Visualization

Visualization is basically the process of turning information into pictures that are easy to understand. Instead of just looking at raw numbers or text, we use graphs, charts, or images to make the data tell a story. It helps us quickly spot patterns, trends, or connections that we might otherwise miss. It makes complex information simpler, clearer, and more engaging.

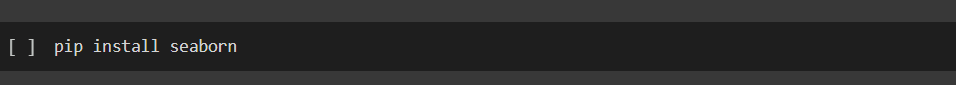
In python, data visualization has multiple libraries like Matplotlib, Seaborn, plotly, Pandas and Bokeh etc.

Seaborn

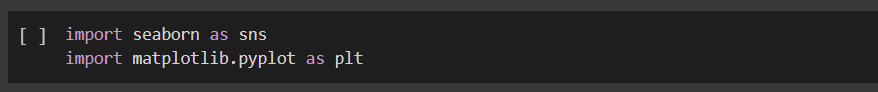
Seaborn is a library for making statistical graphics in Python. It builds on top of [matplotlib](https://matplotlib.org/) and integrates closely with [pandas](https://pandas.pydata.org/) data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on DataFrames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Install and import of Seaborn in python:

We install Seaborn (if not installed) (in the command prompt):

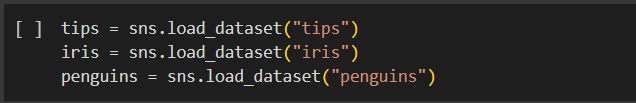


We also import seaborn for the plots and import matplotlib.pyplt to show plots:



Seaborn Datasets:

Seaborn comes with same datasets to practice:

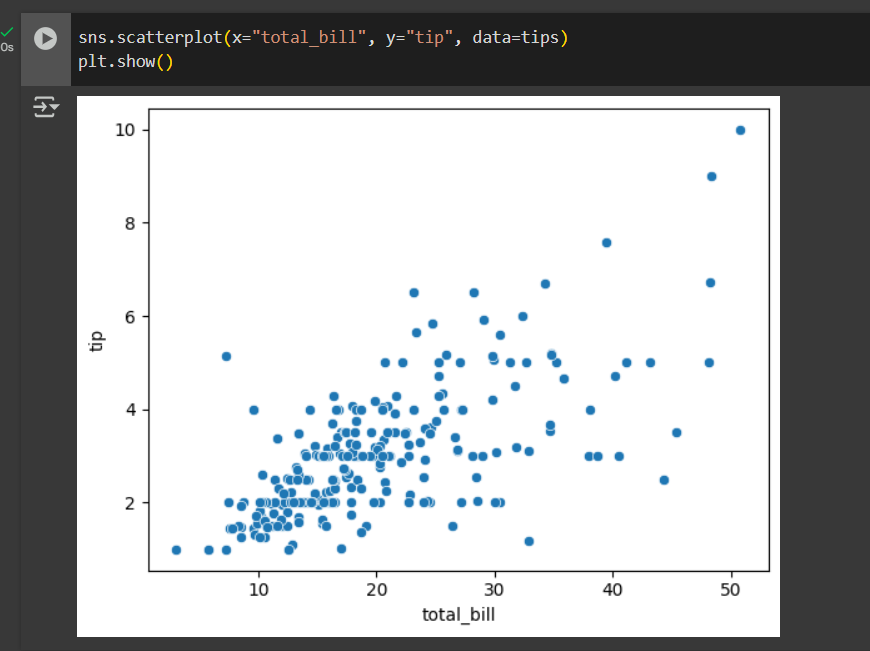


Here, **tips** dataset contains restaurant bill details including total bill, tip, gender, time etc.

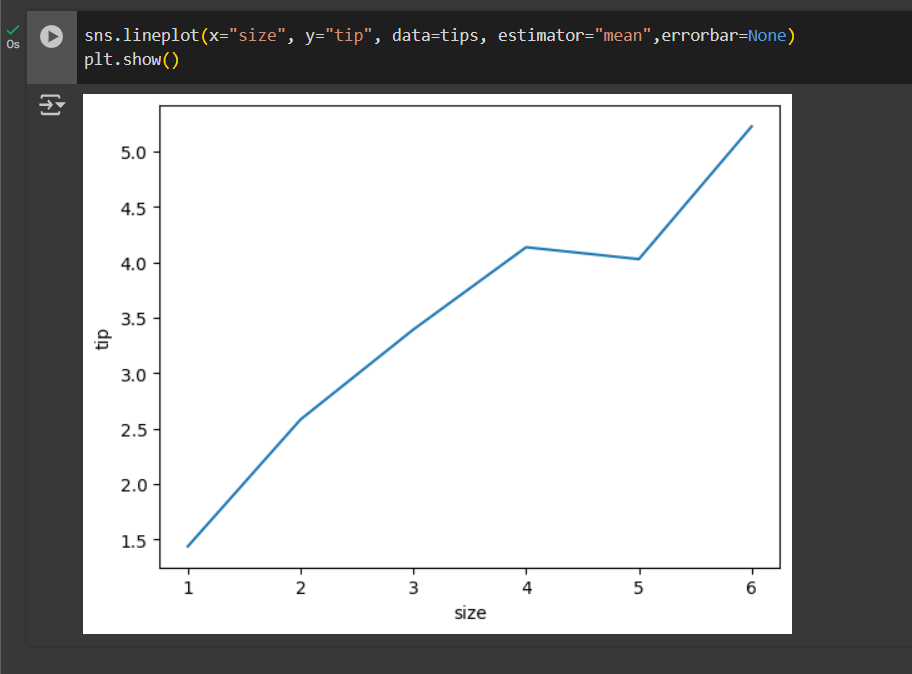
Types of graphs in Seaborn:

1. **Relational Plots**

* scatterplot()- Shows the relationship between two numeric variables. It identifying correlations, clusters, and outliers. *hue* adds colour based on a category, *style* changes marker type.

