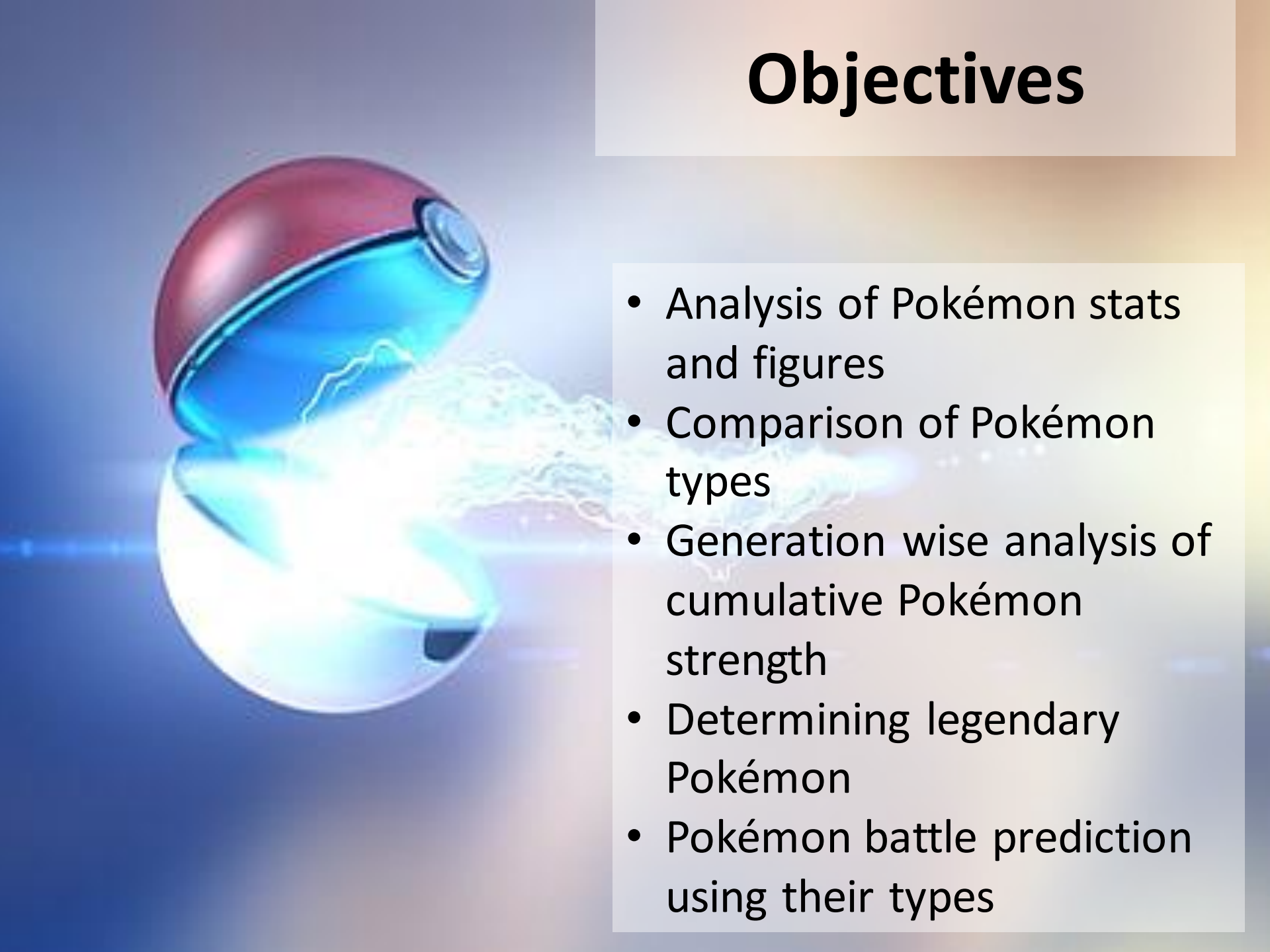
POKEMON BATTLE PREDICTION SYSTEM

Problem Statement

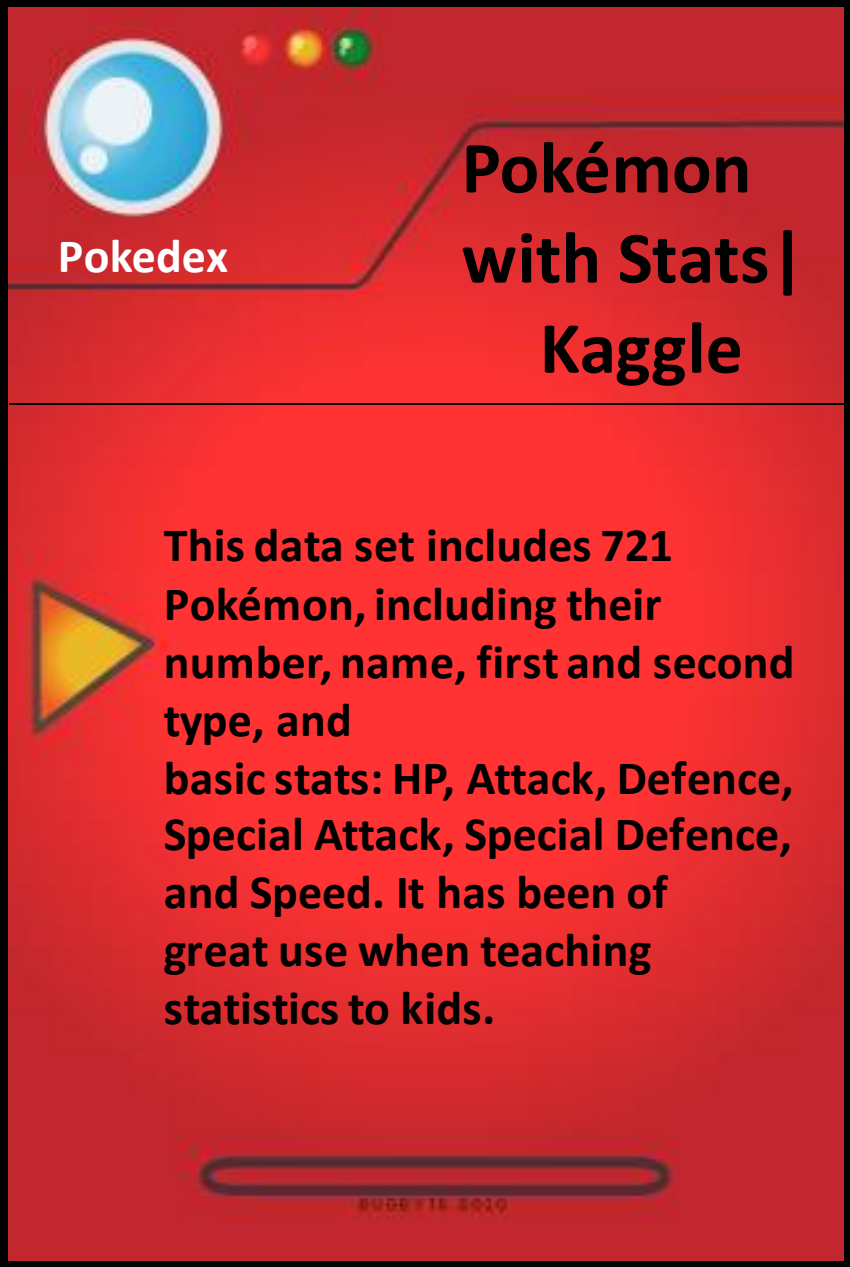
- The problem statement of this project is to develop a Pokémon battle prediction system.
- It should be able to determine the Pokémon with higher chances of winning in a Pokémon battle.
- For this, relevant data of Pokémon generation, types and statistics is obtained.
- The system will include the following primary modules/components: statistics display, generation display and strength comparison, type comparison, determining whether legendary or not and most importantly, type-based Pokémon battle prediction.



Objectives

- Analysis of Pokémon stats and figures
- Comparison of Pokémon types
- Generation wise analysis of cumulative Pokémon strength
- Determining legendary Pokémon
- Pokémon battle prediction using their types

About The Dataset



References:
https://www.kaggle.com/abcsds/pokemon

Data Preprocessing

- Missing values handling
- Plot Graphs
- Tuple Duplicate
- Correlation matrix
- Min max normalization
- Z score
- Histogram
- Categorical values

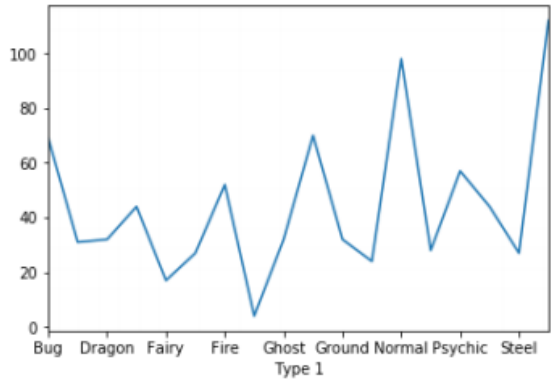
Data Preprocessing

Missing values handling

```
In [112]:
pokemon.fillna(0,inplace=True)

In [113]:
pokemon.dropna(inplace=True)

In [114]:
pokemon.isnull().sum()

Out[114]:
Type 1         0
Type 2         0
Total          0
HP             0
Attack         0
Defense        0
Sp. Atk        0
Sp. Def        0
Speed          0
Generation     0
Legendary      0
dtype: int64
```

Plot graphs

```
In [120]:
graph1.plot()

Out[120]:
<matplotlib.axes._subplots.AxesSubplot at 0xe4
```



Tuple duplication

```
In [144]:
print("Before removing duplicates:")
print(pokemon.shape)
print(pokemon.size)
print("After removing duplicate tuples:")
print(pokemon_unique.shape)
print(pokemon_unique.size)

Before removing duplicates:
(800, 11)
8800
After removing duplicate tuples:
(794, 11)
8734
```

Correlation matrix

```
pokemon.corr(method='pearson')

Out[146]:
```

	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
Total	1.000000	0.618748	0.736211	0.612787	0.747250	0.717609	0.575943	0.048384	0.501758
HP	0.618748	1.000000	0.422386	0.239622	0.362380	0.378718	0.175952	0.058683	0.273620
Attack	0.736211	0.422386	1.000000	0.438687	0.396362	0.263990	0.381240	0.051451	0.345408
Defense	0.612787	0.239622	0.438687	1.000000	0.223549	0.510747	0.015227	0.042419	0.246377
Sp. Atk	0.747250	0.362380	0.396362	0.223549	1.000000	0.506121	0.473018	0.036437	0.448907
Sp. Def	0.717609	0.378718	0.263990	0.510747	0.506121	1.000000	0.259133	0.028486	0.363937
Speed	0.575943	0.175952	0.381240	0.015227	0.473018	0.259133	1.000000	-0.023121	0.326715
Generation	0.048384	0.058683	0.051451	0.042419	0.036437	0.028486	-0.023121	1.000000	0.079794
Legendary	0.501758	0.273620	0.345408	0.246377	0.448907	0.363937	0.326715	0.079794	1.000000

Data Preprocessing

Min-max normalization

```
In [158]:
from sklearn import preprocessing

In [159]:
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(pokemon[['Total']].values.astype(float))
pokemon_normalized = pd.DataFrame(x_scaled)
pokemon['normalized2']=x_scaled
pokemon.sort_values(by='Total')

Out[159]:
```

