Report

Poking around the Pokémon: The Pokémon Analysis

By:

Priyanka Lalwani(202218058)

Ayush Jain(202218039)

Anisha Anilkumar(202218038)

Description:

The motivation of this project is to further understand the dynamics of the Pokémon universe through data. We have taken dataset comprising of various attributes of different Pokémons and performed data analysis on it.

With this dataset, our aim is to be able to answer questions like:

- How many Pokémons are there in different generations?
- How is height and weight of a Pokémon correlated to each other?
- Which type of Pokémon is the easiest to capture?
- How different attributes like attack, defense, HP, speed are correlated to each other?
- Which Pokémon is the most powerful overall? Which is the weakest?

And many more…

The dataset contains information of over 800 Pokémons with 35 attributes from seven generations. The information contained in this dataset include attributes like Generation, Legendary, Base Stats, Performance against Other Types, Height, Weight, Abilities, etc.

The main attributes used in this dataset:

- **Type1:**

  The primary type of the Pokémon.

- **Type2:**

  The secondary type of the Pokémon. It also contains some null values because some Pokémons belong to only one type.

- **Generation:**

  The numbered generation in which the Pokémon was first introduced.

- **Legendary Pokémon:**
  Legendary Pokémon are typically rare and hard to get, usually being restricted to one or two of each species.

- **HP (Hit point):**

  All Pokémon start out with full HP at capture but HP can be depleted during battle by taking hits. When you use your Pokémon in a battle against other players, say when you are attacking or defending, any damage done to your Pokémon subtracts from the Pokémon's total no. of Hit Points. So, HP is kind of like a measure of your Pokémon's stamina and health. Defense reduces damage from each attack your Pokémon receives, while Higher HP Pokémon will require more hits to take down.

- **Speed(mph):**
  Speed is how fast the Pokémon is.

- **Attack:**

  Attack is a value that determines how much damage a Pokémon will cause to the opponent while using a physical move.

- **Defense:**

  Defense determines how much damage a Pokémon will resist when hit by a physical move.

- **Base_happiness:**

  A hidden value from 0 to 255 called Happiness (also known as Tameness or Friendship) is given to all Pokémon in the game, determining how friendly your Pokémon is.

- **Base_total:**

  Sum of HP, attack, defense, special attack, special defense and speed.

- **Capture Rate:**
  The Pokémon's Catch Rate is a number between 0 and 255, the higher the better because it would require less Pokéballs to capture the Pokémon. The Pokémon with lower capture rate is difficult to capture.

- **height_m:**

  Height of the Pokémon in metres.

- **weight_kg:**

  The weight of the Pokémon in kilograms.

- **abilities:**

  A stringified list of abilities that the Pokémon is capable of having.

Libraries used in the analysis of the data

```
!pip install squarify
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import squarify
from ast import literal_eval
```

- import numpy as np: We have imported the numpy library as ‘np’ variable. Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- import pandas as pd: We have imported pandas library as ‘pd’ variable. Pandas uses fast, flexible, and expressive data structures designed to make working with relational or labeled data both easy and intuitive.

- import matplotlib.pyplot as plt: We have import matplotlib.pyplot as ‘plt’ variable. It is a state-based interface to matplotlib. It provides an implicit, MATLABlike, way of plotting. It also opens figure on your screen and acts as the figure GUI manager.

- import seaborn as sns: We have imported seaborn library as ‘sns’ variable. It is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

- Import squarify: Squarify is the best fit when you have to plot a Treemap. Treemaps display hierarchical data as a set of nested squares/rectangles-based visualization.

- The ast. literal_eval method is one of the helper functions that helps traverse an abstract syntax tree. This function evaluates an expression node or a string consisting of a Python literal or container display.

Reading the Data:

We start by loading the dataset. For this we use:

```
pokemon_df = pd.read_csv('C:\\Users\\ayxxh\\Desktop\\DataSpell\\Project\\pokemon.csv')
```

Our dataset is in .csv format so we use pd.read.csv followed by the path where the file is located.
After loading the file, we perform some actions to get the idea about the structure and attributes in the data.

```
pokemon_df
```

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	against_fire	against_flyi
0	['Overgrow', 'Chlorophyll']	1.00	1.0	1.0	0.5	0.5	0.50	2.0	
1	['Overgrow', 'Chlorophyll']	1.00	1.0	1.0	0.5	0.5	0.50	2.0	
2	['Overgrow', 'Chlorophyll']	1.00	1.0	1.0	0.5	0.5	0.50	2.0	
3	['Blaze', 'Solar Power']	0.50	1.0	1.0	1.0	0.5	1.00	0.5	
4	['Blaze', 'Solar Power']	0.50	1.0	1.0	1.0	0.5	1.00	0.5	
5	['Blaze', 'Solar Power']	0.25	1.0	1.0	2.0	0.5	0.50	0.5	
6	['Torrent', 'Rain Dish']	1.00	1.0	1.0	2.0	1.0	1.00	0.5	
7	['Torrent', 'Rain Dish']	1.00	1.0	1.0	2.0	1.0	1.00	0.5	
8	['Torrent', 'Rain Dish']	1.00	1.0	1.0	2.0	1.0	1.00	0.5	
9	['Shield Dust', 'Run Away']	1.00	1.0	1.0	1.0	1.0	0.50	2.0	
10	['Shed Skin']	1.00	1.0	1.0	1.0	1.0	0.50	2.0	
11	['Compoundeyes', 'Tinted Lens']	0.50	1.0	1.0	2.0	1.0	0.25	2.0	
12	['Shield Dust', 'Run Away']	0.50	1.0	1.0	1.0	0.5	0.25	2.0	
13	['Shed Skin']	0.50	1.0	1.0	1.0	0.5	0.25	2.0	
14	['Swarm', 'Sniper']	0.50	1.0	1.0	1.0	0.5	0.25	2.0	
15	['Keen Eye', 'Tangled Feet', 'Big Pecks']	0.50	1.0	1.0	2.0	1.0	1.00	1.0	
16	['Keen Eye', 'Tangled Feet', 'Big Pecks']	0.50	1.0	1.0	2.0	1.0	1.00	1.0	
17	['Keen Eye', 'Tangled Feet', 'Big Pecks']	0.50	1.0	1.0	2.0	1.0	1.00	1.0	
18	['Run Away', 'Guts', 'Hustle', 'Glutton...	1.00	1.0	1.0	1.0	1.0	2.00	1.0	
19	['Run Away', 'Guts', 'Hustle', 'Glutton...	1.00	1.0	1.0	1.0	1.0	2.00	1.0	
20	['Keen Eye', 'Sniper']	0.5	1.0	1.0	2.0	1.0	1.0	1.0	
21	['Keen Eye', 'Sniper']	0.5	1.0	1.0	2.0	1.0	1.0	1.0	
22	['Intimidate', 'Shed Skin', 'Unnerve']	0.5	1.0	1.0	1.0	0.5	0.5	1.0	
23	['Intimidate', 'Shed Skin', 'Unnerve']	0.5	1.0	1.0	1.0	0.5	0.5	1.0	
24	['Static', 'Lightningrod']	1.0	1.0	1.0	0.5	1.0	1.0	1.0	

801 rows × 41 columns

We can see that we have 801 rows and 41 columns in our data frame.

```
pokemon_df.columns
```

```
Index(['abilities', 'against_bug', 'against_dark', 'against_dragon',
       'against_electric', 'against_fairy', 'against_fight', 'against_fire',
       'against_flying', 'against_ghost', 'against_grass', 'against_ground',
       'against_ice', 'against_normal', 'against_poison', 'against_psychic',
       'against_rock', 'against_steel', 'against_water', 'attack',
       'base_egg_steps', 'base_happiness', 'base_total', 'capture_rate',
       'classfication', 'defense', 'experience_growth', 'height_m', 'hp',
       'japanese_name', 'name', 'percentage_male', 'pokedex_number',
       'sp_attack', 'sp_defense', 'speed', 'type1', 'type2', 'weight_kg',
       'generation', 'is_legendary'],
      dtype='object')
```

The name of the 41 columns can be fetched by the above code.

Cleaning the Data:

After getting idea about the shape of the data, we will now clean it for our needs, we start by dropping the unwanted columns from our code:

```
pokemon_df.drop(['japanese_name', 'pokedex_number', 'base_egg_steps', 'classfication', 'experience_growth',
 'percentage_male'], axis=1, inplace=True)
```

Explanation:
Drop(): it is used for dropping the unwanted columns from the dataset. axis=1 means column and inplace = True means performed action is permanent.
Now we look for null values in each column if any:

```
pokemon_df.isnull().sum()[pokemon_df.columns[pokemon_df.isnull().any()]]
```

	data
height_m	20
type2	384
weight_kg	20

Length: 3, dtype: int64 Open in new tab

Explanation:
Isnull(): returns a Boolean if the value is a null value or not.
Sum(): here it is used to add up all the null values.
Any(): here it is used to return true if there are null values present.

From the above output we can see that we have 20 missing values in height_m and weight_kg each. While type2 has 384 missing values because not all pokemon have second type.

Now we need to add the missing value from the data so we will fetch the index of rows where data is missing:

```
pokemon_df[pokemon_df['height_m'].isna()]
```

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fair
18	['Run Away', 'Guts', 'Hustle', 'Glutton...	1.0	1.0	1.0	1.0	
19	['Run Away', 'Guts', 'Hustle', 'Glutton...	1.0	1.0	1.0	1.0	
25	['Static', 'Lightningrod', 'Surge Surfer']	1.0	1.0	1.0	0.5	
26	['Sand Veil', 'Sand Rush', 'Snow Cloak'...	1.0	1.0	1.0	0.0	
27	['Sand Veil', 'Sand Rush', 'Snow Cloak'...	1.0	1.0	1.0	0.0	
36	['Flash Fire', 'Drought', 'Snow Cloak',...	0.5	1.0	1.0	1.0	
37	['Flash Fire', 'Drought', 'Snow Cloak',...	0.5	1.0	1.0	1.0	
49	['Sand Veil', 'Arena Trap', 'Sand Force...	1.0	1.0	1.0	0.0	
50	['Sand Veil', 'Arena Trap', 'Sand Force...	1.0	1.0	1.0	0.0	
51	['Pickup', 'Technician', 'Unnerve', 'Pi...	1.0	1.0	1.0	1.0	
52	['Limber', 'Technician', 'Unnerve', 'Fu...	1.0	1.0	1.0	1.0	
73	['Rock Head', 'Sturdy', 'Sand Veil', 'M...	1.0	1.0	1.0	0.0	
74	['Rock Head', 'Sturdy', 'Sand Veil', 'M...	1.0	1.0	1.0	0.0	
75	['Rock Head', 'Sturdy', 'Sand Veil', 'M...	1.0	1.0	1.0	0.0	
87	['Stench', 'Sticky Hold', 'Poison Touch...	0.5	1.0	1.0	1.0	
88	['Stench', 'Sticky Hold', 'Poison Touch...	0.5	1.0	1.0	1.0	
102	['Chlorophyll', 'Harvest', 'Frisk', 'Ha...	4.0	2.0	1.0	0.5	
104	['Rock Head', 'Lightningrod', 'Battle A...	1.0	1.0	1.0	0.0	
719	['Magician']	1.0	4.0	1.0	1.0	
744	['Keen Eye', 'Sand Rush', 'Steadfast', ...	1.0	1.0	1.0	1.0	

20 rows × 35 columns

Similarly, we did for weight_kg. Then we created dictionary to store missing values against indices as keys.

```
heights_to_add = {18:0.3, 19:0.7, 25:0.8, 26:0.6, 27:1.0, 36:0.6, 37:1.1, 49:0.2, 50:0.7, 51:0.4, 52:1.0, 73:0.4,
 74:1.0, 75:1.4, 87:0.9, 88:1.2, 102:2.0, 104:1.0, 719:0.5, 744:0.8 }

weights_to_add = {18:3.5, 19:18.5 , 25:30.0 , 26:12.0 , 27:29.5, 36:9.9, 37:19.9, 49:0.8, 50:33.3, 51:4.2, 52:32.0,
 73:20.0, 74:105.0, 75:300.0, 87:30.0, 88:30.0 , 102:120.0, 104:45.0, 719:9.0, 744:25.0}
```

Now we need to add these values to the dataset:

```
for i in heights_to_add.keys():
    pokemon_df.at[i, 'height_m'] = heights_to_add[i]

for i in weights_to_add.keys():
    pokemon_df.at[i, 'weight_kg'] = weights_to_add[i]
```

To add the data, we used a for loop and at() method .

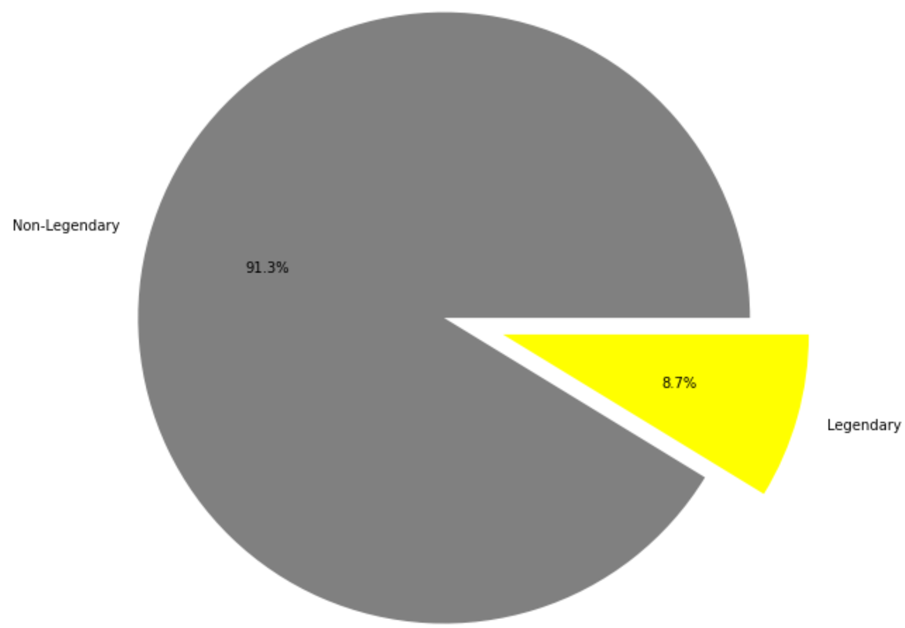
Now, we have cleaned the data to perform required analysis on it.