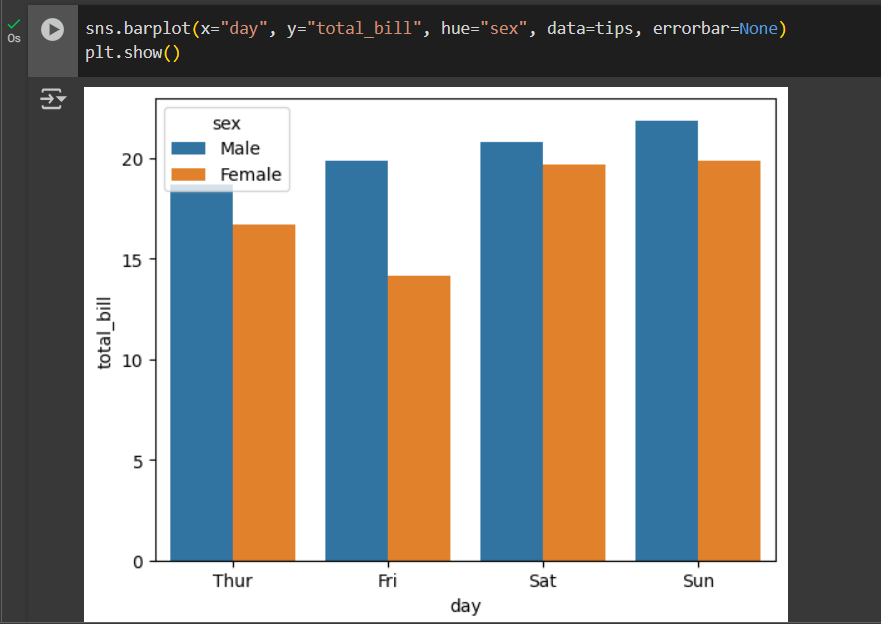


* lineplot()- Shows a trend or change over a continuous variable (like time). *estimator="mean"* computes average tip for each size, *errorbar=None* removes confidence interval shading. Excellent for trend visualization.

