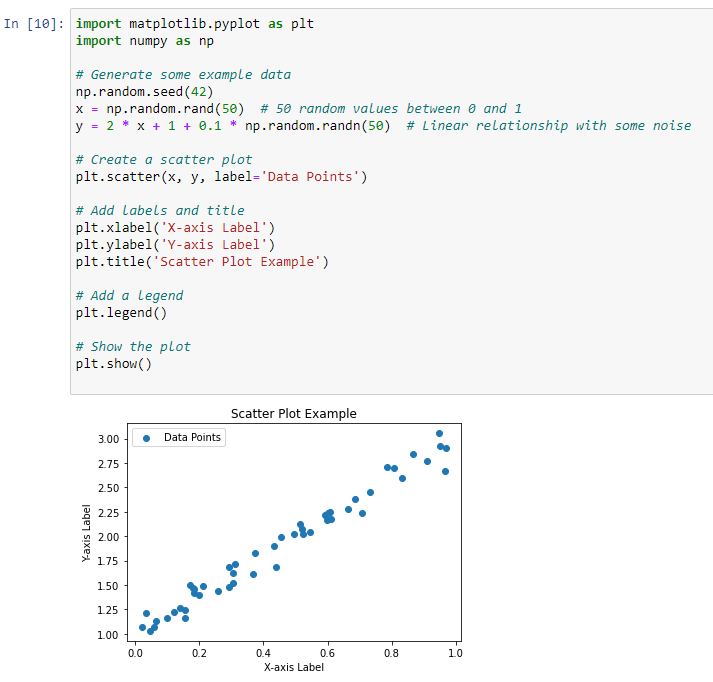
**Visualization 4: Nested Pie**



This example creates a nested pie chart with an outer ring and three inner rings. You can customize the labels and sizes according to your dataset. This is a basic example, and you can adjust the code based on your specific requirements for nested pie charts.

**Question 3:**

**Visualization 1: Scatter plots**



np.random.rand(50) generates 50 random values between 0 and 1 for the x-axis.

2 \* x + 1 + 0.1 \* np.random.randn(50) creates a linear relationship with some random noise for the y-axis.

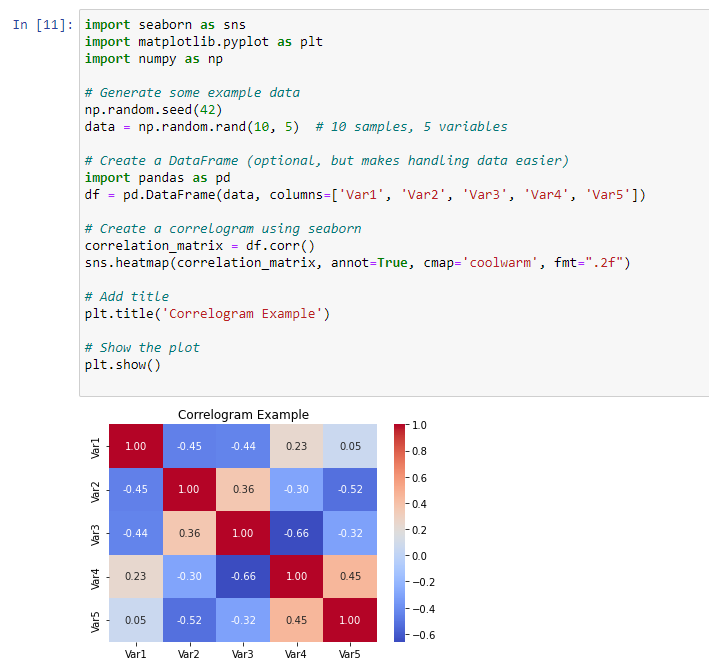
plt.scatter(x, y, label='Data Points') creates a scatter plot using the generated data.

plt.xlabel, plt.ylabel, and plt.title set labels for the axes and the title of the plot.

