Can we predict Pokemon Type 1 or HP based on given information?

Authors:

- Edgar Leos Almanza
- Cesar Sanchez
- Moses Garcia

Introduction:

For our project, we worked on a Pokemon dataset that includes the type of Pokemon and their statistics. The objective of this project is to predict the Pokemon's HP based off of the given information. This project might be a bit challenging but definitely worth the struggle as we will be discovering new ways to identify a Pokemon's HP based on essential information. This project allows us to relive our childhood. If we are able to predict the Pokemon's HP based on the Attack, Defense, Sp. Atk, Sp. Def, Speed, and Generation we essentually become better "Pokemon trainers" and eventually Pokemon masters.

Clear Description of the Source Data Set (Predictions and Features)

This project consists of the Pokemon.csv data set which includes **13 features**, with **800 samples**. The data set consists of columns pertaining to the Pokemon's statistics such as: Type, HP, Attack, Defense, etc. Some of the features that will be used as predictors are: *HP*, *Attack*, *Defense*, *Sp. Atk*, *Sp. Def*.

After getting an idea of the data that we were working with, we decided to drop **Total** and **Type 2** as those features would not be needed in order to make our predictions. After dropping some features, the data that was left behind contained **2 categorical features**: *Name*, and *Type 1* and **8 numerical features**: #, *HP*, *Attack*, *Defense*, *Sp. Atk*, *Sp. Def*, *Speed*, and *Generation*.