Analysis based on Legendary :

```
pokemon_df['is_legendary']
l_freq = pokemon_df.is_legendary.value_counts()
print(l_freq)
```

```
0    731
1     70
Name: is_legendary, dtype: int64
```

```
plt.figure(figsize=(10, 10))
plt.pie(l_freq,
        labels=['Non-Legendary', 'Legendary'],
        autopct='%2.1f%%',
        colors=['grey', 'yellow'],
        startangle=0,
        explode=[0.1, 0.1])
```



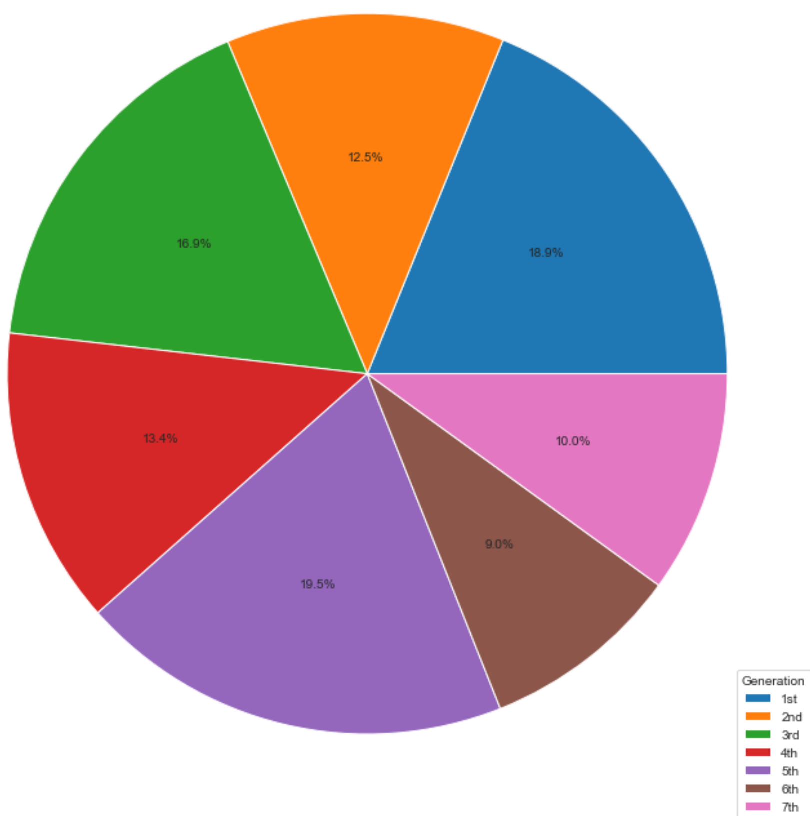
Conclusion : *From the pie chart we can see that out of all the Pokemons, only 8.7% are legendary and the rest 91.3% are non-legendary. We can conclude that the legendary Pokemon are very rare to find.*

Analysis based on Generation of Pokémons:

```
gen_freq = pokemon_df.generation.value_counts(sort=False)
print(gen_freq)
```

```
1    151
2    100
3    135
4    107
5    156
6     72
7     80
Name: generation, dtype: int64
```

```
gen = ['1st', '2nd', '3rd', '4th', '5th', '6th', '7th']
plt.figure(figsize=(10, 10))
plt.pie(gen_freq,
        autopct='%2.1f%%',
        startangle=0,
        )
plt.legend(labels=gen, loc=4, title='Generation')
plt.tight_layout()
```



Conclusion: *5th generation has the most number of Pokemons, followed by 1st and 3rd generations. Also, odd genartions have more Pokemons compared to even ones.*

Analysis of Primary Type of the Pokémons:

```
t1_freq=pokemon_df.type1.value_counts()
print(t1_freq)
```

```
poison      32
ground      32
dark        29
fighting    28
ghost       27
dragon      27
steel       24
ice         23
fairy       18
flying       3
Name: type1, dtype: int64
```

```
plt.figure(figsize=(18, 12))
tm=squarify.plot(t1_freq,
                 label=t1_freq.index,
                 color=['#00BFFF','#EED5D2','#C0FF3E','#8B4500','#FF3030','#FF8000','#696969','#FFD700','#68228B','#8A360F',
                  '#000000','#FF00FF','#CDCDC1','#8B2252','#A9A9A9','#C6E2FF','#EE82EE','#000080'],
                 pad=0.1,
                 text_kwargs={'fontsize': 22, 'color': 'white'})
tm.set_title('Primary Type', fontsize=28, pad=20)
plt.axis('off')
```

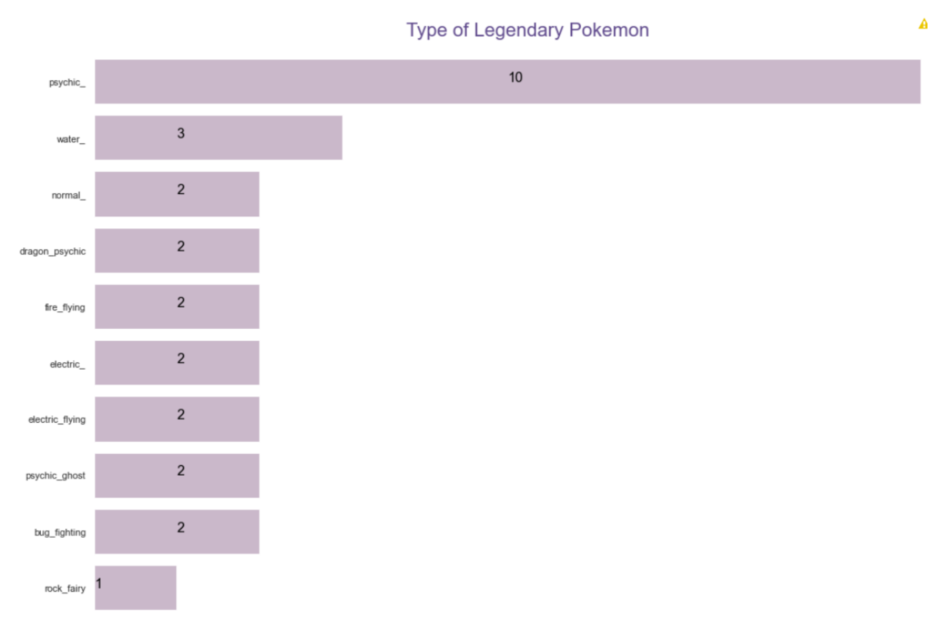
Conclusion: *From the TreeMap we can see that water type Pokemon are the most common, followed by normal and grass. While the least common one is flying type.*

Type of legendary Pokémon Bar plot:

```
pokemon_df['type'] = pokemon_df['type1'] + '_' + pokemon_df['type2'].fillna('')
legendary = pokemon_df[pokemon_df['is_legendary'] == 1]
legendary_type = legendary["type"].value_counts(sort=True)[:10]
legendary_type
```

```
psychic_          10
water_             3
normal_            2
dragon_psychic     2
fire_flying        2
electric_          2
electric_flying    2
psychic_ghost      2
bug_fighting       2
rock_fairy         1
Name: type, dtype: int64
```

```
plt.figure(figsize=(18, 12))
sns.set(style='white')
b=sns.barplot(y=legendary_type.index,
              x=legendary_type.values,
              orient='h',
              color='#CDB5CD')

b.set(xticklabels=[])
sns.despine(top=True, right=True, left=True, bottom=True)
b.set_title('Type of Legendary Pokemon', fontsize=22, pad=20, color='#5D478B')

for index, value in enumerate(legendary_type):
    plt.annotate(f'{value}', xy=(value//2, index), color='black', fontsize=16)
```



Conclusion: *From the bar plot, we can conclude that psychic is the most common type(primary or secondary) of legendary pokemon.*

Primary and Secondary Type:

```
type1_freq = pokemon_df.type1.value_counts()
type2_freq = pokemon_df.type2.value_counts()
types_data = pd.concat([type1_freq, type2_freq], axis=1)
types_data
```

	type1	type2
water	114	17
normal	105	4
grass	78	20
bug	72	5
psychic	53	29
fire	52	13
rock	45	14
electric	39	9

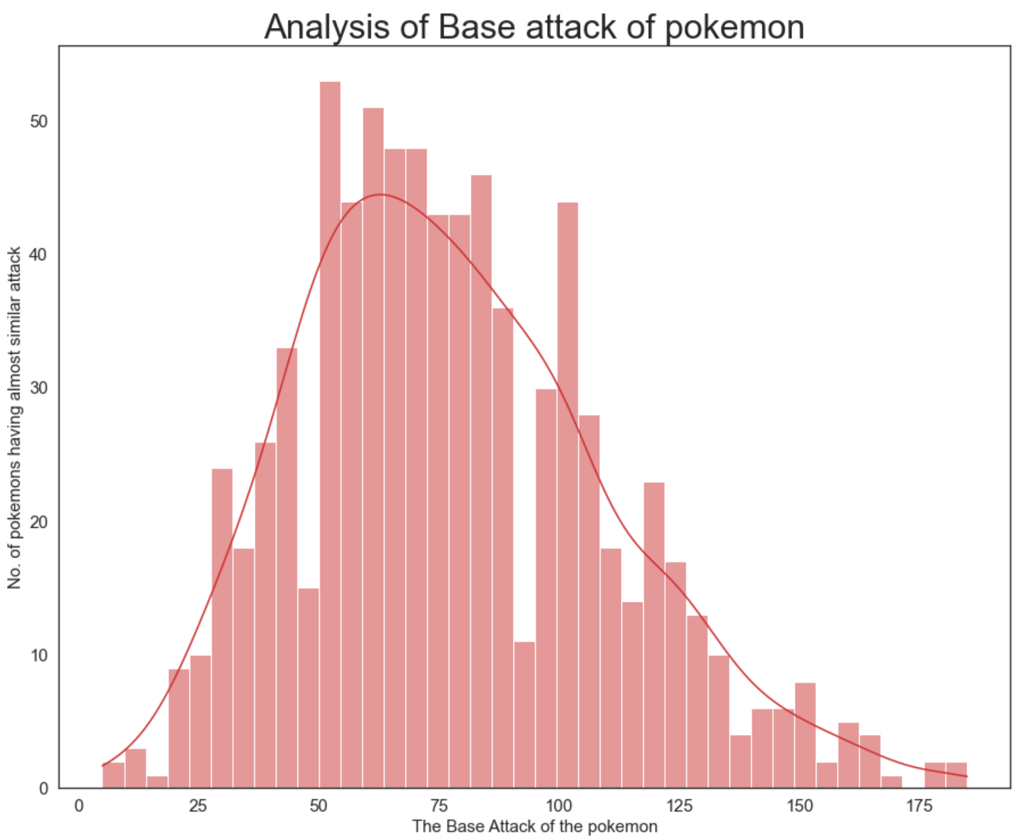
18 rows × 2 columns Open in new tab

```
types_data.plot.barh(stacked=True, color=['#8B7B8B', '#CDB5CD'], width=0.8, figsize=(20, 10))
sns.despine(top=True, right=True, left=True)
for index, row in enumerate(types_data.iterrows()):
    plt.annotate(f'{row[1]["type1"]}', xy=(1, index), color='white')
    plt.annotate(f'{row[1]["type2"]}', xy=(row[1]['type1'] + 1, index), color='white')
```

Conclusion: *From the stacked bar plot, we can conclude that flying is the most common secondary type, followed by poison and ground. Whereas normal is the least common secondary type.Water is overall the most common type.*Though flying is the least common primary type but it is one of the most common type overall.

Analysis of Base attack of Pokémons:

```
plt.figure(figsize=(15,12))
sns.histplot(pokemon_df,x='attack', bins=40, color='#CD3333', kde=True)
plt.xlabel("The Base Attack of the pokemon",fontsize=15)
plt.ylabel("No. of pokemons having almost similar attack",fontsize=15)
plt.title("Analysis of Base attack of pokemon",fontsize=30)
```



Conclusion: *From the distribution plot, we can conclude that the base attack is normally distributed.Many Pokemon have attack values between 50 and 100.There are very few Pokemon with attack value greater than 150.*

```
pokemon_df.attack.describe()
```

```
count    801.000000
mean      77.857678
std       32.158820
min        5.000000
25%       55.000000
50%       75.000000
75%      100.000000
max      185.000000
Name: attack, dtype: float64
```

Data of top 5 Pokémons having best Base Attack:

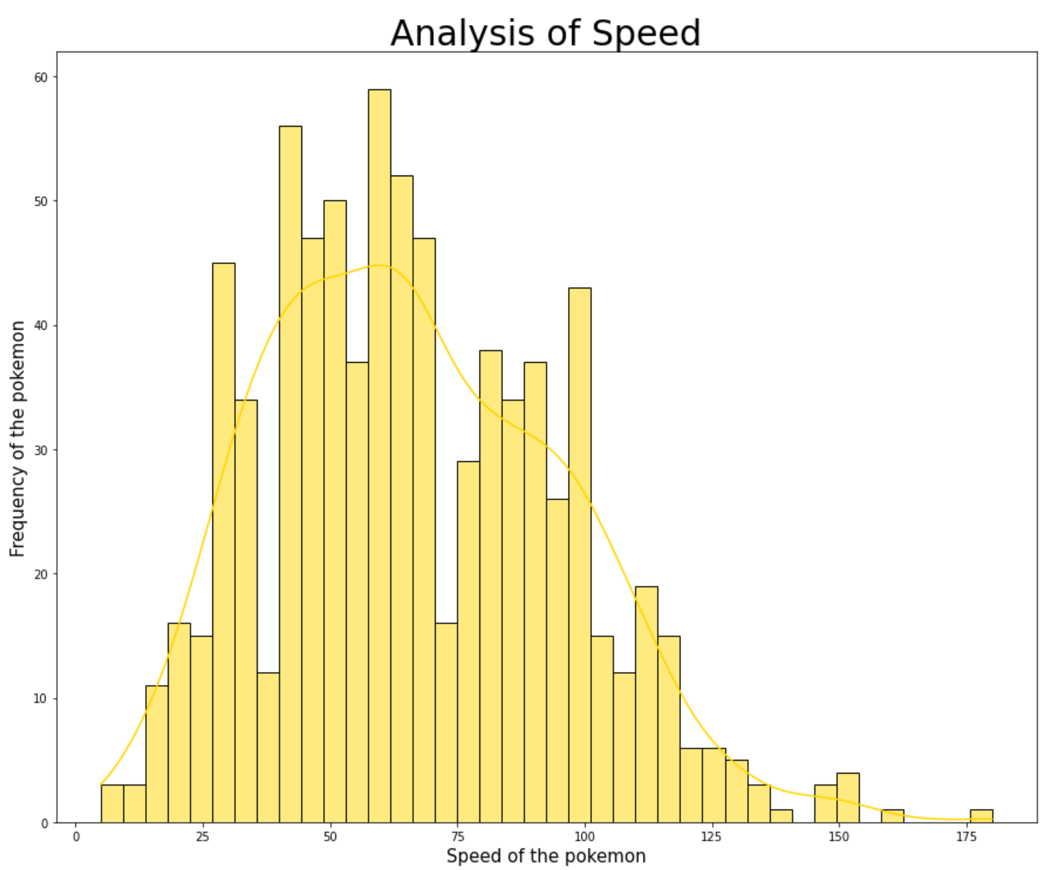
```
pokemon_df.nlargest(5,'attack')
```

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy
213	['Swarm', 'Guts', 'Moxie']	0.5	0.5	1.0	1.0	
797	['Beast Boost']	1.0	1.0	0.5	0.5	
382	['Drought']	1.0	1.0	1.0	0.0	
383	['Air Lock']	0.5	1.0	2.0	1.0	
444	['Sand Veil', 'Rough Skin']	1.0	1.0	2.0	0.0	

