



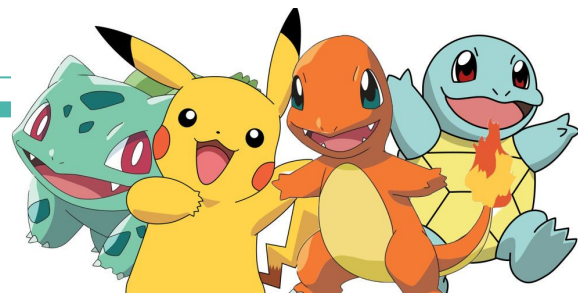
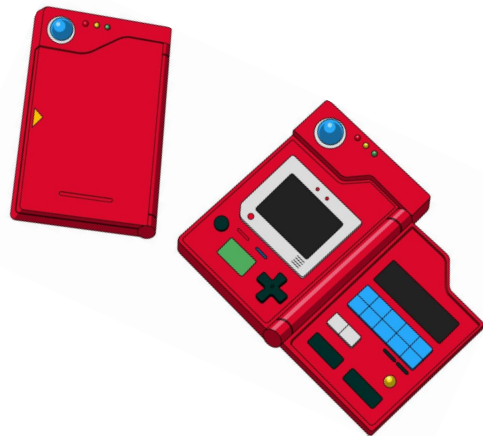
---

---

# POKÉMON

Predicting Pokemon types with  
stats from over 721 Pokémon

---



# Background Info

- Nintendo released Pokemon Red & Green for Gameboy Japan in 1996
- Each Pokemon has a type and specific base stat that made it strong or weak against other Pokemon
- 150 Pokemon in Gen I, with 15 types
- 802 Pokemon in Gen VII with 18 types
- Pokemon can have a primary and secondary type
- For this project, we will be looking at primary type only because with the combination possibilities, there are 146 type combinations

# Problem Statement and Hypothesis

By looking at and comparing the pokemon base stats (including HP, attack, special attack, defense, special defense, and speed) can we accurately predict the pokemon's primary type.



There are a lot of pokemon out there, and it'd be cool to predict its strengths and weaknesses by looking at base stats. I've been playing Pokemon since the original games came to USA in 1997, and still love it

- Note- we will only be looking at the game data. This will not include the trading cards, Pokemon Go, or the animated series

# About the Data

- I found this data on Kaggle <https://www.kaggle.com/abcsds/pokemon>
- Most of the data was sourced from data found on Bulbapedia, a Pokemon encyclopedia: [http://bulbapedia.bulbagarden.net/wiki/Main\\_Page](http://bulbapedia.bulbagarden.net/wiki/Main_Page)
- The data set contains information from Gen I to Gen VI, so I pulled my own testing data from Bulbapedia and used Pokemon from Gen VII as testing

# First Looks

There are 801 rows and 13 columns

```
In [18]: df.shape
```

```
Out[18]: (801, 13)
```

```
In [29]: df.head()
```

```
Out[29]:
```

	Pokemon_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

# Looking at Pokemon Types

```
In [30]: df.Type_1.value_counts()
```

```
Out[30]: Water      112  
Normal      98  
Grass       70  
Bug         69  
Psychic     57  
Fire        52  
Electric    44  
Rock        44  
Ground      32  
Ghost       32  
Dragon      32  
Dark        31  
Poison      28  
Fighting    27  
Steel       27  
Ice         24  
Fairy       17  
Flying      4  
Name: Type_1, dtype: int64
```

# Graphing

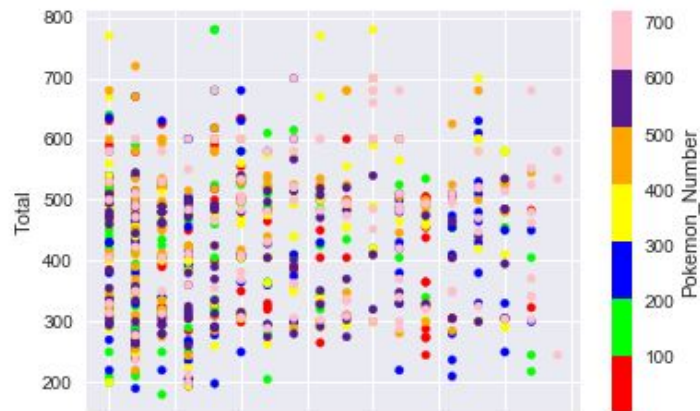
Renaming the types with a numeric value

```
In [34]: df['Type_1'] = df.Type_1.map({'Water':0, 'Normal':1, 'Grass':2, 'Bug':3, 'Psychic':4, 'Fire':5, 'Electric':6, 'Rock':7, 'Ground':8, 'Ghost':9, 'Dragon':10, 'Dark':11, 'Poison':12, 'Fighting':13, 'Steel':14, 'Ice':15, 'Fairy':16, 'Flying':17})
```

