Data Visualization Using Python:

====================== Matplotlib

Matplotlib is a python 2D plotting library providing features like plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. It provides the pyplot module which is quite similar to MATLAB providing a similar interface. Common functions to MATLAB includes full control of line styles, font properties, axes properties, etc.

Matplotlib is heavily dependent upon other packages like Numpy which can be considered as one of its drawbacks. To import Matplotlib in Python we use the following syntax -

import matplotlib.pyplot as plt

We will be understanding the basics of Matplotlib functioning along with the plots. To do so, we take a Kaggle dataset named "Pokémon for Data Mining and Machine Learning" as part of our case study.

**Bar Graph**

The Pokemon dataset consists of several features and values as can be seen in the downloaded [CSV](https://lex.infosysapps.com/content-store/Infosys/Infosys_Ltd/Public/lex_auth_0126429276066693121779/web-hosted/assets/pokemon_alopez247.csv) file. Let us try to plot the number of Pokemon against their *Type\_1* category. To initiate, we store the dataset in a dataframe *df*.

2. import pandas as pd
3. df = pd.read\_csv('pokemon\_alopez247.csv')

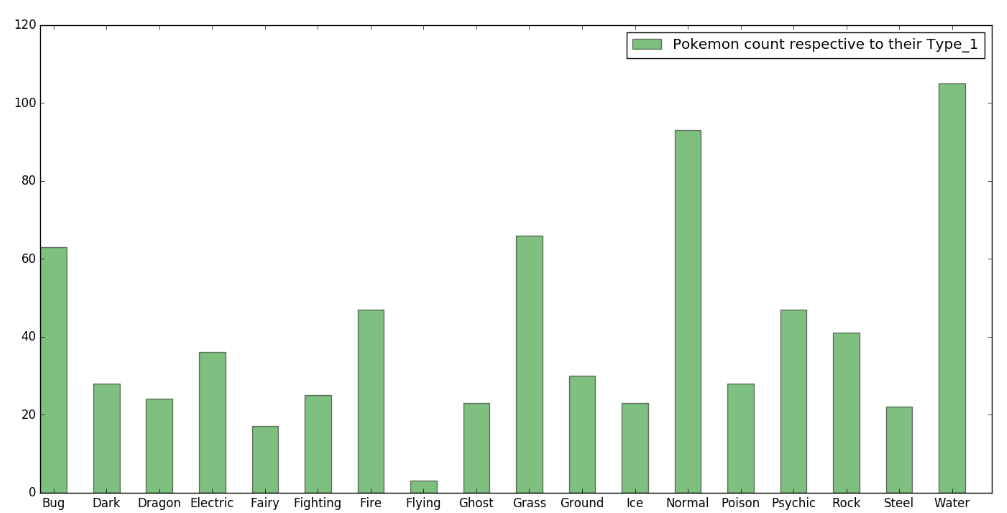
Next, let us count the number of distinct groups along with the number of Pokemon(s) belonging to that group available under *Type\_1* category. We use *scipy.stats* module to import *itemfreq* function which results in the total number of distinct samples with their frequency.

2. from scipy.stats import itemfreq
3. *# Retrieve a list of group types against the*
4. *# number of pokemon(s) belonging to that group*
5. type\_1 = itemfreq(df.iloc[:,2])
6. print(type\_1)
7. *# Total number of distinct groups*
8. type\_1\_grps = len(type\_1)
9. print(type\_1\_grps)
10. *# Names of group*
11. type\_1\_names = type\_1[:,0]
12. print(type\_1\_names)
13. *# Pokemon count particular to each group*
14. type\_1\_count = type\_1[:,1]
15. print(type\_1\_count)

Since so far we are able to retrieve the number of groups and their Pokemon count, now let us plot a bar graph against these variables.

2. type\_1\_grps = np.arange(type\_1\_grps)
3. bar\_width = 0.5
4. plt.bar(type\_1\_grps, type\_1\_count, bar\_width,
5. alpha = 0.5, *# tranparency factor*
6. color = 'g', *# color factor*
7. label='Pokemon count respective to their Type\_1')
8. plt.legend(loc='best')
9. plt.xticks(type\_1\_grps + bar\_width/2, type\_1\_names)

We get a bar plot with names of groups in X-axis ticks with a legend stating "Pokemon count respective to their Type\_1" as shown below:



# Advanced bar graph

Previously, we have seen how to plot a simple bar graph with some given groups. Now, let us improve our graph by adding some details like labels, legend, and colors.