5 rows × 36 columns Open in new tab

Analysis based on Speed of Pokémons:

```
plt.figure(figsize=(15,12))
sns.histplot(pokemon_df,x='speed', bins=40, color='#FFD700', kde=True)
plt.xlabel("Speed of the pokemon",fontsize=15)
plt.ylabel("Frequency of the pokemon",fontsize=15)
plt.title("Analysis of Speed",fontsize=30)
```



Conclusion: *The distribution is not normal. It is right skewed.Many Pokemon have speed between 50 and 75.There are very few Pokemon with speed greater than 150. Many of the Pokémons are in the range of 25-100mph.*

```
pokemon_df.speed.describe()
```

```
count    801.000000
mean      66.334582
std       28.907662
min        5.000000
25%       45.000000
50%       65.000000
75%       85.000000
max      180.000000
Name: speed, dtype: float64
```

Data of top 5 Pokémons having best speed:

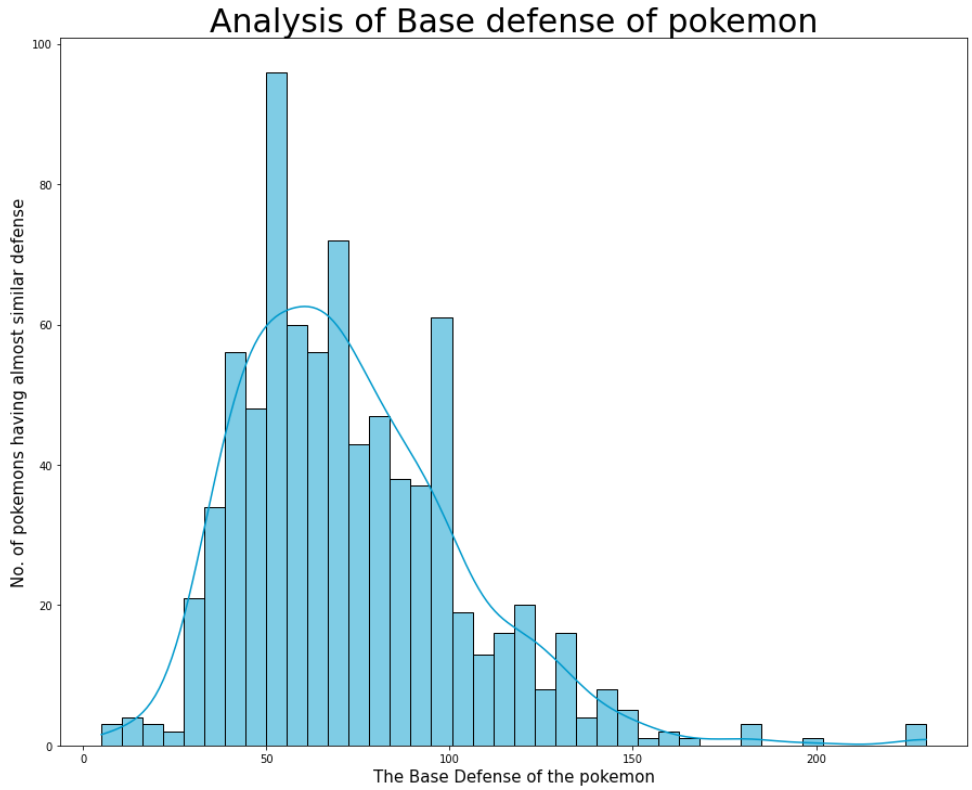
```
pokemon_df.nlargest(5,'speed')
```

	abilities	against_bug	against_dark	against_dragon	against_electric
385	['Pressure']	2.0	2.0	1.0	1.0
290	['Speed Boost', 'Infiltrator']	0.5	1.0	1.0	2.0
794	['Beast Boost']	0.5	0.5	1.0	1.0
64	['Synchronize', 'Inner Focus', 'Magic G...	2.0	2.0	1.0	1.0
100	['Soundproof', 'Static', 'Aftermath']	1.0	1.0	1.0	0.5

5 rows × 36 columns Open in new tab

Analysis of Base defense of Pokémon:

```
plt.figure(figsize=(15,12))
sns.histplot(pokemon_df, x='defense', bins=40, color='#009ACD', kde=True)
plt.xlabel("The Base Defense of the pokemon",fontsize=15)
plt.ylabel("No. of pokemons having almost similar defense",fontsize=15)
plt.title("Analysis of Base defense of pokemon",fontsize=30)
```



Conclusion: *The distribution is not normal. It is rightly skewed.Many Pokemon have defence value between 50 and 100.There are very few Pokemon with defence value greater than 150.*

```
pokemon_df.defense.describe()
```

```
count    801.000000
mean      73.008739
std       30.769159
min        5.000000
25%       50.000000
50%       70.000000
75%       90.000000
max      230.000000
Name: defense, dtype: float64
```

<u>Data of top 5 Pokémons having best Base Defence</u>

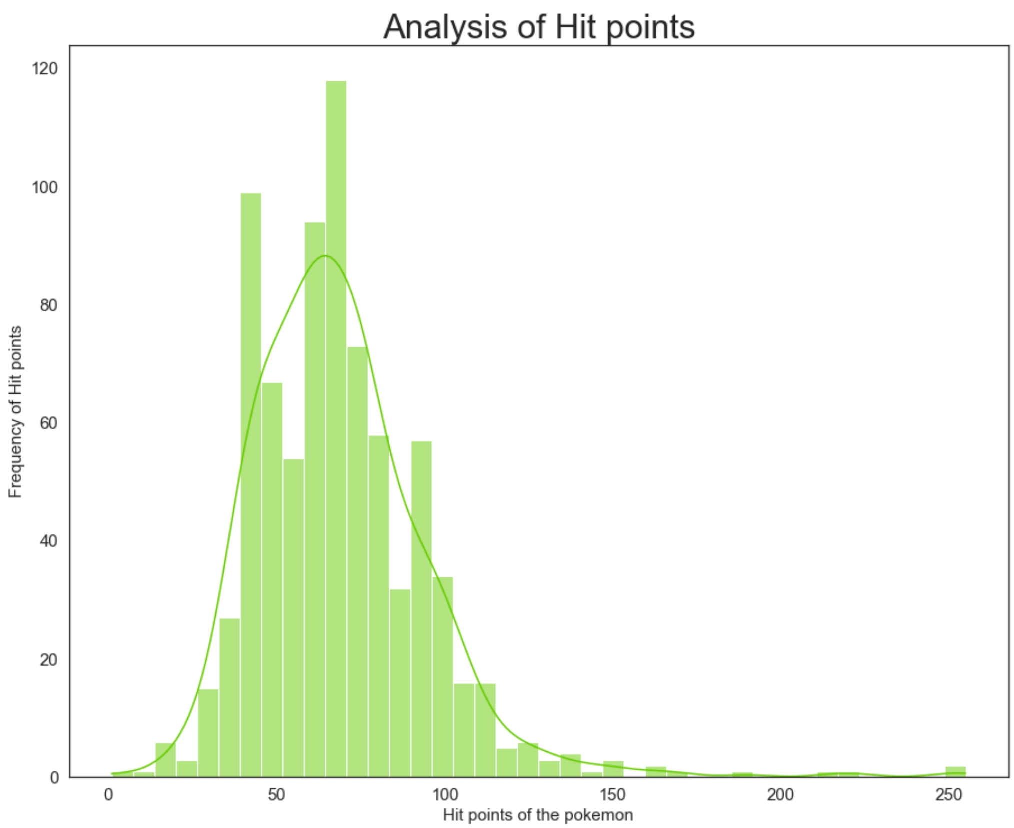
```
pokemon_df.nlargest(5,'defense')
```

	abilities	against_bug	against_dark	against_dragon	against_electric	ag
207	['Rock Head', 'Sturdy', 'Sheer Force']	0.5	1.0	0.5	0.0	
212	['Sturdy', 'Gluttony', 'Contrary']	1.0	1.0	1.0	1.0	
305	['Sturdy', 'Rock Head', 'Heavy Metal']	0.5	1.0	0.5	1.0	
376	['Clear Body', 'Sturdy']	1.0	1.0	1.0	1.0	
712	['Own Tempo', 'Ice Body', 'Sturdy']	1.0	1.0	1.0	1.0	

5 rows × 36 columns Open in new tab

Analysis of Hit Points:

```
plt.figure(figsize=(15,12))
sns.histplot(pokemon_df,x='hp', bins=40, color='#66CD00', kde=True)
plt.xlabel("Hit points of the pokemon",fontsize=15)
plt.ylabel("Frequency of Hit points",fontsize=15)
plt.title("Analysis of Hit points",fontsize=30)
```



Conclusion: *The graph is rightly skewed.Many Pokemon have HP value between 40 and 100.There are very few Pokemon with HP value greater than 150. The frequency of no. of Pokémon having hit point 60 is highest.*

```
pokemon_df.hp.describe()
```

```
count    801.000000
mean      68.958801
std       26.576015
min        1.000000
25%       50.000000
50%       65.000000
75%       80.000000
max      255.000000
Name: hp, dtype: float64
```

Data of top 5 Pokémons having best Base HP

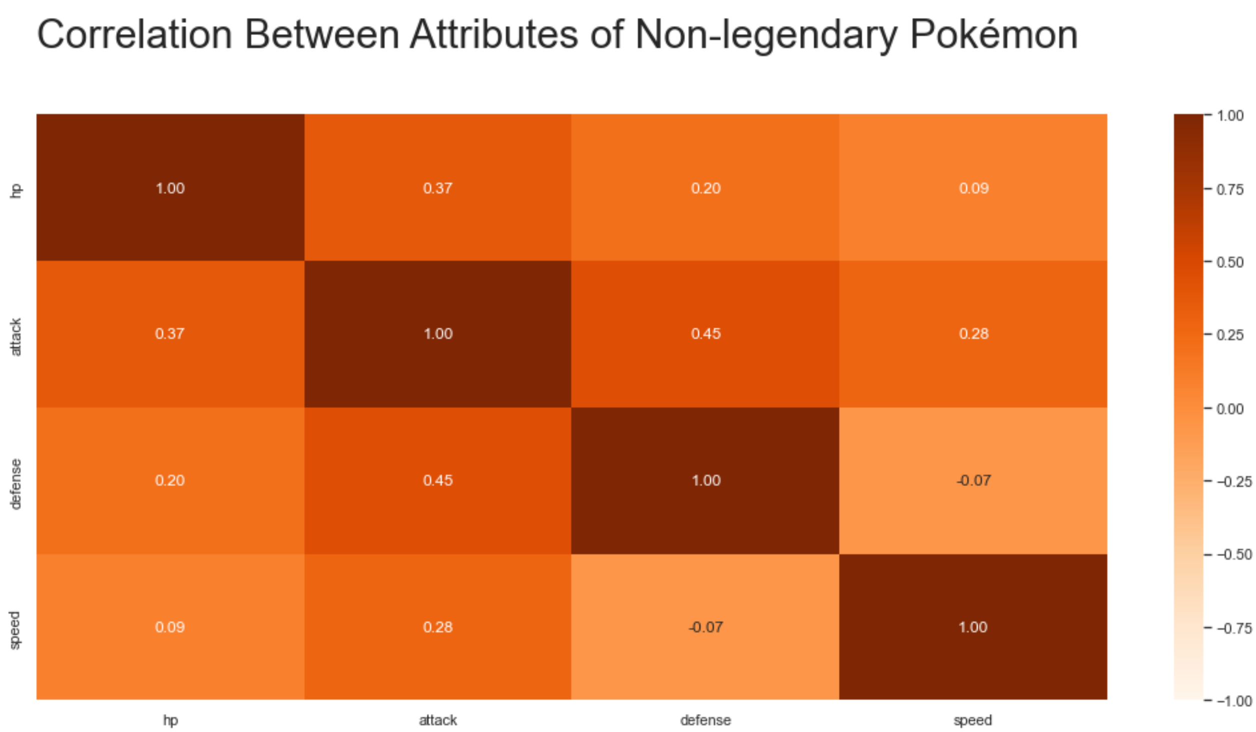
```
pokemon_df.nlargest(5,'hp')
```

	abilities	against_bug	against_dark	against_dragon	against_electric
241	['Natural Cure', 'Serene Grace', 'Healer']	1.0	1.0	1.0	1.0
112	['Natural Cure', 'Serene Grace', 'Healer']	1.0	1.0	1.0	1.0
798	['Beast Boost']	2.0	0.5	2.0	0.5
717	['Aura Break', 'Power Construct']	1.0	1.0	2.0	0.0
201	['Shadow Tag', 'Telepathy']	2.0	2.0	1.0	1.0

5 rows × 36 columns Open in new tab

Correlation Between Attributes (non-legendary):

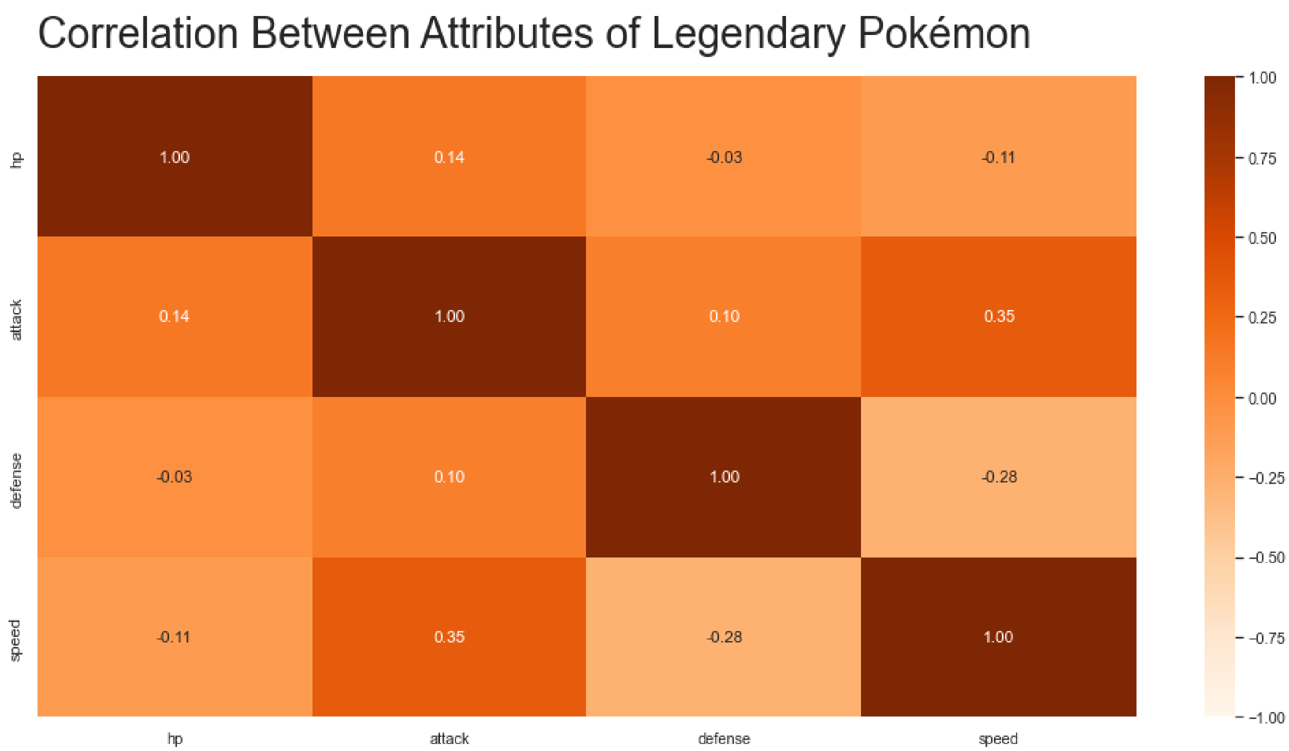
```
plt.figure(figsize=(18,8))
hads = sns.heatmap((pokemon_df[pokemon_df['is_legendary']==0].loc[:,['hp','attack','defense','speed']]).corr(),
                   annot= True,
                   fmt = ".2f",
                   vmin = -1,
                   vmax = 1,
                   cmap='Oranges')
hads.set_title('Correlation Between Attributes of Non-legendary Pokémon', loc='left', pad=50,fontsize=30);
```



