Data Visualization Using MATPLOTLIB

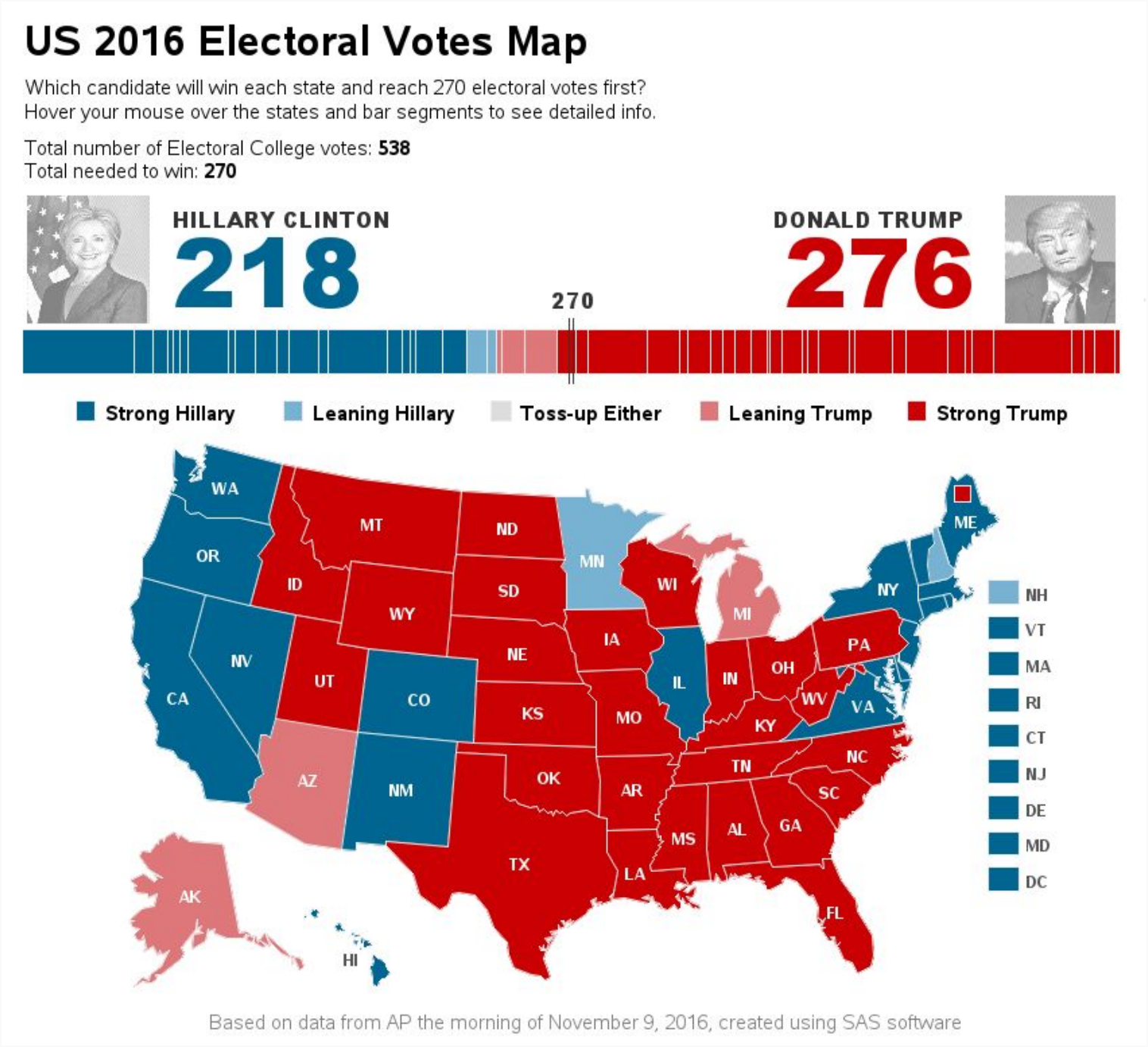
6.1.1 Why Data Visualization?

"The data is a mess" - This was the statement reiterated by every data scientist when asked what was the most difficult part of the job according to a report by Harvard Business Review. It was found that the problem was not with the quality of the data (noisy/clean) but that the size of data is too overwhelming to derive any insights.

Let's look at a snapshot of the US 2016 Election data by electoral votes polled per state to each candidate.
Source: Offical 2016 Election Results

STATE	ELECTORAL VOTES	ELECTORAL VOTES CAST FOR DONALD J. TRUMP (R)	ELECTORAL VOTES CAST FOR HILLARY CLINTON (D)
AL	9	9	
AK	3	3	
AZ	11	11	
AR	6	6	
CA	55		55
CO	9		9
CT	7		7
DE	3		3
DC	3		3
FL	29	29	
GA	16	16	
HI	4		3 *(1 was cast for Bernie Sanders)*
ID	4	4	
IL	20		20
IN	11	11	
IA	6	6	
KS	6	6	
KY	8	8	
LA	8	8	
ME	4	1	3
MD	10		10
MA	11		11
MI	16	16	
MN	10		10
MS	6	6	
MO	10	10	
MT	3	3	
NE	5	5	
NV	6		6
NH	4		4
NJ	14		14
NM	5		5
NY	29		29
NC	15	15	
ND	3	3	
OH	18	18	
OK	7	7	
OR	7		7
PA	20	20	
RI	4		4
SC	9	9	
SD	3	3	
TN	11	11	
TX	38	36 *(1 was cast for John Kasich; 1 was cast for Ron Paul*	
UT	6	6	
VT	3		3
VA	13		13
WA	12		8 *(3 were cast for Colin Powell; 1 was cast for Faith Spotted Eagle)*
WV	5	5	
WI	10	10	
WY	3	3	

Deriving meaningful insights from this data as to how many states Trump has won in comparison to Clinton is not seen clearly. Now, look at this image w.r.t the same data.



The image above conveyed a lot about the 2016 US Presidential election in one single frame.

This is the power of Data visualization. It is an effective way to observe trends, anomalies as well as patterns in the data. More importantly, it is also one of the simplest and the best communicator for your results. Thus data visualisation is an important tool in the arsenal of a data scientist.

Visualization comes very easy in Python, as it offers libraries like `matplotlib`, `seaborn`, `bokeh`, `plotly` etc. In this concept, we will be discussing how to use `matplotlib` to visualize data effectively.

Bar-chart and Anatomy of a Plot

In the previous concept, you have done extensive work on the Pokemon dataset. Let's continue working on the same dataset. Imagine a layman Pokemon enthusiast has come to you with a set of questions about Pokemons. How would you answer them?

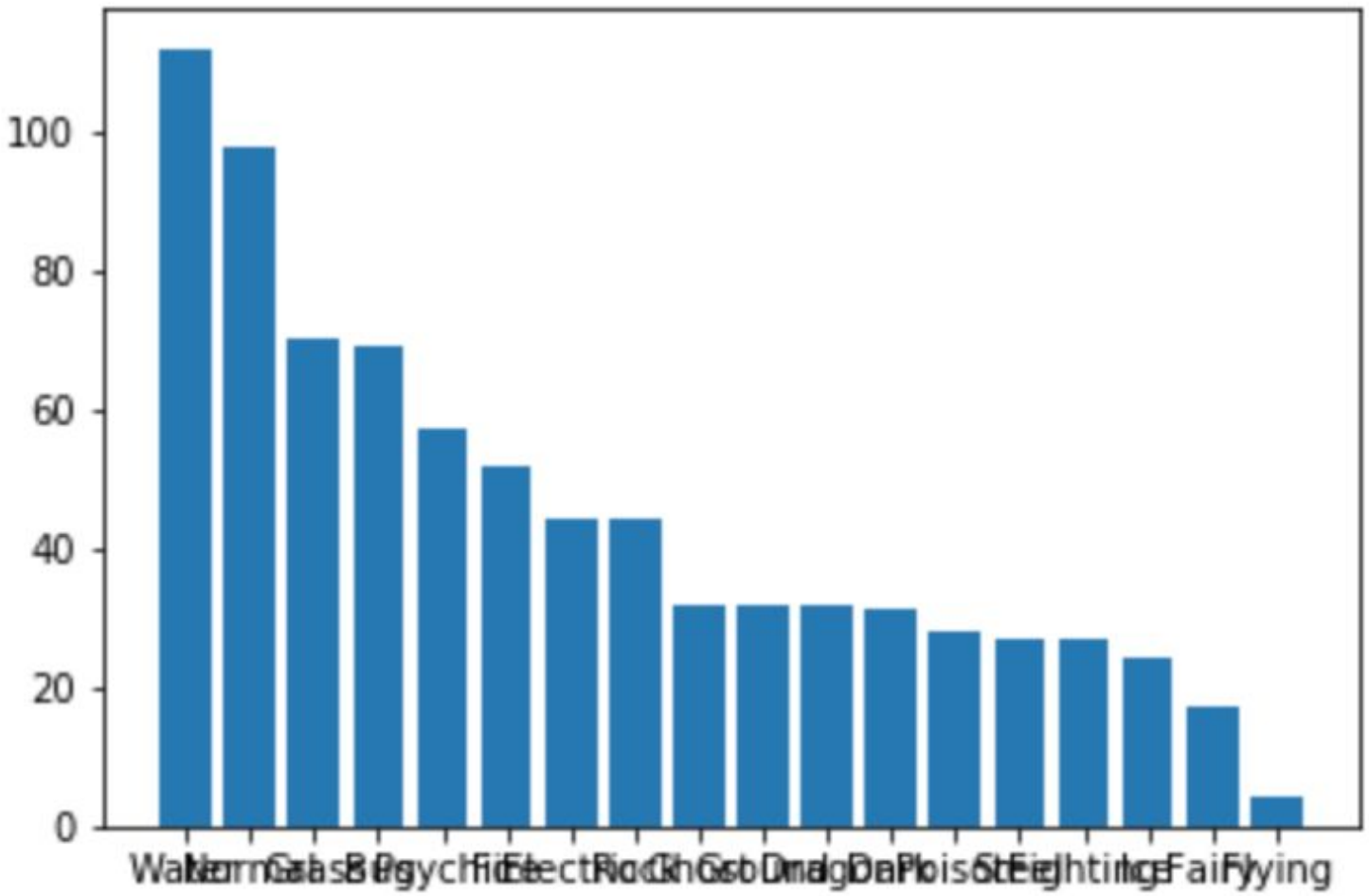
Let's start with one of the questions asked in the previous concept - `How many different variants of Type 1 pokemon are there along with the counts?`. First, you have to perform data wrangling on pandas to come to the right data.

```
import pandas as pd

# load data

df = pd.read_csv('../data/pokemon.csv')

# reset index to 'Name'

df.set_index('Name', inplace=True)

# drop column '#'

df.drop('#', inplace=True, axis=1)

type_1_data = df['Type 1'].value_counts()
```