## Adding distinct colors

We are aware of the number of groups i.e. 18 therefore, we will create some 18 random colors for each of our group.

**Note:** The final graph shown below can vary from your graph based on the random colors.

2. import random
3. import numpy as np
4. *# Creating 18 random colors range (0 - 1)*
5. clrs = np.linspace( 0, 1, 18 )
6. random.shuffle(clrs)
7. *# Creating final list of 18 random colors*
8. colors = []
9. for i in range(0, 72, 4):
10. idx = np.arange( 0, 18, 1 )
11. random.shuffle(idx)
12. r = clrs[idx[0]]
13. g = clrs[idx[1]]
14. b = clrs[idx[2]]
15. a = clrs[idx[3]]
16. colors.append([r, g, b, a])

Next, we pass these random colors into the bar graph. Also, we are not providing label argument within the bar function which will be taken care of inside legend function.

2. bar\_graph = plt.bar(type\_1\_grps, type\_1\_count, bar\_width,
3. alpha = 0.5, *# tranparency factor*
4. color = colors) *# color factor*

## Legend

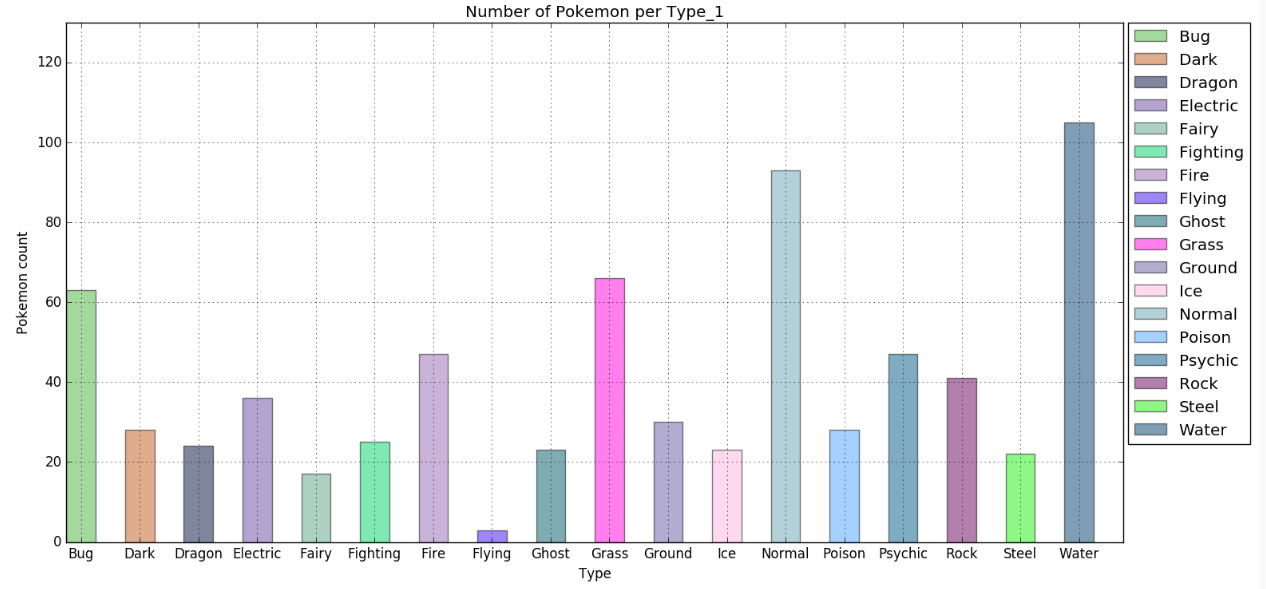
We saw only one label inside the legend section in our last plot. This time we're setting up labels particular to each group name. This can be done as follows:

2. plt.legend(bar\_graph,
3. type\_1\_names, *# List of group names*
4. bbox\_to\_anchor=(1.128, 1.015)) *# Position of legend*

## Adding text to plot

Next, we're going to add names of X-axis and Y-axis using xlabel() and ylabel() functions. We can also add a suitable title using title() function along with defining the visibility of grid and setting limit on axis.

