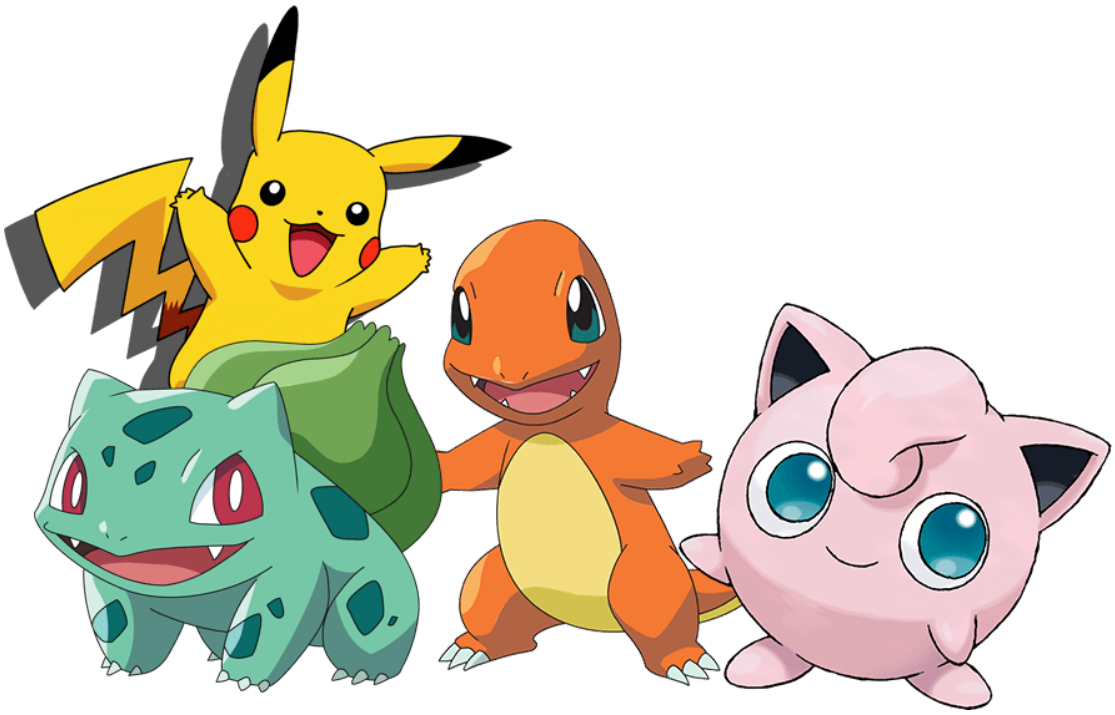
Pokémon Winner Prediction

Predict the winner Pokémon in a battle

Dec 4, 2021



Group 5

Harshini Gadige (015351232)

Andrew Kehl (010586069)

Viritha Vanama (015356991)

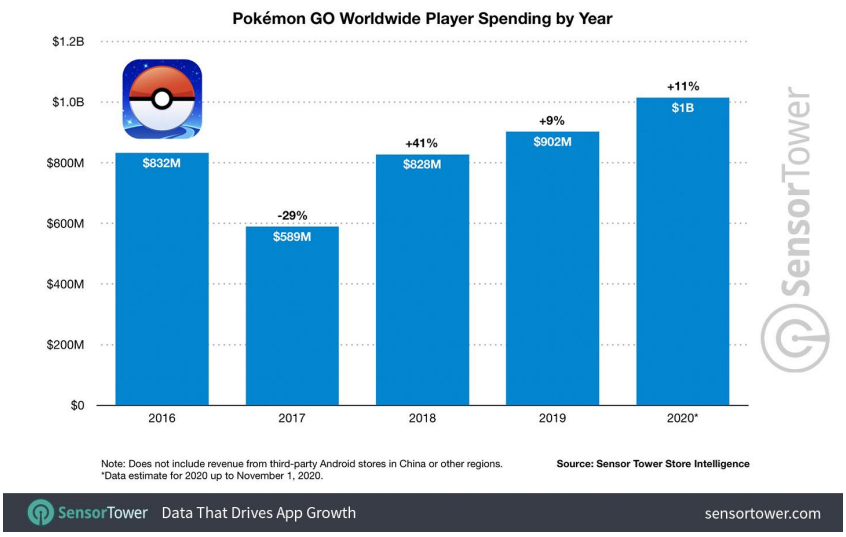
Aneshaa Kasula (014558427)

Introduction

Pokémon (a.k.a *Pocket Monsters* in Japanese) was created by Satoshi Tajiri in 1996. It is the world's largest media franchise with successful anime series, movies, and merchandise, with spin-off game *Pokémon, Go* and managed by 'The Pokémon Company', a company founded by Nintendo, Game Freak, and Creatures. The concept of the Pokémon universe in fictional works started from a popular pastime of the creator as a child - hobby of insect collecting. Pokémon Go - Augmented Reality mobile game is a successful mobile application developed by Niantic Labs, a former division of Google. It is the most used and profitable mobile application of all time. In this video game, players are designated as Pokémon Trainers who catch and train Pokémon to battle with other Pokémon.

Importance of the Problem

Winning the battle excites players and increases the interest to play again and again, ultimately resulting in profits to the company.



Objective

Our project aim is to use machine learning for predicting the winner of Pokémon among player choices. Potential audiences are the general public interested in playing the game and company management.

Literature Survey

Due to the increasing popularity of online games, there are a fair amount of machine learning projects available with similar context for our references. Even though factors affecting the

winner are different for each game, the use of machine learning resulted in better predictions of the winning team or player in every paper.

One among them is 'Predicting the winning side of DotA2', which is another most popular online game. A regular DotA2 match has ten players with five players on each side. Each player will choose a hero and their goal is to defeat their opponents. The goal of this paper is to predict whether a hero is strong or weak, by building a better model which is quite similar to ours. The authors used python module sci-kit to do the logistic regression which minimized cost function on their data set and performed 10-fold cross-validation as it is the most commonly used k-fold CV in practice as part of designing evaluation experiments. To avoid prediction error they added additional features as the probability of winning labels were almost the same i.e., data points are too close to the decision boundary so that it is hard to classify such matches. They used stepwise regression for feature selection with the idea to start with no feature and add one feature that increased the model most at each step and achieved good predictions with collected data but concluded saying that taking players' information into account might give better results.

Another paper's aim is also similar to ours, that is to predict the winner of an NFL game using machine learning, even though it is not an online game. The author says, in any kind of game the outcome can be predicted deterministically, hence his goal is to come up with a system that is comparable or better than human prediction. Used a scheme similar to leave-one-out classification with an additional constraint i.e., cannot use the data from games that happened in the same week or future weeks and we will not be using this concept as the time feature is not part of this project. Built logistic regression and svm with different kernels while logistic regression gave the best results, although SVM with linear kernel has similar performance. This paper also suggests exploring other algorithms apart from the proposed ones and concludes machine learning can be a reasonable solution especially compared to humans.

Another research paper that predicts the winner for the Indian Premier League (IPL) which is an Indian-origin cricket league, 'The Cricket Winner Prediction With Application Of Machine Learning And Data Analytics'. The authors implemented a range of prediction models on historical data and each model's performance increased after performing features selection and fine-tuning parameters; the accuracy score of the Decision Tree model was enhanced from 76.9% to 94.87%. Random Forest model predicted with 80% of accuracy while XGBoost gave 94.23 % accuracy rate without tuning of parameters. The authors concluded by saying that domain information and expertise saved a lot of time during analysis and the decision tree model outperformed in their case.

There are two more papers, predicting the winning team in the IPL using machine learning. One among them used the Naive Bayes approach and the accuracy score was 85.71%. It concluded that using an appropriate feature selection algorithm is important not only in reducing the dimensionality but helps in reducing the impact from the violation of independent assumption. The other paper implemented different supervised machine learning classification algorithms such as Random Forest, Naive Bayes, SVM, K-nearest Neighbor, Logistic Regression, ExtraTreesClassifier, XGBoost. The highest accuracy was achieved by Random Forest with an ambiguity of 1.4% after hyperparameter tuning. Hence, we have decided to use feature selection algorithms and also perform hyperparameter tuning on the candidate model for better predictions.

Technical Survey

In this case of supervised machine learning (classification), we decided to build the basic to best models that can overcome bias, variance, and overfitting for the dataset, also considering suggestions from literature surveys. So, the finalized predictive models are Decision Tree, Random Forest, AdaBoost, Support vector machine, and Logistic regression.

We know that while performing classification predictions, there are four types of outcomes that

could occur: True Positive(TP), True Negative (TN), False Positive(TP), and False Negative (TN). We use a confusion matrix, for better visualization of the performance of any model as these four outcomes are plotted.

As we want our model to focus on True positive and True Negative, Accuracy is one of the metrics which gives the fraction of predictions to show that our model got right (i.e., Accuracy = Number of correct predictions / Total number of predictions).

For classification problems with probability outputs, a threshold converts probability outputs to classifications and is able to control the confusion matrix by changing the threshold. When there is a change in a threshold we don't want to look at each confusion matrix separately, in such cases we could use the receiver operator characteristic (ROC/AUC) curve, which captures and plots out the sensitivity and specificity for every possible decision rule cutoff between 0 and 1 for a range of selected predictive models. One way to compare classifiers is to measure the area under the curve for ROC.