We have now come to the right form of data that contains the answer. But is there an interesting way to onboard the answer to our friend? Well, a picture is worth a thousand words. And thankfully with Python's `matplotlib` library it fortunately, takes far less than a thousand words of code to create a production-quality graphic.

Let's look at a simple plot of the data constructed with default values. We will be using the `bar` plot to show the result. A bar chart or bar graph is a chart or graph that presents grouped data with rectangular bars with

lengths proportional to the values that they represent. Bar plots can be both vertical and horizontal. To draw bar charts, we require two arrays; one representing the categories and the other denoting the heights of individual categories.

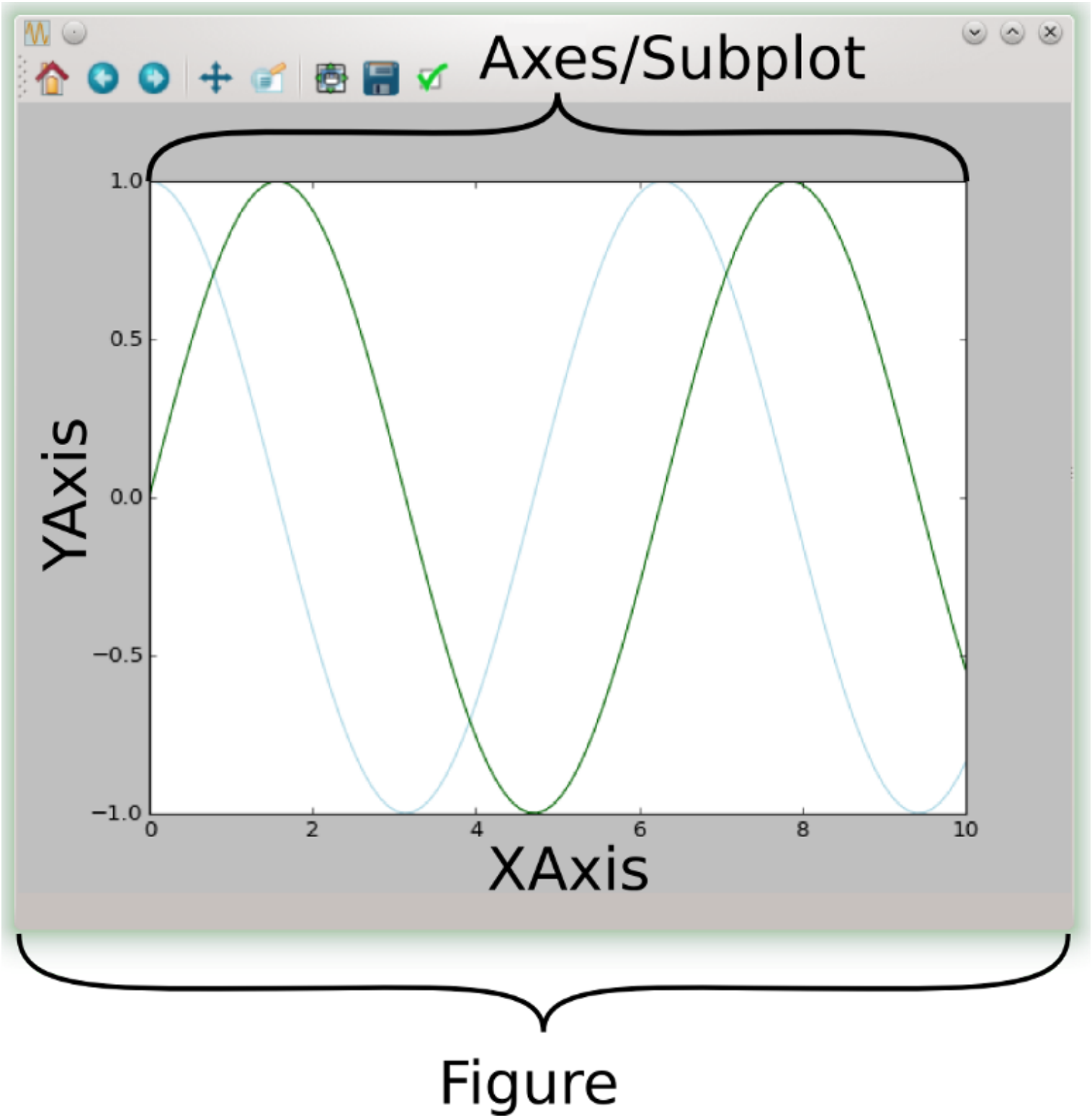


Look at the plots and can you see what is wrong? The plot has no information about what it is trying to show and no proper labels to clearly tell what they actually mean. The labels on the x-axis also are completely overlapping. We need to fix this. To do that first, we need to understand the basic anatomy of a plot made up of two important objects - `Figure` and `Axes`

- `Figure`: A Figure object is an outermost container for a `matplotlib` graphic. Within the Figure, everything else is contained. You can choose to create multiple independent figures.
- `Axes`: This actually refers to an individual plot and is added to a `Figure`; so, a `Figure` can contain multiple `Axes`. Usually, we'll set up an `Axes` with a call to `subplot` and so, both of them can be used interchangeably.

Further, each `Axes` has an `XAxis` and a `YAxis`. These contain the ticks, tick locations, labels etc. Almost every 'element' of a chart is its own manipulable Python object. But for our purpose, a basic understanding is

sufficient. You will get a better understanding of the difference between a `Figure` and `Axes` in the snapshot below:



Now we will try to fix the bar chart that we have created using matplotlib. The steps we would need to do are:-

- adjust the size of the image so that the xticks become visible.
- label the x-axis and y-axis with meaningful labels.

- give a title for the plot.

  We will apply these fixes to the plot and revisualize the plot and see if we can show it better. The labels for axes and the title must be concise and as self-explanatory as possible.