Conclusion: *The diagonal of the plot all the correlations are 1.0, which is perfectly positively correlated. This is becase the diagonal compares each feature to itself. Also, if we were to fold the matrix in half down the diagonal, it would be perfectly symmetrical. The top half above the* *diagonal provides the same information as the lower half.Attack and defense are strongly correlated.Defence has a weak negative correlation with speed.*

Correlation Between Attributes (legendary):

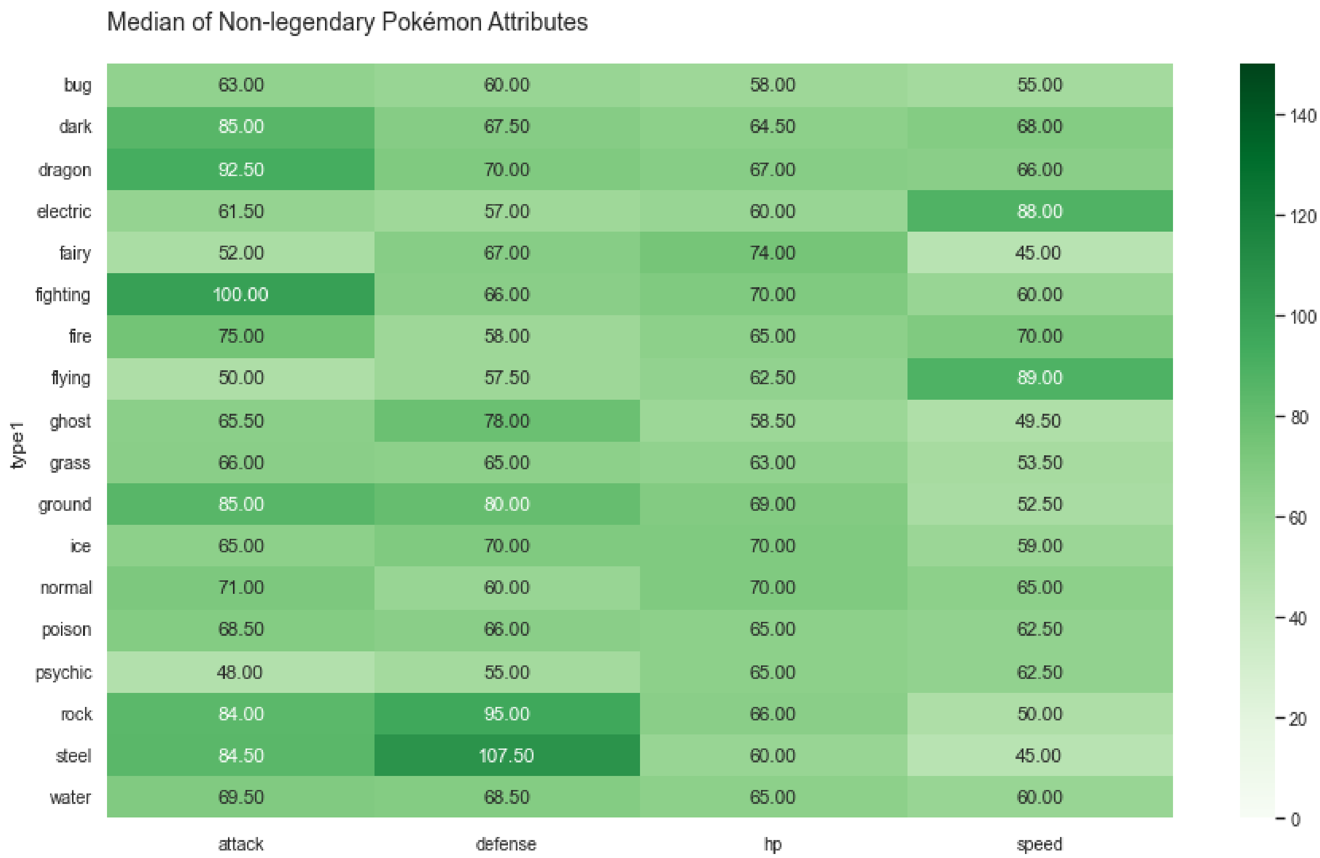
```
plt.figure(figsize=(18,8))
hads = sns.heatmap((pokemon_df[pokemon_df['is_legendary']==1].loc[:,['hp','attack','defense','speed']]).corr(),
                   annot= True,
                   fmt = ".2f",
                   vmin = -1,
                   vmax = 1,
                   cmap='Oranges')

hads.set_title('Correlation Between Attributes of Legendary Pokémon', loc='left', pad=20, fontsize=30);
```



Conclusion: *The correlation between attack and defense is weaker here compared to non-legendary.Whereas, attack & speed and defence & speed are strongly correlated positively and negatively respectively, unlike that in non-legendary.*

Median of Attributes based on non-legendary:

```
non_legendary = pokemon_df[pokemon_df['is_legendary']==0].groupby(['type1']).median()[['attack','defense','hp','speed']]
plt.figure(figsize=(15,8))
hadss=sns.heatmap(non_legendary,
                  annot= True,
                  fmt = ".02f",
                  vmin = 0,
                  vmax = 150,
                  cmap='Greens')
hadss.set_title('Median of Non-legendary Pokémon Attributes', loc='left', pad=20);
```



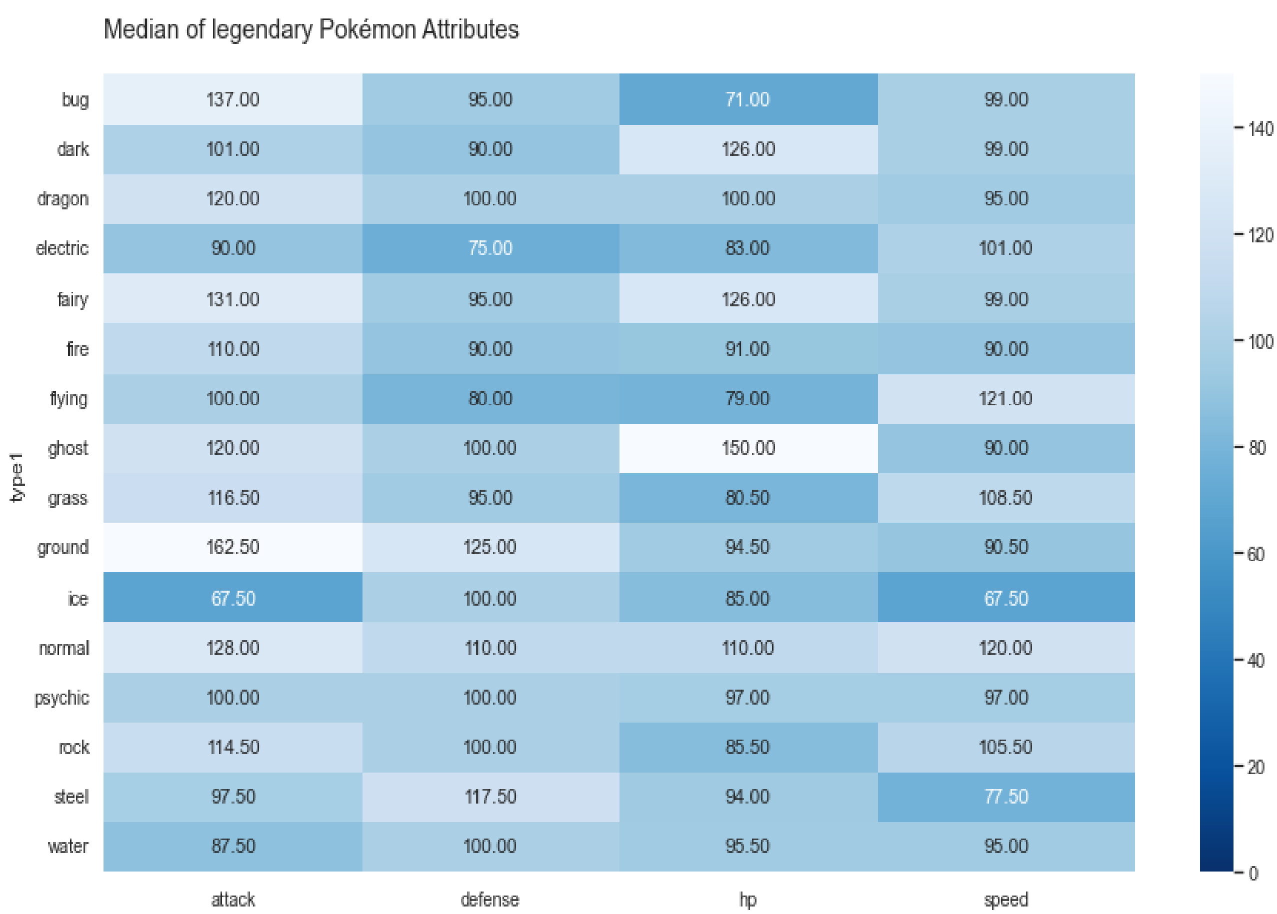
Conclusion: *Regarding non-legendary pokémon:*

- *Top 5 types - attack: fighting, dragon, ground, dark, steel*
- *Top 5 types - defense: steel, rock, ground, ghost, ice*
- *Top 5 types - hp: fairy, normal, fighting, ice, ground*
- *Top 5 types - speed: flying, electric, fire, dark, dragon*

Good types to attack are electric, fire, dark and dragon, since they are in the top 5 for attack and speed.Good types to defend are fairy, ice and ground, since they are in the top 5 for defense and hp.

Median of Attributes based on Legendary:

```
legendary = pokemon_df[pokemon_df['is_legendary'] == 1].groupby(['type1']).median()[['attack', 'defense', 'hp', 'speed']]
plt.figure(figsize=(15, 8))
hadss = sns.heatmap(legendary,
                    annot=True,
                    fmt=".02f",
                    vmin=0,
                    vmax=150,
                    cmap='Blues_r')

hadss.set_title('Median of legendary Pokémon Attributes', loc='left', pad=20);
```

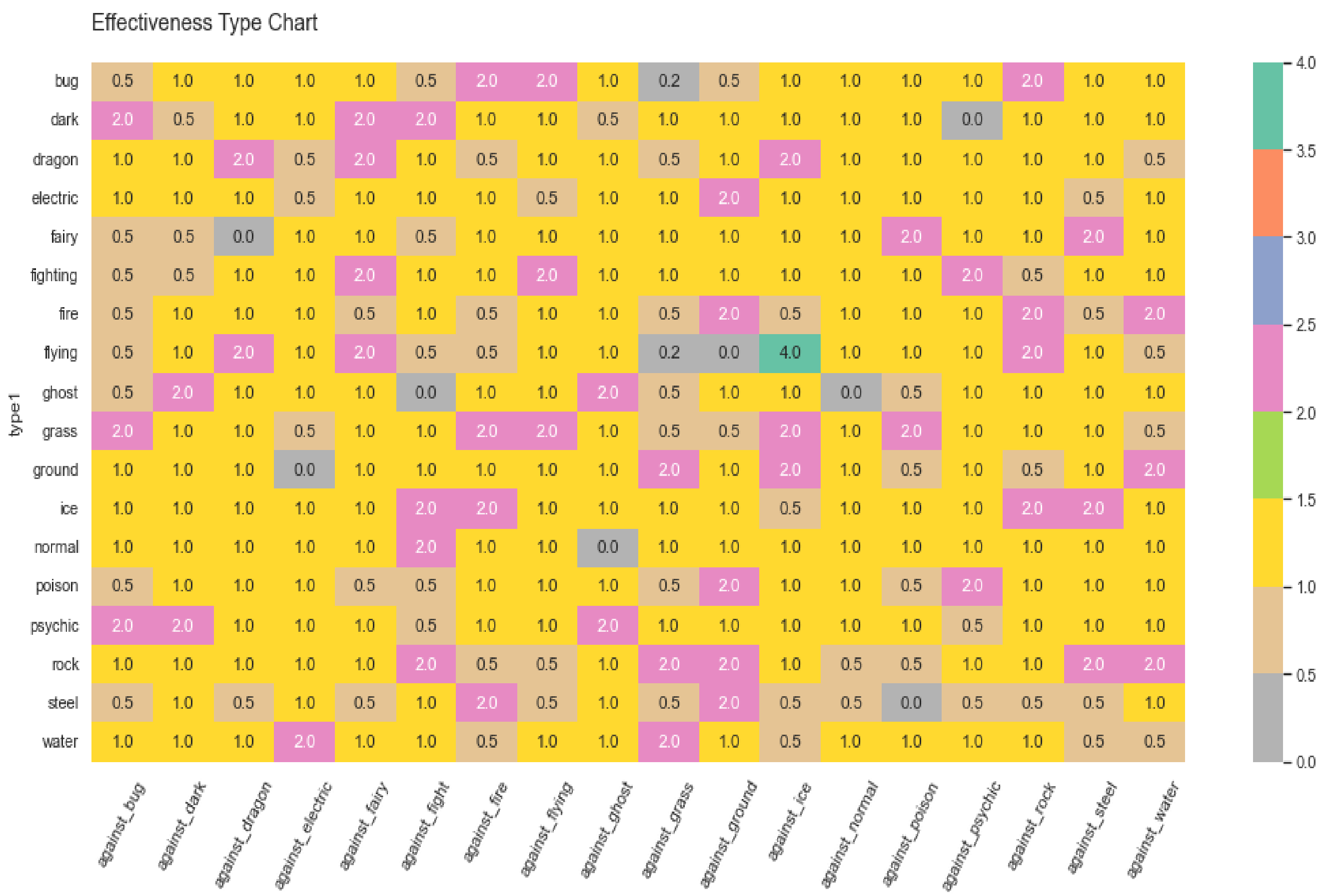


- Conclusion: *Regarding legendary pokémon:*
    - *Top 5 types - attack: ground, bug, fairy, normal, dragon*
    - *Top 5 types - defense: ground, steel, normal, dragon, ghost*
    - *Top 5 types - hp: ghost, dark, fairy, normal, dragon*
    - *Top 5 types - speed: flying, normal, grass, rock, electric*

Good types to attack are normal and electric, since they are in the top 5 for attack and speed.Good types to defend are ghost, normal and dragon, since they are in the top 5 for defense and hp.