2. plt.xticks(type\_1\_grps + bar\_width/2, type\_1\_names)
3. plt.xlabel('Type')
4. plt.ylabel('Pokemon count')
5. plt.title('Number of Pokemon per Type\_1')
6. plt.grid()
7. plt.ylim(0,130)



# Pie chart

We can clearly observe the major groups from Type\_1 feature with more number of belonging Pokemons. Now, let us compare the Attack, Defence, Speed and HP of top 4 major groups for our next analysis. These characteristics define following traits:

* **Attack:** Aggressiveness
* **Defence:** Safeguarding
* **Speed:** Swiftness
* **HP:** Number of hits a Pokemon can receive before it faints

Also, we select only these 4 features out of the complete dataframe to reduce time complexity.

2. df\_pie = df[['Type\_1', 'Attack', 'Defense', 'Speed', 'HP']].copy()
3. print(df\_pie.head())

Next, we again find the frequency of each group using the itemfreq() command as we did before and thereby, select top 4 most frequent groups.

2. from scipy.stats import itemfreq
3. frequent\_grp = itemfreq(df\_pie.iloc[:,0])
4. frequent\_grp = np.array(sorted(frequent\_grp, key=lambda x: x[1]))[::-1][0:4,:]
5. print(frequent\_grp)

From the frequent\_grp array we can observe that topmost four frequent groups are Water, Normal, Grass and Bug. We can now select the 4 listed features (Attack, Defence, Speed and HP) corresponding to these groups.

2. df\_pie = df\_pie.loc[df\_pie.loc[:,'Type\_1'].str.contains(r'(Water|Normal|Grass|Bug)')]
3. print(df\_pie)

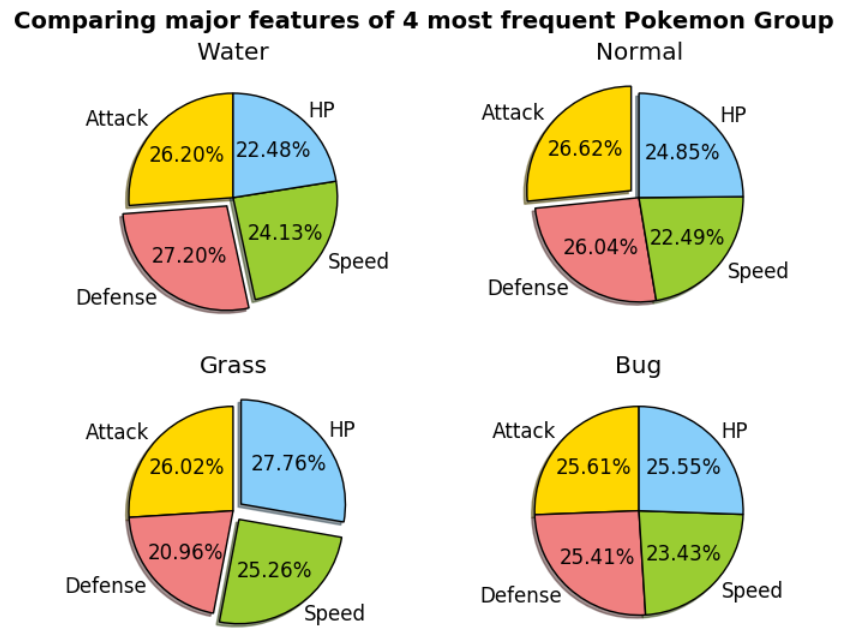
So finally, we are left with the features of the targeted 4 groups. Now, let us find the **mean** of each group's feature from which we can analyze which group is strong/lack in which area.

2. *# Names of the group*
3. type\_1\_names = frequent\_grp[:,0]
4. print(type\_1\_names)
5. *# Mean of samples for each feature corresponding to all 4 group*
6. df\_grp = df\_pie.groupby('Type\_1').mean()
7. print(df\_grp)

The variable df\_grp stores the mean value of each feature corresponding to all four groups. Now, let us plot a visualization using pie charts for more clarification.