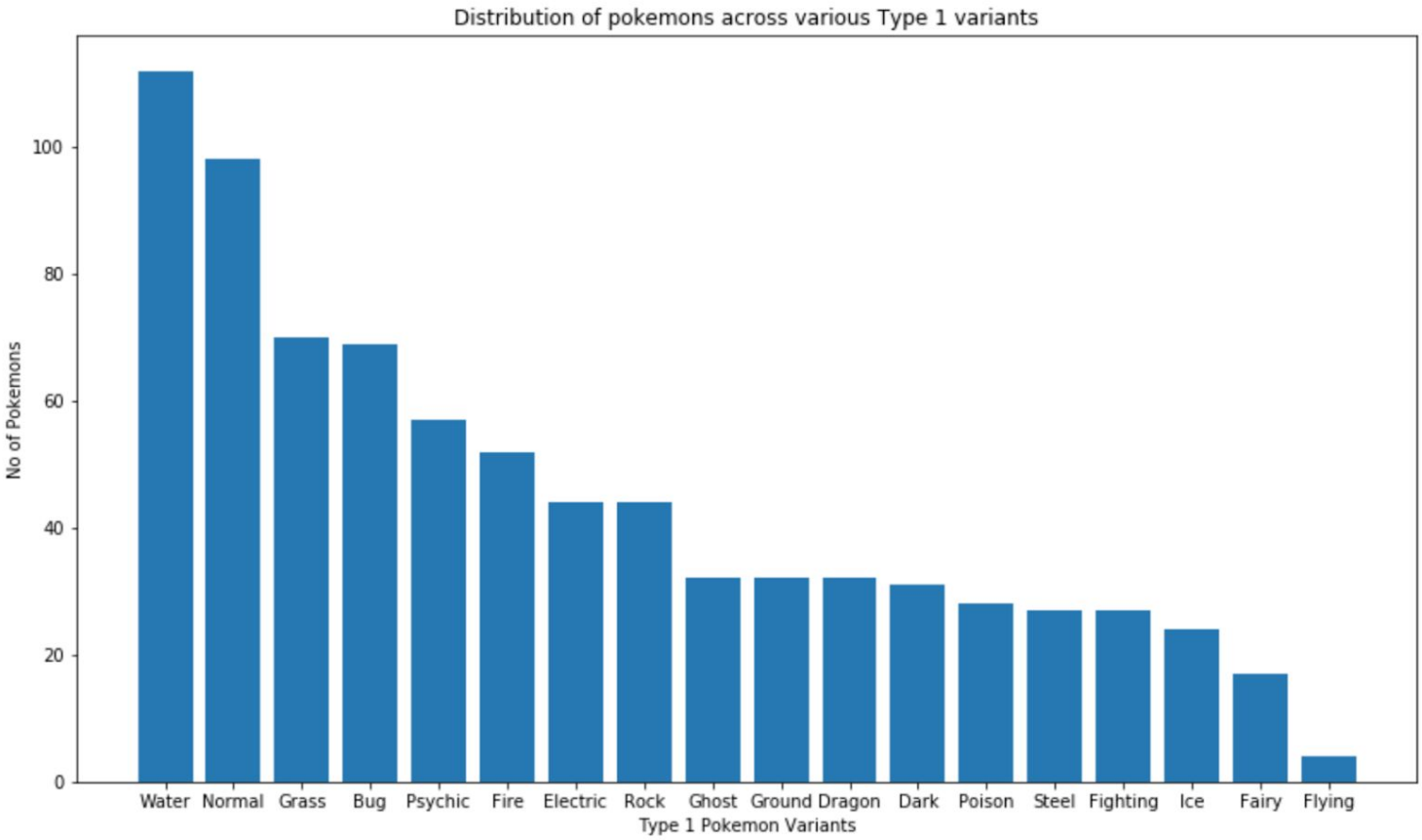
The steps generally associated with generating a plot through `matplotlib` are:

- First, importing `matplotlib.pyplot` as `plt` (customary)
- Initialize the figure with `plt.figure()`. It is the entire drawing canvas and you can adjust its size with the argument `figsize()`
- Then we label each of the axes through `plt.xlabel()` and `plt.ylabel`.
- Next, we title the plot using `plt.title()`
- Finally, build and show the plot with `plt.bar()`

Some of the steps are interchangeable like we can build the plot and label later, but it is a good practice to follow a neat sequence of steps.

```
import matplotlib.pyplot as plt

%matplotlib inline

# initialize the figure

plt.figure(figsize=[14,8])

# label the axes

plt.xlabel("Type 1 Pokemon Variants")

plt.ylabel("No of Pokemons")

# title the plot

plt.title("Distribution of pokemon across various Type 1 variants")

# build and show the plot

plt.bar(type_1.index,type_1_data)
```

Output

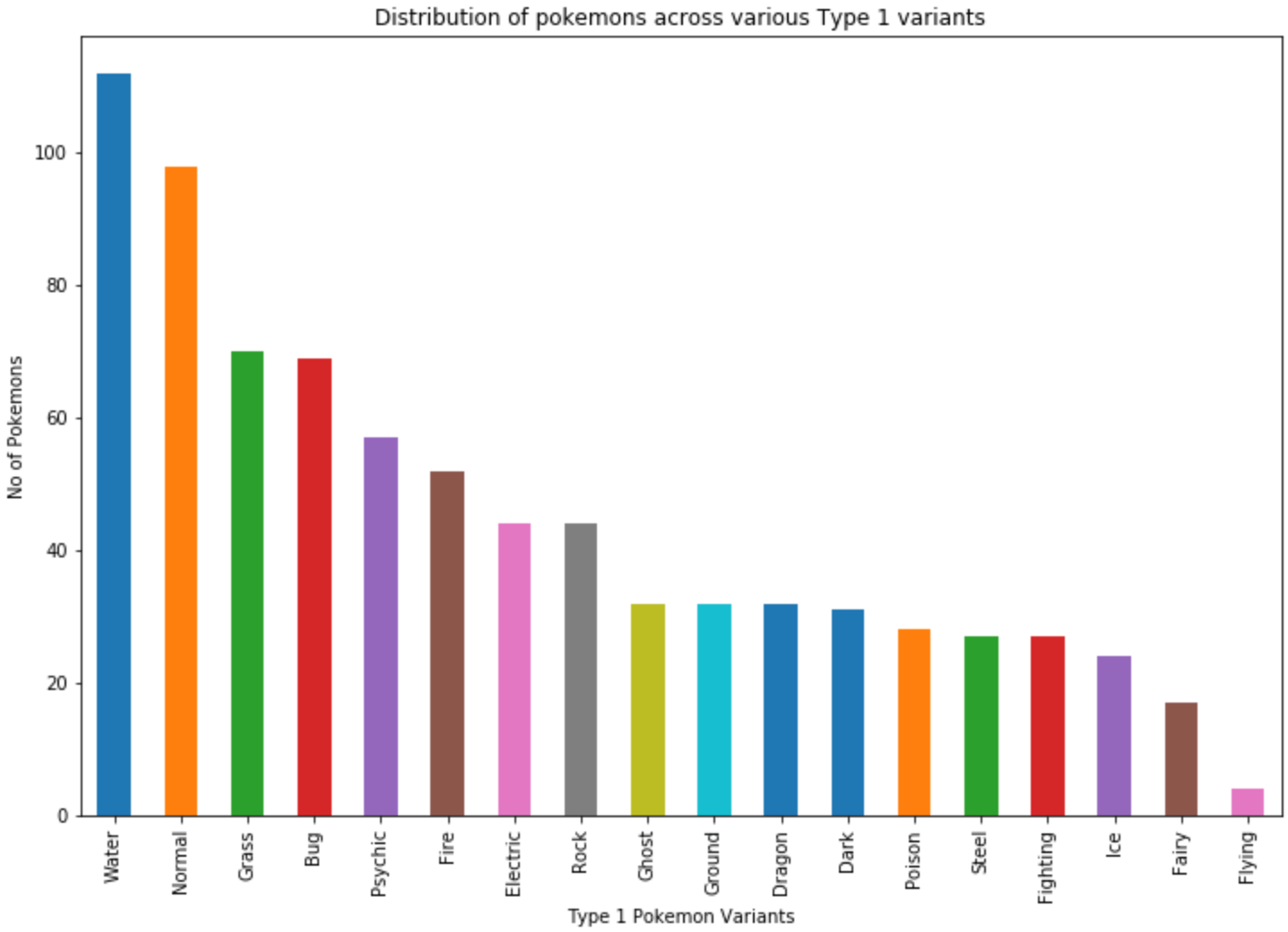


This looks much better and gives the answer directly to our pokemon enthusiast friend about the different `Type 1` variants and their respective counts. Of course, the plot can be made better with legends and colours, which we will see in the next chapter.

Additionally, from the pandas dataframe itself, you can call the plot function and plot the same data. The internal defaults are a little different, so the graph might look a little different. But, essentially it is the same visualization.

```
# keeping the same axis labels as earlier

df['Type 1'].value_counts().plot(kind="bar")
```

Output



Why use bar charts?

If you have comparative data that you would like to represent through a chart then a bar chart is the best option. A bar chart uses bars to show comparisons between categories of data. These bars can be displayed horizontally or vertically. A bar graph will always have two axes. One axis will generally have numerical values, and the other will describe the types of categories being compared.

Advantages and disadvantages of using bar charts

- The advantage is the bar chart is easy to read and understand. You get a good overview of values when using bar charts.
- The bar chart does not work so well with many dimension values due to the limitation of the axis length.

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.

