FinalProject_group156

June 10, 2019

1 Scientifically Differentiating Legendary Pokémon from the Rest

Group Number: 156
Team Name: Gotta Catch 'Em All
Team Members:

- Ryan Keng (A12956059)
- Steven Khai (A14077103)
- Bradley Tran (A14694034)
- Elliot Yoon (A12977251)
- Adam Yu (A13468697)
- Ricky Zhao (A12037809)

2 Overview

Our goal was to summarize the aspects that differentiate legendary Pokemon from the standard fare. We compared physical attributes such as size, height, and typing, as well as base combat stats and usage in the competitive scene. We compared the distribution for these aspects in legendary Pokemon versus all Pokemon and ascertained whether or not legendary Pokemon followed the same trends as the rest of the Pokemon.

3 Research Question

Legendary Pokémon are some of the most revered Pokémon in the Pokémon universe. But what sets them apart from regular Pokémon? Is it their stats? Their types? Do players use legendary Pokémon more over other Pokémon due to these attributes? According to Bulbapedia, one of the most extensive Pokémon encyclopedias, the only thing that distinguishes legendaries from non-legendaries is a naming convention. If Nintendo decides a certain Pokémon should be legendary, then it is. However we believe there has to be more to it than that. For our project, we will examine various statistics regarding Pokémon and see if we can scientifically deduce what sets legendary Pokémon apart from the rest.

4 Background and Prior Work

Over two decades old and a household icon worldwide, the world of Pokémon surpassed 800 unique characters over seven generations with another already announced for later this year. Each Pokémon has unique stats, types, moveset, and design - making every new Pokémon you encounter a brand new sight to behold. But not all Pokémon are created equal - within each generation, there are several designated "legendary Pokémon" that are set a tier above the average Pokémon you might encounter while playing the game. These special Pokémon are set apart through varying combinations of the aforementioned features, and are often the headliners for each generation of Pokémon released.

But what defines a legendary is not always clear nor consistent. For example some legendaries do not show up on the cover of the game box, such as in the first generation. Starter Pokémon, which is the first Pokémon the player ever sees, are sometimes highlighted instead of Pokémon in the pool of legendaries. For statistics, certain Pokémon see incredibly high usage in the competitive scene while some legendaries are not even worth considering.

Most of the research regarding legendary Pokémon focuses solely on the lore and the constraints of the legendary Pokémon themselves. However, not much research has been done analyzing the numbers that surround legendary Pokémon. There has been, however, many attempts of looking at Pokémon data as a whole, and analyzing the data that way. For example, the following two scatter plots closely mirror the type of analysis we will be performing in our project.

https://www.reddit.com/r/dataisbeautiful/comments/780wfi/

The first scatter plot, created by Reddit user jmerlinb, compares Pokémon in terms of their height and weight. This is a visualization we drew a lot of inspiration from, but we will make it easier to read, as well as distinguish legendary Pokémon from non-legendary Pokémon in order to prove our hypothesis.

https://www.reddit.com/r/dataisbeautiful/comments/91u20t/

5 Hypothesis

From our knowledge of the games and franchise, we can deduce possible aspects that categorizes these Pokémon apart. We know that Legendary Pokémon are considered special, and as such they probably have special features that set them apart from normal Pokémon. We decided to examine this question by looking at two main factors: their capabilities and physical composition. Their capabilities can be defined in two ways. One of these ways is their stats. Every Pokémon have base stats that are universal across all Pokémon of a particular species, we believe that Legendary Pokémon will have higher base stats than all of the other non-Legendary Pokemon. For their physical composition, we believe that Legendary Pokémon are predominantly of the Dragon-type. Also regarding their actual size, we believe that Legendary Pokémon exist in the extremes of the Pokémon world - meaning that they are either extremely large, or extremely small. We believe that these attributes will also mean that their competitive usage is notably higher than normal Pokémon. We came to this hypothesis because we believed that as the headliner Pokemon for every generation, Nintendo would want these Pokemon to stand out amongst the other 100+ Pokemon introduced with every new iteration.

6 Datasets

PokéAPI https://pokeapi.co/

Pokebase https://pypi.org/project/pokebase/

In this dataset, we plan to collect all legendary and non-legendary Pokémon stats, types, and physical attributes and analyze trends we observed that set the two categories apart. There are 6 stats (Hit Points, Attack, Defense, Special Attack, Special Defense, and Speed) and 18 types (Fire, Water, Grass, Electric, Psychic, Steel, Normal, Fairy, Dark, Flying, Ghost, Poison, Ice, Ground, Rock, Dragon, Fighting, and Bug). Physical attributes, i.e. height and weight, are measured in decimeters and hectograms respectively. This data is publicly available for everyone to access. We are using a package called Pokebase that is provided by the same creators to easily access the API. pokebase is licensed under the 3-Clause BSD License which is a fairly permissive license. We determined whether a Pokemon is legendary or not based on its classification on Bulbapedia, and only used the default forms for Pokemon with multiple in our analysis for consistency.