Apart from performance measures used while evaluating selected trained models, designing evaluation experiments is much needed as its goal is to ensure that our calculations for the best estimate of how a candidate model performs most appropriately when it is actually deployed. Cross-validation (CV) in Machine learning generally refers to the ability of an algorithm to be effective across various inputs. CV is said to be easily understood, implemented, and it tends to have a lower bias than other methods used to count the model's efficiency scores. The CV algorithm is similar across various techniques, first, the dataset is divided into training and testing, then train the model on the training set, validate the model on the test set and repeat these three steps depending on the choice of cross-validation method used. We decided to use standard k-fold which minimizes the bottleneck problem from the hold-out technique by splitting the data into k equal parts and choosing k-1 folds for training and remaining to the test set. This technique saves the scores of each validation and results out the average score of each

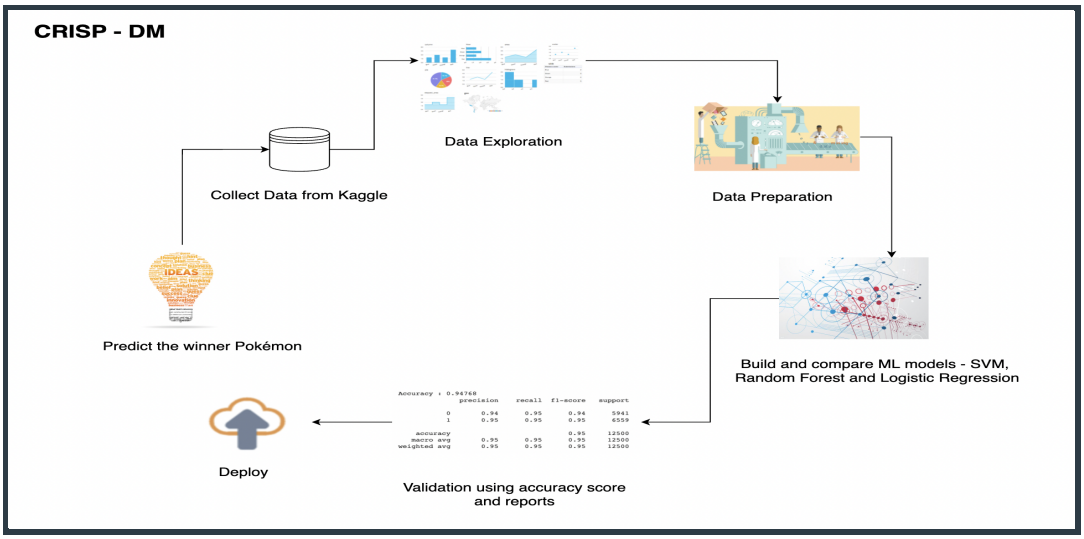
model in the selection. It gives more stable and trustworthy results since training and testing is performed on different parts of a given dataset. Since we will be using 10 folds, it is less expensive and the process does not consume time. For hyperparameter tuning, we use RandomizedSearchCV.

Technical Requirements

Hardware: Currently, this model requires the use of an x86 processor. There are a number of ARM-based processors capable of running Python but they have not been validated and the members of this team cannot guarantee they work. The accompanying notebook for this project was run on an AMD Ryzen 9 3950X (16 core 32 thread) desktop processor in less than one hour from start to finish. There is a very minor graphics requirement, simply the ability to display static images in the Jupyter notebook. There is no graphics-based computing for the machine learning models.
Software: Google Colab, Python3 (libraries: Pandas, Seaborn, Matplotlib, Numpy, Scipy, Scikit learn)

Project Methodology

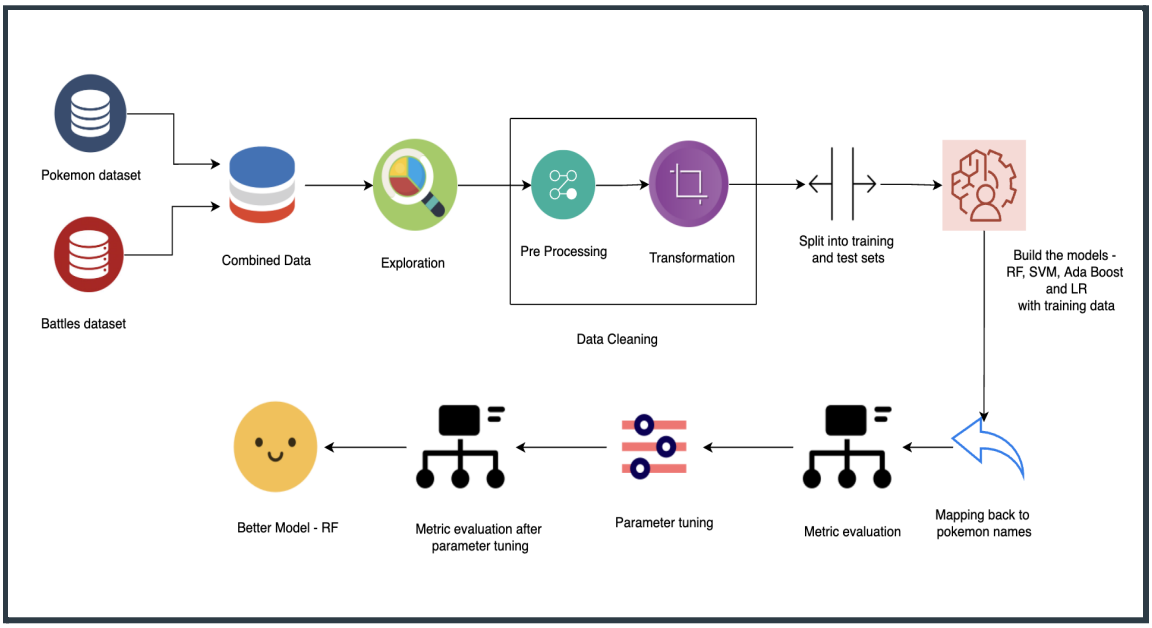
Predictive data analytics projects for any kind of application involve a lot more than just choosing the right machine learning algorithm, to become successful, a standard process is needed to manage the project lifecycle. So we chose one of the most commonly used processes for predictive data analytics projects, the Cross-Industry Standard Process for Data Mining (CRISP-DM) which is broadly divided into six phases namely Business Understanding, Data Understanding, Data Preparation, Model development, Evaluation, and Deployment. Figure 1 shows all the tasks involved in this project using CRISP-DM.



- In the first phase of CRISP-DM, a data analyst has to understand the business problem and solve it using an analytical approach. So we set our goal from the perspective of solving both participants of the Pokémon game and company management; make better predictions of the winner Pokémon not by intuitive or involving humans but using machine learning.
- In the next two phases, we planned to leverage the publicly available Pokémon dataset (Kaggle) and get to know more about data by performing Exploratory Data Analysis (EDA) such as plotting histograms, bar graphs, box plots. After collecting the required data, we have to transform raw data to the required form and build an analytics base table (ABT) for building predictive data analytics models.
- In the modeling phase, different machine learning algorithms are used to build a range of models from which the best predictive model will be selected for deployment. Here we have to build a supervised classification model to predict the winner when two Pokémon names are given by a player. Data might be biased in reality so keeping that in mind, we have decided to build ensemble models such as Random Forest, AdaBoost also included support vector machine and logistic regression models from a literature survey.
- The next phase in CRISP-DM covers all the evaluation tasks required to show that a

prediction model will be able to make accurate predictions after being deployed and that it does not suffer from overfitting or underfitting. In the final phase, we will deploy the candidate prediction model that gives better results compared to other models.

The entire project flow is explained in the below diagram.



Data Collection and Exploration

Data Exploration is an integral part of the CRISP-DM process. In this stage, the datasets are chosen according to the project requirements. Once datasets are gathered, it's important to analyze each feature for its distribution and data quality as it has direct effects on the machine learning models.

Pokémon Dataset

For this project, we have used two Pokémon datasets from Kaggle. The first dataset is the "Pokémon" dataset which is of type CSV and is of size 40kb. The Pokémon dataset has 800 Pokémon with 12 columns. This has a "#" column which represents the ID, Name column and statistics columns for different features like HP, Speed, Attack, Special Attack, Special Defense,

Defense. This dataset also has the Pokémon types and whether it is legendary Pokémon or not. The below figure is from the raw Pokémon dataset.

#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	FALSE
2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	FALSE
3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	FALSE
4	Mega Venusaur	Grass	Poison	80	100	123	122	120	80	1	FALSE
5	Charmander	Fire		39	52	43	60	50	65	1	FALSE
6	Charmeleon	Fire		58	64	58	80	65	80	1	FALSE
7	Charizard	Fire	Flying	78	84	78	109	85	100	1	FALSE
8	Mega Charizard X	Fire	Dragon	78	130	111	130	85	100	1	FALSE
9	Mega Charizard Y	Fire	Flying	78	104	78	159	115	100	1	FALSE
10	Squirtle	Water		44	48	65	50	64	43	1	FALSE

This raw file is downloaded from Kaggle and stored in google drive. Using Google Colab, the CSV file is accessed and data quality analysis is done. Firstly, the accessed CSV file is converted into a pandas data frame.

```
pokemon.head()
```

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	4	Mega Venusaur	Grass	Poison	80	100	123	122	120	80	1	False
4	5	Charmander	Fire	NaN	39	52	43	60	50	65	1	False

Data Quality Report

A data quality report can be used to ensure the quality of the data. The accuracy of the models depends upon the data quality. We have to make sure that the data is relevant and doesn't contain any inconsistencies. We have two separate tables one for each continuous variable and categorical variable.

Continuous Report

A Continuous report is generated to analyze the dataset and identify the quality issues like missing values, cardinality problems, outliers.

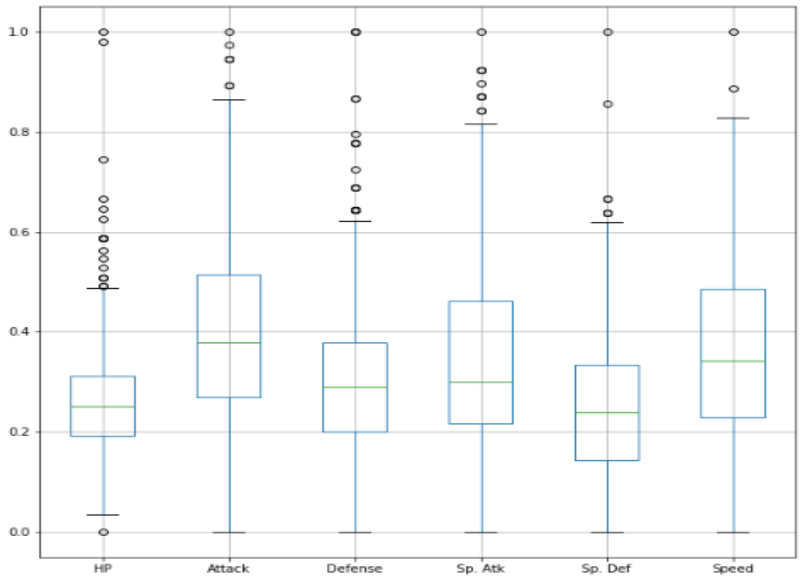
From the below Continuous report, it is clear that the data collected was pretty clean. There are no missing values, the cardinality seems to be correct. The range between the third quartile and the maximum value is a bit high. The general range seems to be between 5 and 200, whereas the standard deviation is around 25-30.

