**Seaborn**

**Importing libraries and dataset.**

Let's start by importing Pandas, which is a great library for managing relational (i.e. table-format) datasets:

Pandas

Python

|  |  |
| --- | --- |
| 1  2 | # Pandas for managing datasets  import pandas as pd |

Next, we'll import Matplotlib, which will help us customize our plots further.

* **Tip:** In Jupyter Notebook, you can also include %matplotlib inline to display your plots inside your notebook.

Matplotlib

Python

|  |  |
| --- | --- |
| 1  2  3 | # Matplotlib for additional customization  from matplotlib import pyplot as plt  %matplotlib inline |

Then, we'll import the Seaborn library, which is the star of today's show.

Seaborn

Python

|  |  |
| --- | --- |
| 1  2 | # Seaborn for plotting and styling  import seaborn as sns |

Now we're ready to import our dataset.**Tip:** we gave each of our imported libraries an **alias**. Later, we can invoke Pandas with pd, Matplotlib with plt, and Seaborn with sns.

Today, we'll be using a cool Pokémon dataset . Here's the dataset:



Dataset for this tutorial.

Once you've downloaded the CSV file, you can import it with Pandas.

* **Tip:** The argument  index\_col=0 simply means we'll treat the first column of the dataset as the ID column.

Import dataset

Python

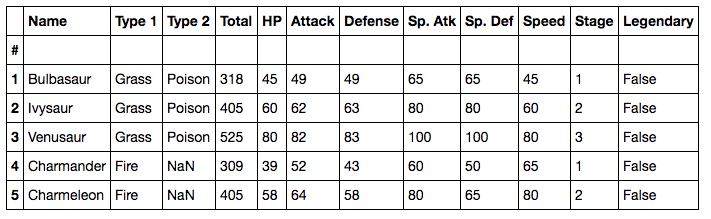
|  |  |
| --- | --- |
| 1  2 | # Read dataset  df = pd.read\_csv('Pokemon.csv', index\_col=0) |

Here's what the dataset looks like:

Example observations

Python

|  |  |
| --- | --- |
| 1  2 | # Display first 5 observations  df.head() |



As you can see, we have combat stats data for the original 151 (a.k.a best 151) Pokémon.

**Step 3: Seaborn's plotting functions.**

One of Seaborn's greatest strengths is its diversity of plotting functions. For instance, making a **scatter plot** is just one line of code using the lmplot() function.

There are two ways you can do so.

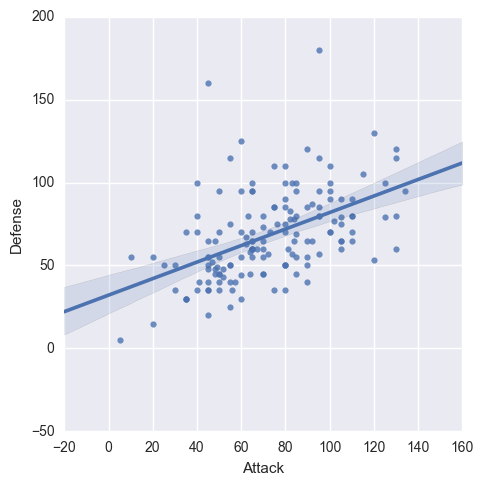
* The first way (recommended) is to pass your DataFrame to the data= argument, while passing column names to the axes arguments, x= and y=.
* The second way is to directly pass in Series of data to the axes arguments.

For example, let's compare the Attack and Defense stats for our Pokémon:

Default Scatterplot

Python

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Recommended way  sns.lmplot(x='Attack', y='Defense', data=df)    # Alternative way  # sns.lmplot(x=df.Attack, y=df.Defense) |



By the way, Seaborn doesn't have a dedicated scatter plot function, which is why you see a diagonal line. We actually used Seaborn's function for fitting and plotting a **regression line**.

