Pokemon Battle

PREDICT THE WINNER OF A POKEMON BATTLE

Submitted To:

1. Mr. Mayur Dev Sewak
   General Manager, Operations
   Eisystem Services
2. Mrs. Mallika Srivastava
   Trainer, Data Science & Analytics Domain
   Eisystem Services

Submitted By:
Vrnika Jain
Machine Learning /
Data Science
Batch: 21DAT-088

Content Table

Link of Project (Google Colab): =
https://colab.research.google.com/drive/1NzHfAky7WmFyROPsokrw9XgU9WdYuZ5q?usp=sharing

Abstract of Project

Project Title: = Pokemon Battle

Pokemon is a turn based video game where players send out their Pokemon to battle against the opponent's Pokemon one at a time. My project attempts to analyze Pokemon's properties and predict which Pokemon can win that battle utilizing a model-free Supervised Machine Learning strategy. I found that a Feature Hasher exploration strategy with Ranfom Forest Classifier resulted in the best performance after qualitatively and quantitatively, using the win rate against a random agent, evaluating it against other approaches.

List of Figures

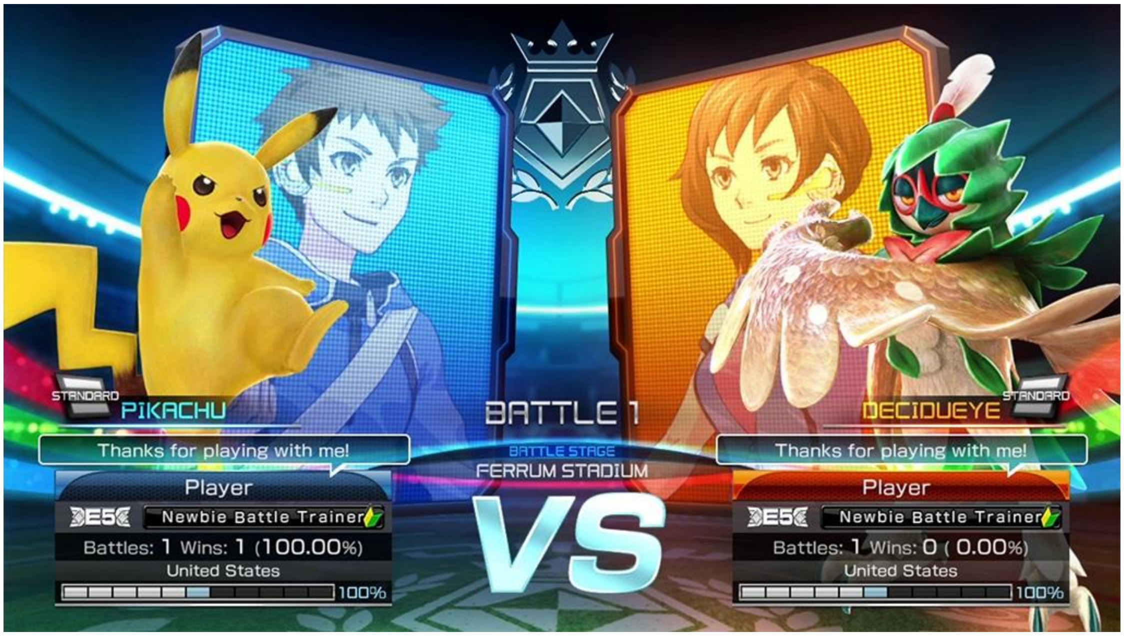


Figure 1: Image of Pokemon Battle

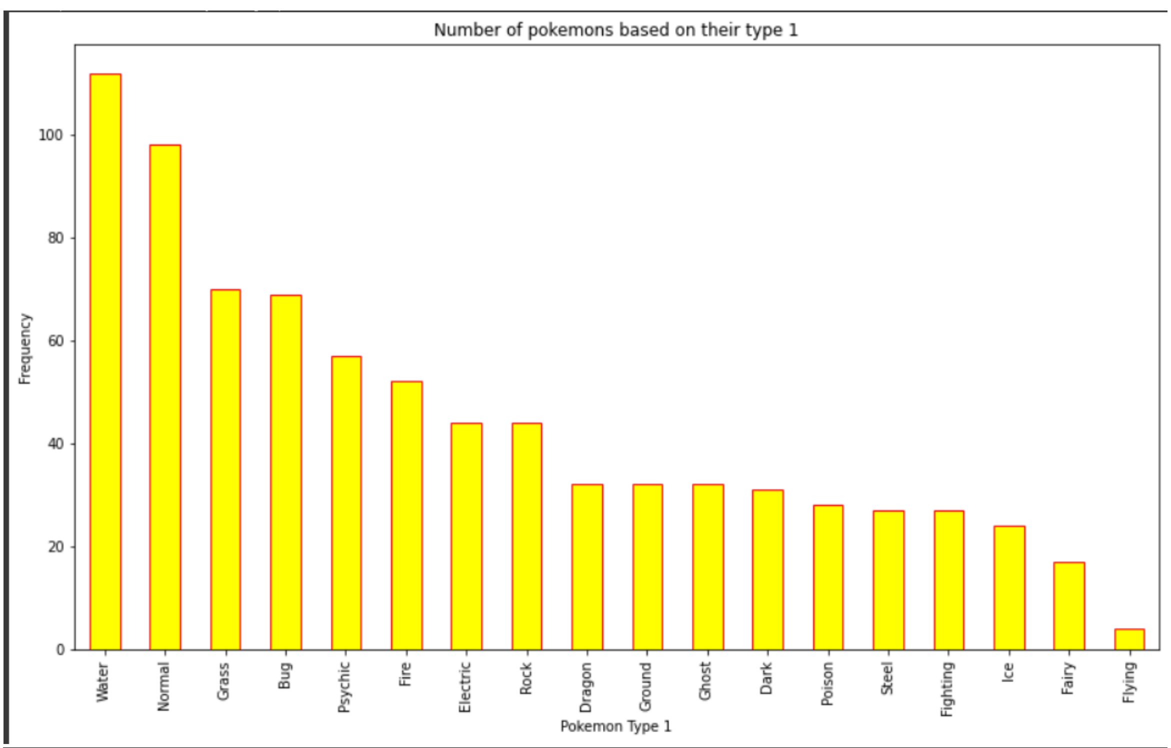


Figure 2: Number of Pokemons of Type 1

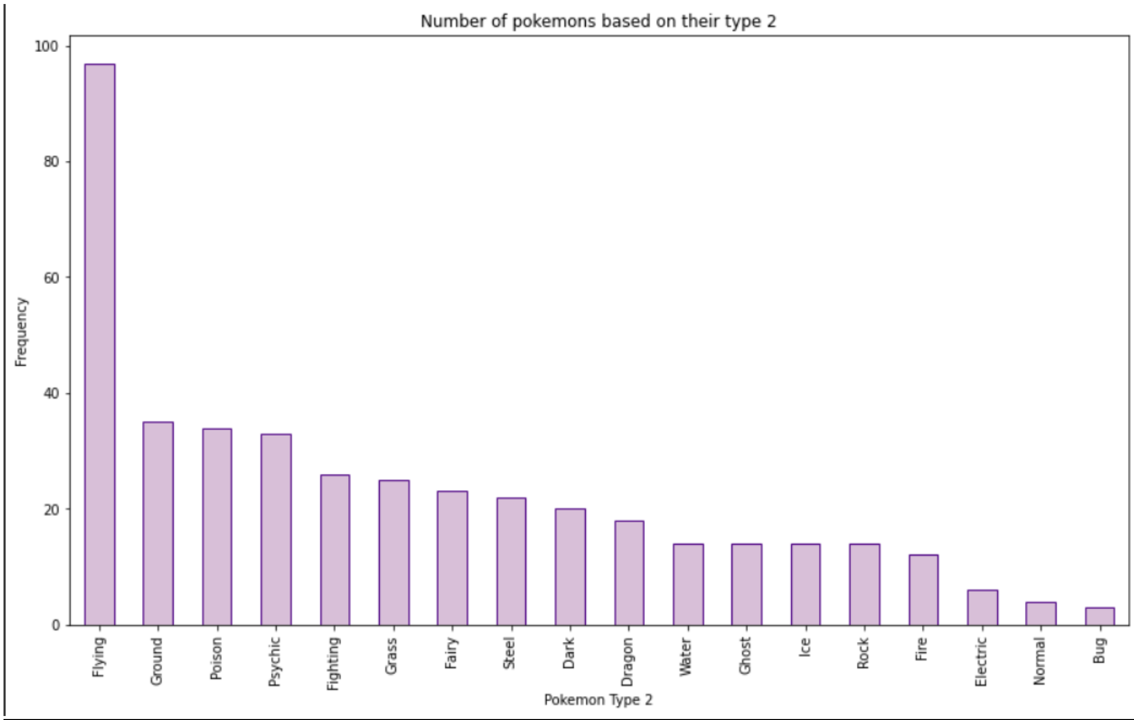


Figure 3: Number of Pokemons of Type 2

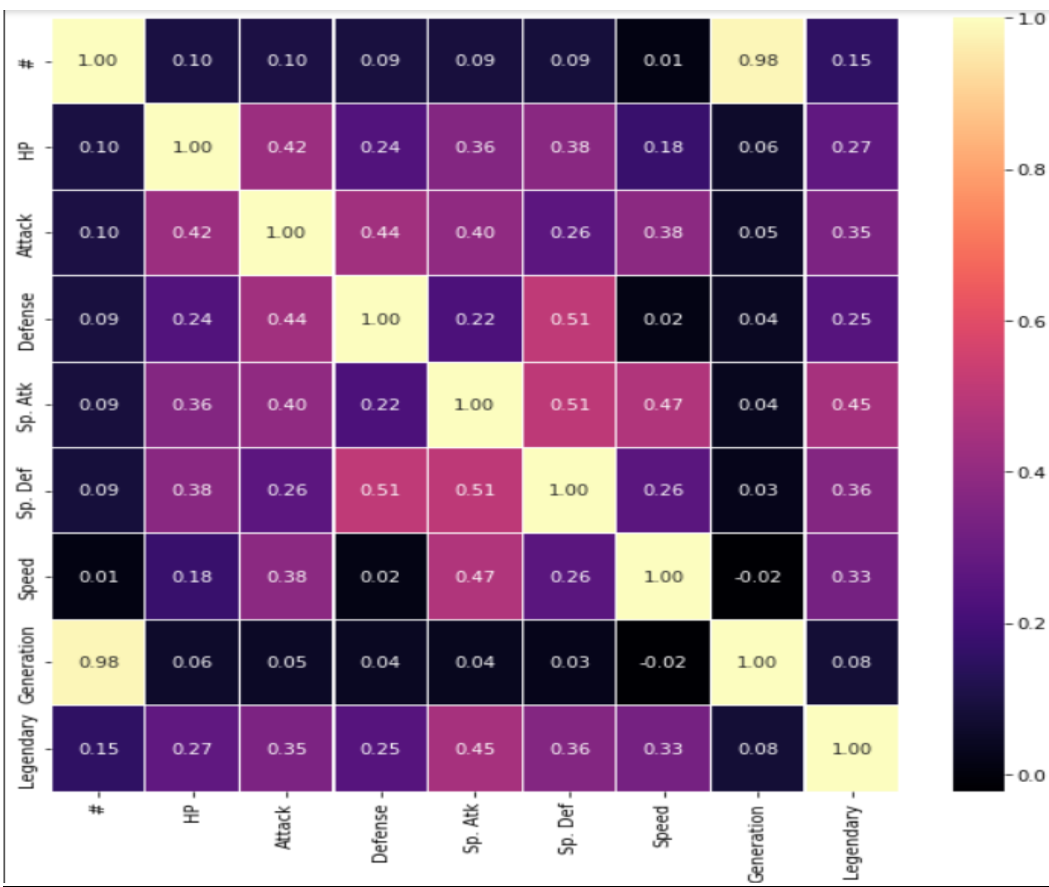


Figure 4: Correlations between any two numerical values

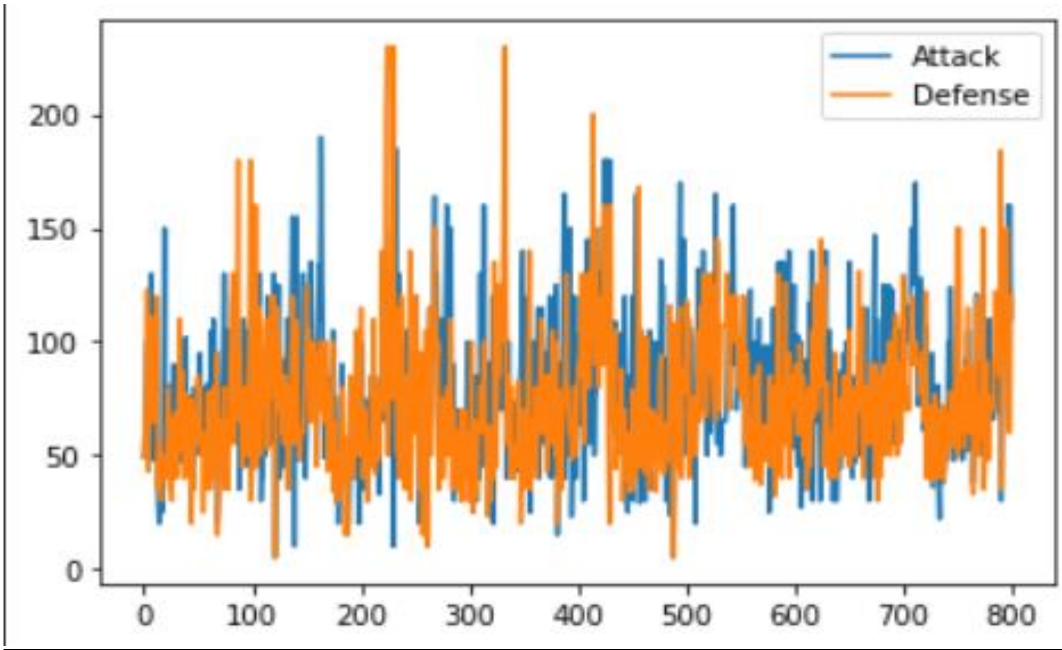


Figure 5: Similarity between Attack and Defense

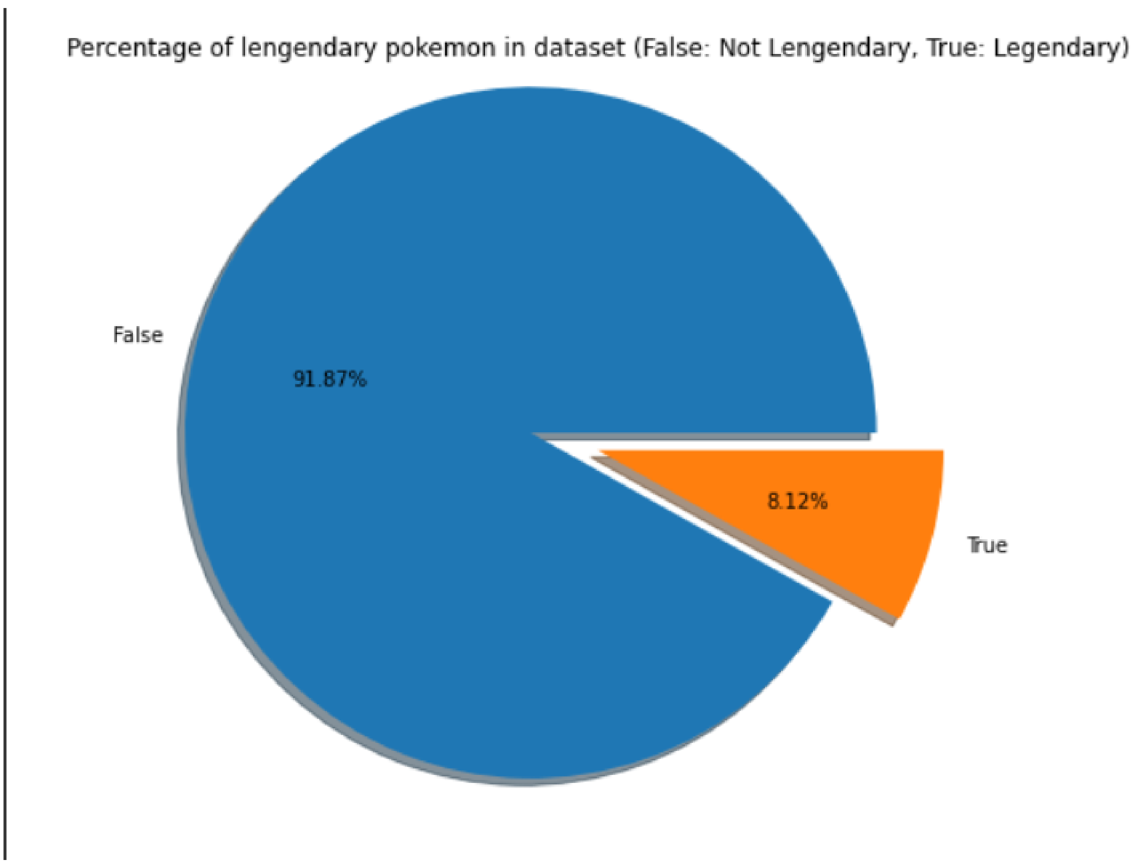


Figure 6: Percentage of Legendary Pokemon

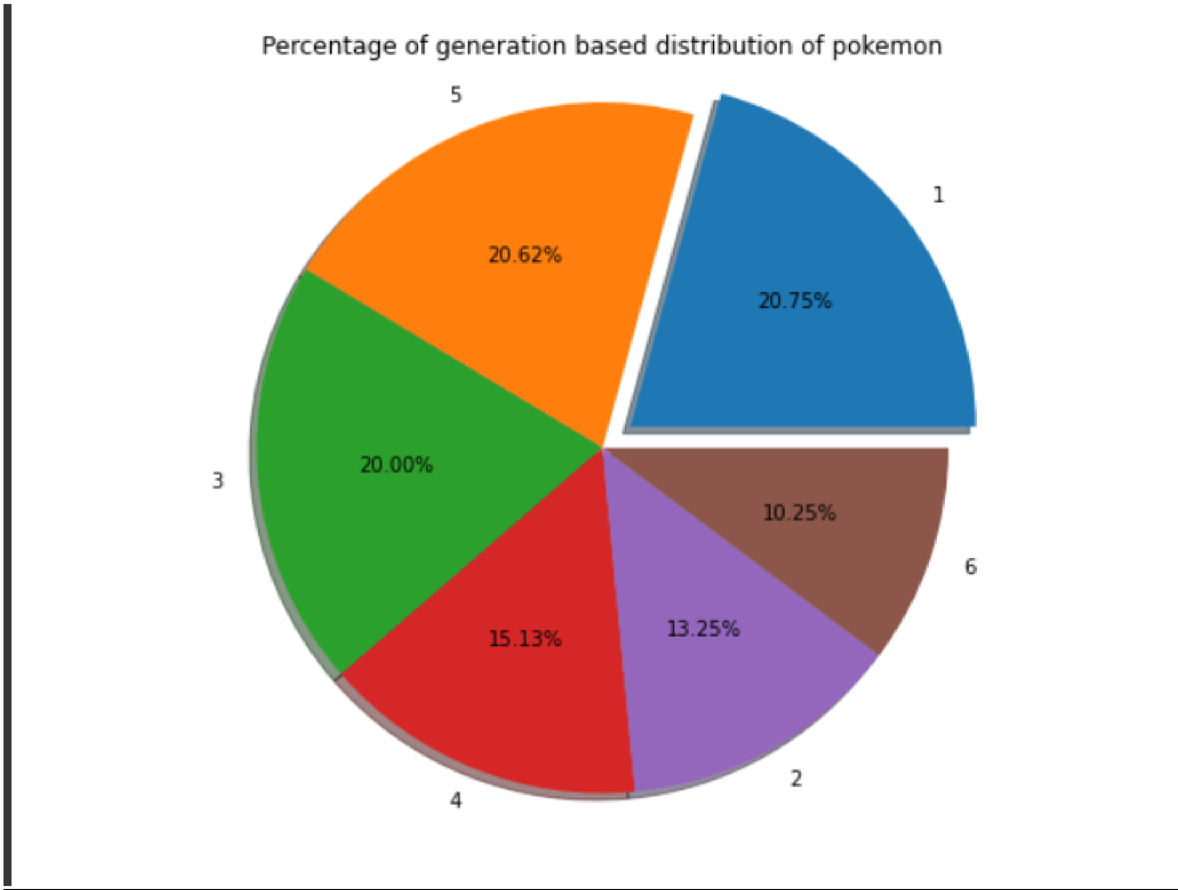


Figure 7: Percentage of Generations

Project Summary

Project Title:= Pokemon Battle

INTRODUCTION :=