	Type 1	Type 2	Total	HP	Attack	Defense	Sp.	Sp.	Speed	Generation	Legendary	norma

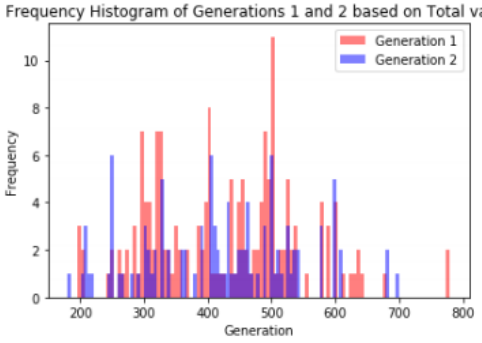
Z-score

```
In [160]:
from scipy import stats
pokemon['zscore']=stats.zscore(pokemon['Total'])
pokemon.sort_values(by='Total')

Out[160]:
```

Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	no
Sunkern	Grass		0	180	30	30	30	30	30	2	False	
Azurill	Normal	Fairy	190	50	20	40	20	40	20	3	False	

Histogram

```
plt.hist(a, alpha=0.5, bins=100, color='r', label='Generation 1')
plt.hist(b, alpha=0.5, bins=100, color='b', label='Generation 2')
plt.gca().set(title='Frequency Histogram of Generations 1 and 2 based on Total value', yl
abel='Frequency', xlabel='Generation')
plt.legend();
```



Categorical values

```
df_categorical = pokemon.select_dtypes(exclude=[np.
In [202]:
df_categorical
Out[202]:
```

Name	Type 1	Type 2	Legendary
Bulbasaur	Grass	Poison	False
Ivysaur	Grass	Poison	False

System Architecture

1. Python
2. Knime
3. Libraries used:

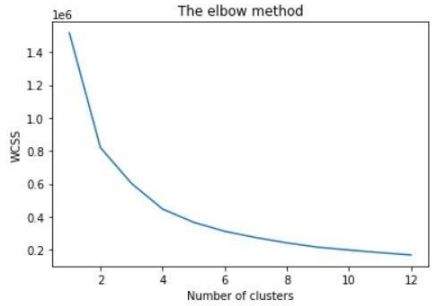
- pandas as pd
- numpy as np
- scikit learn as sklearn
- seaborn as sns
- matplotlib.pyplot as plt



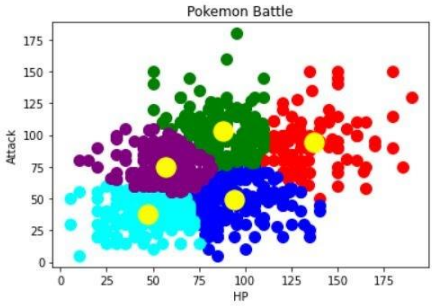
Data Mining tasks performed

K Means Clustering using Elbow Method

In [8]:
```
wcss = []

for i in range(1, 13):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init =10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 13), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```



In [12]:
```
plt.scatter(x[y_kmeans==0, 0], x[y_kmeans==0, 1], s=100, c='red', label ='Cluster 1')
plt.scatter(x[y_kmeans==1, 0], x[y_kmeans==1, 1], s=100, c='blue', label ='Cluster 2')
plt.scatter(x[y_kmeans==2, 0], x[y_kmeans==2, 1], s=100, c='green', label ='Cluster 3')
plt.scatter(x[y_kmeans==3, 0], x[y_kmeans==3, 1], s=100, c='cyan', label ='Cluster 4')
plt.scatter(x[y_kmeans==4, 0], x[y_kmeans==4, 1], s=100, c='purple', label ='Cluster 5')
#visualisation of clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='yellow', label = 'Centroids')
plt.title('Pokemon Battle')
plt.xlabel('HP')
plt.ylabel('Attack')
plt.show()
#Plotting the centroids
```



Pokémon Battle Prediction

Calculations - Based on Statistics and Type Advantage :

In [22]:
```
def replace_things(data):
    #map each battles to pokemon data

    data['First_pokemon_stats'] = data.First_pokemon.map(stats_dict)
    data['Second_pokemon_stats'] = data.Second_pokemon.map(stats_dict)

    data['First_pokemon'] = data.First_pokemon.map(type_dict)
    data['Second_pokemon'] = data.Second_pokemon.map(type_dict)

    return data
```

In [23]:
```
def calculate_stats(data):
    #calculate stats difference

    stats_col = ['HP_diff', 'Attack_diff', 'Defense_diff', 'Sp.Atk_diff', 'Sp.Def_diff', 'Speed_diff', 'Legendary_diff']
    diff_list = []

    for row in data.itertuples():
        diff_list.append(np.array(row.First_pokemon_stats) - np.array(row.Second_pokemon_stats))

    stats_df = pd.DataFrame(diff_list, columns=stats_col)
    data = pd.concat([data, stats_df], axis=1)
    data.drop(['First_pokemon_stats', 'Second_pokemon_stats'], axis=1, inplace=True)

    return data
```

Pokémon Battle Prediction

Calculations - Based on Statistics and Type Advantage :

In [24]:
```
def calculate_effectiveness(data):

    '''
        this function creates a new column of each pokemon's effectiveness against it's enemy.
        every effectiveness starts with 1, if an effective type is found on enemy's type, effectiveness * 2
        if not very effective is found on enemy's type, effectiveness / 2
        if not effective is found on enemy's type, effectiveness * 0

