Day9_Pandas_End

August 5, 2021

Day 9: Data Analysis and Visualization

When we plot we need more libraries. You learned about matplot on Day 5, today we are going to build on what you learned and plot directly from pandas and I'm going to introduce seaborn.

Goals for the day:

- Practice making a pivot table
- More understanding of groupby
- Introduction to seaborn
- Introduction to graph automation

Functions Learned:

- Create a heatmap: `sns.heatmap()`
- Create a pairplot: `sns.pairplot()`
- Create a swarmplot: `sns.swarmplot()`
- Create a violinplot: `sns.violinplot`

I will not be able to finish the contents of this notebook but there are many useful tips here so I highly recommend that you complete on your own.

[1]:
```
# this line prints your matplot based plots in this notebook, in some computer␣
 ↪without this line your graph will pop out as a window
%matplotlib inline

#call all the matplot libraries I may need
import matplotlib.pyplot as plt
from matplotlib import cm

#the usual
import pandas as pd
import numpy as np

#seaborn
import seaborn as sns; sns.set(color_codes=True)

#some libraries that allow is to run from stats
```

```
import scipy
from scipy import stats
```

0. Set Directory

[2]:
```
#this is the specific directory where the data we want to use is stored
datadirectory = '../data/'

#this is the directory where we want to store the data we finish analyzing
data_out_directory='../output/'
```

Seaborn

For all the plot examples I am going to do below I am going to use seaborn. You can look at the seaborn gallery and look at all the plotting options it offers.

14.Combination skills, Heatplots

I really want to know what the strongest pokemon type is for each generation and stats (Attack. Def. etc).

[3]:
```
# call in my data
pokemon=pd.read_csv(datadirectory+'Pokemon.csv')
pokemon.head()
```

[3]:

	Number	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False

[4]:
```
### make a pivot table
data_subset=pokemon.pivot_table(values='Attack',index=['Type␣
 ↪1'],columns=['Generation'],aggfunc='min')
data_subset
```

[4]:

Generation	1	2	3	4	5	6
Type 1						
Bug	20.0	10.0	30.0	25.0	40.0	22.0

```
Dark          NaN  60.0  55.0  90.0   50.0  54.0
Dragon       64.0   NaN  70.0  70.0   87.0  50.0
Electric     30.0  40.0  40.0  45.0   55.0  38.0
Fairy        45.0  20.0   NaN  50.0    NaN  38.0
Fighting     80.0  35.0  40.0  70.0   80.0  82.0
Fire         41.0  40.0  60.0  58.0   30.0  45.0
Flying        NaN   NaN   NaN   NaN  100.0  30.0
Ghost        35.0  60.0  40.0  50.0   30.0  66.0
Grass        40.0  30.0  40.0  30.0   27.0  61.0
Ground       50.0  60.0  40.0  72.0   66.0   NaN
Ice          50.0  30.0  40.0  60.0   50.0  69.0
Normal        5.0  10.0  20.0   5.0   50.0  36.0
Poison       45.0  90.0  43.0  50.0   50.0  60.0
Psychic      20.0  33.0  23.0  25.0   25.0  48.0
Rock         40.0  64.0  41.0  42.0   75.0  50.0
Steel         NaN  80.0  55.0  24.0   55.0  50.0
Water        10.0  20.0  15.0  20.0   40.0  53.0
```

Coding Challenge

Make a function that makes a pivot table from the pokemon data. This function will allow me to use any aggregation function I want on any pokemon stats I want. The argument for this fuction should be stats_wanted and aggfun_wanted. The index is 'Type 1' and the columns 'Generation'.

[5]:
```
##### your code here
##hint use the code above and change what you need!

def get_summary(stats_wanted,aggfun_wanted):
    data_subset=pokemon.pivot_table(values=stats_wanted,index=['Type␣
 ↪1'],columns=['Generation'],aggfunc=aggfun_wanted)
    return data_subset


get_summary('Speed','max')
```

[5]:
```
Generation      1      2      3      4      5      6
Type 1
Bug         145.0   95.0  160.0   95.0  145.0   89.0
Dark          NaN  115.0  115.0  125.0  106.0   99.0
Dragon       80.0    NaN  120.0  102.0   97.0   95.0
Electric    140.0  115.0  135.0   95.0  116.0  109.0
Fairy        60.0   45.0    NaN   80.0    NaN   99.0
Fighting     95.0   70.0  100.0  112.0  105.0  118.0
Fire        105.0  100.0  100.0  108.0  101.0  126.0
Flying        NaN    NaN    NaN    NaN  121.0  123.0
Ghost       130.0   85.0   75.0  105.0   80.0   99.0
Grass        80.0  110.0  145.0  127.0  116.0   68.0
```

```
Ground      120.0   85.0  100.0   95.0  101.0    NaN
Ice          95.0   75.0  100.0  110.0  105.0   28.0
Normal      121.0  100.0  125.0  135.0  128.0  102.0
Poison       90.0  130.0   65.0   95.0   75.0   44.0
Psychic     150.0  110.0  180.0  115.0  114.0  104.0
Rock        150.0   71.0   75.0   58.0  110.0  110.0
Steel         NaN   70.0  110.0   90.0  108.0   75.0
Water       115.0   85.0  105.0  115.0  108.0  122.0
```

Answer

[6]:
```
##make a fucntion that will aggregate my data based on whatever aggfun I want

def get_summary(stat_wanted,aggfun_wanted):
    data_subset=pokemon.pivot_table(values=stat_wanted,index=['Type␣
 ↪1'],columns=['Generation'],aggfunc=[aggfun_wanted])

    ###drop a level in the columns so the table looks nicer
    data_subset.columns=data_subset.columns.droplevel(0)
    return data_subset

get_summary('Attack','max')
```

[6]:
```
Generation      1      2      3      4      5      6
Type 1
Bug         155.0  185.0   90.0   94.0  135.0   52.0
Dark          NaN   95.0  150.0  125.0  125.0  131.0
Dragon      134.0    NaN  180.0  170.0  170.0  100.0
Electric     90.0   95.0   75.0  123.0  115.0   58.0
Fairy        70.0  120.0    NaN   50.0    NaN  131.0
Fighting    130.0   95.0  120.0  145.0  140.0  124.0
Fire        130.0  130.0  160.0  104.0  140.0  110.0
Flying        NaN    NaN    NaN    NaN  115.0   70.0
Ghost        65.0   60.0  165.0  120.0   55.0  110.0
Grass       105.0   82.0  130.0  132.0   98.0  107.0
Ground      130.0  120.0  180.0  140.0  145.0    NaN
Ice          85.0  100.0  120.0  130.0  110.0  117.0
Normal      125.0  130.0  160.0  160.0  128.0   80.0
Poison      105.0   90.0  100.0  106.0   95.0   75.0
Psychic     190.0  100.0  180.0  165.0  100.0  160.0
Rock        135.0  164.0  125.0  165.0  140.0  160.0
Steel         NaN  125.0  145.0  120.0  100.0  150.0
Water       155.0  105.0  150.0  120.0  108.0   95.0
```

[7]:
```
##make a fucntion that will aggregate my data based on whatever aggfun I want

def get_summary(stat_wanted,aggfun_wanted):
```

```
    data_subset=pokemon.pivot_table(values=stat_wanted,index=['Type␣
↪1'],columns=['Generation'],aggfunc=[aggfun_wanted])

    ###drop a level in the columns so the table looks nicer
    #data_subset.columns=data_subset.columns.droplevel(1)

    ## I can select more than one poke stat toplot, but from what I know with␣
↪pivot you can only do one aggfunction
    return data_subset

get_summary(['Attack','Speed'],('max','min'))
```

[7]:

	max						min			
	min									
	Attack									
	max						min			
Generation	1	2	3	4	5	6	1	2	3	4
Type 1										
Bug	155.0	185.0	90.0	94.0	135.0	52.0	20.0	10.0	30.0	25.0
Dark	NaN	95.0	150.0	125.0	125.0	131.0	NaN	60.0	55.0	90.0
Dragon	134.0	NaN	180.0	170.0	170.0	100.0	64.0	NaN	70.0	70.0
Electric	90.0	95.0	75.0	123.0	115.0	58.0	30.0	40.0	40.0	45.0
Fairy	70.0	120.0	NaN	50.0	NaN	131.0	45.0	20.0	NaN	50.0
Fighting	130.0	95.0	120.0	145.0	140.0	124.0	80.0	35.0	40.0	70.0
Fire	130.0	130.0	160.0	104.0	140.0	110.0	41.0	40.0	60.0	58.0
Flying	NaN	NaN	NaN	NaN	115.0	70.0	NaN	NaN	NaN	NaN
Ghost	65.0	60.0	165.0	120.0	55.0	110.0	35.0	60.0	40.0	50.0
Grass	105.0	82.0	130.0	132.0	98.0	107.0	40.0	30.0	40.0	30.0
Ground	130.0	120.0	180.0	140.0	145.0	NaN	50.0	60.0	40.0	72.0
Ice	85.0	100.0	120.0	130.0	110.0	117.0	50.0	30.0	40.0	60.0
Normal	125.0	130.0	160.0	160.0	128.0	80.0	5.0	10.0	20.0	5.0
Poison	105.0	90.0	100.0	106.0	95.0	75.0	45.0	90.0	43.0	50.0
Psychic	190.0	100.0	180.0	165.0	100.0	160.0	20.0	33.0	23.0	25.0
Rock	135.0	164.0	125.0	165.0	140.0	160.0	40.0	64.0	41.0	42.0
Steel	NaN	125.0	145.0	120.0	100.0	150.0	NaN	80.0	55.0	24.0
Water	155.0	105.0	150.0	120.0	108.0	95.0	10.0	20.0	15.0	20.0