Pokemon is a popular video game franchise where players play as a trainer who owns monsters called Pokemon. Players can battle other trainers by having their Pokemon fight in a turn-based combat system. The game of Pokemon has evolved in major ways over the years, with each new iteration of the game making the game more and more complex. The first Pokemon game featured 151 unique Pokemon, but now there exists over 800 of them. Pokemon battles contain an unique blend of strategy, domain knowledge, and luck that make them well-regarded amongst the video game community. In addition, due to the extremely large amount of both Pokemon and moves, there exists an incredible amount of variety to the battles. I was interested in exploring this battle space by using Machine learning to create an agent that can analyze Pokemon's properties and predict which Pokemon can win that battle.

Dataset and its Variables: =

My data source comes from Kaggle named "**Pokemon- Weedle's Cave**" that provides entries of Pokemon and their combats. The original dataset includes 13 variables including their # (for number), name, type 1, type 2, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, Generation, Legendary, First_pokemon, Second_pokemon, Winner that mainly defines their ability to fight.

The Data is: =

- # (for numbers): ID for each Pokemon
- Name: Name of each Pokemon
- Type 1: Each Pokemon has a type, this determines weakness/resistance to attacks
- Type 2: Some Pokemon are dual type and have 2
- HP: hit points, or health, defines how much damage a pokemon can withstand before fainting
- Attack: Base modifier for normal attacks (eg. Scratch, Punch)
- Defense: base damage resistance against normal attacks
- Sp. Def: base damage resistance against special attacks