        This function creates 4 new columns
            1. P1_type1, pokemon 1 first type effectiveness against the enemy's type
            2. P1_type2, pokemon 1 second type effectiveness against the enemy's type
            3. P2_type1, pokemon 2 first type effectiveness against the enemy's type
            4. P2_type2, pokemon 2 second type effectiveness against the enemy's type
    '''

    very_effective_dict = {'Normal': [],
                           'Fight': ['Normal', 'Rock', 'Steel', 'Ice', 'Dark'],
                           'Flying': ['Fight', 'Bug', 'Grass'],
                           'Poison': ['Grass', 'Fairy'],
                           'Ground': ['Poison', 'Rock', 'Steel', 'Fire', 'Electric'],
                           'Rock': ['Flying', 'Bug', 'Fire', 'Ice'],
                           'Bug': ['Grass', 'Psychic', 'Dark'],
                           'Ghost': ['Ghost', 'Psychic'],
                           'Steel': ['Rock', 'Ice', 'Fairy'],
                           'Fire': ['Bug', 'Steel', 'Grass', 'Ice'],
                           'Water': ['Ground', 'Rock', 'Fire'],
                           'Grass': ['Ground', 'Rock', 'Water'],
                           'Electric': ['Flying', 'Water'],
                           'Psychic': ['Fight', 'Poison'],
                           'Ice': ['Flying', 'Ground', 'Grass', 'Dragon'],
                           'Dragon': ['Dragon'],
                           'Dark': ['Ghost', 'Psychic'],
                           'Fairy': ['Fight', 'Dragon', 'Dark'],
                           'None': []}
```

GAME OVER

Charizard

Results

Based on all statistics

```
In [122]: p1 = input('Enter name of 1st Poekmon')
          p2 = input('Enter name of 2nd Poekmon')
          def search(str):
              count=1
              names = pokemon_df["Name"].tolist()
              for name in names:
                  if name == str:
                      x = count
                  count+=1
              return(x)
          y = search(p1)
          x = search(p2)
          print(y,x)
          #x = 10
          #y = 31

          Enter name of 1st PoekmonCharizard
          Enter name of 2nd PoekmonSquirtle
          7 10

In [123]: names = pokemon_df["Name"].tolist()

          name2 = names[x-1]

          pokemon_df[pokemon_df.Name == name2]
```

Out[123]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total_stats
9	10	Squirtle	Water	None	44	48	65	50	64	43	1	0	314

Out[127]:

	HP_diff	Attack_diff	Defense_diff	Sp.Atk_diff	Sp.Def_diff	Speed_diff	Legendary_diff	P1_type1	P1_type2	P2_type1	P2_type2
0	34	36	13	59	21	57	0	0.5	1	2	1

```
In [128]: classifier.predict(test1)

Out[128]: array([0], dtype=int64)

In [129]: result = classifier.predict(test1)
          if result[0] == 1 :
              print("The stronger pokemon is " + name2 )
          else :
              print("The stronger pokemon is " +name1)

          The stronger pokemon is Charizard

In [ ]:
```

Squirtle

GAME OVER

Electabuzz

Results

Based on types

Seaking

```
Enter name of 1st Pokemon :- Electabuzz
Enter name of 2nd Pokemon :- Seaking
135 129
```

In [140]:
```
names = pokemon_df["Name"].tolist()

name2 = names[x-1]

pokemon_df[pokemon_df.Name == name2]
```

Out[140]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total_stats
128	129	Seaking	Water	None	80	92	65	65	80	68	1	0	450

In [141]:
```
name1 = names[y-1]
pokemon_df[pokemon_df.Name == name1]
```

Out[141]:

	#	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary	Total_stats
134	135	Electabuzz	Electric	None	65	83	57	95	85	105	1	0	490

In [142]:
```
test1 = pd.DataFrame({"First_pokemon" : [y] ,"Second_pokemon" : [x]})
```

In [143]:
```
test1
```

Out[143]:

	First_pokemon	Second_pokemon
0	135	129

Out[143]:

	First_pokemon	Second_pokemon
0	135	129

In [144]:
```
test1 = replace_things(test1)
test1 = calculate_stats(test1)
test1 = calculate_effectiveness(test1)
test1.head()
```

Out[144]:

	HP_diff	Attack_diff	Defense_diff	Sp.Atk_diff	Sp.Def_diff	Speed_diff	Legendary_diff	P1_type1	P1_type2	P2_type1	P2_type2
0	-15	-9	-8	30	5	37	0	2	1	1	1

In [145]:
```
classifier.predict(test1)
```

Out[145]:
```
array([0], dtype=int64)
```

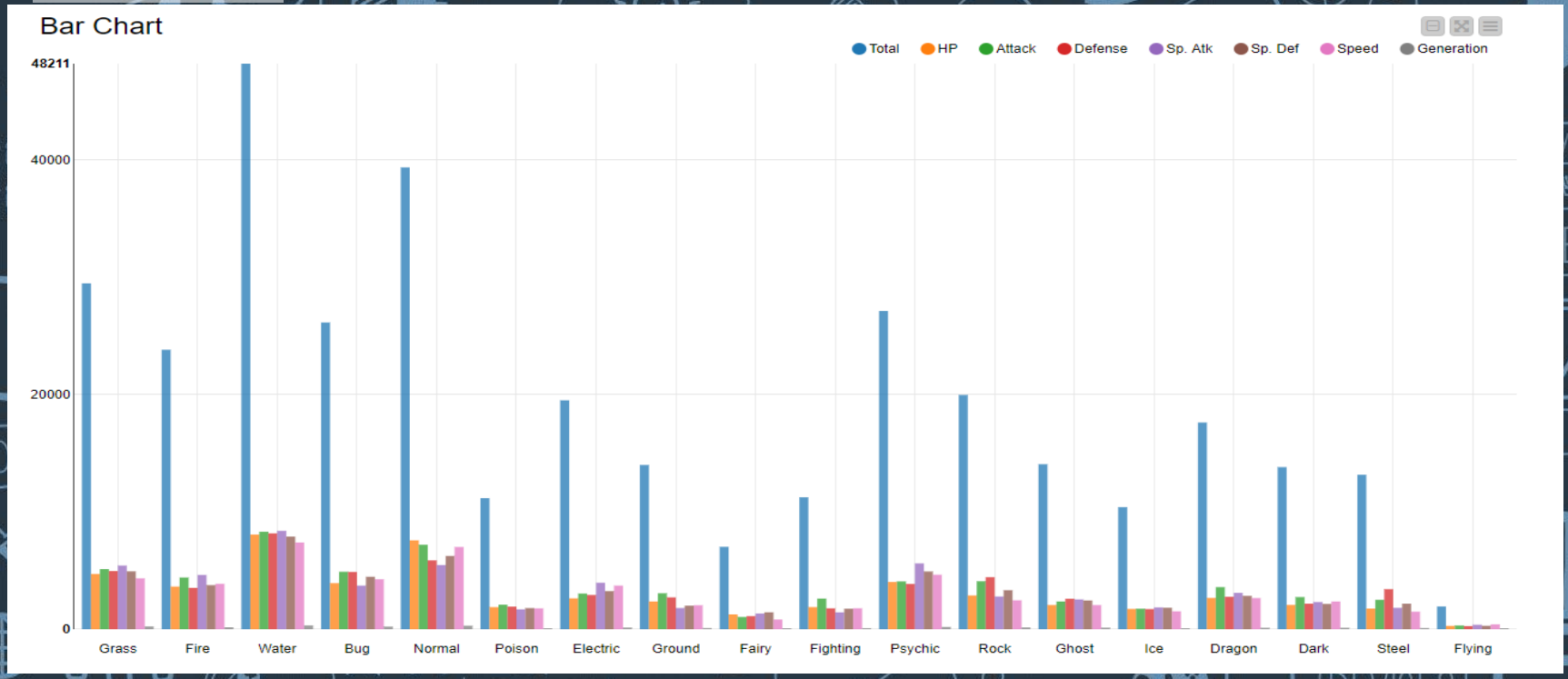
In [146]:
```
result = classifier.predict(test1)
if result[0] == 1 :
    print("The stronger pokemon is " + name2 )
else :
    print("The stronger pokemon is " +name1)
```

