7-11 Working with Data-Notes

July 11, 2017

1 Modules

- Packages of functions that we can load into memory at the start of a script
- import *moduleName* as *abbreviation*

```
In [1]: import random as rn

        for x in range(10):
            print(rn.randint(0,10))
```

```
6
8
0
3
3
7
2
6
1
5
```

- Instead of loading the entire module, a specific function or functions can be loaded instead
- the function seed() just allows our supposedly "random" number generator to give the same results each run

```
In [4]: from random import randint,seed
        seed(20)
        age = []
        survived = []
        for x in range(100):
            age.append(randint(20,70))
            survived.append(randint(0,1))
```

```
In [3]: print(age)
        print(survived)
```

```
[66, 36, 40, 21, 46, 26, 40, 57, 46, 32, 60, 41, 25, 45, 33, 22, 26, 32, 64, 39, 30
[0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1
```

2 Matplotlib (pyplot) vs ggplot2

- matplotlib is the plotting library for python (and matlab)
- ggplot2 is the plotting library for R
- they have different approaches for creating a figure and are not easily translated between one another
- we will be focusing on using matplotlib

```
In [5]: import matplotlib.pyplot as plt
        #next line just lets plots appear inline; ipython terminal specific
        %matplotlib inline
```

Creating figures in matplot take the following approach:

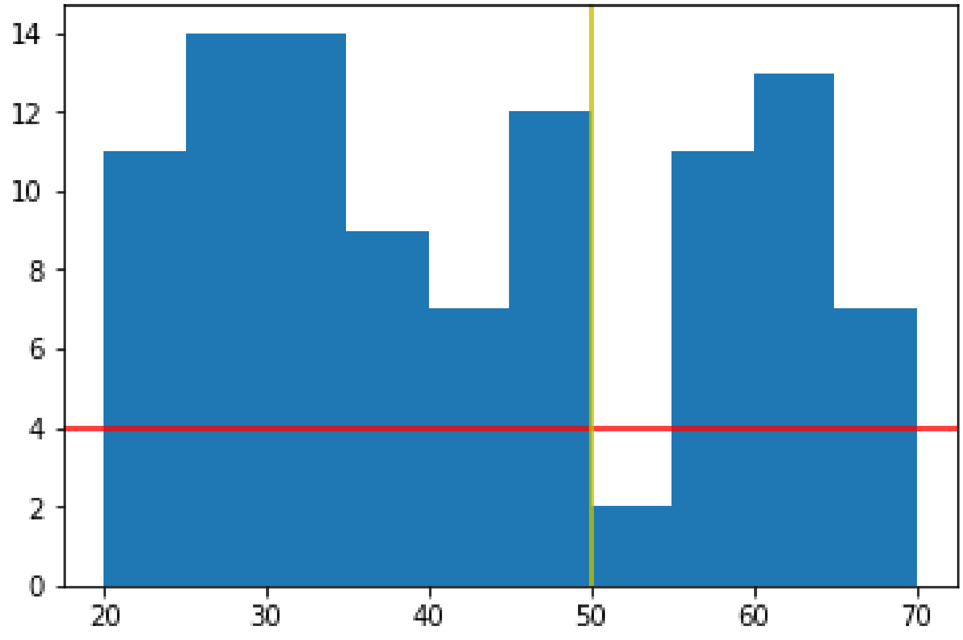
1) create your canvas (your figure)

2) add layers to your canvas (each being applied on top of the previous)

3) show your canvas

Layers can take the form of anything from full histogram graphs to single dots and lines

```
In [7]: #Create canvas
        plt.figure()

        #Add Layers
        plt.hist(age)                 #histogram graph
        plt.axhline(4,color='r')      #horizontal line
        plt.axvline(50,color='y')     #vertical line

        #Show canvas
        plt.show()
```

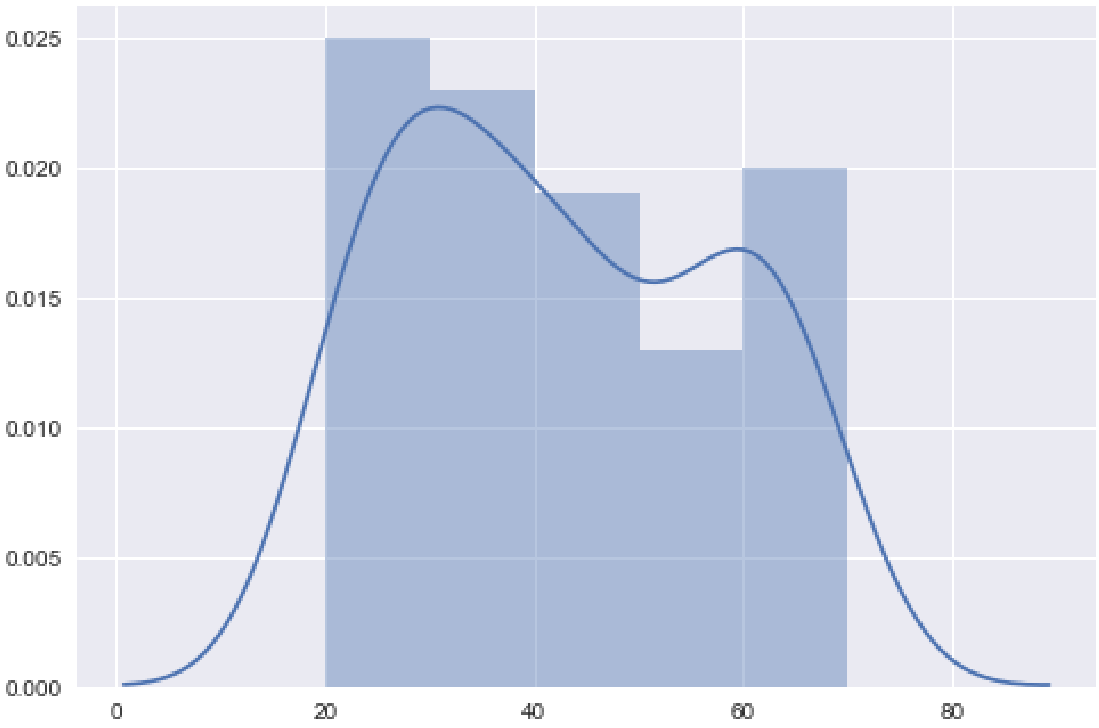


3 Seaborn Module

Matplotlib can get very confusing to use and can get quite messy, so the seaborn module was developed as a high-level (i.e. easier to interpret/use) interface to using matplotlib

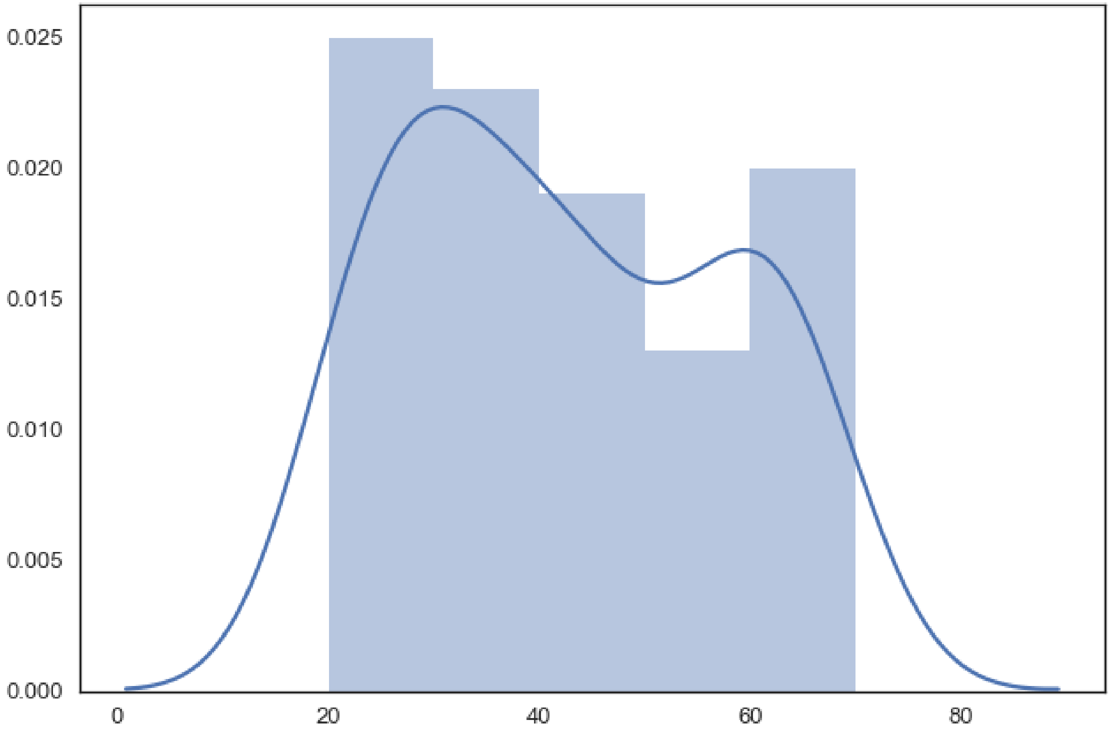
```
In [9]: import seaborn as sns
```

```
In [10]: #Create Canvas
         sns.plt.figure()

         #Layers
         sns.distplot(age)

         #Show
         sns.plt.show()
```