- Sp. Atk: special attack, base modifier for special attacks (eg. Fire blast, bubble beam)
- Speed: determines which pokemon attacks first each round
- Generation: number of generations
- Legendary: true if legendary pokemon, false if not (more revision on mythical vs legendary needed)

Motivation to this Project

The battling aspect of Pokemon is so popular that there is a relatively large competitive scene. Countless databases, forums, and other online resources exist to give players the information needed to increase battling skill. Moreover, there are sanctioned competitive battle tournaments in real life where the winners receive cash prizes. There also exists a popular battle simulator website called Pokemon Showdown, where play can create teams of Pokemon and battle others on the internet. Given the widespread popularity of Pokemon battling, I wished to further explore the competitive scene by developing a successful agent. Additionally, I was interested in the project applications in general game-playing. Attempts to find optimal strategies for various games such as chess or Go are common throughout the literature. Yet, given that Pokemon is a video game and that it has an immense state space, comparatively less research has been put into creating an optimal battle agent. Therefore, I was interested in seeing if it was indeed possible to create an AI agent for Pokemon that was able to match or even exceed human performance levels. By using game-playing algorithms explored in other games, I hoped to find the existing strategy that would have the best performance when applied to Pokemon battling.

Apparatus

- Laptop
- Datasets
- Google Colab
- Libraries like matplotlib, seaborn, pandas, numpy, etc.
- Microsoft Excel
- Microsoft Word
- Snipping Tool