FEATURE NAME	Count	Miss %	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt	Max	Std. Dev.
HP	800	0	94	1	50	69.26	65	80	255	25.53
Attack	800	0	111	5	55	79	75	100	190	32.46
Defense	800	0	103	5	50	73.84	70	90	230	31.18
Sp. Atk	800	0	105	10	49.75	72.82	65	95	194	32.72
Sp. Def	800	0	92	20	50	71.9	70	90	230	27.83
Speed	800	0	108	5	45	68.28	65	90	180	29.06

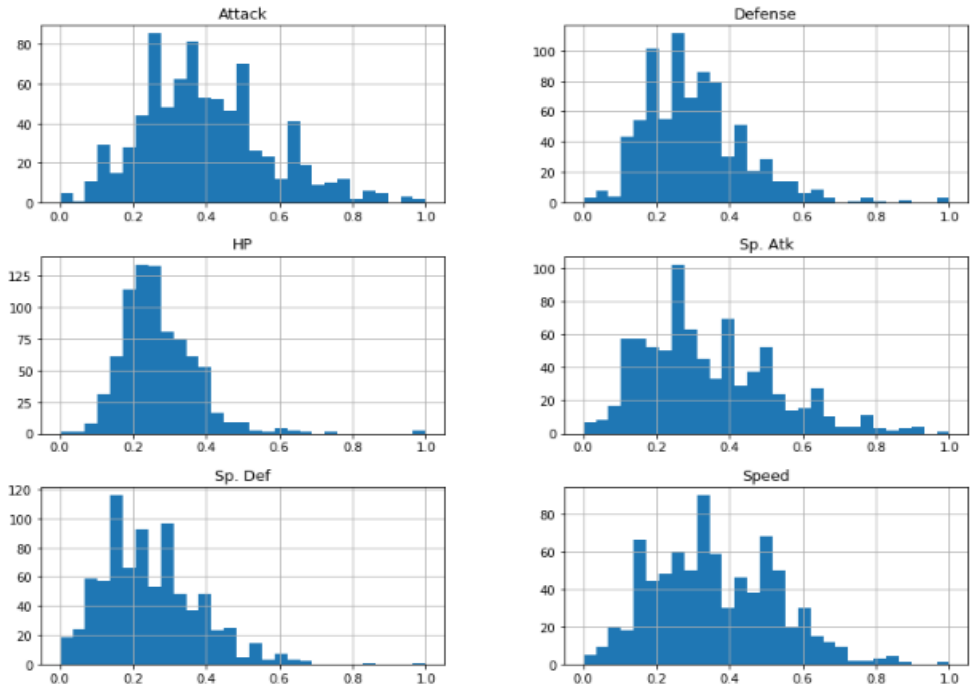
Box Plot & Histograms

A box plot is generated for the continuous features to analyze the outliers that are identified during the data quality report. There are a few outliers for all the features.

```
joined.boxplot(['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed'],figsize = (10,10))
```



The histograms are generated using Freedman–Diaconis rule. This generated bin count in the range of 25-30 for all the features. So, 29 is chosen as the bin size for all the features. Using 29 as bin count and equal-width binning methodology below histograms are generated.



It is clear that only HP is unimodal right-skewed and all others are multimodal distributions. The right skewing is because of the outliers.

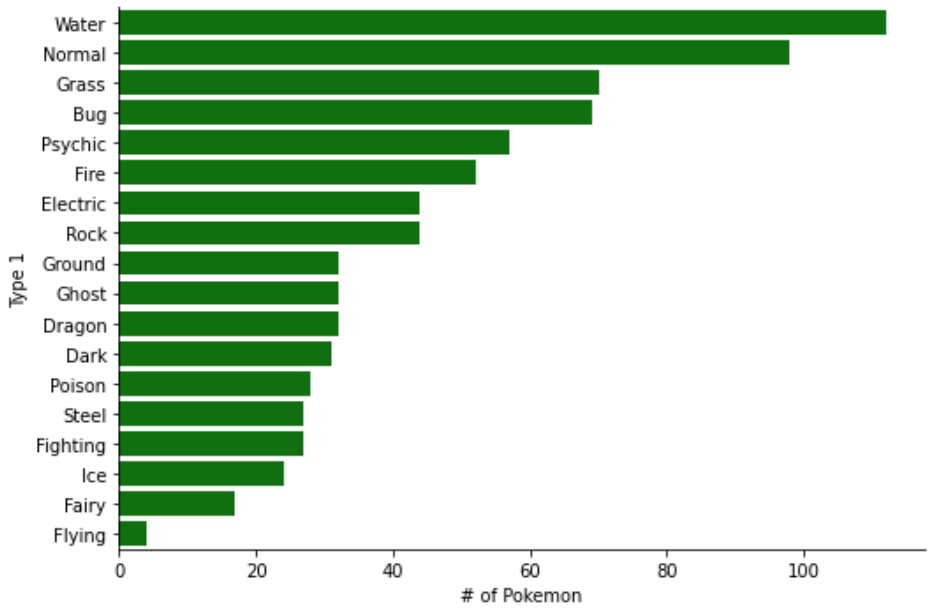
Categorical Report

The data quality report for Categorical features is also generated. The below figure shows that one of the Pokémon is missing "Name" and almost 38 Pokémon are missing "Type 2."

FEATURE NAME	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2nd Mode Freq	2nd Mode %
#	800	0	800	800	1	0.12	263	1	0.12
Name	799	0	799	Sawk	1	0.13	Cobalion	1	0.13
Type 1	800	0	18	Water	112	14	Normal	98	12.25
Type 2	414	0	18	Flying	97	23.43	Ground	35	8.45
Generation	800	0	6	1	166	20.75	5	165	20.62
Legendary	800	0	2	FALSE	735	91.88	TRUE	65	8.12

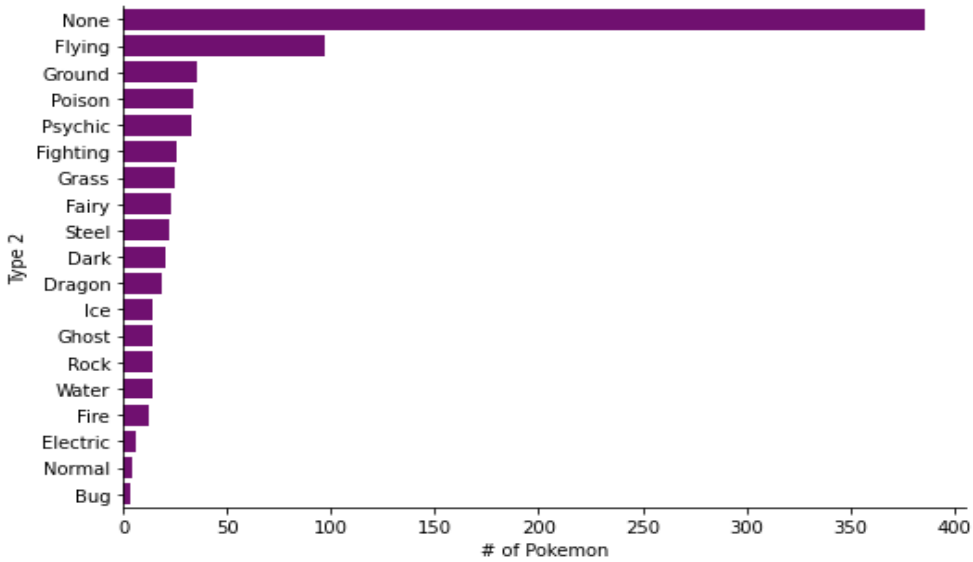
The Cardinality of Type 1 and Type 2 indicate that Pokémon are of 18 types. The below figure clearly indicates that every 1 in 4 Pokémon have their Type 1 as either Water and Normal.

```
#Barplots for categorical variables
sns.factorplot(
    y='Type 1',
    data=pokemon,
    kind='count',
    order=pokemon['Type 1'].value_counts().index,
    aspect=1.5,
    color='green'
).set_axis_labels('# of Pokemon', 'Type 1')
```



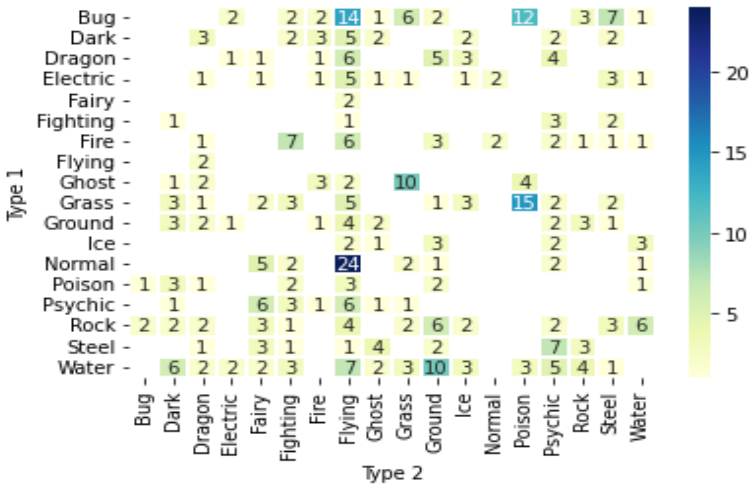
When Type 2 is analyzed based on the count, it is dominated by the “Flying” type.

```
sns.factorplot(
    y='Type 2',
    data=pokemon,
    kind='count',
    order=pokemon['Type 2'].value_counts().index,
    aspect=1.5,
    color='purple'
).set_axis_labels('# of Pokemon', 'Type 2');
```