```
The stronger pokemon is Electabuzz
```

In [ ]:

Visualizations performed

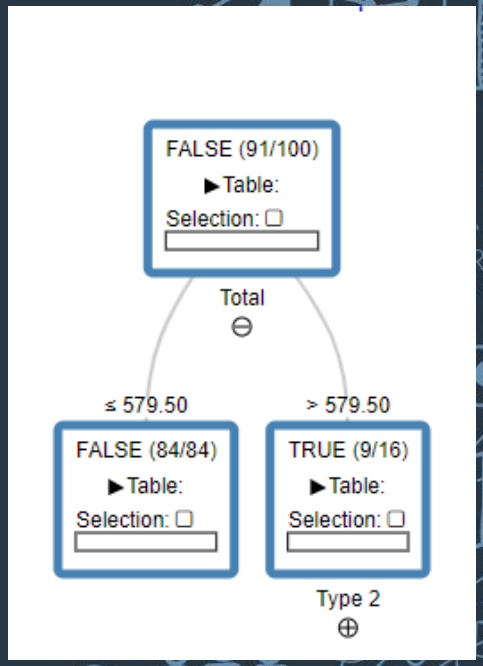
1. Bar Chart



Here we have analysed all Pokémon based on various factors and also in totality and have drawn our inferences. For example, water type is the strongest and flying is the weakest in totality. Another observation is dragon type Pokémon attack more than they defend.

Visualizations performed

