**Beginner Level Task – Visualization Library**

**Documentation**

Intern Name: BhupathiRaju.Yuva Tejaswini

Domain:Data Science

Task Level: Beginner

Libraries Chosen: Pandas and Seaborn

# **Pandas**

This document explores how to leverage Pandas' integrated plotting capabilities in Python to create effective data visualizations. We will delve into five core plot types: Line, Bar, Histogram, Box, and Pie charts, demonstrating their application with practical examples and code snippets. By mastering these techniques, users can unlock deeper insights from their data and communicate complex information clearly and concisely.

# **Pandas Plotting**

Pandas, a fundamental library for data manipulation in Python, offers a seamless integration with Matplotlib, providing powerful plotting methods directly accessible via Series.plot() and DataFrame.plot(). This built-in functionality simplifies the process of creating visualizations, allowing for rapid exploration of data distributions, relationships, and trends without the need for extensive Matplotlib setup.

#### **Example: Loading Data**

CODE:

import pandas as pd

df = pd.read\_csv('retail\_sales.csv')

print(df.head())

This snippet demonstrates how to load a sample CSV file, retail\_sales.csv, into a Pandas DataFrame, which will serve as our dataset for the subsequent plotting examples. The .head() method displays the first few rows, providing a quick overview of the data structure.

# **Line Plots:**

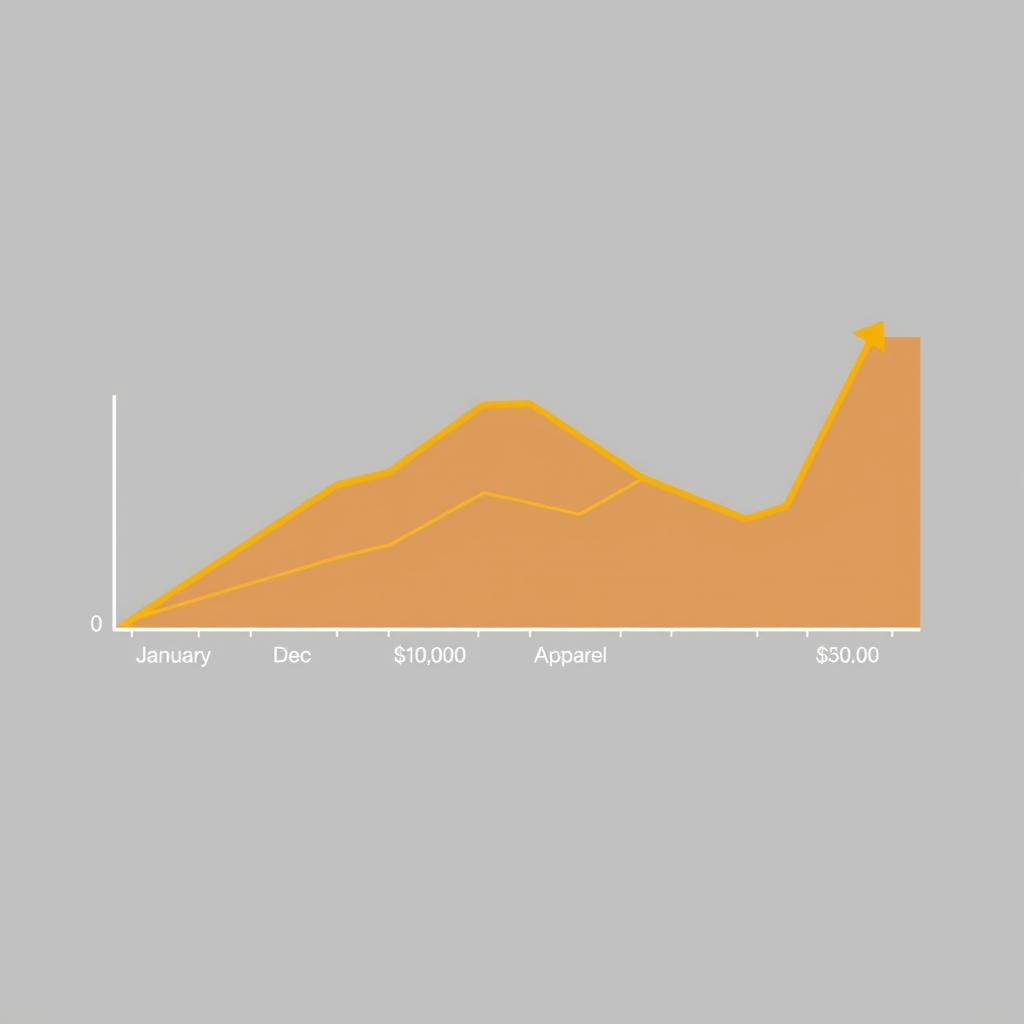
Line plots are indispensable for visualizing continuous data, especially when dealing with time series. They effectively illustrate changes, trends, and patterns over time or across ordered categories. Each point on the line represents a data value, and the line connecting them highlights the progression or relationship between consecutive data points.