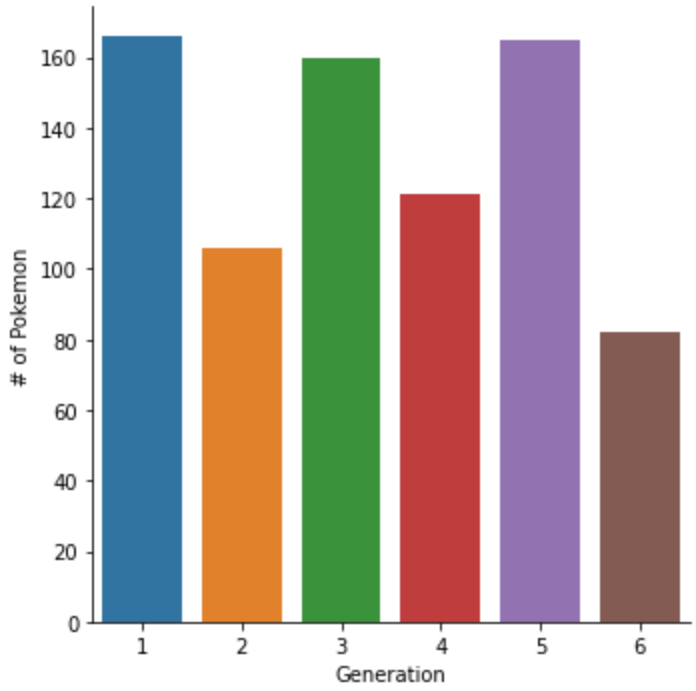
As per the below heatmap, “Normal” as Type 1 and “Flying” as Type 2 seems to be the popular combination.

```
dual_types = pokemon[pokemon['Type 2'] != 'None']

sns.heatmap(
    dual_types.groupby(['Type 1', 'Type 2']).size().unstack(),
    linewidths=1,
    annot=True,cmap="YlGnBu"
);
```

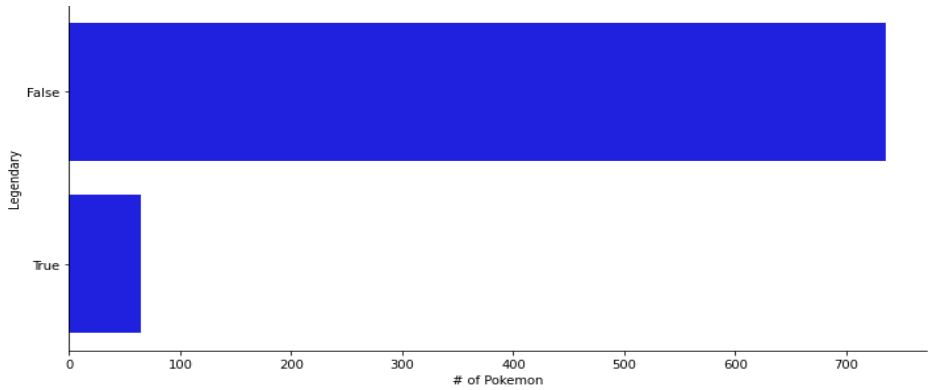


The cardinality of the generation feature indicates that there are 6 generations of Pokémon.

```
#pokemon distrbution across generations
import seaborn as sns
sns.factorplot(
    x='Generation',
    data=pokemon,
    kind='count'
).set_axis_labels('Generation', '# of Pokemon');
```



The Legendary Pokémon are difficult to collect, hence they are only 8%



Data Quality Plan

A data quality plan defines the potential steps to be taken to have good data quality. We make use of the data quality report that we have prepared in the previous section.

Missing Values

- 1 Pokémon which is missing its name.
- 386 Pokémon are missing their Type 2 value.

Feature	Data Quality Issue	Potential Handling Strategies
Name	1 pokemon's name is missing	Get the name from a different dataset
Type 2	Has 386 null values	Null values are valid, so just replace null with "None"

Outliers

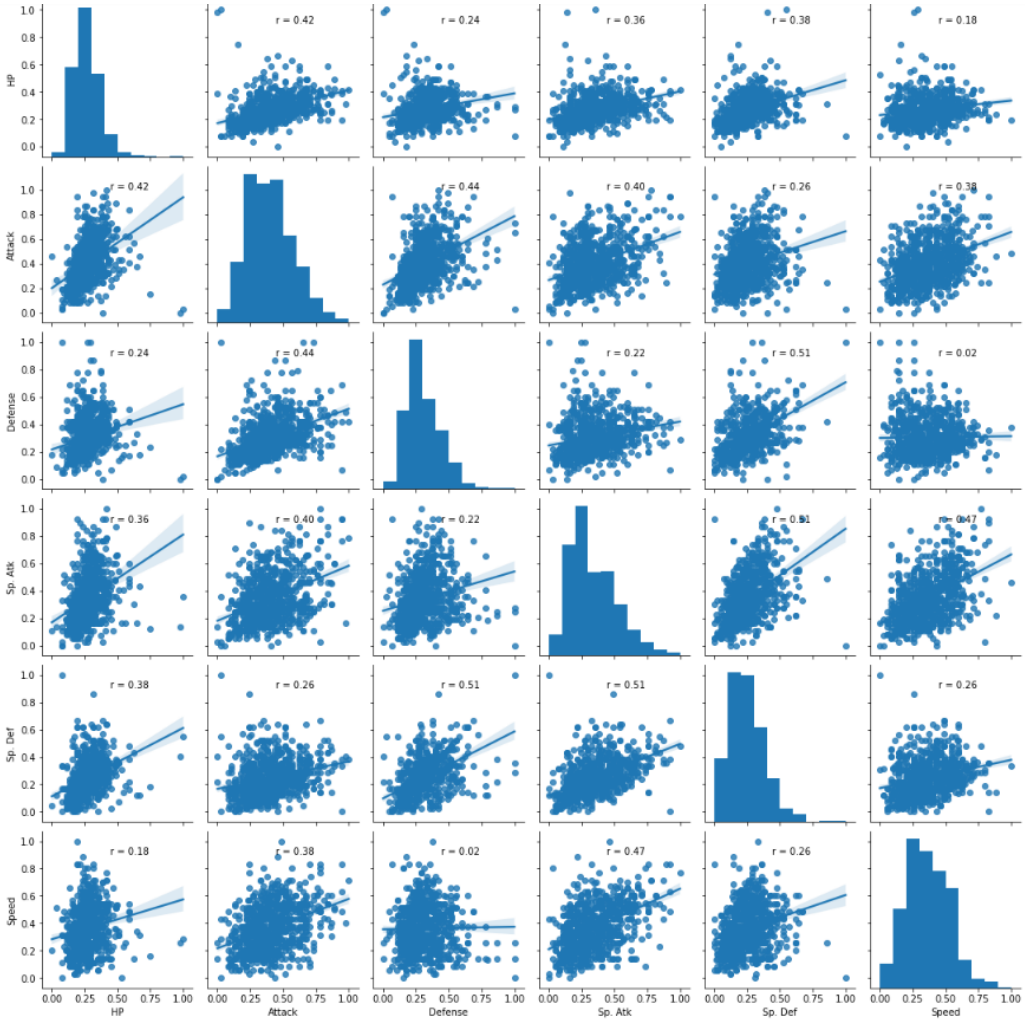
All the outliers identified are valid, so no action is required.

Advanced-Data Exploration

Visualizing Relationships between Continuous Features

A correlation matrix plot is generated to understand the relationship between different continuous features.

```
from scipy import stats
def show_corr(x, y, **kwargs):
    (r, _) = stats.pearsonr(x, y)
    ax = plt.gca()
    ax.annotate(
        'r = {:.2f}'.format(r),
        xy=(0.45, 0.85),
        xycoords=ax.transAxes
    )
sns.pairplot(
    data=joined.loc[:, 'HP':'Speed'],
    kind='reg'
).map_offdiag(show_corr);
```



	#	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
#	1.000	0.098	0.103	0.095	0.089	0.086	0.012
HP	0.098	1.000	0.422	0.240	0.362	0.379	0.176
Attack	0.103	0.422	1.000	0.439	0.396	0.264	0.381
Defense	0.095	0.240	0.439	1.000	0.224	0.511	0.015
Sp. Atk	0.089	0.362	0.396	0.224	1.000	0.506	0.473
Sp. Def	0.086	0.379	0.264	0.511	0.506	1.000	0.259
Speed	0.012	0.176	0.381	0.015	0.473	0.259	1.000

It is clear that all continuous features are related to each other, where the value of r is positive. But this relationship is not strong enough as none of them have a value of more than 0.7

Combat Dataset

Combat is the second dataset that details all the battles between different Pokémon. This dataset file is of CSV with 50000 battle details and is of 1.1MB memory. Below is the screenshot of the raw file.