Preliminary Work on Data Preperation

Out[2]: Click here to display/hide the code.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   #           800 non-null    int64
 1   Name        800 non-null    object
 2   Type 1      800 non-null    object
 3   Type 2      414 non-null    object
 4   Total       800 non-null    int64
 5   HP          800 non-null    int64
 6   Attack      800 non-null    int64
 7   Defense     800 non-null    int64
 8   Sp. Atk     800 non-null    int64
 9   Sp. Def     800 non-null    int64
 10  Speed       800 non-null    int64
 11  Generation  800 non-null    int64
 12  Legendary   800 non-null    bool
dtypes: bool(1), int64(9), object(3)
memory usage: 75.9+ KB
```

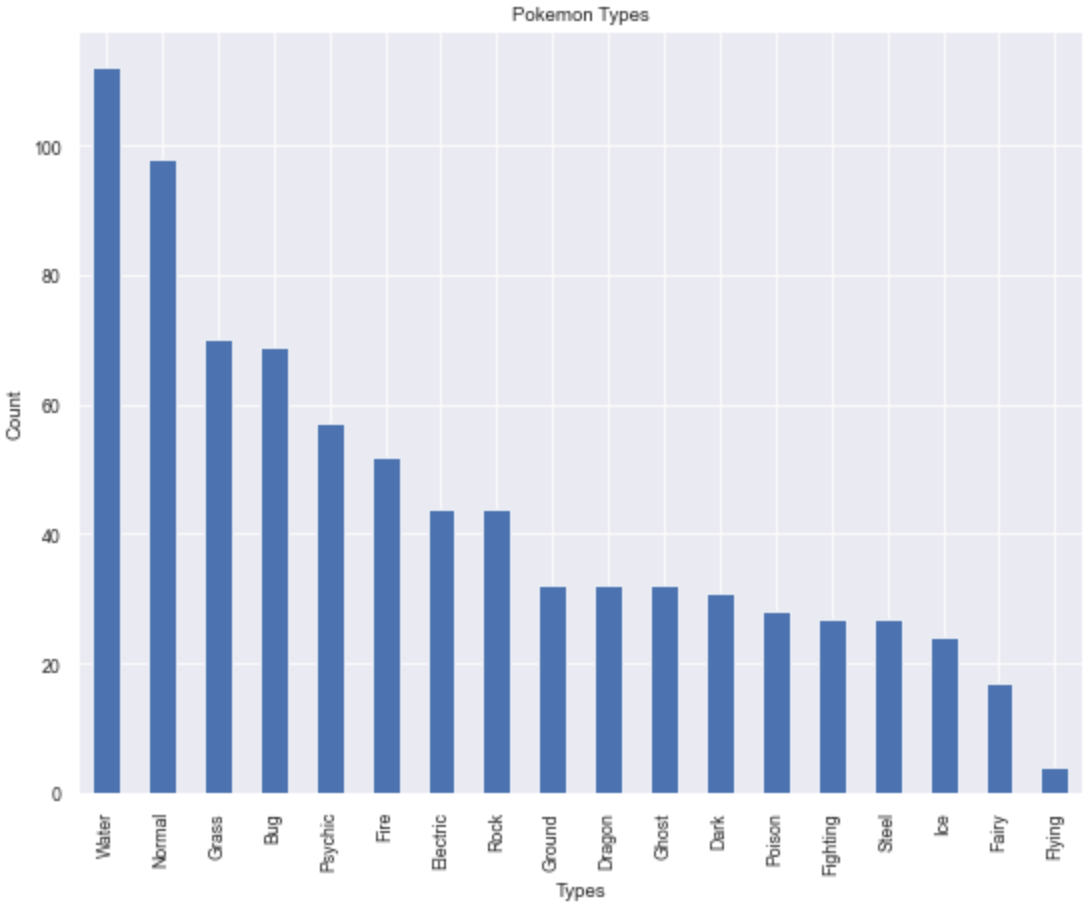
Out[4]:

	#	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
5	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	False
6	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
7	6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	False
8	6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	False
9	7	Squirtle	Water	NaN	314	44	48	65	50	64	43	1	False

Premiliminary Work on Data Exploration and Visualization

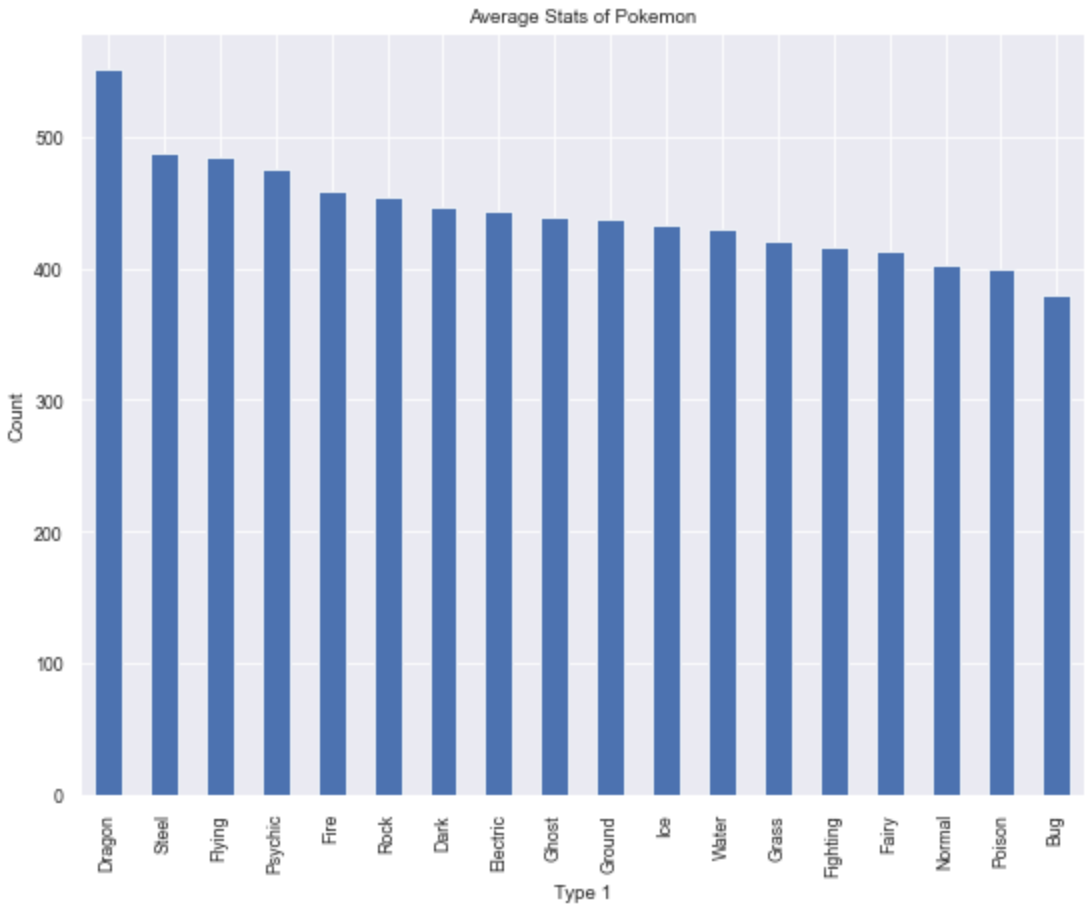
Showing the types of Pokemon in our datset.

Out[5]: Text(0.5, 1.0, 'Pokemon Types')



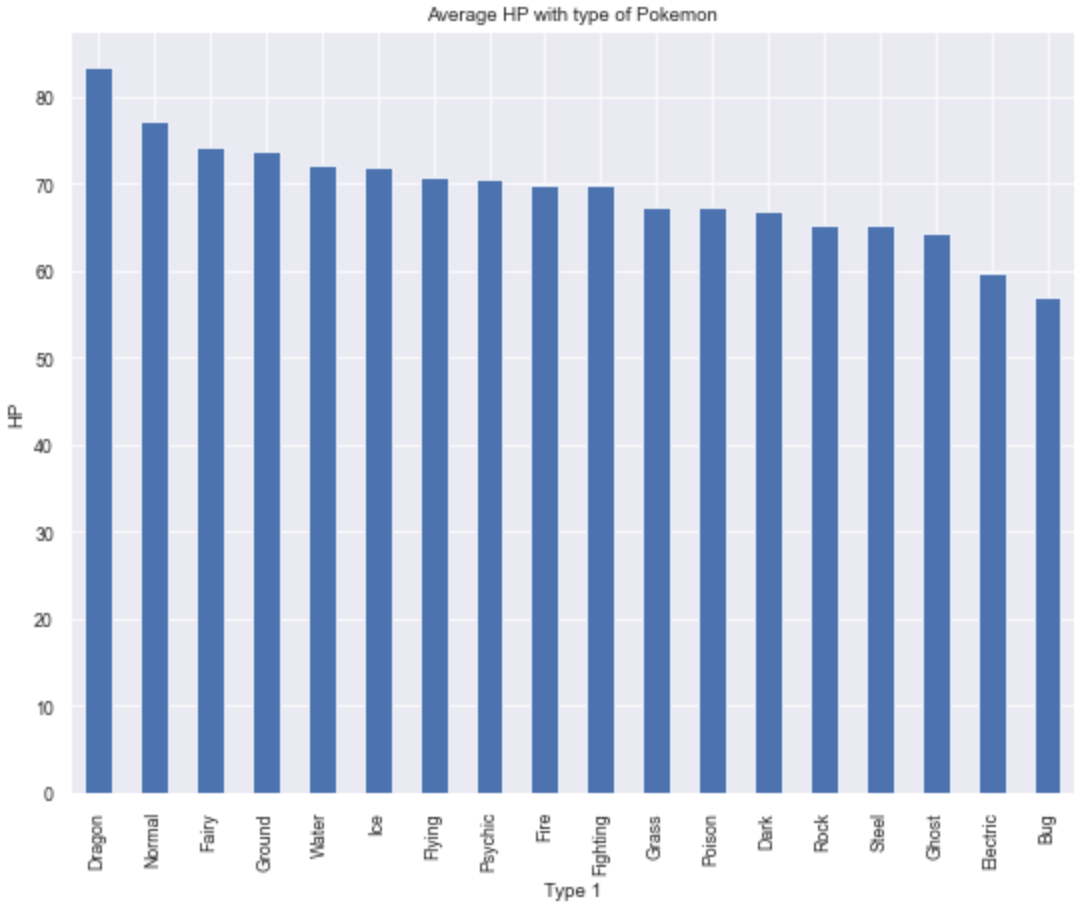
Showing the total average of the Pokemon's stats combined per type

Out[6]: Text(0.5, 1.0, 'Average Stats of Pokemon')



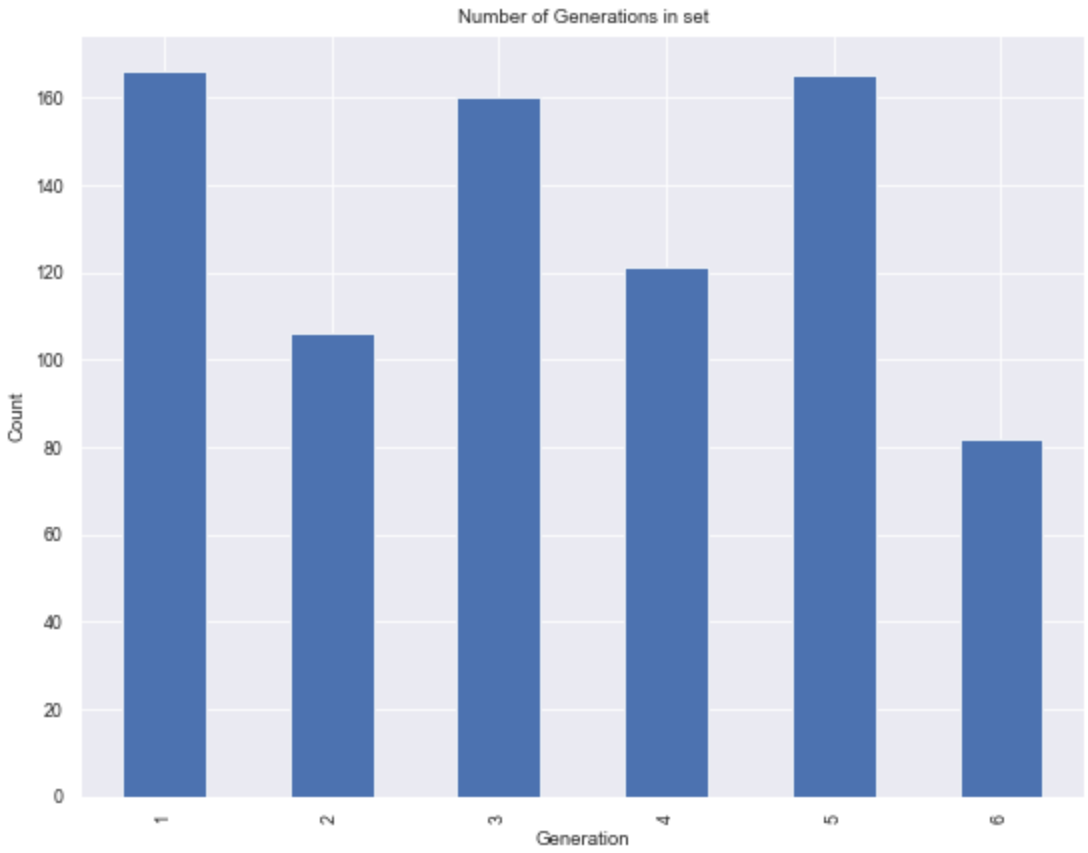
Displaying average HP per Pokemon Type

Out[7]: Text(0.5, 1.0, 'Average HP with type of Pokemon')



Displaying the amount of pokemon created in each generation within our dataset

Out[8]: Text(0.5, 1.0, 'Number of Generations in set')



Dropping "Total" and "Type 2" since they were not going to be relevant of our predictions and model.

Out[9]:

	#	Name	Type 1	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...
795	719	Diancie	Rock	50	100	150	100	150	50	6	True
796	719	DiancieMega Diancie	Rock	50	160	110	160	110	110	6	True