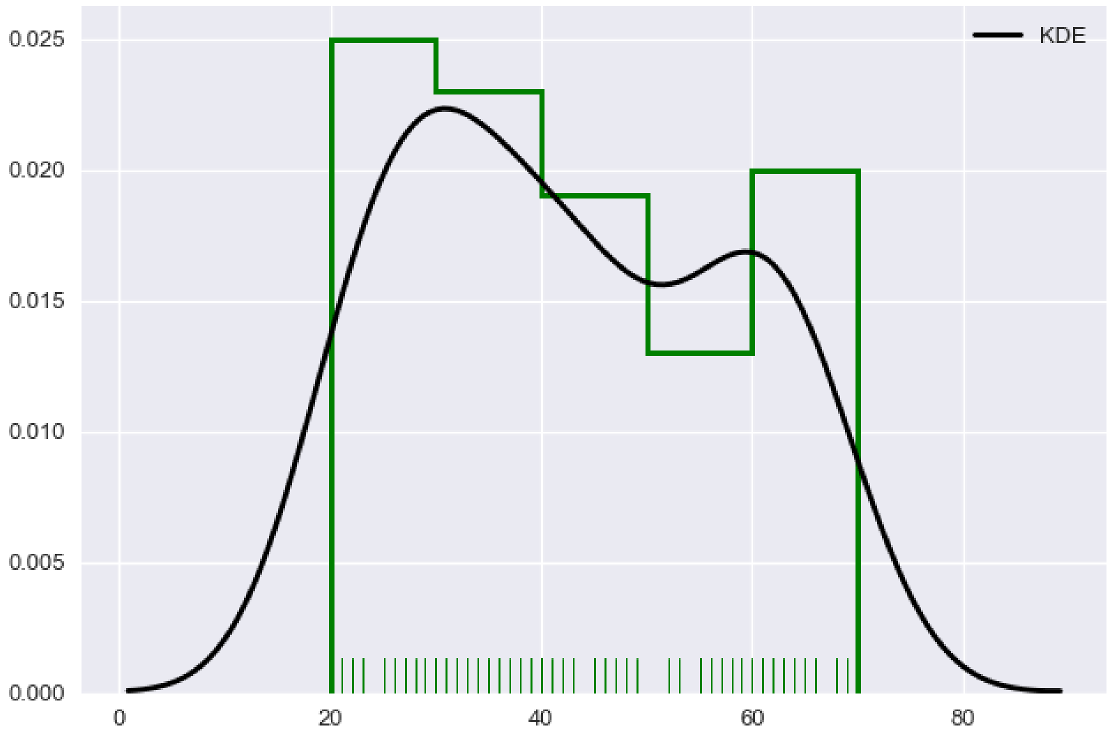
seaborn comes with many default styles

```
In [12]: sns.set_style('white')
         sns.set_context('talk')
         sns.plt.figure()
         sns.distplot(age)
         sns.plt.show()
```



seaborn also has very nice documentation that describes all the different ways you can change a plot through parameters http://seaborn.pydata.org/generated/seaborn.distplot.html

```
In [13]: sns.set_style("darkgrid")
         sns.plt.figure()
         sns.distplot(age, rug=True, rug_kws={"color": "g"}, kde_kws={"color": "k",
         sns.plt.show()
```



4 numpy

Numpy allows the use of matrix and linear algebra operations. It is essential for scientific computing and is a commonly required module for most data analysis. Most of the time you will not be directly using numpy but the modules will.

```
In [14]: import numpy as np

In [15]: np.array(age)

Out[15]: array([66, 36, 40, 21, 46, 26, 40, 57, 46, 32, 60, 41, 25, 45, 33, 22, 26,
                32, 64, 39, 30, 21, 65, 34, 23, 61, 37, 64, 58, 29, 55, 30, 45, 64,
                69, 42, 27, 53, 47, 31, 27, 20, 63, 49, 62, 27, 55, 20, 56, 25, 36,
                62, 62, 70, 32, 52, 30, 62, 39, 46, 26, 26, 35, 34, 39, 55, 28, 48,
                20, 32, 45, 38, 43, 29, 20, 23, 43, 45, 26, 56, 33, 64, 68, 65, 31,
                38, 20, 33, 59, 64, 57, 70, 48, 61, 59, 55, 45, 20, 25, 40])
```

5 Pandas and DataFrames

- Pandas introduces DataFrames to python (a commonly used data structure in R)
- DataFrames allows easy handling of multi-dimensional data
- Easy reading and writing of files

```
In [21]: import numpy as np
         import pandas as pd
```

```
In [22]: dictionary = {'age':age, 'survived':survived}
         print(dictionary)
```

```
{'age': [66, 36, 40, 21, 46, 26, 40, 57, 46, 32, 60, 41, 25, 45, 33, 22, 26, 32, 64
```

```
In [23]: #Turn a dictionary into a dataframe
         df = pd.DataFrame(dictionary)
```

```
In [24]: #Print first 5 rows
         df.head()
```

```
Out[24]:    age  survived
         0   66         0
         1   36         0
         2   40         0
         3   21         1
         4   46         0
```

5.1 Slicing, Indexing, Columns

- https://pandas.pydata.org/pandas-docs/stable/10min.html

```
In [27]: #Select column
         df['age'].head()
```

```
Out[27]: 0    66
         1    36
         2    40
         3    21
         4    46
         Name: age, dtype: int64
```

```
In [28]: #Select row
         df.loc[0]
```

```
Out[28]: age         66
         survived     0
         Name: 0, dtype: int64
```

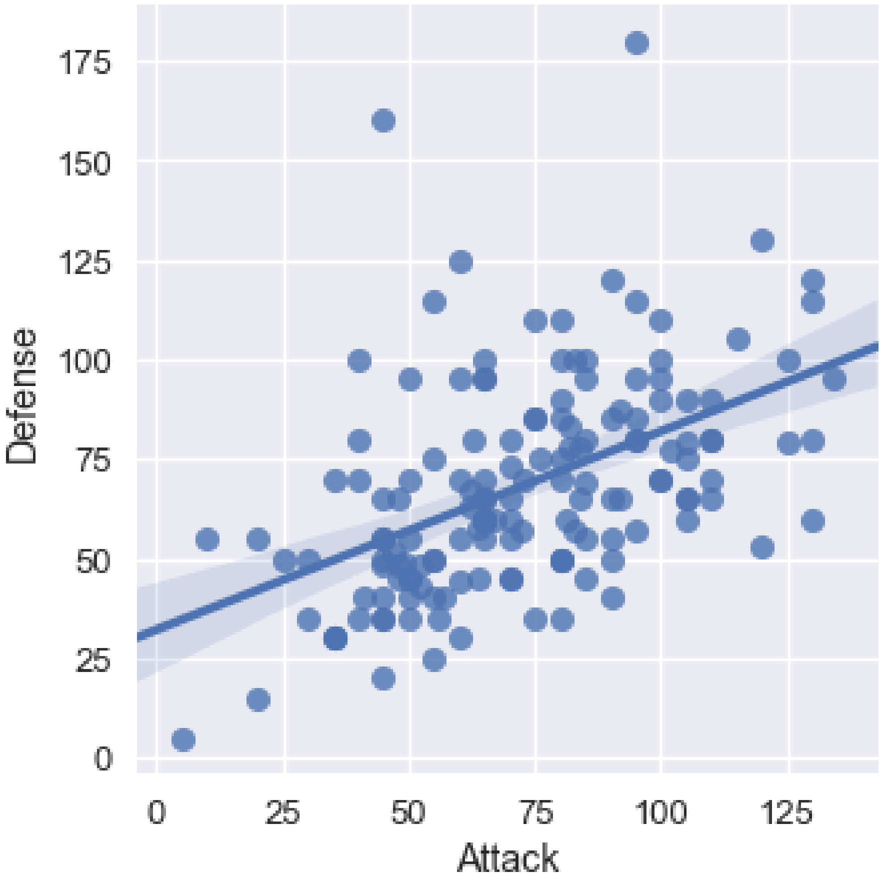
6 Plotting

Seaborn is nicely integrated with pandas and makes plotting with dataframes very intuitive

```
In [29]: #Read in entire csv file as a dataframe
         df = pd.read_csv('Pokemon.csv',index_col=0)
```

```
In [30]: df.head()

Out[30]:                 Name Type 1  Type 2  Total  HP  Attack  Defense  Sp. Atk
         Pokemon#
         1          Bulbasaur  Grass  Poison    318  45      49       49       65
         2            Ivysaur  Grass  Poison    405  60      62       63       80
         3           Venusaur  Grass  Poison    525  80      82       83      100
         4         Charmander   Fire     NaN    309  39      52       43       60
         5         Charmeleon   Fire     NaN    405  58      64       58       80

                   Sp. Def  Speed  Stage  Legendary
         Pokemon#
         1              65     45      1      False
         2              80     60      2      False
         3             100     80      3      False
         4              50     65      1      False
         5              65     80      2      False
```

