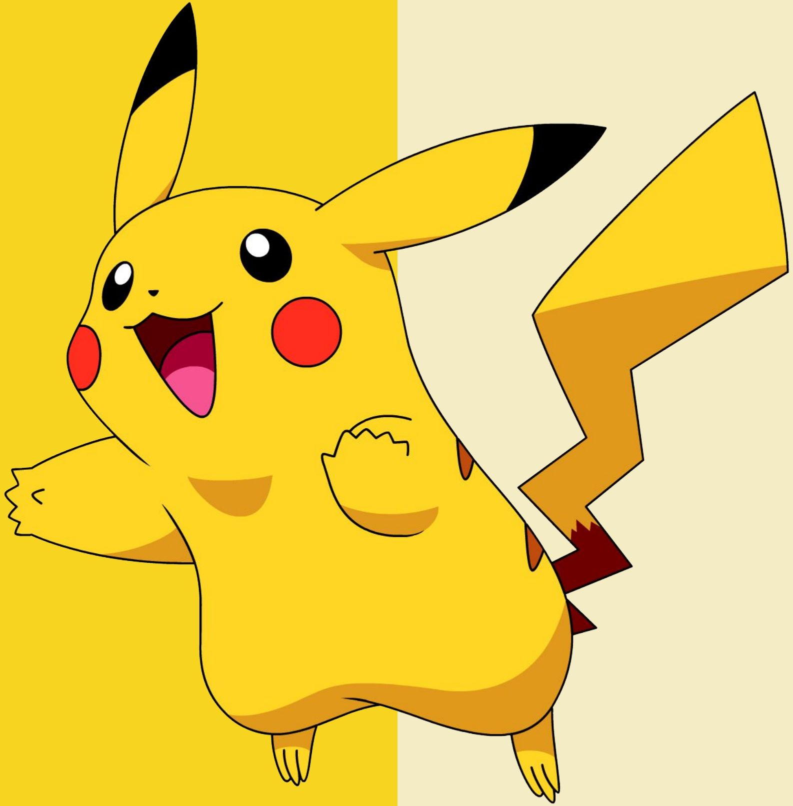
Pokémon
Gotta catch 'em all!

Machine Learning



Background

Pokémon is a multimedia franchise that began in 1996 with video games for the Game Boy. It features fictional creatures called Pokémon that trainers catch and train for battles. With over 800 species, players aim to become Pokémon Masters by collecting and battling with their teams. The franchise includes an animated TV series, Video games, movies, and merchandise. It has become a global phenomenon, capturing the hearts of millions of fans of all ages around the world.

For the purpose of this presentation, we will be simulating as a team that works in the part of the company that deals with the specific stats of the Pokémon.

Dataset Description

This dataset is focused on the stats and features of the Pokémon in the video games. The data was recorded by a bored individual named Alberto Rabbadas and published online. The dataset contains data on 721 Pokémon in total, so up to the 6th generation. (There are currently 8 generations)

Some Key Features

- ➔ Type - Main elemental type for the pokemon
- ➔ HP - Total health points
- ➔ Attack - Total Attack level
- ➔ Defense - Total Defense level
- ➔ Special Attack/Defense - Total points for elemental actions
- ➔ Catch Rate - Rate at which pokemon can be caught

	Number	Name	Type_1	Type_2	Total	HP	Attack	Defense	Sp_Atk	Sp_Def	...	Color	hasGender	Pr_Male	Egg_Group_1	Egg_Group_2
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	...	Green	True	0.875	Monster	Grass
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	...	Green	True	0.875	Monster	Grass
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	...	Green	True	0.875	Monster	Grass
3	4	Charmander	Fire	NaN	309	39	52	43	60	50	...	Red	True	0.875	Monster	Dragon
4	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	...	Red	True	0.875	Monster	Dragon
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
716	717	Yveltal	Dark	Flying	680	126	131	95	131	98	...	Red	False	NaN	Undiscovered	NaN
717	718	Zygarde	Dragon	Ground	600	108	100	121	81	95	...	Green	False	NaN	Undiscovered	NaN
718	719	Diancie	Rock	Fairy	600	50	100	150	100	150	...	Pink	False	NaN	Undiscovered	NaN
719	720	Hoopa	Psychic	Ghost	600	80	110	60	150	130	...	Purple	False	NaN	Undiscovered	NaN
720	721	Volcanion	Fire	Water	600	80	110	120	130	90	...	Brown	False	NaN	Undiscovered	NaN

```
Index(['Number', 'Name', 'Type_1', 'Type_2', 'Total', 'HP', 'Attack',
       'Defense', 'Sp_Atk', 'Sp_Def', 'Speed', 'Generation', 'isLegendary',
       'Color', 'hasGender', 'Pr_Male', 'Egg_Group_1', 'Egg_Group_2',
       'hasMegaEvolution', 'Height_m', 'Weight_kg', 'Catch_Rate',
       'Body_Style'],
      dtype='object')
```

Research Questions

- Do clusters exist within our pokemon?
- Can we accurately predict catch rate?
- Can we accurately predict the type of a pokemon?
- Can we accurately predict if a pokemon is legendary?
- Can we accurately predict pokemons body style?

Predicting Catch Rate

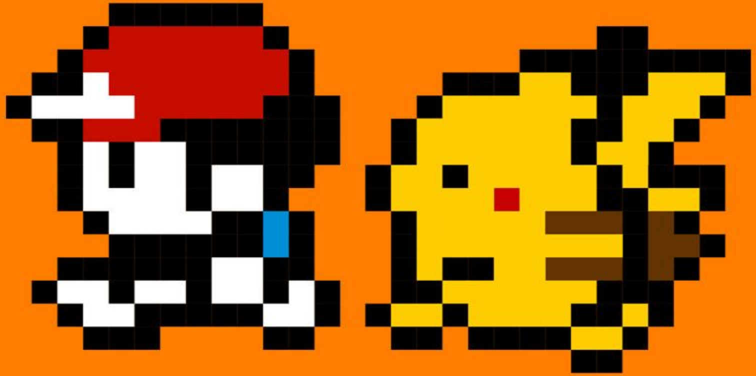
Why?