797	720	HoopaHoopa Confined	Psychic	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	80	160	60	170	130	80	6	True
799	721	Volcanion	Fire	80	110	120	130	90	70	6	True

800 rows × 11 columns

Preliminary Work on Machine Learning to make Predictions

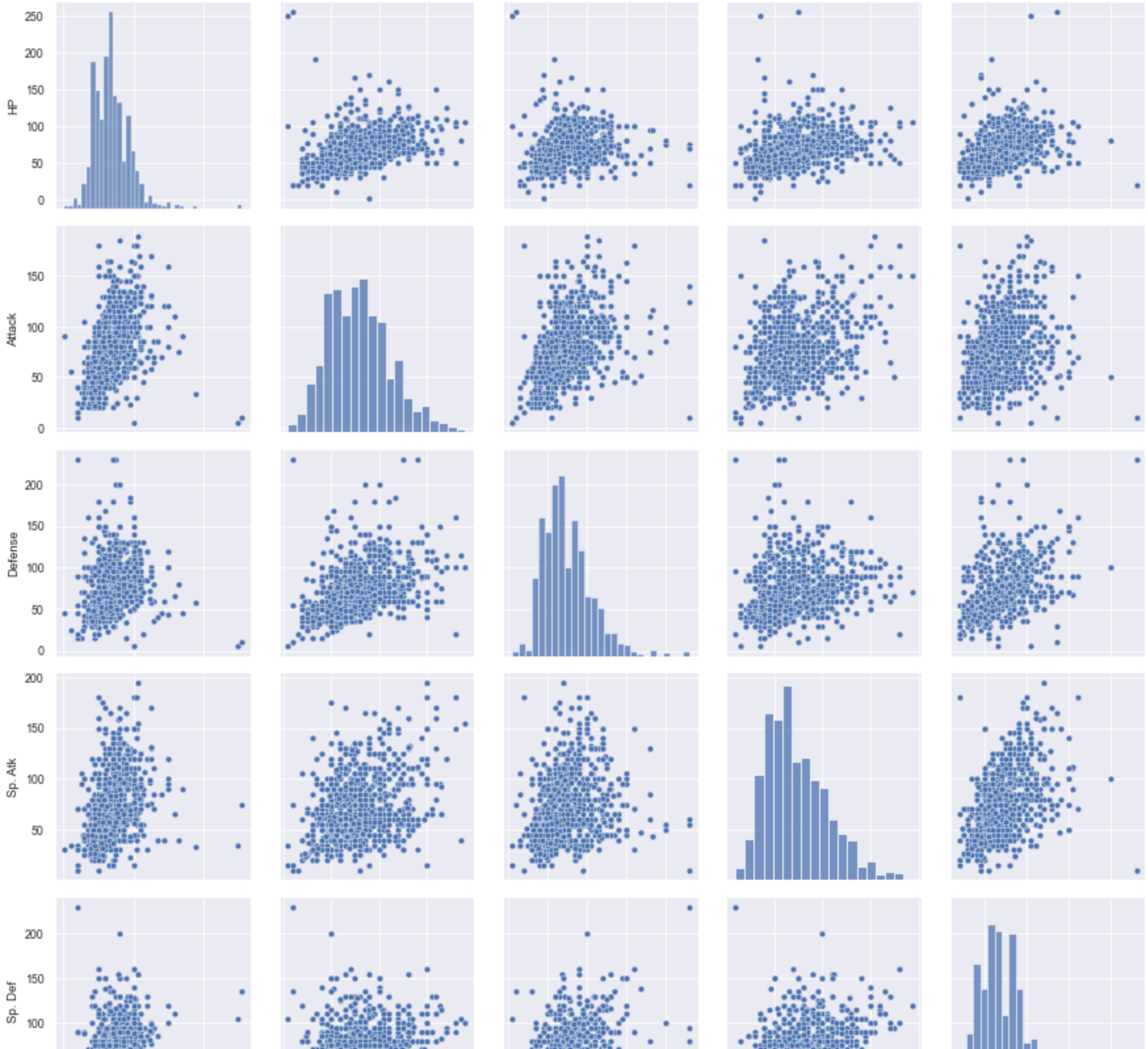
We will use numerical data from the "HP" and calculations (total's/averages) from the "Attack" , "Defense", "Sp. Attack", "Sp. Defense" to try to predict a Pokemon's Type as well as Generation and HP. We will be using kNN classification and regression algorithms for our predictions and will determine which one is the most accurate one as well as calculate which prediction will be the most accurate

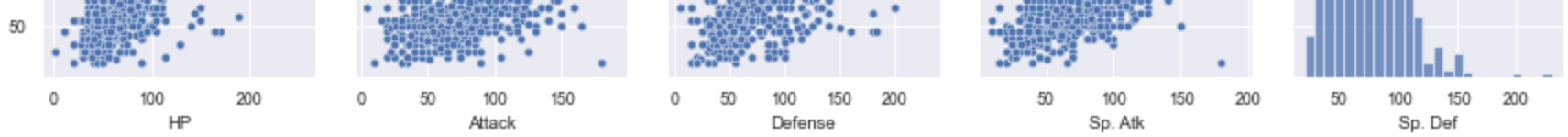
Linear Regression Prediction on HP

We will use a subset of the data set for our training and test data. We will keep an unscaled version of the data for one of the experiments that we will be conducting. Since our dataframe is not too large we will then conduct the prediction experiments with the entire dataframe to compare the differences.

This first set of plots we see the relationships between HP and some of the other features.

We will be analyzing the relationship that the Pokemon's statistics have. In order to to come up with a strategy to effectively predict a Pokemon's HP.


HP
Attack
Defense
Sp. Atk
Sp. Def

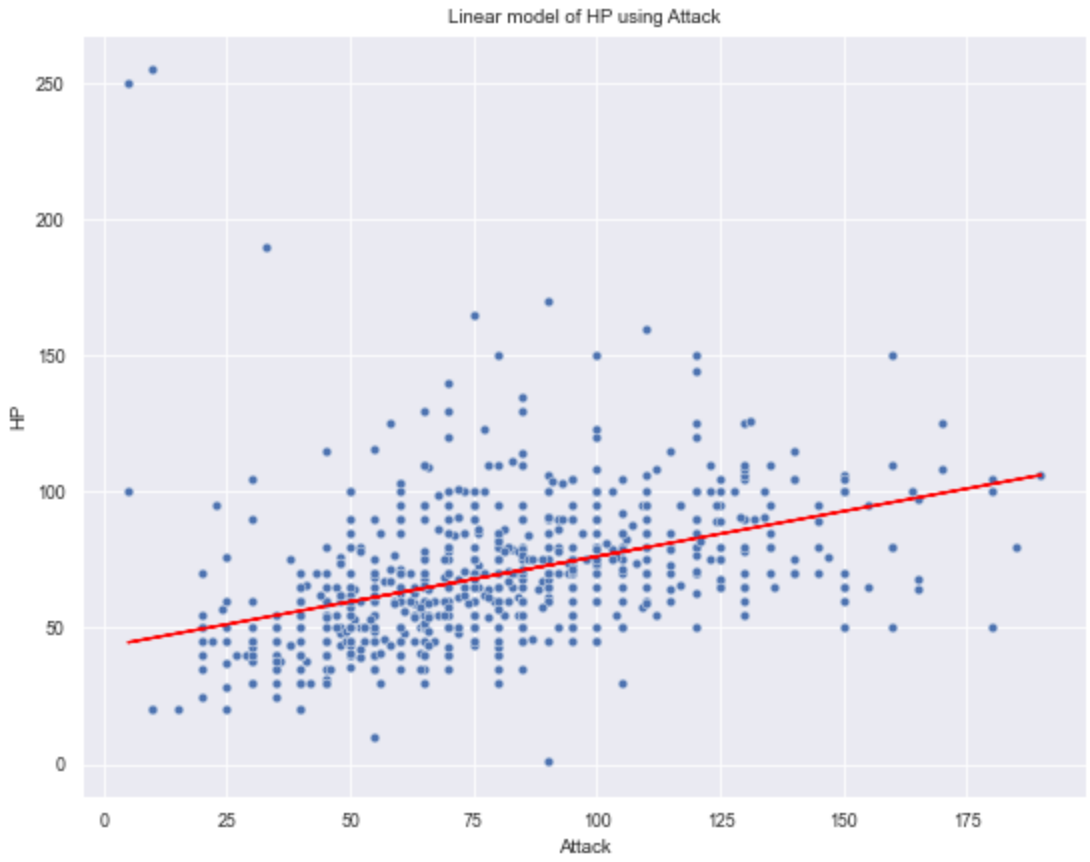


Linear regression with one feature "Attack"

To start off our linear regression predictions, we will only use one feature at first and see what is returned. Based off of the analysis we conduct, we will add more features.

Out[12]: `LinearRegression()`

In this graph we will be displaying the relationship between HP and Attack of a Pokemon. In doing so, we can visually see if there are some correlations between the two. We can see that the majority of the data follows the regression line with only a few outliers.



This model shows that for every increase on Attack units our prediction in HP will go up by .33 health points!