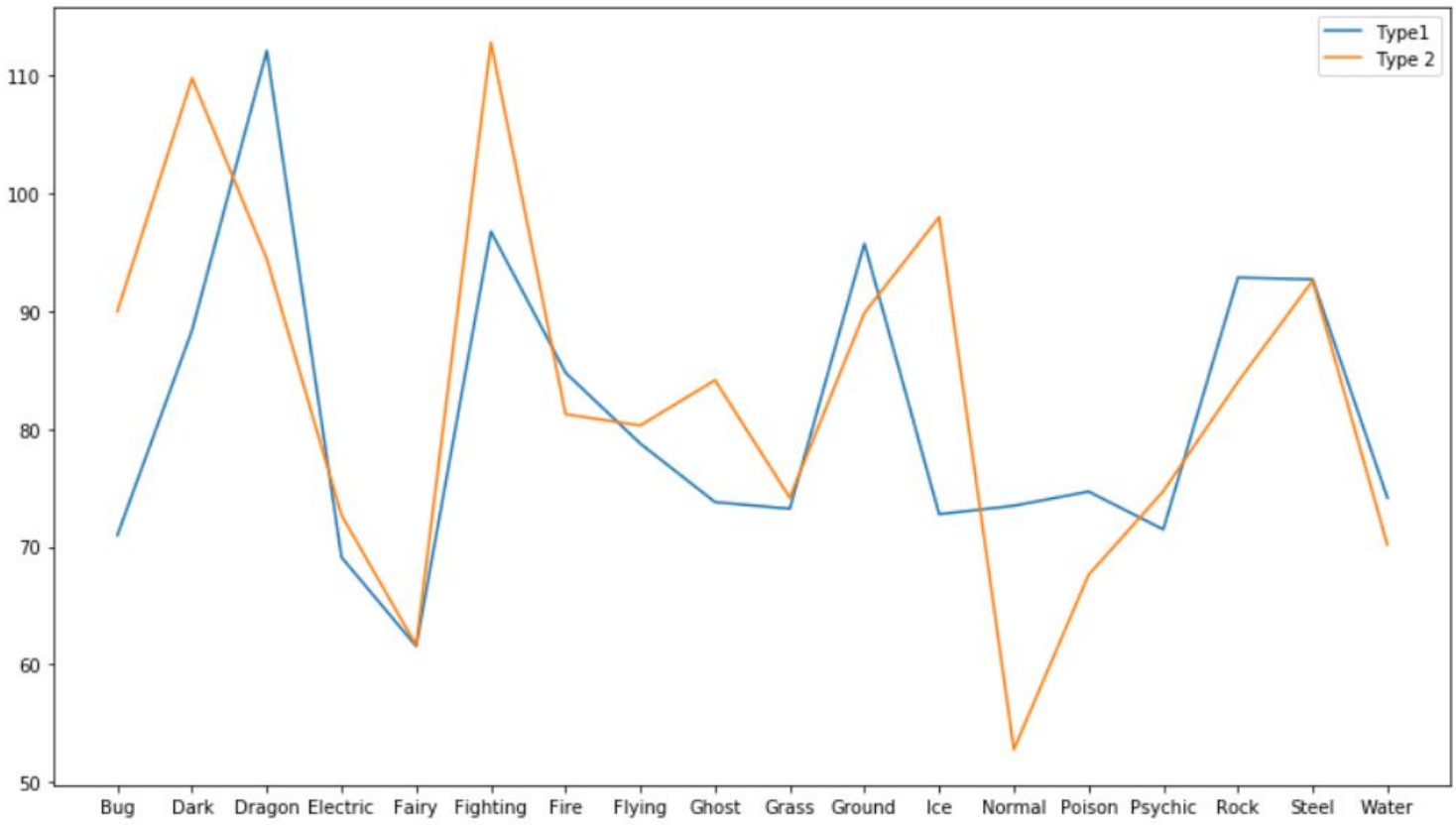
Example

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

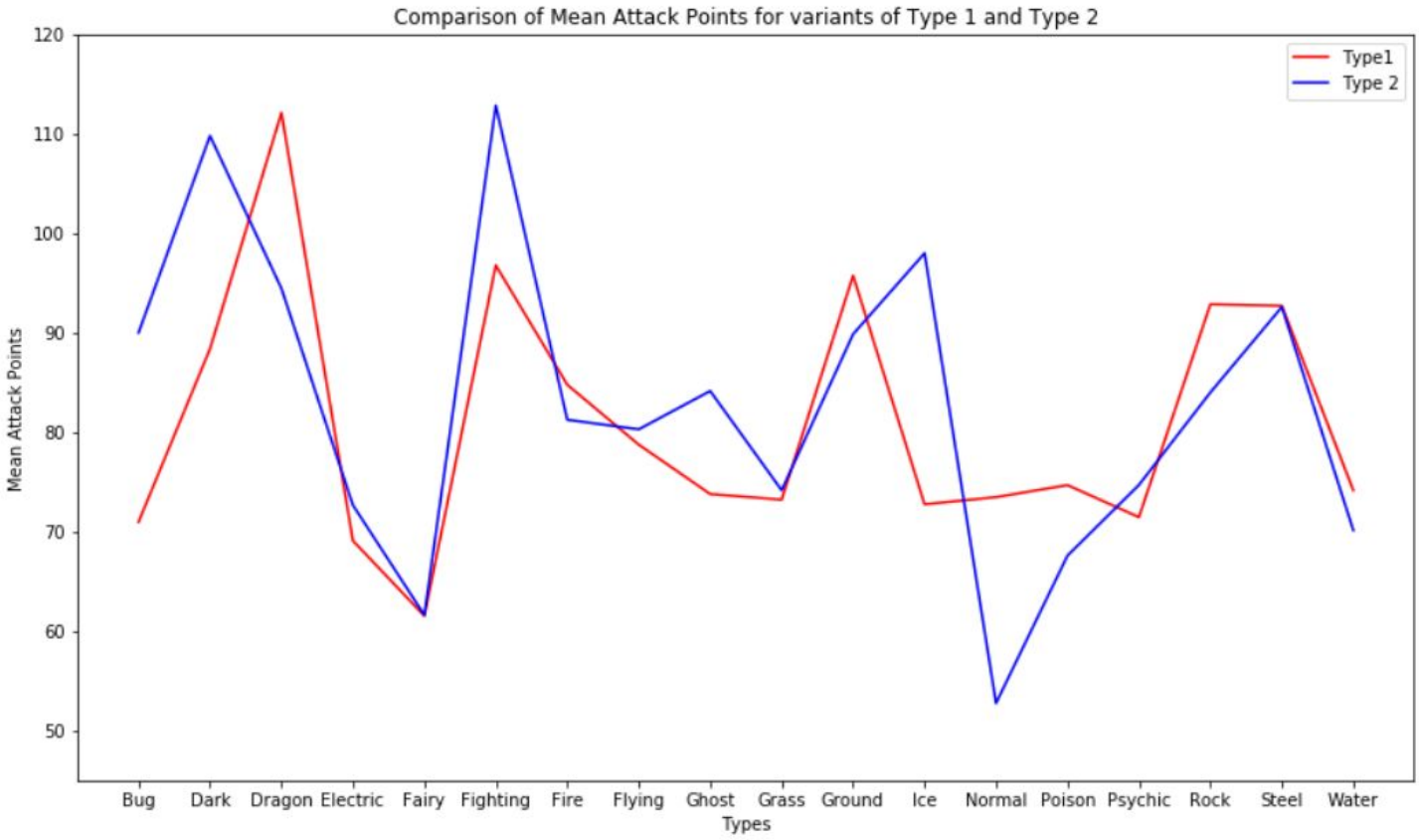
4) Changing color: Use the `color` argument inside the different types of plot functions to change its color.

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



Doesn't it look better than the previous unlabelled plot in terms of information? Definitely yes.

