**MATPLOTLIB (Plotting library)**

**MATPLOTLIB IN PYTHON**

* Matplotlib is Python's primary plotting library.
* Widely used for data visualization.
* Fast and supports export to popular image formats.
* Enables creation of diverse plots like line graphs, scatter plots, histograms, and more.
* Capable of 3D charting.

**NEED OF MATPLOTLIB**

* Humans grasp visual representations better than textual data.
* Visualization aids in efficient data analysis and decision-making.
* Matplotlib is crucial for creating various plots and graphs in Python.

**DATA VISUALIZATION**

* Data visualization is a new term. It expresses the idea that involves more than just representing data in the graphical form (instead of using textual form).

**WHAT IS DATA?**

Data refers to raw facts, figures, or information, often in the form of numbers, text, or multimedia

**TYPES OF DATA**

**Numerical Data:**

* Involves measurable quantities and is expressed with numbers.
* Examples include temperatures, ages, heights, and income.

**Categorical Data:**

* Descriptive and non-numeric information that is typically divided into categories.
* Examples include colours, gender, types of fruits, or opinions.

**NOTE**

Choosing graphs based on types of data

**INSTALLING MATPLOTLIB**

pip install matplotlib

**IMPORTING MATPLOTLIB:**

import matplotlib.pyplot as plt

**LINE PLOTS**

* The line plots are simplest and most widely used data visualization techniques.
* A line plots displays information as a series of data points or markers connected by straight lines.
* You can customize the shape, size, color, and other aesthetic elements of the lines and markers for better visual clarity.

**PLT.PLOT() FUNCTION:**

* The plot() function in Matplotlib is a versatile function used to create a wide range of plots, including line plots, scatter plots, and more.
* It is the primary function for creating 2D plots.
* <https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html>

**Parameters:**

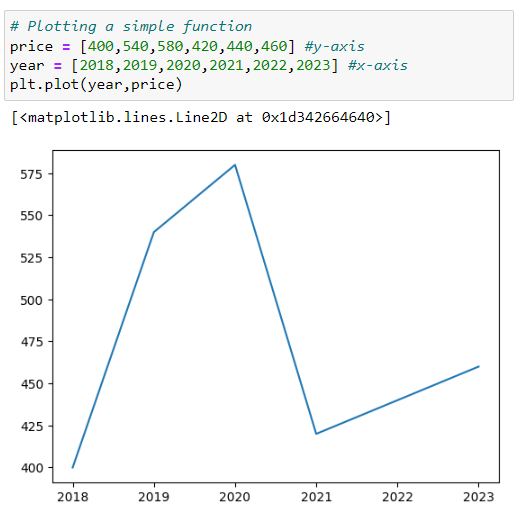
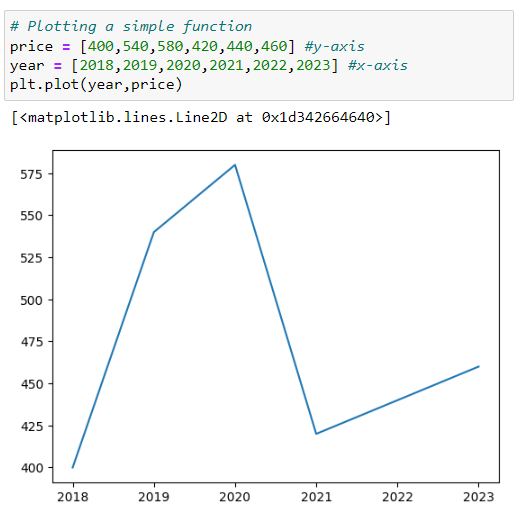
|  |  |
| --- | --- |
| **x** | The x-coordinates of the data points. This can be a list, array, or other iterable. |
| **y** | The y-coordinates of the data points. Like x, it can be a list, array, or other iterable. |
| **format** | (Optional) A format string that defines the line and marker styles. It is a shorthand notation to specify color, marker, and line style in a single string. For example, 'b-' denotes a blue solid line, 'ro' denotes red circles, etc. |
| **color or c** | The color of the line, colors like hexa code & normal colors |
| **linestyle or ls** | The line style, such as '-' for solid, '--' for dashed, ':' for dotted, etc. |
| **marker** | The marker style, such as 'o' for circles, 's' for squares, etc. |
| **markersize or ms** | The size of markers. |
| **label** | A label for the plot, used for creating a legend. |
| **alpha** | The transparency of the plot (0 to 1). |
| **linewidth or lw** | The width of the line. |
| **markeredgecolor or mec** | Set the edge color for markers |
| **markeredgewidth or mew** | Set the edge width for markers |
| **markerfacecolor or mfc** | Set the fill color for markers |

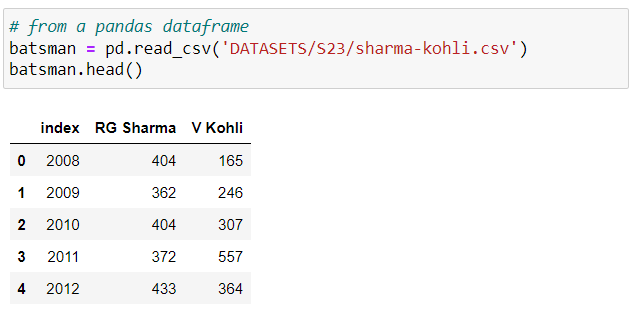
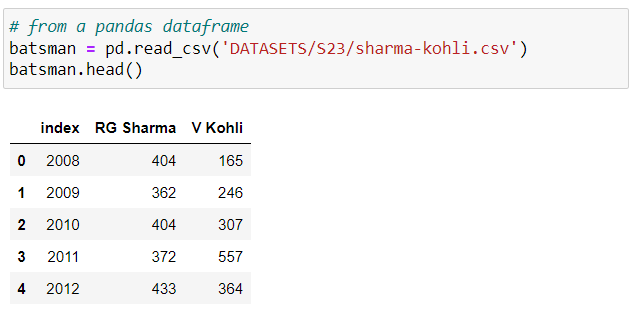
* Check out the documentation for **plt.plot** to learn more:

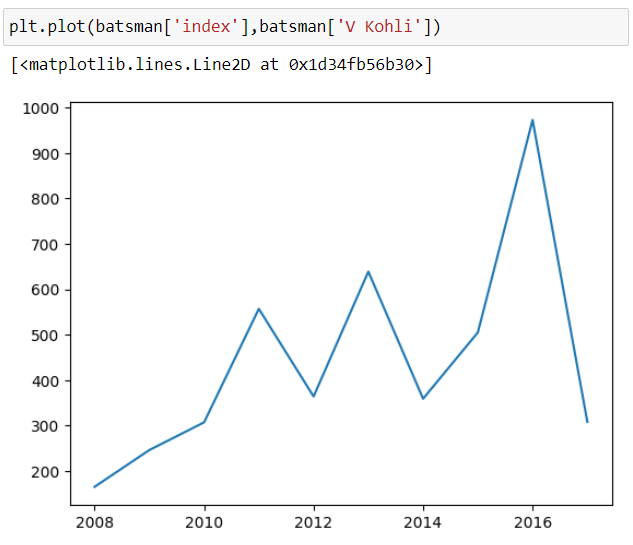
<https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot>

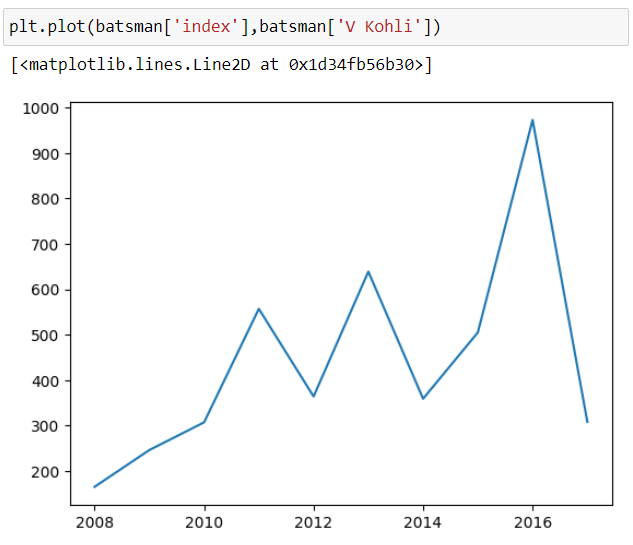
**2D LINE PLOTS**[**¶**](http://localhost:8888/notebooks/SESSION%2023%20-%20MATPLOTLIB.ipynb#2D-Line-plots)

* 2D line plots are used in bivariate analysis, specifically applied to two columns.
* They are suitable for cases where either both columns are numerical or one is categorical and the other is numerical. - - Generally, we use 2D line plots for scenarios such as time series data, for example, in stock market analysis or tracking a company's revenue.



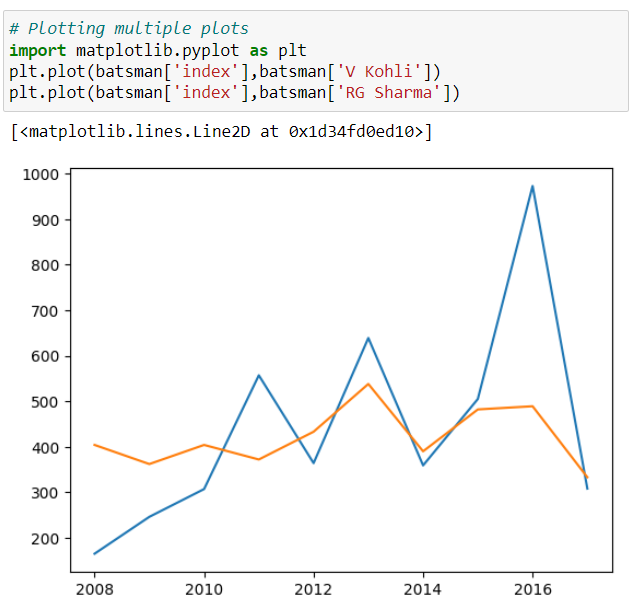
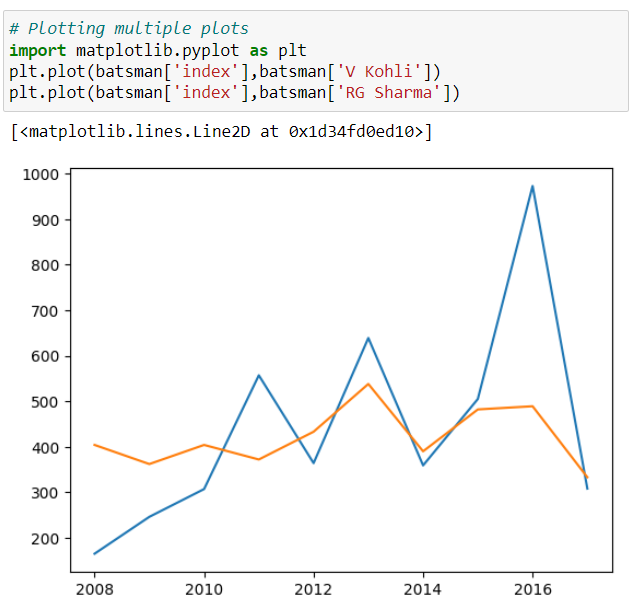