```
Intercept:  43.01
Coefficient for Attack:  0.33
r-squared value: 0.18
```

Linear regression with features: "Attack", "Defense", and "Sp. Atk"

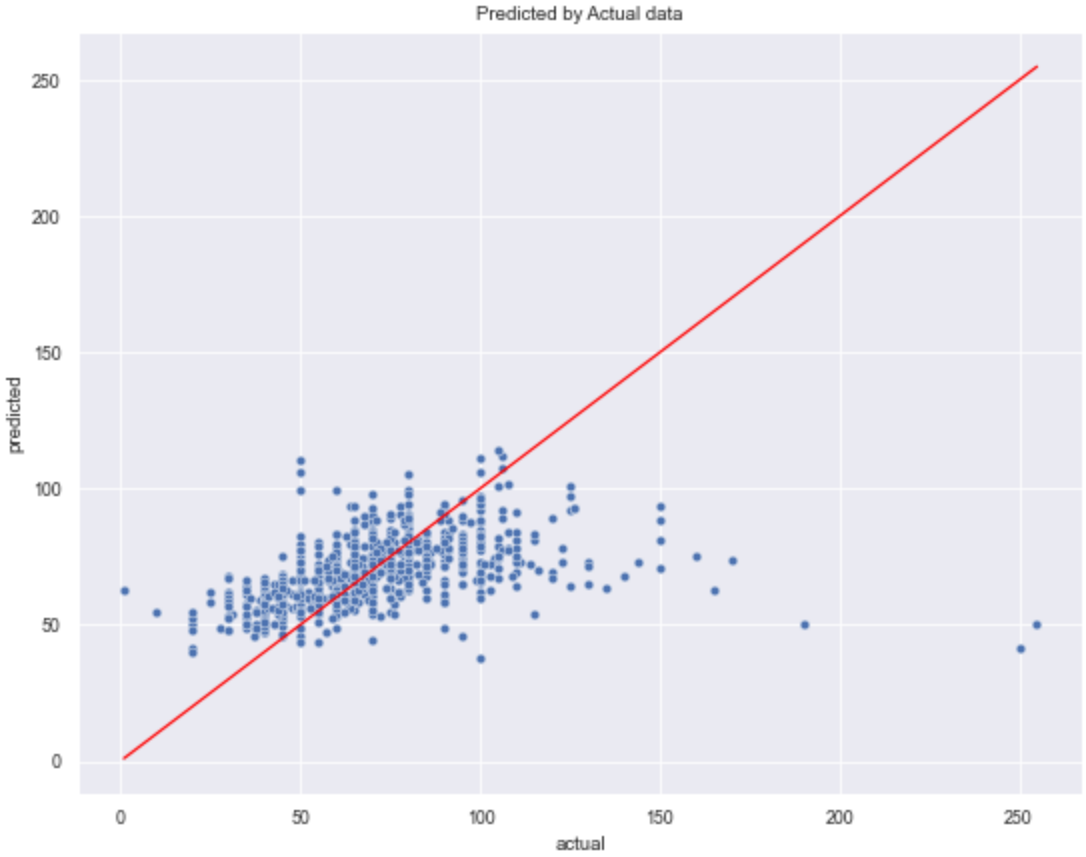
Continuing with our linear regression predictions, we willbe adding the features: *Attack*, *Defense*, and *Sp. Atk* and see what is returned. Based off of the analysis we conduct, we will look into adding a few more features until we obtain the reslults desired. As we can see, *Attack* still shows as the one that most affects HP change.

Out[15]:
```
LinearRegression()
```