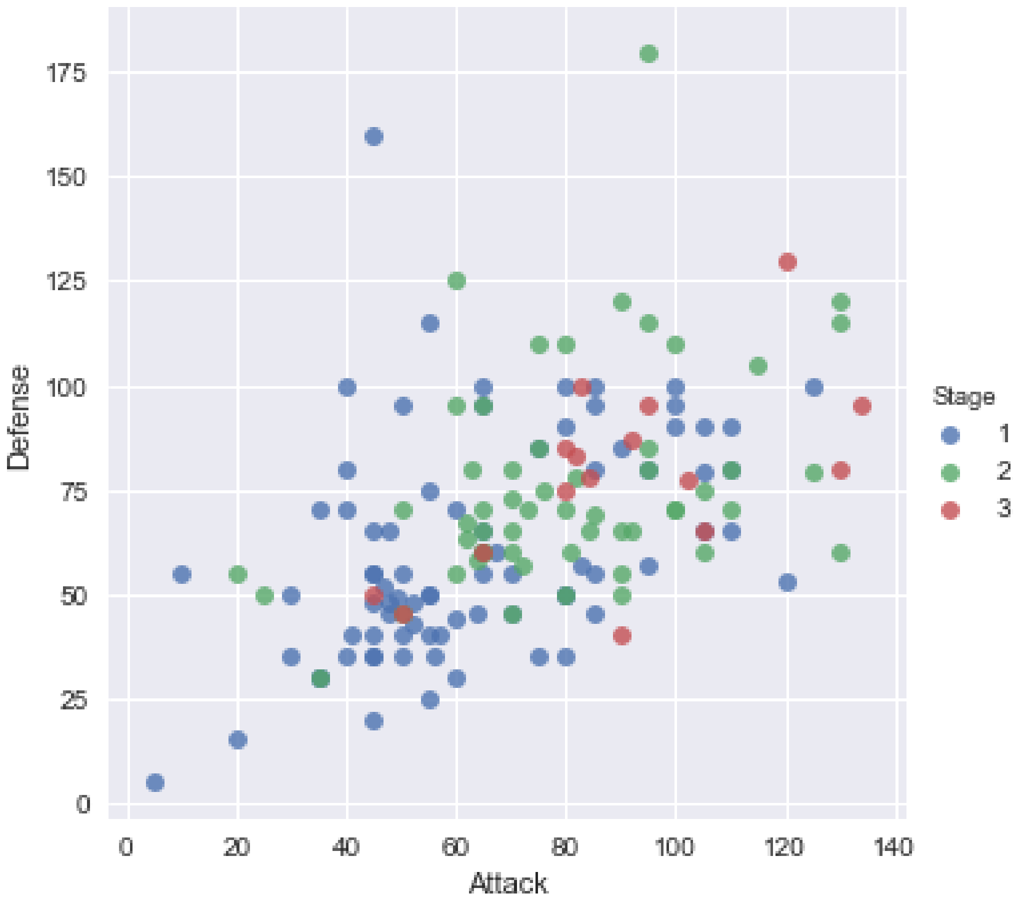
```
In [48]: sns.set_context('talk')
         sns.set_style('darkgrid')
         sns.plt.figure()
         #Select the dataframe to pull data from, then specify column names for x a
         sns.lmplot(data=df,x='Attack',y='Defense')
         sns.plt.show()

<matplotlib.figure.Figure at 0x7f6ea7d5a0b8>
```



There is no "scatterplot" in seaborn. However, you can easily use lmplot and turn off the regression line. In addition, we can tell seaborn to color our points using another column in the dataframe

```
In [49]: sns.set_context('notebook')
         sns.plt.figure()
         sns.lmplot(data=df,x='Attack',y='Defense',fit_reg=False,hue='Stage')
         sns.plt.show()
```

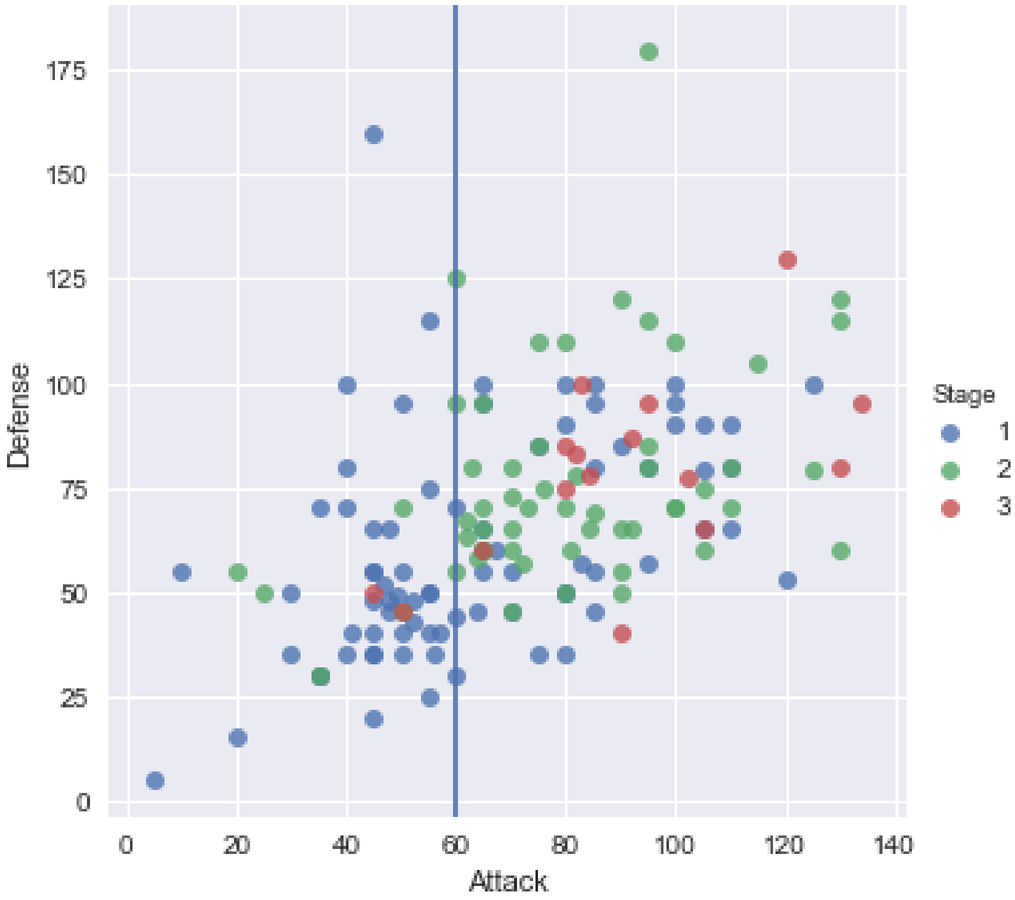
```
<matplotlib.figure.Figure at 0x7f6ea62d5470>
```



Seaborn is matplotlib at its core so we can use matplotlib commands to further customize our figure

```
In [50]: sns.set_context('notebook')
         sns.plt.figure()
         sns.lmplot(data=df,x='Attack',y='Defense',fit_reg=False,hue='Stage')
         sns.plt.axvline(60)
         sns.plt.show()
```