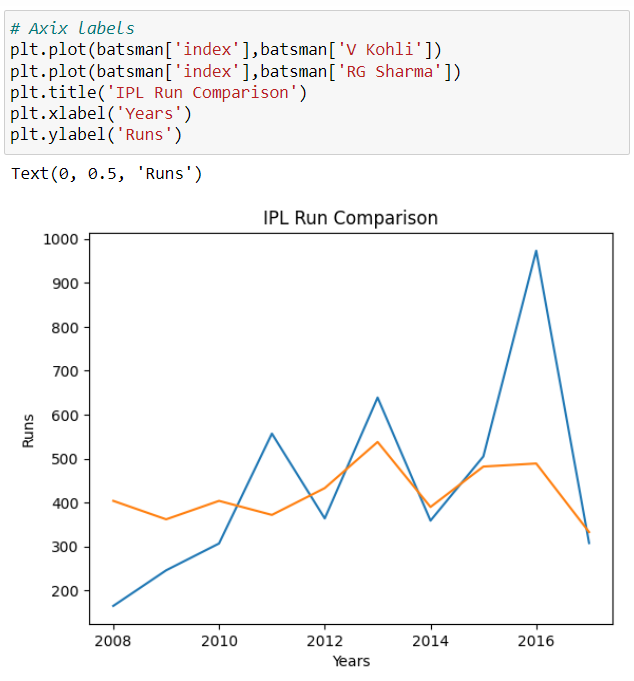
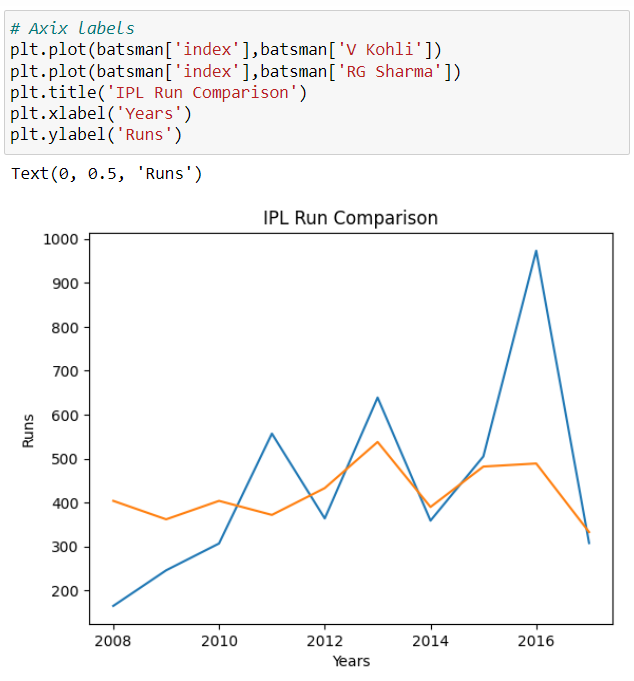
**PLOTTING MULTIPLE LINES**

You can invoke the **plt.plot()** function once for each line to plot multiple lines in the same graph.

****

**AXIS LABELS**

We can add labels to the axes to show what each axis represents using the **plt.xlabel()** and **plt.ylabel()** methods.

****

**COLORING**

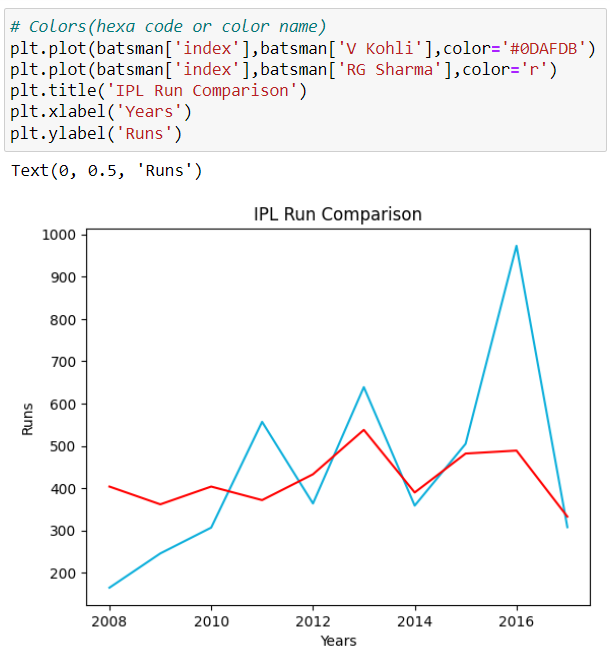
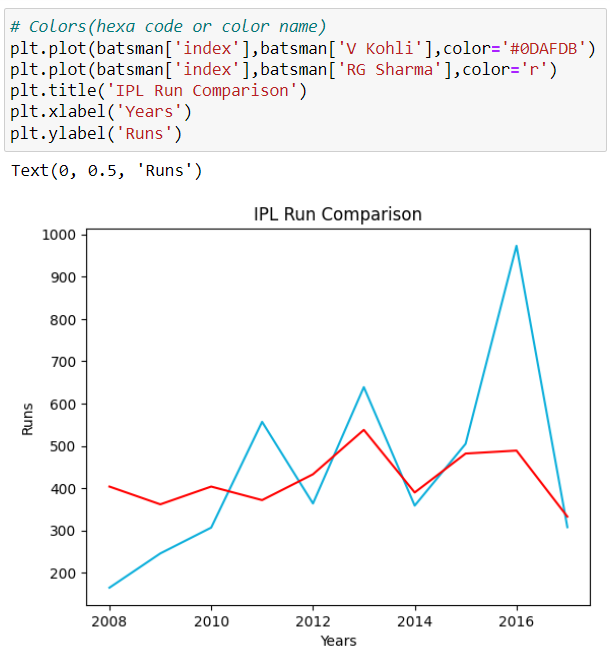
* In Matplotlib, you can customize the color of your plot using the color parameter in the **plt.plot()** function.
* The color parameter can take various forms:

**String representation a color names:**

* 'b' for blue
* 'g' for green
* 'r' for red
* 'c' for cyan
* 'm' for magenta
* 'y' for yellow
* 'k' for black
* 'w' for white

**Hexadecimal color codes:**

* '#RRGGBB' where RR, GG, and BB are hexadecimal values for red, green, and blue components.

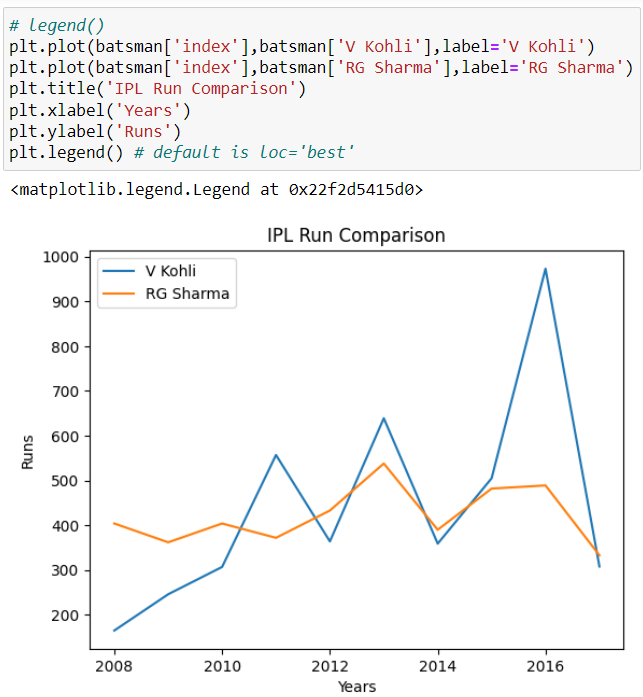
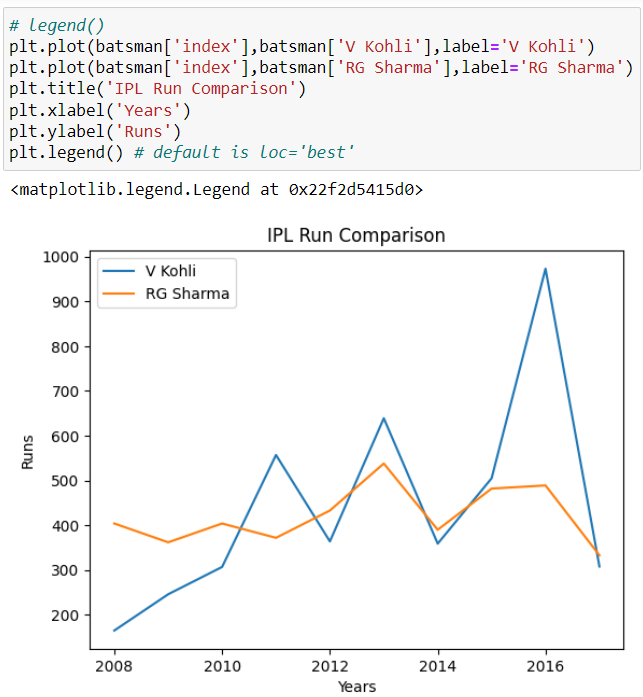
****

**LEGEND**

* To differentiate between multiple lines, we can include a legend within the graph using the **plt.legend()** function.
* We can also set a title for the chart using the **plt.title** function
* Here are some commonly used values for the **loc parameter:**

**String Values:**

* **'best' or 0:** Automatically choose the best location to avoid overlapping with the plot.
* **'upper right' or 1**
* **'upper left' or 2**
* **'lower left' or 3**
* **'lower right' or 4**
* **'right' or 5**
* **'center left' or 6**
* **'center right' or 7**
* **'lower center' or 8**
* **'upper center' or 9**
* **'center' or 10**



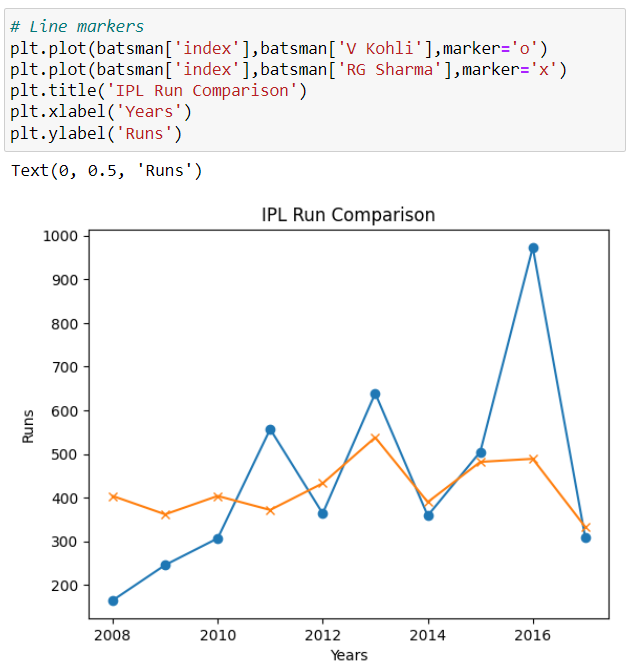
#### LINE MARKERS

* We can also show markers for the data points on each line using the **marker** argument of **plt.plot.**
* Matplotlib provides many different markers, like a circle, cross, square, diamond, etc.
* Here is a short list of commonly used markers:

|  |  |
| --- | --- |
| **'o':** | Circle marker - widely used for scatter plots and line plots. |
| **'s'** | Square marker - provides a clear and simple marker for data points. |
| **'+'** | Plus marker - used to highlight specific points of interest. |
| **'x'** | X marker - often used for denoting errors or special points. |

* You can find the full list of marker types here:

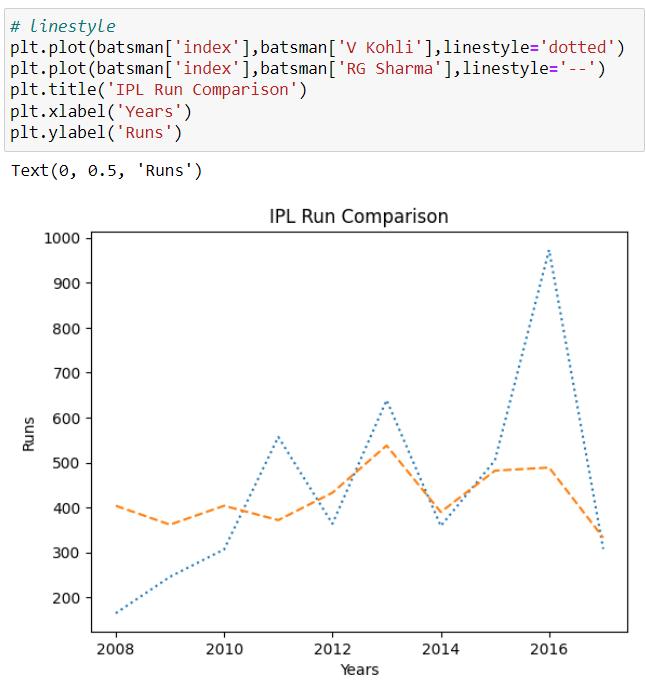
<https://matplotlib.org/stable/api/markers_api.html>

