Analyzing Pokedex for Dream Team Selection and Building Legendary Pokemon Classifier

Table of Contents

Abstract

As a couple of people who've grown up playing or even watching Pokémon, we'd always root for different Pokémon to win. If we weren't spending our time watching the cartoons, we'd be spending our time on the Gameboy beating Pokémon Trainers on those games with our team of Pokémon. We've done it multiple times with different combinations of Pokémon in our team. What if there was a way to build a dream team?

A team that could be undefeatable at any level in the game. This could be achieved by analyzing various factors such as hitpoints (HP), attack, defense, special levels, move sets, rarity of the Pokémon, etc.

Introduction

Pokémon are creatures of all shapes and sizes who live in the wild or alongside humans. For the most part, Pokémon do not speak except to utter their names. Pokémon are raised and commanded by their owners (called "Trainers"). During their adventures, Pokémon grow and become more experienced and even, on occasion, evolve into stronger Pokémon. There are currently more than 700 creatures that inhabit the Pokémon universe. In most games, the player takes on the role of a young Trainer whose journey involves traveling from place to place, catching and training Pokémon, and battling against other Trainers' Pokémon on a quest to become the Pokémon League Champion. An additional goal is to catch and catalog all of the many Pokémon within the game's world. Despite this battling aspect of the games, the Pokémon games avoid explicit violence; Pokémon never die during the course of the game. Trainers are invited to take part in many peripheral activities when playing, including talent and beauty contests, tournaments, and fishing (for Pokémon, of course!).
Our goal is to form a dream team for this young trainer with the use of statistics which are present in a dataset for each of these creatures. And form an analysis which could help this 'Trainer' be the best trainer there ever is.

Dataset Description

The dataset for this project is the Pokemon dataset from a lot of generations of the Pokemon series. The attributes pertaining to this dataset are :

- name: The English name of the Pokemon (string)
- japanese_name: The Original Japanese name of the Pokemon (string)
- pokedex_number: The entry number of the Pokemon in the National Pokedex (Integer)
- percentage_male: The percentage of the species that are male. Blank if the Pokemon is genderless. (Decimal)
- type1: The Primary Type of the Pokemon (string)
- type2: The Secondary Type of the Pokemon (string)
- classification: The Classification of the Pokemon as described by the Sun and Moon Pokedex (string)
- height_m: Height of the Pokemon in metres (Decimal)
- weight_kg: The Weight of the Pokemon in kilograms (Decimal)
- capture_rate: Capture Rate of the Pokemon (string)
- Base *egg* steps: The number of steps required to hatch an egg of the Pokemon (Integer)
- abilities: A stringified list of abilities that the Pokemon is capable of having (string)
- experience_growth: The Experience Growth of the Pokemon (Integer)
- base_happiness: Base Happiness of the Pokemon (Integer)
- against_?: Eighteen features that denote the amount of damage taken against an attack of a particular type (Decimal)
- hp: The Base HP of the Pokemon (Integer)
- attack: The Base Attack of the Pokemon (Integer)
- defense: The Base Defense of the Pokemon (Integer)
- sp_attack: The Base Special Attack of the Pokemon (Integer)
- sp_defense: The Base Special Defense of the Pokemon (Integer)
- speed: The Base Speed of the Pokemon (Integer)
- generation: The numbered generation which the Pokemon was first introduced (Integer)
- is_legendary: Denotes if the Pokemon is legendary. (Integer)

This dataset contains 21 decimal attributes, 13 integer attributes and 7 string attributes with 802 row instances.
Link to dataset : https://www.kaggle.com/rounakbanik/pokemon

Problem Description

Our problem description can be put together in a series of questions that need to be answered about the dataset. Pokémon games essentially require for you to form a team of 6 Pokemon which you would use to play the entire game. The selection of these 6 Pokemon need to be strategic, and cannot be random for a successful completion of the game. With the help of this dataset which provides information about the Pokemon, we can make a more informed decision about the selection of these Pokemon.

Additionally, we can also create a successful classifier to determine whether a Pokemon is a legendary Pokemon. A legendary Pokemon is a Pokemon that is rare, hard to catch and is one of a kind. It's hard to determine whether a Pokemon is legendary or not.