2. names = df\_grp.columns
3. colors = ['gold', 'lightcoral',
4. 'yellowgreen', 'lightskyblue']
5. explode = (0, 0, 0, 0.1) *# takes out only the 4th slice*
6. fig, [[ax1, ax2], [ax3, ax4]] = plt.subplots(nrows=2, ncols=2)
7. ax = [ax1, ax2, ax3, ax4]
8. for i in range(0,4):
9. percent = df\_grp.iloc[i,:]
10. ax[i].pie(percent, explode = explode,
11. labels = names, colors = colors,
12. autopct='%.2f%%', *# display value*
13. shadow=True,
14. startangle=90)
15. ax[i].set\_aspect('equal')
16. ax[i].set\_title(type\_1\_names[i])
17. plt.suptitle('Comparing major features of 4 most frequent Pokemon Group',
18. fontsize = 14,
19. fontweight = 'bold')

Since we have 4 groups to visualize, therefore, we needed something called as subplots which plot many charts into the same figure. The final figure looks something like this:



From the chart, we can state the following about the given groups of Pokemon:

* **Fastest group:** Grass Type
* **Aggressive group**: Normal Type
* **Defensive group:** Water Type
* **Strongest Group:** Grass Type

 make animated graphs using Matplotlib Animation module. (Compile and execute in Spyder)

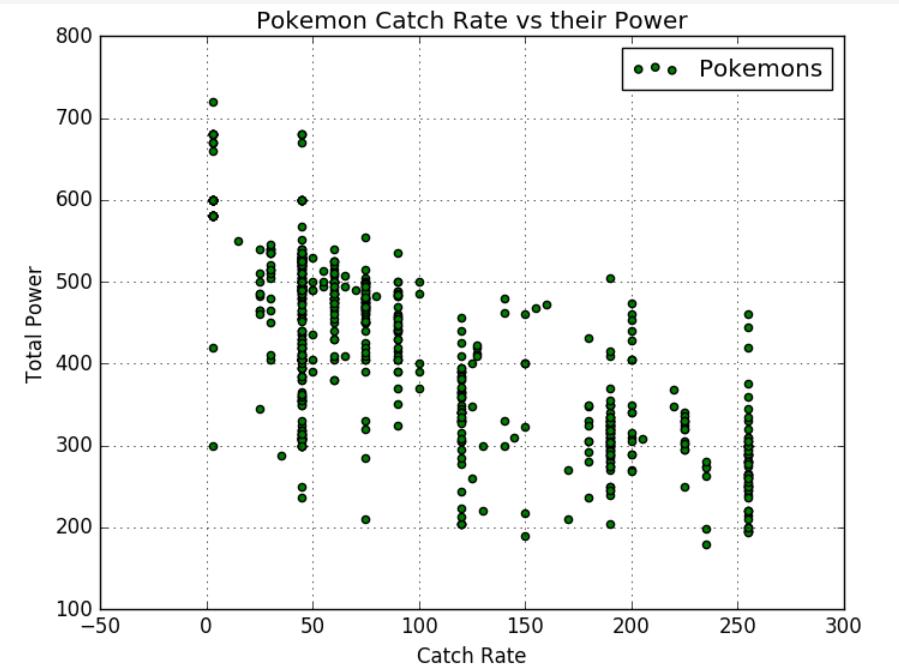
2. import numpy as np
3. import matplotlib.pyplot as plt
4. from matplotlib import animation
6. n = 30
7. x = np.arange(0,1,0.001)
8. y = np.ones( (1000,n) )
9. for i in range(0,n):
10. y[:,i] = np.sin(2 \* np.pi \* x) \* i+1
11. def func(arg):
12. plt.cla() *#Clear axis*
13. plt.plot(y[:,arg])
14. plt.ylim(-30,30)
15. fig = plt.figure(figsize=(5,4))
17. to\_save = animation.FuncAnimation(fig, func, frames=30)

# Scatter plot

In the last section, we analysed the topmost 4 groups based on their features. Now let us try to figure out if there is any relation between the Pokemon catch rate and total power. To do so, we plot two features namely Total and Catch\_Rate against each other using scatter plot.