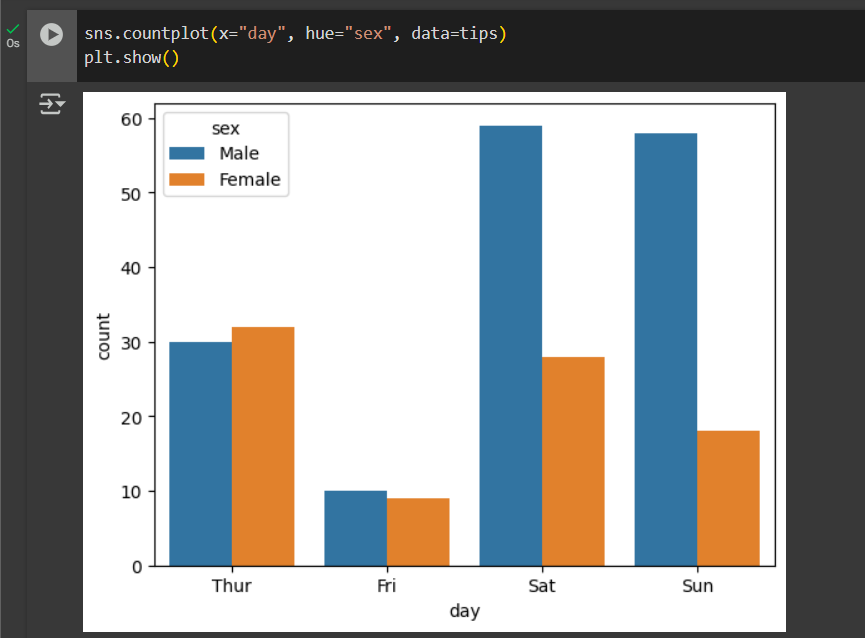


1. **Categorical Plots**

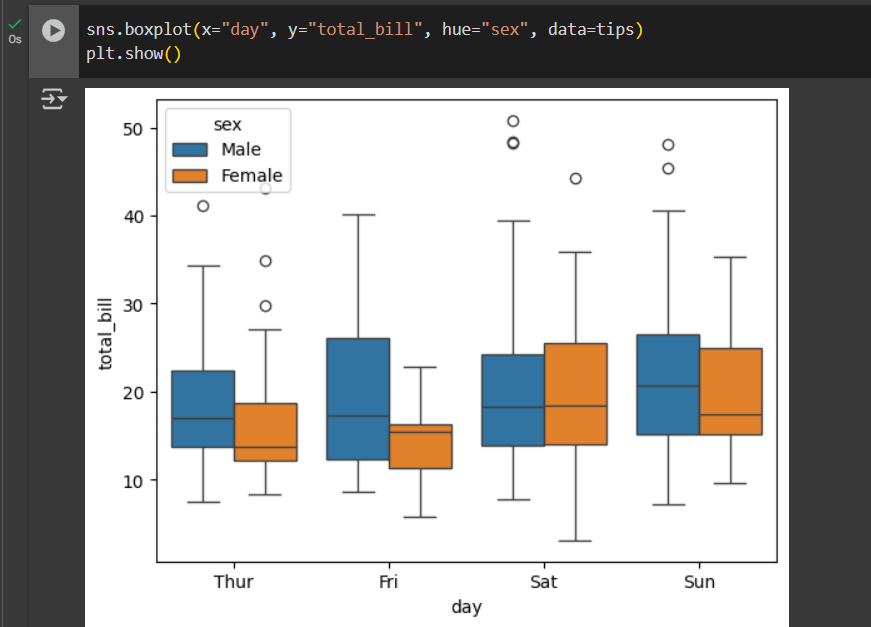
* barplot()- Displays the average value of a numeric variable across categories. Used for comparing group means. Automatically computes confidence intervals by default. Useful when comparing categories like sales by region, income by gender, etc.



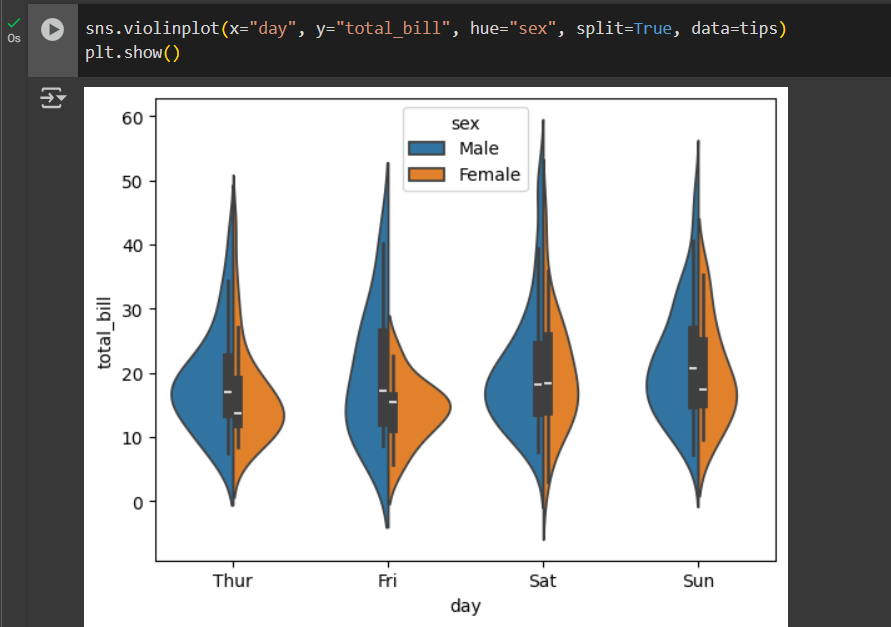
* countplot()- Shows the frequency of categories. Used for categorical distributions. Similar to barplot, but counts data instead of averaging.



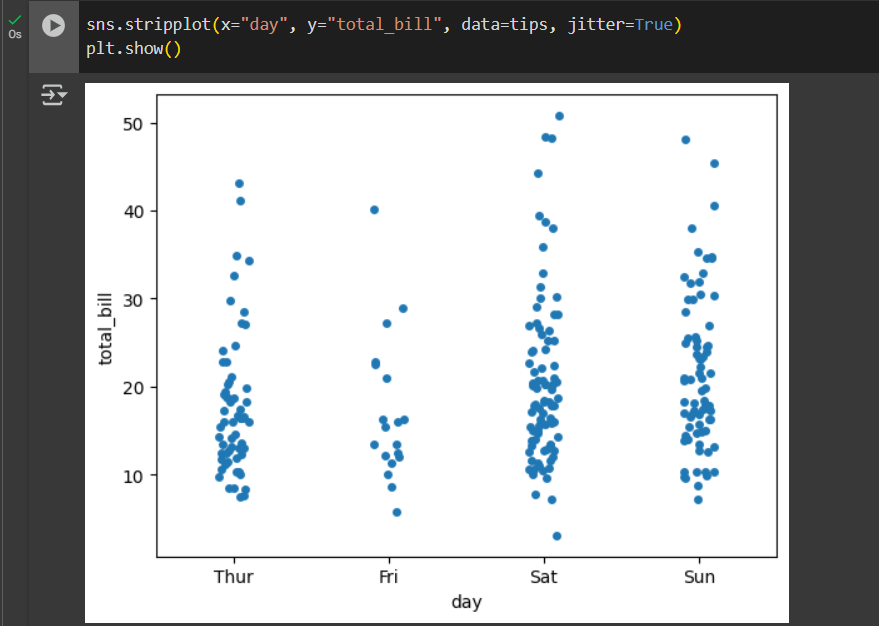
* boxplot()- Shows the spread of data using quartiles. For detecting outliers and comparing distributions.