#### **Example: Monthly Sales Trends**

CODE:

df.groupby('Month')['Sales'].sum().plot(kind='line', title='Monthly Apparel Sales Trends', figsize=(10, 6))

OUTPUT:



These plots are ideal for spotting seasonality, cyclical patterns, or long-term growth/decline. For instance, tracking monthly sales of a product category over a year provides immediate insight into peak seasons and lean periods, aiding in inventory management and marketing strategies.

# **Bar Plots:**

Bar plots are a fundamental tool for displaying and comparing quantities across distinct categories. Each bar represents a category, and its length (or height) corresponds to the value it represents, such as counts, sums, or averages. They are particularly useful for visualizing discrete data and making direct comparisons between different groups or segments.

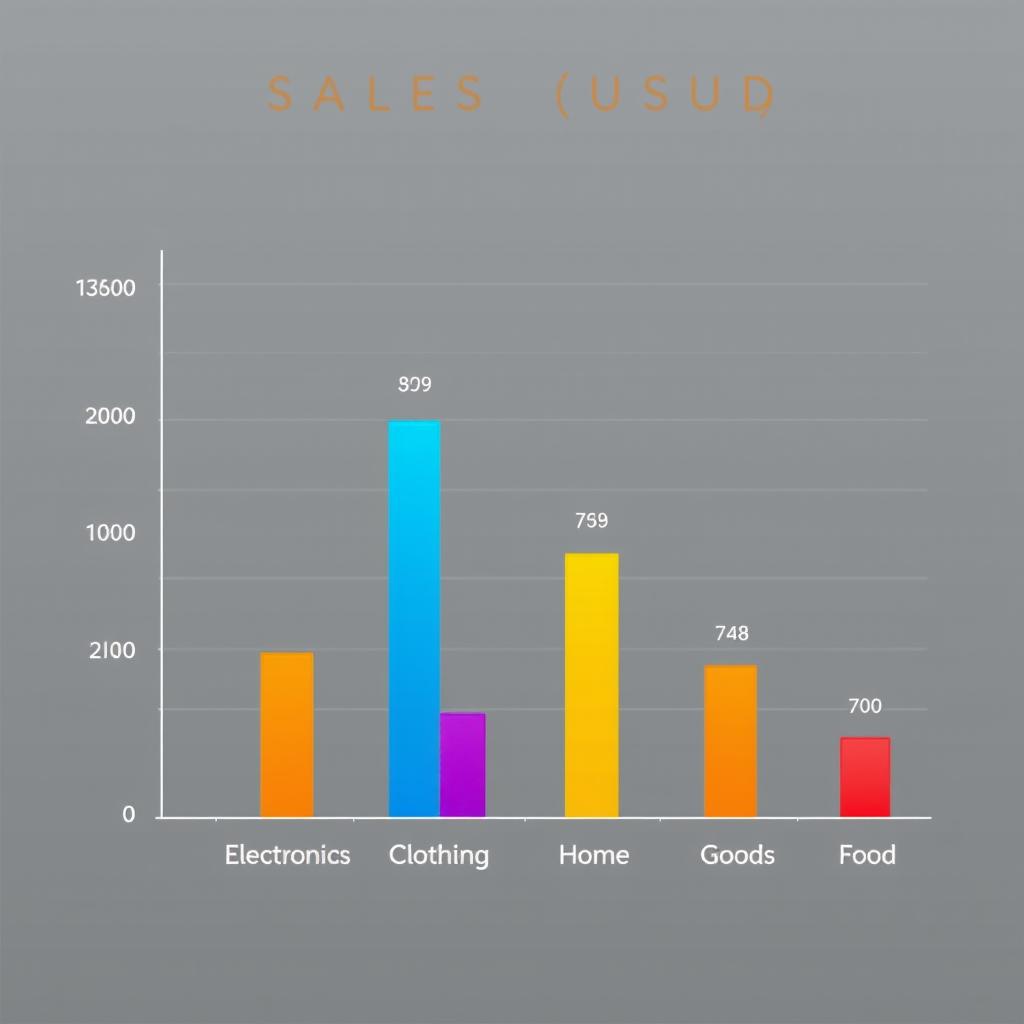
For example, a bar plot can effectively show total sales generated by each product category, allowing for quick identification of top-performing or underperforming segments. This visual comparison can guide business decisions related to product development, marketing allocation, and resource distribution.