Effectiveness Against Types:

```
against_columns = ['against_bug', 'against_dark', 'against_dragon', 'against_electric', 'against_fairy',
                   'against_fight', 'against_fire',
                   'against_flying', 'against_ghost', 'against_grass', 'against_ground', 'against_ice',
                   'against_normal', 'against_poison',
                   'against_psychic', 'against_rock', 'against_steel', 'against_water']

effectiveness = pokemon_df.groupby(['type1']).median()[against_columns]
plt.figure(figsize=(18, 8))
effect = sns.heatmap(effectiveness,
                     annot=True,
                     fmt=".1f",
                     vmin=0,
                     vmax=4,
                     cmap='Set2_r')
plt.xticks(rotation=60)
effect.set_title('Effectiveness Type Chart', loc='left', pad=20);
```



Conclusion: *To highlight that electric is only weak against (against>=2) ground and normal is only weak against fighting. It is curious that ghost is weak against ghost and that flying is extremely weak against ice. There are some types that are pretty useless against (against=0):psychic against dark,dragon against fairy,ground against* *flying,fighting against ghost,normal against ghost,ghost against normal,electric against ground,poison against steel.*

Plotting Height based on Pokémon-Type:

There are 18 types of Pokémon, so to plot the height of each type separately, we have stored it in a dictionary.

```
heights = {}
for i in range(18):
    heights['height_'+pokemon_types[i]]= []

j = 0
for i in heights:
    heights[i] = pokemon_df.loc[pokemon_df['type1']==pokemon_types[j], "height_m"]
    j = j + 1

print(heights)
```

Explanation: First we initialize a dictionary 'heights' to store height separated by their type. Then we run a for loop 18 times to add keys to the dictionary with relevant names e.g., 'height_grass' and empty list as values.

In the second for loop, we iterate on the dictionary and add height to the list when type1 == pokemon_types[j](here j is a counter which starts from 0 and gets incremented after each loop).

Output:

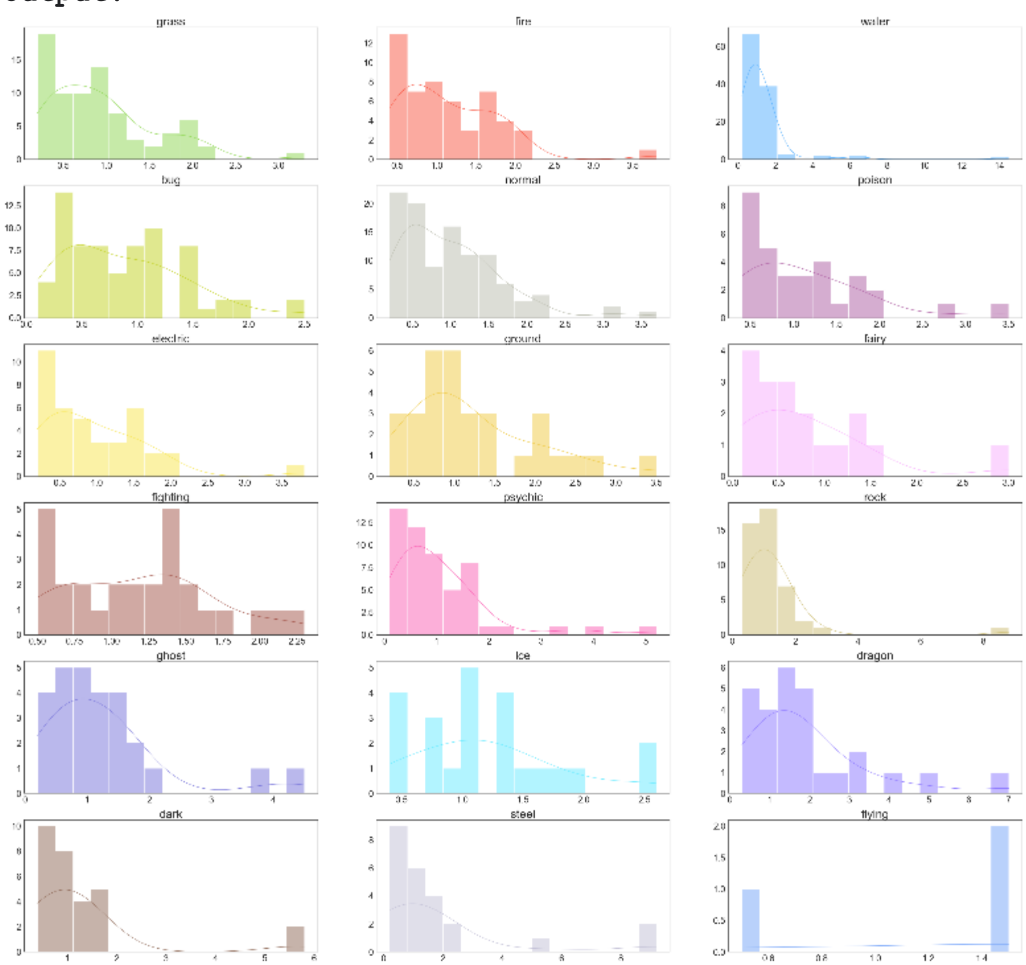
```
{'height_grass': 0      0.7
1      1.0
2      2.0
42     0.5
43     0.8
      ...
760    0.3
761    0.7
762    1.2
786    1.9
797    0.3
Name: height_m, Length: 78, dtype: float64, 'height_fire': 3      0.6
4      1.1
5      1.7
36     0.6
37     1.1
```

Now we have plotted subplots of height for each type of pokemon.

```
plt.subplots(figsize=(60,57))
sns.set(rc={'xtick.labelsize': 30, 'ytick.labelsize': 30, 'axes.titlesize': 40}, style='white')
j = 0
for i in heights:
    plt.subplot(6,3,j+1)
    sns.histplot(heights[i], bins=15 , color=pokemon_colors[pokemon_types[j]], kde= True).set(title=pokemon_types[j],
     xlabel=None,ylabel=None)
    j = j + 1
```

Explanation: In the above code we again we ran a for loop to plot histograms in a 6x3 grid.
Subplots(figsize=): used to determine the size of the plot.
Set(): used to set the size of ticks and labels and to select the style of the plot.
Subplot(r,c,p): used to determine the position of the plot in the grid, here r is no. of rows c is no. of columns and p is the position of the plot (like 1,2,3,4,5……).
Histplot(): used to create histogram, bins are the number of bin we want and kde is kernel density estimate plot is a method for visualizing the distribution of observations in a dataset, analogous to a histogram.

Output:

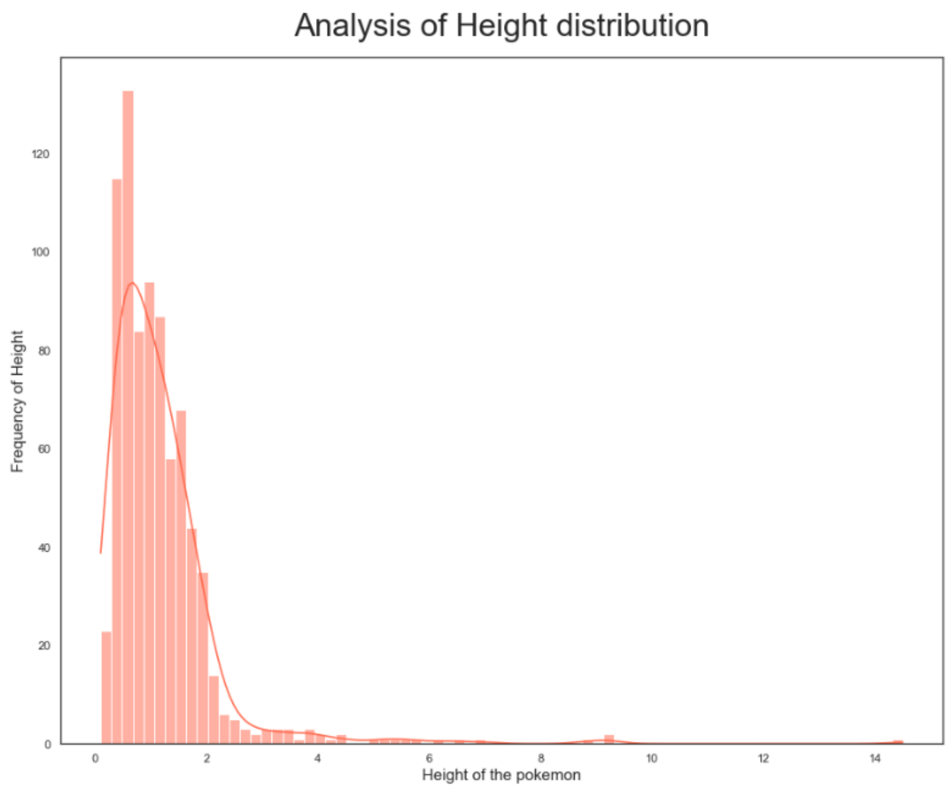


Here we can most of the plots have right skewness, while flying type have only 2 bins which means their height distribution is either less than 0.6m or more than 1.4m.

Now we plot a histogram of all the types together:

```
sns.set(rc={'xtick.labelsize': 11, 'ytick.labelsize': 11, 'axes.titlesize': 15, "axes.grid":False}, style='white')
plt.figure(figsize=(15,12))
sns.histplot(pokemon_df,x='height_m', color='#FF6347', kde=True )
plt.xlabel("Height of the pokemon",fontsize=15)
plt.ylabel("Frequency of Height",fontsize=15)
plt.title("Analysis of Height distribution",fontsize=30, pad=20)
```

Output:



From the above plot we can see it is heavily skewed on the right this is because some Pokémon are very tall compared to the rest. We have described the height attribute for further insights:

```
pokemon_df.height_m.describe()
```

```
count    801.000000
mean       1.155556
std        1.069952
min        0.100000
25%        0.600000
50%        1.000000
75%        1.500000
max       14.500000
Name: height_m, dtype: float64
```

From this we can see that mean height is more than median height due to right skewness, and maximum height is 14.5m.

Plotting Weight based on Pokémon-Type:

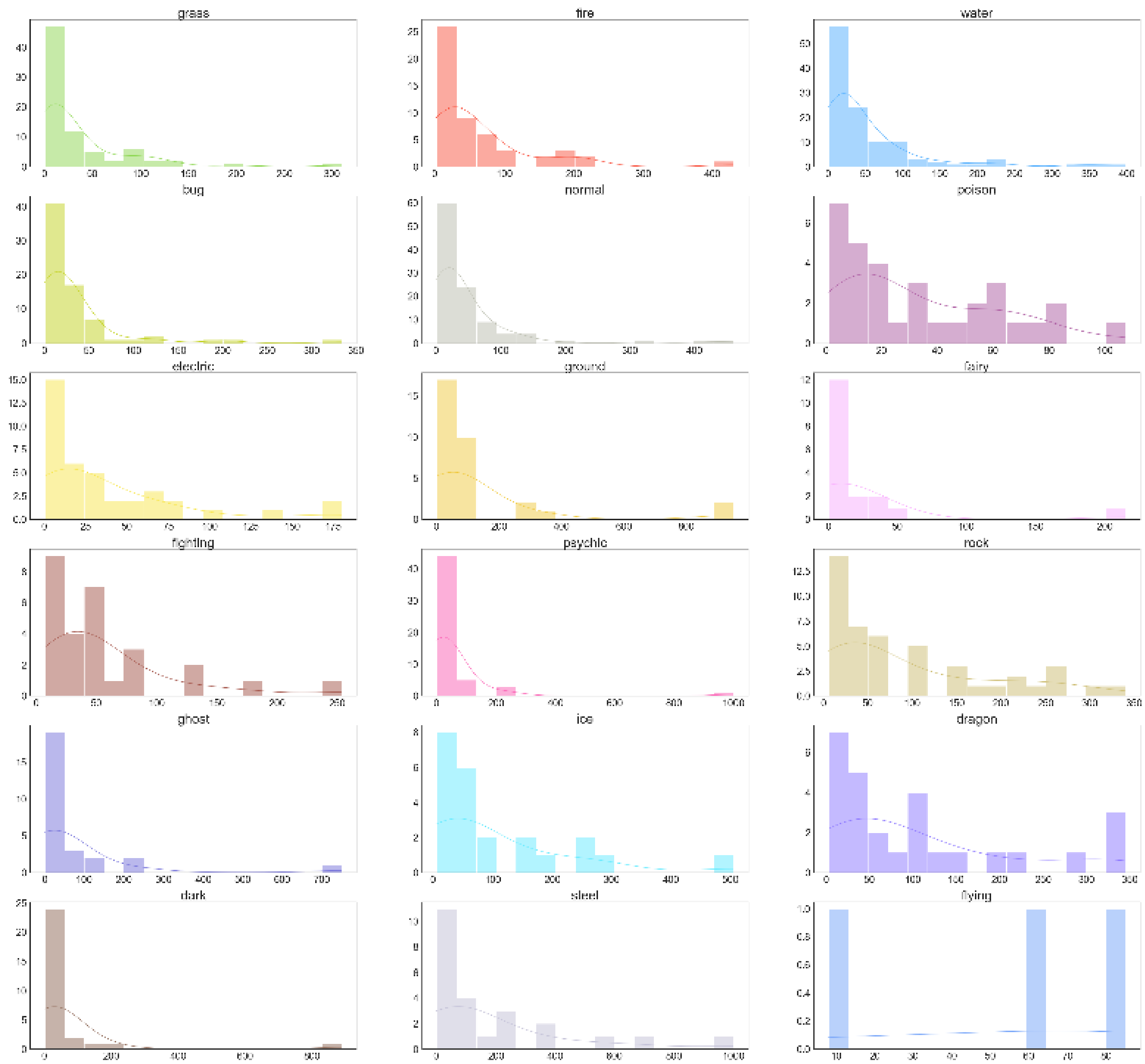
Just like height attribute we have used methods to plot weight of the Pokémons based on their type. So, we can directly talk about the graphs.

```
weights = {}
for i in range(18):
    weights['weight_'+pokemon_types[i]]= []

j = 0
for i in weights:
    weights[i] = pokemon_df.loc[pokemon_df['type1']==pokemon_types[j], "weight_kg"]
    j = j + 1

print(weights)
```

```
637    250.0
678      2.0
679      4.5
680     53.0
706      3.0
796    999.9
800     80.5
Name: weight_kg, dtype: float64, 'weight_flying': 640    63.0
713      8.0
714     85.0
Name: weight_kg, dtype: float64}
```

```python
plt.subplots(figsize=(60,57))
sns.set(rc={'xtick.labelsize': 30, 'ytick.labelsize': 30, 'axes.titlesize': 40}, style='white')
j = 0
for i in weights:
    plt.subplot(6,3,j+1)
    sns.histplot(weights[i], bins=15 , color=pokemon_colors[pokemon_types[j]], kde=True).set(title=pokemon_types[j],xlabel=None,ylabel=None)
    j = j + 1
```