* Middle line = median.
* Box edges = Q1 and Q3.
* Whiskers = data spread.
* Dots = outliers.
* violinplot()- Combines boxplot with KDE distribution. Best for seeing distribution shape within categories.Useful when comparing distribution + outliers.

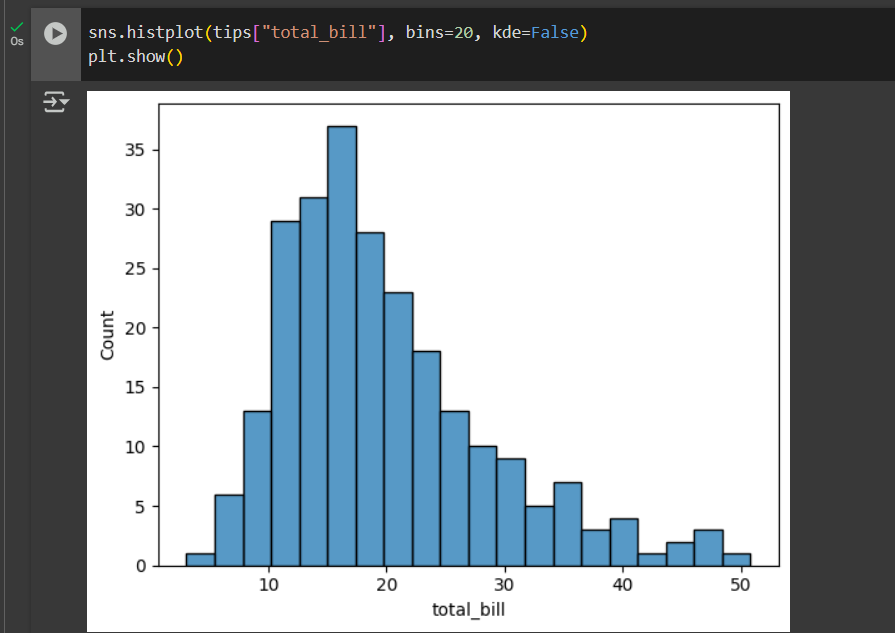


* split=True shows male vs female distributions in one violin.
* stripplot()- Displays all individual data points in a category. Used for small datasets where you want to see every value.

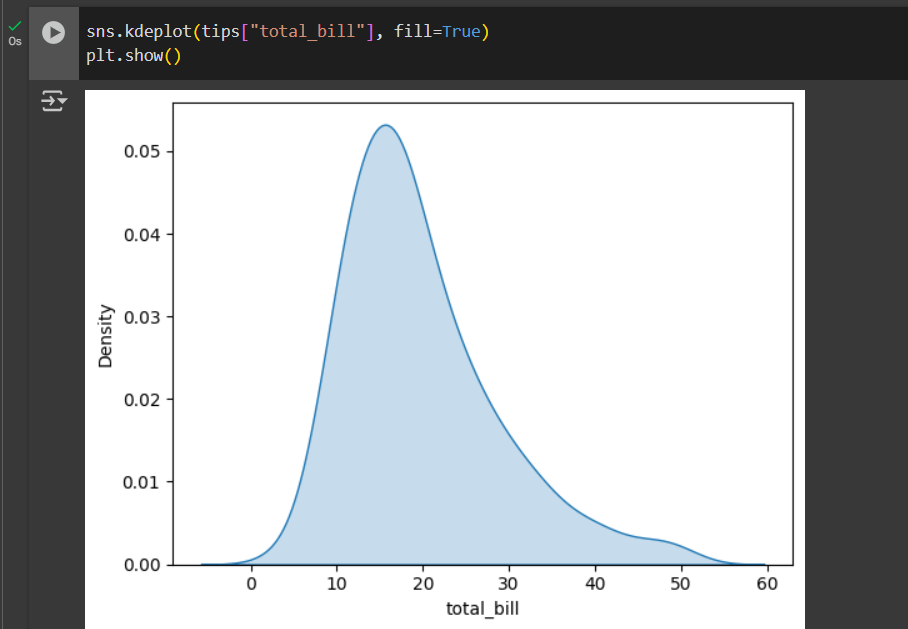


1. **Distributed Plots**

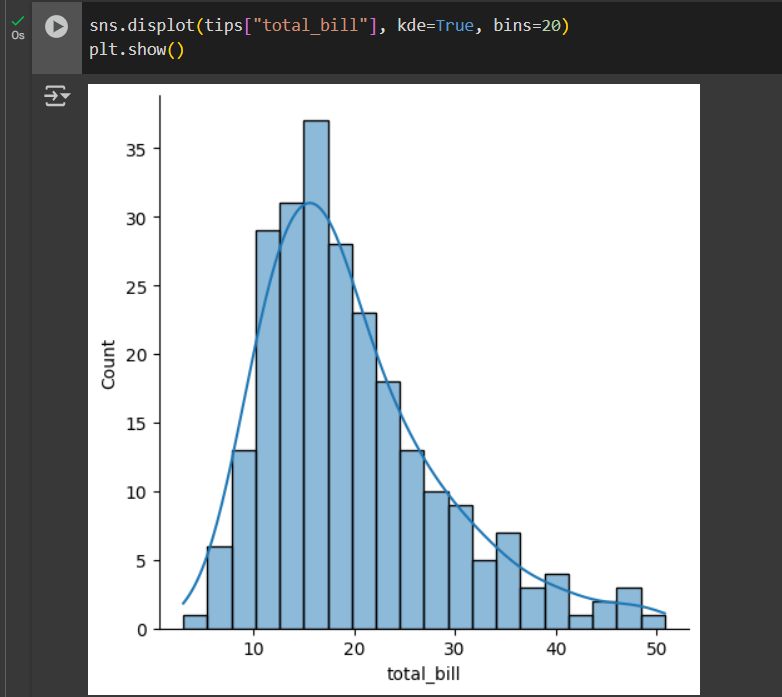
* histplot()- Shows frequency of numeric values grouped into bins. Understanding distribution shape (normal, skewed, etc.).