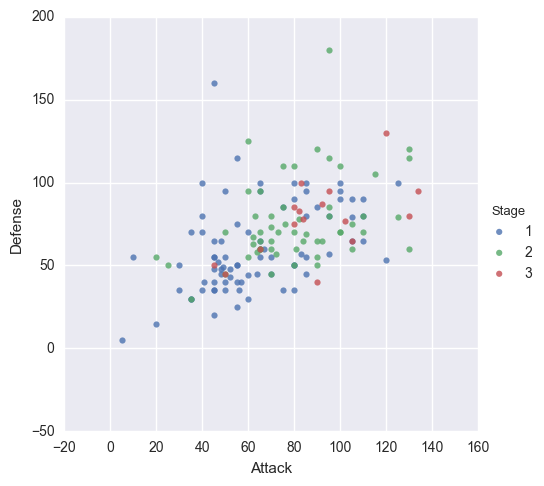
Thankfully, each plotting function has several useful options that you can set. Here's how we can tweak the lmplot():

* First, we'll set fit\_reg=False to remove the regression line, since we only want a scatter plot.
* Then, we'll set hue='Stage' to color our points by the Pokémon's evolution stage. This **hue**argument is very useful because it allows you to express a third dimension of information using color.

Scatterplot parameters

Python

|  |  |
| --- | --- |
| 1  2  3  4 | # Scatterplot arguments  sns.lmplot(x='Attack', y='Defense', data=df,             fit\_reg=False, # No regression line             hue='Stage')   # Color by evolution stage |



Looking better, but we can improve this scatter plot further. For example, all of our Pokémon have positive Attack and Defense values, yet our **axes limits** fall below zero. Let's see how we can fix that...

**Step 4: Customizing with Matplotlib.**

Remember, Seaborn is a high-level interface to Matplotlib. From our experience, Seaborn will get you *most* of the way there, but you'll sometimes need to bring in Matplotlib.

Setting your axes limits is one of those times, but the process is pretty simple:

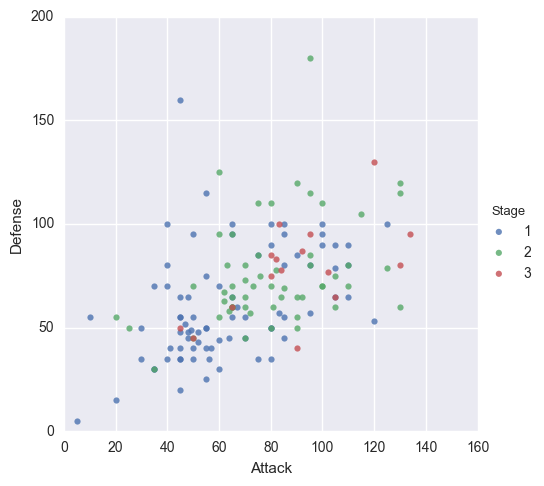
1. First, invoke your Seaborn plotting function as normal.
2. Then, invoke Matplotlib's customization functions. In this case, we'll use its ylim() andxlim() functions.

Here's our new scatter plot with sensible axes limits:

Customizing with Matplotlib

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # Plot using Seaborn  sns.lmplot(x='Attack', y='Defense', data=df,             fit\_reg=False,             hue='Stage')    # Tweak using Matplotlib  plt.ylim(0, None)  plt.xlim(0, None) |



For more information on Matplotlib's customization functions, check out its [documentation](https://matplotlib.org/users/pyplot_tutorial.html).

**Step 5: The role of Pandas.**

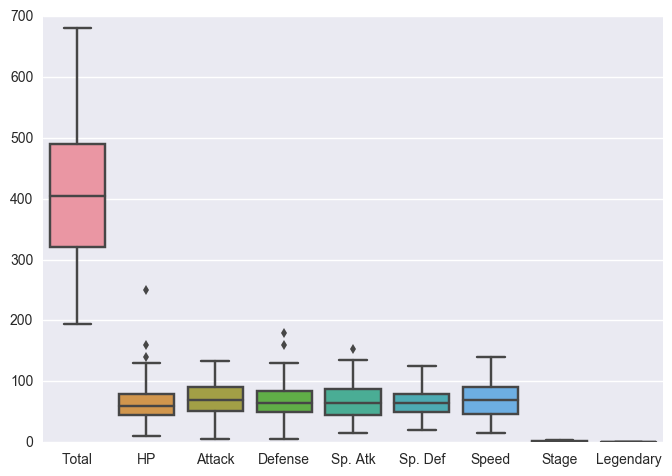
Even though this is a Seaborn tutorial, Pandas actually plays a very important role. You see, Seaborn's plotting functions benefit from a base DataFrame that's reasonably formatted.