#### **Example: Total Sales by Product Category**

CODE:

df.groupby('Category')['Sales'].sum().plot(kind='bar', title='Total Sales by Product Category', xlabel='Product Category', ylabel='Total Sales (USD)', rot=45)

OUTPUT:



# **Histograms:**

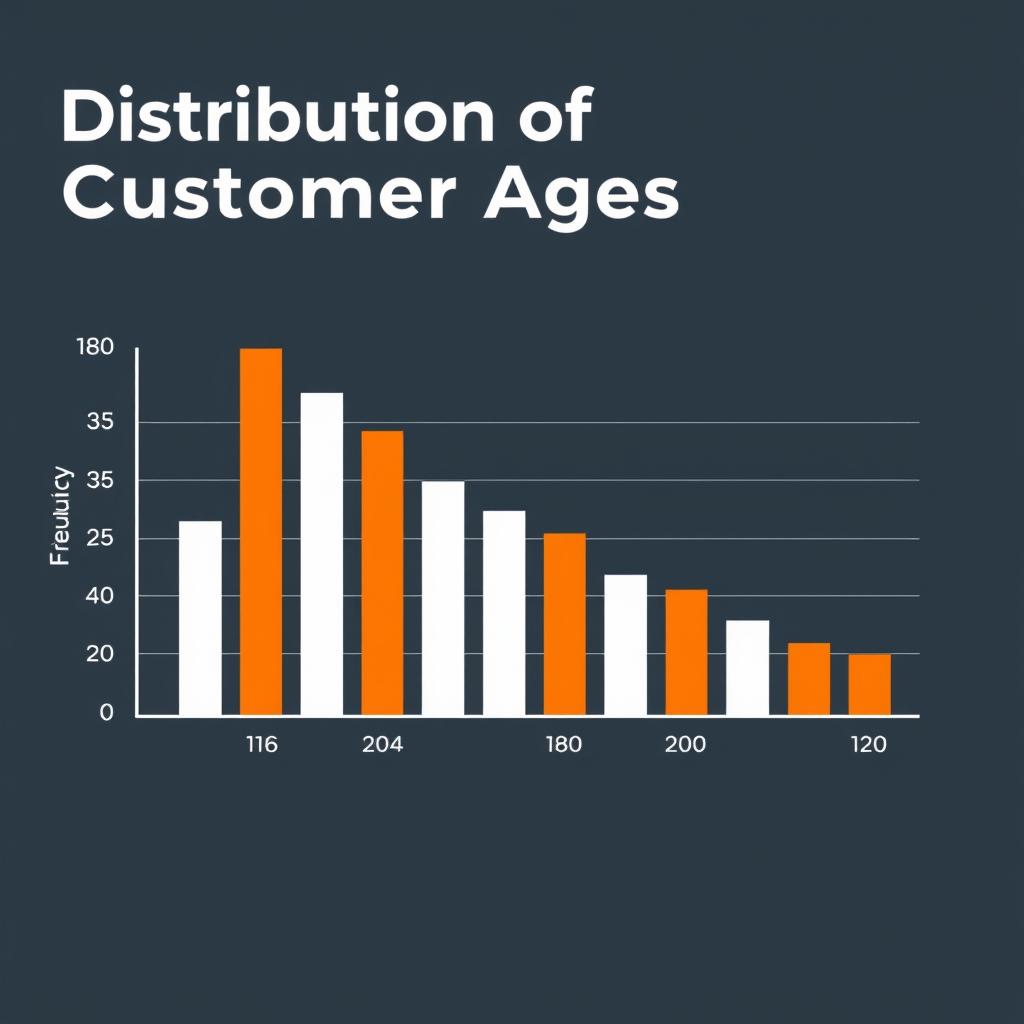
Histograms are powerful visualizations for illustrating the frequency distribution of numerical data. They divide the data into a series of intervals, or "bins," and then count how many data points fall into each bin. The height of each bar in the histogram represents the frequency (or count) of data points within that specific bin.