****

**LINESTYLE**

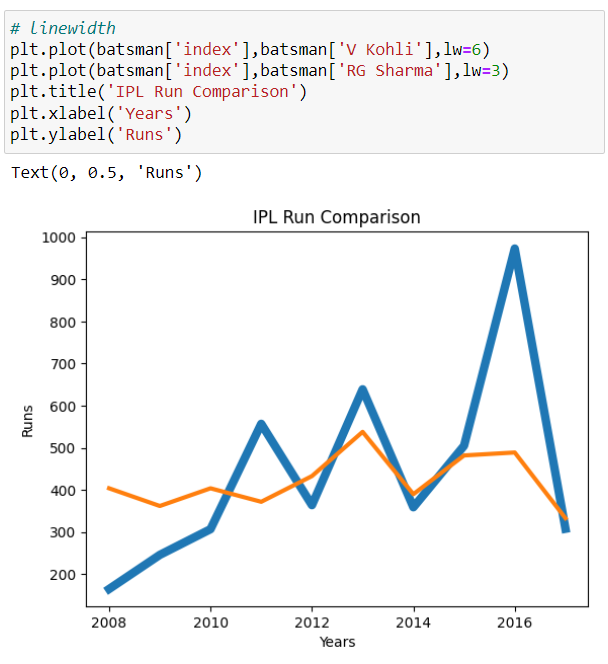
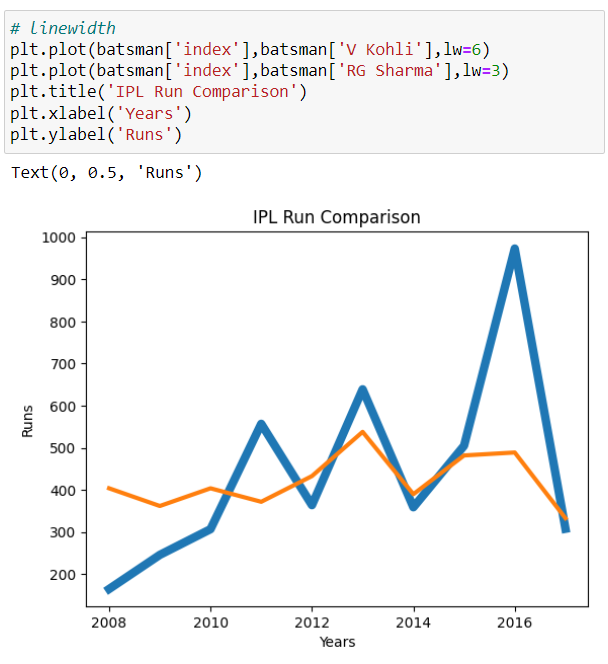
* Matplotlib, the linestyle parameter is used to specify the style of the line in a plot.
* Here are some commonly used linestyle options:

|  |  |
| --- | --- |
| **' - '** | Solid line (default) |
| **' -- '** | Dashed line |
| **' : ' or ' dotted ’** | Dotted line |
| **' -. ' or 'dashdot'** | Dash-dot line |



**LINEWIDTH**

In Matplotlib, the **linewidth** or **lw** parameter is used to control the thickness of the lines in a plot.



**LIMITING AXES**

* You can limit the axis ranges in a Matplotlib plot using the **plt.xlim()** and **plt.ylim()** functions.
* These functions allow you to set the minimum and maximum values for the x-axis and y-axis, respectively.
* Limiting axes can be crucial for several scenarios where you want to focus on specific aspects of your data or highlight particular trends.
* Here are some common scenarios where limiting axes can be beneficial:

**Zooming In on Specific Regions:**

To analyze detailed patterns or trends in a large dataset, limiting the axes helps focus on a specific region of interest.

**Comparing Subsets of Data:**

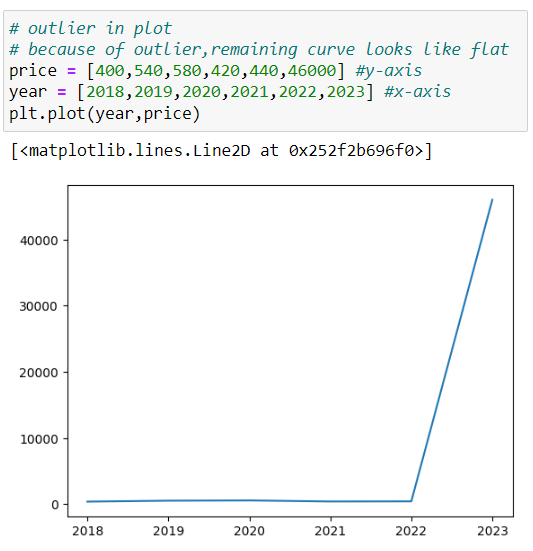
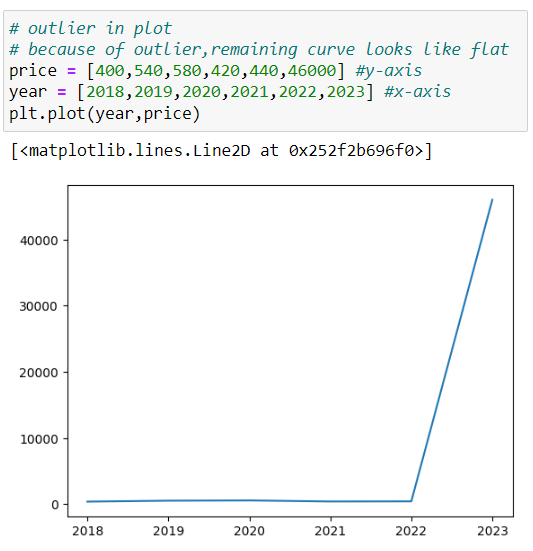
When comparing model performance, limiting axes allows a focused analysis on a specific range of input variables or target values.

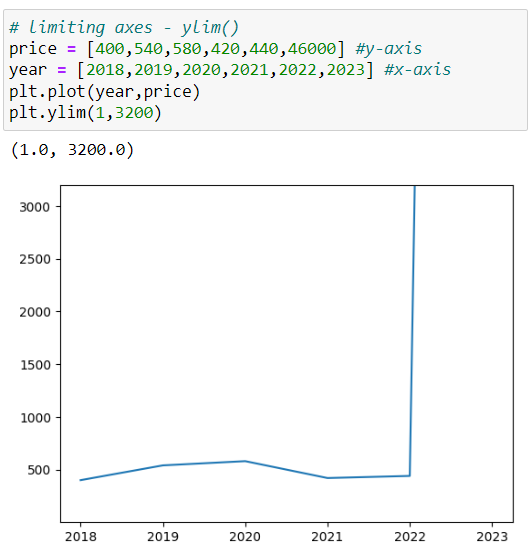
**Highlighting Outliers:**

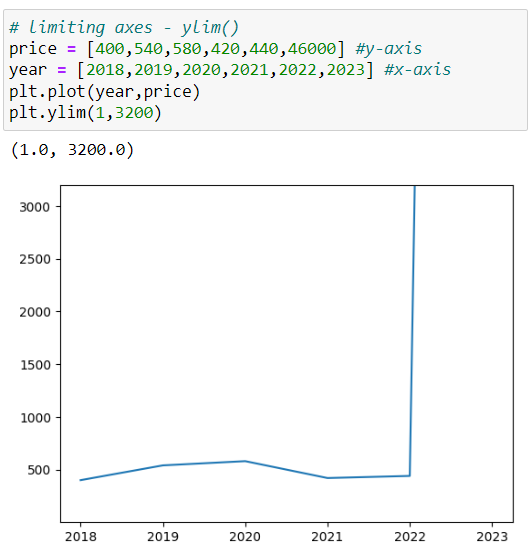
To emphasize the main cluster of data points and disregard outliers, limiting axes is useful.

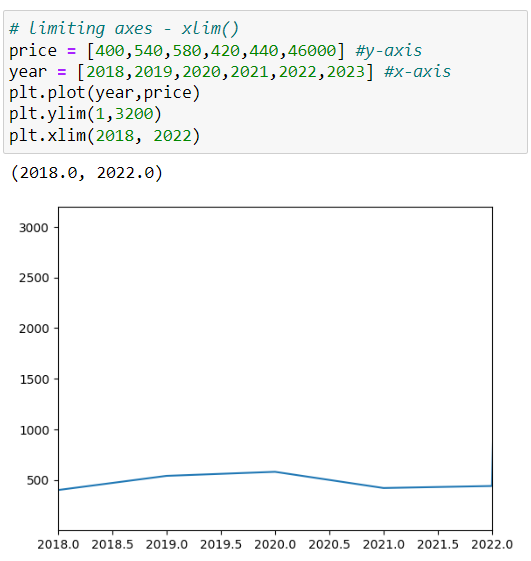
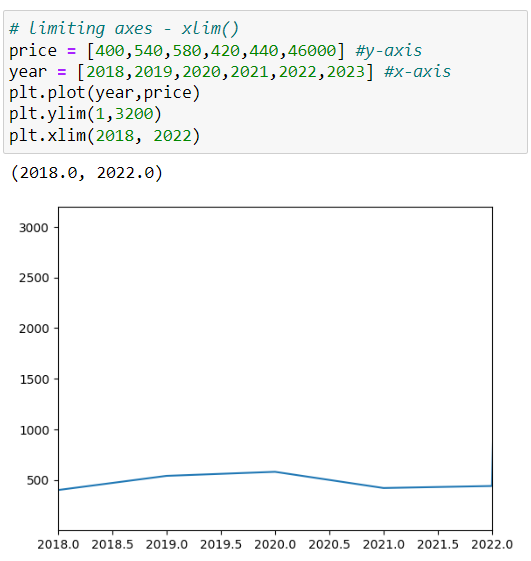
**Improving Visualization of Trends:**

In time series data, limiting axes to a specific time period enhances the visualization and analysis of trends.



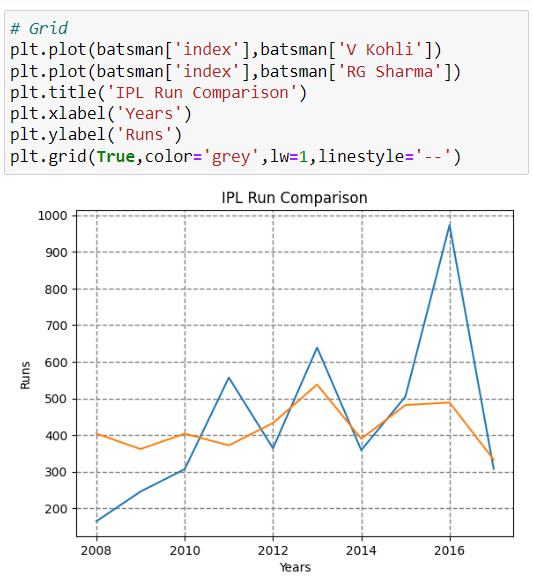


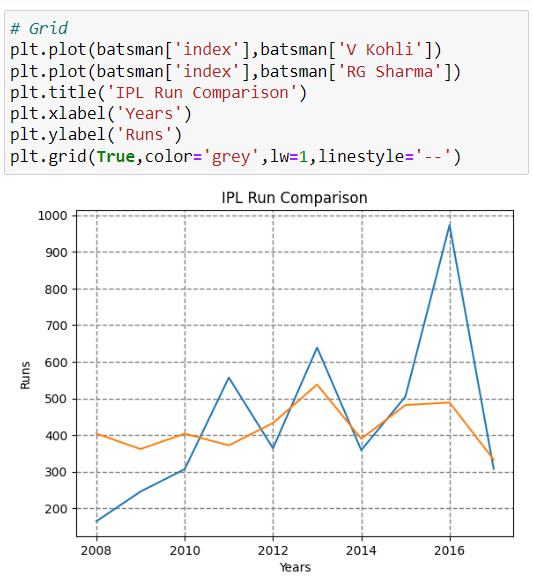


****

**GRID**

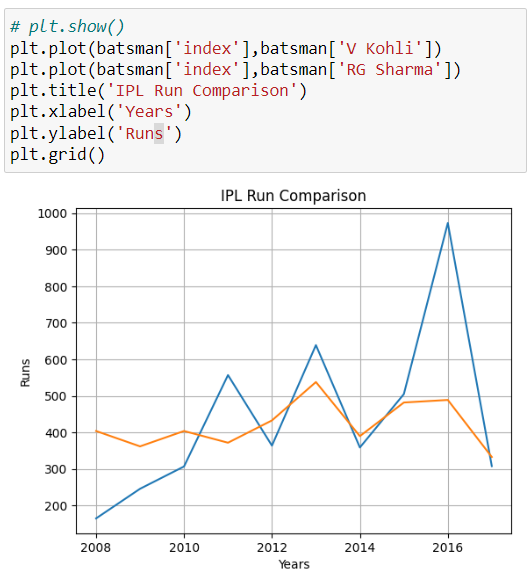
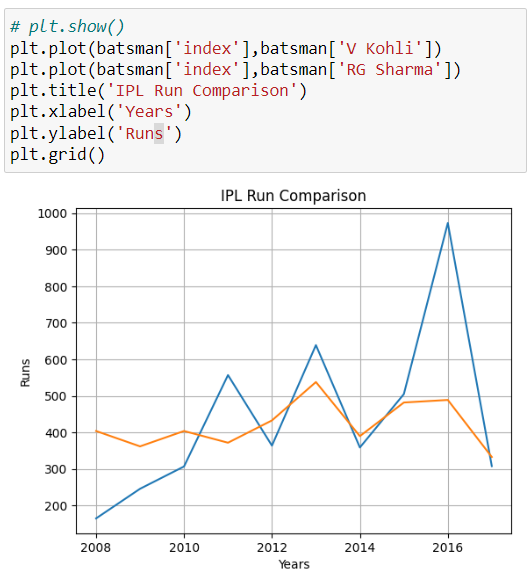
* The **plt.grid()** function in Matplotlib has a simple syntax.
* It can be used without any arguments to turn on the grid or with optional arguments to customize its appearance.
* **Syntax: plt.grid(True, linestyle, linewidth=0.5, color='gray')**