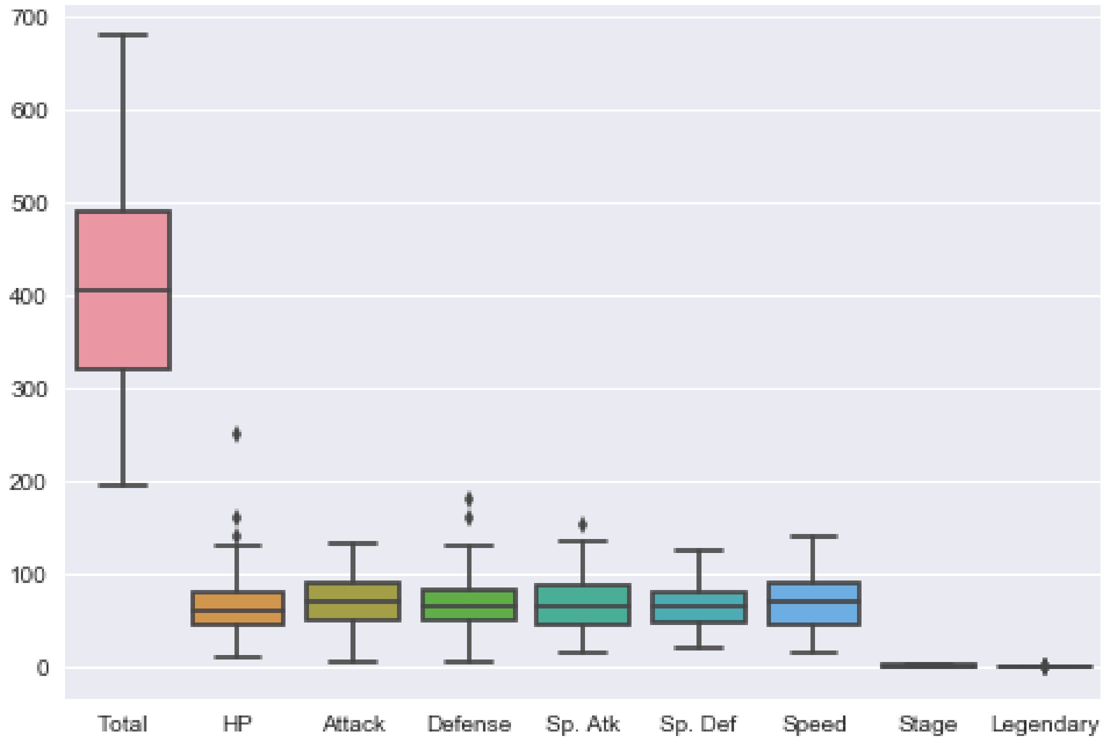
```
<matplotlib.figure.Figure at 0x7f6ea62b3940>
```



Seboarn will try to interpret what you ask.

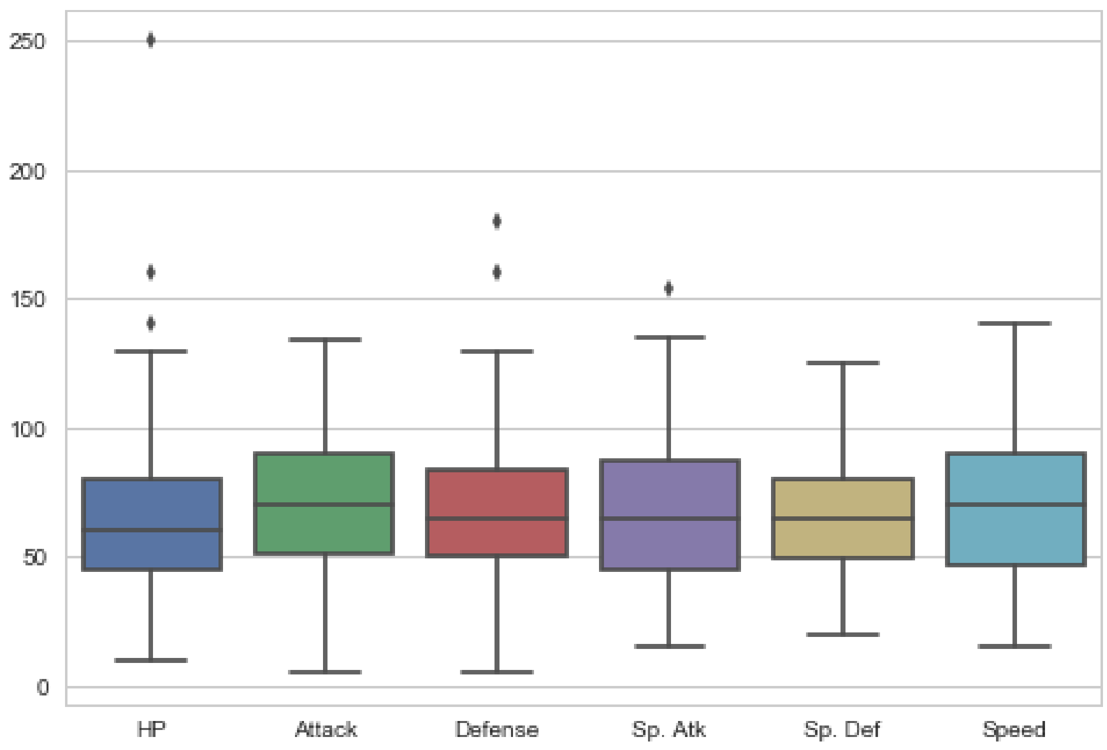
Seaborn will try to interpret what you ask. Just calling boxplot here will create a boxplot for each column in your dataframe

```
In [51]: sns.plt.figure()
         sns.boxplot(data=df)
         sns.plt.show()
```



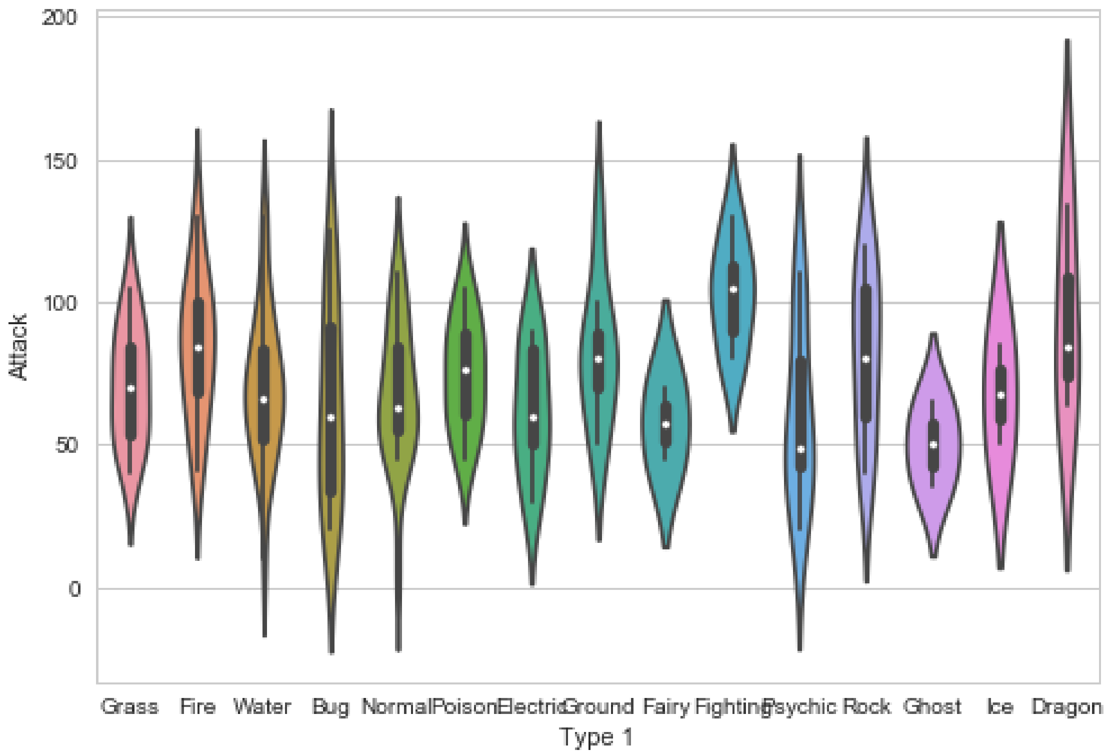
To fix this, we use pandas to create a subset dataframe from our original

```
In [52]: sns.set_style("whitegrid")
         stats_df = df.drop(['Total','Stage','Legendary'], axis=1)
         sns.boxplot(data=stats_df)

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6ea86329e8>
```



Violinplots

```
In [53]: sns.violinplot(data=df, x='Type 1', y='Attack')

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6ea60bf7b8>
```