```
Intercept:  33.89
Coefficient for Attack:  0.24
Coefficient for Defense:  0.04
Coefficient for Sp. Atk:  0.18
```

Displaying first prediction results with 3 features

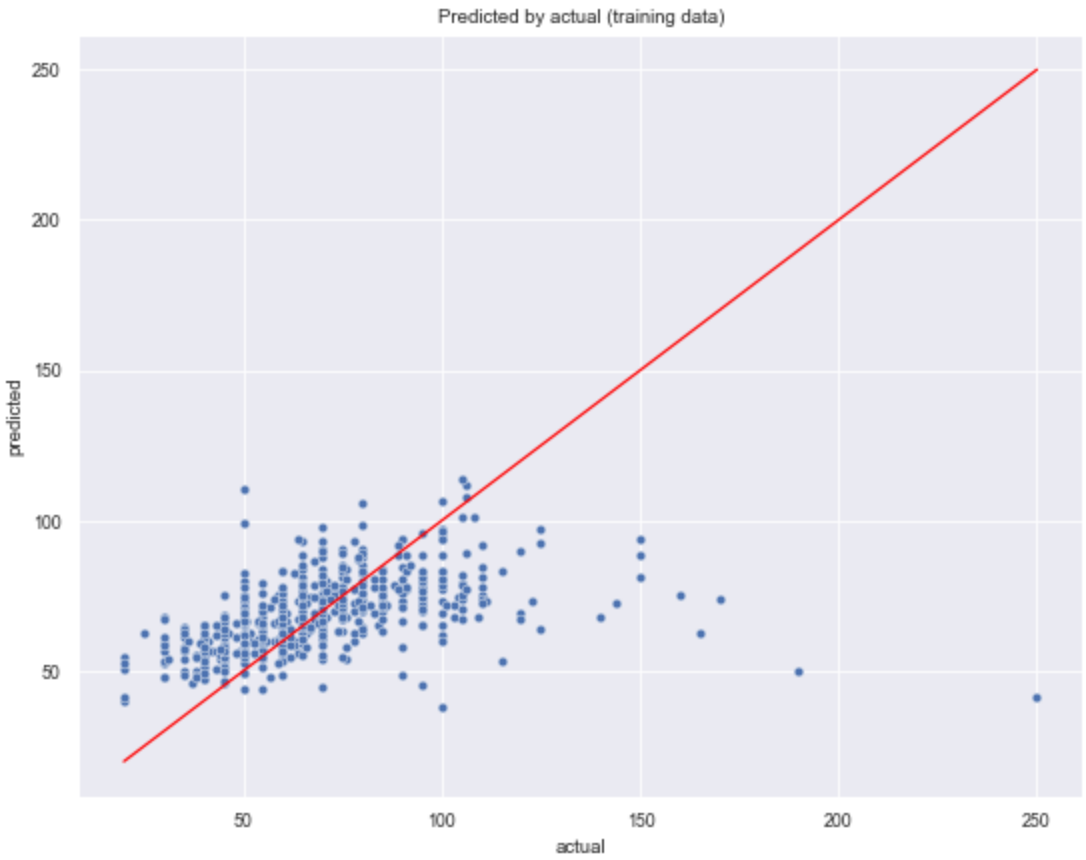
Here we are displaying the correlation that occurs between *Attack*, *Defense*, and *Sp. Atk*. With this linear regression graph, we are able to see that the majority of the data is getting closer together with less outliers. The more features being added, the more correlation that can be seen in the graph and the better the predictions are becoming.


Predicted by Actual data
predicted
actual
0
50
100
150
200
250

Adding "Generation" to the go along with previous features used

In this graph, we are able to see that the predicted versus the actual is realtively close to the linear regression with a few outliers spreading off to the right of the data.

Out[20]: `LinearRegression()`



Comparing train and test data

Squareroot of the Variance of the residuals result on test set