Scatter Plot: X = Type\_1, Y = Total (all stats added together) and c = Pokemon Number

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1187394d0>
```

You can see the 18 columns for the different types. You can also see that there is a high range with the TOTAL metric

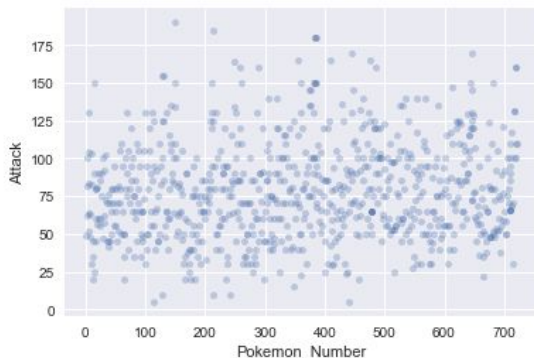


# Scatter Plots of Pokemon Stats

## Attack

```
In [50]: # add transparency  
df.plot(kind='scatter', x='Pokemon_Number', y='Attack', alpha=0.3)
```

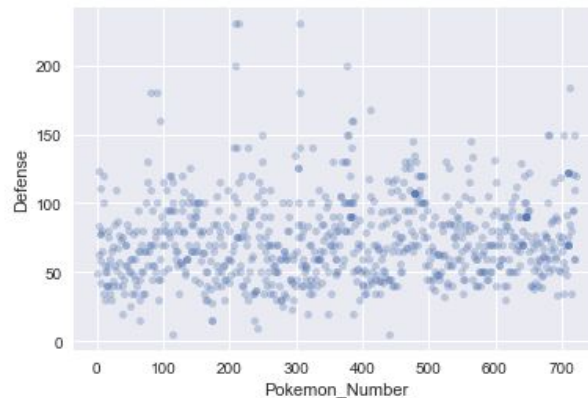
```
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x11874b990>
```



## Defense

```
In [51]: df.plot(kind='scatter', x='Pokemon_Number', y='Defense', alpha=0.3)
```

```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x118dab490>
```



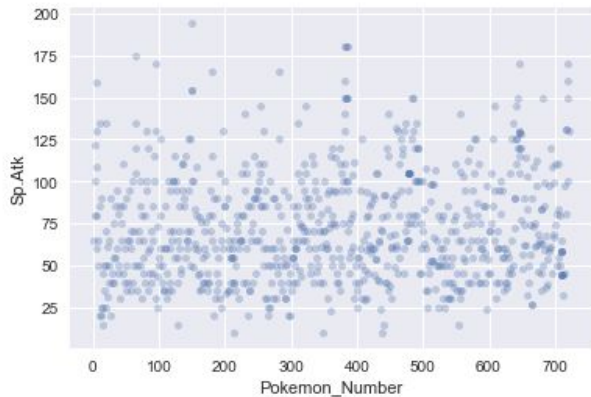


# Continued

## Special Attack

```
In [54]: df.plot(kind='scatter', x='Pokemon_Number', y='Sp.Atk', alpha=0.3)
```

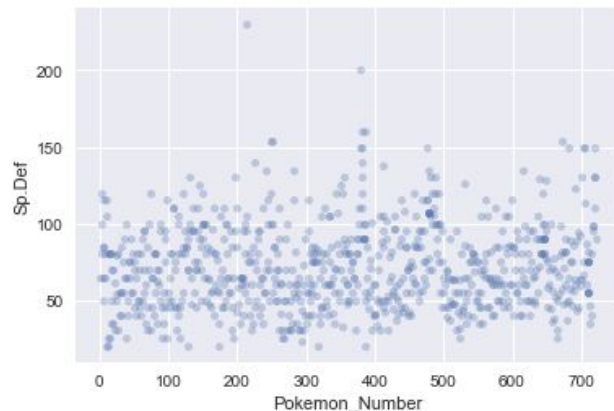
```
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x1191dba90>
```



## Special Defense

```
df.plot(kind='scatter', x='Pokemon_Number', y='Sp.Def', alpha=0.3)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x110b886d0>
```

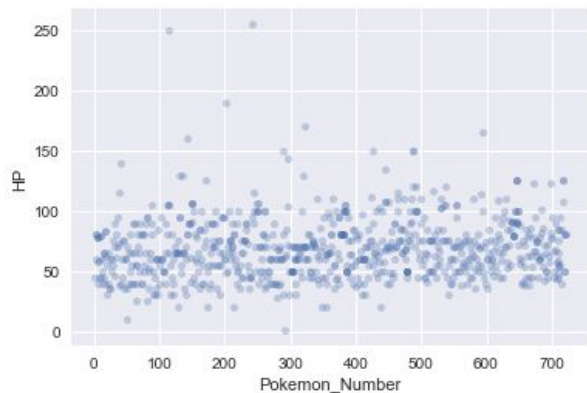


# Continued

## HP