First_pokemon	Second_pokemon	Winner
266	298	298
702	701	701
191	668	668
237	683	683
151	231	151
657	752	657
192	134	134
73	545	545
220	763	763

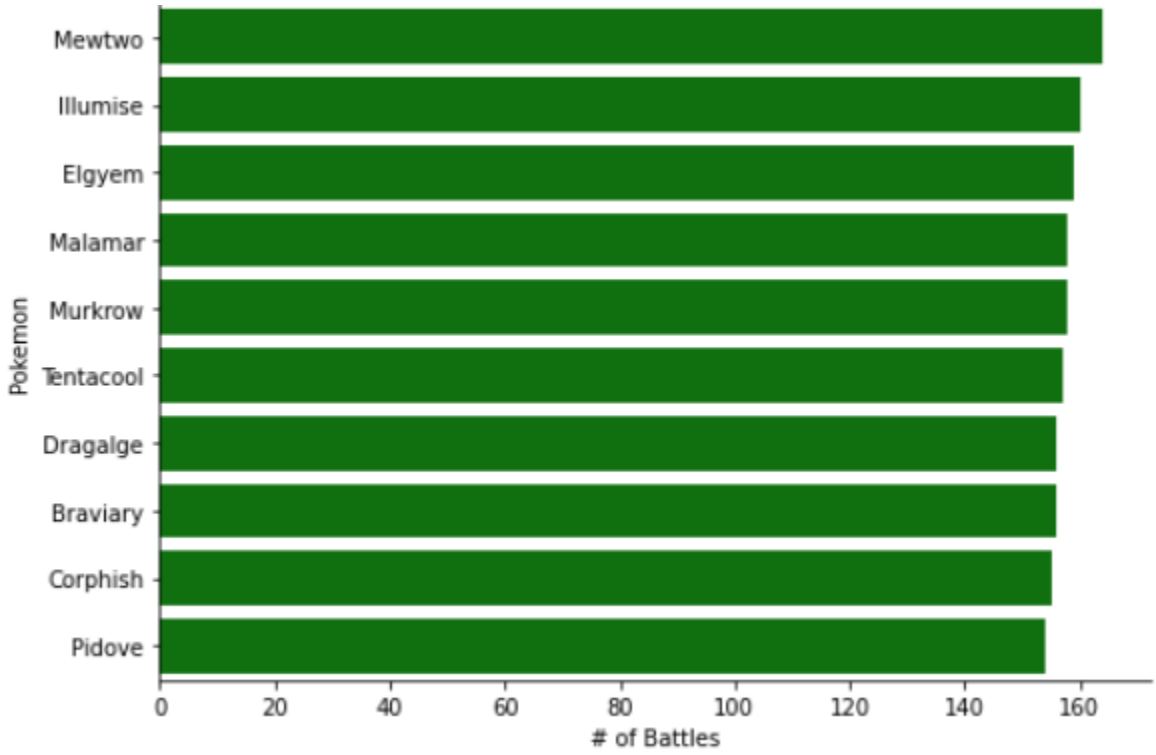
Similar to the Pokémon dataset, this dataset is also loaded to google drive and accessed using Google Colab. Once we have converted to the data frame, we can see that there are 3 columns, the first Pokémon, 2nd Pokémon, and the third column gives the id of the winner Pokémon. This dataset is completely neat and doesn't have any missing values or invalid data.

Determining if the Pokémon dataset has a simple linear Relationship

To check if the continuous features are having a simple linear relationship with the target label, the data is normalized and a new field "Strength" is generated by adding all the continuous feature values.

The below figure represents the Top 10 Pokémon who won the highest number of battles:

```
#Pokemons with highest number of wins
sns.factorplot(
    y='Winner',
    data=df,
    kind='count',
    order=df['Winner'].value_counts().iloc[:10].index,
    aspect=1.5,
    color='green'
).set_axis_labels('# of Wins', 'Pokemon')
```



But when the strength of these top Pokémon is compared it is clear that mere strength is not the deciding factor for the winner. Because when we check the strength of “Murkrow” Pokémon, the strength is just 1.833 which is very less but it is one of the top 5 Pokémon with the most wins.

```
pokicat=pokemon[['#', 'Name', 'Type 1', 'Type 2', 'Generation', 'Legendary']]
pokidigit = pd.merge(
    pokemon,
    pokicat,
    on='#'
).loc[:, ['#', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed']]

#Normalization
std_stats = pokidigit.set_index('#').apply(
    lambda x: (x - min(x)) /(max(x)-min(x)))
std_stats['strength'] = std_stats.sum(axis='columns')
```

```
#Top winners and their statistics
joined[joined.Name.isin(['Mewtwo','Aerodactyl','Infernape','Jirachi','Slaking',
'Deoxys Speed Forme','Mega Ansol','Murkrow','Mega Houndoom','Mega Aerodactyl'])].sort_values('strength', ascending=False)
```

	#	Name	Type 1	Type 2	Generation	Legendary	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	strength
162	163	Mewtwo	Psychic	None	1	True	0.413	0.568	0.378	0.783	0.333	0.714	3.189
313	314	Slaking	Normal	None	3	False	0.587	0.838	0.422	0.462	0.214	0.543	3.066
154	155	Mega Aerodactyl	Rock	Flying	1	False	0.311	0.703	0.356	0.326	0.357	0.829	2.881
431	432	Deoxys Speed Forme	Psychic	None	3	True	0.193	0.486	0.378	0.462	0.333	1.000	2.852
248	249	Mega Houndoom	Dark	Fire	2	False	0.291	0.459	0.378	0.707	0.333	0.629	2.797
427	428	Jirachi	Steel	Psychic	3	True	0.390	0.514	0.422	0.489	0.381	0.543	2.738
437	438	Infernape	Fire	Fighting	4	False	0.295	0.535	0.293	0.511	0.243	0.589	2.466
153	154	Aerodactyl	Rock	Flying	1	False	0.311	0.541	0.267	0.272	0.262	0.714	2.366
213	214	Murkrow	Dark	Flying	2	False	0.232	0.432	0.164	0.408	0.105	0.491	1.833

So to determine the relationship between all the features and target labels, we will have to use machine learning.

Data Preprocessing

Data preparation is one of the important steps in the CRISP-DM methodology. The data exploration from the previous step is used to clean and transform the data. Our project data is substantially stable and needs less effort to prepare before passing it to the models. This data preprocessing is done in two steps - Cleaning and Transforming.

Cleaning and filtering

Generally, the cleaning process involves identifying and handling outliers, noise, empty and irrelevant data. As we can see from the Boxplot that they all are valid outliers, we are not performing any mechanism to handle them. Also, from the correlation matrix, we observe that all the features are considerably correlated. This eliminates the need of dropping unnecessary features of the data. The only thing we need to handle is the missing values in the dataset. That is done in two steps.

1. One of the values for feature 'Name' was missing in the dataset. We don't want to remove that row from the Pokémon dataset as that Pokémon has significant entries in the battles dataset. So, we picked the name of that particular Pokémon manually from another dataset by matching the feature values and replaced it with the existing empty cell.

```
1  pokemon['Type 2'].isna().sum()
```

```
386
```

2. The above figure shows there are 386 null values in the 'Type 2' column. This could be possible because some of the Pokémon may be of only one type such as Grass, Fire, Poison, etc. As such, we replace the null values with 'None'.

```
# renaming the nameless pokemon
pokemon['Name'].fillna(value='Primeape', inplace=True)
#Replace type 2 null values with "None"
pokemon['Type 2'].fillna(value='None', inplace=True)
```

Transformation

After the datasets are cleaned they need to be transformed into the desired format. Machine learning models do not accept any other type of values except numerics. So, we need to do some data engineering.

The 'Legendary' column is of binary type. We need to convert them into integer values such that False is represented as 0 and True is represented as 1.

```
# Convert "Legendary" column, False is converted to 0 and True is converted to 1.
pokemon["Legendary"] = pokemon["Legendary"].astype(int)
```

Hashing Trick and Feature Hasher

Further, we need to convert 'Type 1' and 'Type 2' columns into continuous variables. For that, we used a hashing trick called 'FeatureHasher' of the sci-kit learn library. Hashing trick is a method that converts arbitrary features into sparse binary vectors. It uses a hashing function to convert the input into numbers. Feature hashing maps each categorical value to an integer within a predetermined range. FeatureHasher class converts strings into scipy.sparse matrices. Binary values are used as-is but, Unicode strings are converted into UTF-8 encoding. Here the hash function used is the signed 32-bit version of the Murmurhash3. This class is an efficient low-memory algorithm designed for large-scale learnings.

class sklearn.feature_extraction.FeatureHasher(*n_features=1048576, *, input_type='dict', dtype=<class 'numpy.float64'>, alternate_sign=True*) [source]

Here, the n_features parameter represents the number of unique columns in the output matrix and we have set that value to 4, and input_type is set to string as per our requirement in the project.

```
# covert strings to integers
h1 = FeatureHasher(n_features=4, input_type='string')
h2 = FeatureHasher(n_features=4, input_type='string')
d4 = h1.fit_transform(pokemon["Type 1"])
d5 = h2.fit_transform(pokemon["Type 2"])
```

```
f = d4.toarray()

unq = np.unique(f, axis=0)
print("nFeature Shape:", f.shape)
print("nUnique Shape:",unq.shape)
```

```
nFeature Shape: (800, 4)
nUnique Shape: (18, 4)
```

```
f1 = d5.toarray()

unq1 = np.unique(f1, axis=0)
print("nFeature Shape:", f1.shape)
print("nUnique Shape:",unq1.shape)
```

```
nFeature Shape: (800, 4)
nUnique Shape: (19, 4)
```

After hashing is done, we convert Type 1 and Type 2 into arrays and attach them to the Pokémon data frame. We no longer require Type 1 and Type 2 columns. So, we drop them from the data frame.

```
# Convert to dataframe
d1 = pd.DataFrame(data=d4.toarray())
d2 = pd.DataFrame(data=d5.toarray())

# Drop Type 1 and Type 2 column from Pokemon dataset and concatenate the above two dataframes.
pokemon = pokemon.drop(columns = ["Type 1", "Type 2"])
pokemon = pd.concat([pokemon, d1, d2], axis=1)
```