The goal of the project is to solve the problem of having to select Pokemon to continue with a successful completion of Generation 1 Pokemon games (Pokemon Red, Pokemon Blue, Pokemon Yellow etc). We can ask the following questions :

1. Can we create a dream team of 6 Pokemon? If so, what would they be?
2. Would there be specific types that are stronger than the others?
3. What are the most common types of Pokemon?
4. Would a Pokemon's height or weight play a factor in their strength?
5. Would it be possible to tell if a Pokemon is legendary just from a couple of attributes?

Solution Approach/Methodology

Our problem would be to analyze various attributes in the dataset, to come up with possibly a grand solution to our problem mentioned in the abstract. This would firstly involve analyzing the attributes in the dataset to deduce the most helpful attributes to determine the strength of a pokemon against other pokemon types in general. We would need to draw up various plots and visualizations to analyze this. We would then proceed to using these attributes to find out what the weight of each attribute might be, and this would involve further analysis of the dataset. This allows us to determine if the stats are important in the selection of the Pokemon for the dream team.

The next independent section of the project is to build a classifier for the Legendary Pokemon. Firstly, we need to figure out if there might be a correlation between the attributes, and the legendary pokemon flag (which determines whether a Pokemon is legendary). We can use a

Logistic Regression, and Decision Tree Classifier to determine which might be the best approach by checking the precision value for both the classifiers.

Code

1. Loading the dataset

We first start off the project by loading the dataset which shows us that the dataset consists of 801 instances with 41 instances which means that there is information about 801 instances with 41 attributes about them. The attributes are the same as mentioned above describing the dataset.

Loading the dataset (aka. the Pokédex)

```
In [256]: pokedex = pd.read_csv('pokemon.csv')
          pokedex.head()
```

Out[256]:

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	against_fire	against_flying	against_ghost	against_gra
0	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0	2.0	1.0	0.
1	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0	2.0	1.0	0.
2	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0	2.0	1.0	0.
3	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5	1.0	1.0	0.
4	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5	1.0	1.0	0.

As we can see, we've got 801 instances/pokemon with about 41 attributes for each pokemon.

```
In [257]: pokedex.shape
Out[257]: (801, 41)
```

2. Cleaning/Tidying the dataset

We then continue to clean the dataset where we would remove any duplicates in names if any, because of the similarity of Pokemon names. Another task we perform is convert NaN attribute values to 'None' string types for type2 attributes of an instance. The reason we do this is that some instances don't have type2 and have NaN as a filler value. To ensure uniformity of data, we would convert them to 'None' string type.

We can see that clearly in the following image.

Cleaning the data

```
In [262]: pokedex.drop_duplicates('name', keep='first', inplace=True)

In [263]: pokedex.shape
Out[263]: (801, 42)
```

We have attributes Type 1 and Type 2. All Pokémon don't necessarily have 2 types. For eg : Charizard can be of Fire and Flying type, however Squirtle is of Water type. The data for Type 2 for squirtle is NaN. So this needs to be set to None.

```
In [264]: pokedex.loc[pokedex['name'] == 'Squirtle']
```

Out[264]:

eight_m	hp	japanese_name	name	percentage_male	pokedex_number	sp_attack	sp_defense	speed	type1	type2	weight_kg	generation	is_legendary	id
0.5	44	Zenigameゼニガメ	Squirtle	88.1	7	50	64	43	water	NaN	9.0	1	0	7

```
In [265]: pokedex['type2'].fillna(value='None', inplace=True)

In [266]: pokedex.loc[pokedex['name'] == 'Squirtle']
```

Out[266]:

eight_m	hp	japanese_name	name	percentage_male	pokedex_number	sp_attack	sp_defense	speed	type1	type2	weight_kg	generation	is_legendary	id
0.5	44	Zenigameゼニガメ	Squirtle	88.1	7	50	64	43	water	None	9.0	1	0	7

For a logical separation of data, we split the attributes into two parts - the general data about a Pokemon, and it’s stats. This would help us keep the visualizations, and let us keep track of the correlations much better.

