Line Plots and Plot Customizations

What is a line plot?

Line plots are simple charts used to display a series of data points connected by straight line segments. The X-axis lists the categories equally and the y-axis represents the corresponding category values. This kind of graph serves to visualize a trend summarized from data periodically and hence has wide applications in time series data. Some popular use cases of line graphs include the price trend of stocks, weather changes during a year etc.

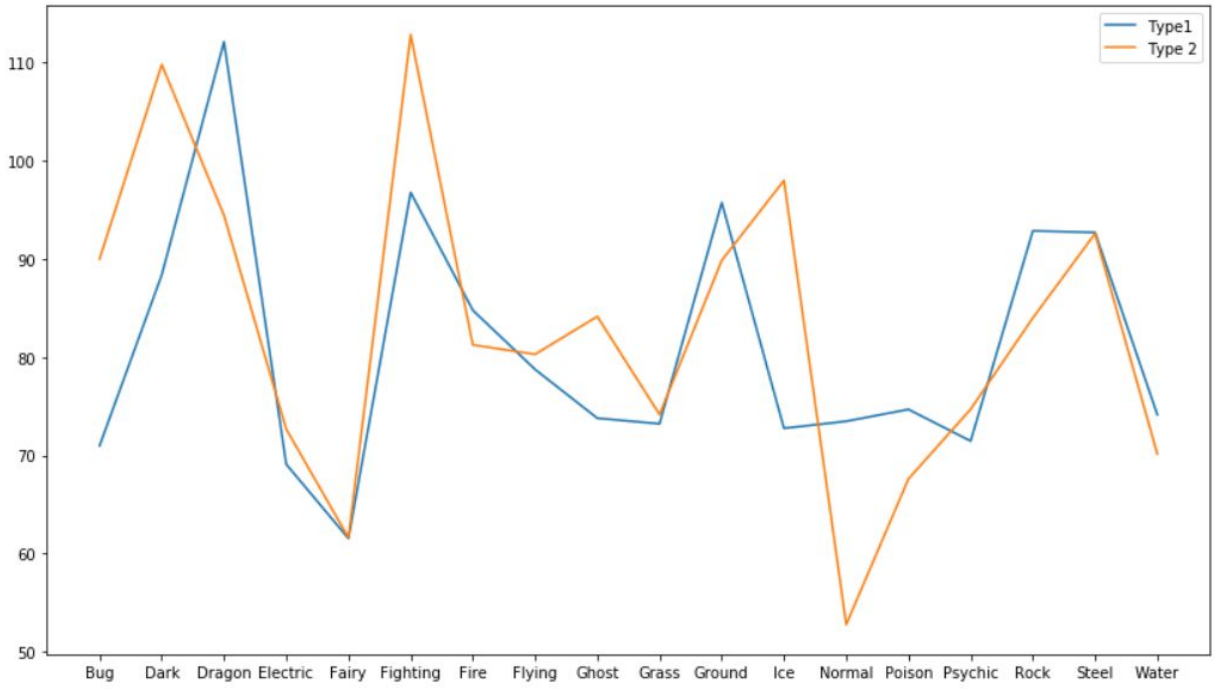
Example

Let's answer this question: How does the mean attack (`Attack`) points `Type 1` Pokemons fare against `Type 2` Pokemons?

The steps to answer this question are:

- First group Pokemons together by their `Type 1` and `Type 2` and calculate the mean `Attack` points for every variant
- Plot them out as line plots in the same plot with types on the X-axis and mean attack points on the Y-axis

You will get a graph that looks like this:



The plot above helps in giving a clear comparison of the mean attack points for different variants of `Type 1` (depicted by blue line) and `Type 2` (depicted by orange line)

Code for implementing line plot

The code for implementing the line plot is quite simple:

```
plt.plot(x, y)
```

where `x` and `y` are the two arrays we want to plot on X-axis and Y-axis respectively.

Advantages and disadvantages

- The line chart is easy to understand and gives an instant perception of trends.
- Sometimes it might mess the entire chart if many categories are compared in one line chart.

Why is it a good practice to customize your plot?

We will answer this question by asking a set of questions. Suppose you are looking at the above image for the very first time. Can you answer these questions?

- What do the X-axes and Y-axes represent?

- Overall what does the line chart represent?