Bulbapedia https://bulbapedia.bulbagarden.net/wiki/Legendary_Pok%C3%A9mon

Bulbapedia is a community-driven resource about Pokémon and is one of the biggest Pokémon resources in the community. We use their page on legendary Pokémon to for a list of legendary Pokémon to classify our data.

Smogon University https://www.smogon.com/stats

Smogon University is a competitive Pokémon community. They are particularly known for running an online Pokémon battle simulator called Pokémon Showdown, where players can build teams and battle in various competitive rulesets. During peak times, the player count can go up to the twenty thousands (https://pokemonshowdown.com/userstats). Every month, Smogon publicly provides plaintext statistics about how Pokémon are used per competitive ruleset. This data is from March 2019, as that is when we began our analysis. We are specifically interested in the Uber ruleset, which allows the use of practically every Pokémon, including potentially powerful legendaries ("Ubers is the most inclusive of Smogon's tiers, allowing the use of any Pokémon species" per https://www.smogon.com/dex/sm/formats/uber/). Smogon's data also is separated by various skill level tiers. Since we want to look at usage data across all skill levels, we use the dataset that includes all results regardless of skill level. There are three usage statistics: usage, raw, and real. "Usage" is the amount a Pokemon is seen on a team, weighted by the player's skill level. "Raw" is the amount a Pokemon is seen on a team, unweighted by skill level. "Real" is the amount a Pokemon is actually "sent out" and battles.

7 Privacy and Ethics Considerations

For the data from PokéAPI, there are no privacy concerns nor terms of use concerns that we will need to comply with. The data on PokéAPI is not personally identifiable whatsoever, as they are based on created data within the games. There are no other issues related to our data and analyses that could be problematic in terms of privacy. Since Pokémon data concerns objective data, there should not be any biases. The data the API provides has been collected from another open source project and a fan-run wiki; however, these have been vetted as they were scraped and should be free of errors.

For the legendary list taken from Bulbapedia, there is a possibility that people may maliciously edit the page which would lead to incorrect data being used for analysis. However, Bulbapedia has many guidelines on editing and user interaction (https://bulbapedia.bulbagarden.net/wiki/Category:Bulbapedia_policies), and is one of the largest and longest standing community resources (https://bulbapedia.bulbagarden.net/wiki/Bulbapedia:About). This combined with the fact that the list of legendary Pokémon is a fairly static list gives us the confidence that this data is safe to use and free of bias.

Competitive usage data was taken from Smogon's Pokémon Showdown statistics. These statistics are collected by Smogon from their Pokémon Showdown server. There are no privacy concerns as the data available here is not personally identifiable. The only data that is recorded is how often a Pokémon shows up on a player's team, and what moves they use. The data can be grouped by player ranking; however, these groups each contain many players and have no personally identifiable information.

Since this usage data is sorted by month/year and optionally by player ranking, there is a potential for bias in both the date and rank. Pokémon usage may rise and/or fall over time, especially if a new game has been released or other rumors. However, since the data we analyzed was collected long after the current generation has been released, and long before a new generation will be released, this should not be too much of an issue. In addition, we used statistics for all players and not just the top players so that the popularity can be seen amongst all skill levels.

We believe that there are no more ethical concerns to address.

8 Part 0: Import Packages

[1]:
```python
import sys
import requests
from bs4 import BeautifulSoup
from urllib.request import urlopen
import json
import pandas as pd
import math
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

!{sys.executable} -m pip install pokebase
import pokebase as pb
```

```
Requirement already satisfied: pokebase in
c:\users\ellio\appdata\local\programs\python\python37\lib\site-packages (1.2.0)
Requirement already satisfied: requests in
c:\users\ellio\appdata\local\programs\python\python37\lib\site-packages (from
pokebase) (2.22.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
c:\users\ellio\appdata\local\programs\python\python37\lib\site-packages (from
requests->pokebase) (1.25.3)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in
c:\users\ellio\appdata\local\programs\python\python37\lib\site-packages (from
requests->pokebase) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in
c:\users\ellio\appdata\local\programs\python\python37\lib\site-packages (from
requests->pokebase) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in
```

```
c:\users\ellio\appdata\local\programs\python\python37\lib\site-packages (from
requests->pokebase) (2019.3.9)
```