Stacked Bar Chart

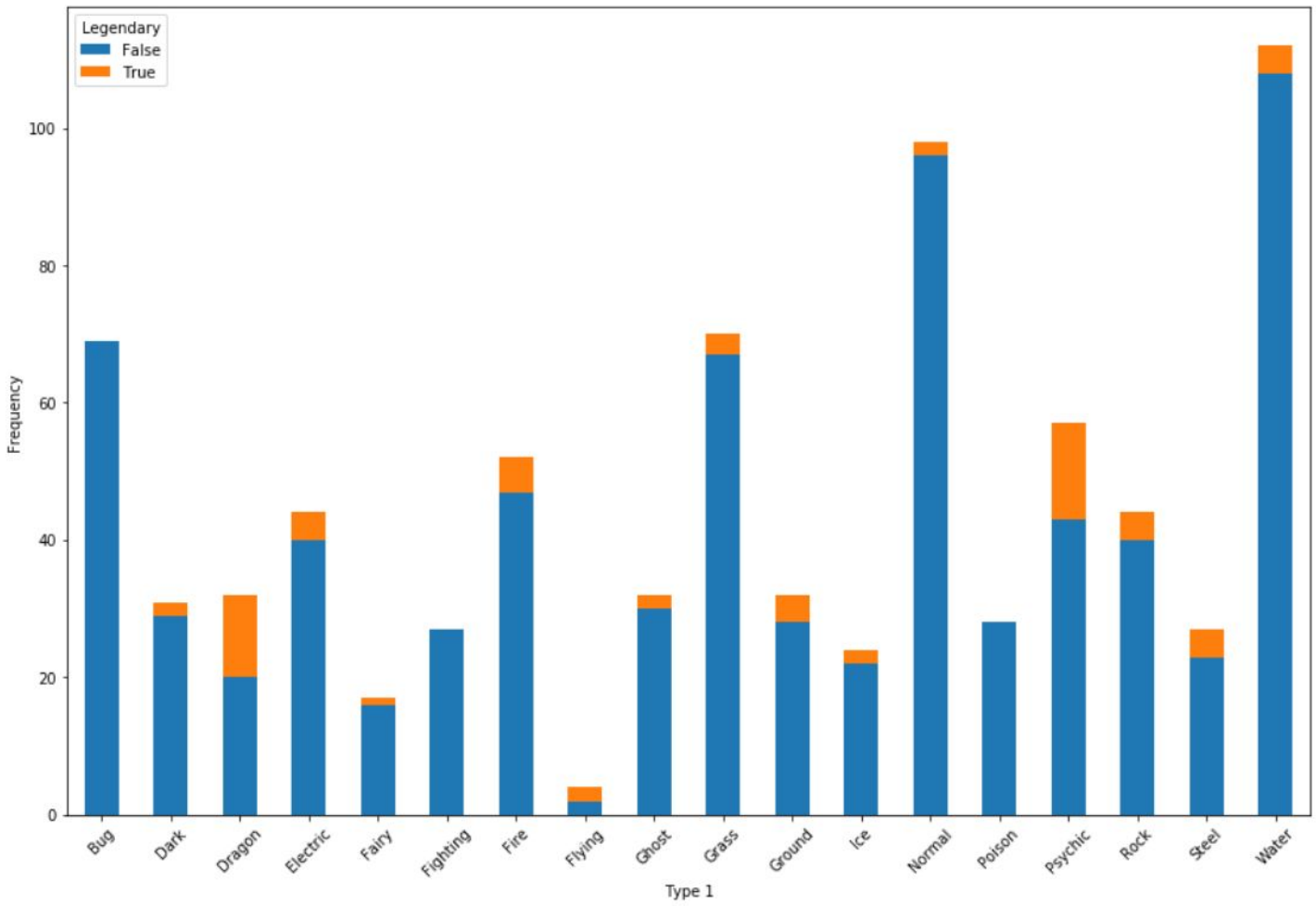
In the previous chapter you had already plotted a bar chart. A bar chart is a powerful visualization capable of representing category counts for a particular feature and can be either horizontal or vertical. But there is another kind of bar chart called stacked bar chart which can factor in another feature besides the feature you already have in the X-axis in a bar chart. Lets say you want to visually inspect the question: Which type (`Type 1`) of Pokemons have the highest chances of being Legendary? The roadmap to answer this question would be:

- First group the Pokemons by `Type 1`
- For every variant of `Type 1` calculate the proportions of Legendary and not Legendary within that variant
- Plot them out Combining `pandas` and `matplotlib` gives an advantage by combining the first two steps i.e. grouping and calculating Legendary counts within each variant. This kind of problem can be effectively visualized with a stacked bar chart. Now lets see how:

```
# Group and calculate Legendary Pokemons within each variant of 'Type 1'

df.groupby(['Type 1', 'Legendary']).size().unstack().plot(kind='bar', stacked=True,
figsize=(15,10))
```

```
# Label X-axes and Y-axes

plt.xlabel('Type 1')

plt.ylabel('Frequency')

# Rotate X-axes labels

plt.xticks(rotation=45)

# Display plot

plt.show()
```

Output



The above figure shows a stacked bar chart with number of legendary pokemons denoted by orange and non-legendary one by blue. It is evident from this plot that type (`Type 1`)`Flying` has the highest chances of being Legendary.

Example:

In this task you will answer the question:

Which generation (`Generation`) has the highest chances of being legendary (`Legendary`)

- First group Pokemons by `Generation` and `Legendary` and then calculate the counts for every variant of generation based on legendary status (`True` or `False`). Save this to a variable `res`
- Plot a stacked bar chart using `.plot(kind='bar', stacked=True, figsize=(15,10))` method

```
# Code starts here

# Group Pokemons
res = df.groupby(['Generation','Legendary']).size().unstack()
# Plot stacked bar chart
res.plot(kind='bar', stacked=True, figsize=(15,10))
# Display plot
plt.show()

# Code ends here
```

Histogram

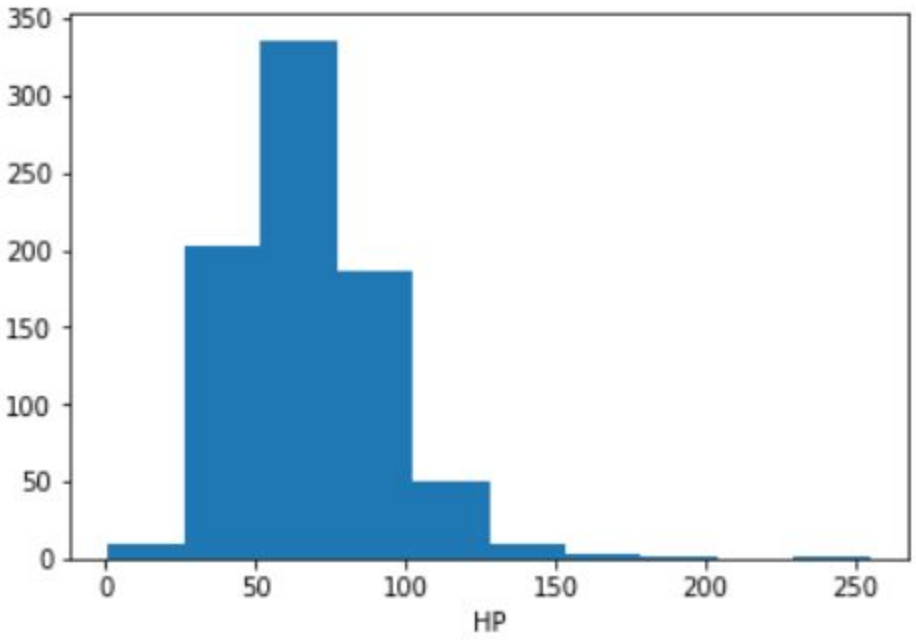
What is a histogram and why do you need it?

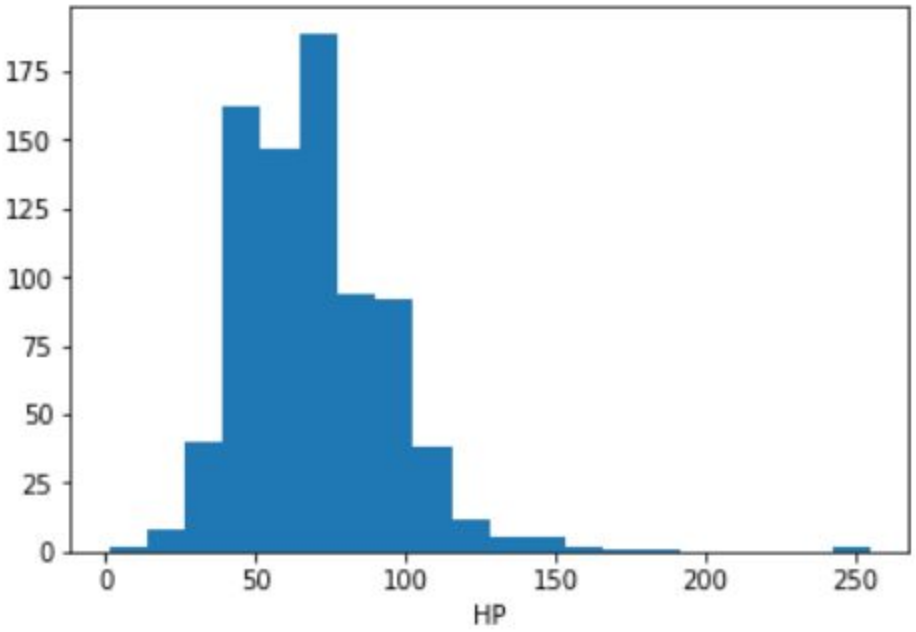
A histogram is a plot that lets you discover, and show, the underlying frequency distribution (shape) of a set of continuous data. This allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc. Note that it requires only one array or series since it displays the frequencies.