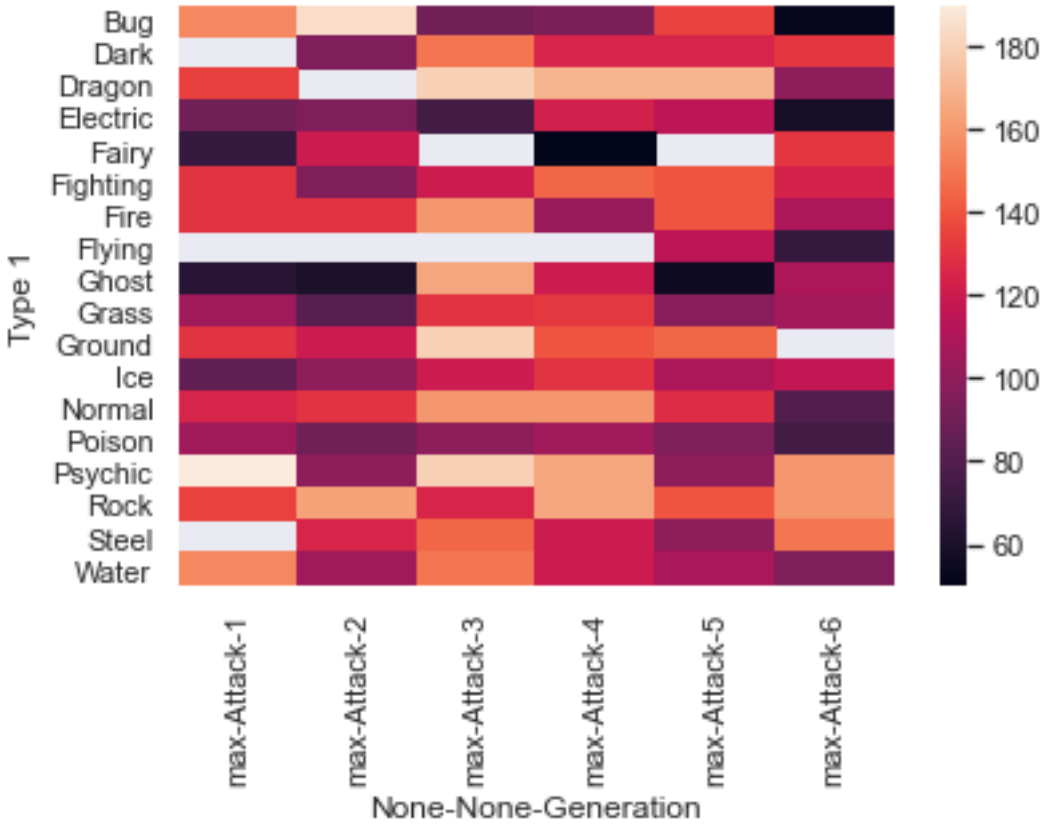
	...									
	...									
	...	Speed								
	...	max				min				
Generation	...	3	4	5	6	1	2	3	4	5
Type 1	...									
Bug	...	160.0	95.0	145.0	89.0	25.0	5.0	15.0	25.0	20.0
Dark	...	115.0	125.0	106.0	99.0	NaN	65.0	20.0	71.0	38.0
Dragon	...	120.0	102.0	97.0	95.0	50.0	NaN	50.0	42.0	48.0
Electric	...	135.0	95.0	116.0	109.0	45.0	35.0	65.0	45.0	40.0

```
Fairy       ...    NaN   80.0    NaN   99.0  35.0   15.0   NaN  80.0    NaN
Fighting    ...  100.0  112.0  105.0  118.0  35.0   35.0  25.0  60.0   35.0
Fire        ...  100.0  108.0  101.0  126.0  60.0   20.0  20.0  61.0   45.0
Flying      ...    NaN    NaN  121.0  123.0   NaN    NaN   NaN   NaN  111.0
Ghost       ...   75.0  105.0   80.0   99.0  80.0   85.0  25.0  35.0   20.0
Grass       ...  145.0  127.0  116.0   68.0  30.0   30.0  30.0  30.0   10.0
Ground      ...  100.0   95.0  101.0    NaN  25.0   40.0  10.0  32.0   32.0
Ice         ...  100.0  110.0  105.0   28.0  85.0   50.0  25.0  65.0   40.0
Normal      ...  125.0  135.0  128.0  102.0  20.0   15.0  20.0   5.0   42.0
Poison      ...   65.0   95.0   75.0   44.0  25.0  130.0  40.0  50.0   65.0
Psychic     ...  180.0  115.0  114.0  104.0  42.0   33.0  23.0  45.0   20.0
Rock        ...   75.0   58.0  110.0  110.0  20.0   30.0  23.0  10.0   15.0
Steel       ...  110.0   90.0  108.0   75.0   NaN   30.0  30.0  23.0   30.0
Water       ...  105.0  115.0  108.0  122.0  15.0   15.0  30.0  34.0   22.0
```

```
Generation     6
Type 1
Bug         29.0
Dark        45.0
Dragon      40.0
Electric    70.0
Fairy       23.0
Fighting    43.0
Fire        60.0
Flying      55.0
Ghost       38.0
Grass       38.0
Ground       NaN
Ice         28.0
Normal      57.0
Poison      30.0
Psychic     68.0
Rock        46.0
Steel       28.0
Water       44.0

[18 rows x 24 columns]
```

[8]:
```
### test if heatmap would look ok
sns.heatmap(get_summary(['Attack'],'max'))
```

[8]:
```
<AxesSubplot:xlabel='None-None-Generation', ylabel='Type 1'>
```



more pivot

[9]:
```
##make a fucntion that will aggregate my data based on whatever aggfun I want

get_summary(['Attack'],'min')
```

[9]:

	min					
	Attack					
Generation	1	2	3	4	5	6
Type 1						
Bug	20.0	10.0	30.0	25.0	40.0	22.0
Dark	NaN	60.0	55.0	90.0	50.0	54.0
Dragon	64.0	NaN	70.0	70.0	87.0	50.0
Electric	30.0	40.0	40.0	45.0	55.0	38.0
Fairy	45.0	20.0	NaN	50.0	NaN	38.0
Fighting	80.0	35.0	40.0	70.0	80.0	82.0
Fire	41.0	40.0	60.0	58.0	30.0	45.0
Flying	NaN	NaN	NaN	NaN	100.0	30.0
Ghost	35.0	60.0	40.0	50.0	30.0	66.0
Grass	40.0	30.0	40.0	30.0	27.0	61.0

```
Ground       50.0  60.0  40.0  72.0   66.0   NaN
Ice          50.0  30.0  40.0  60.0   50.0  69.0
Normal        5.0  10.0  20.0   5.0   50.0  36.0
Poison       45.0  90.0  43.0  50.0   50.0  60.0
Psychic      20.0  33.0  23.0  25.0   25.0  48.0
Rock         40.0  64.0  41.0  42.0   75.0  50.0
Steel         NaN  80.0  55.0  24.0   55.0  50.0
Water        10.0  20.0  15.0  20.0   40.0  53.0
```

[10]:
```
##make a fucntion that will aggregate my data based on whatever aggfun I want

get_summary(['Attack','Speed'],('min','max'))
```

[10]:
```
              min                                                    \
              max
           Attack
              max                                          min
Generation      1      2      3      4      5      6      1     2     3     4
Type 1
Bug         155.0  185.0   90.0   94.0  135.0   52.0  20.0  10.0  30.0  25.0
Dark          NaN   95.0  150.0  125.0  125.0  131.0   NaN  60.0  55.0  90.0
Dragon      134.0    NaN  180.0  170.0  170.0  100.0  64.0   NaN  70.0  70.0
Electric     90.0   95.0   75.0  123.0  115.0   58.0  30.0  40.0  40.0  45.0
Fairy        70.0  120.0    NaN   50.0    NaN  131.0  45.0  20.0   NaN  50.0
Fighting    130.0   95.0  120.0  145.0  140.0  124.0  80.0  35.0  40.0  70.0
Fire        130.0  130.0  160.0  104.0  140.0  110.0  41.0  40.0  60.0  58.0
Flying        NaN    NaN    NaN    NaN  115.0   70.0   NaN   NaN   NaN   NaN
Ghost        65.0   60.0  165.0  120.0   55.0  110.0  35.0  60.0  40.0  50.0
Grass       105.0   82.0  130.0  132.0   98.0  107.0  40.0  30.0  40.0  30.0
Ground      130.0  120.0  180.0  140.0  145.0    NaN  50.0  60.0  40.0  72.0
Ice          85.0  100.0  120.0  130.0  110.0  117.0  50.0  30.0  40.0  60.0
Normal      125.0  130.0  160.0  160.0  128.0   80.0   5.0  10.0  20.0   5.0
Poison      105.0   90.0  100.0  106.0   95.0   75.0  45.0  90.0  43.0  50.0
Psychic     190.0  100.0  180.0  165.0  100.0  160.0  20.0  33.0  23.0  25.0
Rock        135.0  164.0  125.0  165.0  140.0  160.0  40.0  64.0  41.0  42.0
Steel         NaN  125.0  145.0  120.0  100.0  150.0   NaN  80.0  55.0  24.0
Water       155.0  105.0  150.0  120.0  108.0   95.0  10.0  20.0  15.0  20.0

              ...                                                    \
              ...
              ...  Speed
              ...    max                          min
Generation    ...      3      4      5      6     1      2     3     4      5
Type 1        ...
Bug           ...  160.0   95.0  145.0   89.0  25.0    5.0  15.0  25.0   20.0
Dark          ...  115.0  125.0  106.0   99.0   NaN   65.0  20.0  71.0   38.0
Dragon        ...  120.0  102.0   97.0   95.0  50.0    NaN  50.0  42.0   48.0
```

```
Electric    ...  135.0   95.0  116.0  109.0  45.0   35.0  65.0  45.0   40.0
Fairy       ...    NaN   80.0    NaN   99.0  35.0   15.0   NaN  80.0    NaN
Fighting    ...  100.0  112.0  105.0  118.0  35.0   35.0  25.0  60.0   35.0
Fire        ...  100.0  108.0  101.0  126.0  60.0   20.0  20.0  61.0   45.0
Flying      ...    NaN    NaN  121.0  123.0   NaN    NaN   NaN   NaN  111.0
Ghost       ...   75.0  105.0   80.0   99.0  80.0   85.0  25.0  35.0   20.0
Grass       ...  145.0  127.0  116.0   68.0  30.0   30.0  30.0  30.0   10.0
Ground      ...  100.0   95.0  101.0    NaN  25.0   40.0  10.0  32.0   32.0
Ice         ...  100.0  110.0  105.0   28.0  85.0   50.0  25.0  65.0   40.0
Normal      ...  125.0  135.0  128.0  102.0  20.0   15.0  20.0   5.0   42.0
Poison      ...   65.0   95.0   75.0   44.0  25.0  130.0  40.0  50.0   65.0
Psychic     ...  180.0  115.0  114.0  104.0  42.0   33.0  23.0  45.0   20.0
Rock        ...   75.0   58.0  110.0  110.0  20.0   30.0  23.0  10.0   15.0
Steel       ...  110.0   90.0  108.0   75.0   NaN   30.0  30.0  23.0   30.0
Water       ...  105.0  115.0  108.0  122.0  15.0   15.0  30.0  34.0   22.0
```

```
Generation     6
Type 1
Bug         29.0
Dark        45.0
Dragon      40.0
Electric    70.0
Fairy       23.0
Fighting    43.0
Fire        60.0
Flying      55.0
Ghost       38.0
Grass       38.0
Ground       NaN
Ice         28.0
Normal      57.0
Poison      30.0
Psychic     68.0
Rock        46.0
Steel       28.0
Water       44.0

[18 rows x 24 columns]
```

more on groupby–Skip in class–

We can accomplish the same results with groupby but in my opinion it is less intuitive than pivot table. I wrote the code below to introduce you to group by, it is by no means comprehensive but

it will give you a place to start.