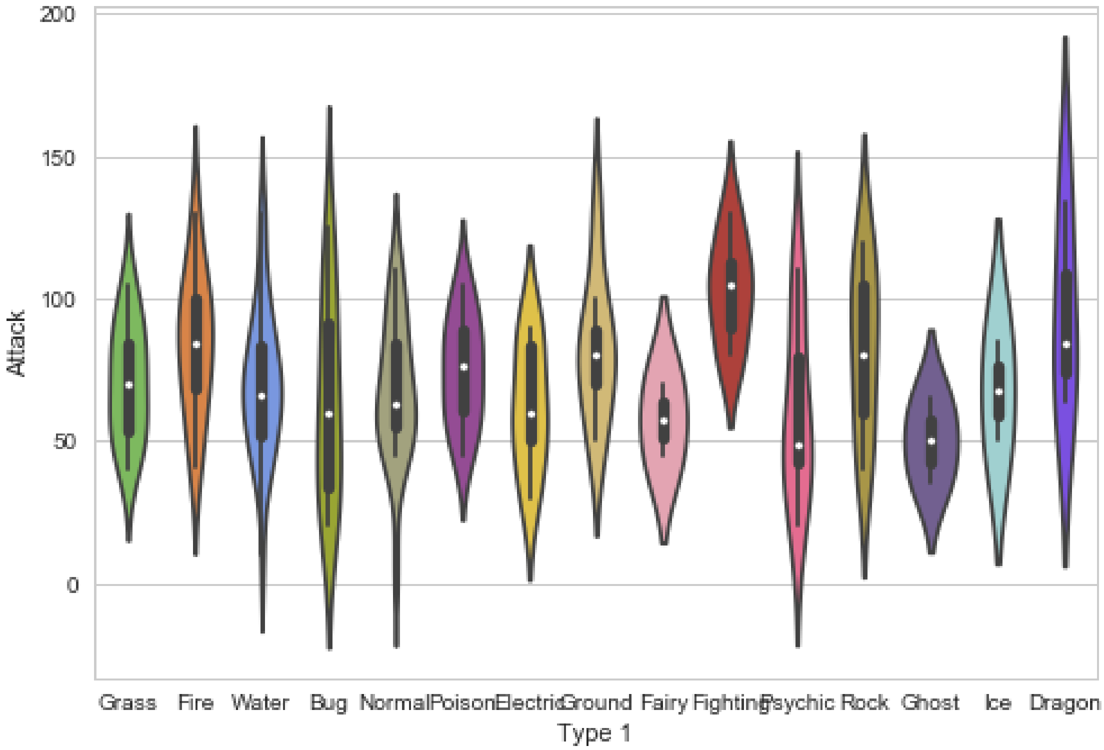
Seaborn has an extensive color options

- http://seaborn.pydata.org/tutorial/color_palettes.html

Here, we use our own manually defined colors in hexidecimal - https://color.adobe.com/

```
In [54]: pkmn_type_colors = ['#78C850',  # Grass
                              '#F08030',  # Fire
                              '#6890F0',  # Water
                              '#A8B820',  # Bug
                              '#A8A878',  # Normal
                              '#A040A0',  # Poison
                              '#F8D030',  # Electric
                              '#E0C068',  # Ground
                              '#EE99AC',  # Fairy
                              '#C03028',  # Fighting
                              '#F85888',  # Psychic
                              '#B8A038',  # Rock
                              '#705898',  # Ghost
                              '#98D8D8',  # Ice
                              '#7038F8',  # Dragon
                             ]
```

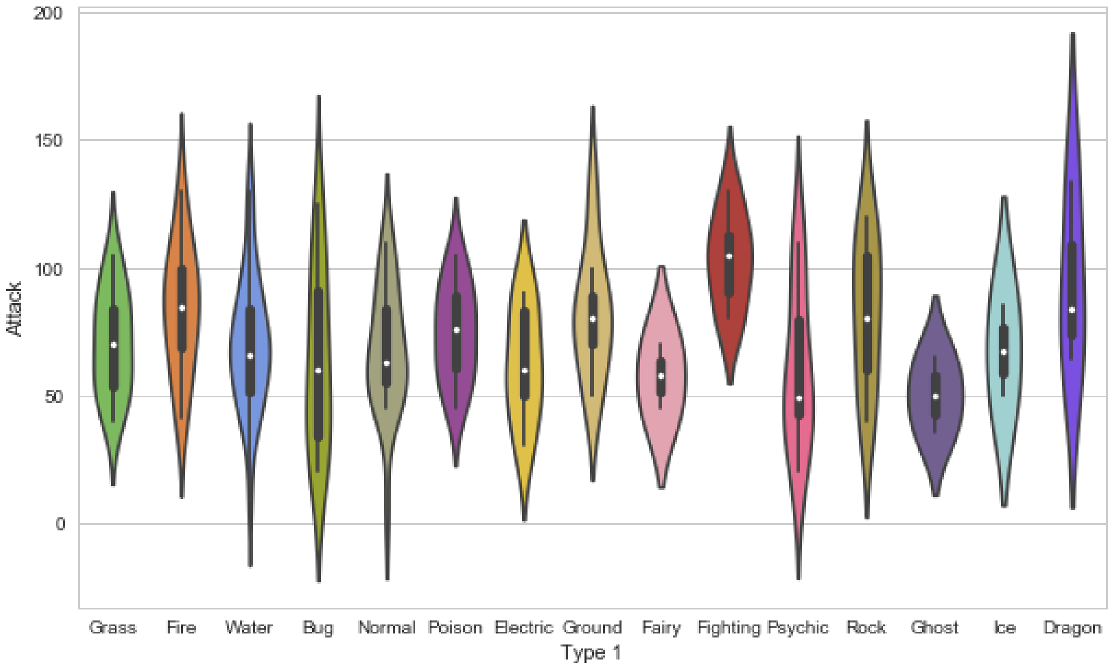
```
In [55]: sns.set_style("whitegrid")

         sns.plt.figure()
         sns.violinplot(data=df, x='Type 1', y='Attack',palette=pkmn_type_colors)
         sns.plt.show()
```



Increase the size of the figure to prevent crowding

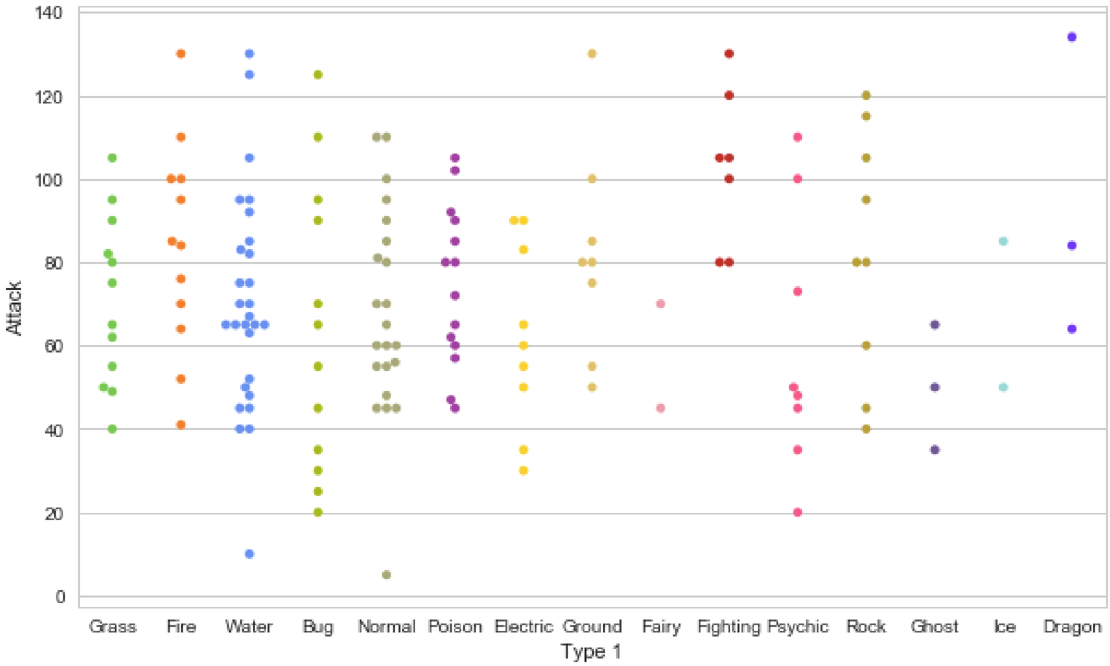
- Define the figure size when 'creating' your canvas

```
In [56]: sns.plt.figure(figsize=(10,6))
         sns.violinplot(data=df, x='Type 1', y='Attack',palette=pkmn_type_colors)
         sns.plt.show()
```