- Internet Connection

Details of Project

Below is the Pokemon dataset I found and used to build the machine learning model.
https://www.kaggle.com/terminus7/pokemon-challenge

I have taken the following steps to build a machine learning model.

1. Data Exploration
2. Data Preprocessing
3. Model Selection and Training
4. Prediction

Data Exploration

I have Seen the first five entries of Pokemon and combats dataset. The Pokemon dataset is made up of different pokemon with their abilities. Combat dataset is made up of battles between two pokemon. The "#" number is used to map between the pokemon dataset and the combat dataset.

Data Preprocessing

1. Handle missing data: There might be several spots in a dataset where the values are missing. We can't leave those spots empty. Also, I don't want to remove those samples because that reduces my dataset.
Numerical values can be filled with mean or medium or maximum occurring value in that column.

Column "Type 2" of Pokemon dataset contains empty spots. It's a categorical column, therefore, I can fill the missing value with the most common value in that column.
But I choose to create another category called NA (Not Applicable). It's like any other category of " Type 2 " column.

2. Categorical value to numerical value: A machine learning model works on numbers. We can't feed it strings or words, therefore we have to convert every categorical value into a numerical value.

There are several ways to do it, like label encoder, one hot encoder, feature hasher.

I used FeatureHasher to convert columns "Type 1" and "Type 2" into numerical values. FeatureHasher also solves the problem of large number of columns created (if the number of categories is very large) due to one-hot encoding.