#### **Example: Customer Age Distribution**

CODE:

df['Customer\_Age'].plot(kind='hist', bins=15, title='Distribution of Customer Age', xlabel='Customer Age', ylabel='Frequency', edgecolor='black')

OUTPUT:



This type of plot is crucial for understanding the shape, spread, and central tendency of a variable. For instance, a histogram of 'Customer Age' can reveal if the customer base is skewed towards younger or older demographics, if there are multiple peaks (bimodal distribution), or if the data is evenly spread. The number of bins chosen can significantly impact the visual interpretation of the distribution, so it's often useful to experiment with different bin sizes.

# **Box Plots:**

Box plots, also known as box-and-whisker plots, provide a concise visual summary of the distribution of numerical data through quartiles. They are excellent for quickly identifying the median, spread (interquartile range), and potential outliers within a dataset. A box plot typically displays five key summary statistics: the minimum value, the first quartile (Q1), the median (Q2), the third quartile (Q3), and the maximum value.

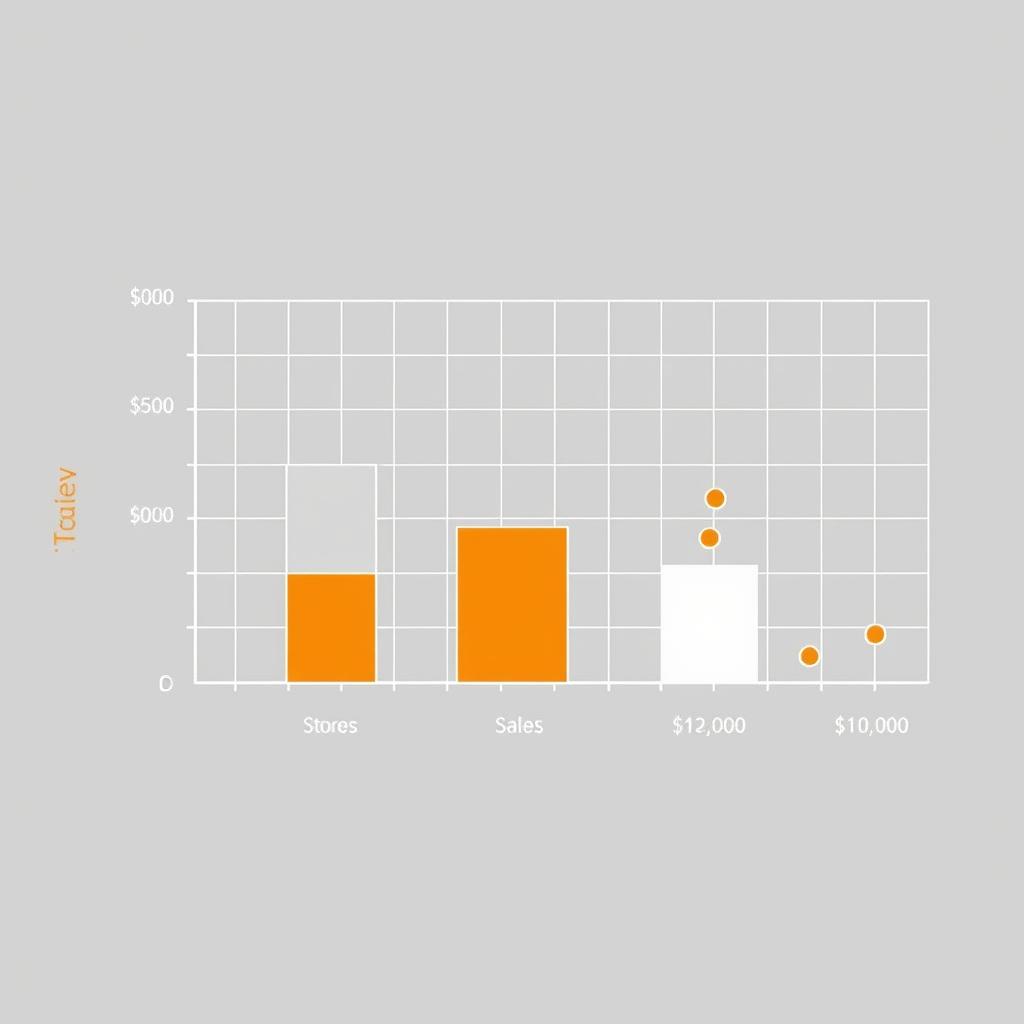
Outliers, data points that fall significantly outside the typical range, are often plotted as individual points beyond the "whiskers." This makes box plots particularly useful for comparing distributions across different groups or categories, such as sales distribution across various store locations, to spot inconsistencies or performance variations.

#### **Example: Sales Distribution by Store Location**

CODE:

df.boxplot(column='Sales', by='Store\_Location', figsize=(12, 7), grid=False)

OUTPUT:



**Pie Charts:**

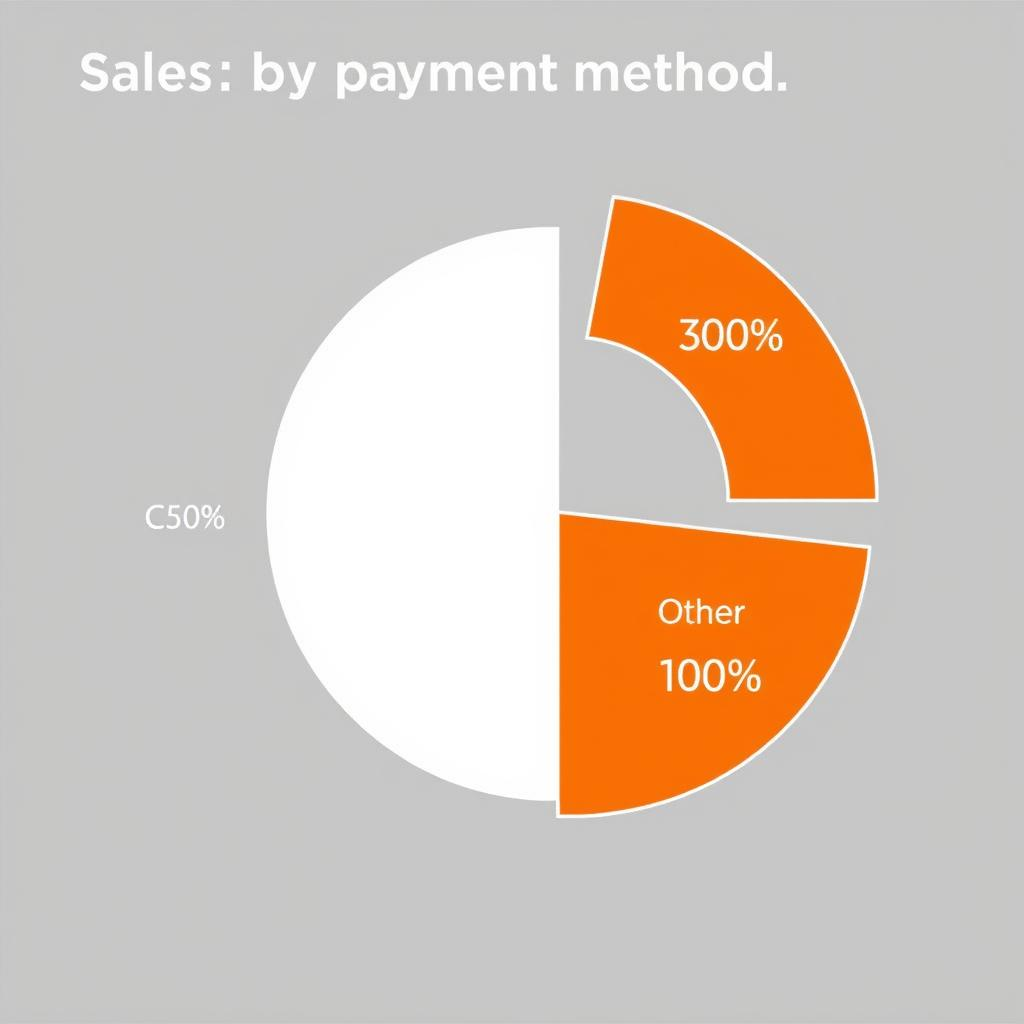
Pie charts are circular statistical graphics divided into slices to illustrate numerical proportion. Each slice represents a category, and the size of the slice (arc length or area) is proportional to the quantity it represents, typically a percentage or fractional composition of a whole.