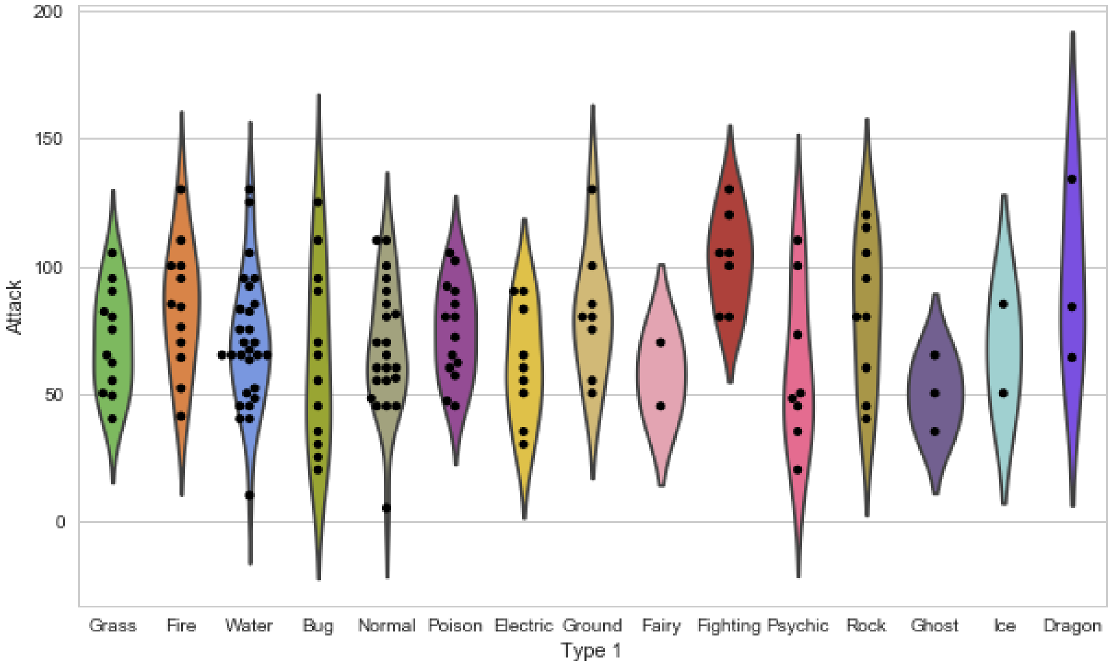
Swarmplot

```
In [57]: sns.plt.figure(figsize=(10,6))
         sns.swarmplot(data=df, x='Type 1', y='Attack', palette=pkmn_type_colors)
         sns.plt.show()
```



Seaborn plots are simple layers that can be combined

```
In [58]: sns.plt.figure(figsize=(10,6))
         sns.violinplot(data=df, x='Type 1', y='Attack',palette=pkmn_type_colors,in
         sns.swarmplot(data=df, x='Type 1', y='Attack', color='black')
         sns.plt.show()
```



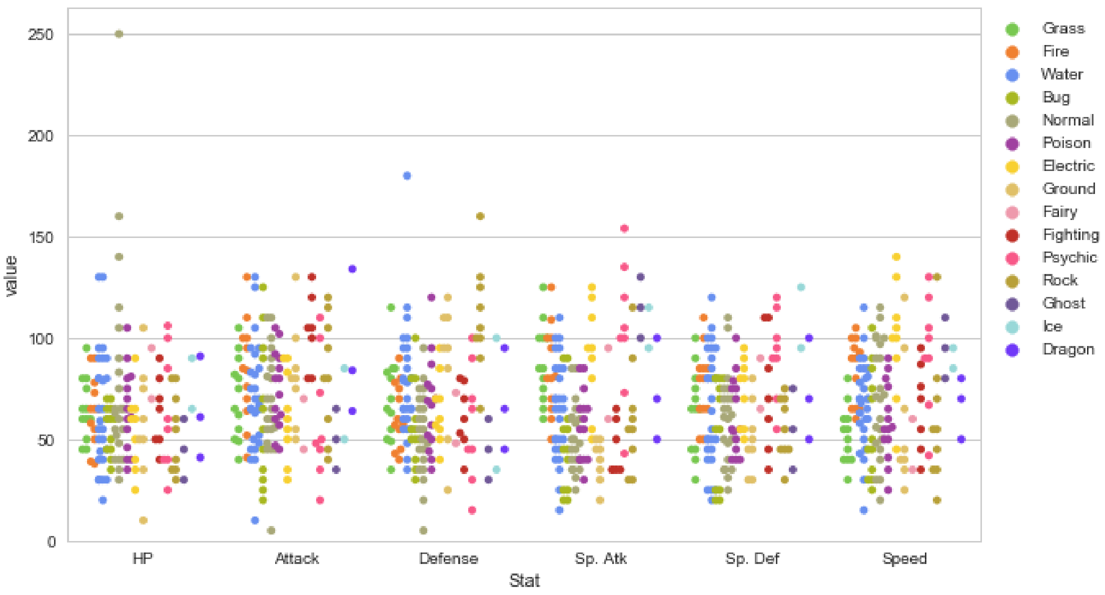
6.0.1 Advanced Plotting

```
In [59]: stats_df.head()
```

```
Out[59]:                 Name Type 1  Type 2  HP  Attack  Defense  Sp. Atk  Sp. Def
         Pokemon#
         1           Bulbasaur  Grass  Poison  45      49       49       65       65
         2             Ivysaur  Grass  Poison  60      62       63       80       80
         3            Venusaur  Grass  Poison  80      82       83      100      100
         4          Charmander   Fire     NaN  39      52       43       60       50
         5          Charmeleon   Fire     NaN  58      64       58       80       65

                   Speed
         Pokemon#
         1            45
         2            60
         3            80
```

```
4            65
5            80
```

```
In [60]: melted_df = pd.melt(stats_df,
                             id_vars=["Name", "Type 1", "Type 2"],
                             var_name="Stat")
         sns.plt.figure(figsize=(10,6))
         sns.swarmplot(data=melted_df, x='Stat', y='value', hue='Type 1',palette=pk
         sns.plt.legend(bbox_to_anchor=(1,1),loc=2)
         sns.plt.ylim(0)
         sns.plt.show()
```