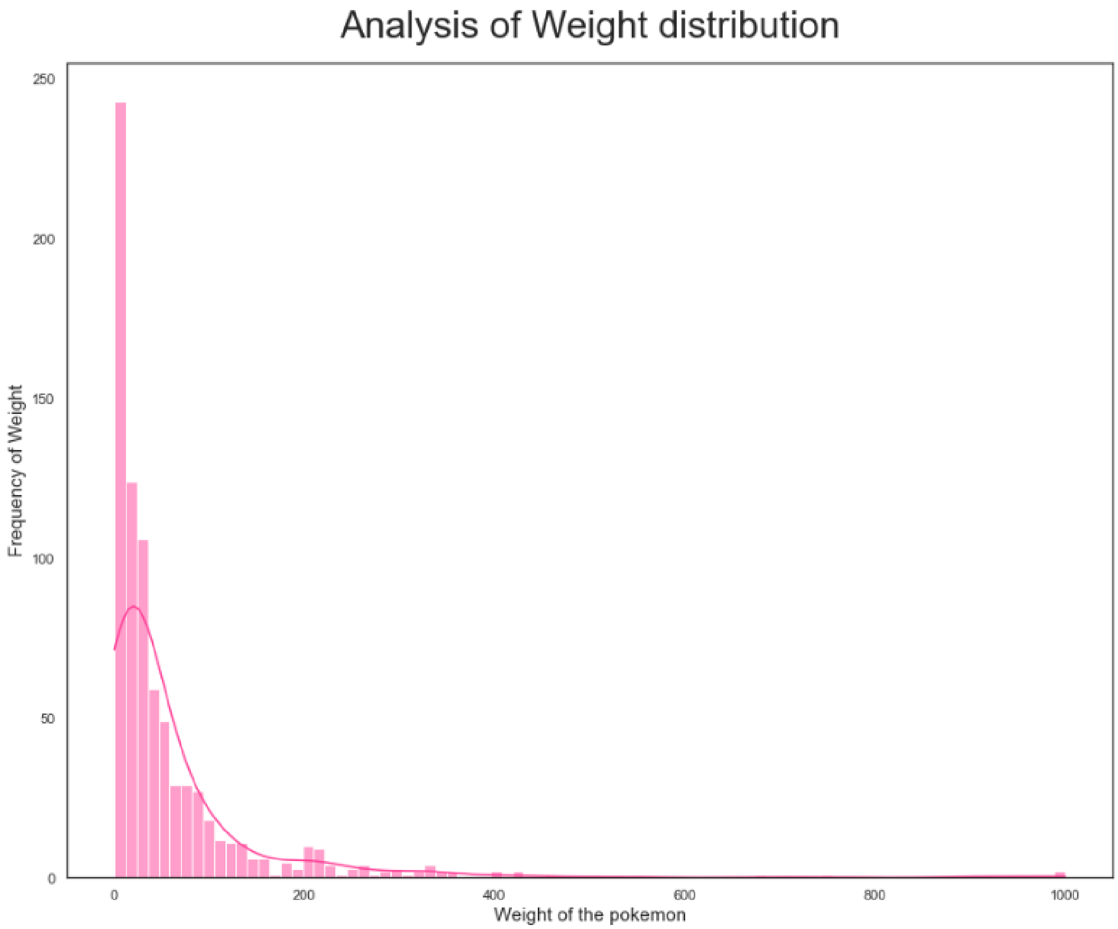
Output:



Just like height, weight of most of the types is rightly skewed while flying type have somewhat isolated values.

```python
sns.set(rc={'xtick.labelsize': 11, 'ytick.labelsize': 11, 'axes.titlesize': 15, "axes.grid": False}, style='white')
plt.figure(figsize=(15,12))
sns.histplot(pokemon_df, x='weight_kg', color='#FF3E96', kde=True )
plt.xlabel("Weight of the pokemon",fontsize=15)
plt.ylabel("Frequency of Weight",fontsize=15)
plt.title("Analysis of Weight distribution",fontsize=30, pad=20)
```

Output:



From the above plot we can see that the graph is skewed on right. Using describe function to get more insights:

```
pokemon_df.weight_kg.describe()
```

```
count    801.000000
mean      60.941199
std      108.514597
min        0.100000
25%        9.000000
50%       27.300000
75%       63.000000
max      999.900000
Name: weight_kg, dtype: float64
```

From this we can see that mean weight is 60.9kg which median weight is 27.3kg. Whereas the maximum weight is 999.9kg and minimum is 100g.

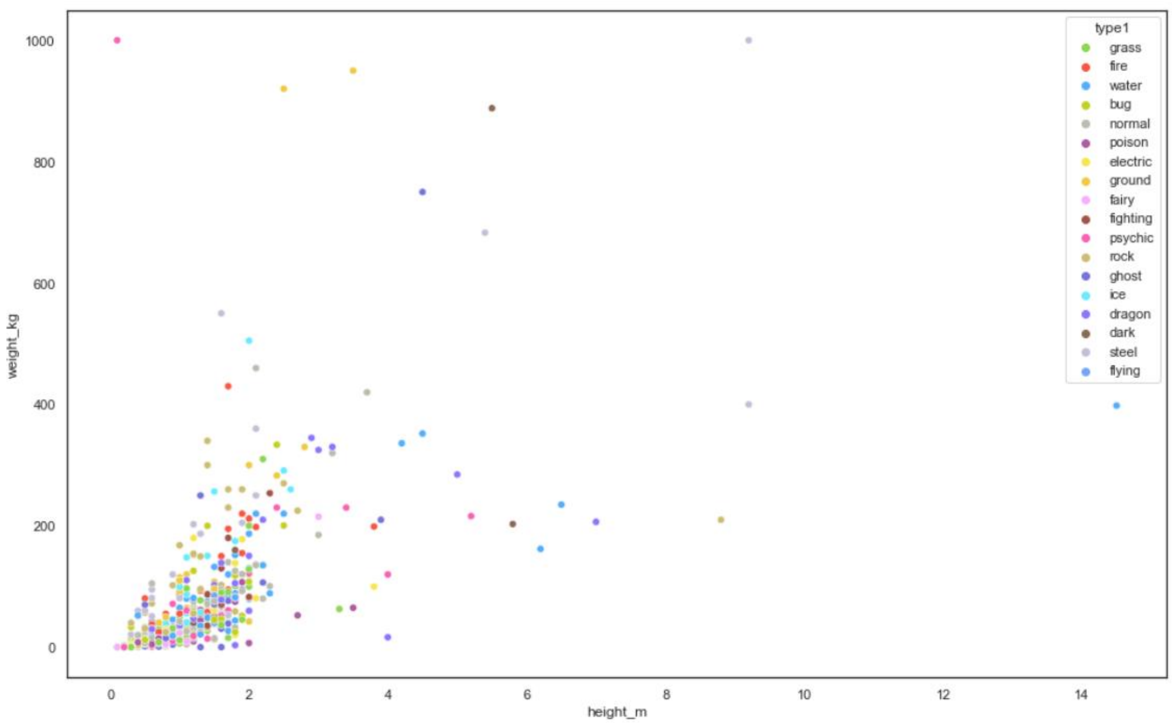
Correlation between height and weight:

Now to find out how the height and weight of the Pokémons are correlated to each other we plotted a scatterplot because height and weight both are numeric variables. To plot the scatter plot, we used the following code:

```
sns.set(rc={'xtick.labelsize': 11, 'ytick.labelsize': 11, 'axes.titlesize': 15, "axes.grid":False}, style='white')
plt.figure(figsize=(16,10))
sns.scatterplot(data=pokemon_df, x='height_m', y='weight_kg', hue='type1', palette=pokemon_colors)
```

Scatterplot(): used to plot scatter plot, hue is used to categories based on the type1 of Pokémon, palette is used for colours.

Output:



From the above plot we can see that height and weight of Pokémons are loosely correlated. We can tell this by seeing that the tallest Pokémon is not the heaviest and vice versa, the reason for this can be their types. Now to get the top 5 heaviest and tallest Pokémon:

Sorted_values(): For returning the values in a sorted order, ascending is False so we can get values in descending order.

Head(x): It is used to return top x number of rows from the data frame.

```python
top5_height = pokemon_df[['height_m','name','type1']].sort_values('height_m', ascending=False).head(5)
print(top5_height)

top5_weight = pokemon_df[['weight_kg','name','type1']].sort_values('weight_kg', ascending=False).head(5)
print(top5_weight)
```

```
     height_m        name    type1
320      14.5     Wailord    water
207       9.2     Steelix    steel
796       9.2  Celesteela    steel
94        8.8        Onix     rock
383       7.0    Rayquaza   dragon
     weight_kg        name    type1
789      999.9     Cosmoem  psychic
796      999.9  Celesteela    steel
382      950.0     Groudon   ground
749      920.0    Mudsdale   ground
798      888.0    Guzzlord     dark
```

BMI of Pokémon:

After we know the correlation between Height and weight, another useful information we can obtain is the BMI(body mass index) of Pokémons. So, we start by creating a BMI column in the data frame.

```python
pokemon_df['BMI'] = pokemon_df.apply(lambda x: x['weight_kg']/(x['height_m']**2), axis=1)
```

In the above code we have used lambda function to fill values of BMI.

Now that we have BMI attribute, we can easily make a bar plot for top10 Pokémon with largest BMI.

```python
top10_bmi = pokemon_df.nlargest(10,'BMI')
plt.figure(figsize=(19,10))
sns.barplot(y=top10_bmi['name'], x=top10_bmi['BMI'], color='darkslateblue')

for index,bmi in enumerate(top10_bmi.BMI):
    plt.annotate(f'{bmi:.2f}', xy=((bmi+2500),index),horizontalalignment='center', verticalalignment='center')
```

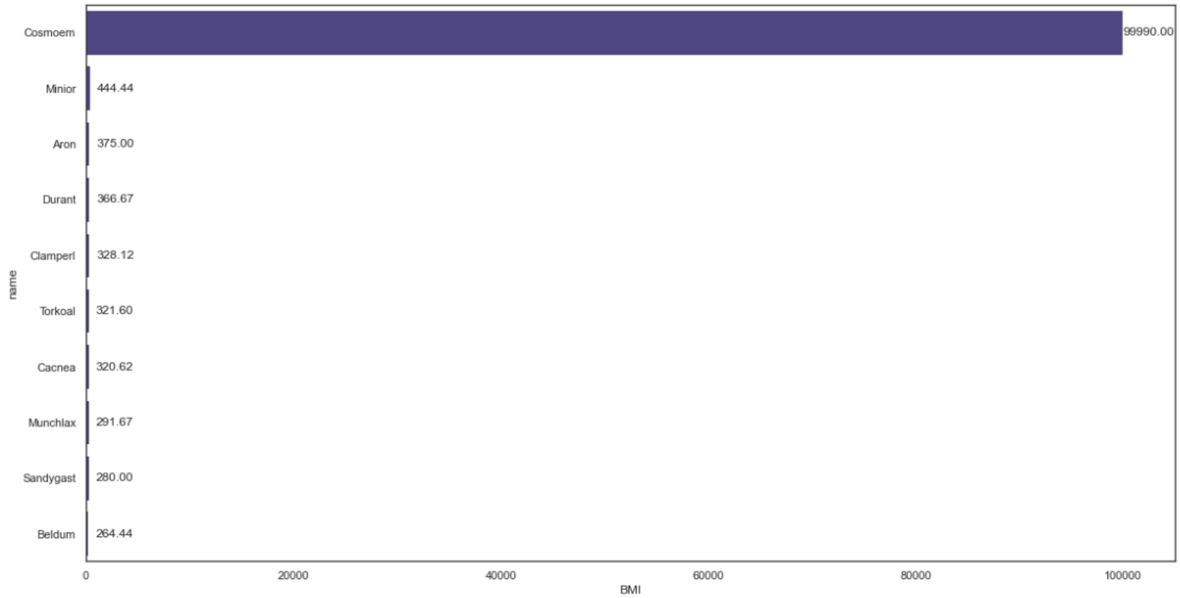
Explanation:

Barplot(): used to create barplot, y is y axis and x is axis.

Enumerate(): used to create a list with tuples in (index,bmi) format.

Annotate(): used to add comments on the plot for better understanding.

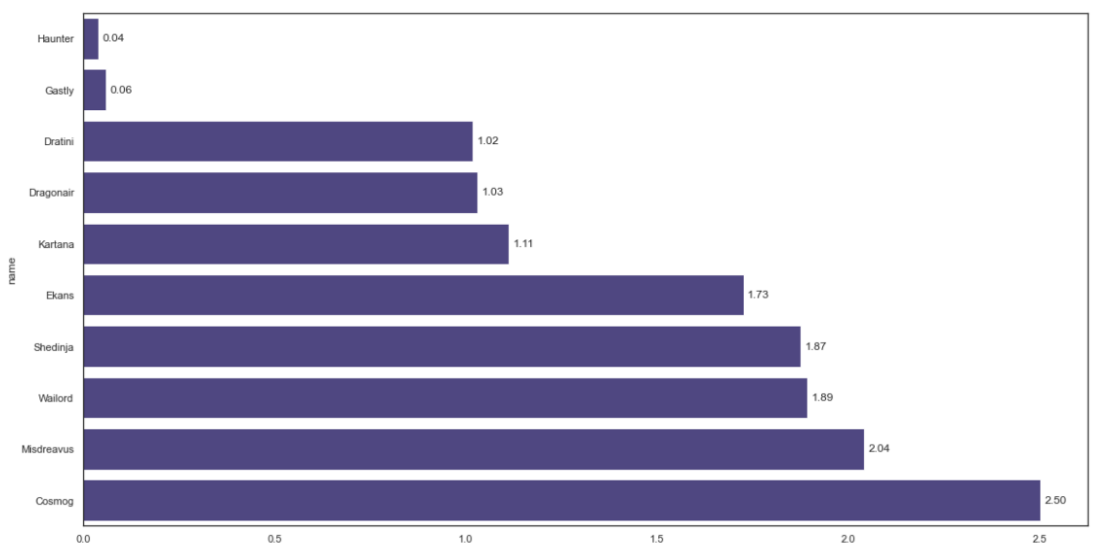
Output:



From the above plot we can see that Cosmoem have the highest BMI, and no one is even close to it.