* kdeplot()- Shows probability density as a smooth curve. Understanding underlying data distribution.

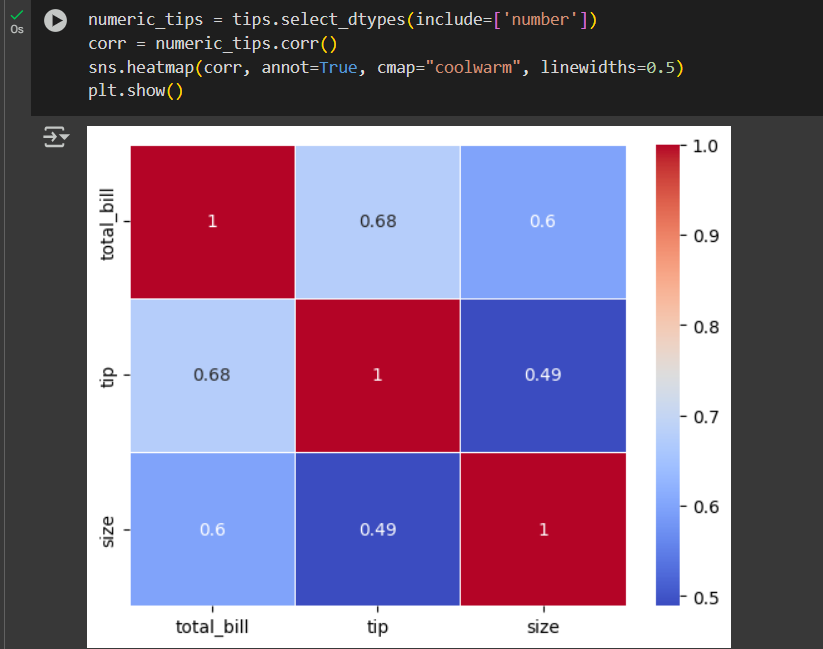


* displot()- Flexible function that can create histograms + KDE plots. Good for exploratory analysis.Combines histogram & smooth density curve.



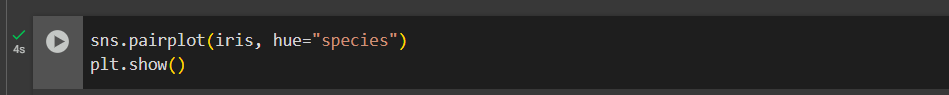
1. **Matrix Plots**

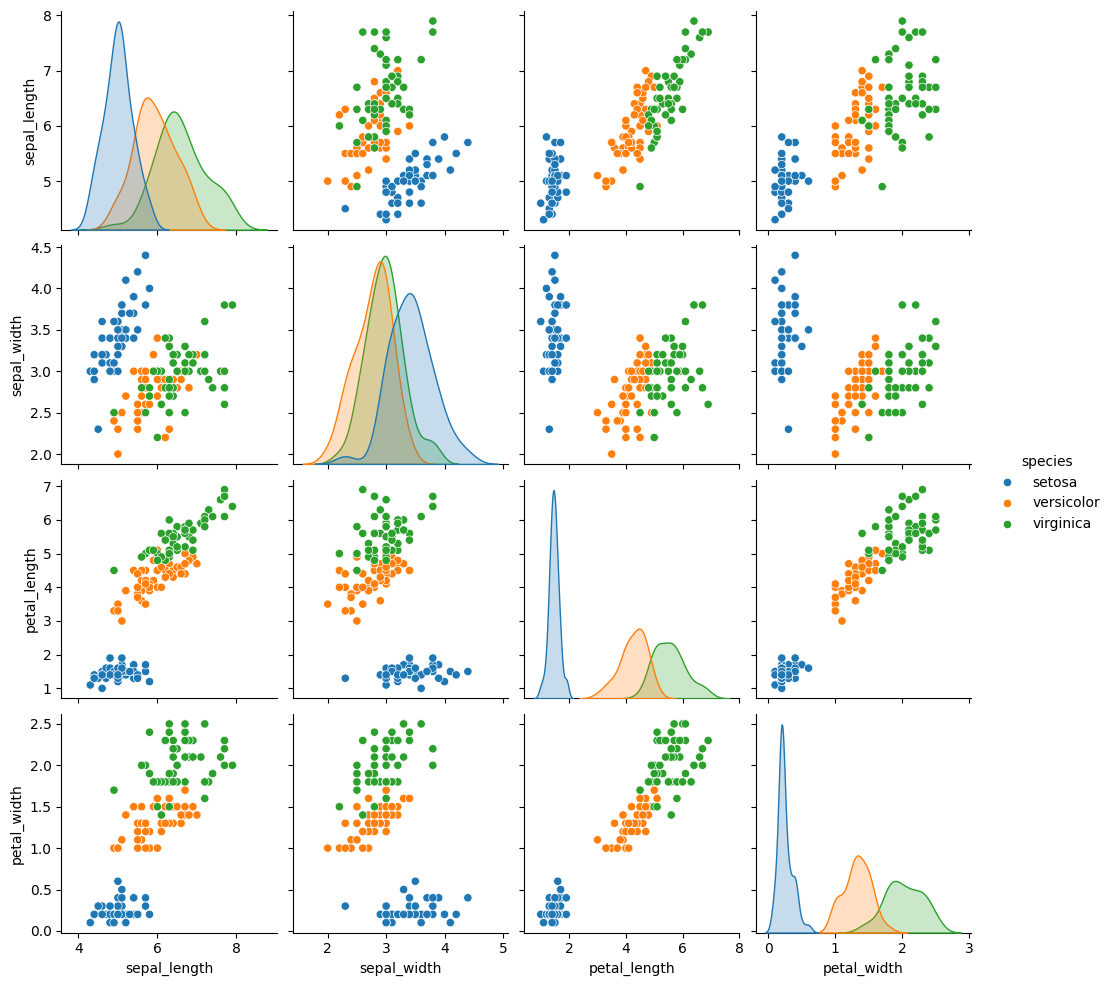
* heatmap()- Visualizes data in matrix form, values represented by colour. Shows correlations, confusion matrices, pivot tables.