```
In [52]: df.plot(kind='scatter', x='Pokemon_Number', y='HP', alpha=0.3)
```

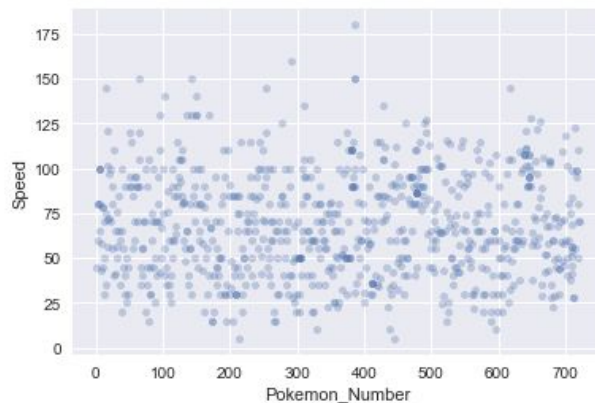
```
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x1191dba10>
```



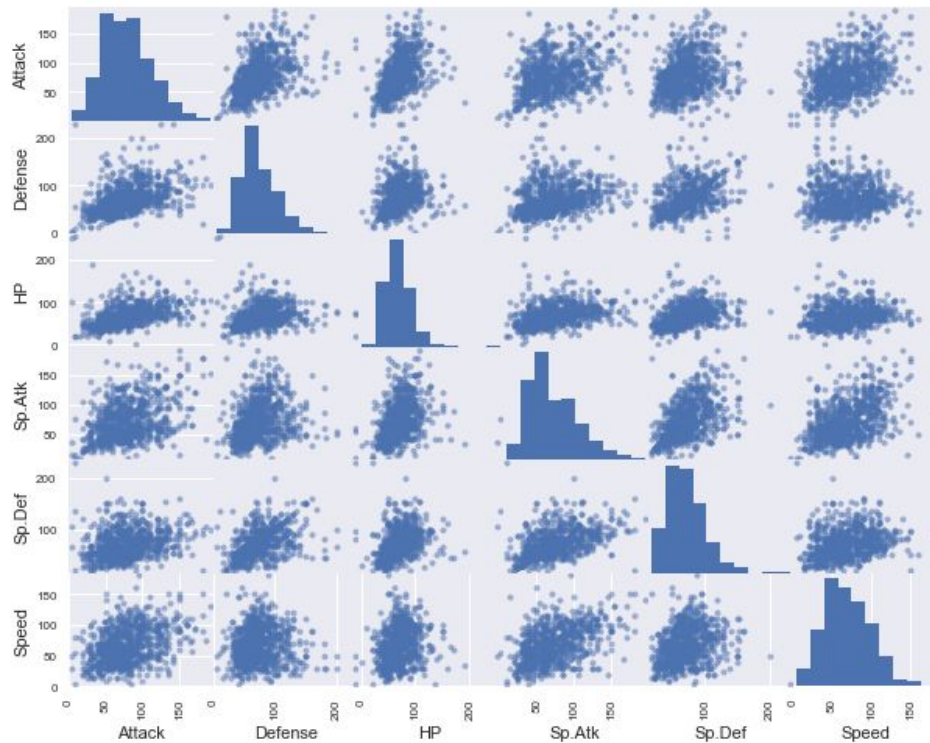
## Speed

```
df.plot(kind='scatter', x='Pokemon_Number', y='Speed', alpha=0.3)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x110b926d0>
```



# Scatter Matrix



# KNN to predict Type based on Stats- Step 1

```
#Starting a KNN to see features versus type to determine strength in type1  
# store feature matrix in "X"  
feature_cols = ['Attack', 'Defense', 'HP', 'Sp.Atk', 'Sp.Def', 'Speed']  
X = df[feature_cols]  
print (X)
```

	Attack	Defense	HP	Sp.Atk	Sp.Def	Speed
0	49	49	45	65	65	45
1	62	63	60	80	80	60
2	82	83	80	100	100	80
3	100	123	80	122	120	80
4	52	43	39	60	50	65
5	64	58	58	80	65	80
6	84	78	78	109	85	100
7	130	111	78	130	85	100
8	104	78	78	159	115	100
9	48	65	44	50	64	43
10	63	80	59	65	80	58
11	83	100	79	85	105	78
12	103	120	79	135	115	78
13	30	35	45	20	20	45

Creating our features (X)

# Step 1 Continued...

Creating our response vector, y and checking types

```
# store response vector in "y"  
y = df.Type_1  
print(y)
```

0	2
1	2
2	2
3	2
4	5
5	5
6	5
7	5
8	5
9	0
10	0