[11]:
```
## we can achive something similar with groupby
## in my experience I use pivot tables when I want one agg func and groupby when␣
 ↪I want multiple agg funcs
## you can achive the same level of data analysis with pivot and groupby, you␣
 ↪just have to find what works for you

def get_summary2(stat_wanted,aggfun_wanted):
  ###step 1, aggregare by type 1 and generation
  #data_subset=pokemon.groupby(['Type 1','Generation']).agg(aggfun_wanted)
  ### step 2 select the column with the stat we want
  #data_subset=pokemon.groupby(['Type 1','Generation'])[stat_wanted].
 ↪agg(aggfun_wanted)
  ##clean up the index
    data_subset=pokemon.groupby(['Type 1','Generation'])[stat_wanted].
 ↪agg(aggfun_wanted)
    return data_subset

get_summary2('Attack',['mean'])
```

[11]:
```
                        mean
Type 1 Generation
Bug    1           76.428571
       2           85.416667
       3           55.833333
       4           62.600000
       5           77.611111
...                      ...
Water  2           68.111111
       3           80.666667
       4           72.461538
       5           73.277778
       6           68.000000

[98 rows x 1 columns]
```

[12]:
```
## we can achive something similar with groupby

def get_summary2(stat_wanted,aggfun_wanted):
  ##note that with groupby you can include more columns in stats wanted and more␣
 ↪aggfun arguments
    data_subset=pokemon.groupby(['Type 1','Generation'])[stat_wanted].
 ↪agg(aggfun_wanted)
    return data_subset

get_summary2(['Attack','Speed'],['max','min'])
```

[12]:

```
                    Attack     Speed
                       max min   max min
Type 1 Generation
Bug    1               155  20   145  25
       2               185  10    95   5
       3                90  30   160  15
       4                94  25    95  25
       5               135  40   145  20
...                    ...  ..   ...  ..
Water  2               105  20    85  15
       3               150  15   105  30
       4               120  20   115  34
       5               108  40   108  22
       6                95  53   122  44

[98 rows x 4 columns]
```

more info on unstack

[13]:

```
## we can achive something similar with groupby
## and groupby allows us to have more aggfunctions that pivot table

def get_summary2(stat_wanted,aggfun_wanted):
  ##note that with groupby you can include more columns in stats wanted and more␣
 ↪aggfun arguments
  ##however the resulting dataframe has a multi index that makes plotting more␣
 ↪dififcult
  ## we can go back to one index by doing unstack
    data_subset=pokemon.groupby(['Type 1','Generation'])[stat_wanted].
 ↪agg(aggfun_wanted).unstack()
    return data_subset

get_summary2(['Attack','Speed'],['max','min'])
```

[13]:

```
           Attack                                                    \
              max                                                     min
Generation      1      2      3      4      5      6     1     2     3     4
Type 1
Bug         155.0  185.0   90.0   94.0  135.0   52.0  20.0  10.0  30.0  25.0
Dark          NaN   95.0  150.0  125.0  125.0  131.0   NaN  60.0  55.0  90.0
Dragon      134.0    NaN  180.0  170.0  170.0  100.0  64.0   NaN  70.0  70.0
Electric     90.0   95.0   75.0  123.0  115.0   58.0  30.0  40.0  40.0  45.0
Fairy        70.0  120.0    NaN   50.0    NaN  131.0  45.0  20.0   NaN  50.0
Fighting    130.0   95.0  120.0  145.0  140.0  124.0  80.0  35.0  40.0  70.0
Fire        130.0  130.0  160.0  104.0  140.0  110.0  41.0  40.0  60.0  58.0
Flying        NaN    NaN    NaN    NaN  115.0   70.0   NaN   NaN   NaN   NaN
Ghost        65.0   60.0  165.0  120.0   55.0  110.0  35.0  60.0  40.0  50.0
```

Grass	105.0	82.0	130.0	132.0	98.0	107.0	40.0	30.0	40.0	30.0
Ground	130.0	120.0	180.0	140.0	145.0	NaN	50.0	60.0	40.0	72.0
Ice	85.0	100.0	120.0	130.0	110.0	117.0	50.0	30.0	40.0	60.0
Normal	125.0	130.0	160.0	160.0	128.0	80.0	5.0	10.0	20.0	5.0
Poison	105.0	90.0	100.0	106.0	95.0	75.0	45.0	90.0	43.0	50.0
Psychic	190.0	100.0	180.0	165.0	100.0	160.0	20.0	33.0	23.0	25.0
Rock	135.0	164.0	125.0	165.0	140.0	160.0	40.0	64.0	41.0	42.0
Steel	NaN	125.0	145.0	120.0	100.0	150.0	NaN	80.0	55.0	24.0
Water	155.0	105.0	150.0	120.0	108.0	95.0	10.0	20.0	15.0	20.0

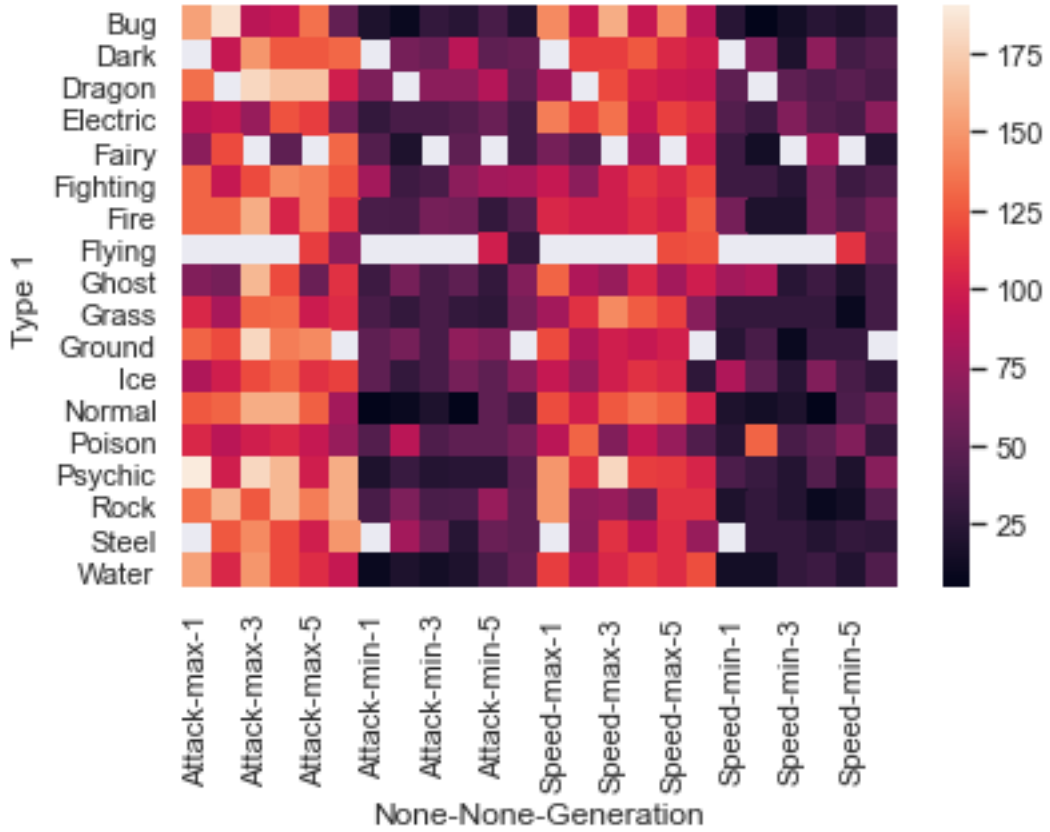
	...	Speed								\
	...	max				min				
Generation	...	3	4	5	6	1	2	3	4	5
Type 1	...									
Bug	...	160.0	95.0	145.0	89.0	25.0	5.0	15.0	25.0	20.0
Dark	...	115.0	125.0	106.0	99.0	NaN	65.0	20.0	71.0	38.0
Dragon	...	120.0	102.0	97.0	95.0	50.0	NaN	50.0	42.0	48.0
Electric	...	135.0	95.0	116.0	109.0	45.0	35.0	65.0	45.0	40.0
Fairy	...	NaN	80.0	NaN	99.0	35.0	15.0	NaN	80.0	NaN
Fighting	...	100.0	112.0	105.0	118.0	35.0	35.0	25.0	60.0	35.0
Fire	...	100.0	108.0	101.0	126.0	60.0	20.0	20.0	61.0	45.0
Flying	...	NaN	NaN	121.0	123.0	NaN	NaN	NaN	NaN	111.0
Ghost	...	75.0	105.0	80.0	99.0	80.0	85.0	25.0	35.0	20.0
Grass	...	145.0	127.0	116.0	68.0	30.0	30.0	30.0	30.0	10.0
Ground	...	100.0	95.0	101.0	NaN	25.0	40.0	10.0	32.0	32.0
Ice	...	100.0	110.0	105.0	28.0	85.0	50.0	25.0	65.0	40.0
Normal	...	125.0	135.0	128.0	102.0	20.0	15.0	20.0	5.0	42.0
Poison	...	65.0	95.0	75.0	44.0	25.0	130.0	40.0	50.0	65.0
Psychic	...	180.0	115.0	114.0	104.0	42.0	33.0	23.0	45.0	20.0
Rock	...	75.0	58.0	110.0	110.0	20.0	30.0	23.0	10.0	15.0
Steel	...	110.0	90.0	108.0	75.0	NaN	30.0	30.0	23.0	30.0
Water	...	105.0	115.0	108.0	122.0	15.0	15.0	30.0	34.0	22.0

Generation	6
Type 1	
Bug	29.0
Dark	45.0
Dragon	40.0
Electric	70.0
Fairy	23.0
Fighting	43.0
Fire	60.0
Flying	55.0
Ghost	38.0
Grass	38.0

```
Ground       NaN
Ice         28.0
Normal      57.0
Poison      30.0
Psychic     68.0
Rock        46.0
Steel       28.0
Water       44.0

[18 rows x 24 columns]
```

[14]:
```
sns.heatmap(get_summary2(['Attack','Speed'],['max','min']))
```

[14]:
```
<AxesSubplot:xlabel='None-None-Generation', ylabel='Type 1'>
```



14.2 Automate graph making

I really want to create a way to visualize the different stats and figure out what the strongest/weakest pokemon types and pokemon are.