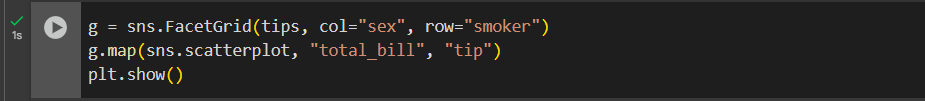
1. **Multi-plot Grids**

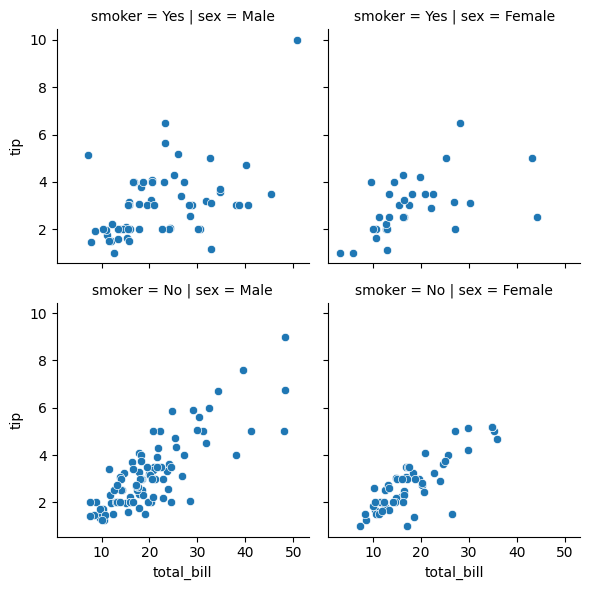
* pairplot()- Shows pairwise relationships between variables in a dataset. Used for Exploratory data analysis on multivariate data.





* Diagonal = distribution plots.
* Off-diagonal = scatter plots.
* facetGrid()- Creates multiple subplots based on categories. Visualizing subgroup patterns. Very powerful for segmentation analysis.





Pandas

Pandas is a Python library for data manipulation and analysis. It provides flexible and powerful tools for working with structured data (like tables, spreadsheets, or SQL results).

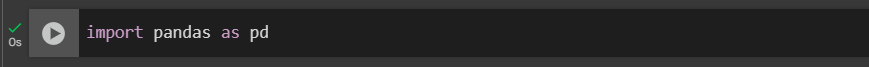
* It is built on top of NumPy.
* The main data structures are:
  + Series → One-dimensional labelled array (like a column).
  + DataFrame → Two-dimensional labelled data structure (like a table).
* Pandas makes it easy to:
  + Load data from different file formats (CSV, Excel, SQL, JSON, etc.).
  + Clean and transform messy data.
  + Perform aggregations, filtering, grouping, and statistical operations.
  + Handle time series data.

Install and import of Seaborn in python:

We install Pandas (if not installed) (in the command prompt):



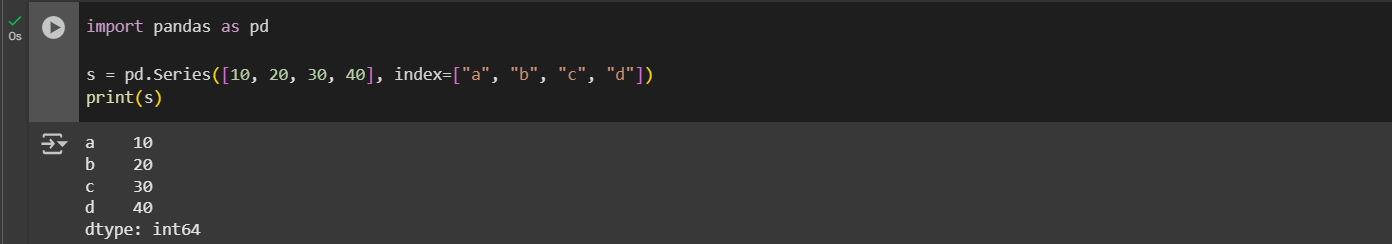
import Pandas is done by:



**Core Data Structures:**

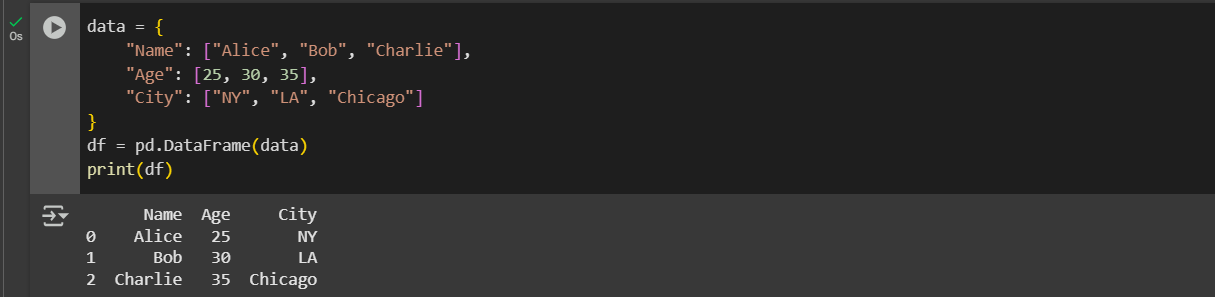
Series

* A 1D labelled array that can hold any data type (int, float, string, etc.).

****

DataFrame

* A 2D table of rows and columns (like an Excel sheet).



**Loading and Saving Data:**

1. **CSV Files:**

* Load CSV: CSV (Comma Separated Values) files are the most widely used for storing tabular data. Each line in the file is a row. First row (by default) is used as column headers.
* Save CSV: Saves DataFrame row by row. Each row becomes a line, with values separated by commas.

1. **Excel Files:**

* Load Excel: Excel is used widely in business reporting. Reads Excel file using an engine (like openpyxl). Loads specified sheet into a DataFrame.





* Save Excel: Creates an Excel workbook. Saves DataFrame into given sheet.



1. **JSON Files:**

* Load Json: JSON is widely used for APIs and web data. Reads JSON structure (list of dictionaries, nested JSON). Converts it into rows and columns.



Ex: data.json

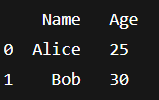
[

{"Name": "Alice", "Age": 25},

{"Name": "Bob", "Age": 30}

]

Output:



* Save Json: Converts DataFrame rows into JSON objects. Saves them in the chosen format.



Example Output:

‘output.json’

{"Name":"Alice","Age":25}

{"Name":"Bob","Age":30}

**Basic Data Exploration Methods:**

***Data exploratory*** methods are techniques used to understand and summarize the main characteristics of a dataset before applying any complex analysis or models. They help in identifying patterns, trends, and relationships within the data while also revealing possible issues such as missing values, outliers, or inconsistencies. The goal is to get a clear picture of the data’s structure and quality, so that better decisions can be made in the later stages of analysis or modelling. Some of the methods used for data exploration are given below.

1. **df.head(n)**- Shows the first n rows (default = 5). Used when you want to quickly peek at the dataset structure.
2. **df.tail(n)**- Shows the last n rows (default = 5). Useful for checking bottom rows or ensuring file loaded correctly.
3. **df.shape**- Returns tuple (rows, columns).
4. **df.info()-** Shows column names, data types, null values, and memory usage. Very useful for detecting missing values or wrong data types.
5. **df.describe()**- Gives summary statistics (count, mean, std, min, max, quartiles) for numeric columns. Helps identify ranges, outliers, and distributions.
6. **df.colums**- Lists column names. Useful when dataset has many columns.
7. **df.index**- Shows row index/labels. Important when working with custom row labels or time series.

**Selecting & Filtering Data:**

Selecting and filtering data in pandas is all about choosing the parts of a dataset that are most relevant to your analysis.

* ***Selecting*** data means picking out specific columns, rows, or cells from a DataFrame or Series. For example, you might want only the "Name" column or the first 10 rows of data. It’s like saying, *“show me this piece of the table.”*
* ***Filtering*** data goes one step further: it involves keeping only the rows that meet certain conditions. For example, you might filter a dataset to include only people over 25 years old or only products with prices above a certain amount. It’s like saying, *“show me the rows that match these rules.”*

Together, selection and filtering allow you to zoom in on the exact data you need, making large datasets easier to manage and analyse. Some of the methods used to select and filter are given below.

1. **Selecting Columns:**

* Single Column:

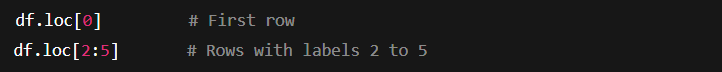


* Multiple Column:



1. **Selecting Rows:**

* By label (using **.loc**):



* By position (using **.iloc**):



1. **Conditional Filtering:** You can filter rows using boolean conditions.

****

1. **Multiple Conditions:**

* **AND** condition (**&**):



* **OR** condition (**|**):



* **NOT** condition(**~**):



1. **Filtering with** isin()**:** Select rows where column value belongs to a list.