9 Part 1: Data Collection and Cleaning

[2]:
```
# Limit dataframe views to reduce page count
pd.options.display.max_rows = 9
```

9.1 Fetch and Clean Smogon Usage Data

[3]:
```
data = urlopen("https://www.smogon.com/stats/2019-03/gen7ubers-0.txt")
usage = []
for line in data:
    decoded = line.decode('utf-8')
    decoded = decoded.replace(" ", "")
    if decoded.startswith('|'):
        # usage - weighted based on matchmaking rating
        # raw - unweighted usage (on team)
        # real - actually used in battle
        decodedList = decoded.split('|')[1:-1]
        decodedList.pop(2) # remove usage %
        decodedList.pop() # remove real
        decodedList.pop() # remove real %
        usage.append(decodedList)
smogon_df = pd.DataFrame(usage[1:], columns=usage[0])
smogon_df
```

[3]:
```
    Rank             Pokemon     Raw        %
0      1      Groudon-Primal  274400  39.068%
1      2             Xerneas  198831  28.309%
2      3  Necrozma-Dusk-Mane  149044  21.220%
3      4             Yveltal  147205  20.959%
..   ...                 ...     ...      ...
953  954          Karrablast       4   0.001%
954  955             Whismur       4   0.001%
955  956     Pumpkaboo-Large       2   0.000%
956  957       Lurantis-Totem       1   0.000%

[957 rows x 4 columns]
```

The specific usage statistic we are interested is called “raw [usage]”, as we do not want to weigh by a player’s skill. In addition, a Pokémon merely appearing on a team means a player has made the decision to reserve a valuable team spot for a Pokémon. The “usage” and “real” columns have been dropped to make data manipulation easier.

From now on, “raw” and “usage” both refer to the this raw statistic.

9.2 Obtain List of Legendary and Mystical Pokémon from Bulbapedia

[4]:
```python
# scrape list of legendary and mystical pokemon from bulbapedia
page = requests.get("https://bulbapedia.bulbagarden.net/wiki/
 ↪Legendary_Pok%C3%A9mon")
soup = BeautifulSoup(page.content, 'html.parser')

# list of <a> tags of legendary pokemon
a_list = soup.select('td[style*="background: #e6e6ff"] a')

# extract text from <a> tags
legend_list = [lp.get_text() for lp in a_list]

# data cleaning: replace whitespace with hyphen, remove colon
legend_list = [p.lower().replace(' ', '-').replace(':', '') for p in␣
 ↪legend_list]
legend_list_smogon = legend_list.copy()
print(legend_list)
```

```
['articuno', 'zapdos', 'moltres', 'mewtwo', 'mew', 'raikou', 'entei', 'suicune',
'lugia', 'ho-oh', 'celebi', 'regirock', 'regice', 'registeel', 'latias',
'latios', 'kyogre', 'groudon', 'rayquaza', 'jirachi', 'deoxys', 'uxie',
'mesprit', 'azelf', 'dialga', 'palkia', 'heatran', 'regigigas', 'giratina',
'cresselia', 'phione', 'manaphy', 'darkrai', 'shaymin', 'arceus', 'victini',
'cobalion', 'terrakion', 'virizion', 'tornadus', 'thundurus', 'reshiram',
'zekrom', 'landorus', 'kyurem', 'keldeo', 'meloetta', 'genesect', 'xerneas',
'yveltal', 'zygarde', 'diancie', 'hoopa', 'volcanion', 'type-null', 'silvally',
'tapu-koko', 'tapu-lele', 'tapu-bulu', 'tapu-fini', 'cosmog', 'cosmoem',
'solgaleo', 'lunala', 'necrozma', 'magearna', 'marshadow', 'zeraora', 'meltan',
'melmetal', 'zacian', 'zamazenta']
```

Their page on legendary and mythical Pokémon has a convenient table at the end with a list of legendary Pokémon. We select this table and scrape it for a list of Pokémon names.

9.3 Test Legendary Name Validity by Searching Each Name in the API

[5]:
```python
for p in legend_list:
    try:
        pb.pokemon(p)
    except ValueError:
        print("Cannot find", p)
```

```
Cannot find deoxys
Cannot find giratina
Cannot find shaymin
Cannot find tornadus
Cannot find thundurus
Cannot find landorus
Cannot find keldeo
Cannot find meloetta
Cannot find meltan
Cannot find melmetal
Cannot find zacian
Cannot find zamazenta
```

9.4 Handle Alternate Forms Manually

In the Pokémon universe, each Pokémon species may have multiple forms. We want to make sure we properly classify each legendary species' form.