For example, let's say we wanted to make a **box plot** for our Pokémon's combat stats:

Default boxplot

Python

|  |  |
| --- | --- |
| 1  2 | # Boxplot  sns.boxplot(data=df) |



Well, that's a reasonable start, but there are some columns we'd probably like to remove:

* We can remove the Total since we have individual stats.
* We can remove the Stage and Legendary columns because they aren't combat stats.

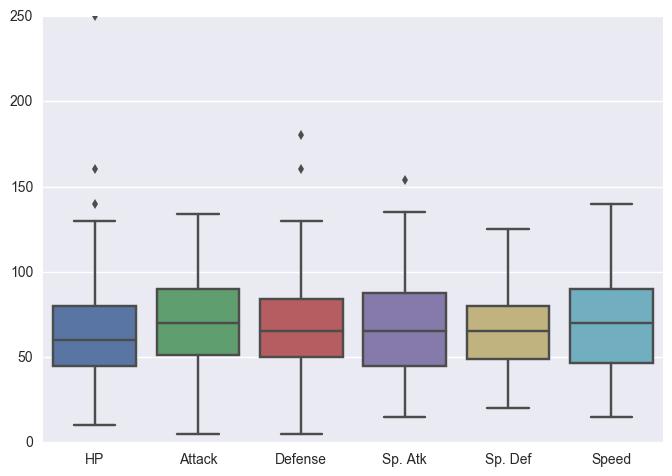
In turns out that this isn't easy to do within Seaborn alone. Instead, it's much simpler to **pre-format** your DataFrame.

Let's create a new DataFrame called stats\_df that only keeps the stats columns:

Pre-format DataFrame

Python

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Pre-format DataFrame  stats\_df = df.drop(['Total', 'Stage', 'Legendary'], axis=1)    # New boxplot using stats\_df  sns.boxplot(data=stats\_df) |



It's outside the scope of this tutorial to dive into Pandas, but here's a handy [cheat sheet](http://pandas.pydata.org/pandas-docs/stable/10min.html).

**Step 6: Seaborn themes.**

Another advantage of Seaborn is that it comes with decent style themes right out of the box. The default theme is called *'darkgrid'*.

Next, we'll change the theme to *'whitegrid'* while making a **violin plot**.

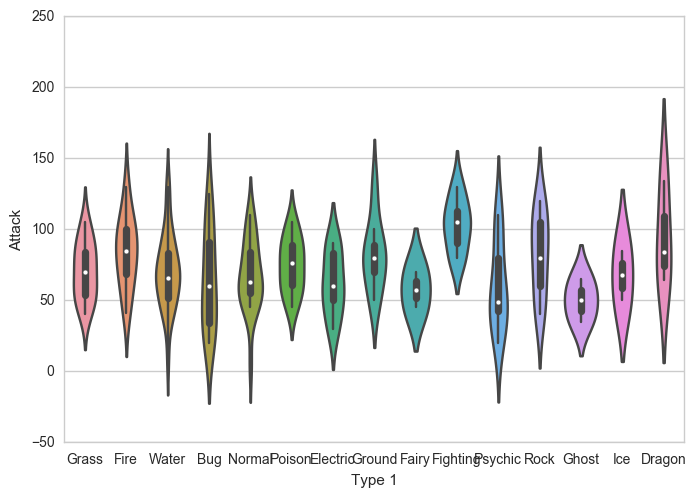
* Violin plots are useful alternatives to box plots.
* They show the distribution (through the thickness of the violin) instead of only the summary statistics.

For example, we can visualize the distribution of Attack by Pokémon's primary type:

Set theme, then plot violin plot

Python

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Set theme  sns.set\_style('whitegrid')    # Violin plot  sns.violinplot(x='Type 1', y='Attack', data=df) |



As you can see, Dragon types tend to have higher Attack stats than Ghost types, but they also have greater variance.

Now, Pokémon fans might find something quite jarring about that plot: *The colors are nonsensical.* Why is the Grass type colored pink or the Water type colored orange? We must fix this!

**Step 7: Color palettes.**

Fortunately, Seaborn allows us to set custom color palettes. We can simply create an ordered **Python list** of color hex values.