2. import pandas as pd
3. import matplotlib.pyplot as plt
4. df = pd.read\_csv('pokemon\_alopez247.csv')
5. tot\_power = df.iloc[:,4]
6. print(tot\_power.head(4))
7. catch\_rate = df.iloc[:,21]
8. print(catch\_rate.head(4))
9. fig, ax = plt.subplots()
10. p = ax.scatter(catch\_rate, tot\_power, c = 'g')
11. ax.grid()
12. ax.set\_xlabel('Catch Rate')
13. ax.set\_ylabel('Total Power')
14. ax.set\_title('Pokemon Catch Rate vs their Power')
15. plt.legend([p],['Pokemons'])

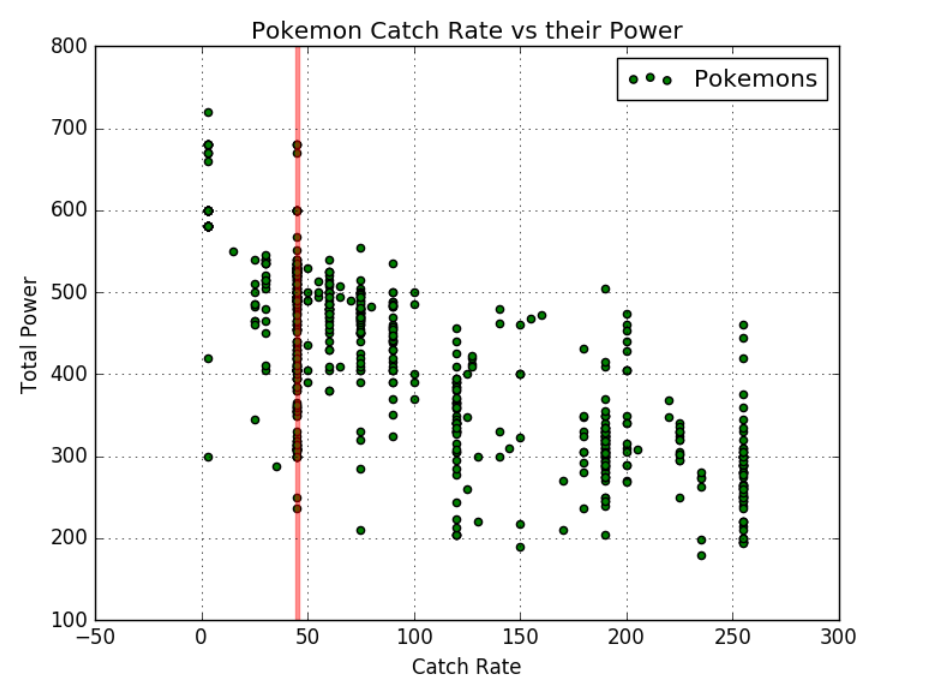
The resultant plot marks all the Pokemon on the graph based upon their catch rate and power.



We can observe that there are various Pokemon who despite having less Power still have less Catch Rate. For more clarity, we can highlight one such common group present at catch rate 45. Matplotlib provides patches through which we can make an overwriting to highlight specified area on our previous plot.

2. import matplotlib.patches as patches
3. import matplotlib.transforms as transforms
4. trans = transforms.blended\_transform\_factory(
5. ax.transData,ax.transAxes)
6. rect = patches.Rectangle((44,0), width=2, height=5,
7. transform=trans, color='red',
8. alpha=0.4)
9. ax.add\_patch(rect)

Once highlighted, we get a plot as shown below. We can also change the shape of the highlighted area as per requirements or plot multiple highlighted areas too.



Since we have visualized that there are some adamant Pokemon who have fairly less catch rate despite having less power. So, let us name few of them with conditions *Catch\_Rate* equals 45 and *Power* less than or equal to 330.

2. catch\_rate\_45 = df[df.loc[:,'Catch\_Rate'] == 45]
3. pow\_330 = catch\_rate\_45[catch\_rate\_45.loc[:,'Total'] <= 330]
4. print("Number of such Pokemons:", len(pow\_330))
5. *# Top 10 adamant Pokets*
6. print(pow\_330.loc[:,'Name'].head(10))

=============== 3D Scatter Plot

Fetch your code from last exercise (3D scatter plotting) and append the below code to observe rotating plots.