****

****

**SHOWING PLOTS**

* **plt.show()** displays Matplotlib plots in scripts or Jupyter cells.
* It's typically the last command after creating a plot.
* The **%matplotlib inline** command in Jupyter enables inline rendering of Matplotlib plots.
* Without it, Jupyter may not display plots inline, requiring explicit use of **plt.show().**



**SCATTER PLOTS**

* A scatter plot is a visualization technique commonly used to display the relationship between two or more numeric variables.
* In a scatter plot, individual data points are represented by dots, and the position of each dot on the horizontal and vertical axes corresponds to the values of two numeric variables.

**Key characteristics**

**Bivariate Analysis:**

* Scatter plots are particularly useful for bivariate analysis, allowing us to examine the relationship between two numerical variables.

**Numeric Columns Only:**

* Scatter plots only work with numerical columns; both the x and y axes must contain numeric values.

**Use Case - Finding Correlation:**

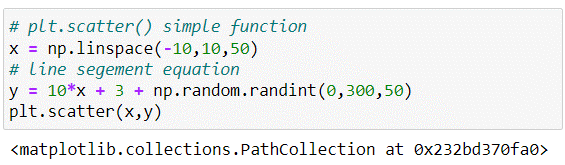
* A common use case for scatter plots is to identify and visualize the correlation between two quantities.
* By observing the distribution of points on the plot, patterns and trends can be analysed.

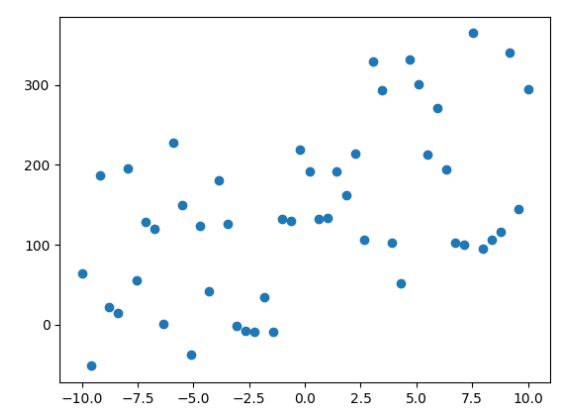
**PLT.SCATTER() FUNCTION**

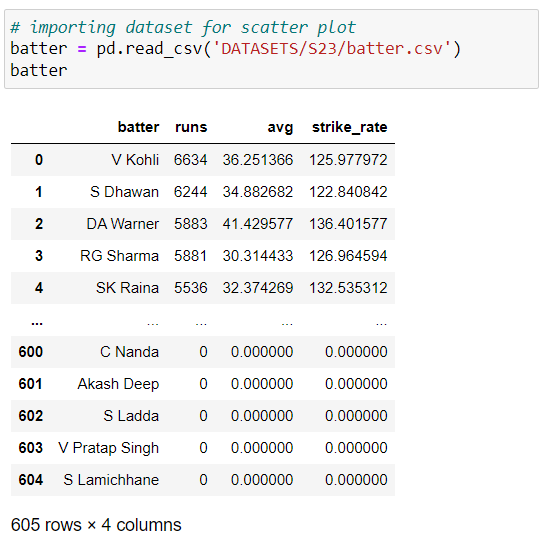
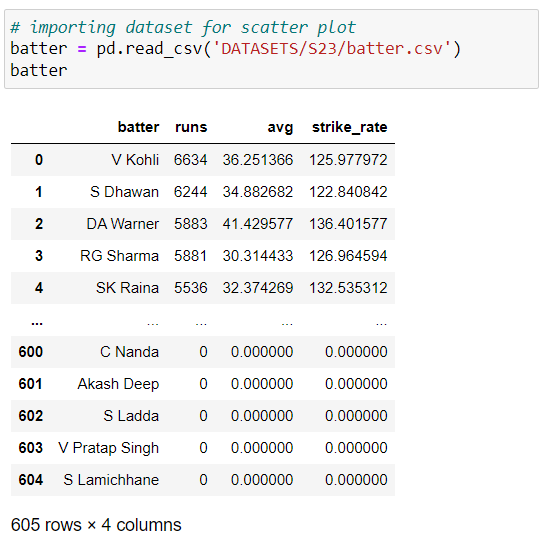
* The plt.scatter() function in Matplotlib is used to create scatter plots.
* Here's the syntax and a brief explanation of its parameters:
* **Syntax: plt.scatter(x, y,s,c,marker,cmap,norm, vmin, vmax, alpha, edgecolors, linewidths, label)**

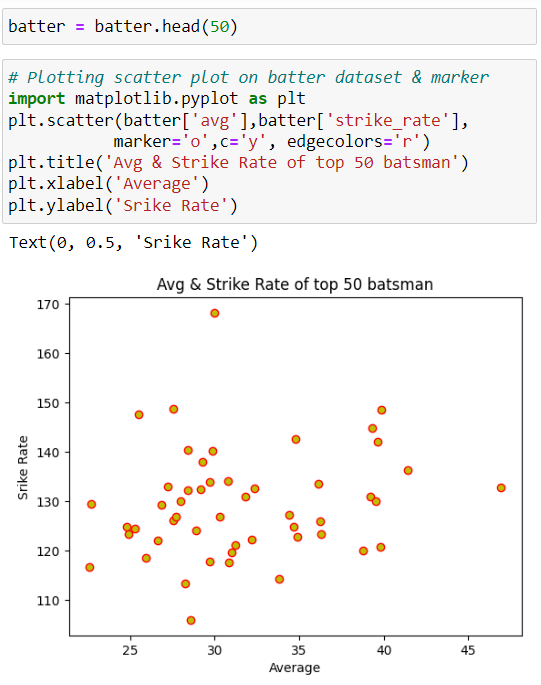
**Parameters:**

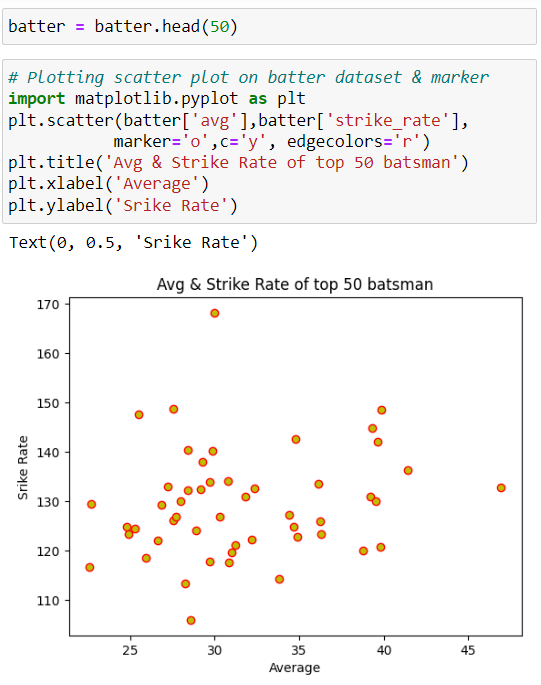
|  |  |
| --- | --- |
| **x, y:** | Numeric values representing the data points. |
| **s:** | Marker size (scalar or array-like, optional). |
| **c:** | Marker color (color or sequence, optional). |
| **marker:** | Marker style (string, optional). |
| **cmap:** | Colormap for mapping numeric values to colors (colormap, optional). |
| **norm:** | Normalize object for mapping data values to colors (Normalize, optional). |
| **vmin, vmax:** | Values to anchor the colormap, otherwise, they are inferred from the data |
| **alpha:** | Marker transparency (float, optional).  Range is between 0 to 1. |
| **linewidths:** | Width of marker edges (float or array-like, optional). |
| **label:** | Label for the plot legend (str, optional). |