We can divide the set of attributes and main data into two sections - based on ID.

The first section would be called the pokedata which would give the details about the pokemon - this would be the name, types, generation of pokemon and whether the pokemon is legendary or not.

The section of data would include the statistics of the pokemon -

```
pokedata = pokedex[['id', 'name', 'type1', 'type2', 'generation', 'is_legendary', 'capture_rate']]
pokestats = pd.merge(pokedex, pokedata,on='id').loc[:, ['id', 'hp', 'attack', 'defense', 'sp_attack', 'sp_defense',
```

```
pokedata.head(10)
```

	id	name	type1	type2	generation	is_legendary	capture_rate
0	1	Bulbasaur	grass	poison	1	0	45
1	2	Ivysaur	grass	poison	1	0	45
2	3	Venusaur	grass	poison	1	0	45
3	4	Charmander	fire	None	1	0	45
4	5	Charmeleon	fire	None	1	0	45
5	6	Charizard	fire	flying	1	0	45
6	7	Squirtle	water	None	1	0	45
7	8	Wartortle	water	None	1	0	45
8	9	Blastoise	water	None	1	0	45
9	10	Caterpie	bug	None	1	0	255

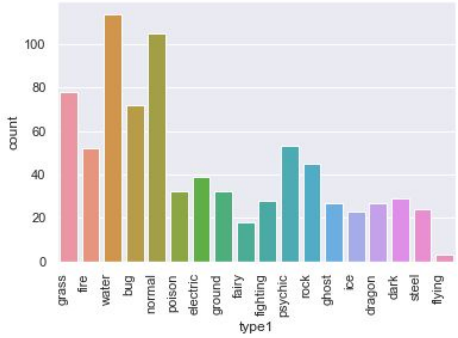
```
pokestats.head()
```

	id	hp	attack	defense	sp_attack	sp_defense	speed	height_m	weight_kg	against_all	base_total
0	1	45	49	49	65	65	45	0.7	6.9	19.25	318
1	2	60	62	63	80	80	60	1.0	13.0	19.25	405
2	3	80	100	123	122	120	80	2.0	100.0	19.25	625

If need be, these splits can be merged on their id to retrieve the original dataset.

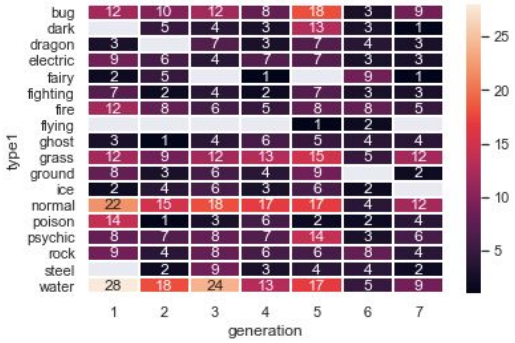
3. Exploring data through visualizations

In [271]:
```
sns.set(style="darkgrid")
ax = sns.countplot(x="type1", data=pokedata)

ax.set_xticklabels(ax.get_xticklabels(), rotation=90, ha="right")
plt.figure(figsize=(50, 23))
plt.tight_layout()
plt.show()
```



The following countplot plot shows the distribution of the Pokemon based on their type1 which is their primary type in the entire dataset. A more interesting visualization would be to see how the pokemon types are distributed among generations.

```
sns.heatmap(
    pokedata.groupby(['type1', 'generation']).size().unstack(),
    linewidths=1,
    annot=True
);
```



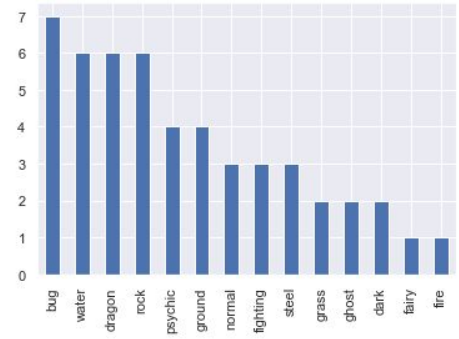
We then proceeded to see what Pokemon had the highest attack, defense, and speed. This was done with the help of the following code snippet for attack. To keep this report concise, I've not repeated this same snippet for defense and speed. For ease of visualization, I also added a bar graph showing the types of Pokemon with the highest attack using a sample of the top 50 Pokemon with the highest attack stats.

```
df1 = pokestats.sort_values(['attack','sp_attack'],ascending=False).groupby('attack')
df1 = df1.apply(pd.DataFrame)

attack_ordered_pokemon = pd.merge(df1, pokedata, on='id').head(50)

attack_ordered_pokemon.head(10)
```