Since we are on a team looking to build the stats for new pokemon, it would be convenient to have a model that could accurately predict the catch rate of our pokemon.

How?

The methods to be used:

- Multiple Linear Regression (Best Subset)
- Random Forest Regressor
- Gradient Boosting Regressor



Multiple Linear Regression

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             Catch_Rate   R-squared:                       0.554
Model:                            OLS   Adj. R-squared:                  0.547
Method:                 Least Squares   F-statistic:                     78.61
Date:                Tue, 25 Apr 2023   Prob (F-statistic):          4.45e-104
Time:                        23:36:23   Log-Likelihood:                -3432.6
No. Observations:                 644   AIC:                             6887.
Df Residuals:                     633   BIC:                             6936.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    357.2773     10.972     32.561      0.000     335.731     378.824
HP            -0.7093      0.098     -7.263      0.000      -0.901      -0.518
Attack        -0.4229      0.095     -4.435      0.000      -0.610      -0.236
Defense       -0.5300      0.096     -5.540      0.000      -0.718      -0.342
Sp_Atk        -0.5429      0.093     -5.865      0.000      -0.725      -0.361
Sp_Def        -0.4694      0.104     -4.518      0.000      -0.673      -0.265
Speed         -0.5480      0.088     -6.195      0.000      -0.722      -0.374
Generation     1.9657      1.201      1.637      0.102      -0.392       4.323
Height_m      -3.5593      2.889     -1.232      0.218      -9.233       2.114
Weight_kg      0.0503      0.046      1.096      0.273      -0.040       0.141
Pr_Male      -70.2573     10.377     -6.770      0.000     -90.635     -49.879
==============================================================================
```

- As probability of being male increases, we expect catch rate to drop substantially.
- As stats increase, catch rate decreases.
- As generation increases, so does catch rate.