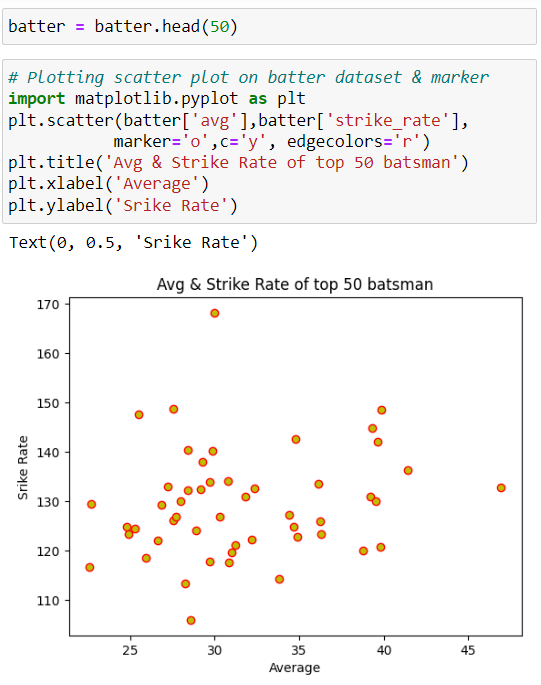


****

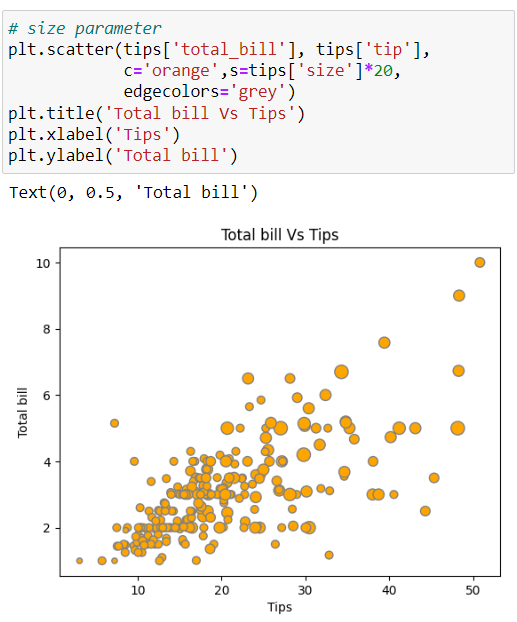


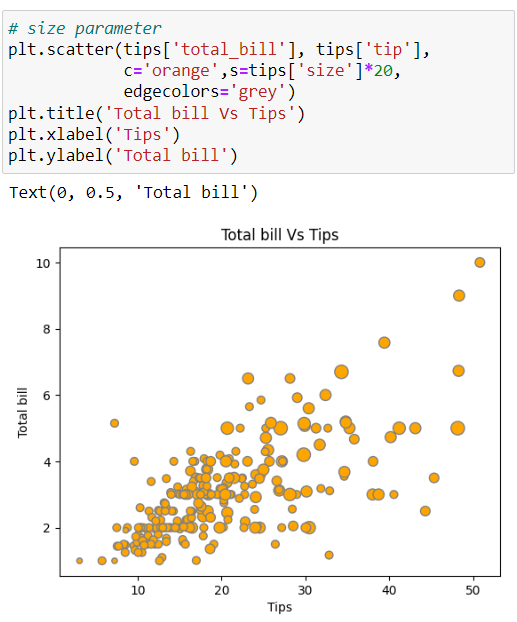
****

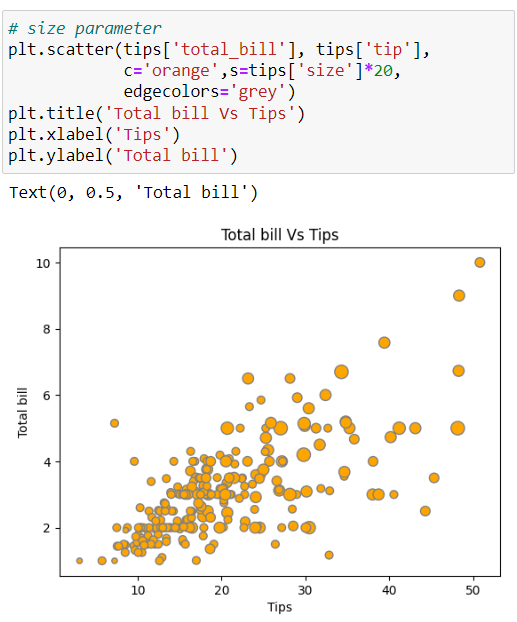
****

****

**SCATTERING USING PLT.PLOT()**

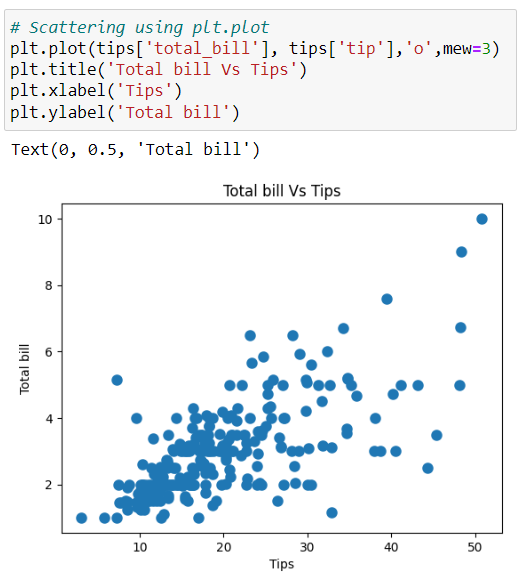
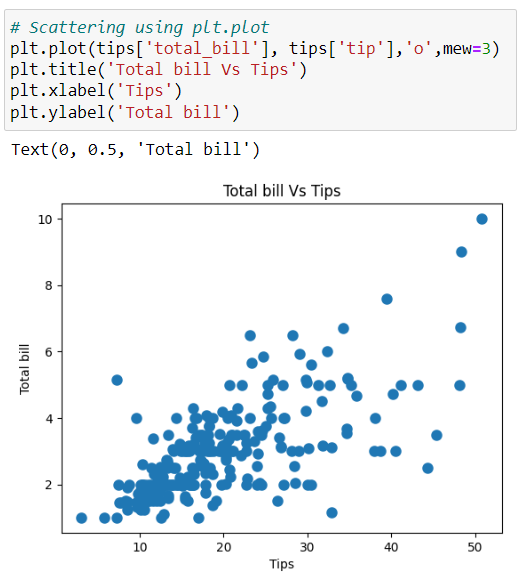
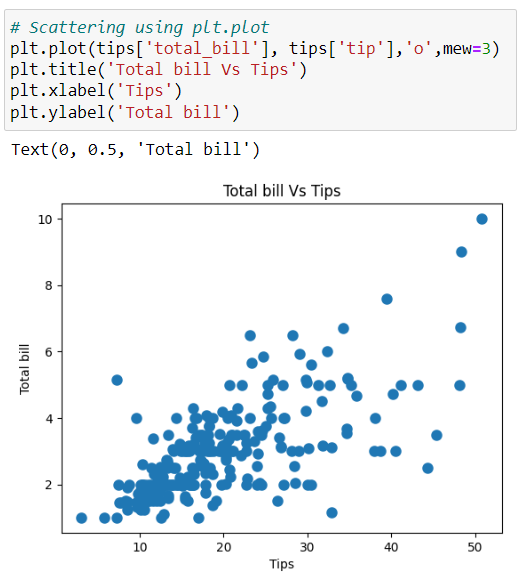
****

****

****

**PLT.SCATTER VS PLT.PLOT**

* Actually, the **plt.plot()** function is generally faster for scatterplots where markers don’t vary in size or color.
* The **plt.scatter()** function does not connect points and has additional capabilities for varying the symbol, size, and color.

****

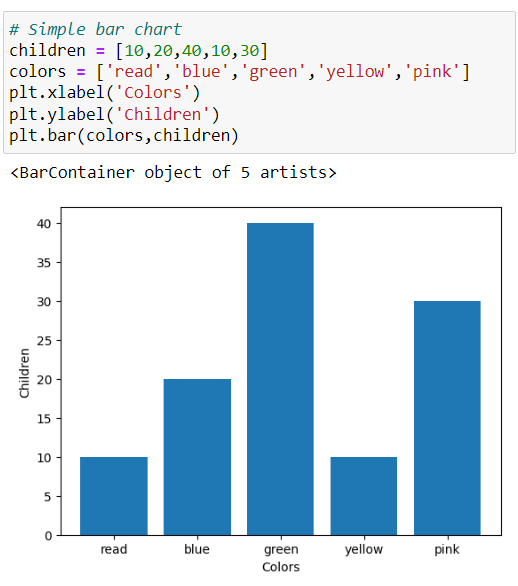
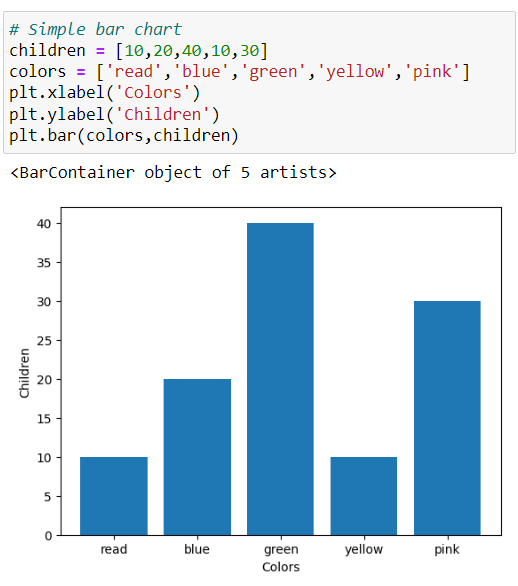
**BAR CHART**

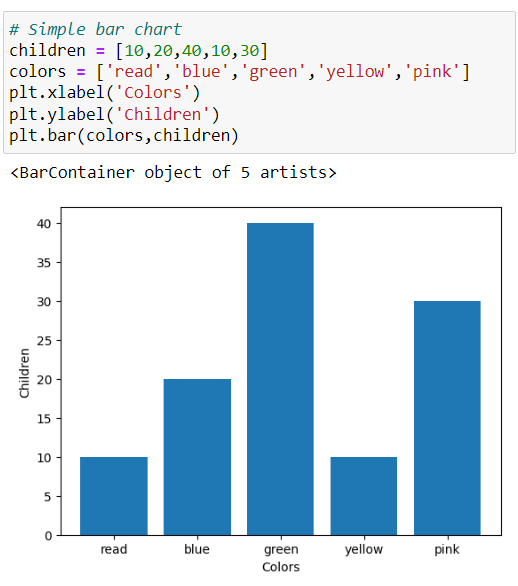
* Bar charts are quite similar to line charts, i.e., they show a sequence of values.
* However, a bar is shown for each value, rather than points connected by lines.
* We can use the **plt.bar()** function to draw a bar chart.
* **Syntax: plt.bar(x, height, width, bottom, align, color, edgecolor, linewidth, tick\_label, label, alpha)**

**Parameters:**

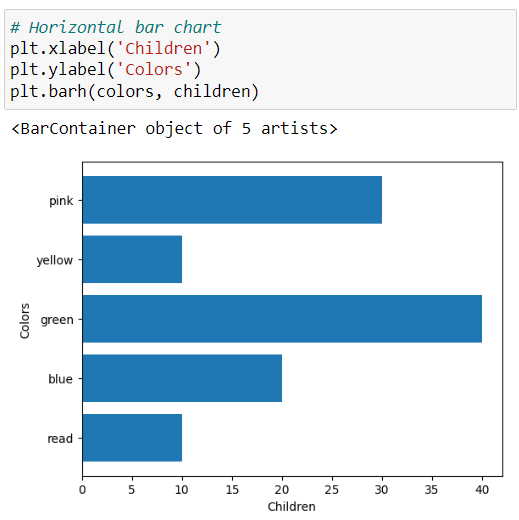
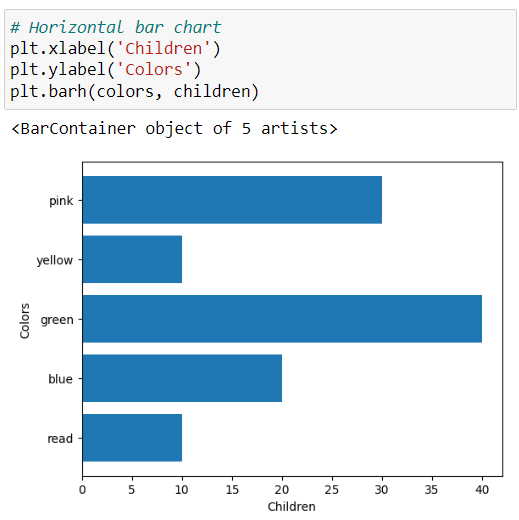
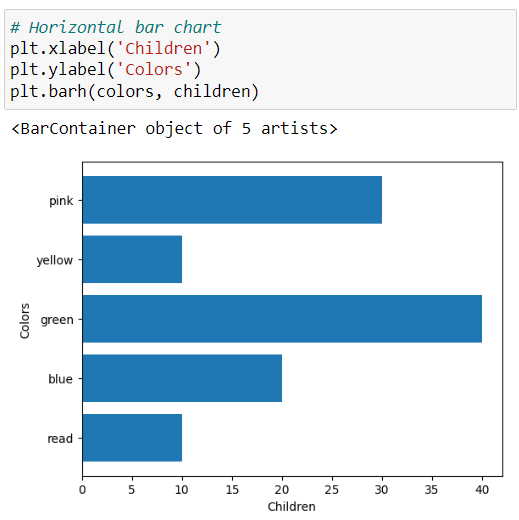
|  |  |
| --- | --- |
| **x**: | x-coordinates of the bars |
| **height**: | height of the bars |
| **width**: | width of the bars |
| **bottom**: | y-coordinate(s) of the bars' baseline |
| **align**: | alignment of the bars with the x-coordinates |
| **color**: | color of the bars. |
| **edgecolor**: | color of the bar edges |
| **linewidth (or lw)**: | width of the bar edges |
| **tick\_label**: | labels for the bars on the x-axis |
| **label**: | label for the bars, used for creating a legend |
| **alpha**: | transparency of the bars |