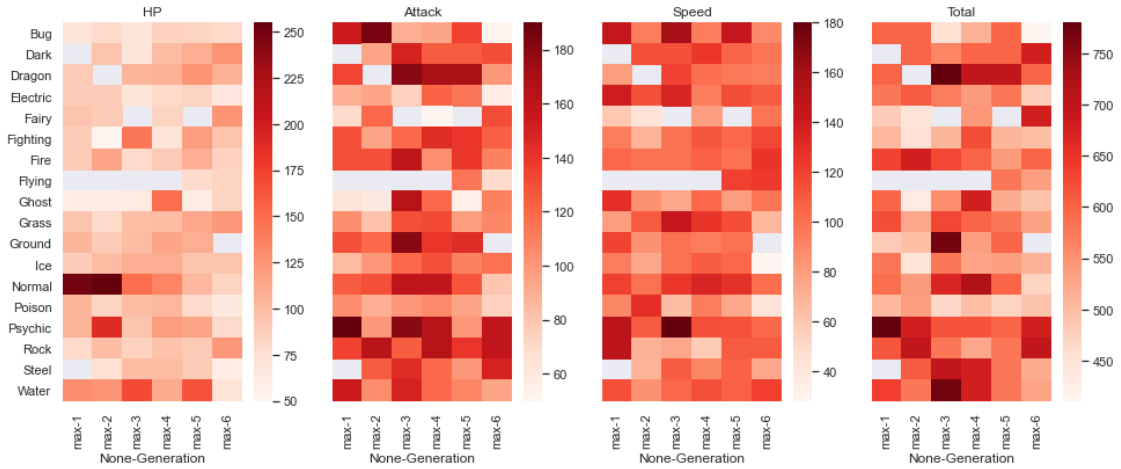
STEP1. I want to make many heatplots to view diff stats so I want to write my code below in such

a way that will allow me to do that.

[15]:
```
for place_in_list,i in enumerate(pokemon.columns):
    print (place_in_list,i)
```

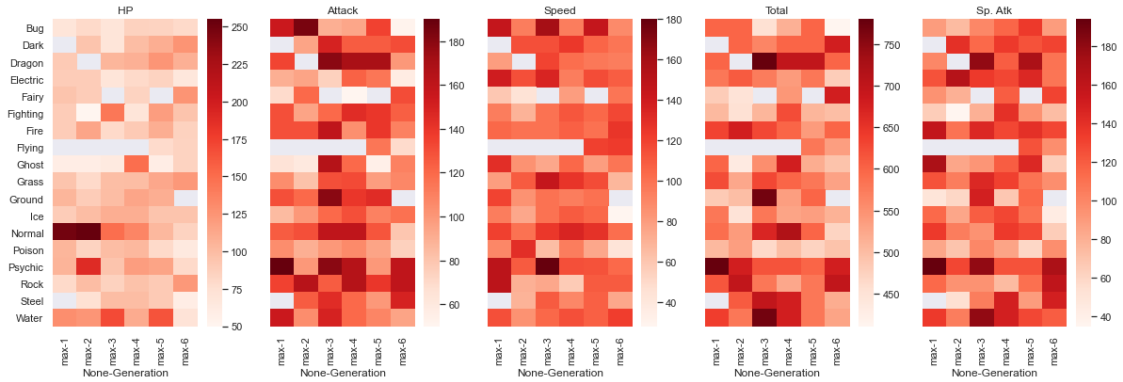
```
0 Number
1 Name
2 Type 1
3 Type 2
4 Total
5 HP
6 Attack
7 Defense
8 Sp. Atk
9 Sp. Def
10 Speed
11 Generation
12 Legendary
```

[16]:
```
## let's make a list that will hold the variables I want
## when I change the contents of my list the rest of my code will react␣
 ↪accordingly
stats_all=['HP','Attack','Speed','Total']

##call a figure object that always has 1 row but the columns change based on the␣
 ↪contents of my list
fig, ax=plt.subplots(nrows=1,ncols=len(stats_all),sharey=True)

#change the figure size by the lenth of my list so my figure is always pretty
fig.set_size_inches(len(stats_all)*4,6)

for count,stat in enumerate(stats_all):
  #print (count,stat)
  ## I'm going to use the output from my function that uses pivot table but you␣
 ↪can you anything
    hold=get_summary(stat,'max')
    sns.heatmap(hold,ax=ax[count],cmap="Reds")
    ax[count].set_ylabel('')
    ax[count].set_title(f'{stat}')
```



[17]:
```
def get_heat(stats_all,aggfunction_wanted):

    ## let's make a list that will hold the variables I want
    ## when I change the contents of my list the rest of my code will react␣
 ↪accordingly

    ##call a figure object that always has 1 row but the columns change based on␣
 ↪the contents of my list
    fig, ax=plt.subplots(nrows=1,ncols=len(stats_all),sharey=True)

    #change the figure size by the lenth of my list so my figure is always pretty
    fig.set_size_inches(len(stats_all*4),6)

    for count,stat in enumerate(stats_all):
        #print (count,stat)
        hold=get_summary(stat,aggfunction_wanted)
        sns.heatmap(hold,ax=ax[count],cmap="Reds")
        ax[count].set_ylabel('')
        ax[count].set_title(f'{stat}')

plt.show()
```

[18]:
```
##cool now we can explore the different pokemon stats based on diff math stats
get_heat(stats_all=['HP','Attack','Speed','Total','Sp.␣
 ↪Atk'],aggfunction_wanted='max')
```



[19]:
```
## make a function that quickly lets me view pokemons based on Type 1 and␣
 ↪Generation

def get_pokemon(type_1,generation,stat_wanted):
    out=pokemon.loc[((pokemon['Type 1']==␣
 ↪type_1)&(pokemon['Generation']==generation))].copy()
    out.sort_values(by=stat_wanted,ascending=False,inplace=True)
    return out

get_pokemon('Psychic',1,'Attack')
```

[19]:

	Number	Name	Type 1	Type 2	Total	HP	Attack	\
163	150	MewtwoMega Mewtwo X	Psychic	Fighting	780	106	190	
164	150	MewtwoMega Mewtwo Y	Psychic	NaN	780	106	150	
162	150	Mewtwo	Psychic	NaN	680	106	110	
165	151	Mew	Psychic	NaN	600	100	100	
105	97	Hypno	Psychic	NaN	483	85	73	
70	65	Alakazam	Psychic	NaN	500	55	50	
71	65	AlakazamMega Alakazam	Psychic	NaN	590	55	50	
104	96	Drowzee	Psychic	NaN	328	60	48	
131	122	Mr. Mime	Psychic	Fairy	460	40	45	
69	64	Kadabra	Psychic	NaN	400	40	35	
68	63	Abra	Psychic	NaN	310	25	20	

	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
163	100	154	100	130	1	True
164	70	194	120	140	1	True
162	90	154	90	130	1	True
165	100	100	100	100	1	False
105	70	73	115	67	1	False
70	45	135	95	120	1	False
71	65	175	95	150	1	False
104	45	43	90	42	1	False

```
131        65        100        120        90            1       False
69         30        120         70       105            1       False
68         15        105         55        90            1       False
```

Conclusion:

Pandas will help you clean, order and analyze your data. Then you can use libraries like seaborn so make figures for your manuscripts.

Google and stackoverflow are your friends, use them! Also YOUTUBE is a gold mine. This is on of my favorite channels but explore and find your own!

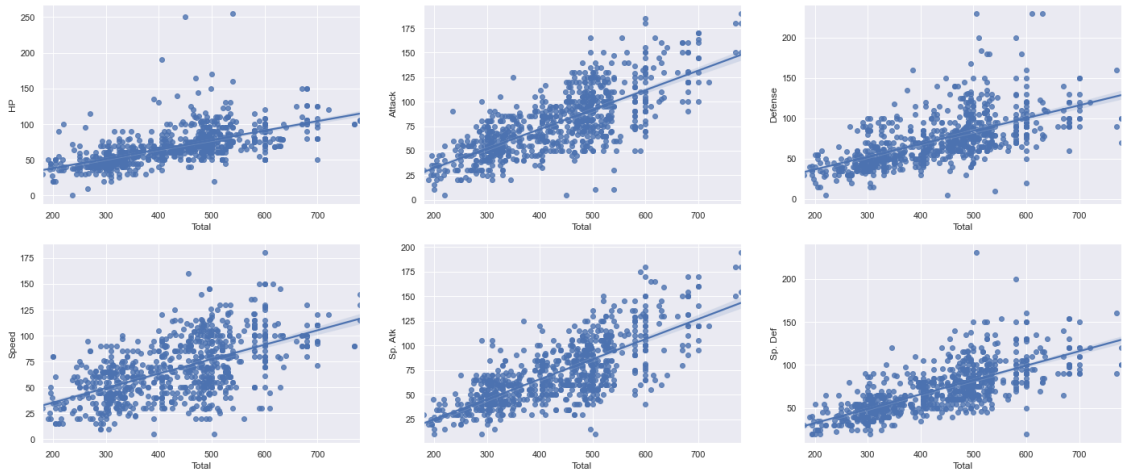
RUN THE REST OF CODE AND SHOW GRAPHS

15. Extra: Combination skills, Correlations

Scatter plots

I want to know what stats values are highly correlated. So I am going to use sns.regplot

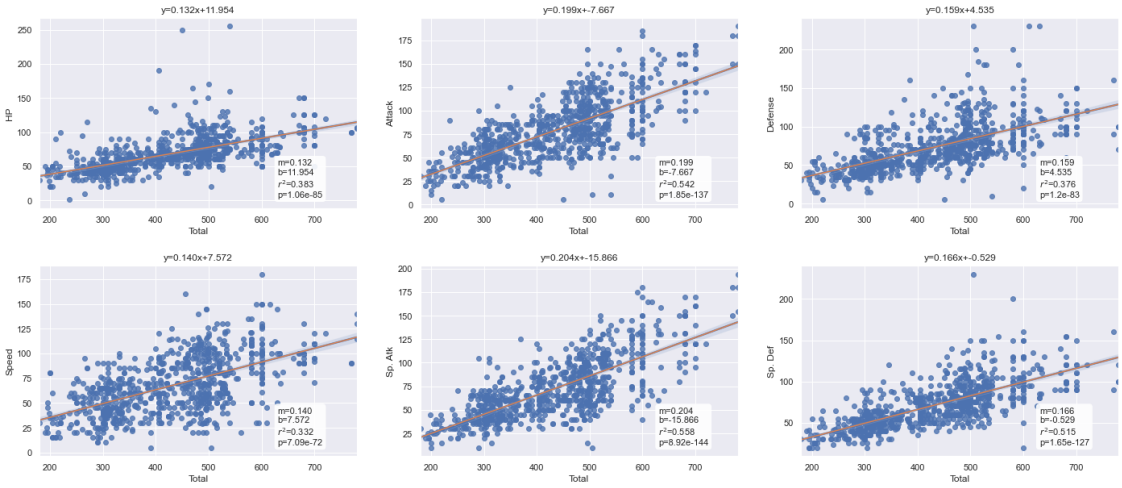
[20]:
```
## not specifying how many subplots I want will allow me to add subplots with␣
 ↪the for loop
fig = plt.figure()

fig.subplots_adjust(hspace=0.2, wspace=0.2)

## this list can change but it muts always change by the nrow value you specify␣
 ↪in the for loop
stats_all=['HP','Attack','Defense','Speed','Sp. Atk','Sp. Def']
fig.set_size_inches(len(stats_all*4),10)

for count,stat in enumerate(stats_all):
    ax = fig.add_subplot(2, 3, count+1)
    ax=sns.regplot(x='Total', y=stat, data=pokemon)

plt.show()
plt.close()
```