How to construct a histogram using matplotlib?

Matplotlib's `.hist()` the method provides an easy way of generating histograms. To construct a histogram from a continuous variable you first need to split the data into intervals, called bins. Each bin contains the

number of occurrences of scores in the data set that are contained within that bin. Below are two plots with bins `10` and `20` carried out with the feature `HP` of the Pokemon dataset.





Key takeaways from these plots

- First and the most important thing; both these plots are the same.
- An important concept with histograms is binning.
    - Binning is a way to group continuous values into a smaller number of bins.
    - More the number of bins more is the number of intervals that leads to less frequency for every interval and vice versa.

    - In the left image, the number of bins is small and so values are tightly packed. The right image has a number of bins, values are loosely packed and hence looks that way.
    - But always remember that both these plots are the same; the difference in their appearance is due to the different number of bins.
- Both these plots have a right tail i.e. have a long tail on the right side
- Some observations have extreme values in the interval `230-260`
- Most of the observations lie in the interval `40-75`

Difference with bar charts

Unlike a bar chart, there are no gaps between the bars (*although some bars might be absently reflecting no frequencies*). This is because a histogram represents a continuous data set, and as such, there are no gaps in the data.

Histograms are based on the area of the bars, not the height of bars

In a histogram, it is the area of the bar that indicates the frequency of occurrences for each bin. This means that the height of the bar does not necessarily indicate how many occurrences of scores there were within each individual bin. It is the product of height multiplied by the width of the bin that indicates the frequency of occurrences within that bin. One of the reasons that the height of the bars is often incorrectly assessed as indicating the frequency and not the area of the bar is due to the fact that a lot of histograms often have equally spaced bars (bins), and under these circumstances, the height of the bin does reflect the frequency.

Example:

In this task, you will plot the distribution of `Attack` points for Pokemons which have their first type (`Type 1`) as `Dragon`. You will also compare the mean values for `Attack` for all the Pokemons against the mean value of `Attack` for dragon type (by drawing a vertical line).

- Calculate the mean attack points for all the Pokemons and store it in a variable `mean_attack`
- Create a dataframe named `dragon` consisting of only `Dragon` type pokemon using conditional filtering (based on `Type 1`)
- Calculate the mean attack points for `dragon` and store it in a variable `mean_dragon`
- Use matplotlib's `.hist()` on `dragon` and pass arguments `column='Attack', bins=8`
- To compare mean attack points you need to draw two lines; one with `mean_attack` and `mean_dragon`
- Use `.axvline()` and pass arguments `x=mean_attack, color='green'` to plot a vertical line representing mean attack points for all the Pokemons

- Use `.axvline()` and pass arguments `x=mean_dragon, color='black'` to plot a vertical line representing mean attack points for Dragon Pokemons
- Calculate mean attack points for all Pokemons using `mean_attack = np.mean(df['Attack'])`
- Make dataframe consisting of only `'Dragon'` type Pokemons with `dragon = df[df['Type 1'] == 'Dragon']`
- Calculate mean attack points for dragon type pokemons using `mean_dragon = np.mean(dragon['Attack'])`
- To visualize distribution of attack points for dragon pokemons use `dragon.hist(column='Attack', bins=8, figsize=(10,10))`
- Use `plt.axvline(x=mean_attack, color='green')` to draw a vertical green colored line indicating the mean attack points for all pokemons
- Use `plt.axvline(x=mean_dragon, color='black')` to draw a vertical black colored line indicating the mean attack points for only dragon pokemons
- Finally use `plt.show()` to display the plot

```
# Code starts here


# Mean 'Attack' for all Pokemons
mean_attack = np.mean(df['Attack'])
print(mean_attack)
# Mean 'Attack' for Dragon type Pokemons
dragon = df[df['Type 1'] == 'Dragon']
print(dragon)
# Histogram for Dragon type Pokemons
mean_dragon = np.mean(dragon['Attack'])
print(mean_dragon)
# Display plot
dragon.hist(column = 'Attack',bins = 8,figsize = (10,10))
plt.axvline(x = mean_attack, color = 'green')
plt.axvline(x = mean_dragon, color = 'black')
plt.show()
# Code ends here
```

Scatter Plot

Why use scatter plots?

A question for many data sets is whether two items are related to each other in some way, that is, are they correlated? In our case, you can ask the question of whether `Attack` and `Defense` points are related to each other. Scatter plot helps us answer this kind of questions.

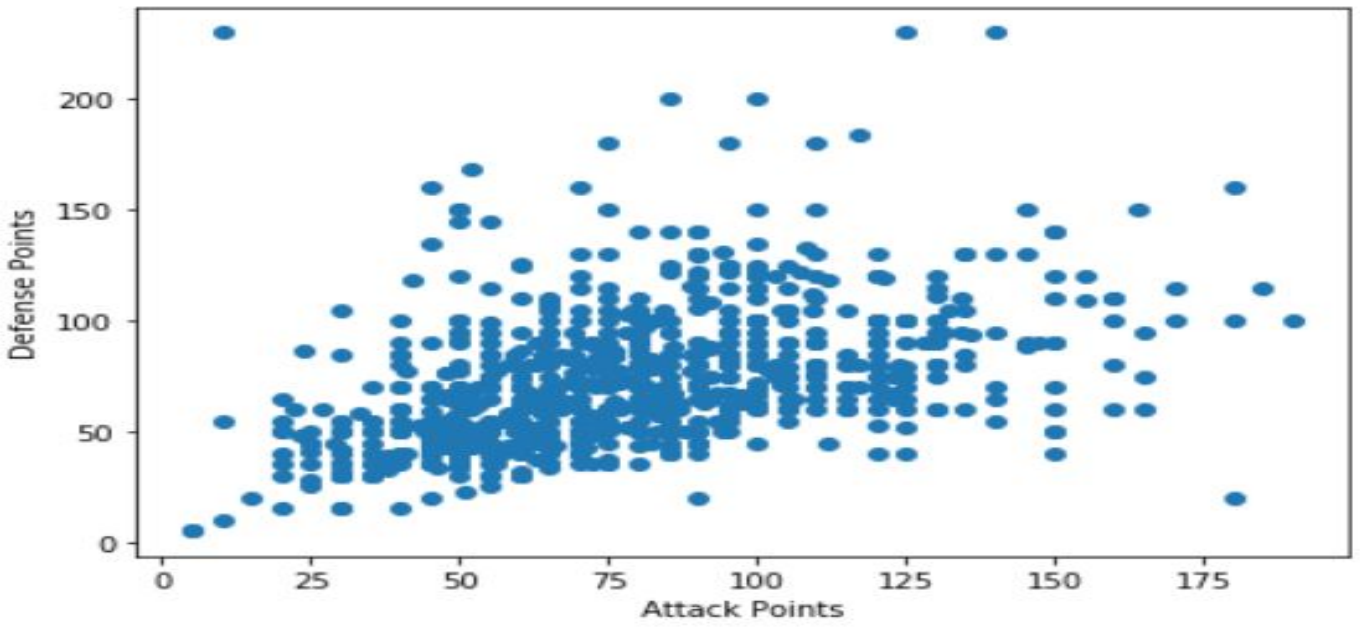
What is a scatter plot?

A scatter plot is a two-dimensional data visualization that is used to represent the values obtained for two different variables - one plotted along the x-axis and the other plotted along the y-axis. For generating a scatter plot you need two numerical arrays of data.

When to use a scatter plot?

A scatter plot helps us determine if two quantities are weakly or strongly correlated. Correlation implies that as one variable changes, the other also changes. While calculating the correlation coefficient will give us a precise number, a scatter plot helps us find outliers, gain a more intuitive sense of how to spread out the data is, and compare more easily.

For example:- The following scatter plot gives a clear indication between `Attack` and `Defense` points. As you can see there is a positive linear relationship between the `Attack` and `Defense` points.



Drawing scatter plots

You can use `.scatter(x, y)` to generate a scatter plot for two numeric arrays `x` and `y`. Using `pandas` you can achieve the same with a dataframe `df` using `df.plot.scatter(x=column1, y=column2)` where `column1` and `column2` are the column names present in the dataframe `df`. Keep in mind that both `column1` and `column2` must be numeric columns.