#### **Example: Sales by Payment Method**

CODE:

df['Payment\_Method'].value\_counts().plot(kind='pie', autopct='%1.1f%%', title='Proportion of Sales by Payment Method', figsize=(8, 8))

OUTPUT:



They are most effective when displaying a small number of categories (ideally 5-7 or less) to avoid clutter and ensure clarity. For example, a pie chart can effectively visualize the proportion of sales attributed to different payment methods, offering a quick overview of customer preferences or payment channel performance. The autopact parameter automatically displays percentages on each slice, enhancing readability.

**Seaborn**

Seaborn seamlessly extends Matplotlib's capabilities, acting as a high-level interface that enhances visual appeal and simplifies complex statistical plots while retaining full compatibility with Matplotlib's underlying functions. This integration means users can leverage Seaborn for quick, beautiful plots and then fine-tune them using Matplotlib's extensive API.

* **Global Styling:** sns.set\_theme() applies global styling overrides, instantly improving plot aesthetics with carefully chosen color palettes and default styles.
* **Display Control:** plt.show() is often required for displaying plots in various Python environments, especially in scripts or non-Jupyter notebooks.
* **Fine-Grained Customization:** Users can access and manipulate Matplotlib axes directly (e.g., ax.set\_title() , ax.set\_xlabel() ) for precise control over plot elements, allowing for highly customized visualizations.

**Creating different types of plots:**

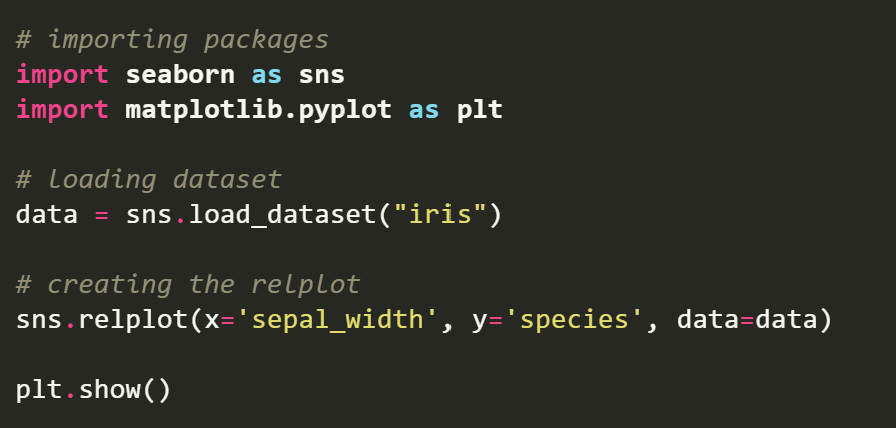
**Relplot():**

This function provides us the access to some other different axes-level functions which shows the relationships between two variables with semantic mappings of subsets. It is plotted using the relplot() method.