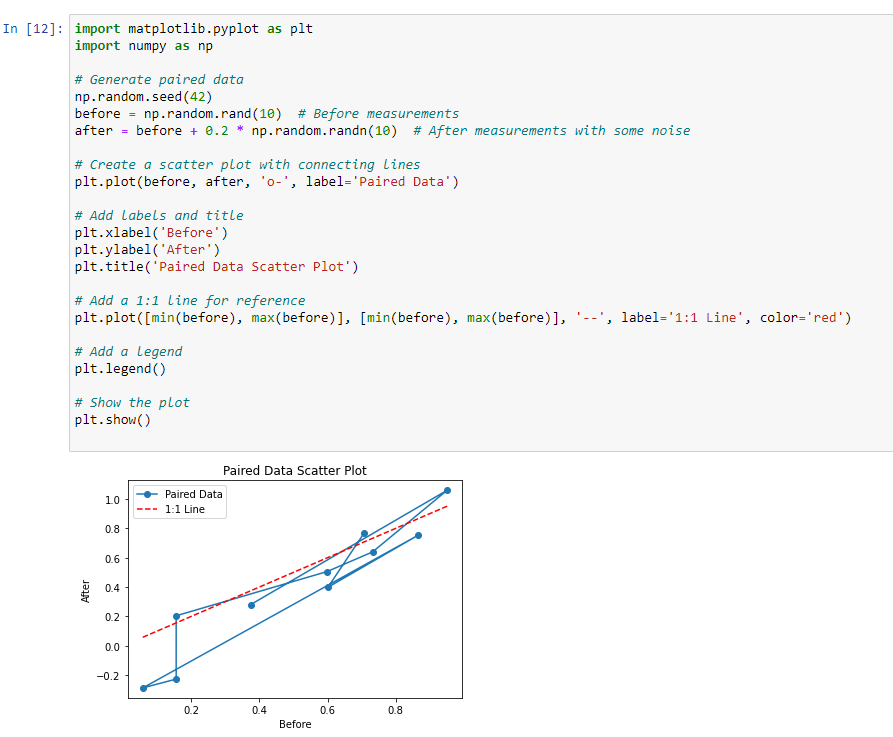
plt.legend() adds a legend to the plot.

plt.show() displays the plot.

**Visualization 2: Correlograms**



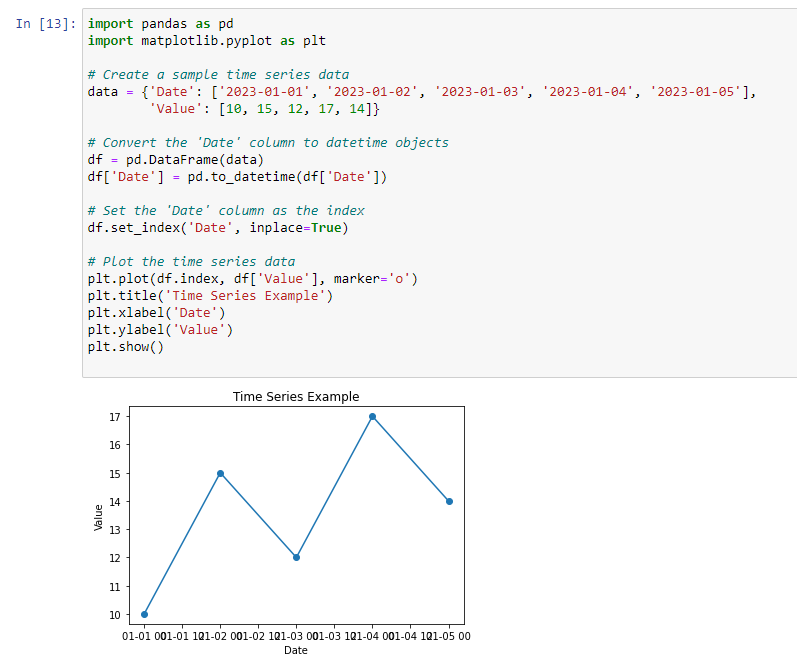
A correlogram is a visual representation of the correlation matrix. It's a matrix plot with correlation coefficients between variables displayed as colors. Here's an example of how to create a correlogram in Python using the **seaborn** library, which is built on top of **matplotlib**:

**Visualization3: Paired data**

When dealing with paired data, where there are two sets of observations that are related in some way (e.g., before and after measurements, or two measurements taken on the same subjects), you might want to visualize the relationship between the pairs. A scatter plot with a line connecting the paired points is a common choice for this.Top of Form

**Question 4:**

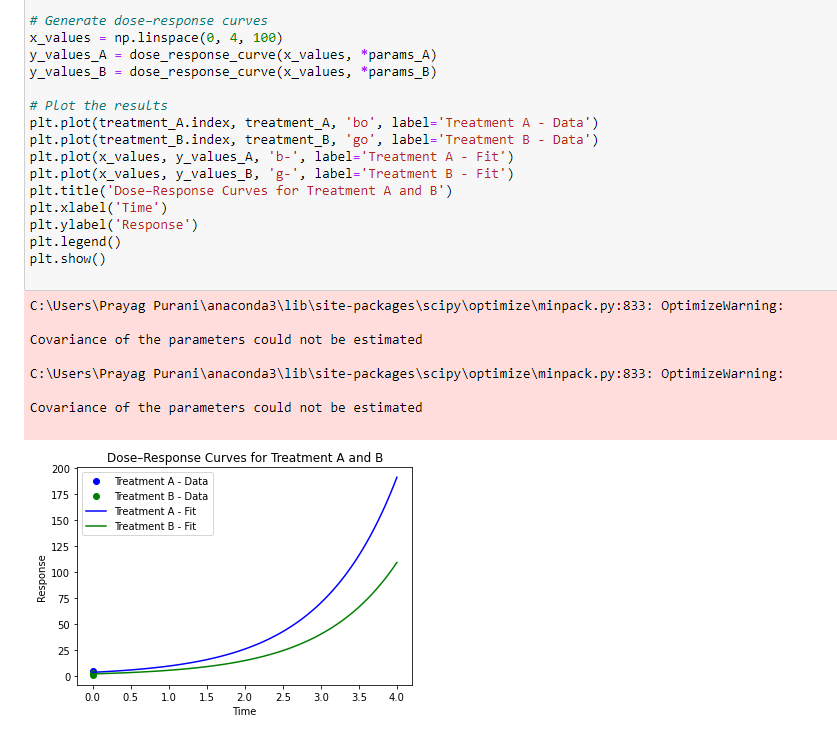
**Visualization 1: Individual time series**