Random Forest

Parameters:

1000 Trees

Depth of 10 per tree

Accuracy:

R-squared = 0.607

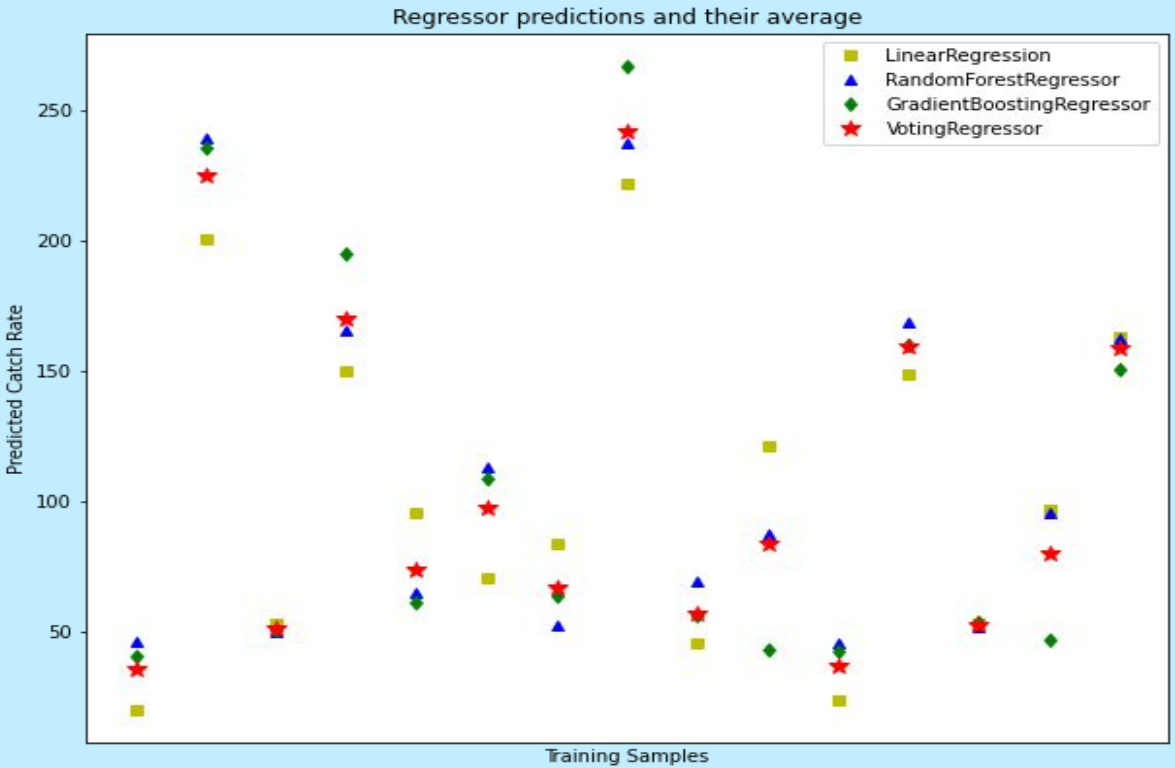
Gradient Boosting

Accuracy:

R-squared = 0.584

So our best model
standalone model is
Random Forest

Voting Regressor



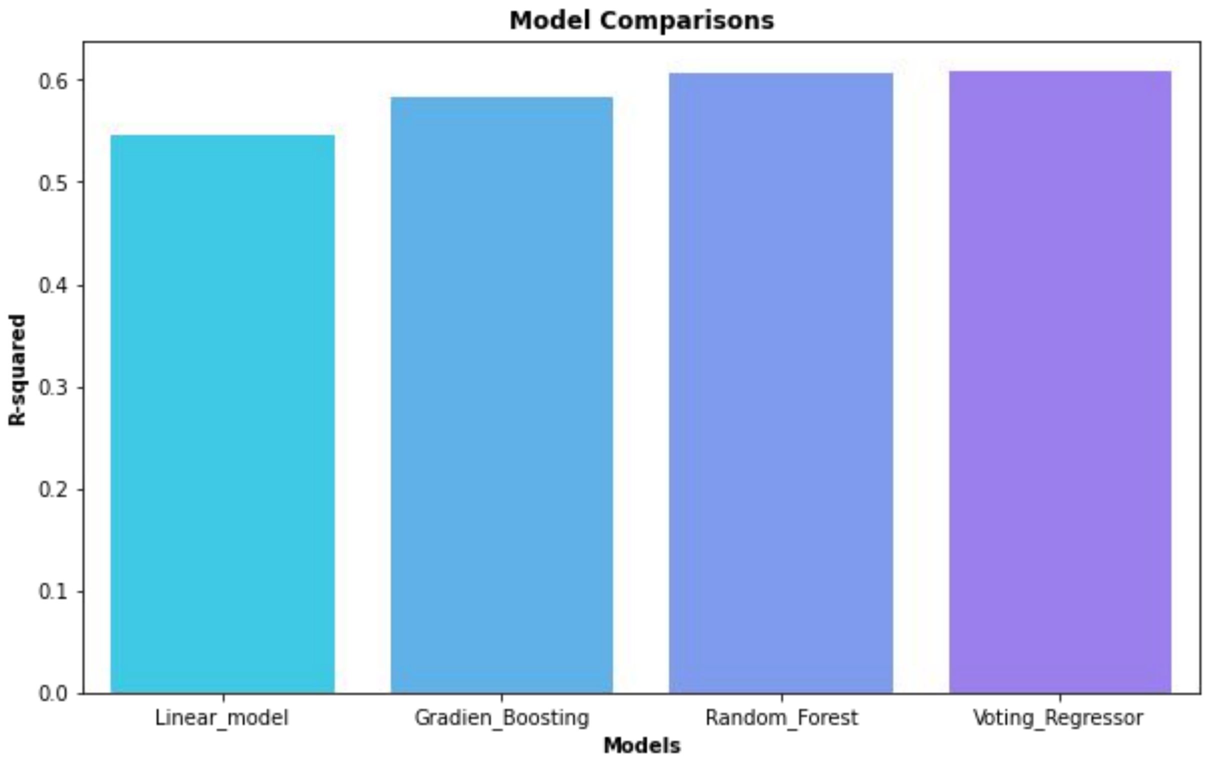
Parameters:

Voting = hard

Accuracy:

R-Squared = 0.608

Regression Models
R-Squared Comparison



Clustering

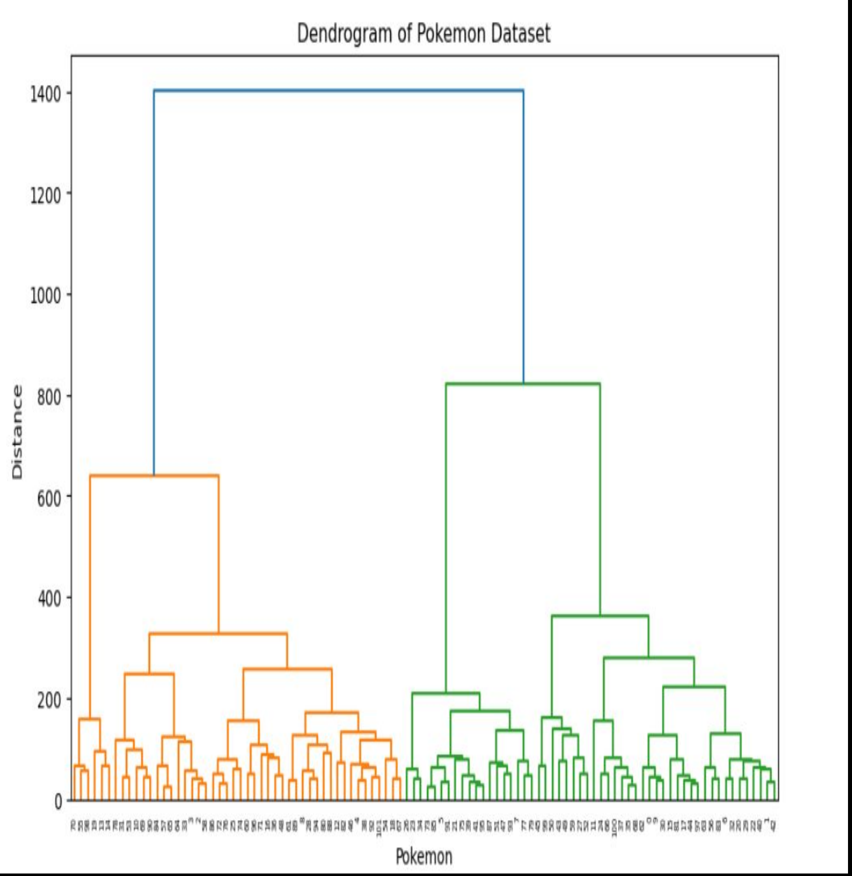
Preprocessing and Standardizing Data

	Pr_Male	Generation	Total	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Pr_Male	Height_m	Weight_kg	Catch_Rate
0	1.628168	-1.430397	-1.117514	-1.199832	-0.980330	-1.077927	-0.176955	-0.246667	-0.626658	1.628168	-0.550801	-0.716665	-0.749981
1	1.628168	-1.430397	-0.183023	-0.481623	-0.515742	-0.506978	0.442591	0.450931	-0.021363	1.628168	-0.188255	-0.623521	-0.749981
2	1.628168	-1.430397	1.105930	0.475990	0.199009	0.308665	1.268652	1.381060	0.785697	1.628168	1.132449	0.704928	-0.749981
5	1.628168	-1.430397	1.202602	0.380229	0.270484	0.104754	1.640380	0.683463	1.592757	1.628168	0.731059	0.559868	-0.749981
33	2.331868	-1.430397	0.891105	0.523870	0.913760	0.063972	0.649106	0.218398	0.987462	2.331868	0.342616	0.124686	-0.749981
45	-0.482931	-1.430397	-1.471976	-1.678639	-0.229841	-0.833235	-1.003017	-0.711732	-1.433718	-0.482931	-1.081673	-0.739569	1.445628
46	-0.482931	-1.430397	-0.183023	-0.481623	0.663597	0.186318	-0.383471	0.450931	-1.231953	-0.482931	-0.188255	-0.371573	-0.295717
78	-0.482931	-1.430397	-1.149738	0.954796	-0.408529	-0.425413	-1.209532	-1.409329	-1.837248	-0.482931	0.070707	-0.272321	1.445628
79	-0.482931	-1.430397	0.729986	1.194199	-0.051154	1.409782	1.268652	0.450931	-1.231953	-0.482931	0.601578	0.376633	-0.295717
82	-0.482931	-1.430397	-0.752311	-0.864668	-0.408529	-0.833235	-0.466077	-0.386186	-0.021363	-0.482931	-0.447217	-0.592982	-0.749981
86	-0.482931	-1.430397	0.568867	0.954796	-0.229841	0.186318	0.029560	1.148528	0.382167	-0.482931	0.731059	1.010319	-0.295717