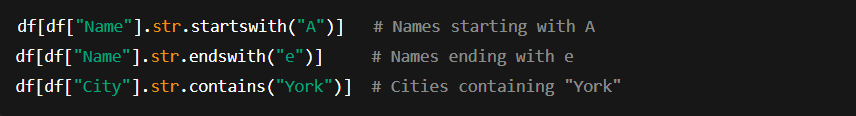
* Returns only rows where City is either NY or LA.

1. **Filtering with** between()**:** Check if values fall between a range.

****

* Selects rows where Age is between 20 and 30 (inclusive).

1. **String-based Filtering:** For text columns, you can filter using string methods. Very useful for searching patterns in text

****

1. **Filtering with Null Values:**

* Find missing values:



* Find non-missing values:



1. **Filtering with Query Method:** The **query()** method allows SQL-like filtering. More readable than multiple brackets for complex conditions.



1. **Filtering with** apply() **for Custom Functions:** Apply a custom function to filter rows.

****

**Data Cleaning:**

***Data cleaning*** is the process of preparing raw data for analysis by fixing or removing errors, inconsistencies, and irrelevant information. It involves tasks like handling missing values, correcting data types, removing duplicates, standardizing formats, and dealing with outliers to ensure accuracy and consistency. Clean data is crucial because even the most advanced models or visualizations can give misleading results if the underlying data is messy. In short, data cleaning ensures that the dataset is reliable, accurate, and ready for meaningful analysis. Some of the methods used to select and filter are given below.

1. **Handling Missing Values:**

* Remove missing values:



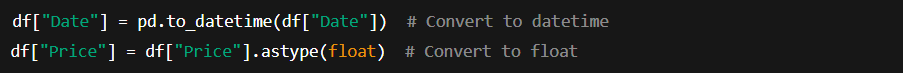
* Fill missing values:



1. **Removing Duplicates:**



1. **Correcting Data types:**



1. **Handling Outliers:**

****

1. **Standardizing Text Data:**

****

1. **Renaming Columns for Consistency:**



**Data Transformation:**

***Data transformation*** is the process of converting data from one format, structure, or value representation into another to make it more suitable for analysis. It includes tasks like normalizing values, aggregating data, encoding categorical variables, scaling numbers, or reshaping tables. Transformation helps align data with the requirements of analytical methods and models, making it easier to uncover patterns and insights. In short, it ensures that data is in the right shape and form for effective processing and interpretation. Some of the methods used to select and filter are given below.

**Grouping and Aggregation:**

***Grouping*** is the process of splitting a dataset into smaller groups based on the values of one or more columns. For example, in a sales dataset, you might group records by “Region” or “Product Category.” This allows you to organize data into meaningful categories before applying any analysis.

***Aggregation*** is the process of applying summary functions to groups of data to get condensed insights. Common aggregations include finding the sum, average, count, maximum, or minimum of values within each group. For instance, after grouping sales data by region, you can aggregate to find the total sales per region.

Together, grouping organizes the data, and aggregation summarizes it. Some of the methods used to select and filter are given below.

**Merging and Joining:**

***Merging*** is the process of combining two datasets based on common columns or keys, similar to how you would match rows in a database using relationships. For example, you might merge a customer table and an order table using the "Customer ID." This allows you to bring related information together into one DataFrame.

***Joining*** is very similar to merging, but it is usually done based on the row index (or a combination of index and column keys) rather than explicitly specifying a column. In pandas, join() is a shortcut for combining datasets when one dataset’s index directly relates to another’s.

In short, merging is more flexible and column-based (like SQL joins), while joining is more index-based and convenient when working with aligned datasets. Some of the methods used to select and filter are given below.

**Time Series Data Handling:**

***Time series data handling*** is the process of working with data that is indexed or ordered by time. It involves managing datasets where observations are recorded at regular intervals, such as daily stock prices, hourly weather readings, or monthly sales figures. Handling time series includes tasks like converting strings to proper datetime formats, setting dates as the index, resampling data into different time intervals (e.g., daily to monthly), handling missing timestamps, and performing rolling or shifting operations to study trends and patterns over time. In short, time series data handling helps us analyse changes, detect patterns, and make forecasts based on time-dependent information. Some of the methods used to select and filter are given below.

**Comparison between Seaborn and Pandas:**

| **Feature** | **Seaborn** | **Pandas** |
| --- | --- | --- |
| **Purpose** | Specialized in statistical data visualization | Primarily for data manipulation, cleaning, and analysis |
| **Built on** | Built on Matplotlib and integrates with Pandas | Built on NumPy, with limited plotting (uses Matplotlib under the hood) |
| **Ease of Use for Visualization** | High-level API → fewer lines of code for complex plots | Basic plots are quick but lack customization; more manual work needed |
| **Types of Plots** | Rich collection: histograms, KDE, violin, box, heatmaps, pairplots, regression plots | Basic plots: line, bar, histogram, scatter, area, box, pie |
| **Customization** | Automatically applies attractive styles and color themes | Very limited styling; plain by default |
| **Integration with DataFrames** | Directly works with Pandas DataFrames and Series | Native since Pandas itself manages the DataFrame |
| **Best Use Case** | When you want insightful, publication-quality visualizations with minimal effort | When you want to quickly explore data while analyzing/manipulating datasets |
| **Learning Curve** | Slightly higher, but powerful once learned | Easier to start with (since plotting is integrated with analysis) |