Cool figure, let's add some labels so we can read it better

[21]:
```
fig = plt.figure()

z = len(stats_all)/2 #this variable is classified as a floating point number, we
 ↪need to change it to an integer.

fig.subplots_adjust(hspace=0.3, wspace=0.2)

stats_all=['HP','Attack','Defense','Speed','Sp. Atk','Sp. Def']
fig.set_size_inches(len(stats_all*4),10)

for count,stat in enumerate(stats_all):
    ax = fig.add_subplot(2, int(z), count+1)
    ax=sns.regplot(x='Total', y=stat, data=pokemon)

    ##get stats for a label
    temp=pokemon[['Total',stat]].dropna()
    results=stats.linregress(temp.iloc[:,0],temp.iloc[:,1])
    #print (results)

    ##this code makes the labels in the box
    props=dict(boxstyle='round',facecolor='white',alpha=.9)
    textstr='m={:.3f}\nb={:.3f}\n$r^2$={:.3f}\np={:.3}'.
 ↪format(results[0],results[1],results[2]**2,results[3]) #grabs the values from
 ↪stats_out
    ax.text(.75,.05,textstr,transform=ax.
 ↪transAxes,va='bottom',fontsize=11,bbox=props) #change the formatting of the
 ↪box

    #this code here allows me to make a line manually
```

```
    x1=np.array([temp.Total.min(),temp.Total.max()])
    y1=results[0]*x1+results[1]
    ax.plot(x1,y1)

    ## add line equation as a title
    m=results[0]
    b=results[1]
    ax.set_title('y={:.3f}x+{:.3f}'.format(m,b))


plt.show()
plt.close()
```



Let's make the code into a function so we can easily change what we plot

[ ]:
```
def correlates_with_wanted(stats_wanted):

    fig = plt.figure()
    fig.subplots_adjust(hspace=0.3, wspace=0.2)

    stats_all=['HP','Attack','Defense','Speed','Sp. Atk','Sp. Def','Total']
    stats_all.remove(stats_wanted)
    fig.set_size_inches(len(stats_all*4),10)

    for count,stat in enumerate(stats_all):
        # remeber that add_subplots is (nrow,ncol,number) and number starts at 1␣
↪while enumerate at 0
        ax = fig.add_subplot(2, int(z), count+1)
        ax=sns.regplot(x=stats_wanted, y=stat, data=pokemon)

        ##get stats for a label
```

```
        temp=pokemon[[stats_wanted,stat]].dropna()
        results=stats.linregress(temp.iloc[:,0],temp.iloc[:,1])
        #print (results)

        props=dict(boxstyle='round',facecolor='white',alpha=.9)
        textstr='m={:.3f}\nb={:.3f}\n$r^2$={:.3f}\np={:.3}'.
↪format(results[0],results[1],results[2]**2,results[3]) #grabs the values from␣
↪stats_out
        ax.text(.75,.05,textstr,transform=ax.
↪transAxes,va='bottom',fontsize=11,bbox=props) #change the formatting of the␣
↪box

        #this code here woul allow to make a line manually
        x1=np.array([temp[stats_wanted].min(),temp[stats_wanted].max()])
        y1=results[0]*x1+results[1]
        ax.plot(x1,y1)

        ## add line equation as a title
        m=results[0]
        b=results[1]
        ax.set_title('y={:.3f}x+{:.3f}'.format(m,b))


plt.show()
plt.close()

correlates_with_wanted('Defense')
```

Table of correlations

[55]:
```
### make a table of correlations

stats_all=['HP','Attack','Defense','Speed','Sp. Atk','Sp. Def','Total']

stats_df=pd.DataFrame(index=stats_all,columns=stats_all)

for i in stats_all:
    for k in stats_all:
        stats_results=stats.linregress(pokemon[i],pokemon[k])
        #print(stats_results)
        stats_df[i][k]=stats_results[2].round(2)

stats_df
```

[55]:

	HP	Attack	Defense	Speed	Sp. Atk	Sp. Def	Total
HP	1.0	0.42	0.24	0.18	0.36	0.38	0.62
Attack	0.42	1.0	0.44	0.38	0.4	0.26	0.74

```
Defense  0.24   0.44     1.0  0.02    0.22    0.51  0.61
Speed    0.18   0.38    0.02   1.0    0.47    0.26  0.58
Sp. Atk  0.36    0.4    0.22  0.47     1.0    0.51  0.75
Sp. Def  0.38   0.26    0.51  0.26    0.51     1.0  0.72
Total    0.62   0.74    0.61  0.58    0.75    0.72   1.0
```

[56]:
```
### a faster trick
corr_all=pokemon.corr()
corr_all
```

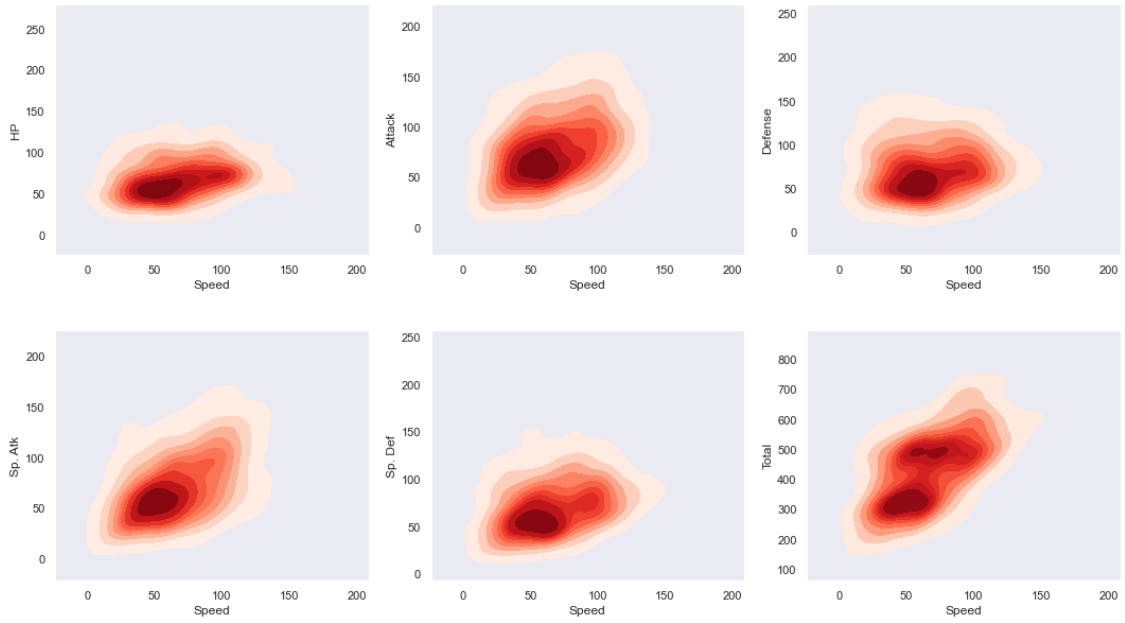
[56]:
```
                Number     Total        HP    Attack   Defense   Sp. Atk  \
Number        1.000000  0.119813  0.097614  0.102298  0.094786  0.088759
Total         0.119813  1.000000  0.618748  0.736211  0.612787  0.747250
HP            0.097614  0.618748  1.000000  0.422386  0.239622  0.362380
Attack        0.102298  0.736211  0.422386  1.000000  0.438687  0.396362
Defense       0.094786  0.612787  0.239622  0.438687  1.000000  0.223549
Sp. Atk       0.088759  0.747250  0.362380  0.396362  0.223549  1.000000
Sp. Def       0.085817  0.717609  0.378718  0.263990  0.510747  0.506121
Speed         0.010733  0.575943  0.175952  0.381240  0.015227  0.473018
Generation    0.982516  0.048384  0.058683  0.051451  0.042419  0.036437
Legendary     0.153396  0.501758  0.273620  0.345408  0.246377  0.448907

                Sp. Def     Speed  Generation  Legendary
Number        0.085817  0.010733    0.982516   0.153396
Total         0.717609  0.575943    0.048384   0.501758
HP            0.378718  0.175952    0.058683   0.273620
Attack        0.263990  0.381240    0.051451   0.345408
Defense       0.510747  0.015227    0.042419   0.246377
Sp. Atk       0.506121  0.473018    0.036437   0.448907
Sp. Def       1.000000  0.259133    0.028486   0.363937
Speed         0.259133  1.000000   -0.023121   0.326715
Generation    0.028486 -0.023121    1.000000   0.079794
Legendary     0.363937  0.326715    0.079794   1.000000
```

Density plots

Once you have code that works all you need to do is change the plot kind and it will work

[57]:
```
sns.set(style="dark")

def density_with_wanted(stats_wanted):

    fig = plt.figure()
    fig.subplots_adjust(hspace=0.3, wspace=0.2)

    stats_all=['HP','Attack','Defense','Speed','Sp. Atk','Sp. Def','Total']
    stats_all.remove(stats_wanted)
```

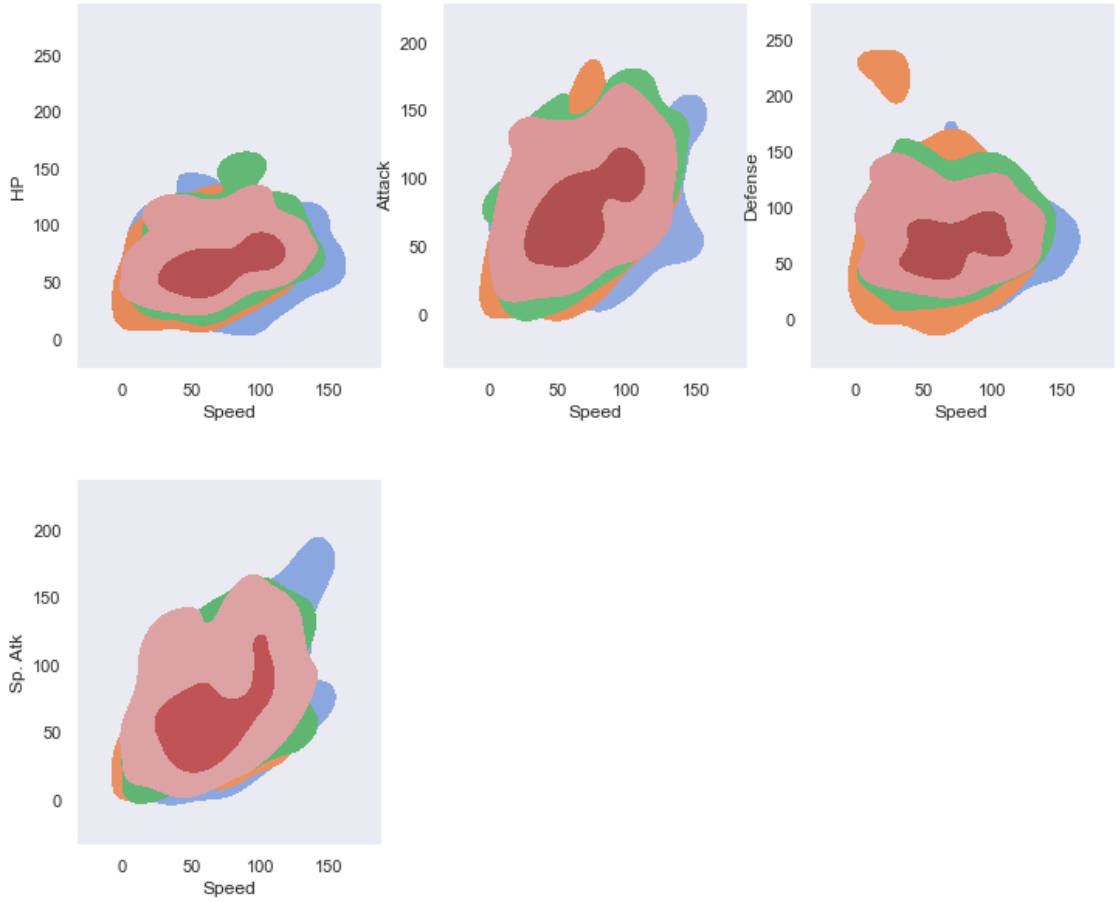
```
    fig.set_size_inches(len(stats_all*3),10)

    for count,stat in enumerate(stats_all):
        # remeber that add_subplots is (nrow,ncol,number) and number starts at 1␣
↪while enumerate at 0
        ax = fig.add_subplot(2, int(z), count+1)
        ##change the kind of figure you want
        sns.kdeplot(x=pokemon[stats_wanted], y=pokemon[stat], cmap='Reds',␣
↪shade=True, ax=ax)

plt.show()
plt.close()

density_with_wanted('Speed')
```