110	-0.482931	-1.430397	-0.827500	0.475990	0.306222	0.798050	-1.622563	-1.874394	-1.433718	-0.482931	-0.188255	0.933971	0.385679
111	-0.482931	-1.430397	0.676279	1.673006	1.914411	1.817603	-1.003017	-1.176796	-0.828423	-0.482931	1.002969	1.010319	-0.522849
129	-0.482931	-1.430397	1.267050	1.194199	1.735723	0.145536	-0.383471	1.381060	0.826050	-0.482931	6.946139	2.766314	-0.749981
130	-0.482931	-1.430397	1.213343	2.870022	0.306222	0.186318	0.649106	1.148528	-0.021363	-0.482931	1.753957	2.537271	-0.749981
137	1.628168	-1.430397	-0.720087	-1.678639	-1.301967	1.001960	0.855622	-0.711732	-1.030188	1.628168	-0.939244	-0.707503	-0.749981
138	1.628168	-1.430397	0.783692	-0.002817	-0.587217	2.021513	1.888198	-0.014134	-0.223128	1.628168	-0.188255	-0.287591	-0.749981
139	1.628168	-1.430397	-0.720087	-1.918042	0.127534	0.594139	-0.589986	-1.176796	-0.223128	1.628168	-0.809763	-0.646425	-0.749981
140	1.628168	-1.430397	0.783692	-0.481623	1.378348	1.205871	-0.176955	-0.014134	0.785697	1.628168	0.213136	-0.203609	-0.749981
148	-0.482931	-1.430397	1.911526	1.002677	2.057361	0.798050	1.268652	1.381060	0.785697	-0.482931	1.391411	2.384576	-0.749981

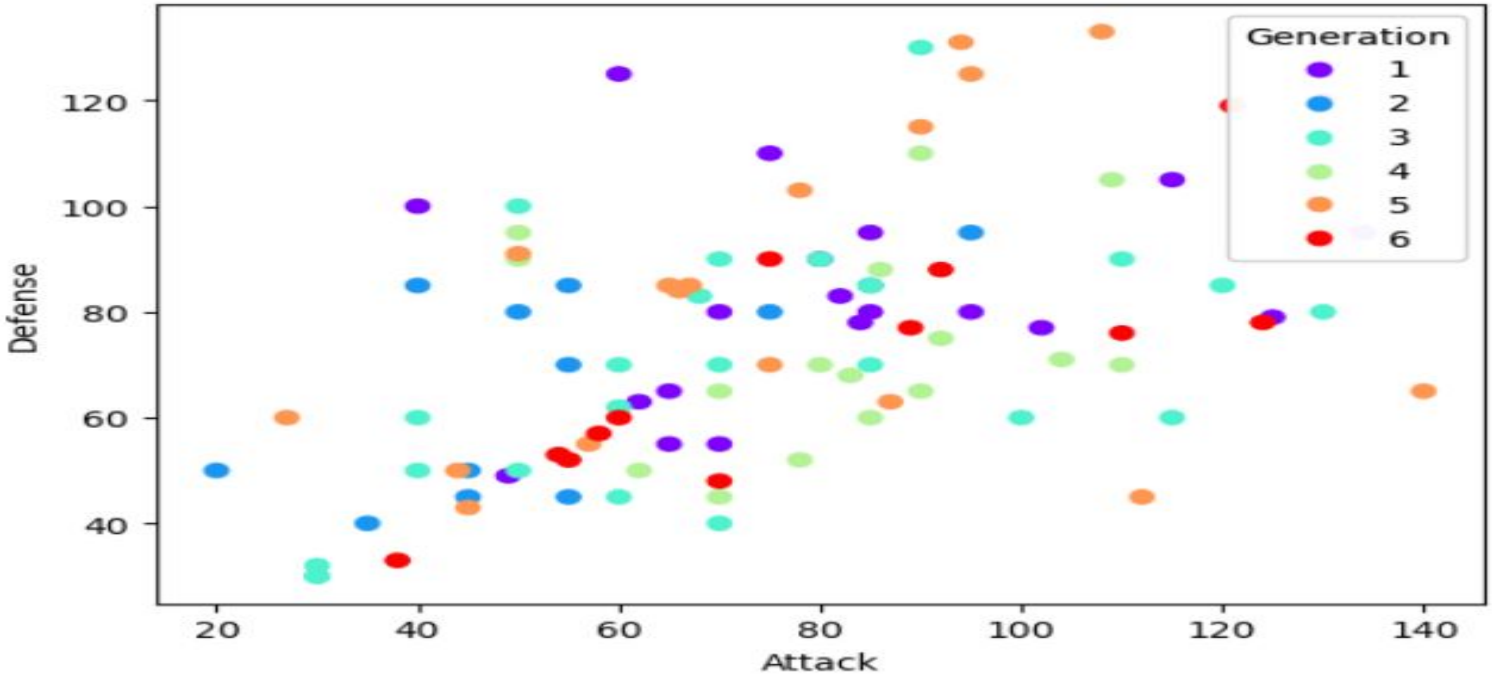
Data cleaning

	Number	Name	Type_1	Type_2	Total	HP	Attack	Defense	Sp_Atk	Sp_Def	...	Color	hasGender	Pr_Male	Egg_Group_1	Egg_Group_2	has
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	...	Green	True	0.875	Monster	Grass	
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	...	Green	True	0.875	Monster	Grass	
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	...	Green	True	0.875	Monster	Grass	
5	6	Charizard	Fire	Flying	534	78	84	78	109	85	...	Red	True	0.875	Monster	Dragon	
33	34	Nidoking	Poison	Ground	505	81	102	77	85	75	...	Purple	True	1.000	Monster	Field	
45	46	Paras	Bug	Grass	285	35	70	55	45	55	...	Red	True	0.500	Bug	Grass	
46	47	Parasect	Bug	Grass	405	60	95	80	60	80	...	Red	True	0.500	Bug	Grass	
78	79	Slowpoke	Water	Psychic	315	90	65	65	40	40	...	Pink	True	0.500	Monster	Water_1	
79	80	Slowbro	Water	Psychic	490	95	75	110	100	80	...	Pink	True	0.500	Monster	Water_1	
82	83	Farfetch'd	Normal	Flying	352	52	65	55	58	62	...	Brown	True	0.500	Flying	Field	