Let's use [Bulbapedia](http://bulbapedia.bulbagarden.net/wiki/Category:Type_color_templates" \t "_blank) to help us create a new color palette:

Pokemon color palette

Python

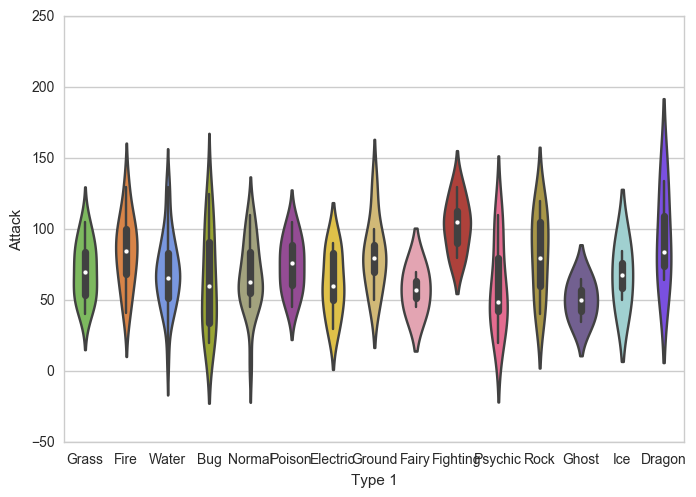
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | pkmn\_type\_colors = ['#78C850',  # Grass                      '#F08030',  # Fire                      '#6890F0',  # Water                      '#A8B820',  # Bug                      '#A8A878',  # Normal                      '#A040A0',  # Poison                      '#F8D030',  # Electric                      '#E0C068',  # Ground                      '#EE99AC',  # Fairy                      '#C03028',  # Fighting                      '#F85888',  # Psychic                      '#B8A038',  # Rock                      '#705898',  # Ghost                      '#98D8D8',  # Ice                      '#7038F8',  # Dragon                     ] |

Wonderful. Now we can simply use the palette= argument to recolor our chart.

Custom color palette

Python

|  |  |
| --- | --- |
| 1  2  3 | # Violin plot with Pokemon color palette  sns.violinplot(x='Type 1', y='Attack', data=df,                 palette=pkmn\_type\_colors) # Set color palette |



Much better!

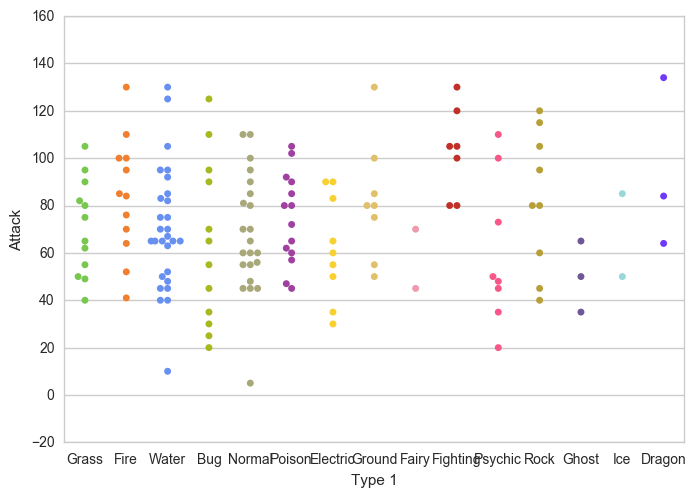
Violin plots are great for visualizing distributions. However, since we only have 151 Pokémon in our dataset, we may want to simply display each point.

That's where the **swarm plot** comes in. This visualization will show each point, while "stacking" those with similar values:

Swarm plot

Python

|  |  |
| --- | --- |
| 1  2  3 | # Swarm plot with Pokemon color palette  sns.swarmplot(x='Type 1', y='Attack', data=df,                palette=pkmn\_type\_colors) |



That's handy, but can't we combine our swarm plot and the violin plot? After all, they display similar information, right?

**Step 8: Overlaying plots.**

The answer is yes.