	#	Name	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	0	1	2	3	0	1	2	3
0	1	Bulbasaur	45	49	49	65	65	45	1	0	0.0	2.0	1.0	-2.0	-1.0	-2.0	0.0	1.0
1	2	Ivysaur	60	62	63	80	80	60	1	0	0.0	2.0	1.0	-2.0	-1.0	-2.0	0.0	1.0
2	3	Venusaur	80	82	83	100	100	80	1	0	0.0	2.0	1.0	-2.0	-1.0	-2.0	0.0	1.0
3	4	Mega Venusaur	80	100	123	122	120	80	1	0	0.0	2.0	1.0	-2.0	-1.0	-2.0	0.0	1.0
4	5	Charmander	39	52	43	60	50	65	1	0	0.0	0.0	-1.0	1.0	0.0	0.0	1.0	-1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
795	796	Diancie	50	100	150	100	150	50	6	1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
796	797	Mega Diancie	50	160	110	160	110	110	6	1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
797	798	Hoopa Confined	80	110	60	150	130	70	6	1	0.0	-5.0	1.0	-1.0	0.0	0.0	0.0	-1.0
798	799	Hoopa Unbound	80	160	60	170	130	80	6	1	0.0	-5.0	1.0	-1.0	0.0	1.0	2.0	-1.0
799	800	Volcanion	80	110	120	130	90	70	6	1	0.0	0.0	-1.0	1.0	-1.0	1.0	1.0	0.0

800 rows × 18 columns

The Pokémon dataset now needs to be combined with the battles dataset to form into one single data frame. For this, the First_Pokémon and Second_Pokémon columns of the battles dataset are mapped with the '#' column of the Pokémon dataset, and details of each Pokémon are retrieved. Then the details of both the Pokémons are combined along with the winner column of the battles dataset. This forms our final dataset.

```
data = []
i = 0
for t in battles.itertuples():
    i += 1
    #print(i)
    first_pokemon = t[1]
    second_pokemon = t[2]
    winner = t[3]

    x = pokemon.loc[pokemon["#"]==first_pokemon].values[:, 2:][0]
    y = pokemon.loc[pokemon["#"]==second_pokemon].values[:, 2:][0]
    diff = (x-y)[:6]
    z = np.concatenate((x,y))

    if winner == first_pokemon:
        z = np.append(z, [0])
    else:
        z = np.append(z, [1])

    data.append(z)
```

```
data = np.asarray(data)
```

Each item in the dataset will be in the below format in the data frame.

```
[78 104 78 159 115 100 1 0 1.0 -1.0 0.0 -1.0 1.0 0.0 -1.0 0.0 -1.0 0.0 70
 80 50 35 35 35 1 0 -1.0 -2.0 -2.0 -1.0 -2.0 0.0 0.0 0.0 0.0 0.0 0]
```

Data Preparation

After the data is cleaned and transformed, we have to prepare individual datasets for training and testing. 75% of the data is taken to train the models. This is the input we pass to the models, which in turn uses them to learn from them and predicts the unknown dataset's target value for us. The remaining 25% of the data is used to test the models. We can use this testing set to verify the accuracy against the predicted output.

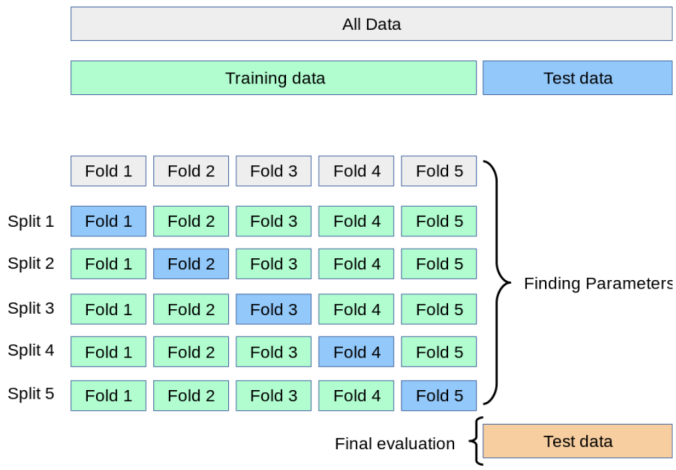
```
X = data[:, :-1].astype(int)
y = data[:, -1].astype(int)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Data Modeling and Evaluation

The next step in the CRISP-DM process after preparation is the modeling of the processes. This is the real phase where we develop our models with training data. These models learn from the existing results and predict the unknown data points for us.

Model Proposals using K-fold Cross-Validation

It is important that the Machine Learning model built gives reliable results. The test dataset results may not be truthful. We cannot rely on one single time split of the training and testing data. The data needs to be split multiple times randomly into training and testing data to ensure the accuracy of model performance. For this purpose, we used the K-fold cross-validation method to choose our models. When we use K-fold, we are actually testing how well a model



can learn from the training data and predict the test data. This can eliminate the issue of overfitting.

The dataset is divided into the number(k) selected by the user. The model is split as many as the number of parts, each part is called fold. A model is trained with k-1 folds and the remaining is used as testing data.

Our project dataset is trained and tested for models such as Logistic regression (LR), K Nearest Neighbors (KNN), Gaussian Naive Bayes (GNB), Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), and AdaBoost Ensemble with 10 folds. We used a cross-validation score metric that uses the roc_auc as the criteria to measure the accuracy of each model.

```
# prepare models
models = []

# Classifiers to use in Voting
boost = xgb.XGBClassifier()
dtc = DecisionTreeClassifier(max_depth=5, criterion='entropy')
lrg = LogisticRegression(solver='liblinear')
vote = VotingClassifier(estimators=[('boost', boost), ('dtc', dtc), ('lrg', lrg)], voting='soft')

models.append(('LRG', LogisticRegression(solver='liblinear', max_iter=250)))
models.append(('KNB', KNeighborsClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RFC', RandomForestClassifier(random_state=0, n_estimators=100)))
models.append(('DTC', DecisionTreeClassifier(random_state=0, criterion='entropy', max_depth=5)))
models.append(('SVC', LinearSVC(random_state=0,dual=False)))
models.append(('ADA', AdaBoostClassifier(random_state=0)))

# evaluate each model by average of roc auc
results_dict = {}

for name, m in models:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(m, X, y, cv=kfold, scoring = 'roc_auc') # or 'accuracy'
    results_dict.update({name: "%f" % cv_results.mean()})
```

```
# results_dict
sorted(((v,k) for k,v in results_dict.items()), reverse=True)
```

```
[('0.984629', 'RFC'),
 ('0.954549', 'DTC'),
 ('0.937886', 'KNB'),
 ('0.927646', 'LRG'),
 ('0.927060', 'SVC'),
 ('0.917330', 'ADA'),
 ('0.842657', 'GNB')]
```

From the above figure we can observe that Random Forest has the highest performance rate. As such, we focus mainly on that. For comparison and justification we use other models like Decision Tree, Logistic Regression and AdaBoost.

Evaluation

It is important to know how our model performs. For this we implement different performance measures that are apt for the model. For our project we have used metrics such as accuracy scores, classification report, auc_curve, and confusion matrix for each model.

Accuracy Score

It is the ratio of the number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Confusion matrix

This gives the matrix as the output which gives the complete performance of the model. There are 4 important terms in the matrix.

- True Positives: The number of cases that predict YES and the expected output was also YES.
- True Negatives: The number of cases that predict NO and the expected output was also NO.
- False Positives: The number of cases that predict YES and the expected output was NO.
- False Negatives: The number of cases that predict NO and the expected output was YES.

The accuracy of the matrix can be calculated by taking the average of the values along the diagonal.

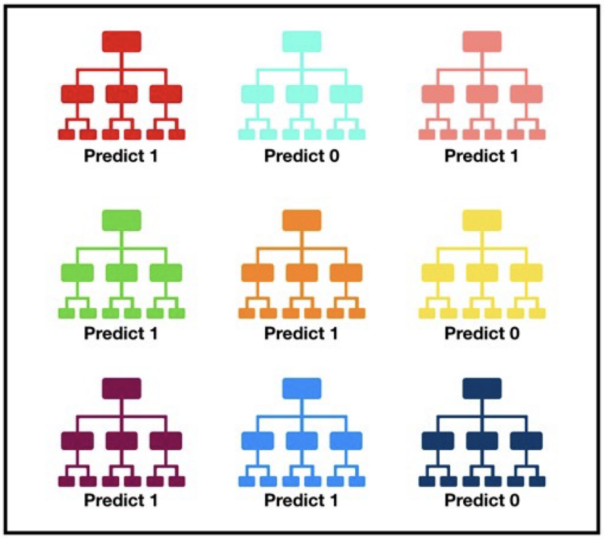
$$Accuracy = \frac{TruePositive + TrueNegative}{TotalSample}$$

Area Under Curve (AUC)

This is mostly used for binary classification of the results. As our dataset predicts whether a Pokémon is a winner or not which represents a binary outcome, this metric best suits our use case. It is the probability that the model ranks a randomly chosen positive example higher than a negative example. In our project, we compare the True Positive Rate (TPR) and False Positive Rate (FPR). The value range for AUC is between 0 and 1. The higher the value of auc_score, the better is the model.

Random Forest (RF)

Random Forest is an ensemble technique that is a collection of multiple decision trees. Each decision tree gives the prediction output of a class and the class with the most number of votes becomes the outcome of the random forest.



Sci-kit learn's RandomForestClassifier is used for our model development. It has different parameters which can tune the performance of the model. We concentrate only on the n_estimators parameter which denotes the number of decision trees to be used by a random forest classifier. We set this value to 100 and then fit the model with X_train and Y_train. We then provide evaluation metrics.

> *class* sklearn.ensemble.RandomForestClassifier(*n_estimators=100*, ***, *criterion='gini'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features='auto'*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *bootstrap=True*, *oob_score=False*, *n_jobs=None*, *random_state=None*, *verbose=0*, *warm_start=False*, *class_weight=None*, *ccp_alpha=0.0*, *max_samples=None*) [source]

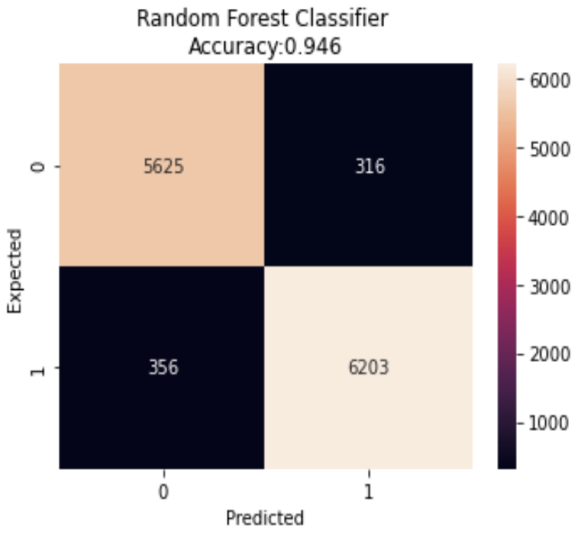
```python
# Random forest
rf_clf = RandomForestClassifier(n_estimators=100)
rf_model = rf_clf.fit(X_train,y_train)
rf_pred = rf_model.predict(X_test)
print(f"Expected : {y_test}")
print(f"Predicted : {rf_pred}")
```

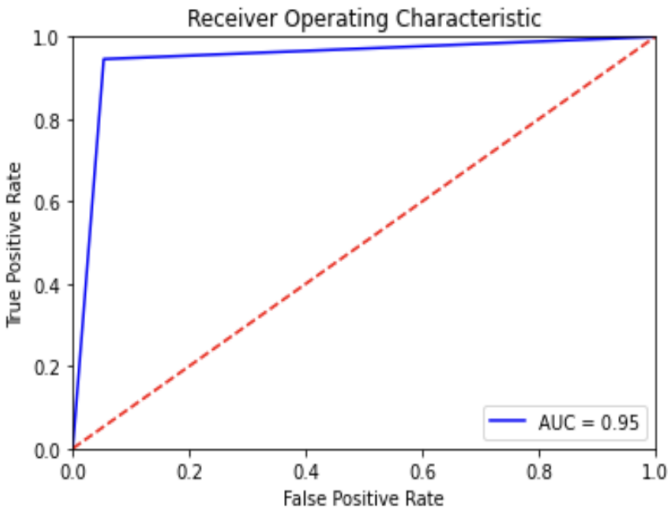
```
Expected : [1 0 0 ... 1 1 1]
Predicted : [1 0 0 ... 1 1 1]
```