we can make the function a little more complicated by adding in another for loop. This will make it so that each subplots has multiple plots inside.

[59]:
```
sns.set(style="dark")

def density_with_wanted(stats_wanted):

    fig = plt.figure()
    fig.subplots_adjust(hspace=0.3, wspace=0.2)

    stats_all=['HP','Attack','Defense','Speed','Sp. Atk']
```

```
    stats_all.remove(stats_wanted)

    fig.set_size_inches(len(stats_all*3),10)

    for count,stat in enumerate(stats_all):
        # remeber that add_subplots is (nrow,ncol,number) and number starts at 1␣
↪while enumerate at 0
        ax = fig.add_subplot(2, int(z), count+1)

        for gen in [1,2,4,5]:
            pokemon_subset=pokemon.loc[pokemon['Generation']==gen]
            #x,y=pokemon_subset[stats_wanted],pokemon_subset[stat]
            sns.kdeplot(x=pokemon_subset[stats_wanted], y=pokemon_subset[stat],␣
↪shade=True, ax=ax,thresh=0.05, n_levels=3)

plt.show()
plt.close()

density_with_wanted('Speed')
```



16. Extra: Data exploration

Once you have cleaned up your datatable it is really easy to plot. You just look for the kind of plot you want and specify the data you want. You can play around with the variables until you get something meaningful.

Pairplots

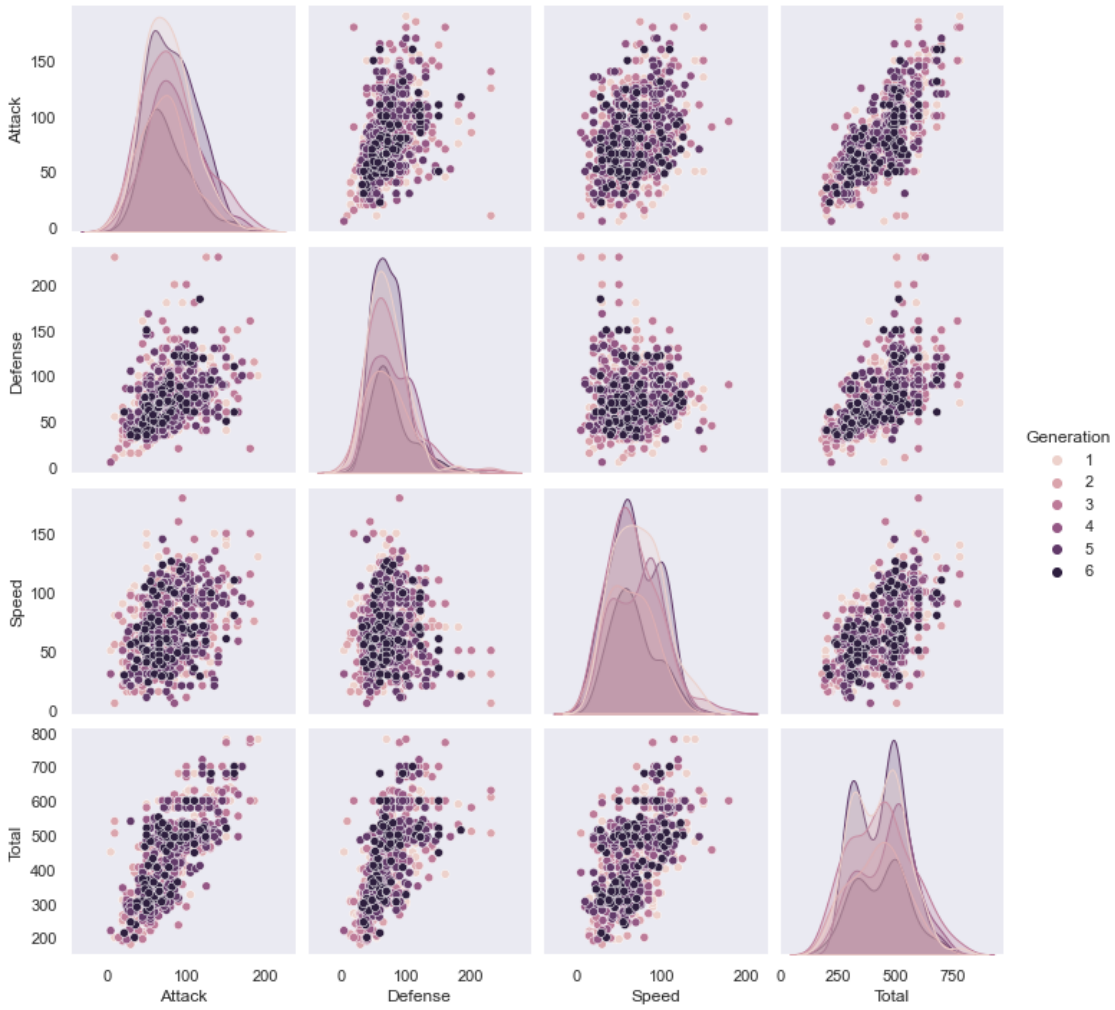
Like the plots we manually made above, Pairplot allows you to see relationships between variables.

[60]:
```
sns.pairplot(pokemon[['Type␣
↪1','Attack','Defense','Speed','Total','Legendary']],hue='Legendary')
```

[60]:
```
<seaborn.axisgrid.PairGrid at 0x7f8f9dda95b0>
```

[61]:
```
sns.pairplot(pokemon[['Type␣
↪1','Attack','Defense','Speed','Total','Generation']],hue='Generation')
```

[61]:
```
<seaborn.axisgrid.PairGrid at 0x7f8f9ee74580>
```



[62]:
```
pokemon['generation_legend']=pokemon['Generation'].
↪astype(str)+'_'+pokemon['Legendary'].astype(str)
pokemon.head()
```

[62]:

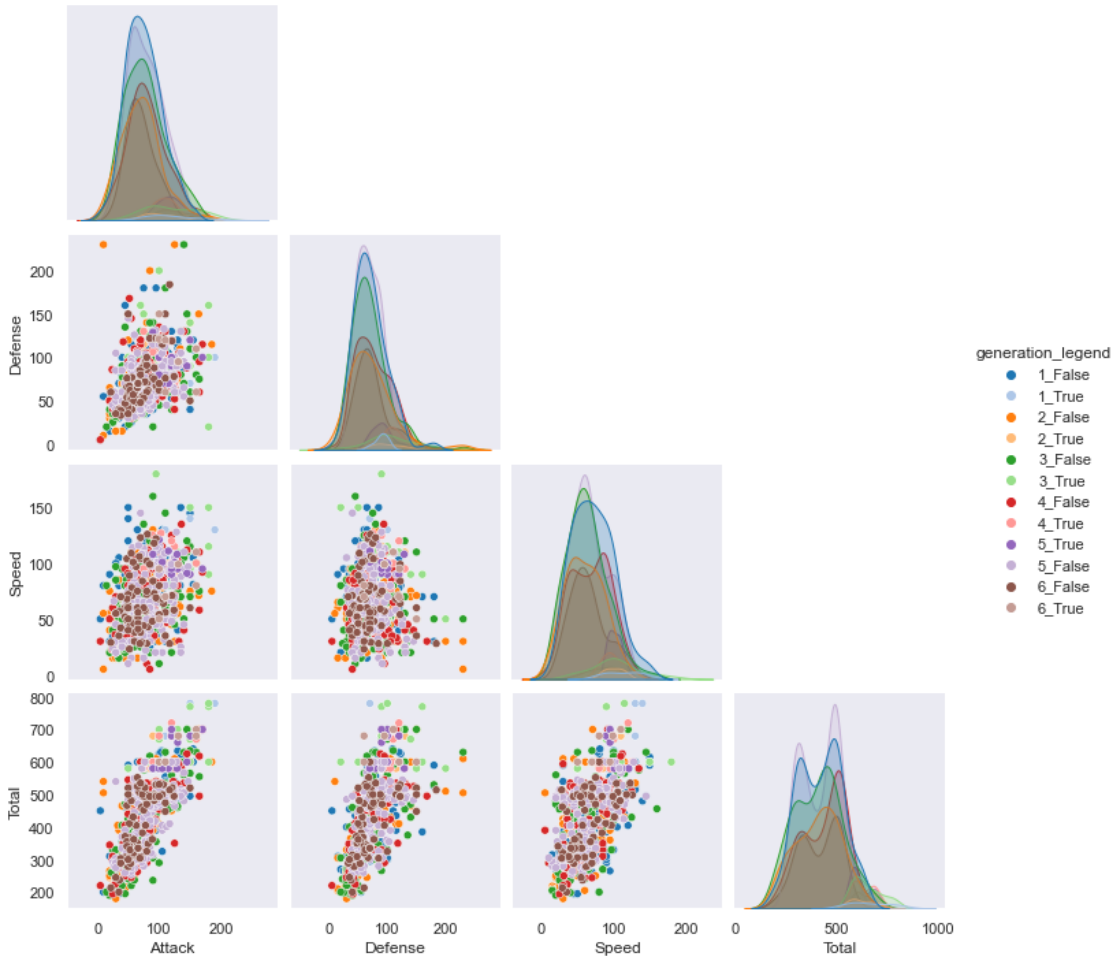
	Number	Name	Type 1	Type 2	Total	HP	Attack	Defense	\
0	1	Bulbasaur	Grass	Poison	318	45	49	49	
1	2	Ivysaur	Grass	Poison	405	60	62	63	
2	3	Venusaur	Grass	Poison	525	80	82	83	
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	
4	4	Charmander	Fire	NaN	309	39	52	43	