**SIMPLE BAR CHART**

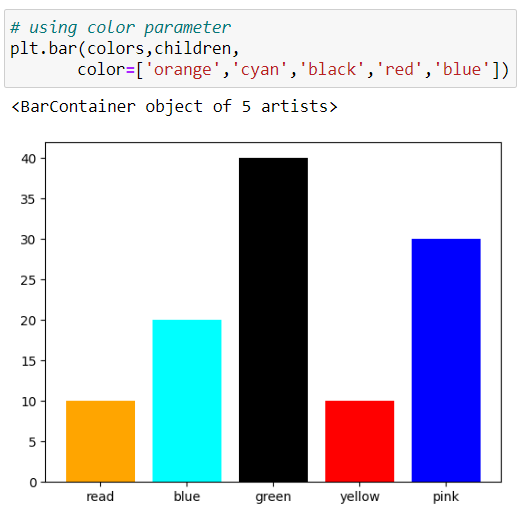
****

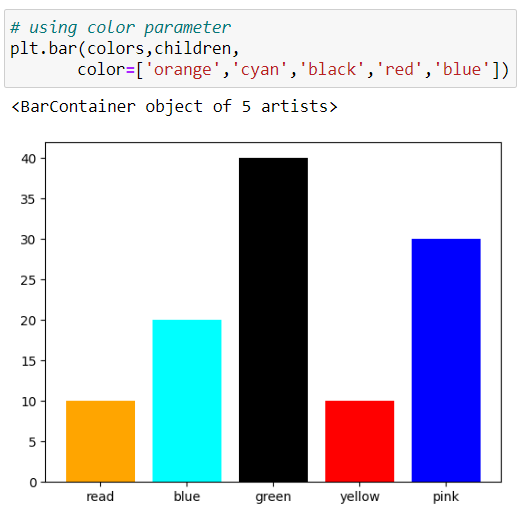
****

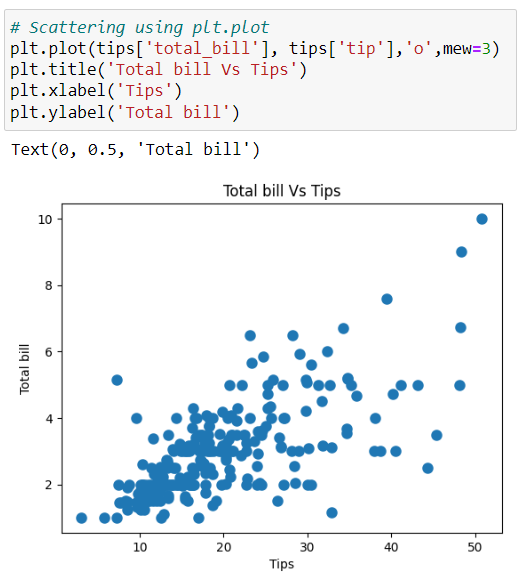
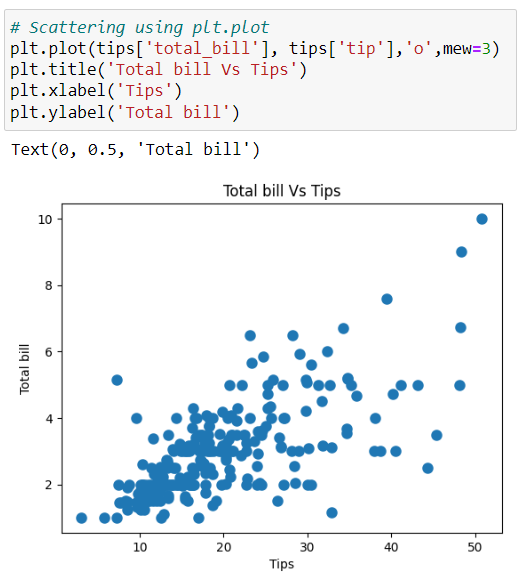
**HORIZONTAL BAR CHART**

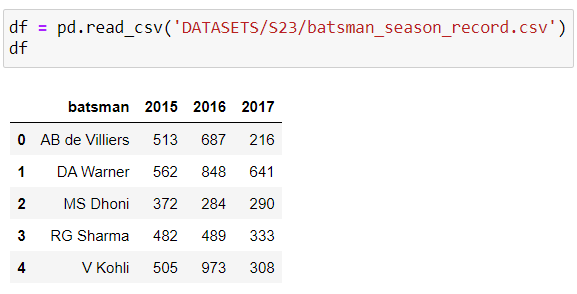
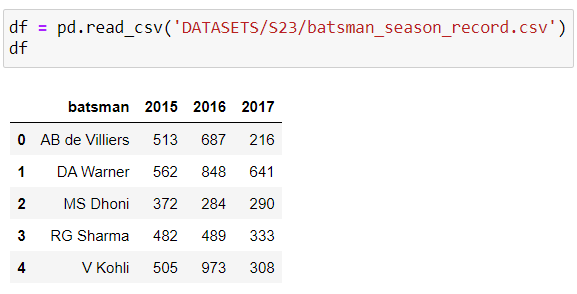
****

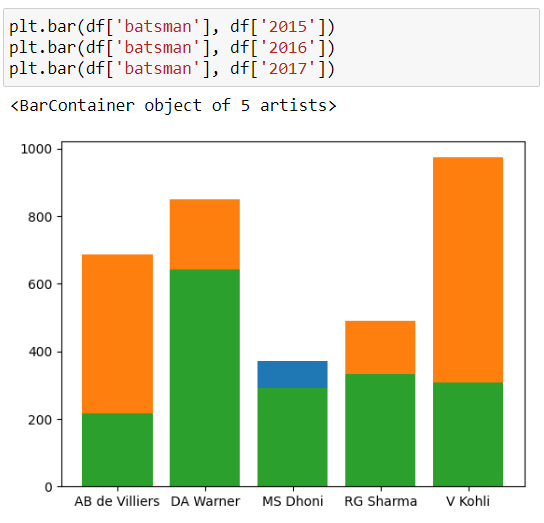
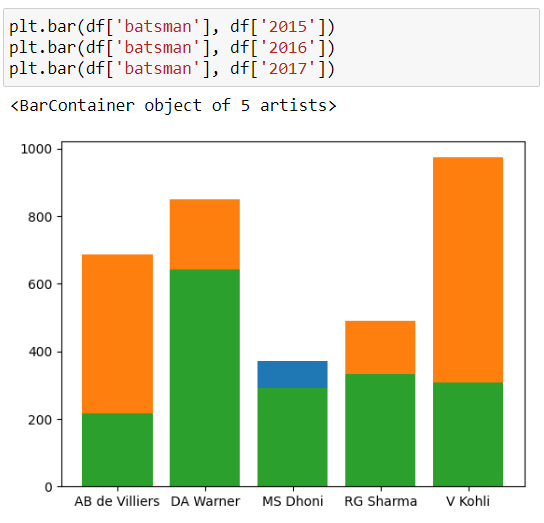
**CHANGING BAR COLOR**

****

****

****

****

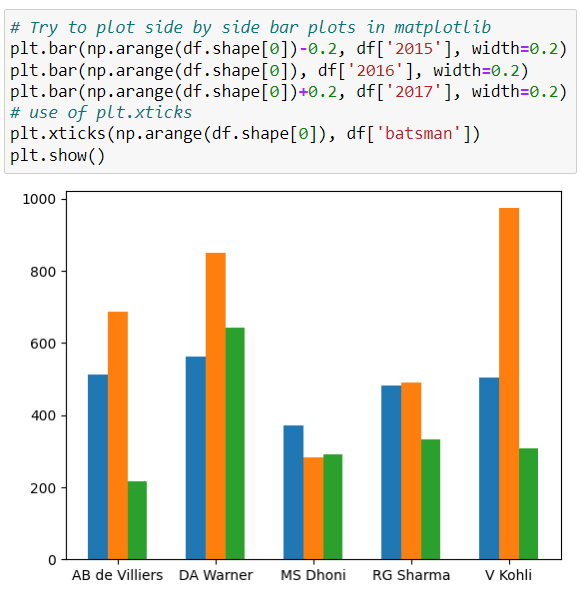
****

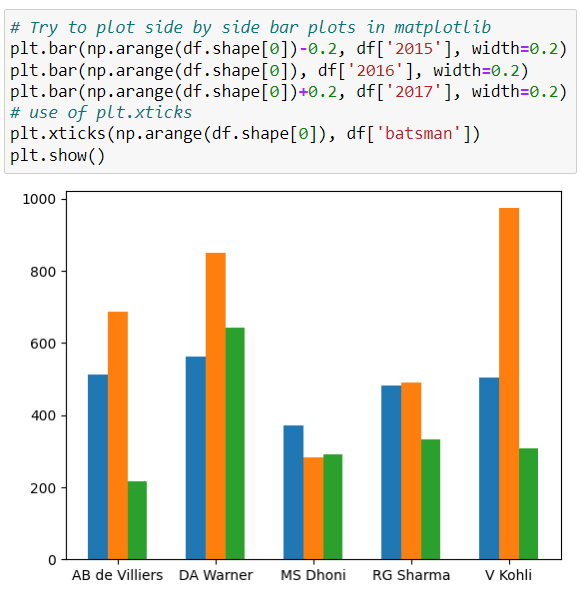
**PLT.XTICKS() FUNCTION:**

* The **plt.xticks()** function in Matplotlib is used to customize the appearance of the tick locations and labels on the x-axis of a plot.
* It allows you to set the positions and labels of the ticks manually.
* **Syntax: plt.xticks(ticks, labels, rotation=angle, ha='alignment')**

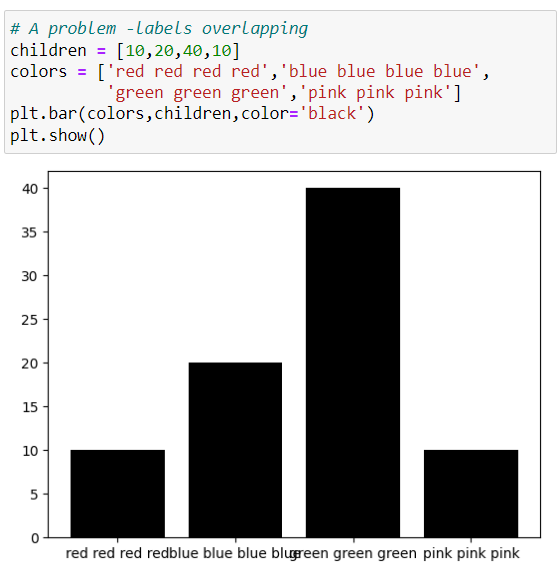
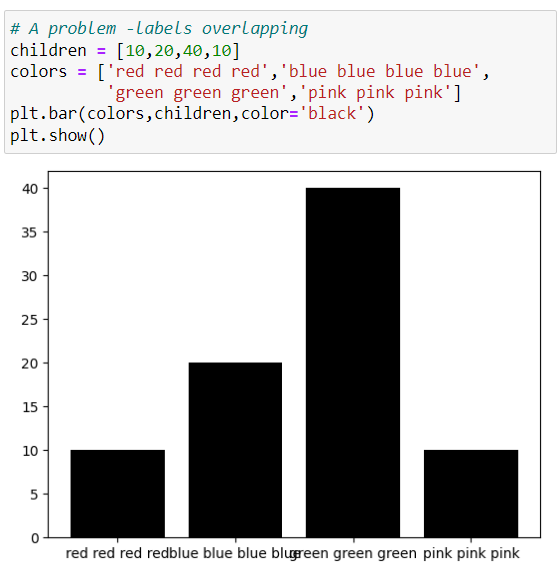
**Parameters**

|  |  |
| --- | --- |
| **ticks** | An array or list specifying the positions of the ticks on the x-axis. |
| **labels** | An array or list providing labels for the ticks.  If not specified, the tick positions will be labelled with their numeric values. |
| **rotation** | (Optional) The angle of rotation for the tick labels. |
| **ha** | (Optional) The horizontal alignment of the tick labels. It can be one of 'left', 'center', or 'right'. |