2. *## rotate the axes and update*
3. from mpl\_toolkits.mplot3d import axes3d
4. *# consiering you saved your Axes under variable ax*
5. for angle in range(0, 360):
6. ax.view\_init(30, angle)
7. plt.draw()
8. plt.pause(.001)

# Line chart

From the last section we have observed that top 3 adamant Pokemons with Power less than or equal to 330 and Catch\_Rate equals 45 are:

* Bulbasaur
* Charmander
* Squirtle

So, let us try to compare their major features to analyze their weak and strong areas. This time we're considering Total, HP, Attack, Defense, Sp\_Atk, Sp\_Def, Speed, Height\_m, and Weight\_kg as judging features.

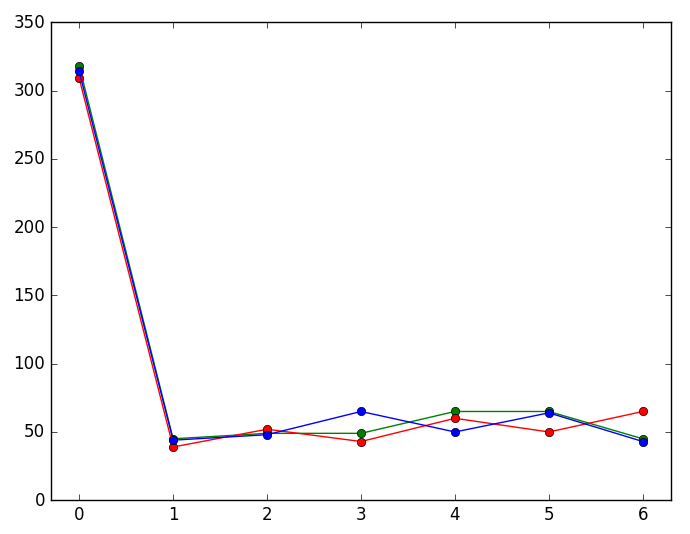
We extract the values corresponding to these features and save them in two variables namely, pokets (containing starting seven features, excluding height and weight) and hw (containing height and weight features).

2. import numpy as np
3. import pandas as pd
4. import matplotlib.pyplot as plt
5. df = pd.read\_csv('pokemon\_alopez247.csv')
6. bulbasaur = list(df.iloc[0,4:11])
7. charmander = list(df.iloc[3,4:11])
8. squirtle = list(df.iloc[6,4:11])
9. pokets = [bulbasaur, charmander, squirtle]
10. *# Converting height from meters to inches*
11. *# Weight remains in kg*
12. bul\_hw = [df.iloc[0,19]\*3.281, df.iloc[0,20]]
13. char\_hw = [df.iloc[3,19]\*3.281, df.iloc[3,20]]
14. squ\_hw = [df.iloc[6,19]\*3.281, df.iloc[6,20]]
15. hw = [bul\_hw, char\_hw, squ\_hw]

This time, let us make our subplots more dynamic in terms of shape and defined location, using subplot2grid() function. For the very first subplot, we establish comparison relationship among all three Pokemon using pokets container. This can be achieved using line and dot charts as shown:

2. fig = plt.figure(0)
3. *# Line plot*
4. ax1 = plt.subplot2grid((4,3), (0,0), colspan=3)
5. ax1.plot(bulbasaur,'g-',bulbasaur,'go')
6. ax1.plot(charmander,'r-',charmander,'ro')
7. ax1.plot(squirtle,'b-',squirtle,'bo')
8. ax1.set\_xlim(-0.3,6.3)

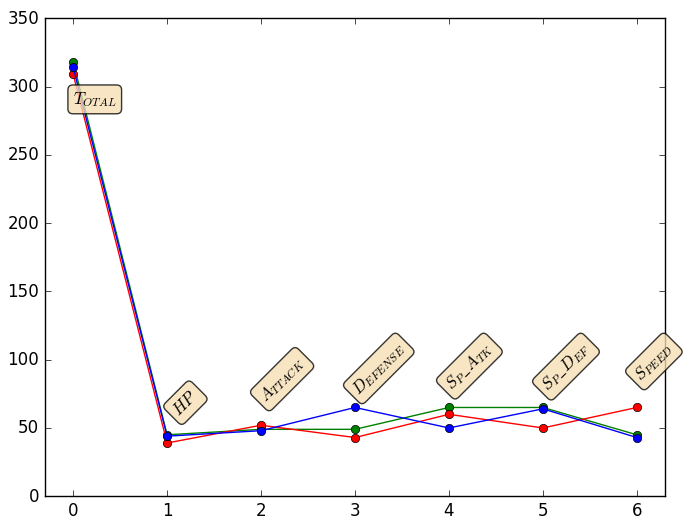
The above scripts combined results in the following plot:



We will be integrating legend in the end when all the subplots are summarized in a single figure. Here, we have 7 dis-localities each of which equals to a certain feature. We can use *text()* function to mention each one of them in TeX format.