86	87	Dewgong	Water	Ice	475	90	70	80	70	95	...	White	True	0.500	Water_1	Field	
110	111	Rhyhorn	Ground	Rock	345	80	85	95	30	30	...	Grey	True	0.500	Monster	Field	
111	112	Rhydon	Ground	Rock	485	105	130	120	45	45	...	Grey	True	0.500	Monster	Field	
129	130	Gyarados	Water	Flying	540	95	125	79	60	100	...	Blue	True	0.500	Water_2	Dragon	
130	131	Lapras	Water	Ice	535	130	85	80	85	95	...	Blue	True	0.500	Monster	Water_1	
137	138	Omanyte	Rock	Water	355	35	40	100	90	55	...	Blue	True	0.875	Water_1	Water_3	
138	139	Omastar	Rock	Water	495	70	60	125	115	70	...	Blue	True	0.875	Water_1	Water_3	
139	140	Kabuto	Rock	Water	355	30	80	90	55	45	...	Brown	True	0.875	Water_1	Water_3	
140	141	Kabutops	Rock	Water	495	60	115	105	65	70	...	Brown	True	0.875	Water_1	Water_3	
148	149	Dragonite	Dragon	Flying	600	91	134	95	100	100	...	Brown	True	0.500	Water_1	Dragon	

Dendrogram of Numericals within Dataset



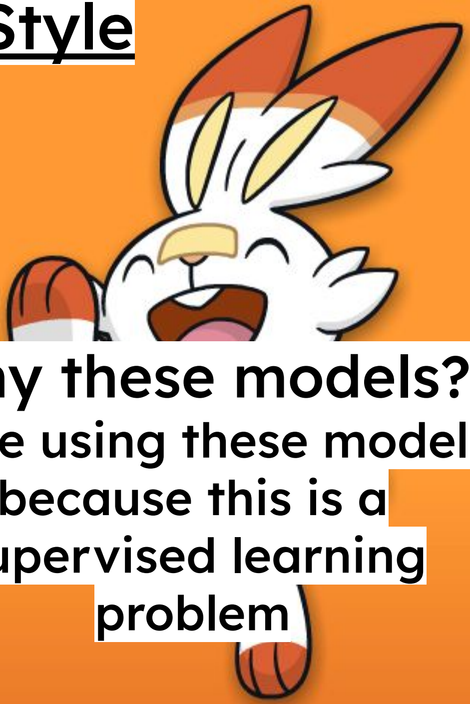
Correlation between its Attack, Defense and Generation



Predicting Body Style

Methods Used:

Deep Learning
K-Nearest
Select Vector Machine
Random Forest
Decision Tree
Ensemble Hard and Soft Voting



Why these models?

We're using these models because this is a supervised learning problem

Data Manipulation

	Total	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Generation	Type_1	Type_2
0	318	45	49	49	65	65	45	1	Grass	Poison
1	405	60	62	63	80	80	60	1	Grass	Poison
2	525	80	82	83	100	100	80	1	Grass	Poison
3	309	39	52	43	60	50	65	1	Fire	NaN
4	405	58	64	58	80	65	80	1	Fire	NaN
...	...	...	...	...	...	...	...	...	...	...
716	680	126	131	95	131	98	99	6	Dark	Flying
717	600	108	100	121	81	95	95	6	Dragon	Ground
718	600	50	100	150	100	150	50	6	Rock	Fairy
719	600	80	110	60	150	130	70	6	Psychic	Ghost
720	600	80	110	120	130	90	70	6	Fire	Water

721 rows × 10 columns

..	Egg_Group_2_Fairy	Egg_Group_2_Field	Egg_Group_2_Flying	Egg_Group_2_Grass	Egg_Group_2_Human-Like
..	0	0	0	1	0
..	0	0	0	1	0
..	0	0	0	1	0
..	0	0	0	0	0
..	0	0	0	0	0
..	...	...	...	...	...
..	0	0	0	0	0
..	0	0	0	0	0
..	0	0	0	0	0
..	0	0	0	0	0
..	0	0	0	0	0