```
RMSE:  25.50
```

Going for broke and adding most features

Out[24]:
```
LinearRegression()
```

```
RMSE of reg4: 25.47
r-squared value of reg4: 0.2531
```

RMSE yileds better results

Plot on test data shows a better fit, still not the best

Next we scale data and test again

The following will be a series of scaling the data and retesting everyting in order to obtain the R-squared values and see what the actual percentage of the prediceted data was accurate.

Comparing New r-squared Value

We can see below that we are getting slightly better results after scaling the data.

```
r-squared value of regs: 0.2954
```

Use ALL the numeric features

Here we will be using every feature starting from *Attack* and ending on *Generation*. In doing so, we expect to see a greater prediction and more correlation between the features. One weird but neat thing that is observed below is that the Defense and the Speed both reduce as the HP goes up. This makes sense if you are visualizing a Pokemon fighting a bigger Pokemon. The chances of the smaller Pokemon being able to hit a bigger target makes sense. Evasiveness would be less for a bigger Pokemon in comparison to a smaller Pokemon causing the defense to be lower. Generation, Attack and SP.Def seem to be the largest contributors to change.

```
Intercept:  28.89
Coefficients:
  Attack 0.29
  Defense -0.04
  Sp. Atk 0.10
  Sp. Def 0.24
  Speed -0.09
  Generation 0.33
```

R-square and RMSE for all features

RMSE should be a bit lower still.

```
r-squared value of reg5: 0.30
RMSE of reg4: 24.44
```

Using PolynomialFeatures

Out[32]: (800, 27)

Test/train on all plynomial features!

We are able to see that we are getting a lower nunmber than previously which is a good thing to see.

```
RMSE of reg6: 22.85
```

Find single best polyfeature

Here, we are trying to get the best polyfeature in order to predict the Pokemon's HP more accurately. The numbers for the Best RMSE keep going down which is a good thing.

```
Best Feature: x0 x3, Best RMSE: 21.44
```

Best combinations of 5 features

```
num features: 1; rmse: 21.44
num features: 2; rmse: 21.30
num features: 3; rmse: 21.18
num features: 4; rmse: 21.11
num features: 5; rmse: 20.98
```

RMSE of this 5 best features

```
Test RMSE with 10 features: 29.08
```

After analyzing the results, the difference between using plynomial features and not is not substantial enough to justify the usage. We should stick to using all initial non-polynomial features.

Adding columns for Predicted HP and Difference

Here we are adding the column *Predicted HP* and *Predicted % Dif* in order to filter out the predictions which have a greater percentage of 0. This would mean that that specific Pokemon's HP was not predicted correctly. This will alllow us to gather a count of all the predictions that were incorrect and in return obtain a percentage of accuracy.

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Predicted HP	Predicted % dif
2	3	Venusaur	Grass	Poison	525	80.0	82	83	100	100	80	1	False	77.331939	2.248379
6	6	Charizard	Fire	Flying	534	78.0	84	78	109	85	100	1	False	73.600362	3.832431
11	9	Blastoise	Water	NaN	530	79.0	83	100	85	105	78	1	False	76.722386	1.940688
13	10	Caterpie	Bug	NaN	195	45.0	30	35	20	20	45	1	False	39.403733	8.649313
14	11	Metapod	Bug	NaN	205	50.0	20	55	25	25	30	1	False	38.718826	16.264805
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
789	713	Avalugg	Ice	NaN	514	95.0	117	184	44	46	28	6	False	70.547916	18.769741
791	715	Noivern	Flying	Dragon	535	85.0	70	80	97	80	123	6	False	66.477514	15.665325
792	716	Xerneas	Fairy	NaN	680	126.0	131	95	131	98	99	6	True	93.857039	18.587426
793	717	Yveltal	Dark	Flying	680	126.0	131	95	131	98	99	6	True	93.857039	18.587426
794	718	Zygarde50% Forme	Dragon	Ground	600	108.0	100	121	81	95	95	6	True	78.034764	20.382104

337 rows × 15 columns

Amount of pokemon with a percentage differance of more than 0%

Below we are able to see the total sum of Pokemon that had the HP incorrectly predicted.

```
Pokemon count with incorrect HP Predictions:  337
```

Running the prediction algo 1000 times and compute mean of pokemon with a percentage differance of more than 0%

Below we are able to see that even after trying the prediction algorithm 1000 times, we still got the same result which is not that great. Next we will try KNN Regression to see if that prediction algorithm could give us better results.