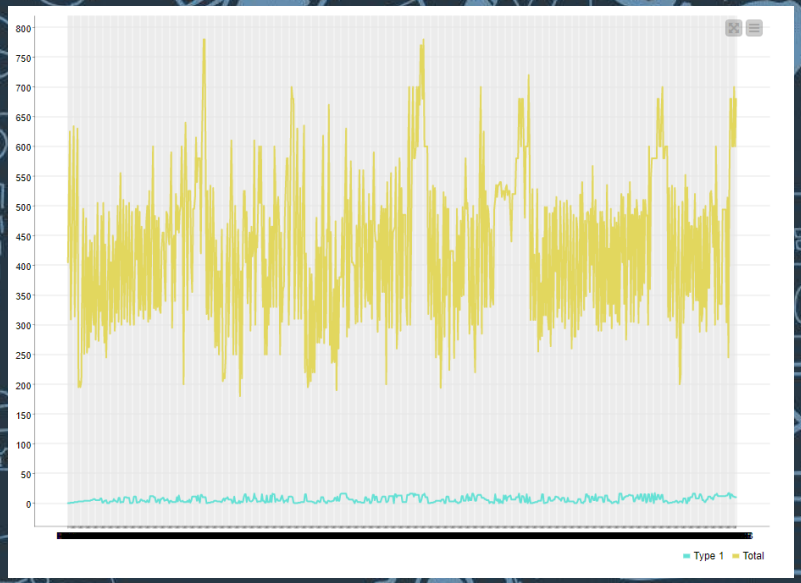
Legendary or not:
9 are legendary but 84 are not.

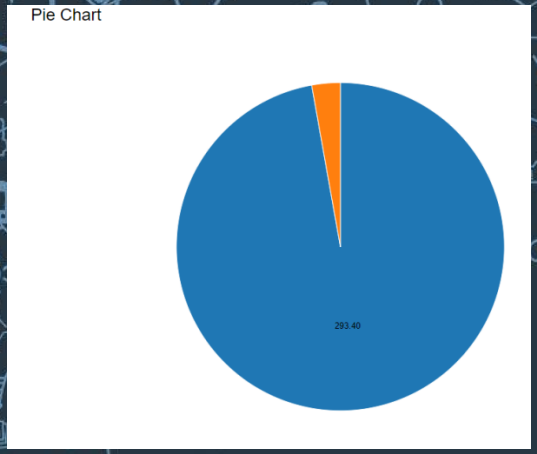


Legendary...	FALSE	TRUE
FALSE	614	15
TRUE	15	31

Correct classified: 645 Wrong classified: 30
Accuracy: 95.556 % Error: 4.444 %
Cohen's kappa (κ) 0.65

Type 1 vs Total: Range of values

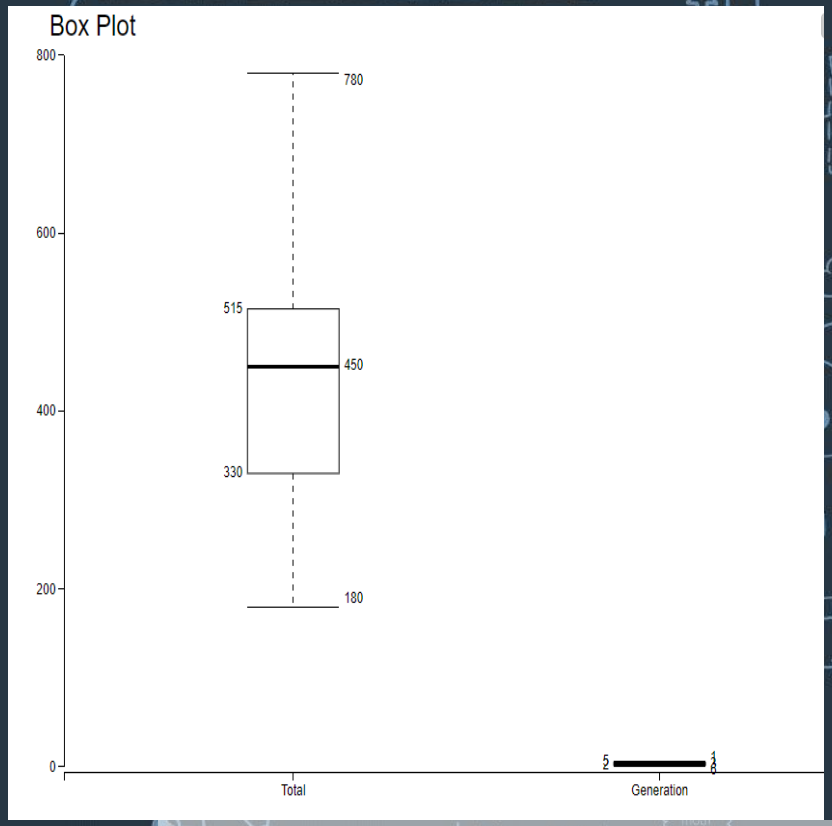


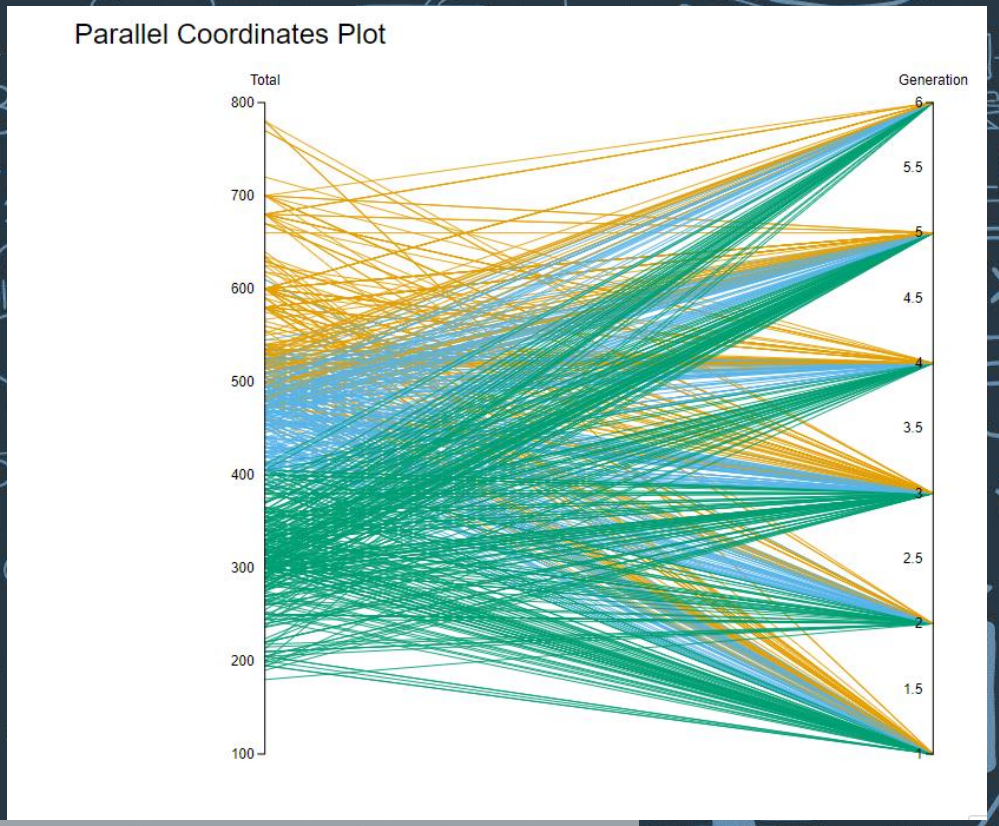


False: 90.11
True: 9.89

Visualizations performed