To work with individual time series data in Python, you can use various libraries, but one of the most commonly used ones is **pandas** for data manipulation and **matplotlib** for data visualization. If you're dealing with time series data, you might also want to consider using **datetime** objects for handling dates.

This example creates a simple time series DataFrame with two columns: 'Date' and 'Value'. It then converts the 'Date' column to datetime objects and sets it as the index. Finally, it plots the time series data using **matplotlib**.

**Visualization 2: Multiple time series and dose–response curves**

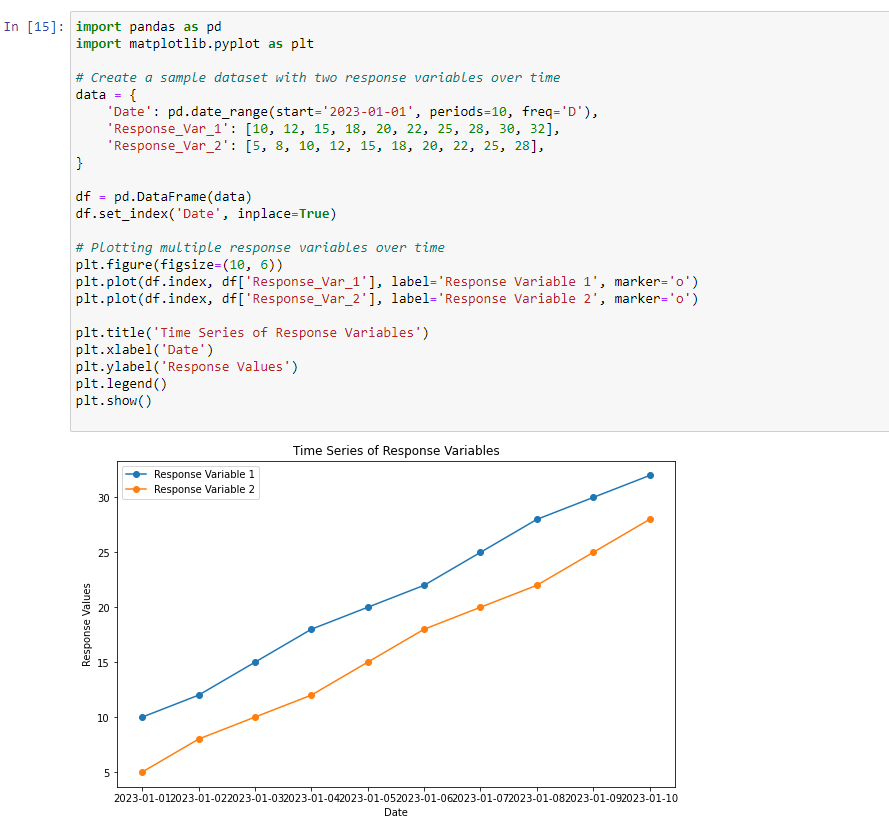


If you have multiple time series and want to create dose–response curves, you might be dealing with experimental or observational data where you have different conditions or treatments. In such cases, you can use Python libraries such as **pandas** for data manipulation, **matplotlib** for visualization, and possibly **scipy** or **numpy** for mathematical operations.

This example assumes a simple exponential dose–response curve (**a \* exp(b \* x)**). You may need to adjust the curve function based on the nature of your data. Also, make sure to replace the sample data with your actual dataset.

Remember that the choice of the dose–response function depends on the characteristics of your data and the underlying biological or experimental context. The **curve\_fit** function from **scipy.optimize** is used to fit the curve to your data.

**Visualization 3: Time series of two or more response variables**



If you have a time series dataset with two or more response variables, and you want to analyze and visualize the trends in these variables over time, you can use Python with libraries like **pandas** for data manipulation and **matplotlib** or **seaborn** for visualization. Additionally, you might want to consider using **plotly** for interactive visualizations. This code assumes that you have a time series dataset with a 'Date' column and two response variables ('Response\_Var\_1' and 'Response\_Var\_2'). It then uses **matplotlib** to create a line plot with markers for each response variable over time.

If you have more response variables, you can extend this approach by adding more lines to the plot or using subplots.