```
Pokemon count with incorrect HP Predictions over 1000 times: 337.0
```

KNN Regression Prediction on HP

First prediction will also only use the full dataframe. We will use partial percentage of the dataframe after.

```
(280, 4)
[[-0.25965729  0.15460605 -0.05489393  0.87258133]
 [-0.10993727 -0.01357458  0.55991809  1.06587466]
 [ 0.78838285 -0.18175521  2.25065114  0.29270133]]
```

Baseline performace

Could This mean that KNNRegression is more effective than LinearRegression? We already have a better result than linear progression without having to modify any models so far. We shall see if KNN Regression could give us the desired predictions or at least be more accurate.

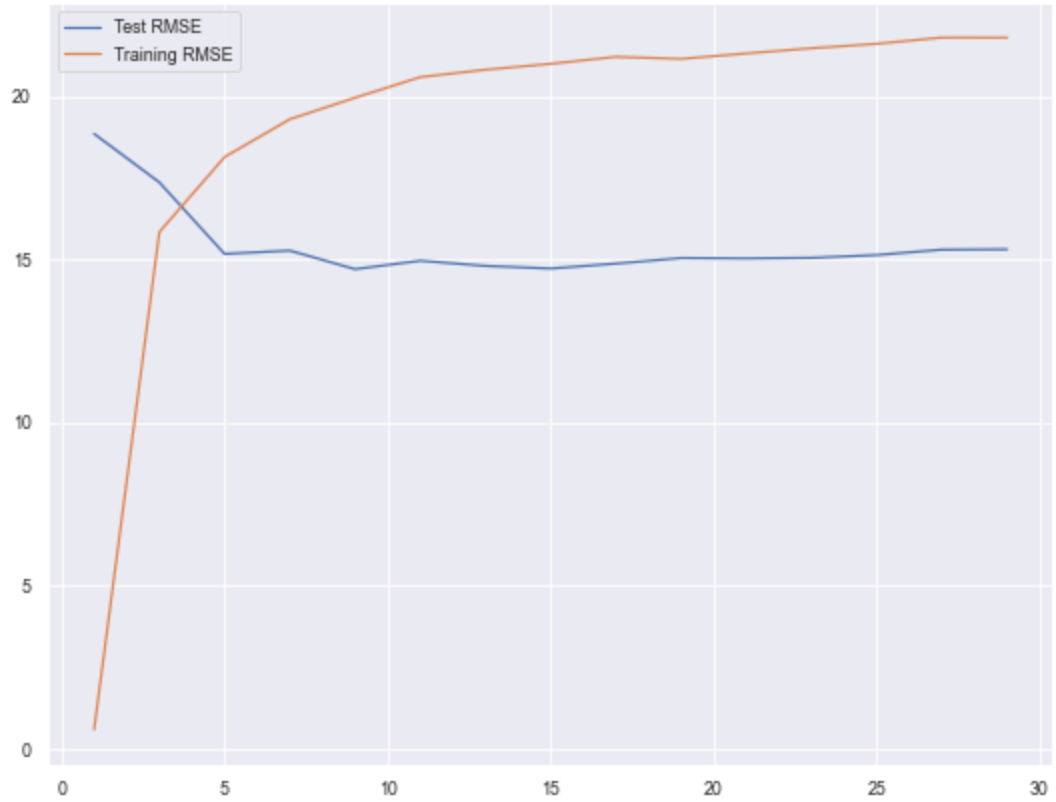
```
RMSE:  15.16431117239861
```

Changing the Ks detrminating which is the best

Here we are testing to see what the K value should be in order to obtain the best predictions and have a higher accuracy.

```
1  3  5  7  9  11  13  15  17  19  21  23  25  27  29  done
Test RMSE when k = 5: 15.2
Best k = 9, Best Test RMSE: 14.7
```

Now it's time to plot the test and training RMSE as a function of k, for all the k values in order to see how well the model will predict.



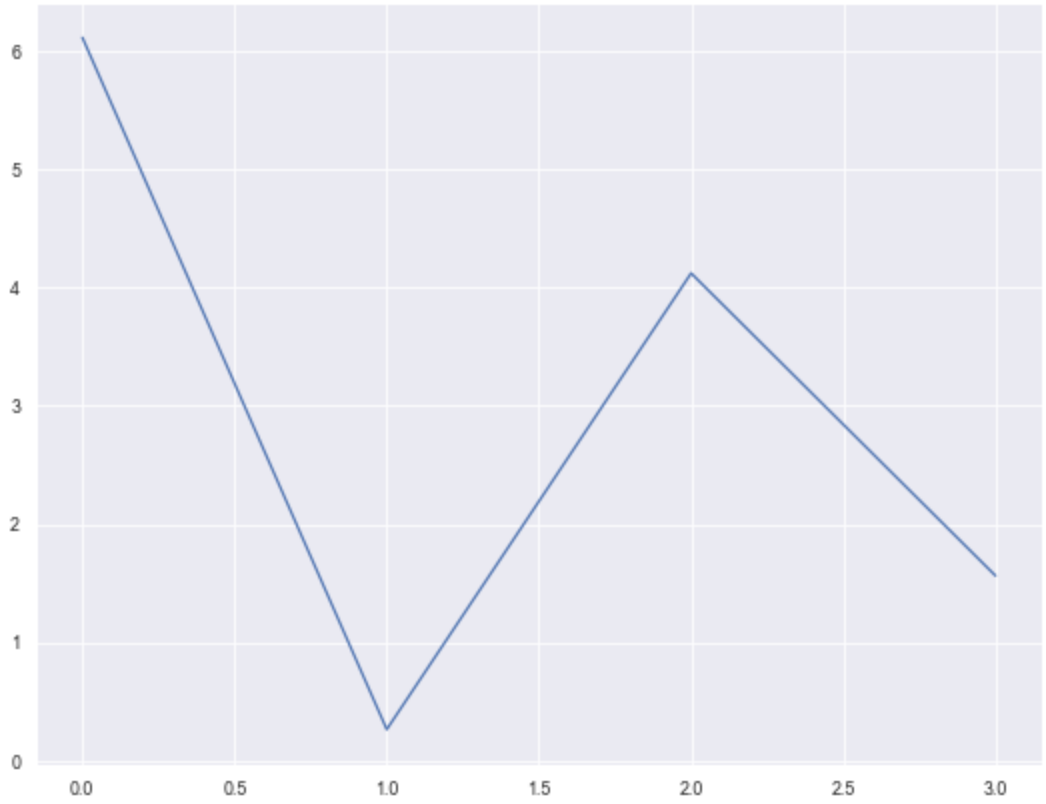
Changing the distance metric

Still trying to get the best RMSE results to have our predictions be more accurate. Changing the disance metric made little to no difference.