3. Total factors of Pokémon based on their generation

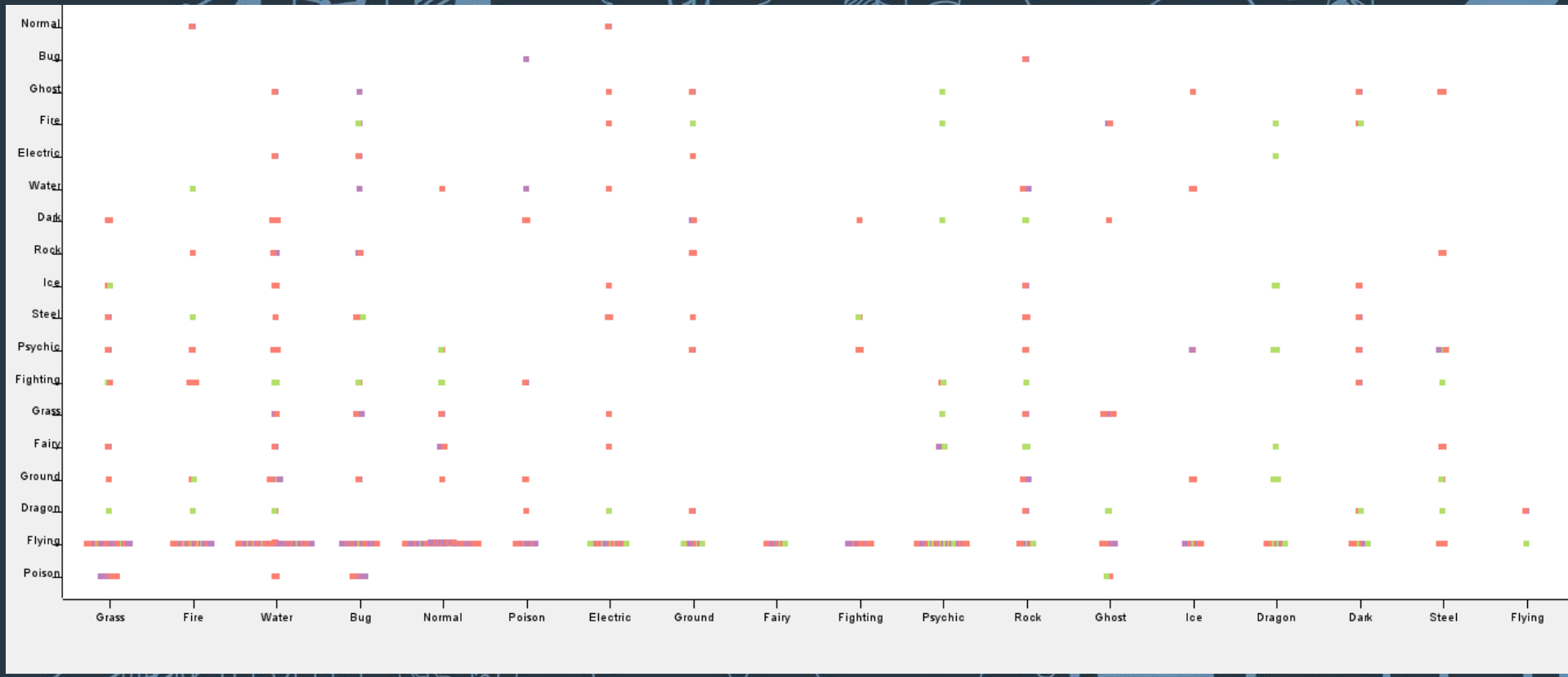




Strongest Generation: 1; Weakest Generation: 6

Visualizations performed

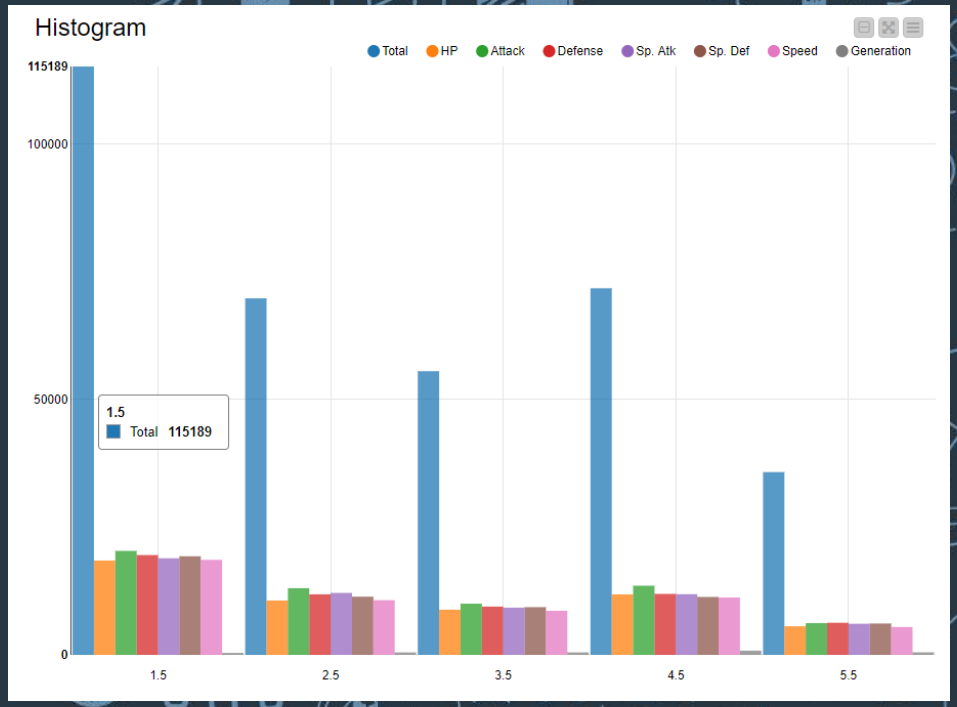
4. Pokémon analysis based on their two types



Flying type is found in many Pokémon.

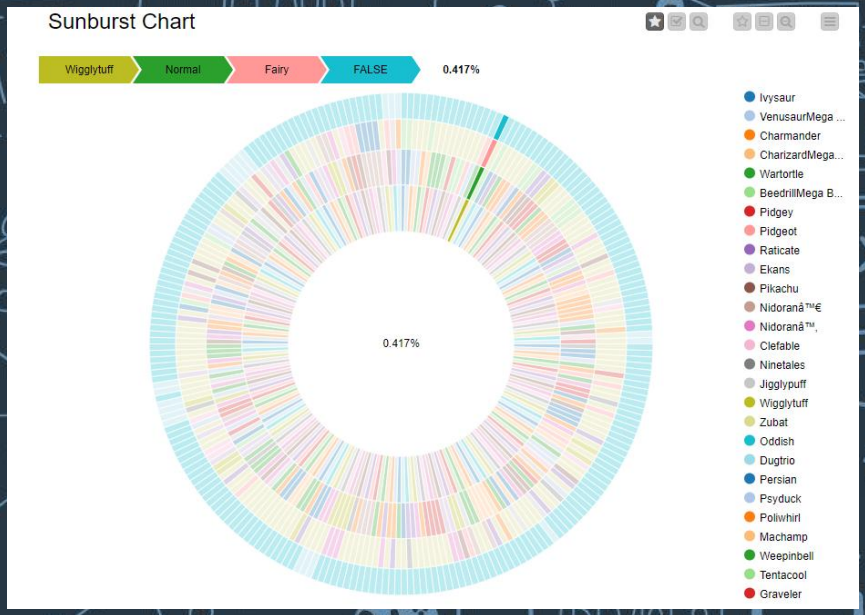
Visualizations performed

5. Factors VS Generation

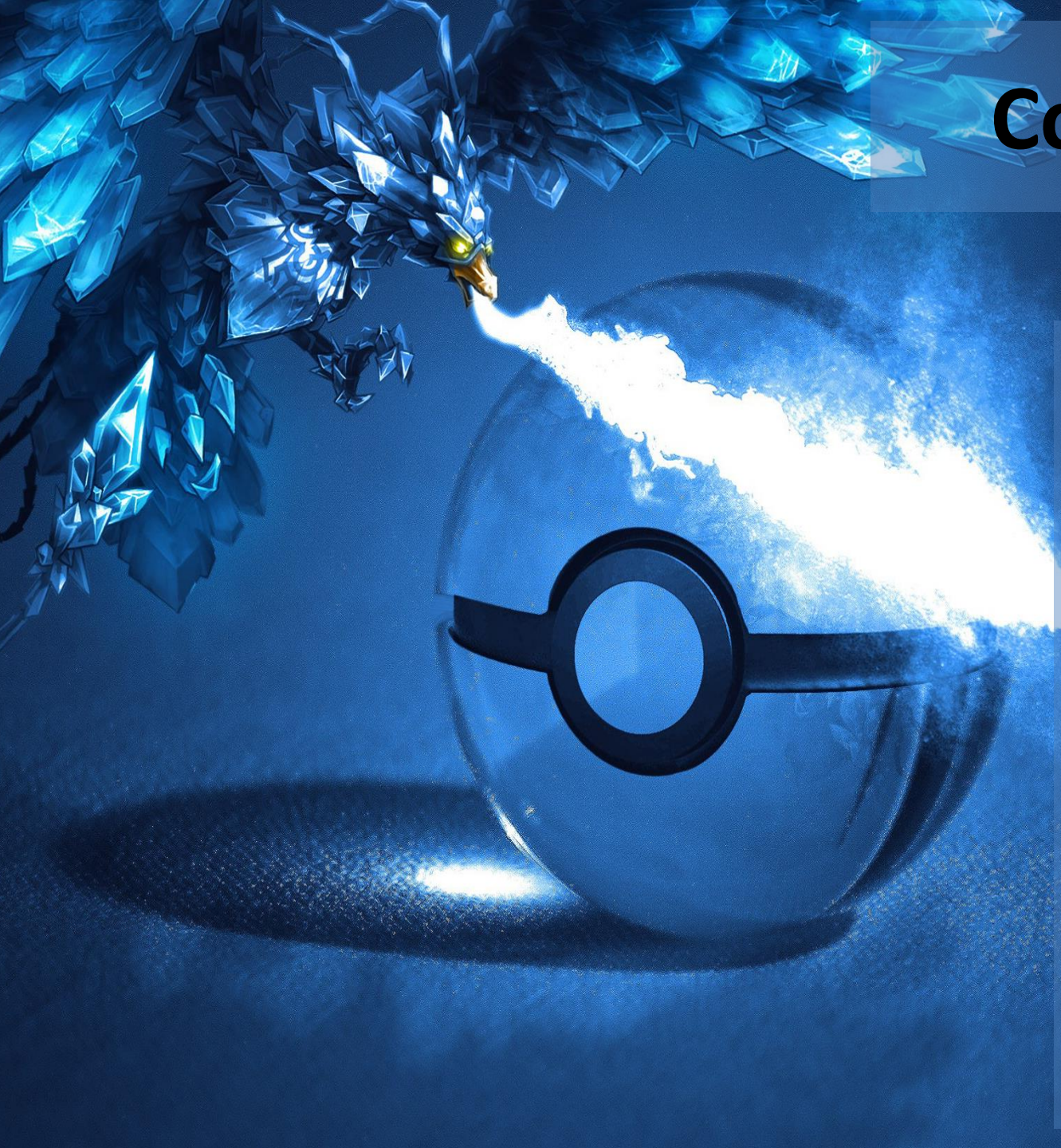


Generation 1 is best in all formats.

6. Displaying all Pokémon- their types, Legendary or not



Eg: Wigglytuff has types Fairy and Normal. It is not legendary.



Conclusion

1. Successfully created a Pokémon battle prediction system to analyse the result of a Pokémon battle by factoring in the total stats and type matchups of each Pokémon.

2.. Pokémon stat and comparison graphs were depicted that give us comprehensive information about each Pokémon.