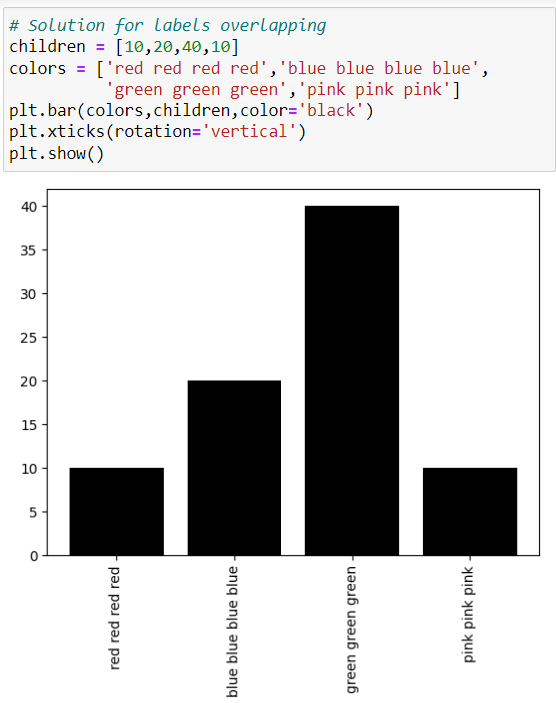
****

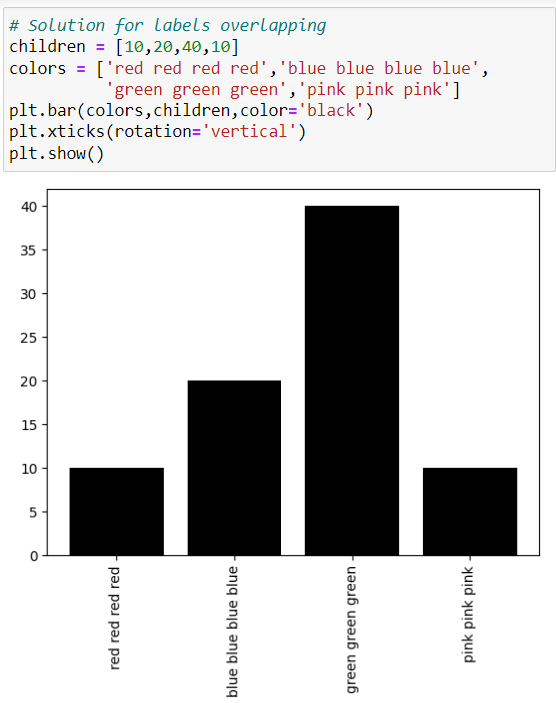
****

**LABELS OVERLAPPING PROBLEM**

****

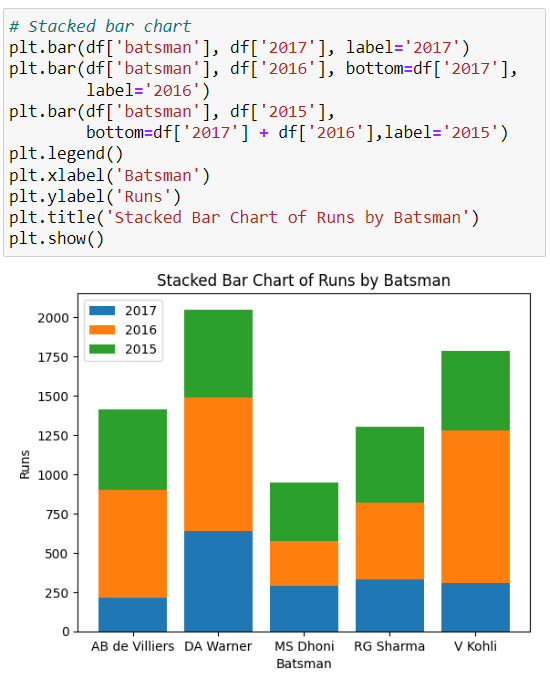
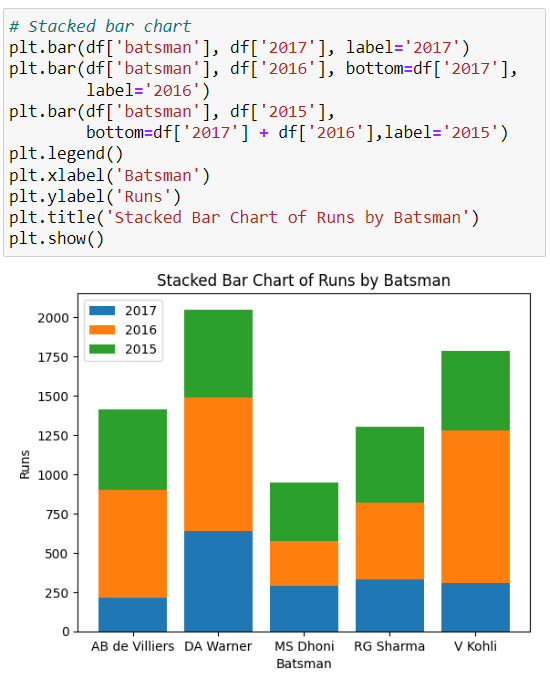
**SOLUTION FOR LABELS OVERLAPPING**

****

****

**STACKED BAR CHART**

The general idea for creating stacked bar charts in Matplotlib is that you'll plot one set of bars (the bottom), and then plot another set of bars on top, offset by the height of the previous bars, so the bottom of the second set starts at the top of the first set.



**HISTOGRAM**

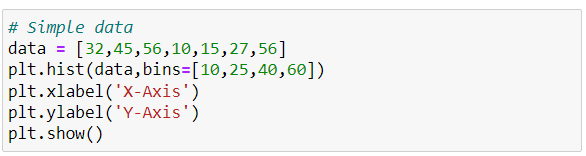
* A histogram visually displays how a variable is distributed by dividing its range into bins and using vertical bars to represent the frequency of observations within each bin.
* It is commonly used for univariate analysis of numerical columns to perform a frequency count of values within specified intervals.
* **Syntax: plt.hist(x, bins, range, density=False, cumulative=False, histtype=’bar’, align=’mid’, log=False, color, label)**

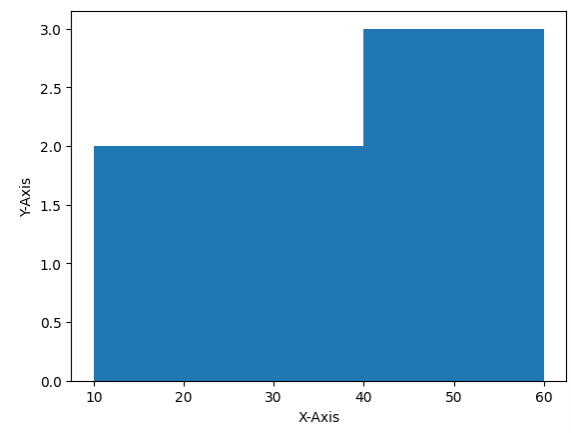
**Parameters:**

|  |  |
| --- | --- |
| **x** | Sequence of data. |
| **bins** | (Optional) Integer or sequence or string. |
| **range** | (Optional) Lower and upper range of the bins. |
| **density** | (Optional) Boolean values. |
| **histtype** | (Optional) Type of histogram. {‘bar’, ‘barstacked’, ‘step’, ‘stepfilled’} |
| **align:** | (Optional) Controls how the histogram is plotted. {‘left’, ‘mid’, ‘right’} |
| **log** | (Optional) Set histogram axis to a log scale. |
| **color** | (Optional) Color spec or sequence of color specs, one per dataset. |
| **label** | (Optional) String or sequence of strings to match multiple datasets. |

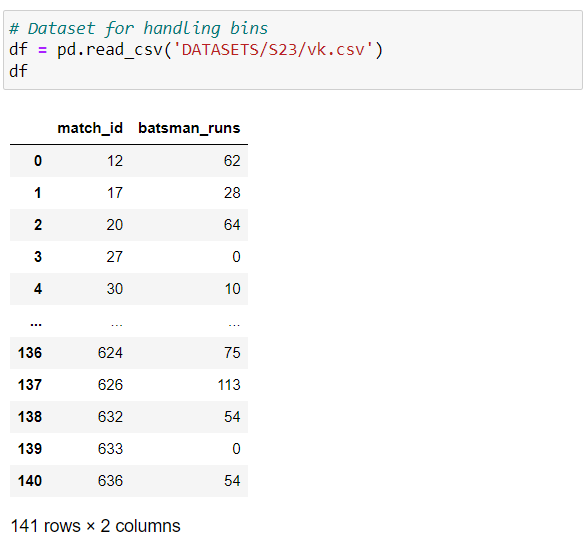
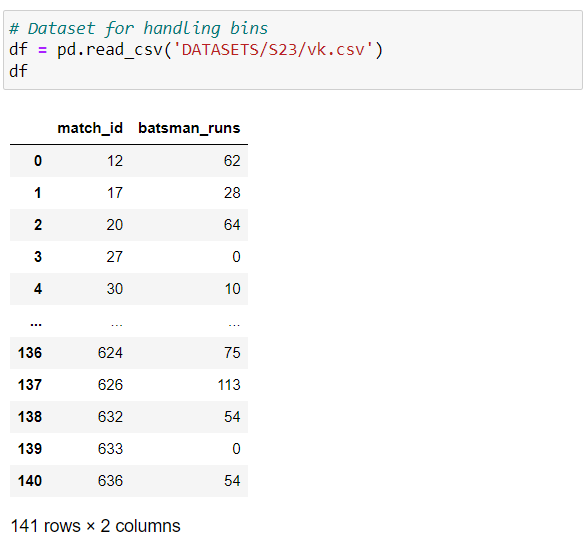
**NOTE:**

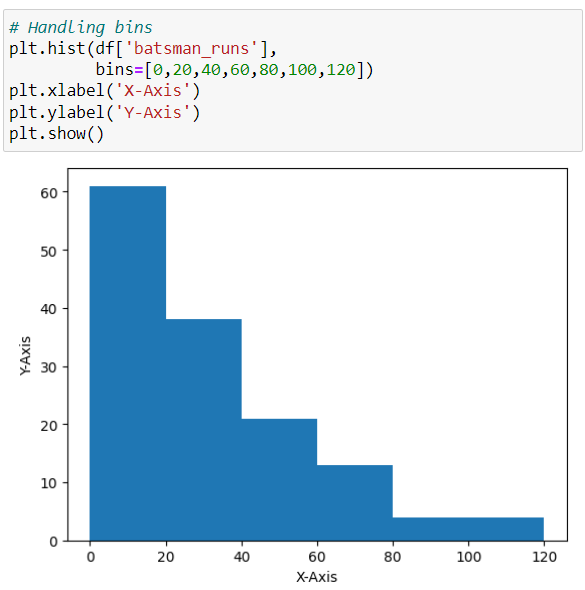
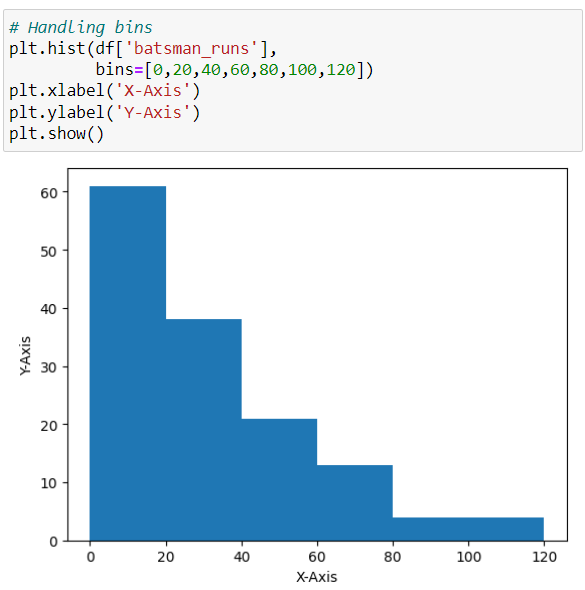
Here we take common parameters that are used in plt.hist() function.