	Sp. Atk	Sp. Def	Speed	Generation	Legendary	generation_legend
0	65	65	45	1	False	1_False
1	80	80	60	1	False	1_False
2	100	100	80	1	False	1_False
3	122	120	80	1	False	1_False
4	60	50	65	1	False	1_False

let's make our new figure, and change some details using palette

[63]:
```
sns.pairplot(pokemon[['Type␣
↪1','Attack','Defense','Speed','Total','generation_legend']],hue='generation_legend',palette='
```

[63]:
```
<seaborn.axisgrid.PairGrid at 0x7f8f9f77ecd0>
```



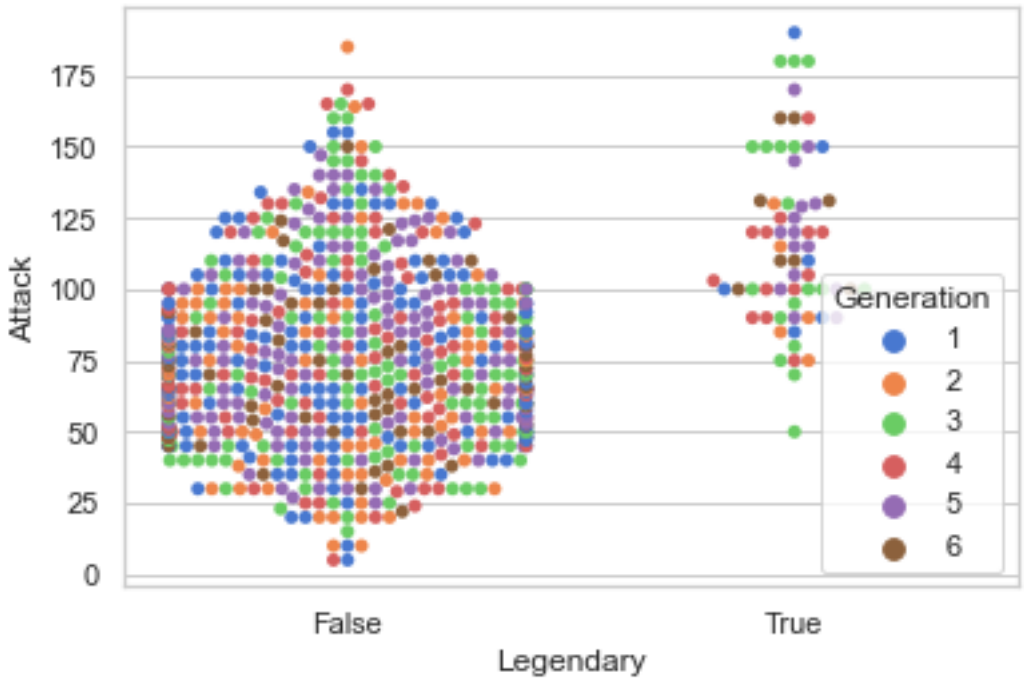
Swarmplots

I wanted to explore the Legendary distrubution more so I looked in seaborn to see what kinds of plots are used for categorical data and swarmplots came up.

[64]:
```
##this calls the style of the plot
sns.set(style="whitegrid", palette="muted")

#this calls the actual swarmplot
sns.swarmplot(x="Legendary", y="Attack", hue="Generation", data=pokemon)

#you will get a warning that some of the plots can't be placed, this is ok␣
 ↪because it's just an example.
```

```
/Users/jsenger/opt/anaconda3/lib/python3.8/site-
packages/seaborn/categorical.py:1296: UserWarning: 28.2% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

[64]:
```
<AxesSubplot:xlabel='Legendary', ylabel='Attack'>
```

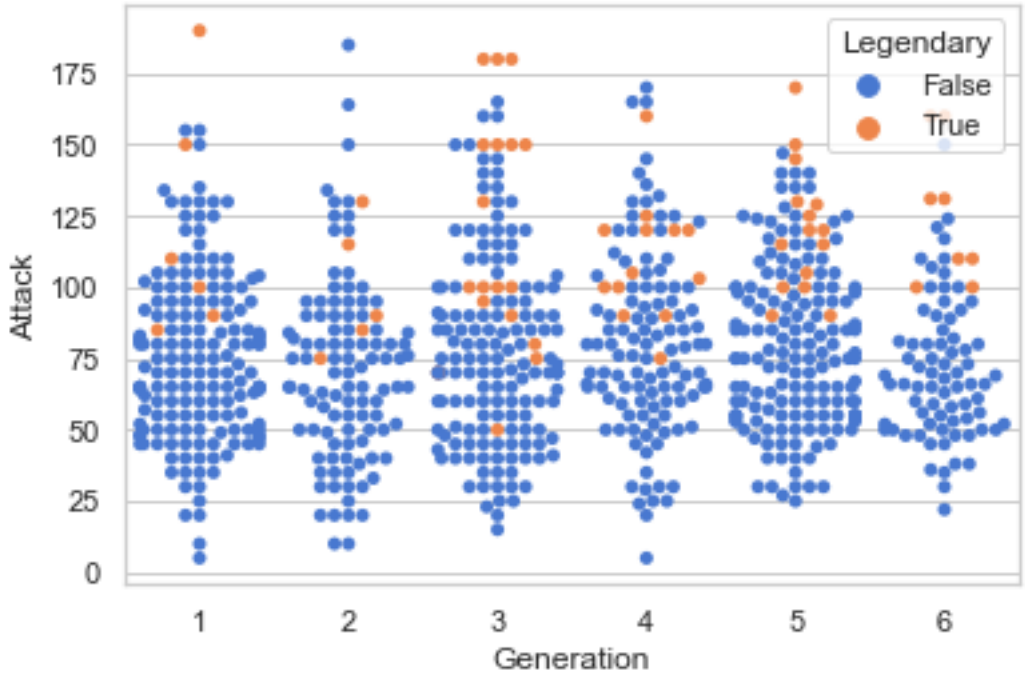


[65]:
```
sns.swarmplot(x="Generation", y="Attack", hue="Legendary", data=pokemon)
```

```
/Users/jsenger/opt/anaconda3/lib/python3.8/site-
packages/seaborn/categorical.py:1296: UserWarning: 16.3% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/Users/jsenger/opt/anaconda3/lib/python3.8/site-
packages/seaborn/categorical.py:1296: UserWarning: 10.0% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
```

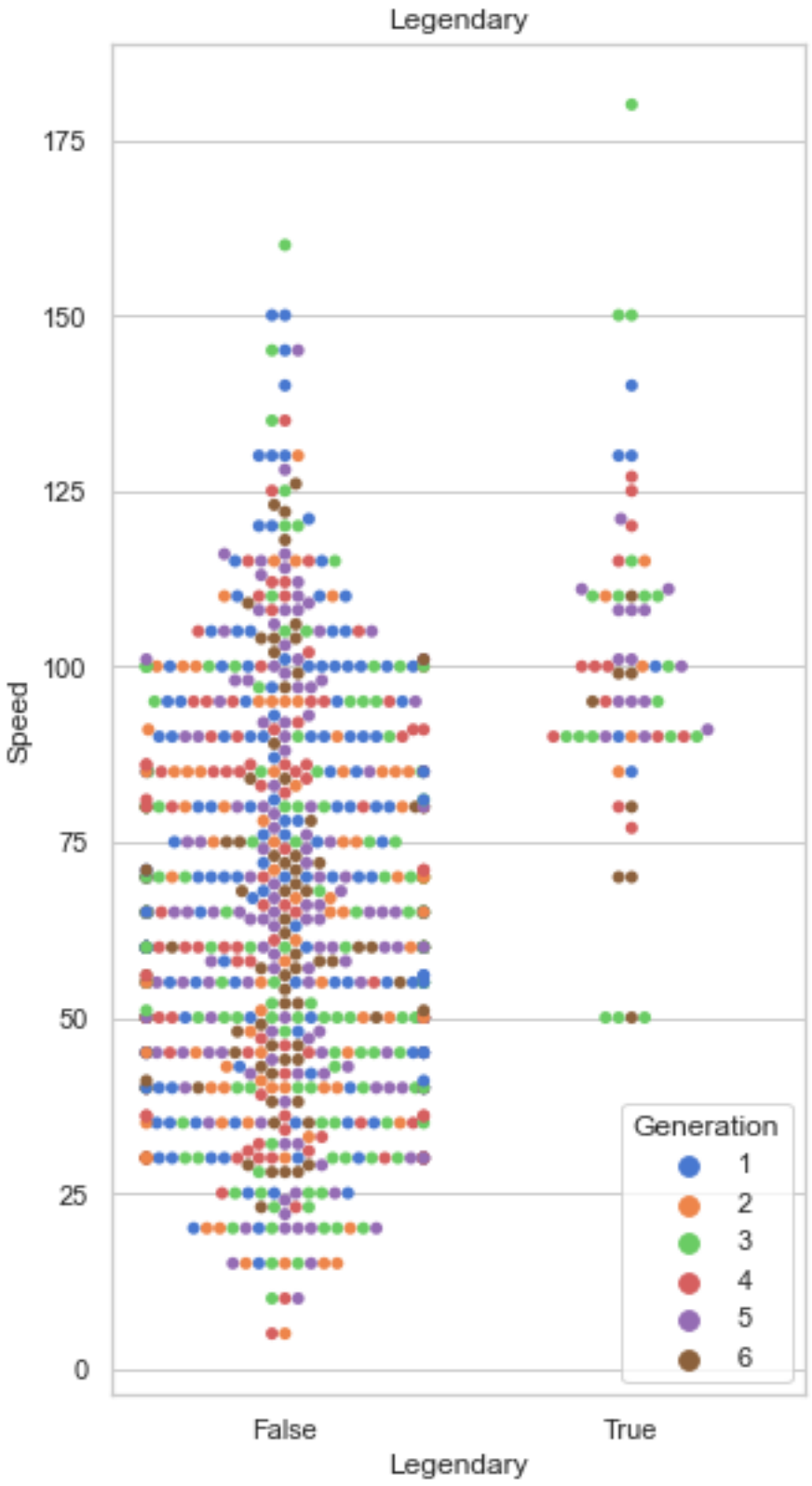
```
  warnings.warn(msg, UserWarning)
/Users/jsenger/opt/anaconda3/lib/python3.8/site-
packages/seaborn/categorical.py:1296: UserWarning: 15.8% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