	id	hp	attack	defense	sp_attack	sp_defense	speed	height_m	weight_kg	against_all	base_total	name	type1	type2	generation	is_legendar
0	214	80	185	115	40	105	75	1.5	54.0	21.50	600	Heracross	bug	fighting	2	
1	798	59	181	131	59	31	109	0.3	0.1	16.25	570	Kartana	grass	steel	7	
2	384	105	180	100	180	100	115	7.0	206.5	20.25	780	Rayquaza	dragon	flying	3	
3	383	100	180	160	150	90	90	3.5	950.0	19.00	770	Groudon	ground	None	3	
4	445	108	170	115	120	95	92	1.9	95.0	20.50	700	Garchomp	dragon	ground	4	
5	354	64	165	75	93	83	75	1.1	12.5	17.00	555	Banette	ghost	None	3	
6	409	97	165	60	65	50	58	1.6	102.5	21.00	495	Rampardos	rock	None	4	
7	475	68	165	95	65	115	110	1.6	52.0	20.00	618	Gallade	psychic	fighting	4	
8	248	100	164	150	95	120	71	2.0	202.0	23.00	700	Tyranitar	rock	dark	2	
9	720	80	160	60	170	130	80	NaN	NaN	21.00	680	Hoopa	psychic	ghost	6	

```
pd.value_counts(attack_ordered_pokemon['type1']).plot.bar()
```

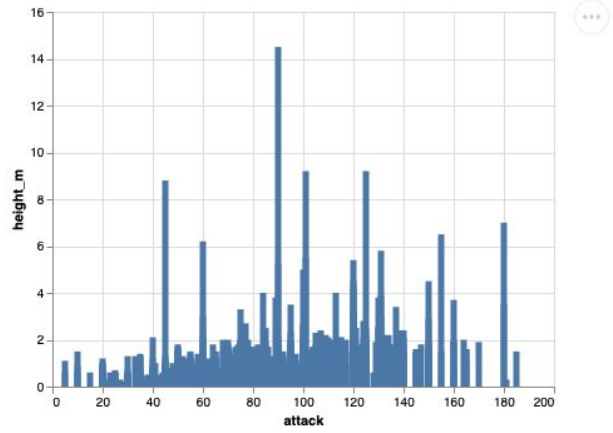
```
<AxesSubplot:>
```



We also try to correlate the height and weight of a Pokemon with the attack, defense, speed stats of a Pokemon. This was done with the help of the following code snippet for attack vs height_m. To keep this report concise, I've not repeated this same snippet for defense and speed.

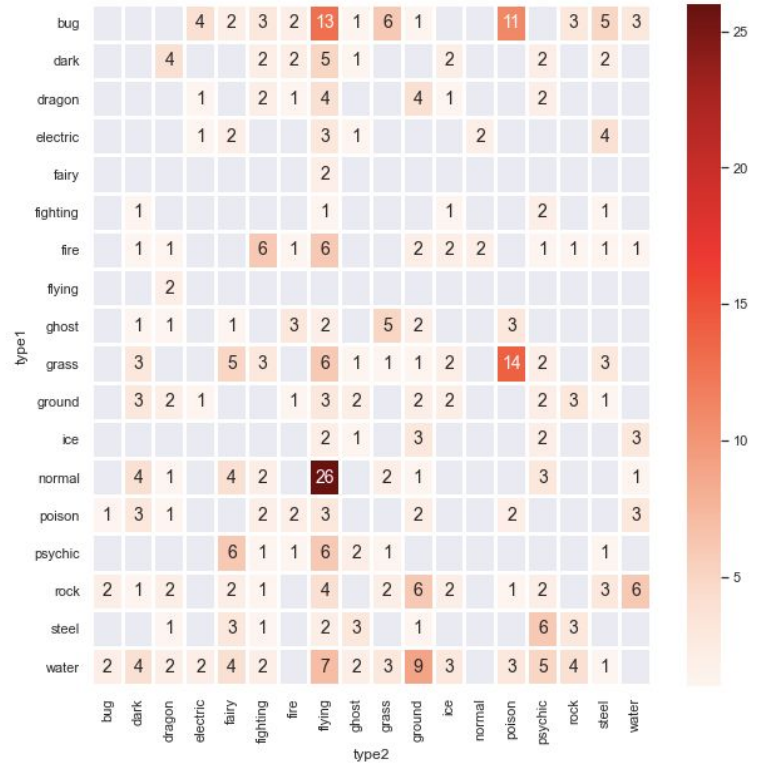
Correlation of height and weight with base stats (attack, defense, etc)

```
alt.Chart(pokestats).mark_bar().encode(
    x='attack',
    y='height_m'
)
```