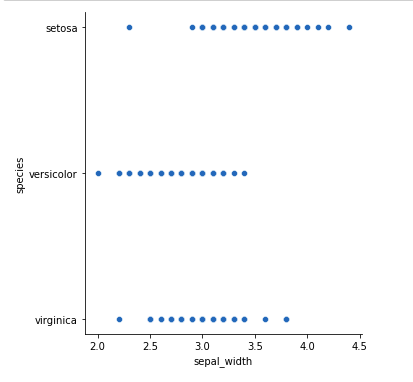
syntax:

*seaborn.relplot(x=None, y=None, data=None, \*\*kwargs)*

Code:



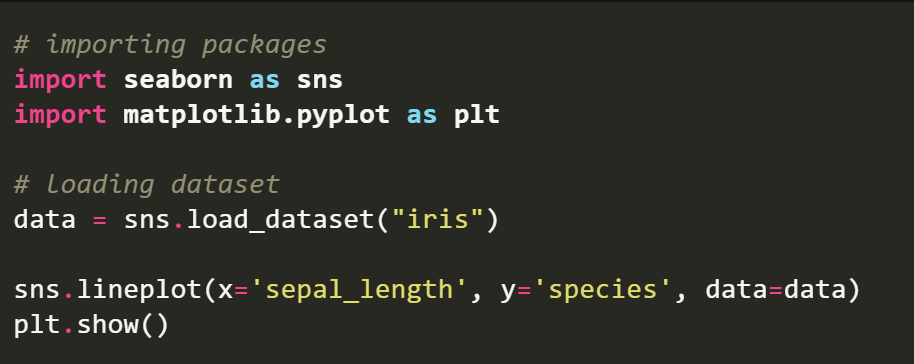
Output:



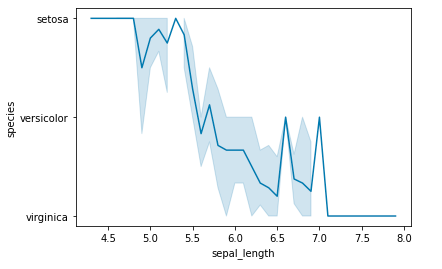
**Line Plot:**

For certain datasets, you may want to consider changes as a function of time in one variable, or as a similarly continuous variable. In this case, drawing a line-plot is a better option. It is plotted using the lineplot() method.

Code:



Output:



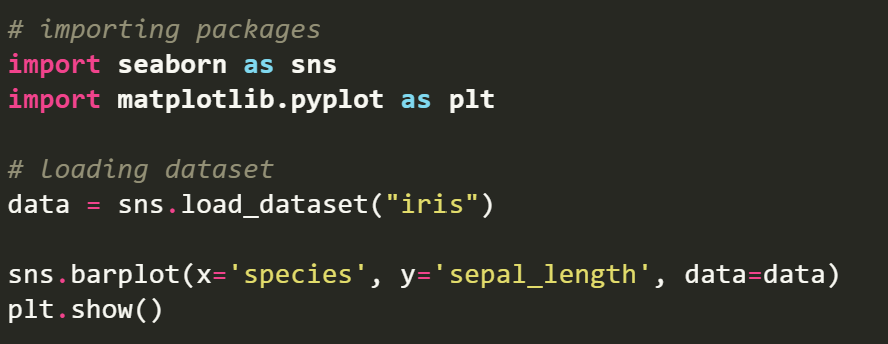
**Bar Plot:**

A barplot is basically used to aggregate the categorical data according to some methods and by default its the mean. It can also be understood as a visualization of the group by action. To use this plot we choose a categorical column for the x axis and a numerical column for the y axis and we see that it creates a plot taking a mean per categorical column. It can be created using the barplot() method.