```
Accuracy : 0.94848
              precision    recall  f1-score   support

           0       0.94      0.95      0.95      5941
           1       0.95      0.95      0.95      6559

    accuracy                           0.95     12500
   macro avg       0.95      0.95      0.95     12500
weighted avg       0.95      0.95      0.95     12500
```

```python
print(roc_auc_score(y_test, rf_model.predict_proba(X_test)[:,1]))
```

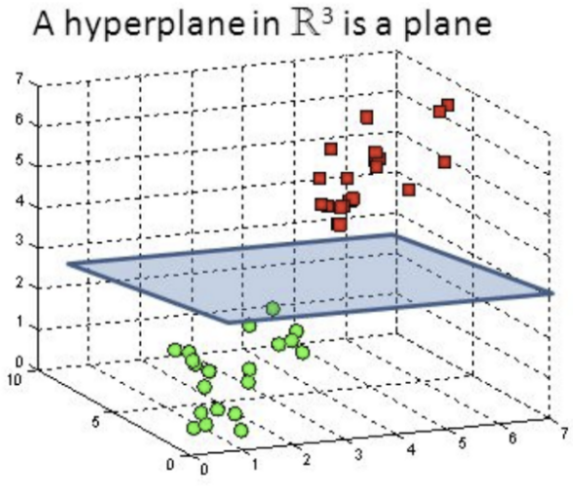
```
0.9828608649791762
```





From the above metrics, we can see that Random Forest has an accuracy rate of 94.8% and the value of auc_score is 0.95. This indicates that random forest is a good model to consider for this dataset.

Support Vector Machine (SVM)



The functionality of the Support Vector Machine involves separating the data points according to groups with the help of a linear regression line called Hyperplane. This SVM is highly useful when we have large dimensional space. To get the optimal hyperplane make sure that the distance between the hyperplane and the nearest data point on both the sides of the hyperplane which are called support vectors should be maximum. The dimension of the hyperplane depends on the number of features.

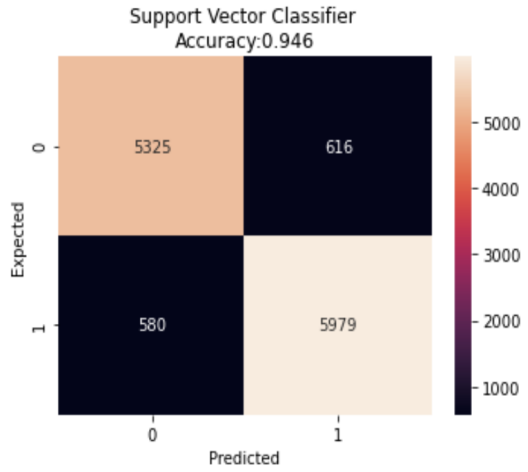
For our project, we use sci-kit learn's Support Vector Classifier module.

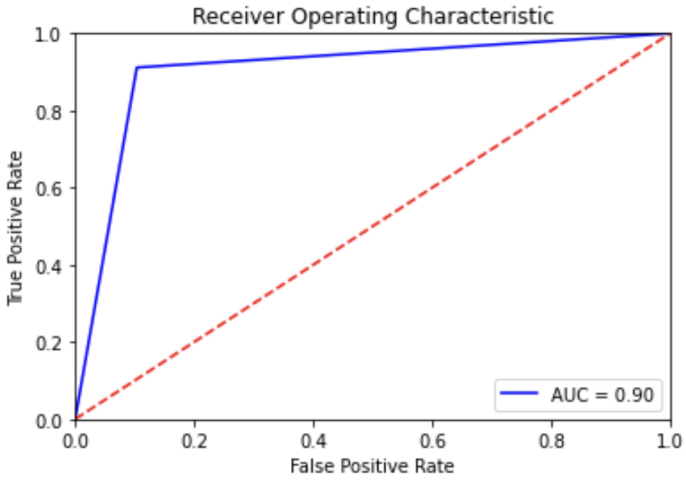
class sklearn.svm.SVC(, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)* [source]

This contains a number of parameters where we concentrate only on the kernel which handles the view in the n-dimensional space. We use the default value 'rbf'. The data needs to be standardized before fitting the model and then should be passed to the model. After we fit the data into the model we can predict unknown data. We finally provide evaluation metrics for the SVM.

```
svm_clf = make_pipeline(StandardScaler(), svm.SVC(kernel='rbf'))
svm_model = svm_clf.fit(X_train,y_train)
svm_pred = svm_model.predict(X_test)
print(f"Expected : {y_test}")
print(f"Predicted : {svm_pred}")
```

```
Expected : [1 0 0 ... 1 1 1]
Predicted : [1 0 0 ... 1 1 1]


Accuracy : 0.90432
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      5941
           1       0.91      0.91      0.91      6559

    accuracy                           0.90     12500
   macro avg       0.90      0.90      0.90     12500
weighted avg       0.90      0.90      0.90     12500
```

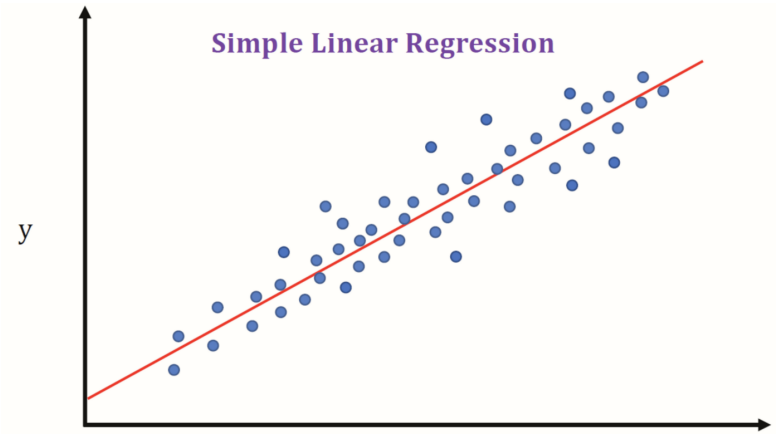




From the evaluation of the above metrics, we can see that the accuracy score for SVM is 90% and the AUC value is 0.90. This determines that this model is also a good selection for our dataset. However, it is less when compared to that of Random Forest Classifier.

Logistic Regression (LR)

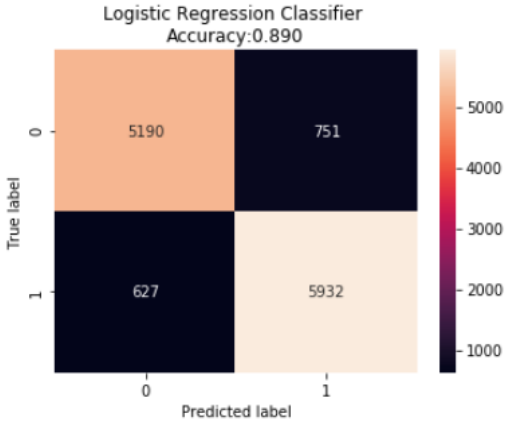
When preparing a model, complexity often comes at the cost of time and money; as stated earlier, there isn't a strong case to be made for implementing complex models when more basic and less costly models can give the results required. For those reasons, it is hard to build a binary classification model and not try implementing a logistic regression model first.



We tried and tested a wide swath of different logistic regression implementations, both with 0-1 normalization and non-normalized data. We eventually settled on using the Sci-Kit Learn package using the Lib-Linear optimization algorithm with an L2 penalty.