We move onto more interesting results which determines what the most common dual type Pokemon would be with the helpful of this very attractive heatmap. Seaborn allows us to do that.

```
dualtype = pokedata[pokedata['type2'] != 'None']
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(dualtype.groupby(['type1', 'type2']).size().unstack(),linewidths=2,annot=True,annot_kws={'size': 14},
    cmap="Reds");
```



Now from all of the information above, we can tell that the strongest pokemon is basically determined by the base stats of the Pokemon which is basically the sum of the attack, defense, special attack, special defense, speed of the Pokemon, and their values of attacks against all the other types of Pokemon. So we sort the list of Pokemon based on this attribute.
This would allow us to get a list of Pokemon based on their strength, which would be easy to tell whether we need to add the following Pokemon onto our dream team.

```
pokestats['grand_total'] = pokestats['base_total'] + pokestats['against_all']

powerful_pokemon = pd.merge(pokedata, pokestats, on='id').sort_values('grand_total', ascending=False)
powerful_pokemon.head(10)
```

	id	name	type1	type2	generation	is_legendary	capture_rate	hp	attack	defense	sp_attack	sp_defense	speed	height_m	weight_kg	agair
383	384	Rayquaza	dragon	flying	3	1	45	105	180	100	180	100	115	7.0	206.5	
149	150	Mewtwo	psychic	None	1	1	3	106	150	70	194	120	140	2.0	122.0	
382	383	Groudon	ground	None	3	1	3	100	180	160	150	90	90	3.5	950.0	
381	382	Kyogre	water	None	3	1	3	100	150	90	180	160	90	4.5	352.0	
492	493	Arceus	normal	None	4	1	3	120	120	120	120	120	120	3.2	320.0	
717	718	Zygarde	dragon	ground	6	1	3	216	100	121	91	95	85	5.0	284.6	
247	248	Tyranitar	rock	dark	2	0	45	100	164	150	95	120	71	2.0	202.0	
645	646	Kyurem	dragon	ice	5	1	3	125	120	90	170	100	95	3.0	325.0	
380	381	Latios	dragon	psychic	3	1	3	80	130	100	160	120	110	2.0	60.0	
379	380	Latias	dragon	psychic	3	1	3	80	100	120	140	150	110	1.4	40.0	