```
df = pd.get_dummies(df, columns=["Type_1"])
df = pd.get_dummies(df, columns=["Type_2"])
df = pd.get_dummies(df, columns=["Egg_Group_1"])
df = pd.get_dummies(df, columns=["Egg_Group_2"])
```

Training and Scaling/normalizing

Splitting the data between either, 0.3 or .25 was the most optimal split for my data. Meaning that 30% or 25% of the dataset will be used as the test set, while the remaining 70% or 65% will be used as the training set.

Support Vector Machine

.3

Accuracy Score:
0.48847926267281105

.25

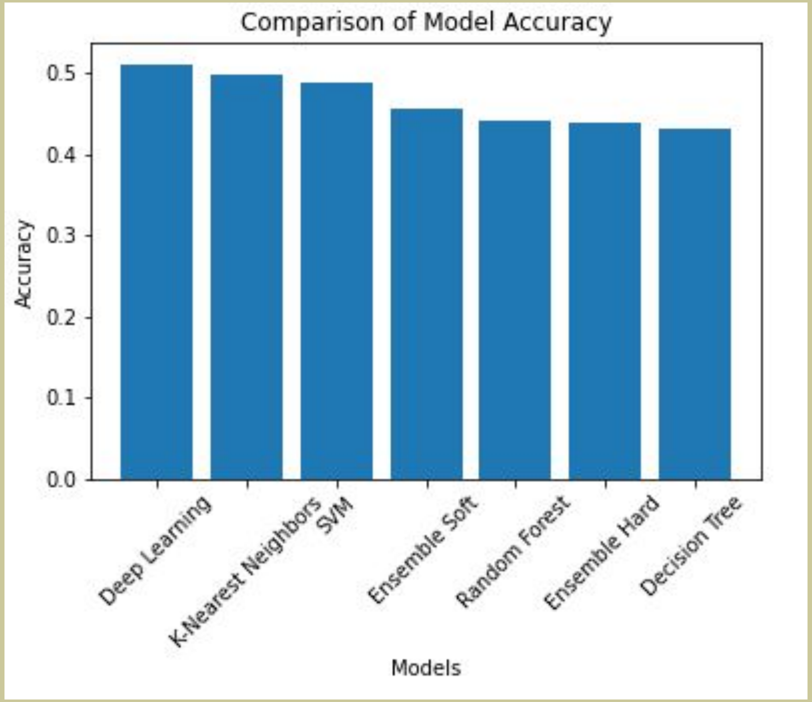
Accuracy Score:
0.46408839779005523

Without Scaler (on .3)

Accuracy Score:
0.2073732718894009

Comparing Model Accuracies