```
1  3  5  7  9  11  13  15  17  19  21  23  25  27  29  done
best k = 11, best test RMSE: 14.1
```

adding noise prediction (not neccesary)

```
0  1  2  3  done
```



Test with unscaled data

There is not much of a difference observed. Only from (0.6) - 4.1%

```
1  3  5  7  9  11  13  15  17  19  21  23  25  27  29  done
best k = 11, best test RMSE: 14.7
```

Testing different algorithms to obtain highest accuracy

```
ball_tree  kd_tree  brute  done
best algorithm = brute, best test RMSE: 14.8
```

Testing different weighting

```
uniform  distance  done
best weighting = uniform, best test RMSE: 14.8
```

Using all best hyperparameters

We can see below that using the best hyperparameters analyzed from before give us much better RMSE results!

```
RMSE:  12.496828050891105
```

Adding columns for Predicted HP and Difference

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Predicted HP	Predicted % dif
44	39	Jigglypuff	Normal	Fairy	270	115.0	45	20	45	25	20	1	False	102.101087	7.768064
45	40	Wigglytuff	Normal	Fairy	435	140.0	70	45	85	50	45	1	False	102.096269	19.839885
96	89	Muk	Poison	NaN	500	105.0	105	75	65	100	50	1	False	102.094763	1.861766
120	112	Rhydon	Ground	Rock	485	105.0	130	120	45	45	40	1	False	102.094820	1.861729
121	113	Chansey	Normal	NaN	450	250.0	5	5	35	105	50	1	False	102.100019	49.128044
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
741	673	Gogoat	Grass	NaN	531	123.0	100	62	97	81	68	6	False	102.094820	12.011198
769	699	Aurorus	Rock	Ice	521	123.0	77	72	99	92	58	6	False	102.094820	12.011198
792	716	Xerneas	Fairy	NaN	680	126.0	131	95	131	98	99	6	True	102.093816	13.502740
793	717	Yveltal	Dark	Flying	680	126.0	131	95	131	98	99	6	True	102.093816	13.502740
794	718	Zygarde50% Forme	Dragon	Ground	600	108.0	100	121	81	95	95	6	True	102.094249	3.713208

66 rows × 15 columns

Amount of pokemon with a percentage differance of more than 0%

Below we are checking the count of the Pokemon's HP that was not predicted correctly.

```
Pokemon count with incorrect HP Predictions:  66
```

Running the prediction algo 1000 times and compute mean of pokemon with a percentage differance of more than 0%

```
Pokemon count with incorrect HP Predictions with 1000 tries:  9.0
```

KNN Classification Prediction on Type 1

First we need to turn Type 1 into a int column

Here we are attempting to see if prediction would be better is we try to predict *Type 1* instead of a Pokemon's HP.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   #                800 non-null    int64
 1   Name             800 non-null    object
 2   Type 1           800 non-null    int64
 3   Type 2           414 non-null    object
 4   Total            800 non-null    int64
 5   HP               800 non-null    float64
 6   Attack           800 non-null    int64
 7   Defense          800 non-null    int64
 8   Sp. Atk          800 non-null    int64
 9   Sp. Def          800 non-null    int64
 10  Speed            800 non-null    int64
 11  Generation       800 non-null    int64
 12  Legendary        800 non-null    bool
 13  Predicted HP     800 non-null    float64
 14  Predicted % dif  800 non-null    float64
dtypes: bool(1), float64(3), int64(9), object(2)
memory usage: 88.4+ KB
```

Set predictors, split and scale the data.

```
[[ 0.19965661  0.88852169  1.29746158 -0.38370156  0.60101506  0.04920228
   1.53085826]
 [ 0.32379543  0.08877219 -0.10309665 -0.03880128  0.18910443  0.76487186
  -1.39901401]
 [-0.71069477 -1.0308771  -0.88118455 -0.97943841 -1.30875242 -0.53634555
  -0.2270651 ]
 [-0.50379673 -1.0308771  -0.72556697 -0.6658927  -0.18535978  0.43956751
  -0.81303955]
 [-0.09000065  0.72857179 -0.10309665  0.43151728  0.37633653  1.25282839
  -1.39901401]
 [-0.50379673 -0.8709272  -1.03680213 -0.82266555 -1.12152031  0.27691533
   0.94488381]
 [-0.91759281 -0.5510274  -0.88118455 -1.29298411 -0.93428821 -0.37369338
   0.94488381]
 [-0.71069477 -0.16714764 -1.19241971 -0.97943841 -1.30875242 -0.04838902
   0.94488381]
 [ 0.11689739 -1.190827   -0.72556697 -0.35234699 -0.7470561  -1.34960644
  -0.2270651 ]
 [-1.33138889  0.24872209 -0.88118455 -0.97943841 -1.30875242  0.27691533
  -1.39901401]]
```

Start training

Out[64]:
```
KNeighborsClassifier(algorithm='brute')
```

Prediction start

```
[ 2  2  2  2  2  2  2 11  2  2]
[3 3 3 3 6 6 6 6 6 1]
0.13125
```

Conclusion

The Pokemon-HP-Predictions project uses a modified regression model based on 8 numerical features: #, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, and Generation. The numeric features in this dataframe are not direclty co-related to the Type 1 field. Best to use KNeighborsRegressor and focus on predicting HP.

After running our predictions, we observed that there was an accuracy of **99.98875%** that the Pokemon's HP would be predicted correctly based on the stats of a Pokemon. That's a really high percantage being predicted correctly.