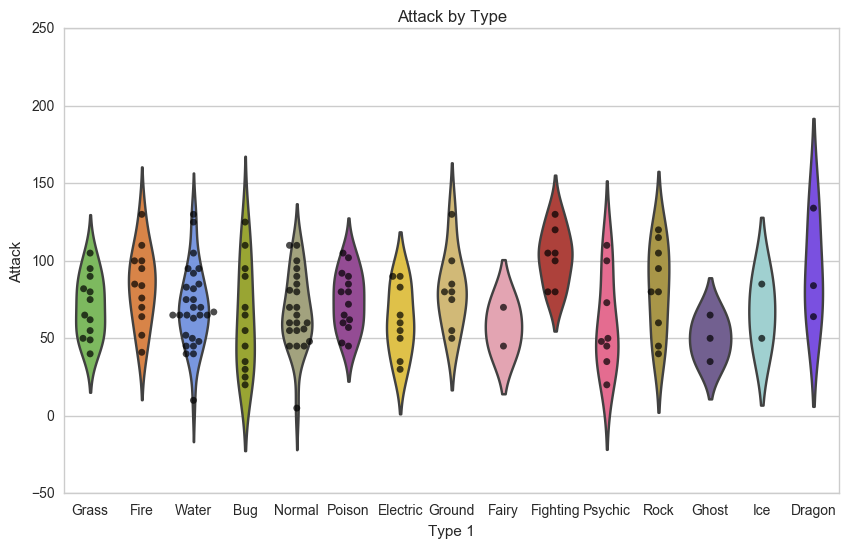
It's pretty straightforward to overlay plots using Seaborn, and it works the same way as with Matplotlib. Here's what we'll do:

1. First, we'll make our figure larger using Matplotlib.
2. Then, we'll plot the violin plot. However, we'll set inner=None to remove the bars inside the violins.
3. Next, we'll plot the swarm plot. This time, we'll make the points black so they pop out more.
4. Finally, we'll set a title using Matplotlib.

Overlaying swarm and violin plots

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | # Set figure size with matplotlib  plt.figure(figsize=(10,6))    # Create plot  sns.violinplot(x='Type 1',                 y='Attack',                 data=df,                 inner=None, # Remove the bars inside the violins                 palette=pkmn\_type\_colors)    sns.swarmplot(x='Type 1',                y='Attack',                data=df,                color='k', # Make points black                alpha=0.7) # and slightly transparent    # Set title with matplotlib  plt.title('Attack by Type') |



Awesome, now we have a pretty chart that tells us how Attack values are distributed across different Pokémon types. But what it we want to see all of the other stats as well?

**Step 9: Putting it all together.**

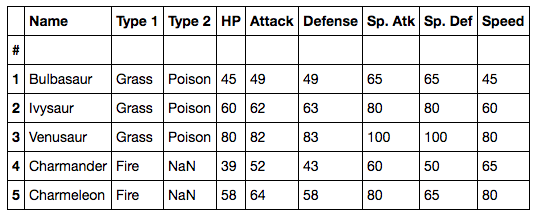
Well, we could certainly repeat that chart for each stat. But we can also combine the information into one chart... we just have to do some **data wrangling** with Pandas beforehand.

First, here's a reminder of our data format:

First 5 rows of stats\_df

Python

|  |  |
| --- | --- |
| 1 | stats\_df.head() |



As you can see, all of our stats are in separate columns. Instead, we want to "melt" them into one column.

To do so, we'll use Pandas's melt() function. It takes 3 arguments:

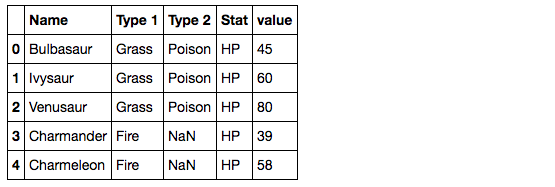
* First, the DataFrame to melt.
* Second, ID variables to keep (Pandas will melt all of the other ones).
* Finally, a name for the new, melted variable.

Here's the output:

Melt DataFrame

Python

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Melt DataFrame  melted\_df = pd.melt(stats\_df,                      id\_vars=["Name", "Type 1", "Type 2"], # Variables to keep                      var\_name="Stat") # Name of melted variable  melted\_df.head() |



All 6 of the stat columns have been "melted" into one, and the new Stat column indicates the original stat (HP, Attack, Defense, Sp. Attack, Sp. Defense, or Speed). For example, it's hard to see here, but Bulbasaur now has 6 rows of data.

In fact, if you print the shape of these two DataFrames...

Shape comparison

Python

|  |  |
| --- | --- |
| 1  2  3  4 | print( stats\_df.shape )  print( melted\_df.shape )  # (151, 9)  # (906, 5) |

...you'll find that melted\_df has 6 times the number of rows as stats\_df.