****

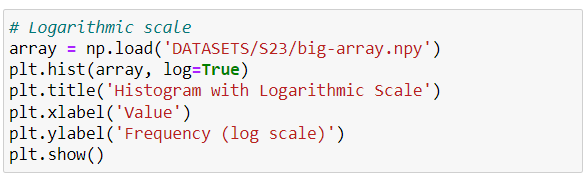
****

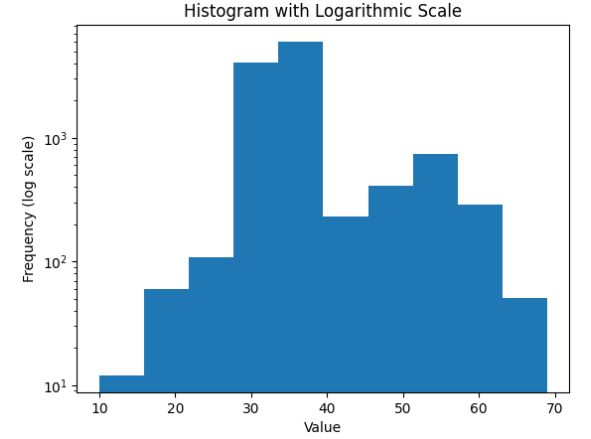
**HANDLING BINS**

****

****

**LOGARITHMIC SCALE**

****

****

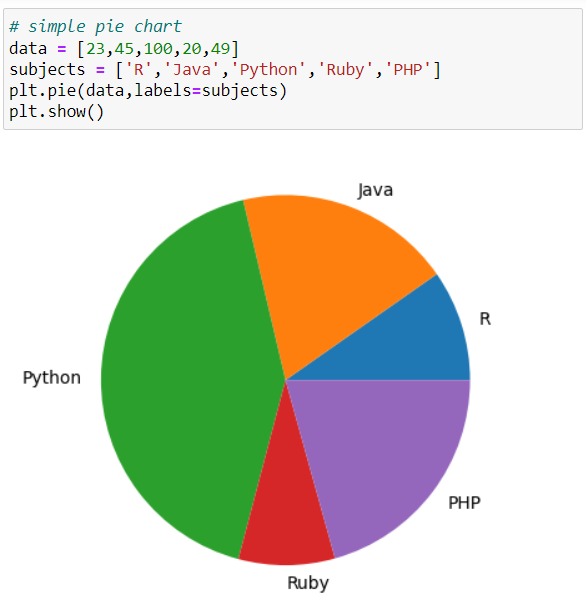
**PIE CHART**

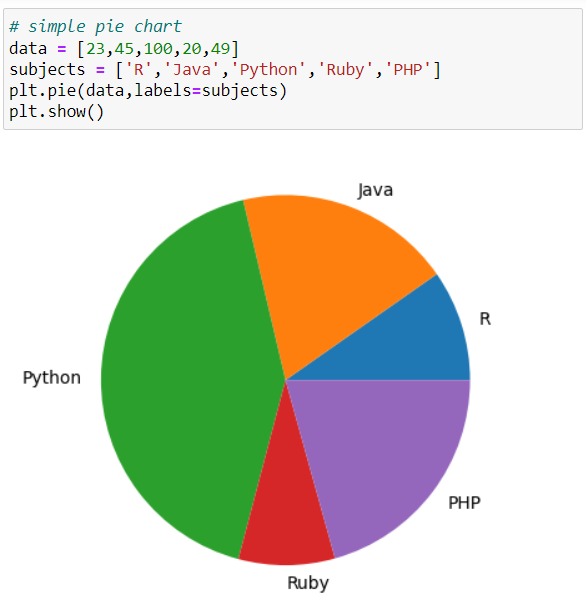
* A pie chart visually represents numerical proportions by dividing a circle into slices (the slices of pie are called wedges), each illustrating a proportionate part of the whole.
* Matplotlib's **plt.pie()** function facilitates the creation of pie charts for straightforward visualization.
* Univariate and bivariate analyses explore relationships within one or between two variables.
* Categorical vs numerical analysis involves examining relationships between categorical and numerical variables in a dataset.
* In use cases, such as finding contributions on a standard scale, pie charts provide a clear representation of proportional contributions.
* **Syntax: plt.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)**

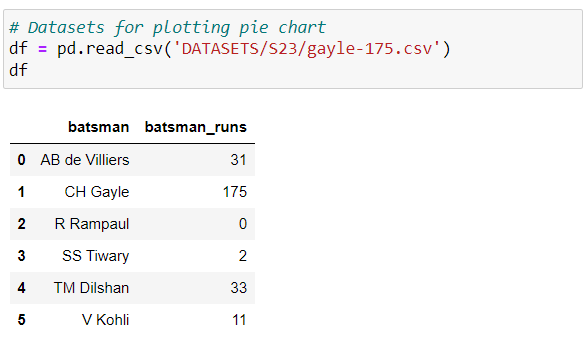
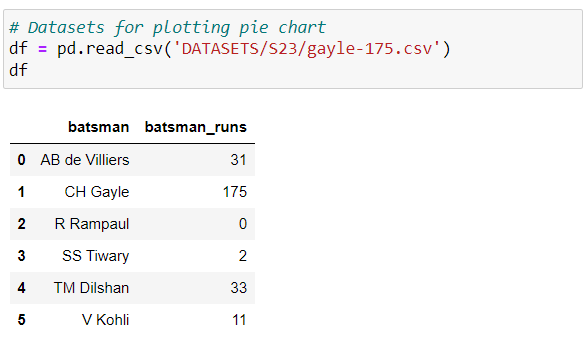
**Parameters:**

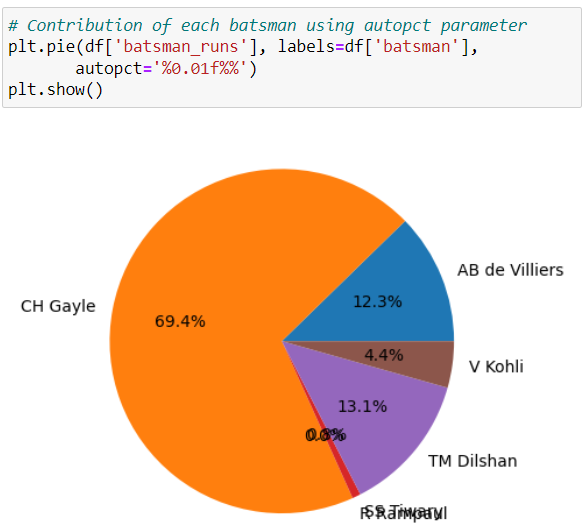
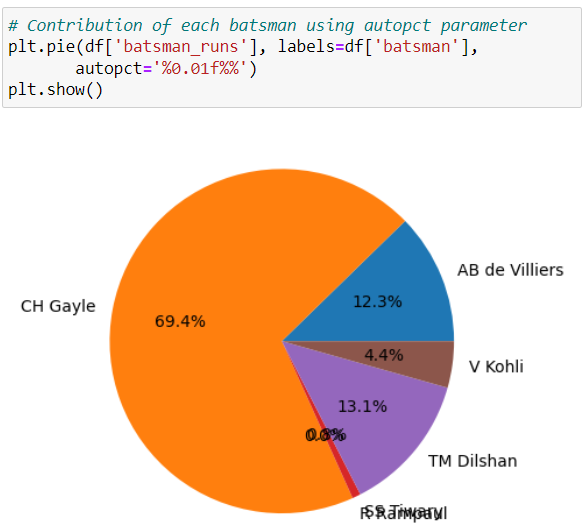
|  |  |
| --- | --- |
| **data** | Represents the array of data values to be plotted.  The fractional area of each slice is represented by data/sum(data).  If sum(data) < 1, then the data values return the fractional area directly, resulting in a pie chart with an empty wedge of size 1 - sum(data). |
| **explode**  (Optional) | Fraction of the radius with which to offset each wedge.  Useful for emphasizing specific slices. |
| **labels**  (Optional) | List or sequence of strings setting the label of each wedge. |
| **colors**  (Optional) | Sequence of valid color specifications to provide colors to the wedges. |
| **autopct**  (Optional) | String used to label the wedges with their numerical values.  Can include format specifications for displaying percentages or other formatting. |
| **shadow**  (Optional) | Boolean parameter.  When set to True, creates a shadow of the pie chart. |

**SIMPLE PIE CHART**

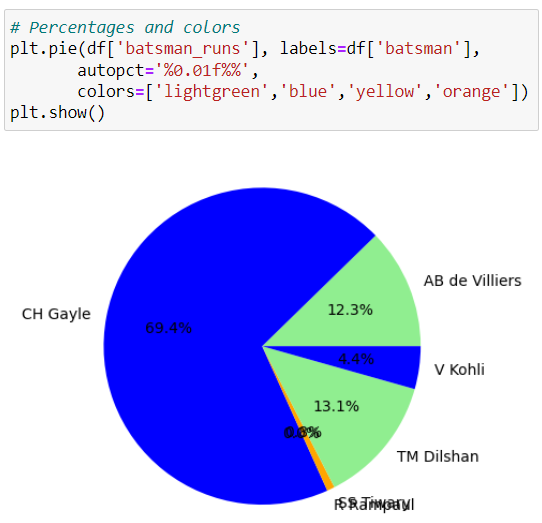
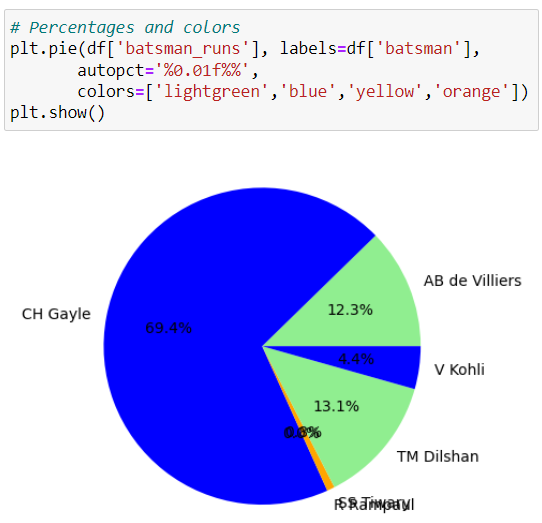
****

****

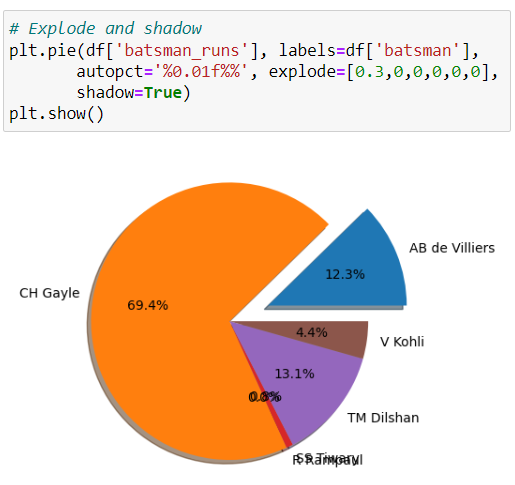
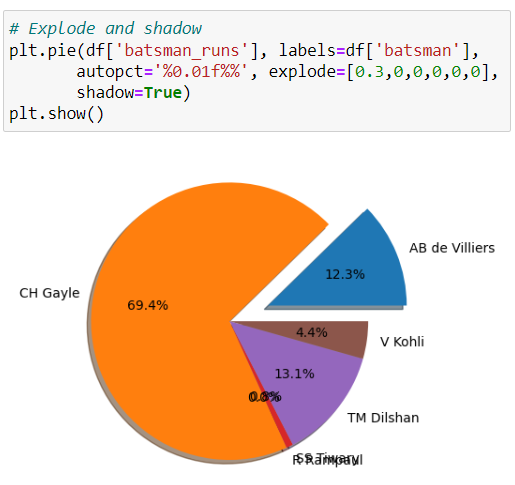
****

****

**PERCENTAGES AND COLORS**

****

**EXPLODE AND SHADOW PARAMETERS**

****

**CHANGING STYLES**

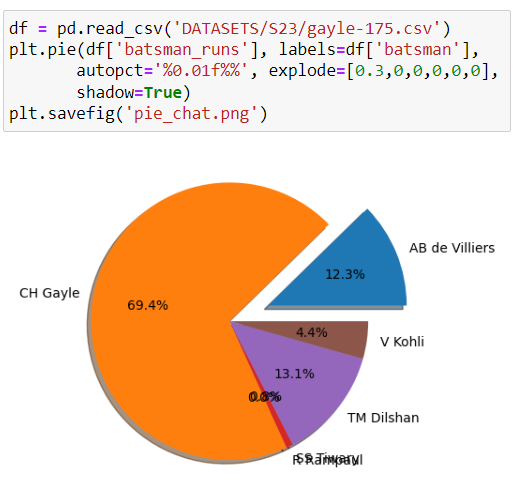
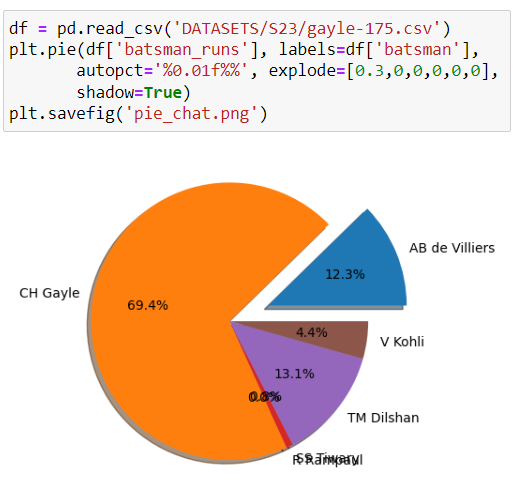
**Restoring defualt matplotlib style**

**Save Figures**

In Matplotlib, you can save figures using the savefig() function and allows you to save the current figure to a file.

**Key Points:**

* savefig() is used to save the current figure to a specified file path.
* The file format is determined by the file extension (e.g., 'png', 'jpg', 'pdf').
* You can provide an absolute or relative path to specify the location where the figure should be saved.
* Remember to call **savefig()** before calling **show()** if you intend to save the figure without displaying it interactively.

****