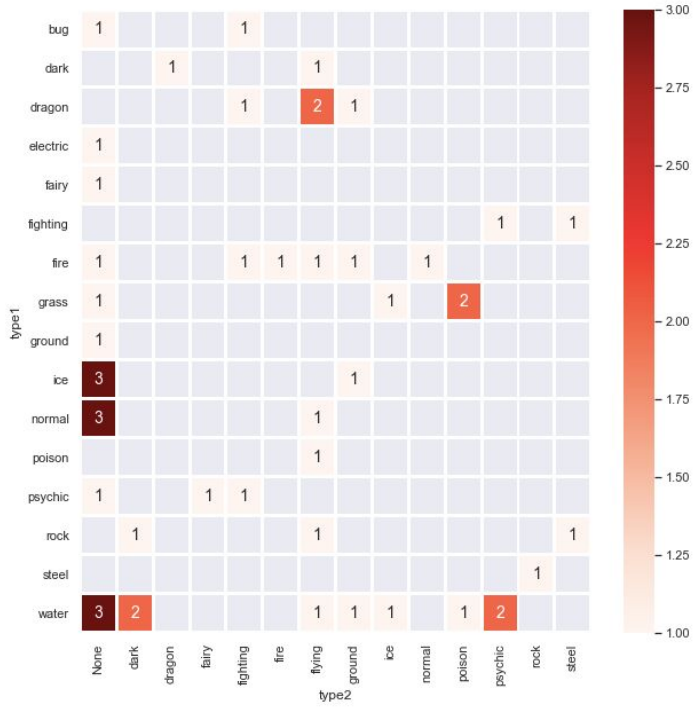
We need to keep in mind, however, that the list of pokemon on our dream team should be able to be captured which rules out legendary pokemon seeing as to how rare it is to find, and catch them. So we rule out legendary pokemon, and compile a list of strong non-legendary Pokemon.

```
powerful_non_legendary_pokemon = powerful_pokemon[powerful_pokemon['is_legendary'] == 0]
powerful_non_legendary_pokemon.head(10)
```

	id	name	type1	type2	generation	is_legendary	capture_rate	hp	attack	defense	sp_attack	sp_defense	speed	height_m	weight_kg	agai
247	248	Tyranitar	rock	dark	2	0	45	100	164	150	95	120	71	2.0	202.0	
444	445	Garchomp	dragon	ground	4	0	45	108	170	115	120	95	92	1.9	95.0	
372	373	Salamence	dragon	flying	3	0	45	95	145	130	120	90	120	1.5	102.6	
375	376	Metagross	steel	psychic	3	0	3	80	145	150	105	110	110	1.6	550.0	
288	289	Slaking	normal	None	3	0	45	150	160	100	95	65	100	2.0	130.5	
657	658	Greninja	water	dark	6	0	45	72	145	67	153	71	132	1.5	40.0	
129	130	Gyarados	water	flying	1	0	45	95	155	109	70	130	81	6.5	235.0	
259	260	Swampert	water	ground	3	0	45	100	150	110	95	110	70	1.5	81.9	
5	6	Charizard	fire	flying	1	0	45	78	104	78	159	115	100	1.7	90.5	
253	254	Sceptile	grass	None	3	0	45	70	110	75	145	85	145	1.7	52.2	

With these results, we can also tell which types of Pokemon are the stronger types of Pokemon. We try to deduce that with the seaborn heatmap visualization.

```
In [391]: fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(capture_ordered_nl_pokemon.head(50).groupby(['type1', 'type2']).size().unstack(),linewidths=2,annot=True
    annot_kws={'size': 14},cmap="Reds");
```



With these results, we were able to compile the list of 6 pokemon with two variations - one based on the capture rate of these pokemon, and the other based on just the grand total of their base stats. The following would be the code to display the results.

```
In [398]: capture_ordered_nl_pokemon = powerful_non_legendary_pokemon.sort_values(['capture_rate','grand_total'],ascending=Fal
capture_ordered_nl_pokemon = capture_ordered_nl_pokemon.apply(pd.DataFrame)

capture_ordered_nl_pokemon.head(6)['name']
```

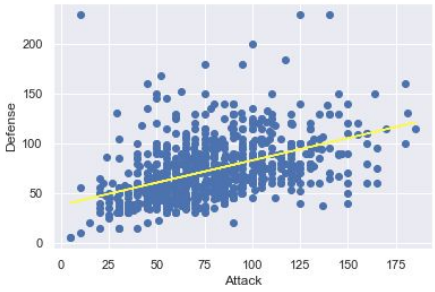
```
Out[398]: 530      Audino
322    Camerupt
669     Floette
168      Crobat
307    Medicham
79      Slowbro
Name: name, dtype: object
```

```
In [399]: capture_ordered_nl_pokemon = powerful_non_legendary_pokemon.sort_values(['grand_total','capture_rate'],ascending=Fal
capture_ordered_nl_pokemon = capture_ordered_nl_pokemon.apply(pd.DataFrame)

capture_ordered_nl_pokemon.head(6)['name']
```

```
Out[399]: 247    Tyranitar
444     Garchomp
372    Salamence
375    Metagross
288      Slaking
657     Greninja
Name: name, dtype: object
```