Similarly, we can plot for least BMI among pokemons.

```
bottom10_bmi = pokemon_df.nsmallest(10,'BMI')
plt.figure(figsize=(19,10))
sns.barplot(y=bottom10_bmi['name'], x=bottom10_bmi['BMI'], color='darkslateblue')

for index,bmi in enumerate(bottom10_bmi.BMI):
    plt.annotate(f'{bmi:.2f}', xy=((bmi+0.04),index),horizontalalignment='center', verticalalignment='center')
```

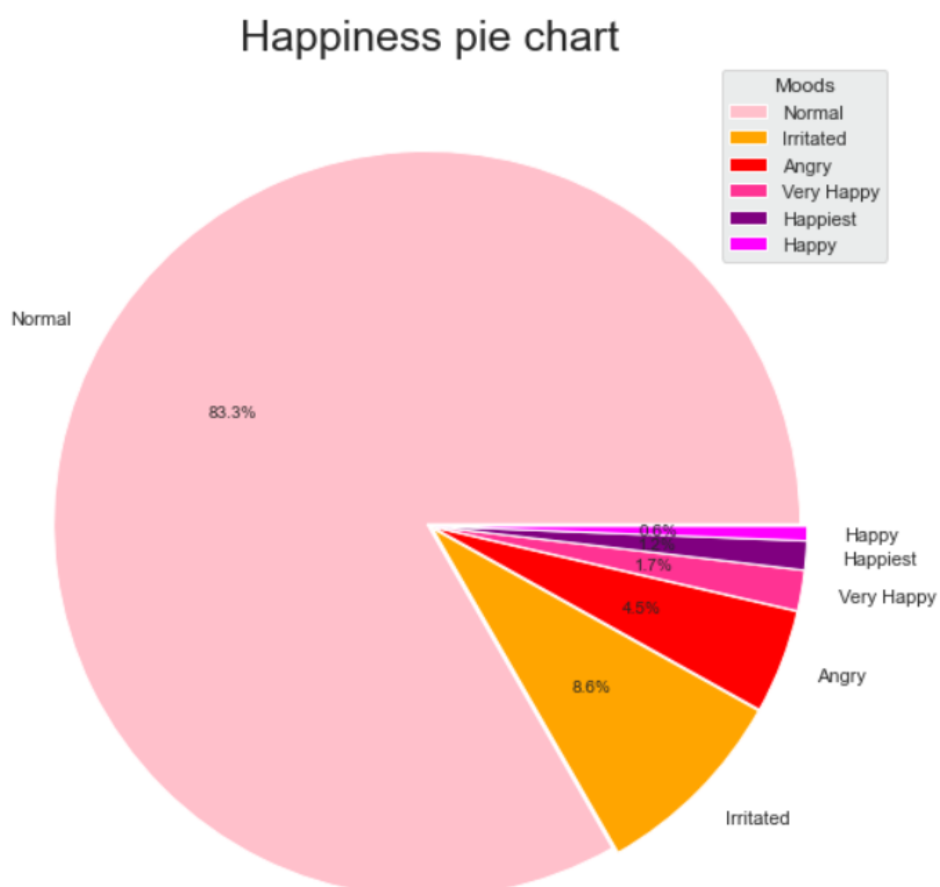


Base Happiness of Pokémon:

Now to analyse base happiness of the Pokémon we observed that it is taking only six distinct values therefore a pie chart would be the most appropriate plot for the analysis. Since base happiness was given is numeric values we categorised it in different moods for better understanding i.e., angry, irritated, normal, happy, very happy and happiest.

```
happiness = pokemon_df['base_happiness'].value_counts()
sns.set(rc={"axes.titlesize":25, "font.size": 10})
plt.figure(figsize=(10, 10))
plt.pie(happiness, colors=['pink', 'orange', 'red','#FF3393','purple', 'magenta'], labels=["Normal", "Irritated",
 "Angry", "Very Happy", "Happiest", "Happy"], autopct='%1.1f%%', explode=[0.01, 0.01, 0.01, 0.01, 0.01, 0.01] )
plt.title("Happiness pie chart")
plt.legend(title="Moods", facecolor="#E5E8E8")
```

Pie(): used for plotting pie charts, labels is used to name each portion of the pie, autopct is used for display percentage and explode is used for exploding effect in the pie chart.



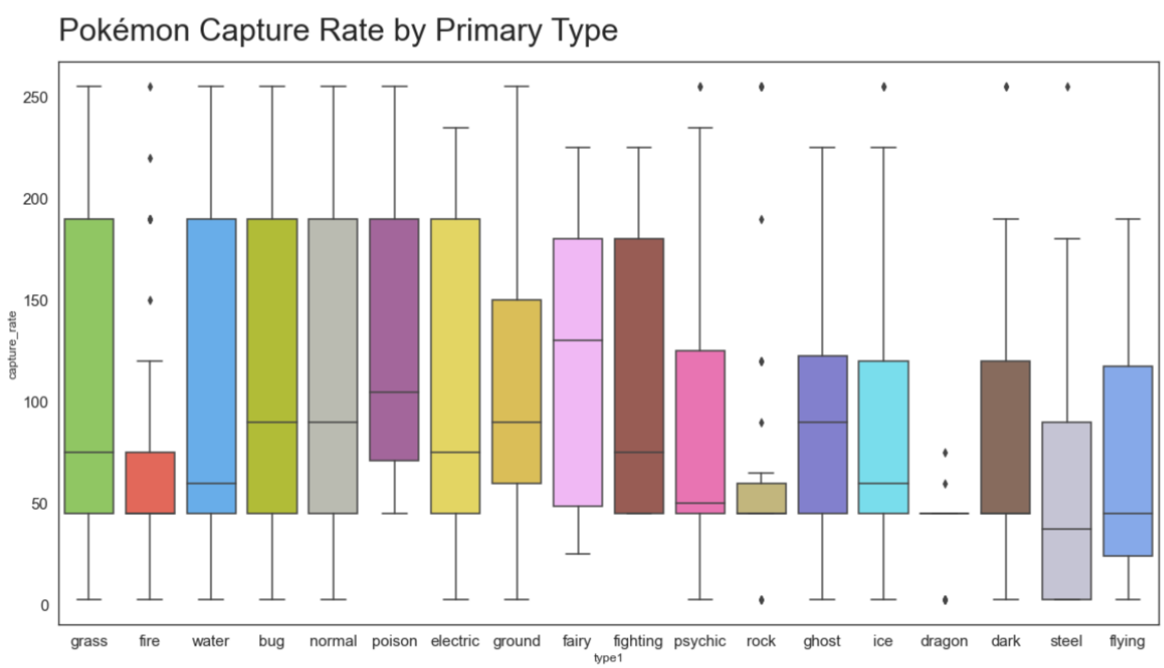
From the above pie chart, we can see most of the Pokémons are in the Normal mood that is 83.3%, followed by 8.6% which are irritated and 4.5% which are angry. So can conclude that if u encounter Pokémon there is 13.1% chance you are getting attacked.

Capture Rate based on Type:

Now to find out how easy it is to capture a Pokémon based on its type we performed an analysis using box plots:

```
pokemon_df['capture_rate'].replace({'30 (Meteorite)255 (Core)': np.nan}, inplace=True)
pokemon_df['capture_rate'] = pd.to_numeric(pokemon_df['capture_rate'])
plt.figure(figsize=(19,10))
sns.set(rc={'xtick.labelsize': 15, 'ytick.labelsize': 15, 'axes.titlesize': 30, "axes.grid":False}, style='white')
sns.boxplot(x='type1',y='capture_rate', data = pokemon_df, palette=pokemon_colors)
plt.title('Pokémon Capture Rate by Primary Type', loc='left', pad=20)
```

In the above code we had to change the datatype of the column to numeric and remove a value to perform the analysis.
Boxplot(): used to plot boxplots.



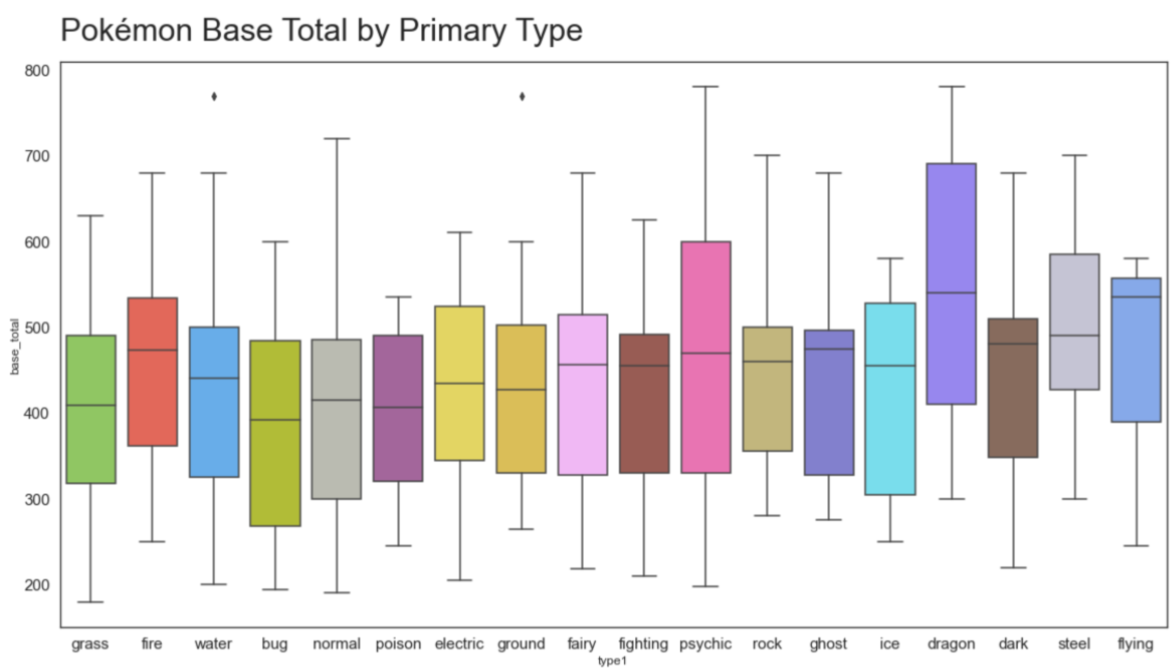
From the above plot we can see that dragon type is hardest to capture whereas fairy type is easiest to capture.

Base Total based on Types:

Now to find out does the type affect base total stats or which type has highest base total stats we performed analysis on base total attribute using boxplot based on their primary types.

```
sns.set(rc={'xtick.labelsize': 15, 'ytick.labelsize': 15, 'axes.titlesize': 30, "axes.grid":False}, style='white')
plt.figure(figsize=(19,10))
sns.boxplot(x='type1',y='base_total', data = pokemon_df, palette=pokemon_colors)
plt.title('Pokémon Base Total by Primary Type', loc='left', pad=20)
```

Code is same as we used in previous analysis.

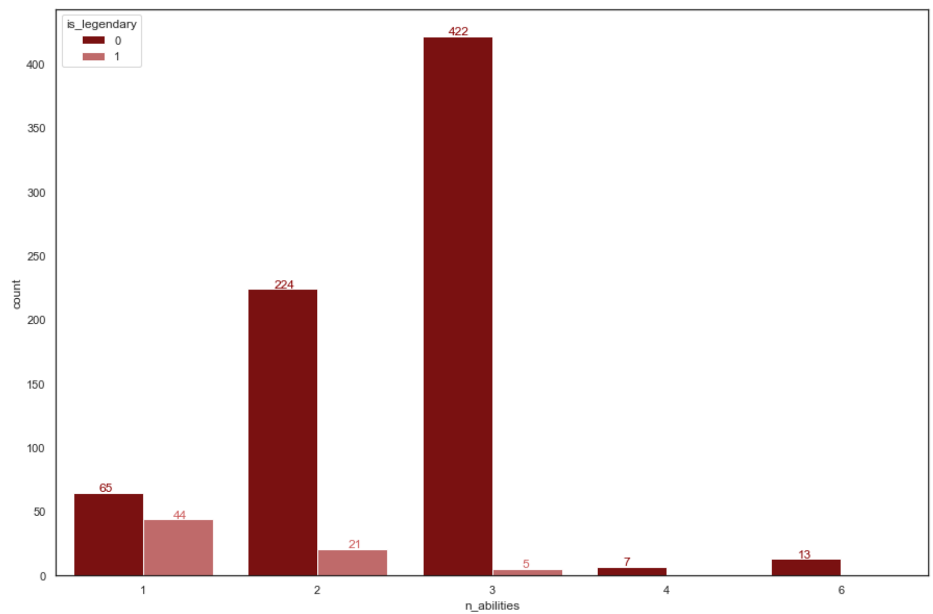


From the above plot we can see that dragon type have the highest base total stats this explains why they are so hard to catch. Whereas Bug type have lowest base total stats. From this we can conclude that base total stats are definitely related to primary type of Pokémon.

Abilities Countplot:

```
pokemon_df['abilities'] = pokemon_df.apply(lambda x: literal_eval(x['abilities']), axis=1)
pokemon_df['n_abilities'] = pokemon_df.apply(lambda x: len(x['abilities']), axis=1)
```

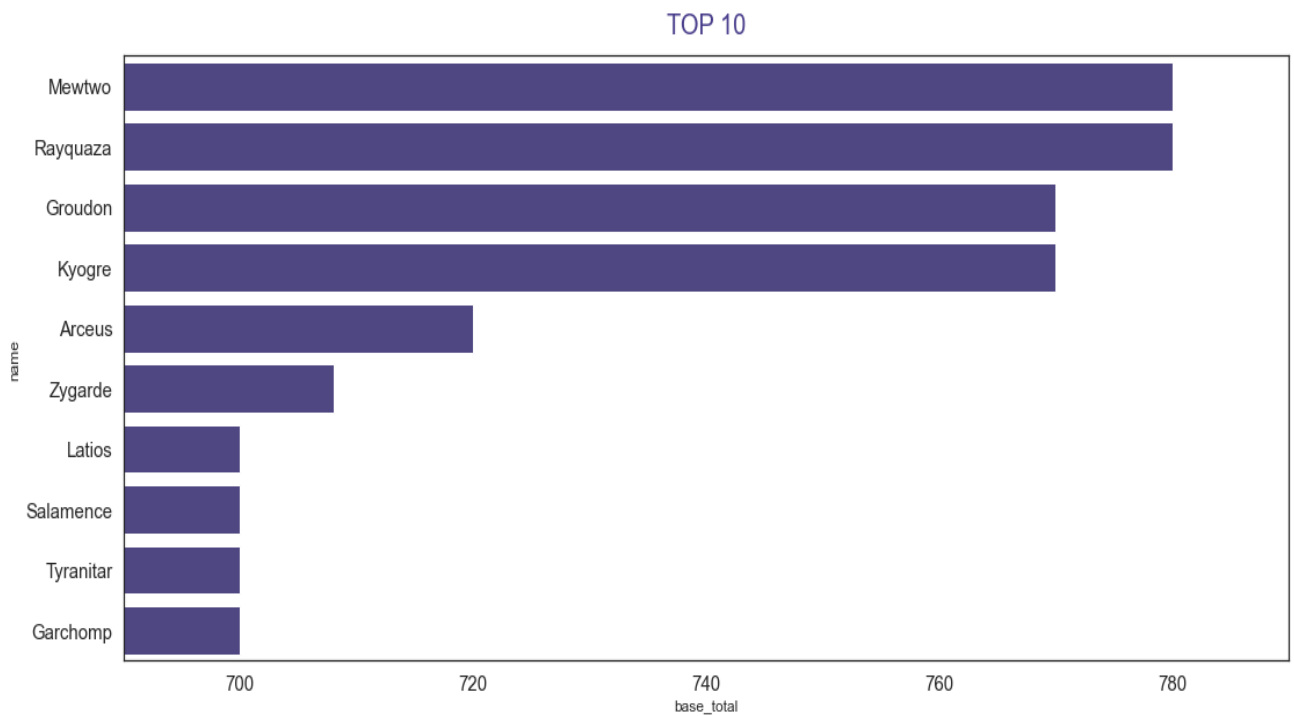
```
plt.figure(figsize=(15,10))
sns.countplot(data=pokemon_df, x='n_abilities', hue='is_legendary', palette=['darkred', 'indianred'])

for index, value in enumerate(pokemon_df[pokemon_df['is_legendary'] == False].n_abilities.value_counts().sort_index()):
    plt.annotate(f'{value}', xy=(index - 0.25, value + 1), color='darkred')

for index, value in enumerate(pokemon_df[pokemon_df['is_legendary'] == True].n_abilities.value_counts().sort_index()):
    plt.annotate(f'{value}', xy=(index + 0.175, value + 1), color='indianred')
```