```
Deep Learning  =  0.5115207433700562
K nearest  =  0.4972375690607735
Support Vector Machine =  0.48847926267281105
Ensemble Soft Voting 0.45622119815668205
Random Forest  =  0.442396313364553
Ensemble Hard Voting 0.4377880184331797
Decision Tree  =  0.430939226519337
```



Predicting Pokemon Type

Why?

Accurately predicting pokemon types would allow us to look for any correlations or trends for any given feature and a Pokemon's type. We also want to see if a given model will produce better predictions.

How?

The methods to be used:

- Classification
  - Decision Tree
  - Random Forest
  - Support Vector Machine
  - Logistic Regression
  - AdaBoost
  - Bagging Ensemble

Pre-Processing Dataset

- Using Type as a predictor
  - 18 types of pokemon
  - Over 300 pokemon have two types
  - Potentially 342 combinations for the predictor to choose from
- Handling categorical features
  - Use one hot encoding
- Removing features
  - Redundant features
    - Total is just the sum of unique pokemon stats
  - Insignificant features
    - Egg type gives away pokemon type

Pre-Processing Continued

- Original Dataset
  - 721 rows and 22 columns
- Clean Dataset
  - 371 rows and 35 columns
    - Dummy columns added for body type and color
    - Pokemon with second type was dropped to lower the number of choices for prediction
    - Columns removed
      - Pr_Male
      - hasMegaEvolution
      - Total
      - Egg_Group 1 and 2
  - Standardized

Body_Style_head_arms	Body_Style_head_base	Body_Style_head_legs
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
...	...	...
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Color_Black	Color_Blue	Color_Brown	Color_Green	Color_Grey
0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0
...	...	...	...	...
0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0

Classification Training

- Logistic Regression
  - 1000 iterations and lbfgs solver
- Random Forest
  - 100 estimators
- Voting Classifier
  - Hard voting between logistic regression, random forest and SVC
- AdaBoost
  - 200 samples
- Bagging
  - Bootstrap
- Decision Tree and SVC
  - Using default parameters

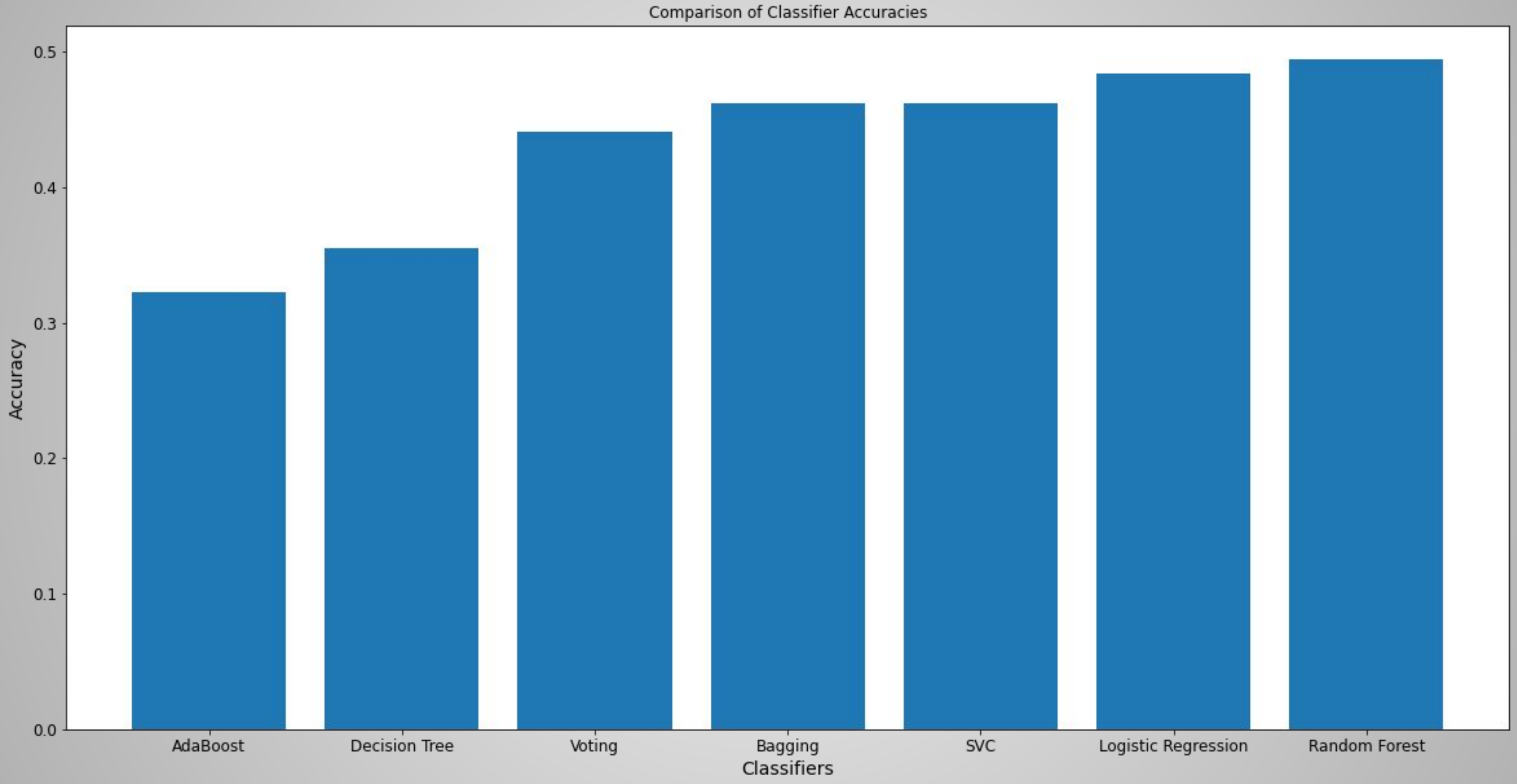
```
LogisticRegression 0.4838709677419355
RandomForestClassifier 0.4946236559139785
SVC 0.46236559139784944
VotingClassifier 0.46236559139784944
AdaBoost  0.3225806451612903
Bagging  0.44086021505376344
Decision Tree  0.3548387096774194
```

Classification Analysis/Comparison

- Are the classifiers weak learners?
  - Not necessarily
    - Weak learners are when a model's accuracy is equal to or less than the chance of random guessing
    - Many options for model to predict from
    - At the end, correlation between a pokemon's features and type to predict at a accurate rate is not strong enough
- What were the best and worst models?

Low Accuracy across all models!