2. text\_x\_coord = [0.045, .19, 0.34, 0.49, 0.64, 0.795, 0.945]
3. text\_y\_coord = [0.7, 0.57, 0.65, 0.75, 0.75, 0.75, 0.72]
4. rot = [0, 45, 45, 45, 45, 45, 45]
5. txt = [r'$T\_{OTAL}$', r'$HP$', r'$A\_{TTACK}$', r'$D\_{EFENSE}$',
6. r'$S\_{P}\\_A\_{TK}$',r'$S\_{P}\\_D\_{EF}$',r'$S\_{PEED}$']
7. for i in range(0, 7):
8. ax1.text(text\_x\_coord[i], text\_y\_coord[i], txt[i],
9. transform = ax1.transAxes, *# makes width and height in percentage*
10. va = 'top',
11. rotation = rot[i],
12. bbox = dict(
13. boxstyle = 'round',
14. facecolor = 'wheat',
15. alpha = 0.78)) *# alpha -> transparency*

This results in our first subplot through which we can compare each Pokemon's feature figuratively.



Analysis of above chart states that almost all three Pokemon are equal in given features except two distinct observations:

* Squirtle is much better in terms of defense.
* Charmander has much more swiftness in its body but less Special Defense.

# Box Plot

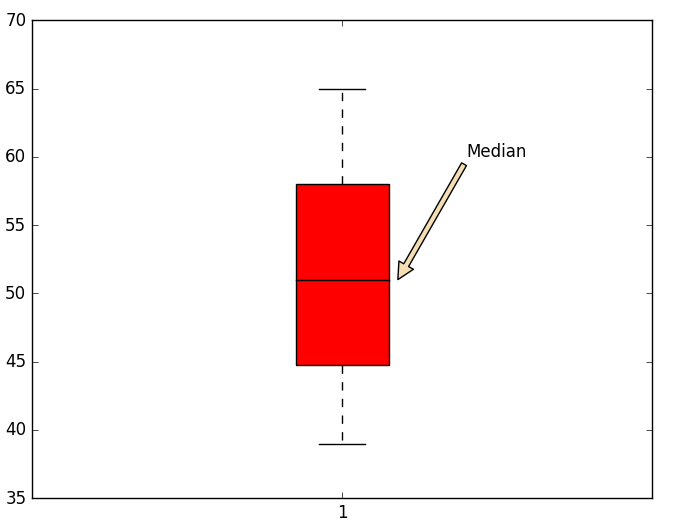
Moving further, let us analyze the minimum, maximum and average traits of each Pokemon. This can be achieved by simply plotting a box plot corresponding to each one of the Pokemon's trait. Therefore, let us chose HP, Attack, Defense, Sp\_Atk, Sp\_Def, and Speed as defining features. We are going to plot them separately and join them with last analyzed plot (Line chart).

1. *# continuing from previous code*
2. ax2 = plt.subplot2grid((4,3), (1,0), colspan=1)
3. ax3 = plt.subplot2grid((4,3), (2,0), colspan=1)
4. ax4 = plt.subplot2grid((4,3), (3,0), colspan=1)
5. ax = [ax2, ax3, ax4]
6. colors = ['g', 'r', 'b']
7. for i in range(0, 3):
8. bp = ax[i].boxplot(pokets[i][1:],patch\_artist=True)
9. *# Adding colors to edges*
10. for element in ['boxes', 'whiskers', 'fliers', 'means', 'medians', 'caps']:
11. plt.setp(bp[element], color='k')
12. *# Adding color inside the box*
13. for patch in bp['boxes']:
14. patch.set(facecolor=colors[i])
15. ax[i].set\_ylim(35,70)

This will result into 3 subplots each meant for individual Pokemon. Meanwhile, we can also add a brief annotation depicting the median.