```
# Logistic Regression
lrg_clf =LogisticRegression(solver='liblinear', max_iter=250)
lrg_model = lrg_clf.fit(X_train,y_train)
lrg_pred = lrg_model.predict(X_test)
```

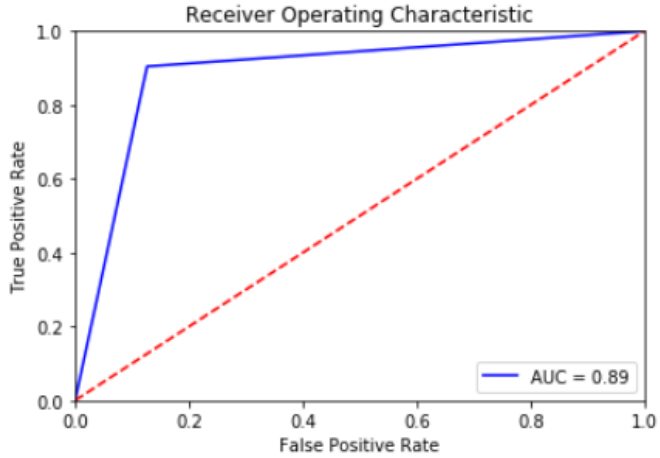
Accuracy is the goal of our models, we don't care much about sensitivity or specificity because it doesn't matter which way our models are wrong, both False Positives and False Negatives affect us the same, given that accuracy is our goal, it will be the measure we focus on.



Accuracy : 0.88976

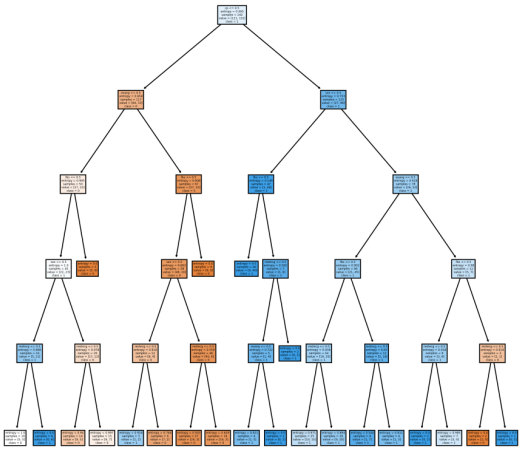
	precision	recall	f1-score	support
0	0.89	0.87	0.88	5941
1	0.89	0.90	0.90	6559
accuracy			0.89	12500
macro avg	0.89	0.89	0.89	12500
weighted avg	0.89	0.89	0.89	12500

We ended up with an accuracy score of just under 89%, this was one of the poorer models we tested, given this score, we didn't waste a lot of time trying to tune it further, instead focusing on models with strong preliminary accuracy scores.



Decision Tree (DT)

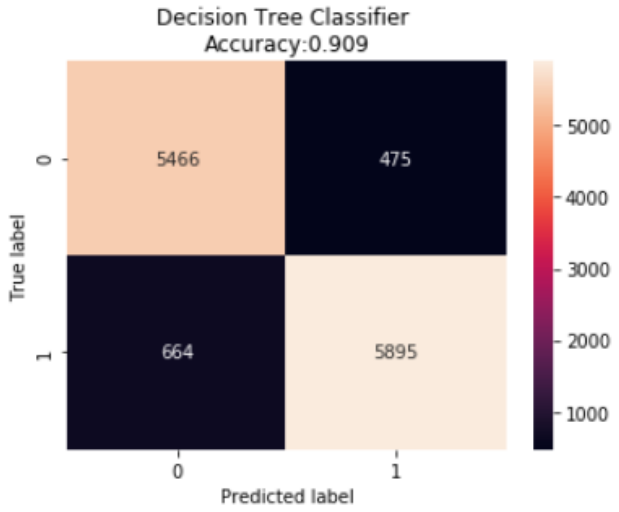
We implemented a Decision Tree Classifier model again to try and predict the winner of the Pokémon battles with the highest accuracy score. Using the Sci-Kit Learn Python package Parameters.

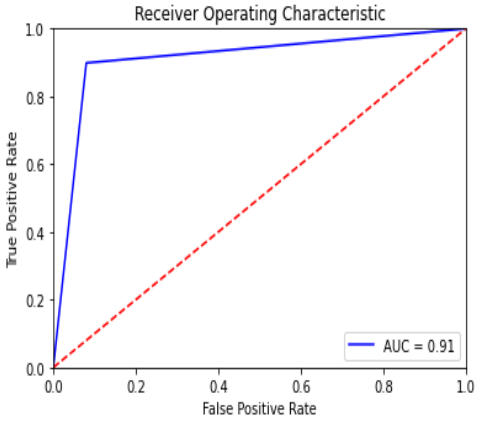


We played with the parameters in this model quite a bit, mainly with the two parameters being tweaked being the maximum depth and the split measure. We ended up setting our final tree depth to five, this gave us a nice balance of computing time and model strength. For the splitting measure, we settle on entropy rather than the Gini index. Our accuracy scores were higher using entropy so the decision was an easy one. We did try changing the model to split on random instead of the default of the 'best' split, but as usual, it didn't improve our results.

```
Accuracy : 0.90888
              precision    recall  f1-score   support

           0       0.89      0.92      0.91      5941
           1       0.93      0.90      0.91      6559

    accuracy                           0.91     12500
   macro avg       0.91      0.91      0.91     12500
weighted avg       0.91      0.91      0.91     12500
```

Looking at our results, we have a fairly balanced 91% accuracy giving us high chances of picking the correct winner of a Pokémon battle. If we had run no other models, this is something we may have been happy with, as it stands, our Random Forest model with mild tuning is already stronger and we should focus on improving that model.





Parameter Tuning

We went through extensive tuning for all models involved in this process. Nearly every user-changeable attribute in all of the Sci-Kit Learn libraries was optimized for the best results given our dataset; however, we decided to only specifically high-level tune our Random Forest model. We used the RandomSearchCV package to do just that. The RandomSearchCV parameter tuning algorithm works much like RandomSearchCV but instead of testing all possible combinations, it tests the parameters randomly based on the number of iterations set. The testing algorithm is multithreaded out of the box; that is, it can run each iteration concurrently up to the number of CPU threads it has access to. We ran the search with the recommended 100 iterations due to the compute cost of each iteration. Other parameters can be seen below.

```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

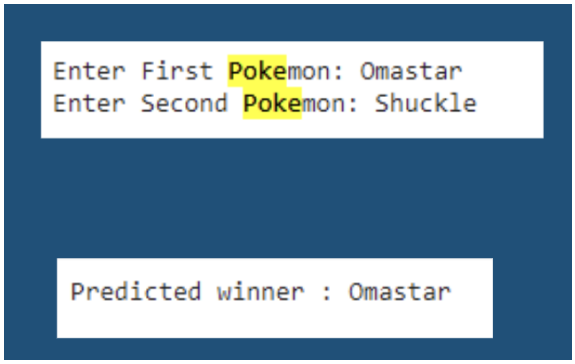
Even still, the process took nearly 30 minutes to complete on 32 x86 CPU threads. The output of our tuning can be seen below.

```
Random Forest Classifier
Accuracy:0.951
: {'n_estimators': 400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': None,
 'bootstrap': False}
```

Final Result

Our strongest model ended up being a Random Forest model with tuning done by the RandomSearchCV model. We decided to stick with focusing our tuning on the Random Forest because without tuning it started as the best model and we figured that with the minor improvements we could expect from tuning, it would still remain the strongest. RandomSearchCV gave us a small but still significant improvement on the Random Forest model we had been running. It told us our n-estimators on Random Forest should be set to 400. Given this increased tuning, it brought our accuracy score up to 95.1% from 94.8% previously.

Once we had decided that the tuned Random Forest was the model we were focusing on, we mapped the Pokémon names back to the model output so that we could actually tell which Pokémon were winning the battles.



Conclusion

We ended up building a very strong model to predict the winner of a Pokémon Go battle based upon the base stats of the two Pokémon involved. We have achieved the ability to do this with over 95% accuracy by simply inputting both Pokémon names. One of the final cherries on top of this project was building out this functionality into a small Python function. Utilizing the PyTorch library we were able to save our tool locally, it can be called upon and the function runs without the entire dataset or original models being run.

The original goal of this project was to create a tool that can be used by Pokémon Go players to help them accurately know before they battle a peer whether or not they will win. We have created said tool, there are some limitations, mainly being this is a codebase that must be run within a Python environment. Eventually, this tool could be made into a small executable, something with a basic user interface, likely something that can even be made to run in a mobile environment instead of an x86 platform.

Further functionality can be enabled; our model uses the base stats of a Pokémon, but the actual Pokémon have individual statistics applied to them, in the Pokémon world called IVs for individual values; we could utilize a model that allows input of more than just the Pokémon names but instead the names along with the IVs. Given this extra information, we could improve the model well beyond the 95% accuracy we are currently getting. Given that the battle is nearly entirely dependent on the IVs of the Pokémon, we could expect upwards of 99% accuracy of such a model.

References

1. https://towardsdatascience.com/increasing-model-reliability-model-selection-cross-validation-1ce0bf506cd
2. https://stats.stackexchange.com/questions/52274/how-to-choose-a-predictive-model-after-k-fold-cross-validation
3. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.FeatureHasher.html
4. https://scikit-learn.org/stable/modules/cross_validation.html
5. https://towardsdatascience.com/understanding-random-forest-58381e0602d2
6. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
7. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
8. https://en.wikipedia.org/wiki/Freedman%E2%80%93Diaconis_rule
9. https://www.kaggle.com/abcsds/Pokémon
10. https://www.kaggle.com/ajisamudra/winner-in-Pokémon-combat-prediction/data?select=combats.csv
11. https://en.wikipedia.org/wiki/Pok%C3%A9mon
12. https://en.wikipedia.org/wiki/Pok%C3%A9mon_(video_game_series)
13. https://www.kaggle.com/terminus7/Pokémon-challenge
14. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
15. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
16. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

17. https://cs229.stanford.edu/proj2015/249_report.pdf
18. https://cs229.stanford.edu/proj2006/BabakHamadani-PredictingNFLGames.pdf
19. https://www.kclas.ac.in/wp-content/uploads/2021/01/The-Cricket-Winner-Prediction-With-Application-Of-Machine-Learning-And-Data-Analytics-.pdf
20. https://indjst.org/download-article.php?Article_Unique_Id=INDJST12050&Full_Text_Pdf_Download=True
21. https://content.iospress.com/download/journal-of-sports-analytics/jsa200436?id=journal-of-sports-analytics%2Fjsa200436
22. Kelleher, J. D., Brian Mac Namee, & Aoife D'arcy. (2015). Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. The MIT Press.