Conclusion: *Many non-legendary Pokemon have 3 as the number of abilities. Whereas most legendary Pokemon have just one ability.*

Top 10 Best Powerful Pokémon

```
top10_pokemon_base_total = pokemon_df.sort_values(by='base_total', ascending=False).reset_index()[:10]
plt.figure(figsize=(18,8))
ax = sns.barplot(y=top10_pokemon_base_total['name'],
                 x=top10_pokemon_base_total['base_total'],
                 orient='h',
                 color='darkslateblue')

plt.xlim(690, 790)

plt.title('TOP 10', color='darkslateblue', fontsize=20, pad=16)
```

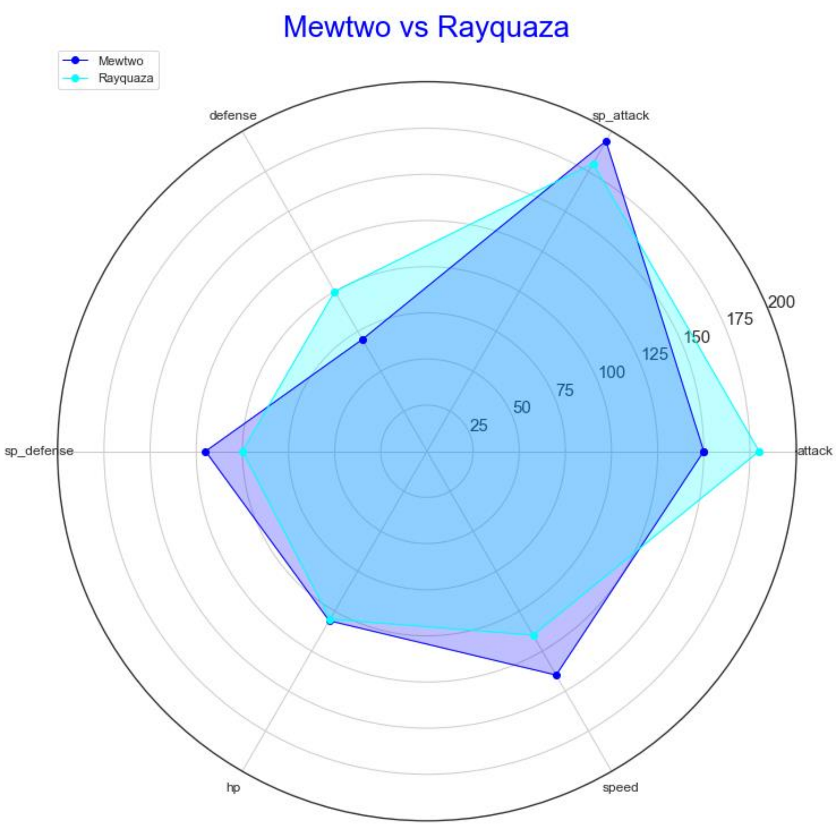


Conclusion: *Mewtwo and Rayquaza have the same base total. So, it's a tie between the two.*

RADAR CHART:

```
attributes=['attack', 'sp_attack', 'defense', 'sp_defense', 'hp', 'speed']
mewtwo=top10_pokemon_base_total[top10_pokemon_base_total['name']=='Mewtwo'][attributes].values.tolist()[0]
rayquaza=top10_pokemon_base_total[top10_pokemon_base_total['name']=='Rayquaza'][attributes].values.tolist()[0]
```

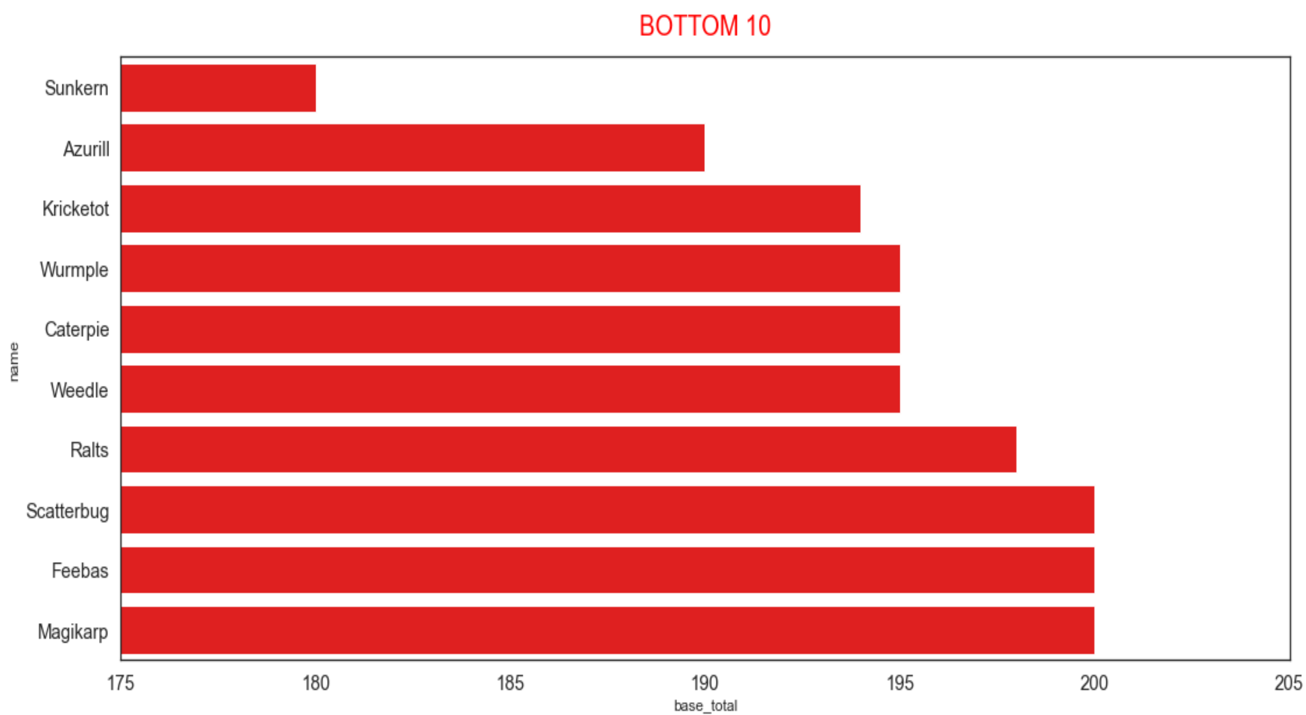
```
angles=np.linspace(0,2*np.pi,len(attributes), endpoint=False)
angles=np.concatenate((angles,[angles[0]]))
attributes.append(attributes[0])
mewtwo.append(mewtwo[0])
rayquaza.append(rayquaza[0])

fig=plt.figure(figsize=(18,12))
rc=fig.add_subplot(111, polar=True)

rc.plot(angles, mewtwo, 'o-', color='blue', linewidth=1, label='Mewtwo')
rc.fill(angles, mewtwo, alpha=0.25, color='blue')

rc.plot(angles,rayquaza, 'o-', color='cyan', linewidth=1, label='Rayquaza')
rc.fill(angles, rayquaza, alpha=0.25, color='cyan')

rc.set_thetagrids(angles[:-1] * 180/np.pi, attributes[:-1], fontsize=12)
plt.grid(True)

handles, labels = rc.get_legend_handles_labels()
rc.legend(handles, ['Mewtwo', 'Rayquaza'], loc=(0,0.99))

rc.set_title("Mewtwo vs Rayquaza", pad=40, fontsize=26, color='blue')
```



Conclusion: *Rayquaza has better attack and defense compared to Mewtwo. While MEwtwo has better HP, speed, sp_attack and sp_defence than Rayquaza.*

Top 10 Least Powerful Pokémon

```
bottom10_pokemon_base_total = pokemon_df.sort_values(by='base_total', ascending=True).reset_index()[:10]
plt.figure(figsize=(18,8))
ax = sns.barplot(y=bottom10_pokemon_base_total['name'],
                 x=bottom10_pokemon_base_total['base_total'],
                 orient='h',
                 color='red')

plt.xlim(175, 205)

plt.title('BOTTOM 10', color='red', fontsize=20, pad=16)
```

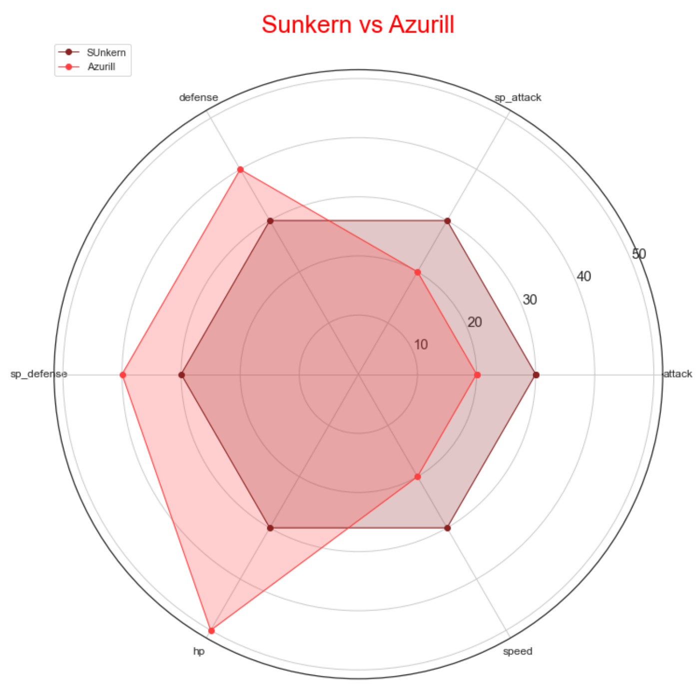


Conclusion: *Sunkern is the Pokemon with the lowest base total and hence is the least powerful Pokemon of all, followed by Azurill.*

RADAR CHART:

```
attributes=['attack', 'sp_attack', 'defense', 'sp_defense', 'hp', 'speed']
Sunkern=bottom10_pokemon_base_total[bottom10_pokemon_base_total['name']=='Sunkern'][attributes].values.tolist()[0]
Azurill=bottom10_pokemon_base_total[bottom10_pokemon_base_total['name']=='Azurill'][attributes].values.tolist()[0]
```

```
angles=np.linspace(0,2*np.pi,len(attributes), endpoint=False)
angles=np.concatenate((angles,[angles[0]]))
attributes.append(attributes[0])
Sunkern.append(Sunkern[0])
Azurill.append(Azurill[0])

fig=plt.figure(figsize=(18,12))
rc_=fig.add_subplot(111, polar=True)

rc_.plot(angles, Sunkern, 'o-', color='#8B2323', linewidth=1, label='Sunkern')
rc_.fill(angles, Sunkern, alpha=0.25, color='#8B2323')

rc_.plot(angles, Azurill, 'o-', color='#FF4040', linewidth=1, label='Azurill')
rc_.fill(angles, Azurill, alpha=0.25, color='#FF4040')

rc_.set_thetagrids(angles[:-1] * 180/np.pi, attributes[:-1], fontsize=12)
plt.grid(True)

handles, labels = rc_.get_legend_handles_labels()
rc_.legend(handles, ['SUnkern', 'Azurill'], loc=(0,0.99))

rc_.set_title('Sunkern vs Azurill', pad=40, fontsize=26, color='red')
```



Conclusion:*Sunkern has more attack power and speed while Azurill has better defense and HP. Here HP is determining factor therefore Sunkern is the weakest one.*

Overall Conclusion:

- Odd generations have more pokemon than even ones. 5th has the most.
- Height and weight are correlated but relation is not strong.
- Fairy type pokemons are easiest to capture while dragon type are the hardest.
- We concluded that attack and defense are strongly correlated and defense and speed has weak negative correlation.
- In the end we concluded Mewtwo is the strongest pokemon and Sunkern is the weakest.

Things we got to learn from the project:

-Python is one of the most popular languages in the world. There are a lot of reasons why Python is popular among developers and one of them is that it has an amazingly

large collection of libraries that users can work with. Python libraries are a great way to data analysis and machine learning. They provide powerful functionality and

flexibility for any task, regardless of the type of data. Python libraries make it easy for developers and data scientists to prototype and scale their models, regardless

of their size or complexity.

-In this project, we got to learn a lot about some Python libraries, namely, matplotlib, pandas, NumPy, seaborn. Using matplotlib, we came to learn about data visualization.

Seaborn, which builds on top of matplotlib, was useful for making statistical graphics like bar plots, pie charts, box plots, histogram. NumPy helped us in performing various

kinds of mathematical operations on our dataset. Pandas was useful in creating data frames to perform numerous operations and analysis on them.

-Performing EDA helped us to examine our dataset and discover its underlying structure. It helped us in comprehending the relationship between the variables, providing us with

a broader view of the data. As mentioned above, we drew some reliable conclusions from the dataset that gave us a better understanding of the data that we are working with.

Acknowledgement:

We would like to express our profound gratitutde to Nishith Sir and Mayank Sir for their time and efforts throughout the semester. Their useful advices and suggestions were really helpful to us during the project's completion.

We'd also like to thank our friends for their valuable assistance as we worked on this project.

Also, this project would not have been completed without the contributions of each and every individual in the team.

References:

- Dataset from https://www.kaggle.com/datasets/rounakbanik/pokemon

- Pokemon Data Analysis Tutorial. Available at https://www.kaggle.com/code/mmetter/pokemon-data-analysis-tutorial

- Pokemon EDA. Available at https://www.kaggle.com/code/yassinealouini/pokemon-eda/data

- Radar Chart. Available at https://www.python-graph-gallery.com/radar-chart/

- How to create radar charts in Python. Available at https://towardsdatascience.com/how-to-create-a-radar-chart-in-python-36b9ebaa7a64

- Treemaps in Python using Squarify. Available at https://www.geeksforgeeks.org/treemaps-in-python-using-squarify/

- Python color constants module. Available at https://www.webucator.com/article/python-color-constants-module/

Individual Contribution in the Project:

- We were working with 35 attributes(features) of the Pokemon out of which 15 were of utmost importance. So, to make things easier, we divided our work based on these attributes. Every three of us was responsible for handling mainly 5 attributes each. Every individual's task was to clean and pre-process the data of their own attributes and to perform various kinds of analysis on them. But we were not restricted to this only. As a team, we came together to perform analysis involving assigned attributes of more than one individual.

- Though we had divided the work among us but we were in constant touch during the entire process and we learned and grew together.

- Attributes covered:

Anisha: Attack, defense, speed, hp, correlation between attributes, median of attributes, Effectiveness against types

Ayush: Height, Weight, Correlation between Height and weight, BMI, Capture Rate, Base Total Stats, Base Happiness.

Priyanka: Primary types, Secondary Types, Legendary, Generation, Strongest and weakest 10, Abilities, strongest and weakest pokemon.