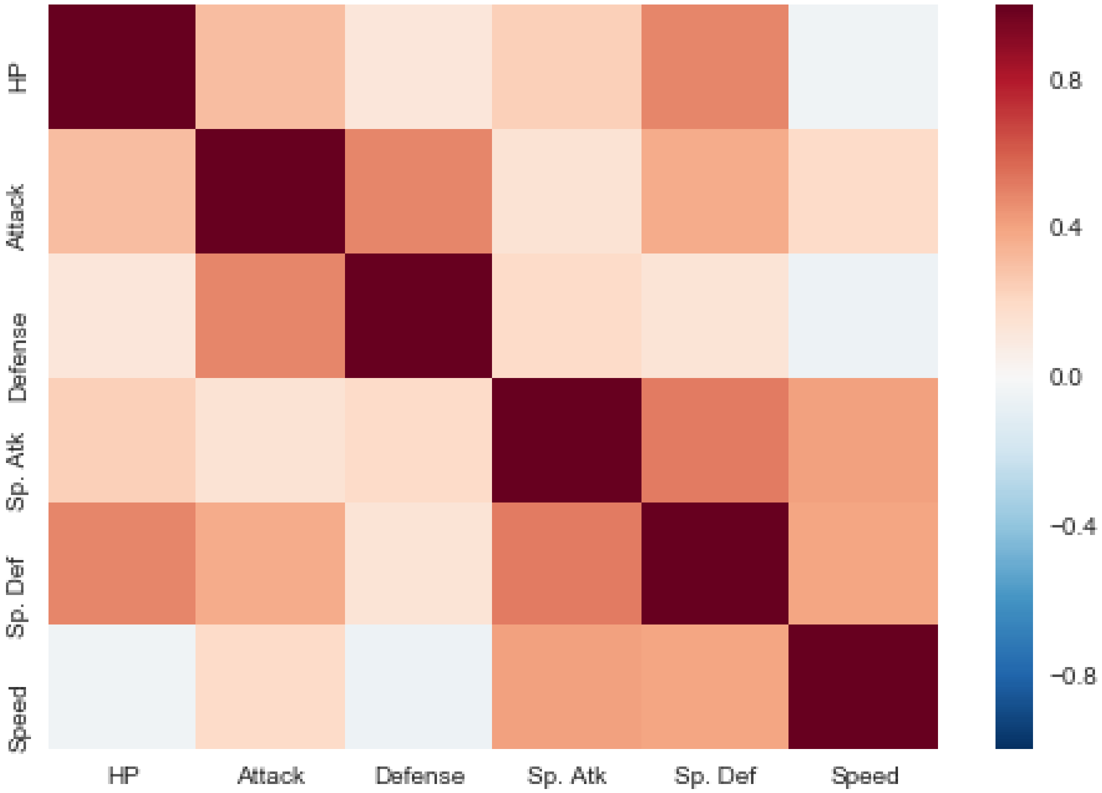


6.0.2 Plotting Showcase

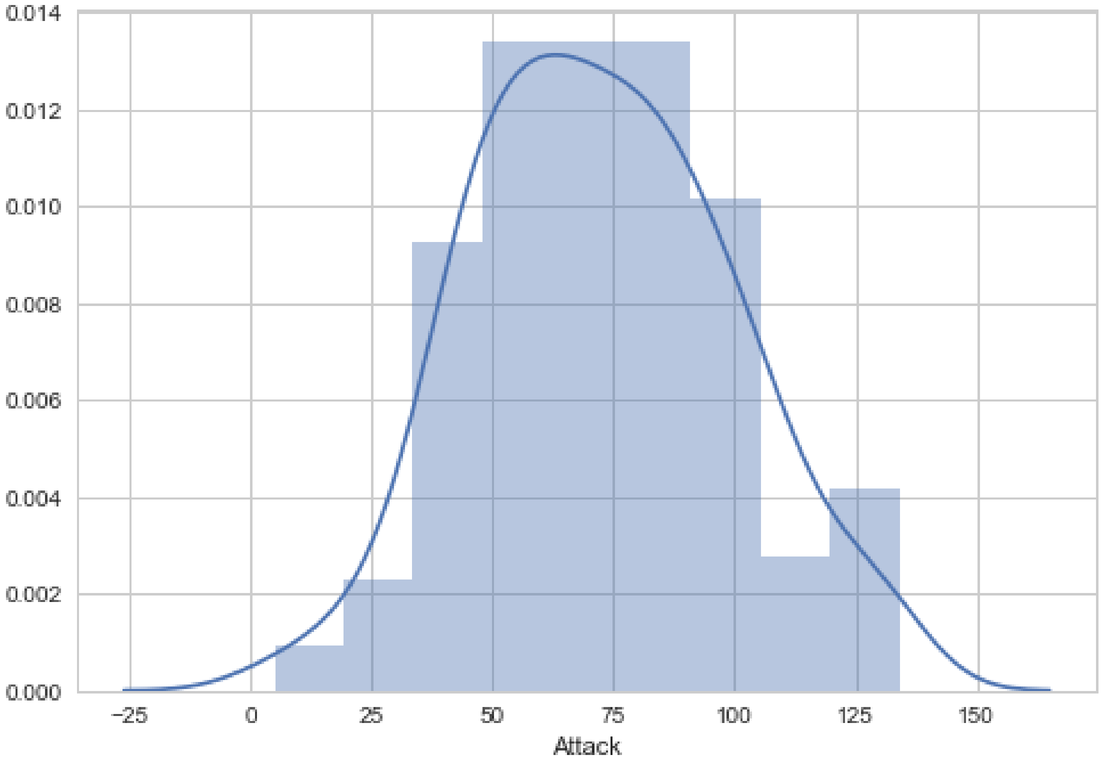
```
In [61]: corr = stats_df.corr()
         print(corr)
```

```
               HP    Attack   Defense   Sp. Atk   Sp. Def     Speed
HP       1.000000  0.306768  0.119782  0.236649  0.490978 -0.040939
Attack   0.306768  1.000000  0.491965  0.146312  0.369069  0.194701
Defense  0.119782  0.491965  1.000000  0.187569  0.139912 -0.053252
Sp. Atk  0.236649  0.146312  0.187569  1.000000  0.522907  0.411516
Sp. Def  0.490978  0.369069  0.139912  0.522907  1.000000  0.392656
Speed   -0.040939  0.194701 -0.053252  0.411516  0.392656  1.000000
```

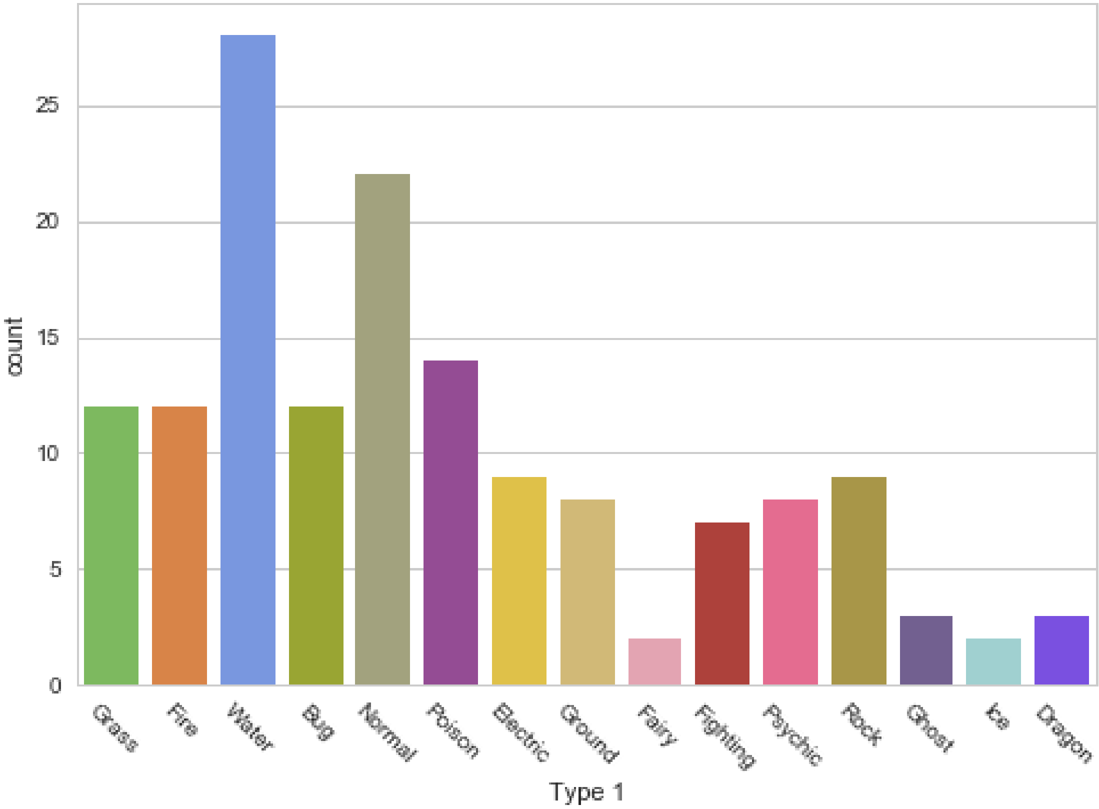
```
In [62]: sns.heatmap(corr)

Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6ea7d14550>
```



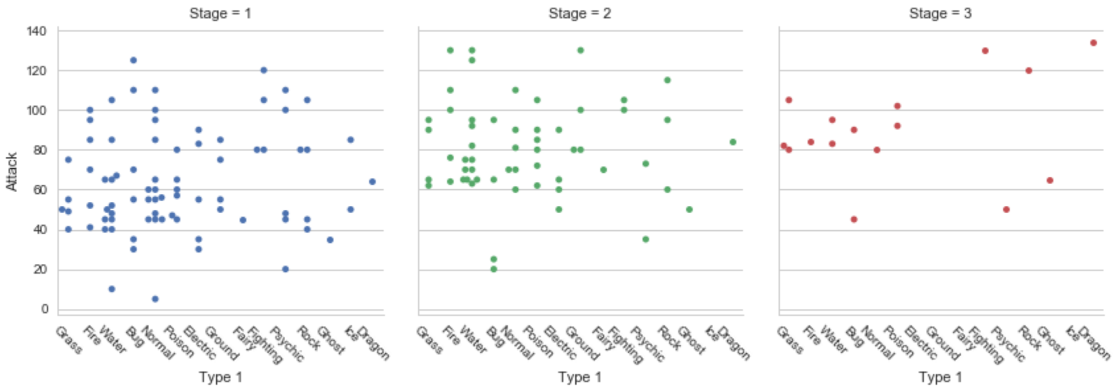
```
In [63]: sns.distplot(df['Attack'])

Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6ea5d14588>
```