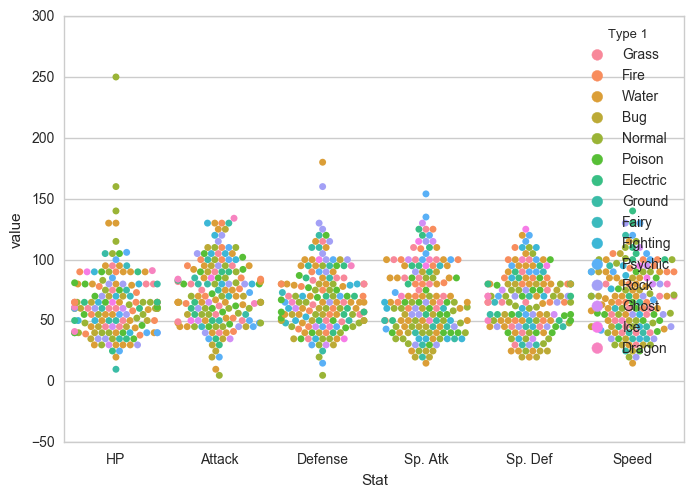
Now we can make a swarm plot with melted\_df.

* But this time, we're going to set x='Stat' and y='value' so our swarms are separated by stat.
* Then, we'll set hue='Type 1' to color our points by the Pokémon type.

Swarmplot with melted\_df

Python

|  |  |
| --- | --- |
| 1  2  3 | # Swarmplot with melted\_df  sns.swarmplot(x='Stat', y='value', data=melted\_df,                hue='Type 1') |



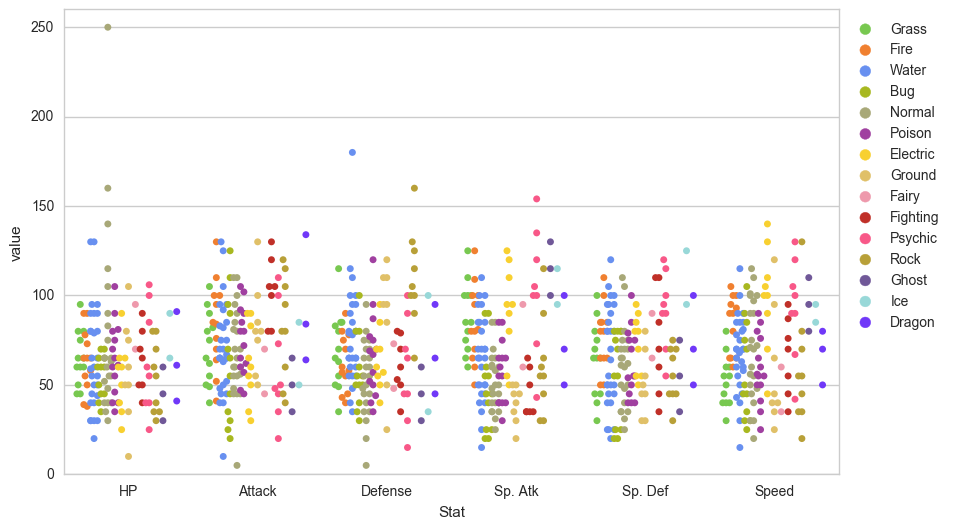
Finally, let's make a few final tweaks for a more readable chart:

1. Enlarge the plot.
2. Separate points by hue using the argument split=True .
3. Use our custom Pokemon color palette.
4. Adjust the y-axis limits to end at 0.
5. Place the legend to the right.

Customizations

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # 1. Enlarge the plot  plt.figure(figsize=(10,6))    sns.swarmplot(x='Stat',                y='value',                data=melted\_df,                hue='Type 1',                split=True, # 2. Separate points by hue                palette=pkmn\_type\_colors) # 3. Use Pokemon palette    # 4. Adjust the y-axis  plt.ylim(0, 260)    # 5. Place legend to the right  plt.legend(bbox\_to\_anchor=(1, 1), loc=2) |



There we go!

**Step 10: Pokédex (mini-gallery).**

We're going to conclude this tutorial with a few quick-fire data visualizations, just to give you a sense of what's possible with Seaborn.