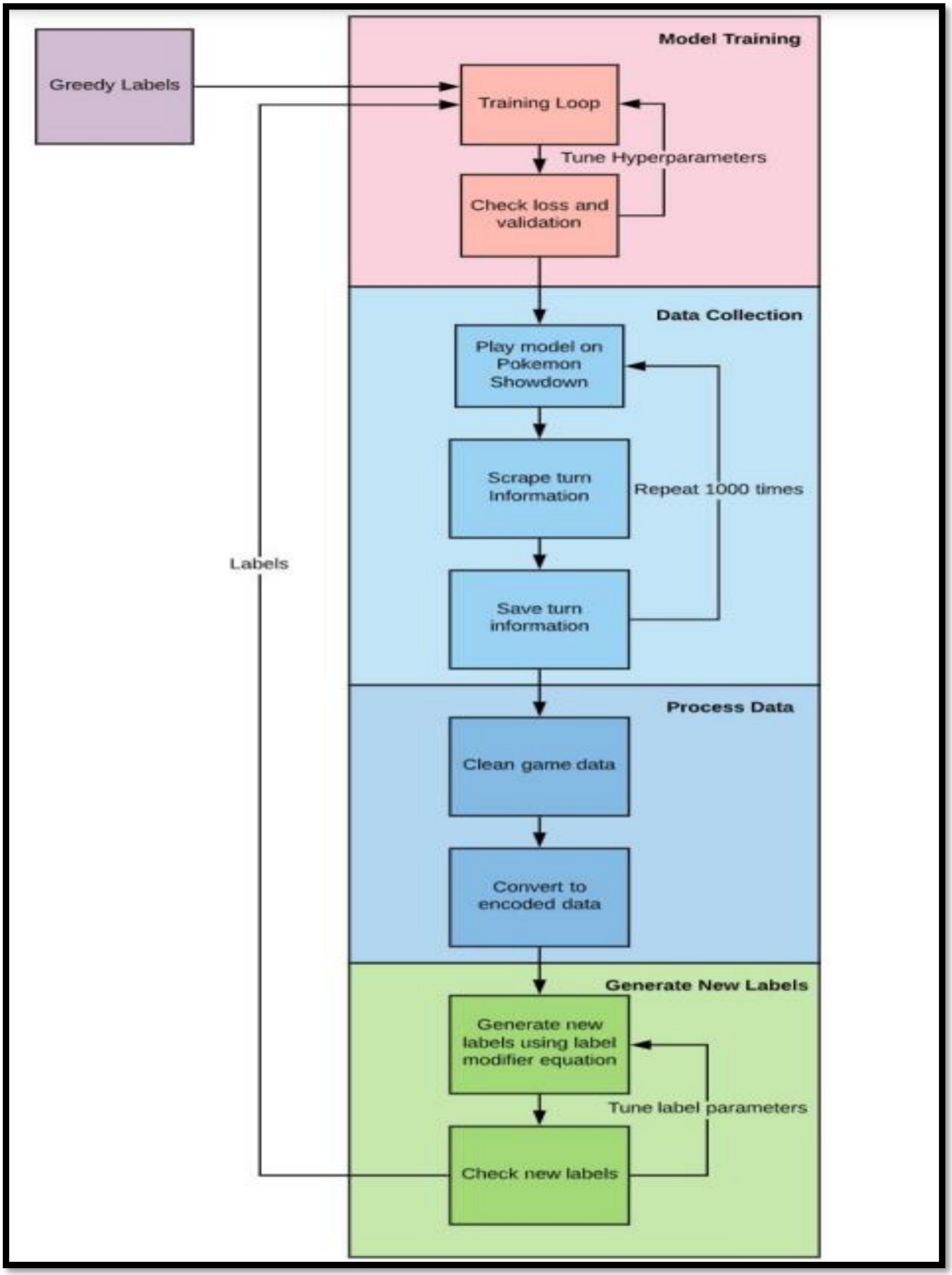
I mapped the data of the combat dataset with the Pokemon dataset and created a new training dataset. For example:
Row[0] of combat is:
first_pokemon = 266, Second_pokemon=298, Winner = 298.

3.Split data into train and test: Split the dataset into train and test dataset. We will use some data to train the model and the remaining data to test the model.

Model Selection, Training, and Prediction

I picked Random Forest Classification algorithm to build my model and got *94.76% accuracy.*

Flow Data Representation



Code in text

```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction import FeatureHasher
from sklearn.metrics import classification_report


pokemon = pd.read_csv("pokemon.csv")  # Load Pokemon Dataset
combats = pd.read_csv("combats.csv")  # Load Combats Dataset


pokemon.head()


combats.head()


print(pokemon.nunique())


# Plot the number of pokemon present in each category of "type 1"
ax = pokemon['Type 1'].value_counts().plot(kind='bar',color="yellow",edgecolor="red", figsize=(14,8), title="N
umber of pokemons based on their type 1")
ax.set_xlabel("Pokemon Type 1")
ax.set_ylabel("Frequency")


ax = pokemon['Type 2'].value_counts().plot(kind='bar',color="thistle",edgecolor="indigo", figsize=(14,8), title=
"Number of pokemons based on their type 2")
ax.set_xlabel("Pokemon Type 2")
ax.set_ylabel("Frequency")


number = pokemon["#"]
print('Total number of Pokemons is', len(number))


Legendary = pokemon["Legendary"]
```

```
rate = np.mean(Legendary == True)
print('legendary rate=', rate)


fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(pokemon.corr(), ax=ax, annot=True, linewidths=0.05, fmt='.2f', cmap="magma")
plt.show()


df2 = pokemon.loc[:, ["Attack", "Defense"]]
df2.plot()


generation =  dict(pokemon['Generation'].value_counts())
gen_counts = generation.values() # No of pokemon in each generation
gen = generation.keys()  # Type of generation
fig = plt.figure(figsize=(8, 6))
fig.suptitle("Percentage of generation based distribution of pokemon")
ax = fig.add_axes([0,0,1,1])
explode = (0.1, 0, 0, 0, 0, 0)  # explode 1st slice
ax.axis('equal')
plt.pie(gen_counts, labels = gen,autopct='%1.2f%%', shadow=True, explode=explode)
plt.show()


generation =  dict(pokemon['Legendary'].value_counts())
gen_counts = generation.values()
gen = generation.keys()
fig = plt.figure(figsize=(8, 6))
fig.suptitle("Percentage of lengendary pokemon in dataset (False: Not Lengendary, True: Legendary)")
ax = fig.add_axes([0,0,1,1])
explode = (0.2, 0)  # explode 1st slice
ax.axis('equal')
plt.pie(gen_counts, labels = gen,autopct='%1.2f%%', shadow=True, explode=explode)
plt.show()
```