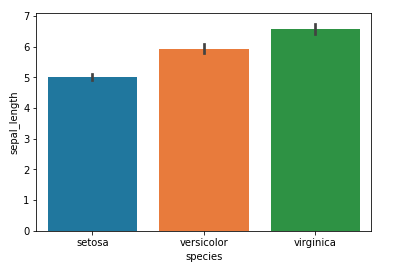
Syntax:

*barplot([x, y, hue, data, order, hue\_order, …])*

Code:



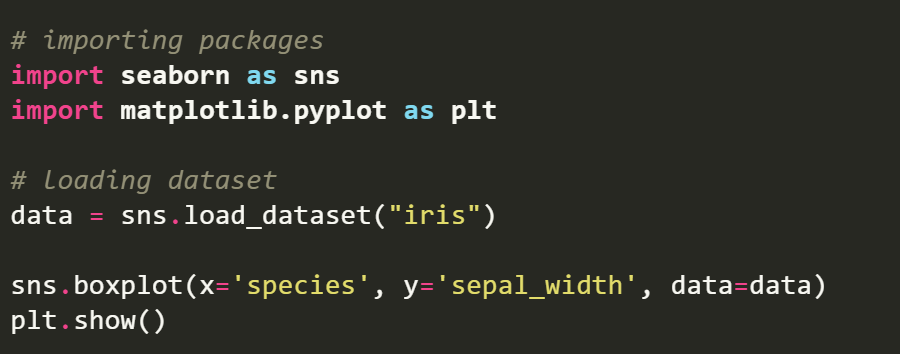
Output:



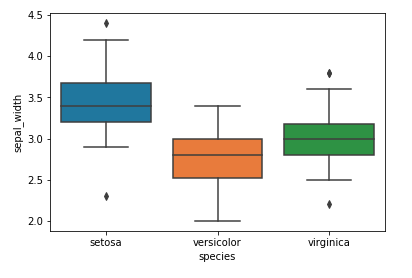
**Box Plot:**

A boxplot is sometimes known as the box and whisker plot.It shows the distribution of the quantitative data that represents the comparisons between variables. boxplot shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution i.e. the dots indicating the presence of outliers. It is created using the boxplot() method.

Code:



Output:



### **COMPARISON OF PANDAS AND SEABORN:**

Key Observations in Code

* **Pandas** handles data creation, manipulation, and basic plotting.
* **Seaborn** excels at advanced, statistical visualizations.
* They complement each other beautifully in data science workflows

Advantages of pandas:

* Pandas simplifies data handling in Python.
* It’s ideal for everything from simple data cleaning to advanced analytics.
* Its integration with visualization and machine learning libraries makes it essential for any data science workflow

Advantages of seaborn:

* **Seaborn = Elegant + Insightful Visualization**
* It saves time by automating statistical plotting.
* Perfect for exploring and presenting data patterns effectively

**Difference between pandas and seaborn:**

| **Feature** | **Pandas** | **Seaborn** |
| --- | --- | --- |
| Line Plot | ✅ (.plot(kind='line')) | ✅ (sns.lineplot) |
| Bar Chart | ✅ | ✅ |
| Histogram | ✅ | ✅ (with more styling options) |
| Box Plot | ⚠️ (less intuitive) | ✅ (rich and clear sns.boxplot) |
| Scatter Plot | ✅ | ✅ (better style and grouping with sns.scatterplot) |
| Heatmap | ❌ (not built-in) | ✅ (sns.heatmap) |
| Regression Plot | ❌ | ✅ (sns.regplot, sns.lmplot) |

### Conclusion:

* Use Pandas for basic and quick data visualizations directly from DataFrames.
* Use Seaborn when you need more elegant and statistically-oriented plots.

**Relplot()**

**This function provides us the access to some other different axes-level functions which shows the relationships between two variables with semantic mappings of subsets. It is plotted using the relplot() method.**