1. ax3.annotate('Median', xy=(1.09, 51),
2. xytext=(1.2, 60),
3. arrowprops=dict(facecolor='wheat',
4. shrink=0.001),)

The box plot for Pokemon Charmander is shown below:



Analysis of all three box plots reveals that Median value of all three Pokemon lies in almost similar range to each other. Hence, on an average, all three of them are equal in strength/weakness.

# Integrated plots

So far, we have used all the starting seven features to illustrate comparison among the traits of all three Pokemon. Next, we take the last two features namely Weight and Height and plot a bar graph as shown:

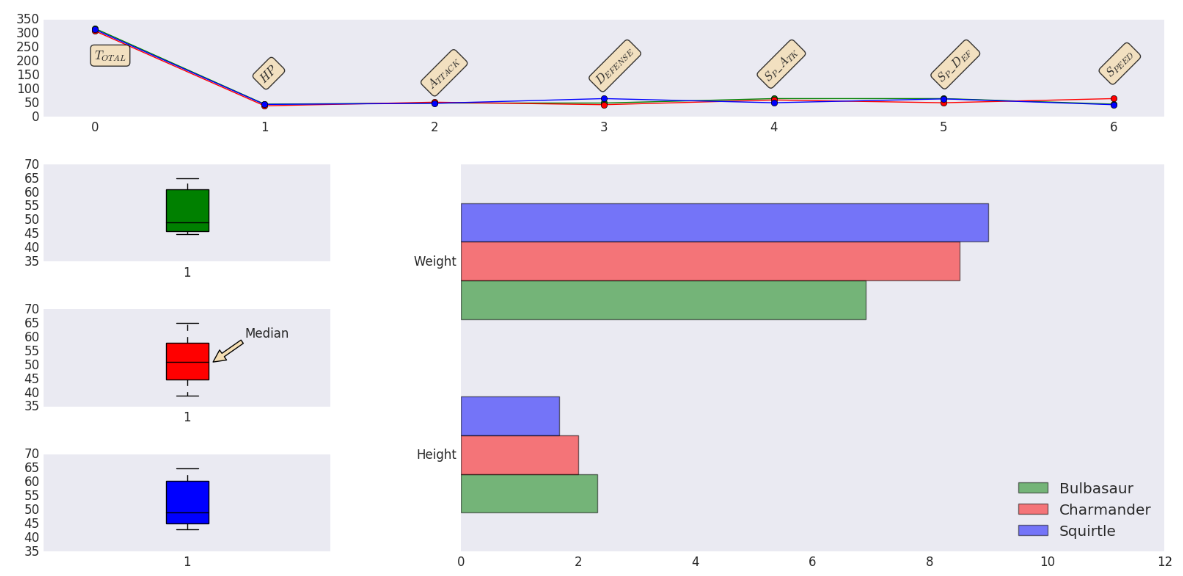
2. *# continuing from previous code*
3. ax5 = plt.subplot2grid((4,3), (1,1), colspan=2, rowspan = 3)
4. bar\_width = 0.2
5. wdt = 0
6. names = ['Bulbasaur','Charmander','Squirtle']
7. for i in range(0, 3):
8. p=ax5.barh(np.arange(2) + wdt, hw[i],bar\_width,
9. alpha = 0.5,
10. color = colors[i],
11. label = names[i])
12. wdt += bar\_width
13. ax5.legend(loc = 'lower right')
14. ax5.set\_xlim(0,12)
15. ax5.set\_ylim(-0.2,1.8)
16. plt.yticks(np.arange(2) + (bar\_width)\*1.5,
17. ('Height', 'Weight'))

This results in a complete figure which constitutes 5 different subplots. However, try minimizing the graph to a smaller scale and you will start noticing the overlapping of graphs and labels. To prevent this problem Matplotlib provides a niche function called tight\_layout(). Let us add this function in our module.

2. plt.tight\_layout()

We can also manipulate the background style of the figure using style.use('name') command. There are many inbuilt styles available which can be found using style.available() command. We use seaborn-dark theme to add final touch on the figure.

2. plt.style.use('seaborn-dark')



Data :;

import seaborn as sns