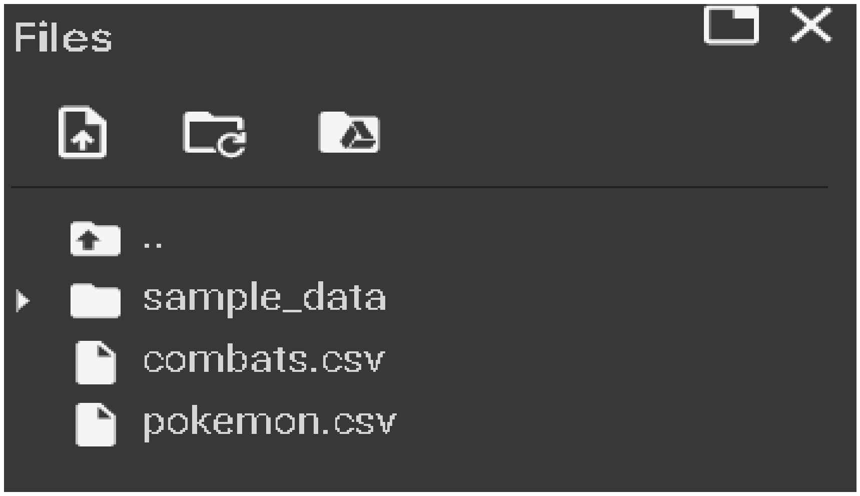
Data Preprocessing

```
pokemon["Type 2"] = pokemon["Type 2"].fillna("NA")


# Convert "Legendary" column, False is converted to 0 and True is converted to 1.
```

```
pokemon["Legendary"] = pokemon["Legendary"].astype(int)


h1 = FeatureHasher(n_features=5, input_type='string')
h2 = FeatureHasher(n_features=5, input_type='string')
d1 = h1.fit_transform(pokemon["Type 1"])
d2 = h2.fit_transform(pokemon["Type 2"])


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


pokemon


x = pokemon.loc[pokemon["#"]==266].values[:, 2:][0]
print(x)
y = pokemon.loc[pokemon["#"]==298].values[:, 2:][0]
print(y)
z = np.concatenate((x,y))
z


data = []
i = 0
for t in combats.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)


X = data[:, :-1].astype(int)
y = data[:, -1].astype(int)


train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.25, random_state=42)


clf = RandomForestClassifier(n_estimators=100)
model = clf.fit(train_x, train_y)
pred = model.predict(test_x)
#print('Accuracy of {}:'.format(name), accuracy_score(pred, test_y))


print('Accuracy :', accuracy_score(pred, test_y))
print(classification_report(test_y, pred))
```

Code with Output Screenshots




```python
# Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction import FeatureHasher
from sklearn.metrics import classification_report
```

```python
[57] pokemon = pd.read_csv("pokemon.csv")  # Load Pokemon Dataset
     combats = pd.read_csv("combats.csv")  # Load Combats Dataset
```

```python
[58] 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

```
[59] combats.head()
```

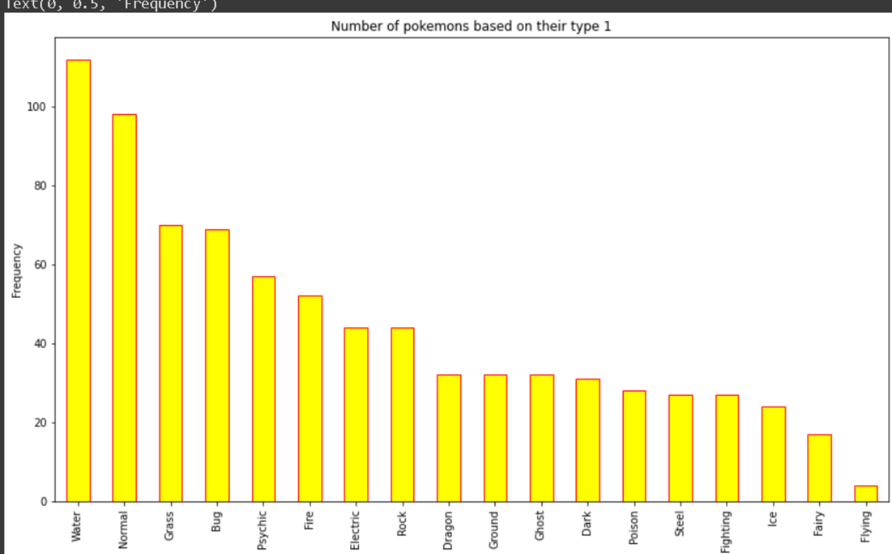
	First_pokemon	Second_pokemon	Winner
0	266	298	298
1	702	701	701
2	191	668	668
3	237	683	683
4	151	231	151

```
print(pokemon.nunique())
```

```
#             800
Name          799
Type 1         18
Type 2         18
HP             94
Attack        111
Defense       103
Sp. Atk       105
Sp. Def        92
Speed         108
Generation      6
Legendary       2
dtype: int64
```

```
# Plot the number of pokemon present in each category of "type 1"
ax = pokemon['Type 1'].value_counts().plot(kind='bar',color="yellow",edgecolor="red", figsize=(14,8), title="Number of pokemons based on their type 1")
ax.set_xlabel("Pokemon Type 1")
ax.set_ylabel("Frequency")
```

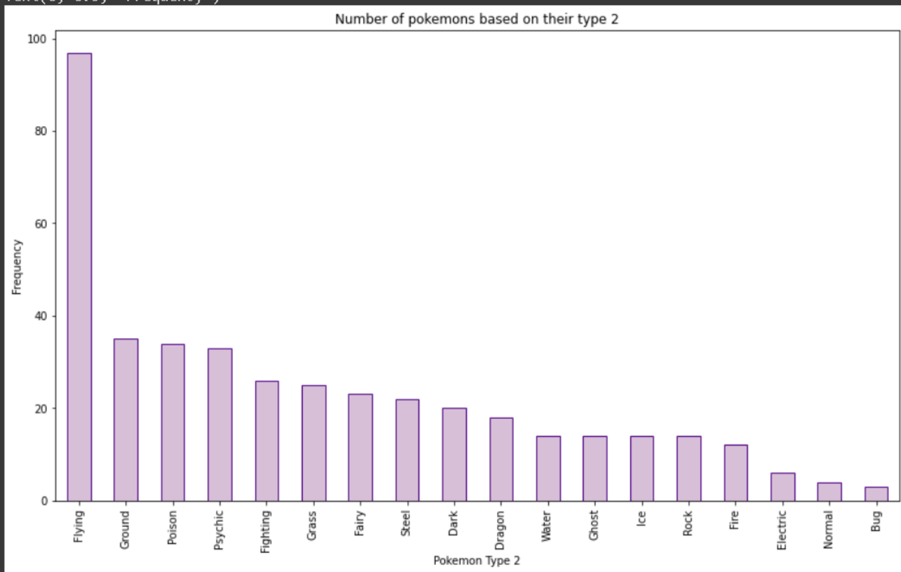
```
Text(0, 0.5, 'Frequency')
```



Executing (1m 59s) Cell > __getitem__() > _take_with_is_copy() > take() > take() > reindex_indexer() > _slice_take_blocks_ax0() > take_nd() > _take_nd_ndarray()

```
ax = pokemon['Type 2'].value_counts().plot(kind='bar',color="thistle",edgecolor="indigo", figsize=(14,8), title="Number of pokemons based on their type 2")
ax.set_xlabel("Pokemon Type 2")
ax.set_ylabel("Frequency")
```