```
LogisticRegression 0.4838709677419355
RandomForestClassifier 0.4946236559139785
SVC 0.46236559139784944
VotingClassifier 0.46236559139784944
AdaBoost  0.3225806451612903
Bagging  0.44086021505376344
Decision Tree  0.3548387096774194
```


Comparison of Classifier Accuracies
Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
AdaBoost
Decision Tree
Voting
Bagging
SVC
Logistic Regression
Random Forest
Classifiers

Predicting Legendary Pokemon

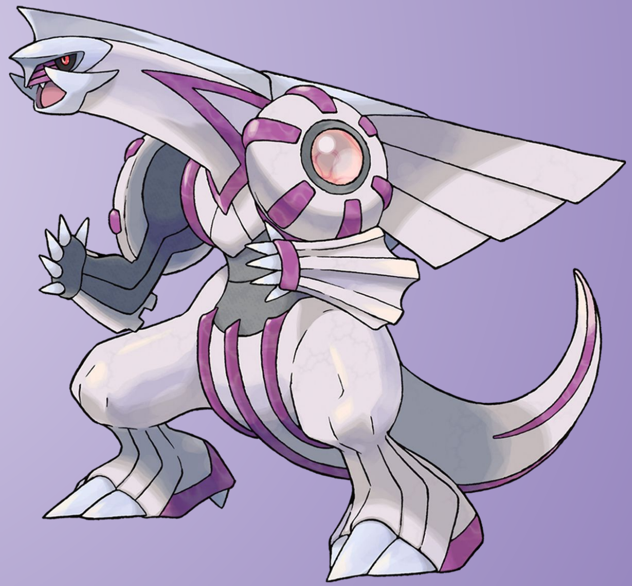
Why?

We want to see if there are any characteristics of a pokemon that will determine if they will be considered legendary or not. This can be used as a reference when creating new legendary pokemon.

How?

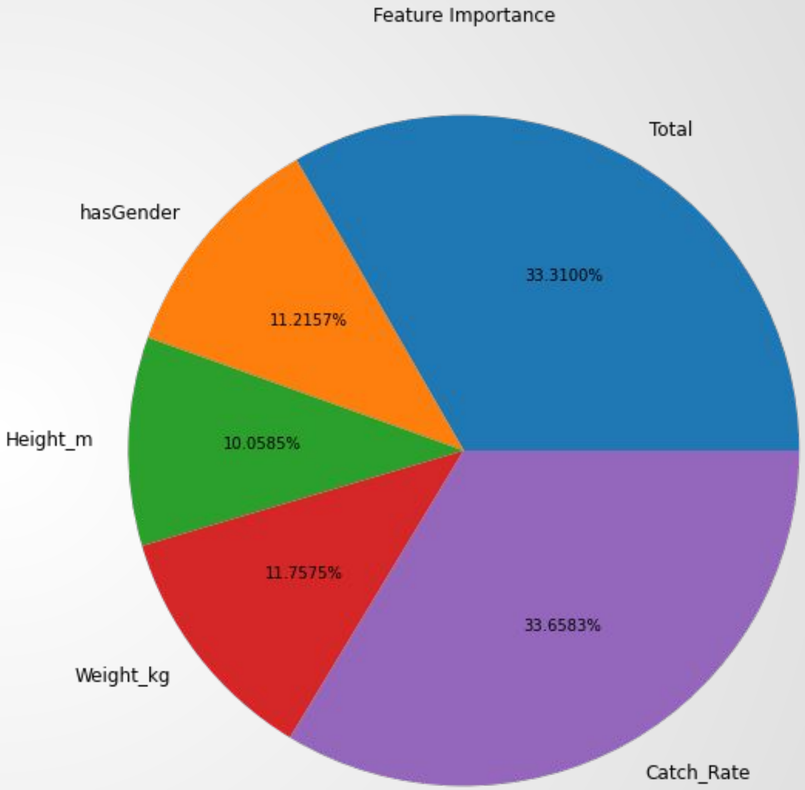
The methods to be used:

- Same Classifiers as before
  - Decision Tree
  - Random Forest
  - Support Vector Machine
  - Logistic Regression
  - AdaBoost
  - Bagging Ensemble



Pre-Processing Dataset

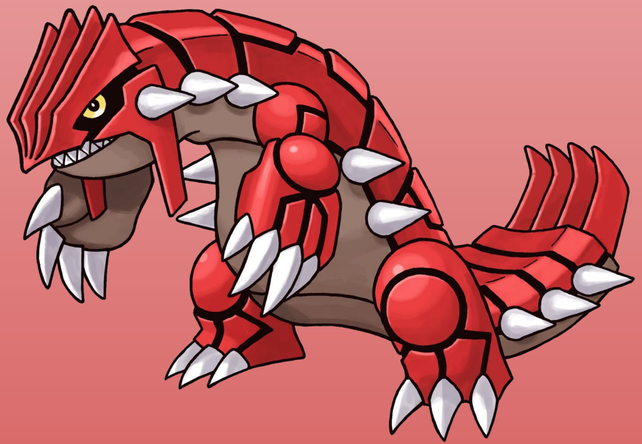
- Using isLegendary as the predictor
  - 50% chance to randomly guess correctly
- Removing features
  - Feature importance
    - After many tests, a few features were selected as the most important features for prediction
- Original Dataset
  - 721 rows and 22 columns
- Clean Dataset
  - 721 rows and 5 columns



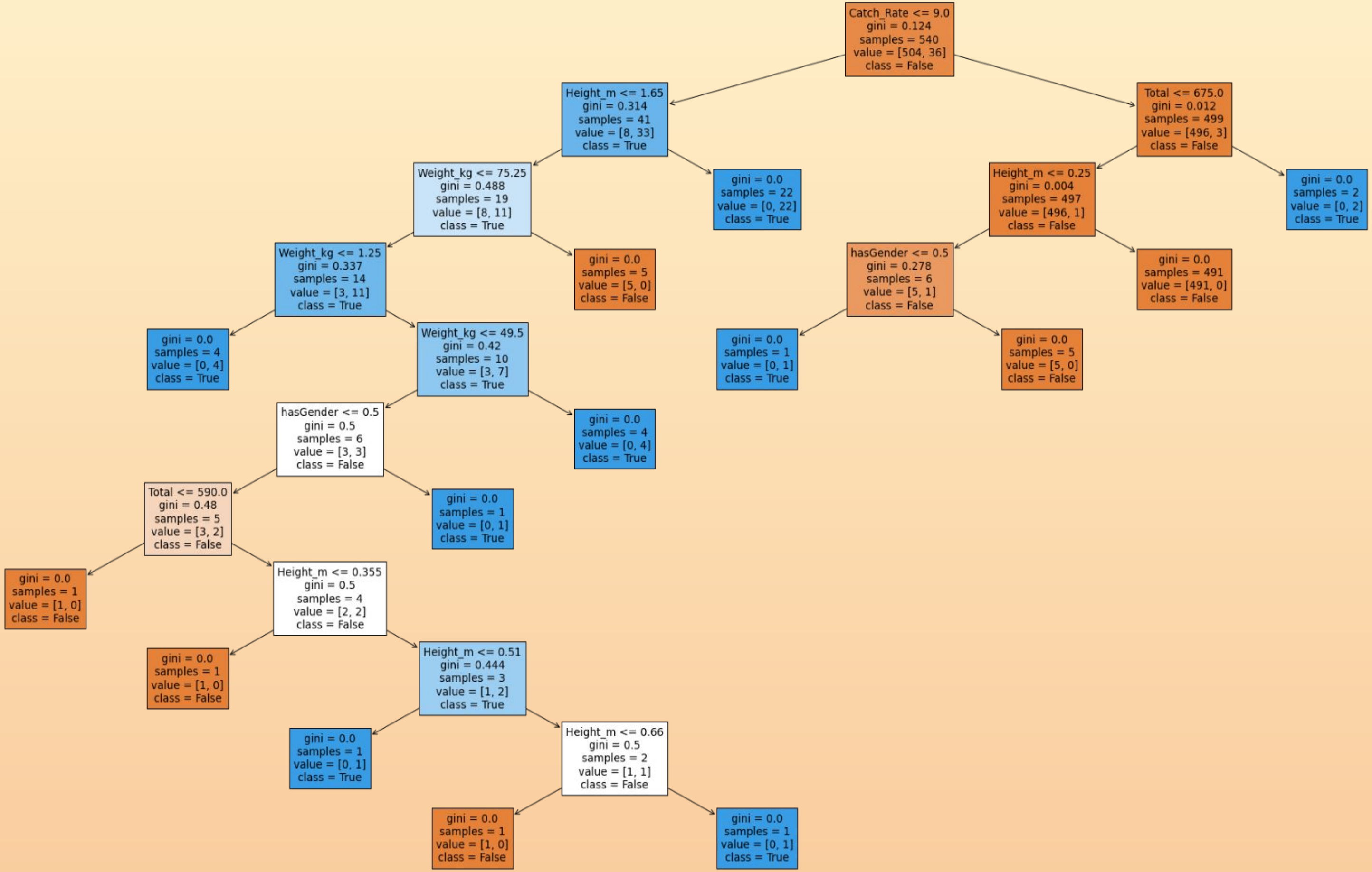
Classification Training and Analysis

- Training
  - Logistic Regression
    - Using newton-cg solver
  - Other classifiers used previous settings
- Analysis
  - Very high accuracy across all models (almost 100%!)
    - This could potentially be explained by either the high correlation between the features and legendary AND because the predictor is a binary choice
  - Support vector machine performed the worst

```
LogisticRegression 1.0
RandomForestClassifier 1.0
SVC 0.9779005524861878
VotingClassifier 1.0
AdaBoost  1.0
Bagging  1.0
Decision Tree  1.0
```



Decision Tree Classifier



Further Research

- Test our models with newer generations of Pokemon
- Predicting outcomes of Pokemon battles
- Predicting a Pokemon’s habitat
- Predicting a Pokemon’s hatch time
- Analyze and predict pokemon moveset statistics