The code snippet is given below:

```
# Scatter plot with matplotlib
plt.scatter(df['Attack'], df['Defense'])
```

or you can do it with `pandas` also by:

```
# Scatter plot with pandas
df.plot.scatter(x='Attack', y='Defense')
```

Drawing Multiple Plots

Subplots

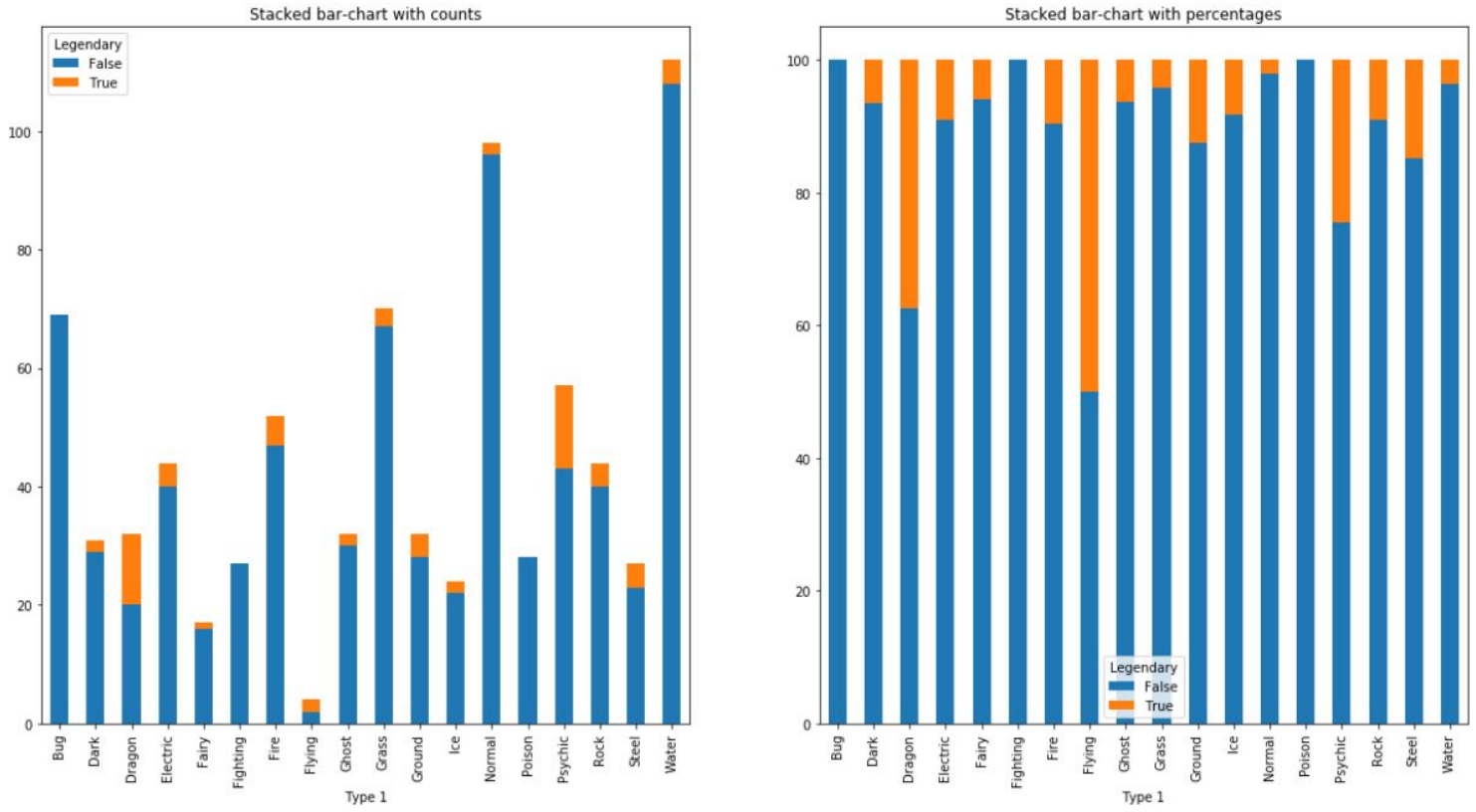
Till now you have drawn only a single plot within the figure. Many times you will be required to draw multiple plots on the same figure for better comparison. Remember the stacked bar chart that had every variant of `Type 1` and displayed their `Legendary` status? It was used to answer the question Which type (`Type 1`) of Pokemons has the highest chances of being `Legendary`?

A better way would answer this question is rather than getting the counts why don't we get the percentages and compare both the side of the plot by side? To achieve it i.e. place two plots side by side in the same figure we will use `.subplots()` method of matplotlib. Let's see how you can do it:

```
# Initialize figure and axes
fig, (ax_1, ax_2) = plt.subplots(1,2, figsize=(20,10))

# Stacked bar-chart representing counts
res = df.groupby(['Type 1', 'Legendary']).size().unstack()
res.plot(kind='bar', stacked=True, ax=ax_1)
ax_1.set_title('Stacked bar-chart with counts')
```

```
# Stacked bar-chart representing percentages
new_res = res.fillna(0)
new_res['Total'] = new_res[True] + new_res[False]
new_res[True] = (new_res[True] / new_res['Total']) * 100
new_res[False] = (new_res[False] / new_res['Total']) * 100
new_res.drop('Total', inplace=True, axis=1)
new_res.plot(kind='bar', stacked=True, ax=ax_2)
ax_2.set_title('Stacked bar-chart with percentages')
```

Output



It is very clear that the second plot (the right one) gives a much better visualization and effectively answers our question as all the categories are on the same scale.

So, what happened in the code snippet is described below:

- In the code snippet, we initialized two axes `ax_1` and `ax_2` along with the figure `fig` using `.subplots(1,2)` with figure size initialized by `figsize=(20,10)`.
- The `1` and `2` inside `.subplots()` simply indicates the number of rows and number of columns in the figure and their product i.e. `1*2 = 2` gives the number of plots or axes
- In the first axes i.e. `ax_1` we draw the stacked bar-chart representing the counts of `Type 1` pokemon with `Legendary` status. To specify the axes, we had used `ax=ax_1` argument inside `.plot()` in line number `6`

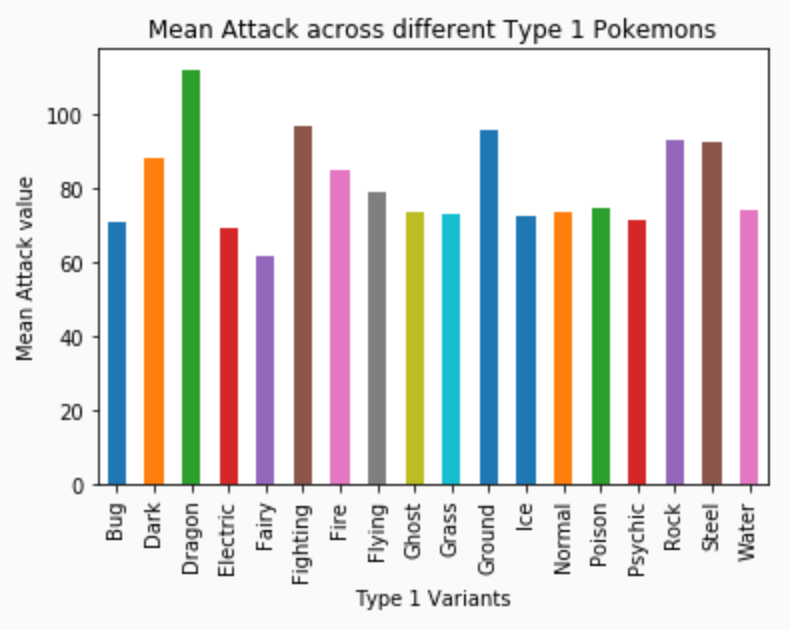
- The second axes `ax_2` gives the representation of percentages of `Legendary` Pokemons according to `Type 1` variants. To specify the axes, we had used `ax=ax_2` argument inside `.plot()` in line number `15`.

Visualizing Data Effectively

Visualize! - The good, the bad and the ugly

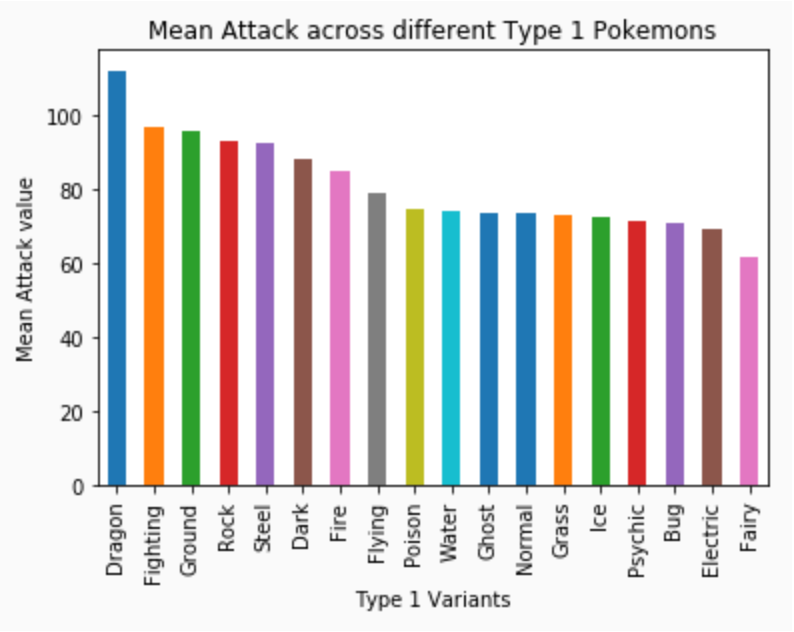
Till now, our focus was on visualizing data through matplotlib and pandas. Let us abstract it out a bit and ask the question - how do we visualize data effectively? With every plot and figure, we try to show, we must be able to convey the insights properly to the end audience. How do we achieve it through `matplotlib` and `pandas`? Let's get back to our friend - the pokemon enthusiast and find out what other questions he wants an answer to.