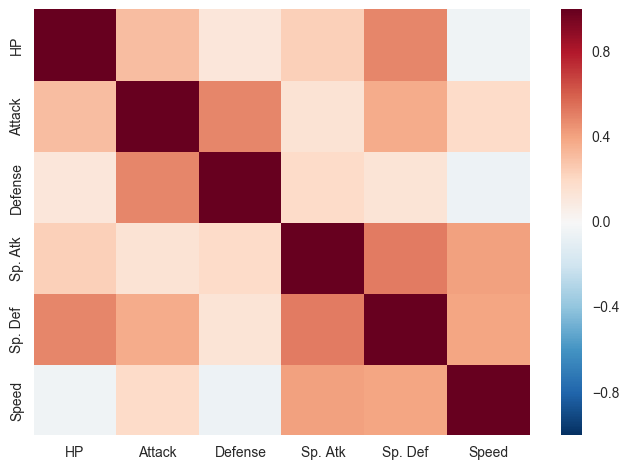
**10.1 - Heatmap**

Heatmaps help you visualize matrix-like data.

Heatmap

Python

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Calculate correlations  corr = stats\_df.corr()    # Heatmap  sns.heatmap(corr) |



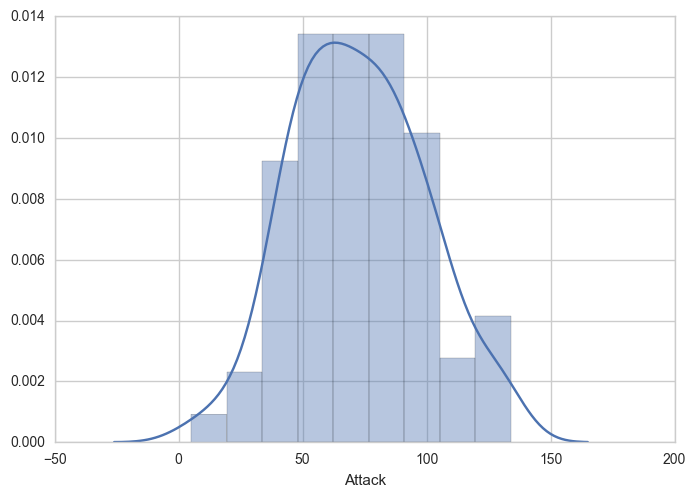
**10.2 - Histogram**

Histograms allow you to plot the distributions of numeric variables.

Histogram

Python

|  |  |
| --- | --- |
| 1  2 | # Distribution Plot (a.k.a. Histogram)  sns.distplot(df.Attack) |



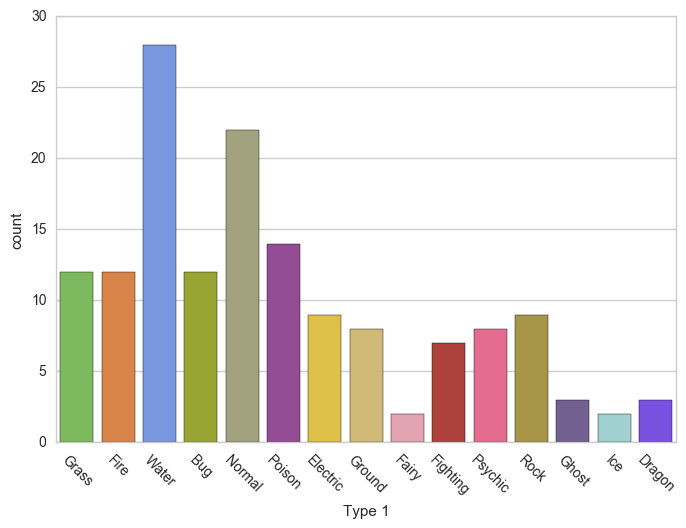
**10.3 - Bar Plot**

Bar plots help you visualize the distributions of categorical variables.

Bar Plot

Python

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Count Plot (a.k.a. Bar Plot)  sns.countplot(x='Type 1', data=df, palette=pkmn\_type\_colors)    # Rotate x-labels  plt.xticks(rotation=-45) |