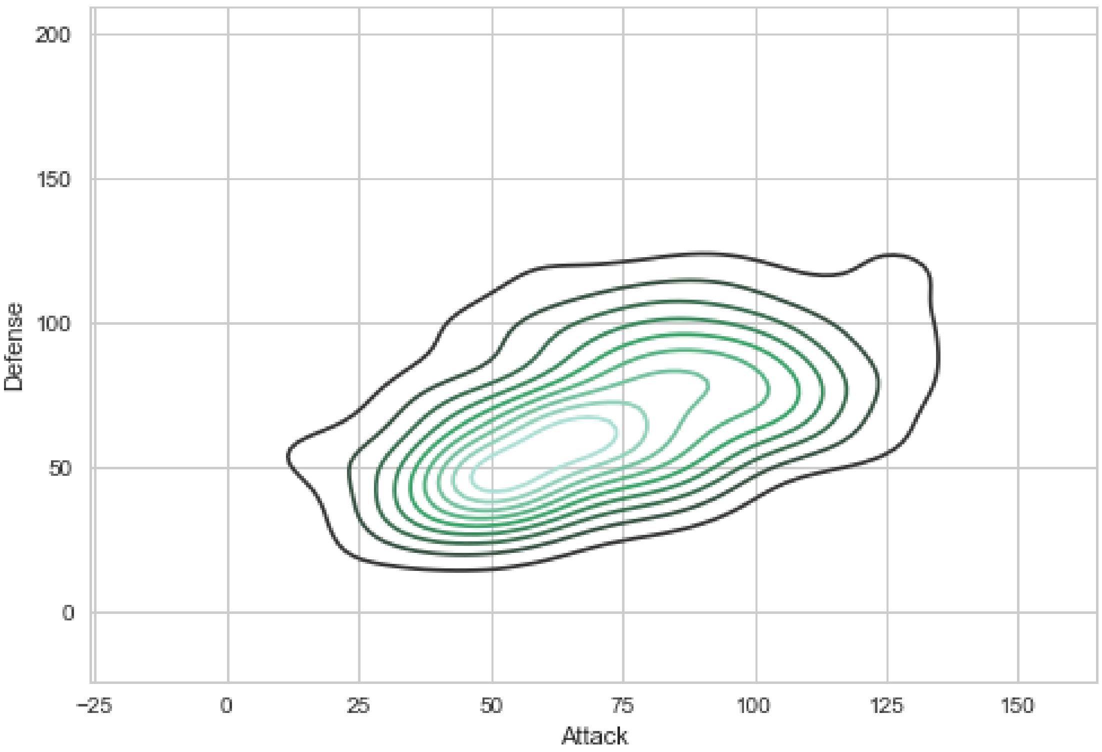
```
In [64]: sns.countplot(data=df,x='Type 1', palette=pkmn_type_colors)
         sns.plt.xticks(rotation=-45)

Out[64]: (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14]),
          <a list of 15 Text xticklabel objects>)
```



```
In [65]: sns.plt.figure(figsize=(10,6))
         g = sns.factorplot(data=df,
                            x='Type 1',
                            y='Attack',
                            hue='Stage',
                            col='Stage',
                            kind='swarm')

         g.set_xticklabels(rotation=-45)
         sns.plt.show()

<matplotlib.figure.Figure at 0x7f6ea61a9278>
```

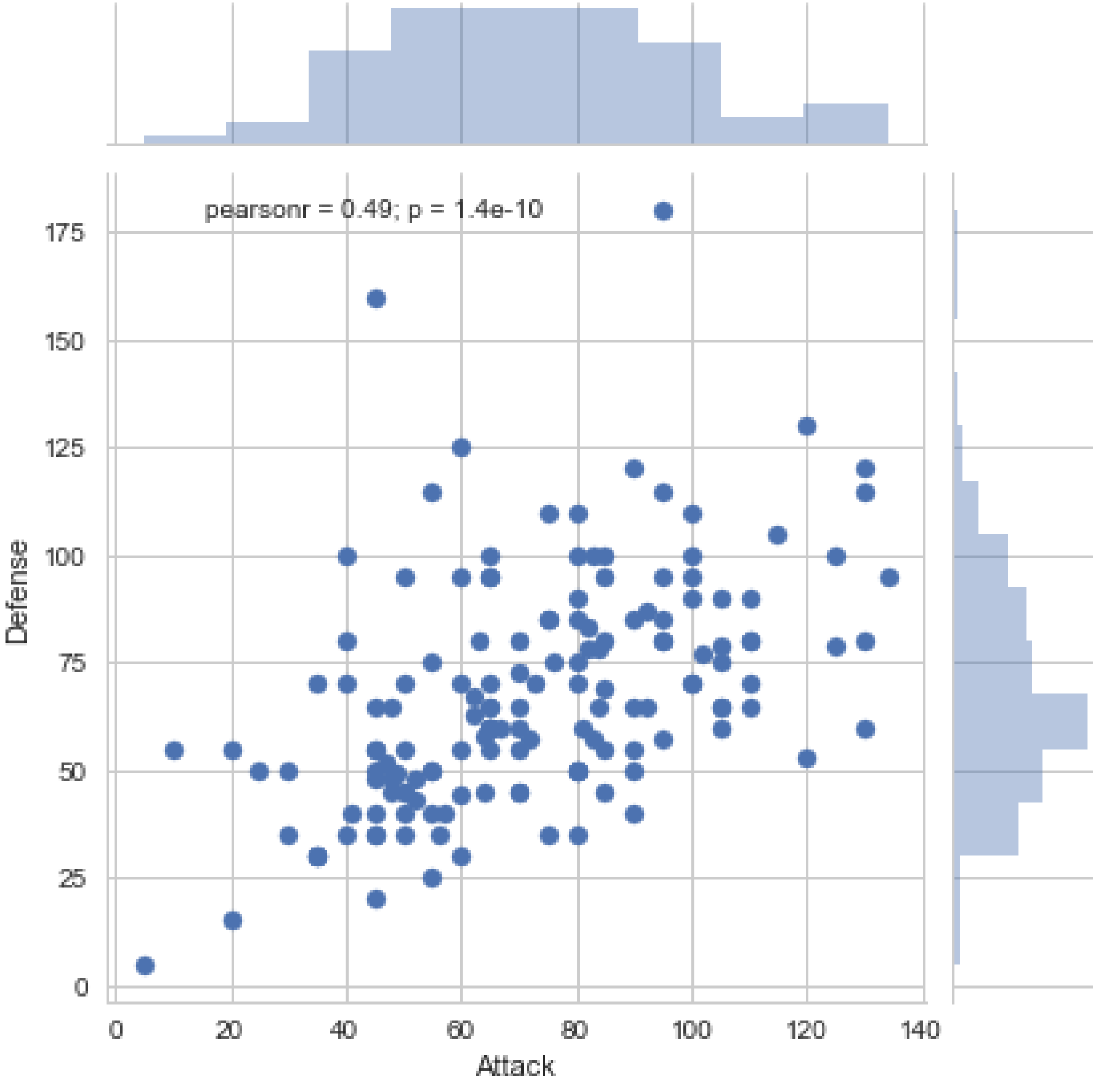


```
In [66]: sns.kdeplot(df['Attack'],df['Defense'])

Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6ea5e8e780>
```



```
In [67]: sns.plt.figure(figsize=(10,6))
         sns.jointplot(data=df,x='Attack', y='Defense')
         sns.plt.show()
```

```
<matplotlib.figure.Figure at 0x7f6ea5c61860>
```



7 Extra

Finding pokemon with highest defense

```
In [68]: df.loc[df['Defense'].argmax()]
```

```
Out[68]: Name          Cloyster
         Type 1           Water
         Type 2             Ice
         Total              525
         HP                    50
         Attack                95
         Defense              180
         Sp. Atk               85
         Sp. Def               45
         Speed                 70
         Stage                  2
         Legendary          False
         Name: 91, dtype: object
```

Finding pokemon with 'best attack and defense' (weighing attack)

```
In [46]: df[['Defense','Attack']].head()

Out[46]:           Defense  Attack
         Pokemon#
         1              49      49
         2              63      62
         3              83      82
         4              43      52
         5              58      64
```

```
In [129]: def calculateImportance(row):
              return int(row['Defense']) + int(row['Attack']*2)

          df.loc[df[['Defense','Attack']].apply(lambda row: calculateImportance(row

Out[129]: Name           Rhydon
          Type 1         Ground
          Type 2           Rock
          Total             485
          HP                105
          Attack            130
          Defense           120
          Sp. Atk            45
          Sp. Def            45
          Speed              40
          Stage               2
          Legendary       False
          Name: 112, dtype: object
```