The enthusiast has asked the question - which type (`Type 1`) of Pokemons have the highest attack on an average? The corresponding graph is

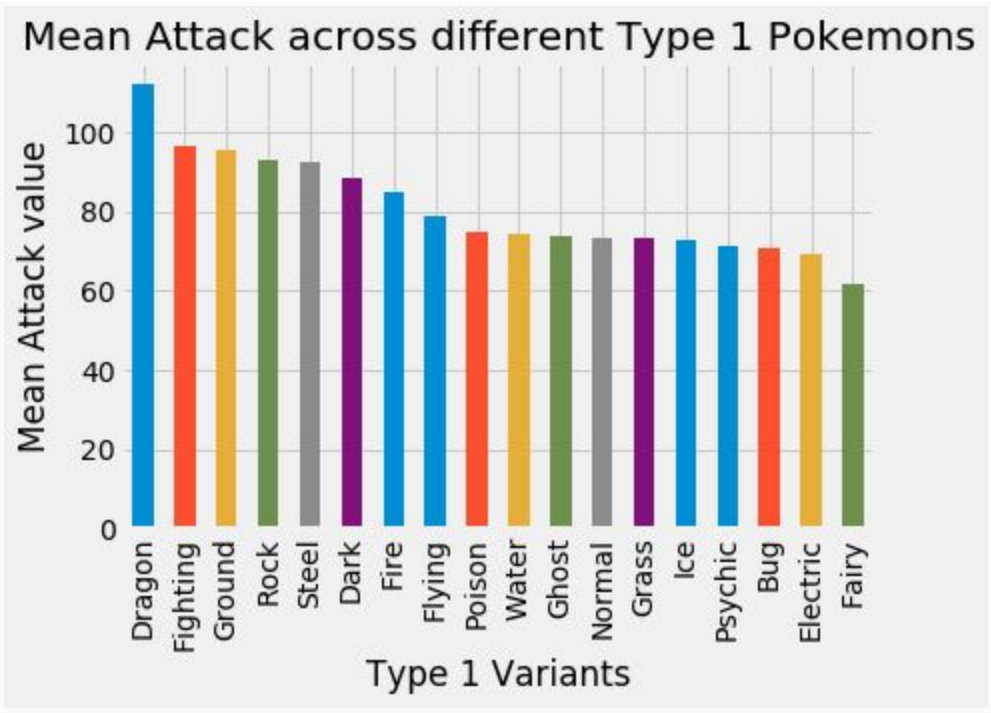


This is a decent visualization. But the answer to our question is not visible clearly for it is lost in the multi-coloured bars. So let's try to customize the graph so that the answer we are trying to convey is highlighted the most.

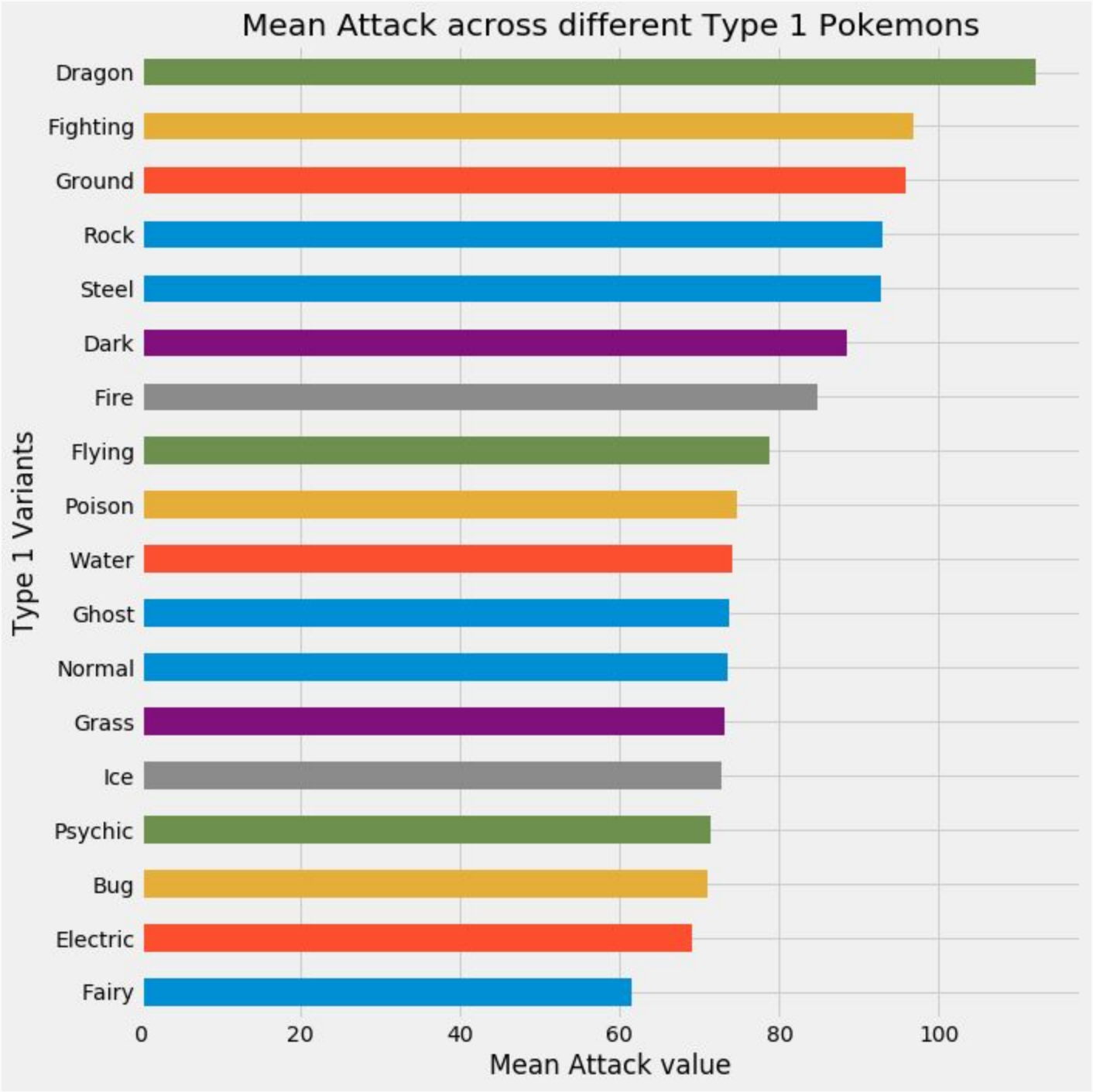
First let's sort the values so that all values are shown in descending order. And then let's look at the graph.



There are different styles available to use. Let's use one of them - `fivethirtyeight`.



Looks much better. But trying to read the labels is difficult. Maybe flipping the graph would be better. Let's try and see if it becomes better to visualize.



In this visualization, the message we wanted to highlight - dragon type pokemon is having the highest mean attack comes out clearly. From the standpoint of proper aesthetic visualization, there are things yet to be done. But at the preliminary level, the graph is communicating the message we want it to.

Guidelines for visualization

Now that we have a pretty good idea in doing visualization using python, let us keep some high-level guidelines in mind while doing visualizations as a data scientist in future.

- Keep your audience in mind.

    - Always be mindful of the audience you are creating the visuals for. Tailor it for the main stakeholder instead of pleasing everyone.
- Make the message the hero.
    - Keep in mind the top insight you want to onboard for the audience. Make the visual convey that message only and tone down the rest.
- Keep it simple.
    - Make the visual as simple as possible. A flashy visual doesn't mean it's a good visual.
- Choose the right visual for your purpose.
    - Know the purpose clearly and choose the visual accordingly. Throughout the concept, you have seen which visuals need to be used for what purposes.

These basic guidelines are good principles to think and implement whenever we are trying to create visualizations in our role as a data scientist. With this, we are ready to go forward in the next step of the journey as a data scientist.