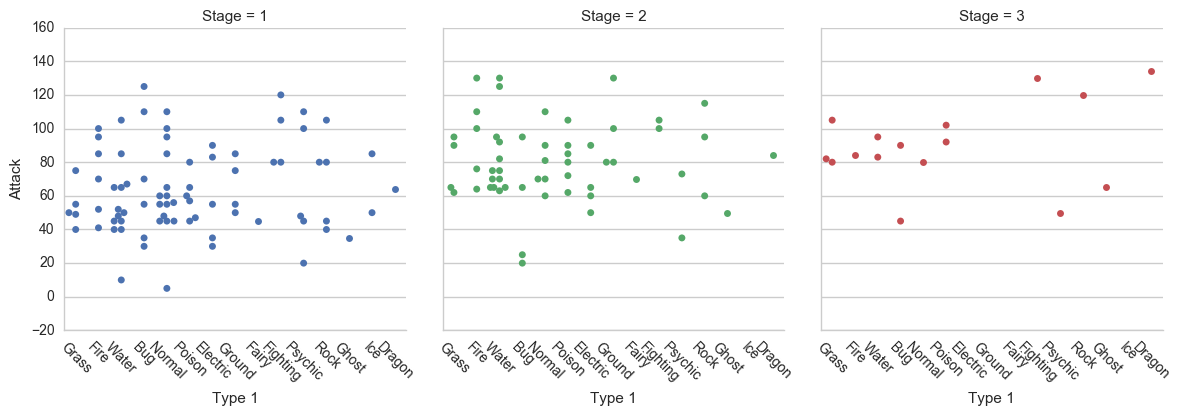
**10.4 - Factor Plot**

Factor plots make it easy to separate plots by categorical classes.

Factor Plot

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # Factor Plot  g = sns.factorplot(x='Type 1',                     y='Attack',                     data=df,                     hue='Stage',  # Color by stage                     col='Stage',  # Separate by stage                     kind='swarm') # Swarmplot    # Rotate x-axis labels  g.set\_xticklabels(rotation=-45)    # Doesn't work because only rotates last plot  # plt.xticks(rotation=-45) |



**10.5 - Density Plot**

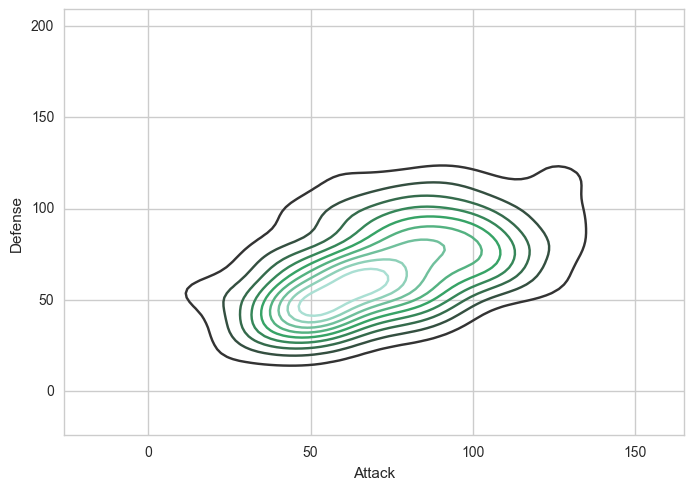
Density plots display the distribution between two variables.

* **Tip:**Consider overlaying this with a scatter plot.

Density Plot

Python

|  |  |
| --- | --- |
| 1  2 | # Density Plot  sns.kdeplot(df.Attack, df.Defense) |



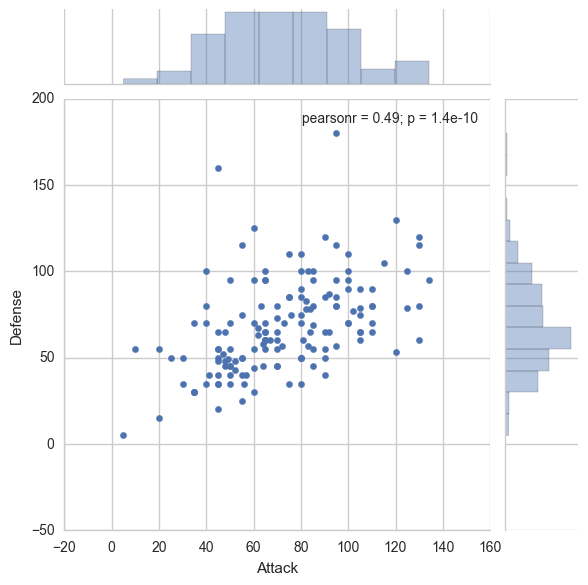
**10.6 - Joint Distribution Plot**

Joint distribution plots combine information from scatter plots and histograms to give you detailed information for bi-variate distributions.

Joint Distribution Plot

Python

|  |  |
| --- | --- |
| 1  2 | # Joint Distribution Plot  sns.jointplot(x='Attack', y='Defense', data=df) |



Congratulations... you've made it to the end of this Python Seaborn.