```
Text(0, 0.5, 'Frequency')
```



0s completed at 8:30 PM

```
[63] number = pokemon["#"]
     print('Total number of Pokemons is', len(number))
```

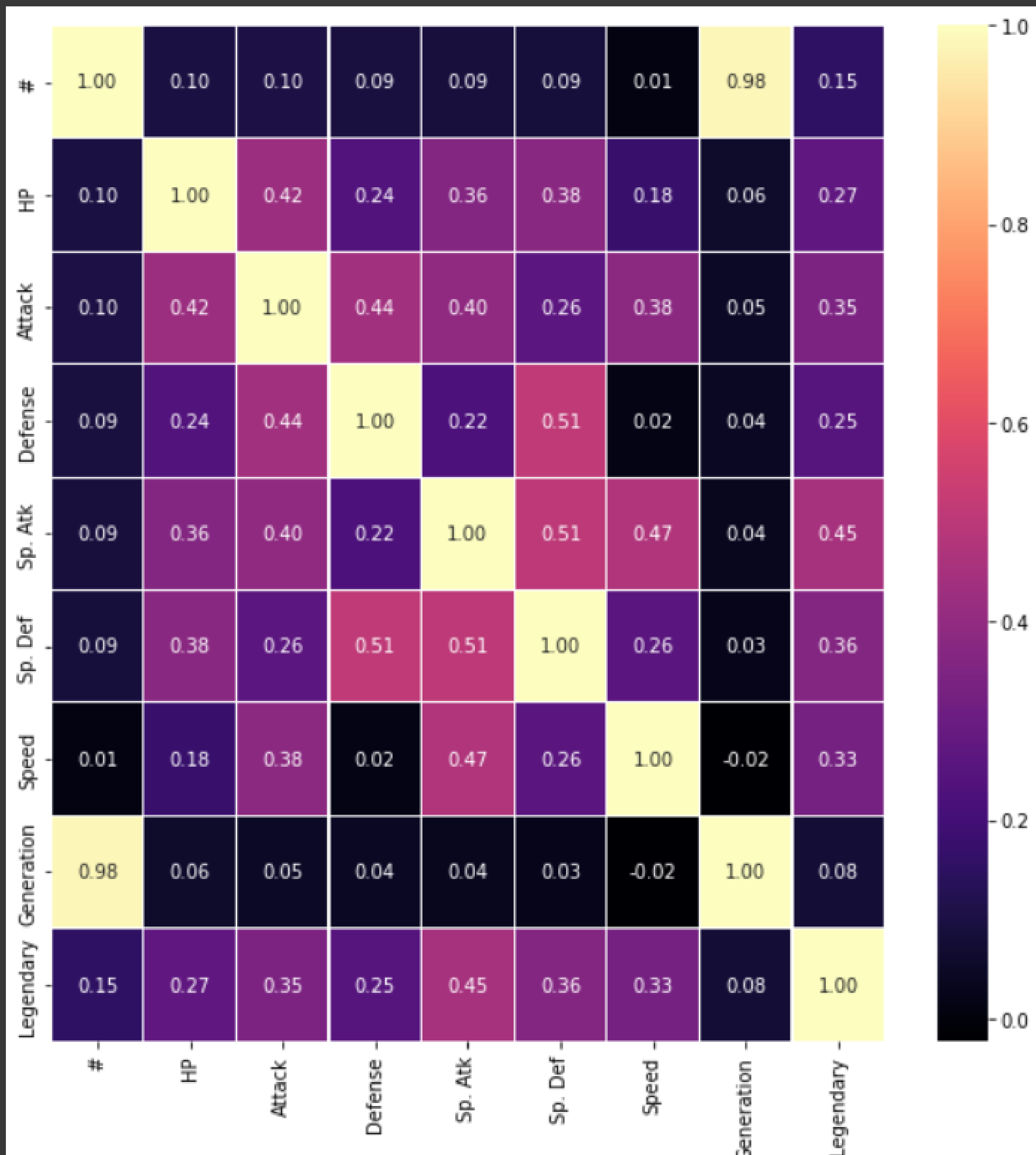
```
Total number of Pokemons is 800
```

Loading...

```
Legendary = pokemon["Legendary"]
rate = np.mean(Legendary == True)
print('legendary rate=', rate)
```

```
legendary rate= 0.08125
```

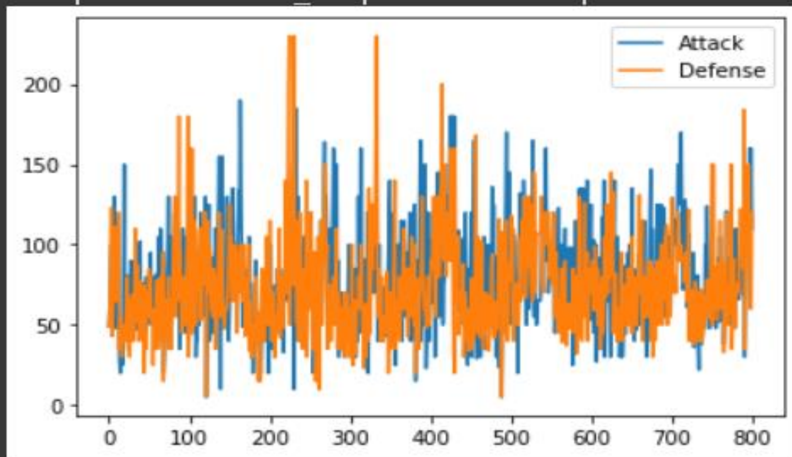
```
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(pokemon.corr(), ax=ax, annot=True, linewidths=0.05, fmt='.2f', cmap="magma")
plt.show()
```



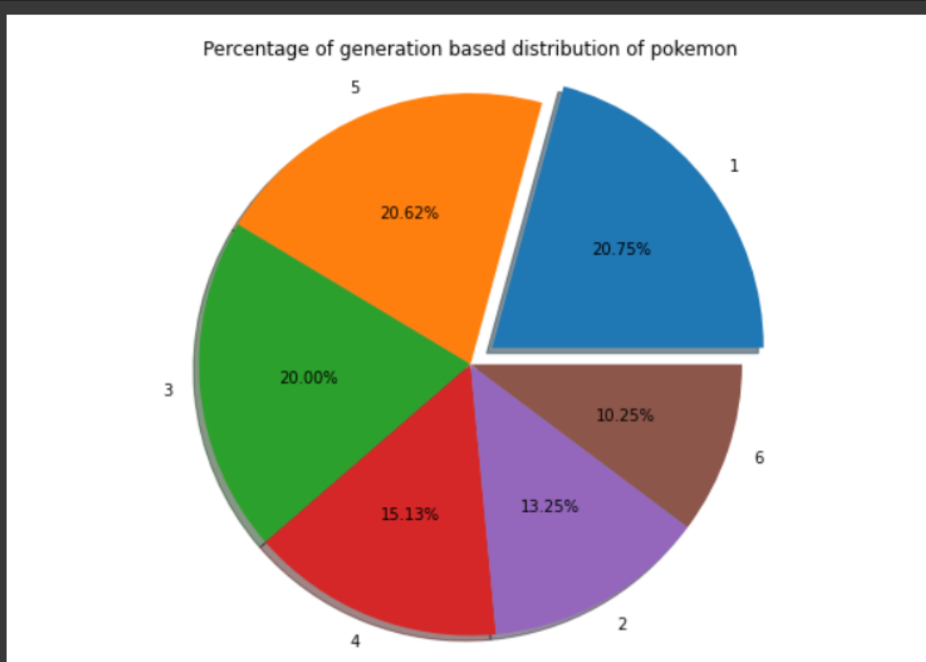
✓ 0s completed at 8:30