4. Building a legendary Pokemon classifier

```
In [431]: # We're going to see if there is a correlation between attack, defense stats to determine whether the Pokemon is leg

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression



X = pokestats[['attack']]
Y = pokestats[['defense']]

model = LinearRegression()
model.fit(X, Y)

Y_pred = model.predict(X)

plt.scatter(X, Y)
plt.plot(X, Y_pred, color = 'yellow')
plt.xlabel('Attack')
plt.ylabel('Defense')
plt.show()
```



In order to build a classifier, we first need to figure out what attributes are necessary to determine whether a Pokemon is legendary or not. This would involve trying to draw conclusions about correlations between Pokemon stats, height, weight, etc with their label of being legendary. We start off by seeing if attack, defense have anything to do with them being legendary. There seems to be a slight correlation. So we would use the base_total attribute along with capture_rate to determine whether a Pokemon is legendary or not. The capture_rate is used because of how rare and hard it is to capture legendary Pokemon, which makes the capture rate of legendary

Pokemon stand out.

```
In [463]: isLegendary = pd.get_dummies(pokedex['is_legendary'], drop_first=True)
          legendary_poke = pokedex[['base_total', 'capture_rate']]
          legendary_poke = pd.concat([legendary_poke, isLegendary], axis=1)
          legendary_poke.columns = ['base_total', 'capture_rate', 'is_legendary']
          legendary_poke.head(10)
```

Out[463]:

	base_total	capture_rate	is_legendary
0	318	45	0
1	405	45	0
2	625	45	0
3	309	45	0
4	405	45	0
5	634	45	0
6	314	45	0
7	405	45	0
8	630	45	0
9	195	255	0

We start off by trying to fit a logistic regression model to see if the model would be able to predict whether the Pokemon is legendary or not on the test set. This is done by a 70-30 split of data, and training the data to build the classifier. The code has been well documented.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn import tree
from sklearn.model_selection import cross_val_score, KFold

X_leg_train, X_leg_test, Y_leg_train, Y_leg_test = train_test_split(X_leg, Y_leg, test_size=0.3)


#Fitting the model to the training data.
LR = LogisticRegression()
LR.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
pred = LR.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       223
           1       0.76      0.89      0.82        18

    accuracy                           0.97       241
   macro avg       0.88      0.93      0.90       241
weighted avg       0.97      0.97      0.97       241
```

We also train a decision tree classifier, and get a better precision with the following data. Again, the code is well documented and self explanatory.

```
kfold = KFold(n_splits=500, random_state=10)

clf = tree.DecisionTreeClassifier().fit(X_leg,Y_leg)

result = cross_val_score(clf, X_leg,Y_leg, cv=kfold, scoring='accuracy')

print(result.mean())
```

```
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_split.py:296: FutureWarning: Setting a random_state
has no effect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its defaul
t (None), or set shuffle=True.
  FutureWarning
```