```
# check X's type  
print type(X)  
print type(X.values)
```

```
<class 'pandas.core.frame.DataFrame'>  
<type 'numpy.ndarray'>
```

```
# check X's shape (n = number of observations, p = number of features)  
print X.shape
```

```
(800, 6)
```

```
# check y's shape (single dimension with length n)  
print y.shape
```

```
(800,)
```

## Step 2. Import the Estimator

In this step, we're going to import KNN Classifier from Sklearn

```
#Step 2: Decide on the estimator you want to use and import that class  
from sklearn.neighbors import KNeighborsClassifier
```

## Step 3. Instantiate the estimator

```
#Step 3: "Instantiate" the "estimator"  
knn = KNeighborsClassifier(n_neighbors=1)  
type(knn)
```

```
sklearn.neighbors.classification.KNeighborsClassifier
```

```
print knn
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                    metric_params=None, n_jobs=1, n_neighbors=1, p=2,  
                    weights='uniform')
```

## Step 4. Fit the Model

```
#Step 4: Fit the model with data (aka "model training")  
knn.fit(X, y)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                    metric_params=None, n_jobs=1, n_neighbors=1, p=2,  
                    weights='uniform')
```

We're using stats from Generation I through Generation VI to fit the KNN model. For our predictions, we'll use Pokemon stats from Gen VII



## Step 5. Use the model to predict the response for a new observation

```
#Step 5: Use the model to predict the response for a new observation
#this data only goes through Gen VI, using pkmn from VII to have knn predict type_1
#Grass: Fomantis num 753,
#Attack55,Defense35,HP40,Sp.Atk50,Sp.Def35,Speed35....Looking for type #2 grass
#Electric: Xurkitree num 796
#Attack89,Defense71,HP 83,Sp.Atk173,Sp.Def71,Speed83....Looking for type #6 electric
#Psychic: Necrozma num 800
#Attack107,Defense101,HP97,Sp.Atk127,Sp.Def89,Speed79....Looking for type #4 psychic
new_observation = [[55, 35, 40, 50, 35, 35], [89, 71, 83, 173, 71, 83], [107, 101, 97, 127, 89, 79]]
knn.predict(new_observation)
```

Adding a new observation to see how it is classified

## Step 5 continued...

```
#adding a new pokemon- dragon type Jangmo-o num 782 - should be type 10  
#adding water, Pyukumuku, num 771 should be type 0  
X_new = [[55, 35, 40, 50, 35, 35], [89, 71, 83, 173, 71, 83], [107, 101, 97, 127, 89, 79], [55, 65, 45,  
45, 45, 45], [60, 130, 55, 30, 130, 5]]  
knn.predict(X_new)
```

```
array([3, 6, 5, 0, 9])
```

```
#predict probability  
knn.predict_proba
```

```
<bound method KNeighborsClassifier.predict_proba of KNeighborsClassifier(algorithm='auto', leaf_size=30, m  
etric='minkowski',  
    metric_params=None, n_jobs=1, n_neighbors=1, p=2,  
    weights='uniform')>
```

# Step 6. Evaluate Accuracy

```
#Step 6: Evaluate the error or accuracy of the model--measure accuracy/cross validation  
# instantiate the model (using the value K=5)  
knn = KNeighborsClassifier(n_neighbors=5)
```

```
# fit the model with data  
knn.fit(X, y)
```

```
# predict the response for new observations  
knn.predict(X_new)
```

```
array([1, 5, 5, 0, 7])
```

```
# calculate predicted probabilities of class membership  
knn.predict_proba(X_new)
```

```
array([[ 0. ,  0.2,  0. ,  0.2,  0. ,  0. ,  0.2,  0. ,  0. ,  0. ,  0. ,  
        0.2,  0.2,  0. ,  0. ,  0. ,  0. ,  0. ],  
       [ 0. ,  0.2,  0. ,  0. ,  0. ,  0.4,  0.2,  0. ,  0. ,  0. ,  0. ,  
        0.2,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],  
       [ 0. ,  0. ,  0. ,  0. ,  0. ,  0.6,  0.2,  0. ,  0. ,  0. ,  0. ,  
        0.2,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],  
       [ 0.2,  0. ,  0.2,  0.2,  0. ,  0. ,  0. ,  0.2,  0. ,  0. ,  0. ,  
        0.2,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ],  
       [ 0. ,  0. ,  0.2,  0. ,  0. ,  0. ,  0. ,  0.6,  0. ,  0.2,  0. ,  
        0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0.]])
```

# Conclusion about KNN Model

- It predicts sometimes, but it's not very accurate.
- We can use different models to increase the accuracy
- I will start working with decisions trees, clustering, and RandomForests to see if the accuracy will improve.