[65]: <AxesSubplot:xlabel='Generation', ylabel='Attack'>



[66]:
```
fig, ax=plt.subplots()
fig.set_size_inches(5,10)

sns.swarmplot(x="Legendary", y="Speed", hue="Generation", data=pokemon,ax=ax)
ax.set_title('Legendary')

plt.savefig(data_out_directory+'swarm.pdf')
```

```
/Users/jsenger/opt/anaconda3/lib/python3.8/site-
packages/seaborn/categorical.py:1296: UserWarning: 24.4% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```


Legendary
175
150
125
100
75
50
25
0
Speed
Generation
1
2
3
4
5
6
False
True
Legendary

Coding challenge

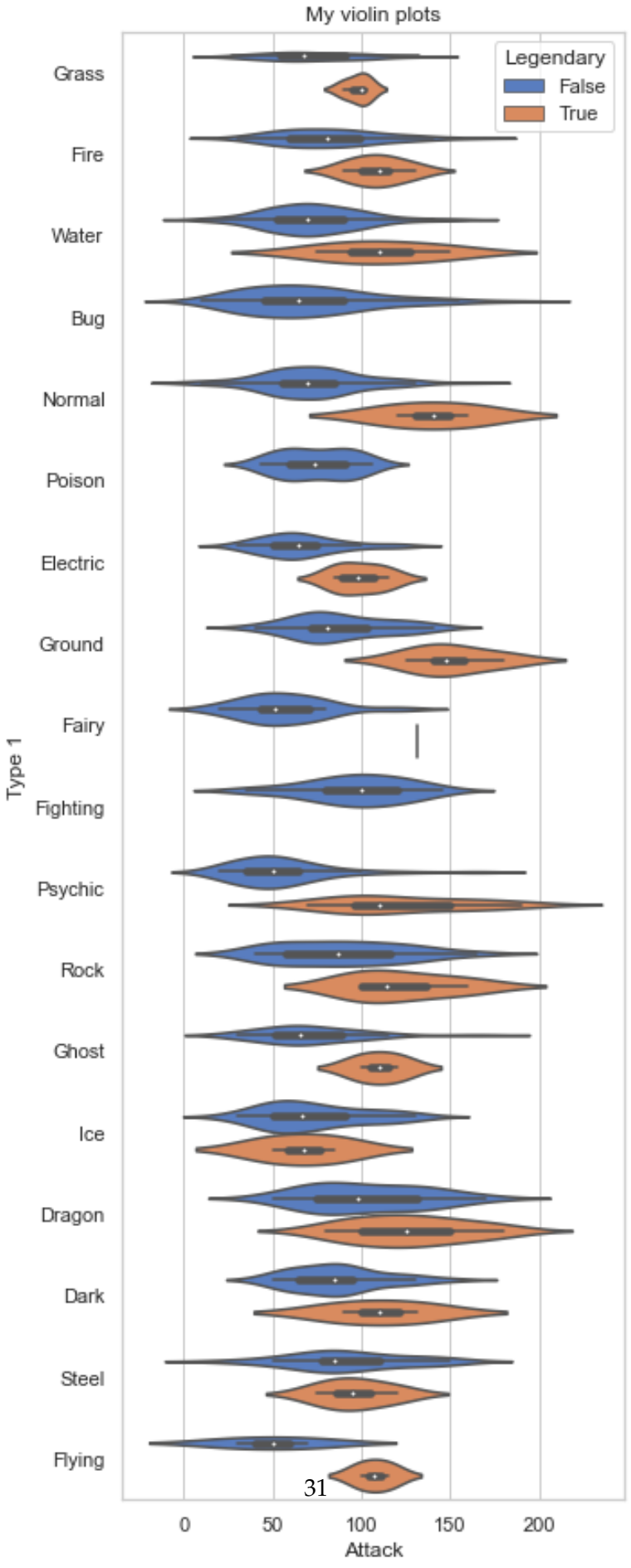
Here is the website on how to make violinplots with seaborn.

[67]:
```
#### using the full pokemon data
### make a violinplot where the x is Attack, y is Type 1, and hue is legendary
## remeber to change the figure size so the figure is readable
```

Answer

[68]:
```
#@title
fig, ax=plt.subplots()

fig.set_size_inches(5,15)

sns.violinplot(x="Attack", y="Type 1", data=pokemon, hue='Legendary',ax=ax)

ax.set_title('My violin plots')
```

[68]:
```
Text(0.5, 1.0, 'My violin plots')
```


My violin plots
Legendary
False
True
Grass
Fire
Water
Bug
Normal
Poison
Electric
Ground
Fairy
Fighting
Psychic
Rock
Ghost
Ice
Dragon
Dark
Steel
Flying
Type 1
0
50
100
150
200
Attack

17.Extra fun

THIS IS HOW YOU MAKE PLOTS

1. call figure and create your canvas . . . fig, ax= plt.subplots()
2. follow instructions on websites for the actual plot you want . . . some plot code
3. change some stuff . . . .ax.set_title('My plot title')
4. save the plot if you want. . . plt.savefig('My_plot')

cluster with heatmap

more code here

[69]:
```
### let's select the legendary and work with those
pokemon_legend=pokemon.loc[pokemon['Legendary']==True,]
pokemon_legend
```

[69]:

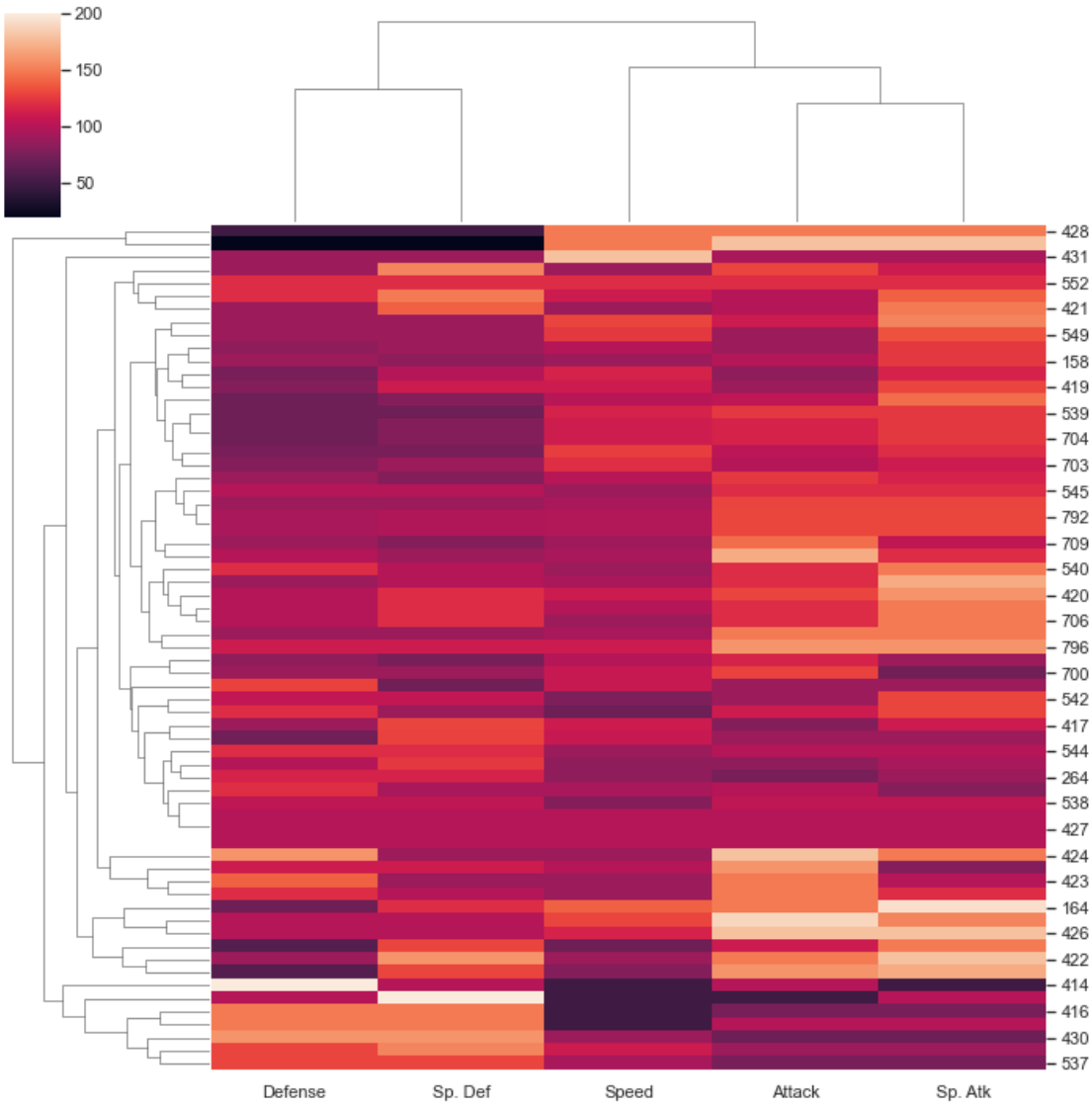
	Number	Name	Type 1	Type 2	Total	HP	Attack	\
156	144	Articuno	Ice	Flying	580	90	85	
157	145	Zapdos	Electric	Flying	580	90	90	
158	146	Moltres	Fire	Flying	580	90	100	
162	150	Mewtwo	Psychic	NaN	680	106	110	
163	150	MewtwoMega Mewtwo X	Psychic	Fighting	780	106	190	
..	...	...	...	...	...	...	...	
795	719	Diancie	Rock	Fairy	600	50	100	
796	719	DiancieMega Diancie	Rock	Fairy	700	50	160	
797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	
798	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	
799	721	Volcanion	Fire	Water	600	80	110	

	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	generation_legend
156	100	95	125	85	1	True	1_True
157	85	125	90	100	1	True	1_True
158	90	125	85	90	1	True	1_True
162	90	154	90	130	1	True	1_True
163	100	154	100	130	1	True	1_True
..	...	...	...	...	...	...	...
795	150	100	150	50	6	True	6_True
796	110	160	110	110	6	True	6_True
797	60	150	130	70	6	True	6_True
798	60	170	130	80	6	True	6_True
799	120	130	90	70	6	True	6_True

```
[65 rows x 14 columns]
```

[70]:
```
sns.clustermap(pokemon_legend[['Attack','Defense','Speed','Sp. Atk','Sp. Def']])
```

[70]:
```
<seaborn.matrix.ClusterGrid at 0x7f8fa1f85340>
```



Venn Diagrams

I just googled python venn diagrams and searched through some of the pages that came out. more info HERE

[ ]:
```
pip install matplotlib_venn
```

[71]:
```
from matplotlib_venn import venn3

# Make the diagram, code from the website I listed
## I'm just going to add some stuff the website did not show

#call figure and create your canvas
fig, ax= plt.subplots()

## follow instructions on website
venn3(subsets = (20, 10, 12, 10, 9, 4, 3), set_labels = ('Group A', 'Group B',␣
 ↪'Group C'), alpha = 0.5,ax=ax)

##change some stuff
ax.set_title('My Venn')

#save the plot if you want.
#plt.savefig('My_Venn')
```

[71]:
```
Text(0.5, 1.0, 'My Venn')
```