```
0.987
```

Results/Interpretation

Following are the observations from the dataset that are seen from the visualizations and understanding the data as with the code above :

1. The greater number of pokemon types are the water types, which is followed by normal, grass and bug respectively.

2. We can see that when the pokemon is grouped by the attack statistics, which include the special attack statistics - we can see that the top 5 Pokemon that has the highest attack would be Heracross, Kartana, Rayquaza, Groudon, and Garchomp.

3. We can see that when the pokemon is grouped by the defense statistics, which include the special defense statistics - we can see that the top 5 Pokemon that has the highest defense would be Shuckle, Steelix, Aggron, Regirock, Avalugg.

4. We can see how Caterpies, Pidgeys, Rattatas are the Pokemons with the higher capture rate. This is pretty obvious from the games as well. They are much more common when you tend to wander off into the wild grass.

5. Another interesting observation to make would be to count the number of legendary Pokemon in the list of Pokemon. Legendary Pokemon are Pokemon that are harder to catch, and if we're going to make the dream team, we should think about having a dream team which is realistic. This involves having Pokemon which aren't hard to catch, and Pokemon which have a good capture rate.

6. The top 5 common single type Pokemon are: Normal/None, Water/None, Grass/None, Psychic/None, Grass/None

7. The more common dual type Pokemon are: Normal/Flying, Grass/Poison, Bug/Flying, Bug/Poison, Water/Ground
8. The most common powerful types of Pokemon would be : Normal/None, Water/None, Dragon/Flying

Conclusion

1. **Can we create a dream team of 6 Pokemon? If so, what would they be?**

Yes, not only we were able to create a successful dream team. We were able to create two variations of the dream - one based on the capture rate, and another based on the base stats. Here are the results.

Therefore, when we consider a dream team, we could go with two approaches -

1. The first approach would be to choose Pokemon based on the capture rate, and form a team from the availability of the Pokemon.
2. The second approach would be to choose Pokemon based on sheer strength, with their base stats total.

The following could be the 2 variations of the dream team of 6 Pokémon

```
: capture_ordered_nl_pokemon = powerful_non_legendary_pokemon.sort_values(['capture_rate','grand_total'],ascending=Fal
  capture_ordered_nl_pokemon = capture_ordered_nl_pokemon.apply(pd.DataFrame)

  capture_ordered_nl_pokemon.head(6)['name']
```

```
: 530      Audino
  322    Camerupt
  669     Floette
  168      Crobat
  307    Medicham
  79      Slowbro
  Name: name, dtype: object
```

```
: capture_ordered_nl_pokemon = powerful_non_legendary_pokemon.sort_values(['grand_total','capture_rate'],ascending=Fal
  capture_ordered_nl_pokemon = capture_ordered_nl_pokemon.apply(pd.DataFrame)

  capture_ordered_nl_pokemon.head(6)['name']
```

```
: 247    Tyranitar
  444     Garchomp
  372    Salamence
  375    Metagross
  288      Slaking
  657     Greninja
  Name: name, dtype: object
```

The above teams looks pretty balanced to succeed in any of the Nintendo Pokemon games released, and they would be fairly easy to capture in the games.

2. **Would there be specific types that are stronger than the others?**
   With the help of visualizations, we were able to determine that certain types of Pokemon were stronger than the others. The most common powerful types of Pokemon would be : Normal/None, Water/None, Dragon/Flying.

3. **What are the most common types of Pokemon?**
   We have two answers for this question depending on whether we are looking for Pokemon with two types or whether they are single types. Both those questions have answers which have been answered in the results/interpretation section.

   The top 5 common single type Pokemon are: Normal/None, Water/None, Grass/None, Psychic/None, Grass/None

   The more common dual type Pokemon are: Normal/Flying, Grass/Poison, Bug/Flying, Bug/Poison, Water/Ground

4. **Would a Pokemon's height or weight play a factor in their strength?**
   We saw that there are quite a few outliers for height or weight to play a factor. For this reason, we used base stats as a determining factor and not the physical factors. So no, there wouldn't be much a correlation with height/weight with their overall strength.

5. **Would it be possible to tell if a Pokemon is legendary just from a couple of attributes?**
   We were able to classify legendary pokemon with a very high precision with the base_stats, capture_rate attributes. So this would be an obvious yes to this question.

Future Work

The results are quite conclusive of what type of Pokemon would be ideal for the dream team. We were able to answer all of the questions that we started out with. Additionally, building a classification system for legendary Pokemon is something that we managed to do, and we clearly saw how decision tree classifiers fared much better than a logistic regression model with the right attributes. If we were to extend the project, future questions that would need to be answered would be whether we can find out Pokemon teams which are strong against specific types (say, Ice, Fire etc).

Reference

1. https://pandas.pydata.org/docs
2. https://seaborn.pydata.org/tutorial.html
3. https://matplotlib.org/contents.html
4. https://altair-viz.github.io/index.html
5. https://scikit-learn.org/stable/user_guide.html
6. https://www.kaggle.com/rounakbanik/pokemon
7. https://pokemondb.net