Obviously not. That's where plot customizations come to the rescue. You can customize your plot to make it unique and pleasing to the eye and depicting the important details; all at the same time. Across different plots, these elements (customizations) more or less remain the same. *Many types of customizations can be done to customize a plot; adjusting the colors, changing markers, lifestyles and linewidths, adding text, legend and annotations, and changing the limits and layout of your plots.*

Widely used plot customizations

Below are some of the widely used customization operations that you would be performing while using matplotlib:

1) Adjusting size of the figure: You can change of the figure using `plt.figure(figsize=(x,y))` where you can set `x` and `y` values to satisfy your requirements

2) Axes labels and title: Use `plt.title('Title')` on the axes to set the title of the plot and `plt.xlabel('xlabel'),plt.ylabel('ylabel')` to set the labels

3) Axes limits: Use `plt.xlim((a,b))` and `plt.ylim((a,b))` to fix the boundaries in the range (a,b) within which you want to display the plot

4) Changing color: Use the `color` argument inside the different types of plot functions to change its color.

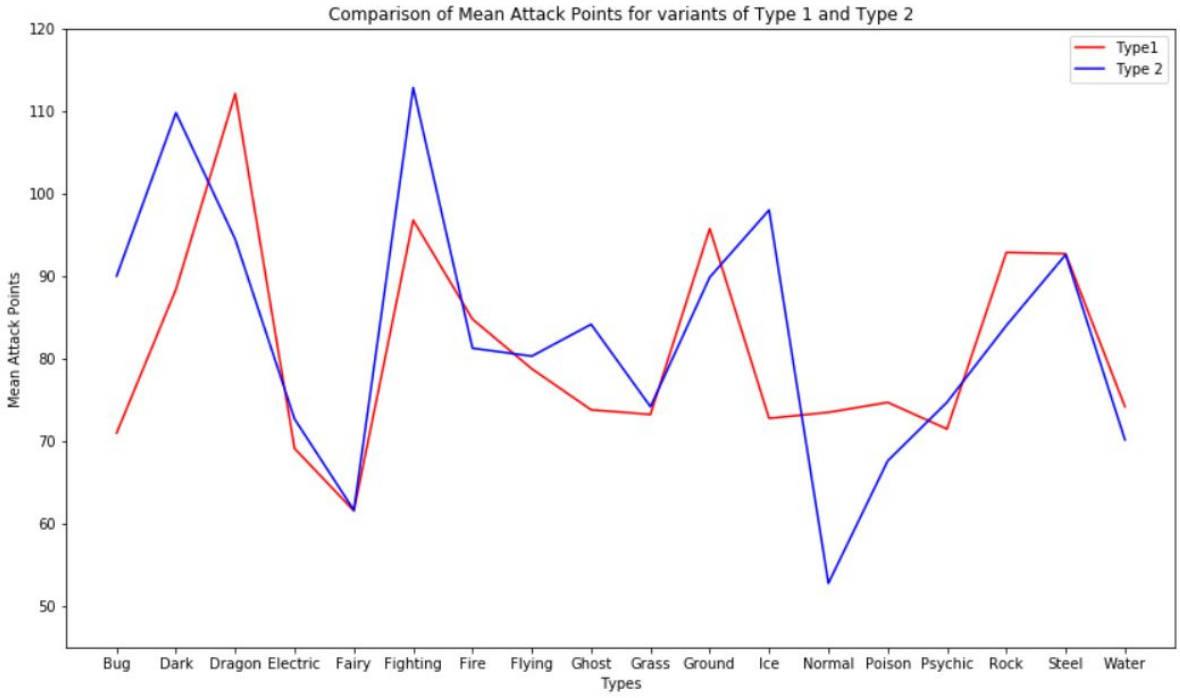
5) Legends: In case we have multiple types of charts in a single plot, you can differentiate them with legends. Use `plt.legend()` to display legends with the help of the labels keyword argument inside it. To

6) Save figure: You can easily save a figure to, for example, a png file by making use of `plt.savefig()`. The only argument you need to pass to this function is the file name, just like in this example:

```
# Save Figure
plt.savefig("foo.png")

# Save Transparent Figure
plt.savefig("foo.png", transparent=True)
```

Output


Comparison of Mean Attack Points for variants of Type 1 and Type 2
Type1
Type 2
Mean Attack Points
120
110
100
90
80
70
60
50
Bug
Dark
Dragon
Electric
Fairy
Fighting
Fire
Flying
Ghost
Grass
Ground
Ice
Normal
Poison
Psychic
Rock
Steel
Water
Types