```python
df2 = pokemon.loc[:, ["Attack", "Defense"]]
df2.plot()
```

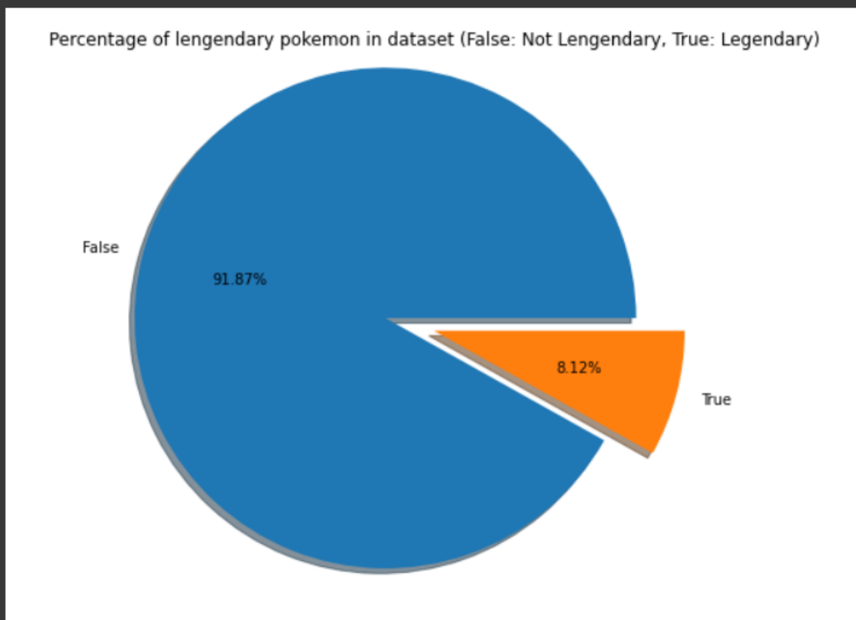
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffaf31c5d50>
```




```python
generation =  dict(pokemon['Generation'].value_counts())
gen_counts = generation.values() # No of pokemon in each generation
gen = generation.keys()  # Type of generation

fig = plt.figure(figsize=(8, 6))
fig.suptitle("Percentage of generation based distribution of pokemon")
ax = fig.add_axes([0,0,1,1])
explode = (0.1, 0, 0, 0, 0, 0)  # explode 1st slice
ax.axis('equal')


plt.pie(gen_counts, labels = gen,autopct='%1.2f%%', shadow=True, explode=explode)
plt.show()
```



```
generation =  dict(pokemon['Legendary'].value_counts())
gen_counts = generation.values()
gen = generation.keys()

fig = plt.figure(figsize=(8, 6))
fig.suptitle("Percentage of lengendary pokemon in dataset (False: Not Lengendary, True: Legendary)")
ax = fig.add_axes([0,0,1,1])
explode = (0.2, 0)  # explode 1st slice
ax.axis('equal')


plt.pie(gen_counts, labels = gen,autopct='%1.2f%%', shadow=True, explode=explode)
plt.show()
```



Data Preprocessing

```
[69] pokemon["Type 2"] = pokemon["Type 2"].fillna("NA")
```

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

```
h1 = FeatureHasher(n_features=5, input_type='string')
h2 = FeatureHasher(n_features=5, input_type='string')
d1 = h1.fit_transform(pokemon["Type 1"])
d2 = h2.fit_transform(pokemon["Type 2"])
```

```
[72] # Convert to dataframe
     d1 = pd.DataFrame(data=d1.toarray())
     d2 = pd.DataFrame(data=d2.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)
```

```
[73] pokemon
```

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

800 rows × 20 columns

```
x = pokemon.loc[pokemon["#"]==266].values[:, 2:][0]
print(x)
y = pokemon.loc[pokemon["#"]==298].values[:, 2:][0]
print(y)
z = np.concatenate((x,y))
z
```

```
[50 64 50 45 50 41 2 0 0.0 -1.0 -1.0 1.0 1.0 1.0 0.0 1.0 1.0 1.0]
[70 70 40 60 40 60 3 0 2.0 0.0 0.0 0.0 -1.0 2.0 0.0 0.0 0.0 0.0]
array([50, 64, 50, 45, 50, 41, 2, 0, 0.0, -1.0, -1.0, 1.0, 1.0, 1.0, 0.0,
       1.0, 1.0, 1.0, 70, 70, 40, 60, 40, 60, 3, 0, 2.0, 0.0, 0.0, 0.0,
       -1.0, 2.0, 0.0, 0.0, 0.0, 0.0], dtype=object)
```

```python
data = []
i = 0
for t in combats.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)
```

```
49929
49930
49931
49932
49933
49934
49935
49936
49937
49938
49939
49940
49941
49942
49943
49944
```

```
[76] data = np.asarray(data)
```

```
X = data[:, :-1].astype(int)
y = data[:, -1].astype(int)
```

```
[78] train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
[79] clf = RandomForestClassifier(n_estimators=100)
     model = clf.fit(train_x, train_y)
     pred = model.predict(test_x)
     #print('Accuracy of {}:'.format(name), accuracy_score(pred, test_y))
```

```
[80] print('Accuracy :', accuracy_score(pred, test_y))
     print(classification_report(test_y, pred))
```

```
Accuracy : 0.94808
              precision    recall  f1-score   support

           0       0.95      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
```

Summary

In this work, I have analyzed how numerical variables are distributed. In the case of categorical variables, I have used histograms with same end. Then I tried to find correlations between the numerical variables. No big dependencies have been found with the generation a Pokemon was released or its probability of being female and male. And then I have built a predictive model that I tried to predict which Pokemon will win the battle base on its numerical variables. In the process of modelling I used different kinds of methods to improve the accuracy.

References

1. Alberto Barradas. Pokemon with stats. https://www.kaggle.com/terminus7/pokemon-challenge
2. Asier LA, spez Zorilla. PokAl mon for Data Mining and Machine Learning. https://www.kaggle.com/alopez247/pokemon
3. David W Scott. Multivariate density estimation: theory, practice, and visualization.2015
4. S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality, Biometrika.