P.S. Meltan and Melmetal, mythical Pokémon introduced in generation VII, haven't been added to the database at the moment. https://github.com/PokeAPI/pokeapi/issues/414

P.S. Zacian and Zamazenta, new legendary Pokémon introduced in generation VIII, haven't been added to the database at the moment.

[6]:
```
# deoxys: deoxys-normal, deoxys-attack, deoxys-defense, deoxys-speed
try:
    legend_list.remove('deoxys')
    legend_list.extend(['deoxys-normal', 'deoxys-attack', 'deoxys-defense',␣
 ↪'deoxys-speed'])
except ValueError:
    print('Error')

# giratina: giratina-altered, giratina-origin
try:
    legend_list.remove('giratina')
    legend_list.extend(['giratina-altered', 'giratina-origin'])
except ValueError:
    print('Error')

# shaymin: shaymin-land, shaymin-sky
try:
    legend_list.remove('shaymin')
    legend_list.extend(['shaymin-land', 'shaymin-sky'])
except ValueError:
    print('Error')

# tornadus: tornadus-incarnate, tornadus-therian
try:
    legend_list.remove('tornadus')
    legend_list.extend(['tornadus-incarnate', 'tornadus-therian'])
except ValueError:
    print('Error')

# thundurus: thundurus-incarnate, thundurus-therian
try:
    legend_list.remove('thundurus')
    legend_list.extend(['thundurus-incarnate', 'thundurus-therian'])
except ValueError:
    print('Error')

# landorus: landorus-incarnate, landorus-therian
try:
    legend_list.remove('landorus')
    legend_list.extend(['landorus-incarnate', 'landorus-therian'])
except ValueError:
    print('Error')

# keldeo: keldeo-ordinary, keldeo-resolute
try:
    legend_list.remove('keldeo')
    legend_list.extend(['keldeo-ordinary', 'keldeo-resolute'])
except ValueError:
    print('Error')

# meloetta: meloetta-aria, meloetta-pirouette
try:
    legend_list.remove('meloetta')
    legend_list.extend(['meloetta-aria', 'meloetta-pirouette'])
except ValueError:
    print('Error')

# meltan: remove for now
try:
    legend_list.remove('meltan')
except ValueError:
    print('Error')

# melmetal: remove for now
try:
    legend_list.remove('melmetal')
except ValueError:
    print('Error')

# zacian: remove for now
try:
    legend_list.remove('zacian')
except ValueError:
    print('Error')

# zamazenta: remove for now
try:
    legend_list.remove('zamazenta')
```

```
except ValueError:
    print('Error')
```

9.5 Test Again for Validity

[7]:
```
for p in legend_list:
    try:
        pb.pokemon(p)
    except ValueError:
        print("Cannot find", p)
```

9.6 Create CSV File for All Pokémon Info

[8]:
```
# #Only run if you need the DF in the code itself, the cell above should␣
↪retrieve the file
# import re
# pokemonList = []
# pokemonList.append(['ID', 'Pokemon', 'Legendary', 'Stat Total', 'ATK Sum',␣
↪'DEF Sum', 'Height', 'Weight', 'Type 1', 'Type 2'])

# for pokemonID in range(1,808):
#   pokemon_stats = []
#   pokemon_info = pb.pokemon(pokemonID)

#   pokemon_stats.append(pokemonID) # Add ID
#   pokemon_stats.append(pokemon_info.name) # Add Name
#   pokemon_stats.append(pokemon_info.name in legend_list) # Add Legendary or␣
↪Not
#   pokemon_stats.extend([0,0,0]) # Add categories for stat totals
#   pokemon_stats.append(pokemon_info.height) # Add Height
#   pokemon_stats.append(pokemon_info.weight) # Add Weight

#   print(pokemonID)
#   j = 0 # Counter because I didnt want to change my code anymore
#   for i in (pokemon_info.stats):
#     stringStats = str(i)
#     value = [int(s) for s in re.findall(r'\b\d+\b', stringStats)[:1]]
#     pokemon_stats[3] += value[0]
#     pokemon_stats[4+(j%2)] += value[0]
#     j += 1

#   for i in pokemon_info.types:
#     stringType = str(i)
#     typing = re.findall('\'name\': \'(.+?)\'', stringType)
#     pokemon_stats.append(typing[0])
#   if len(pokemon_info.types) == 1:
#     pokemon_stats.append("N/A")

#   pokemonList.append(pokemon_stats)

# statsDF = pd.DataFrame(pokemonList[1:], columns=pokemonList[0])
# statsDF.to_csv ('./statsDF.csv', index = None, header=True)

# statsDF
```

[9]:
```
df = pd.read_csv('statsDF.csv')
df
```

[9]:
```
      ID       Pokemon  Legendary  Stat Total  ATK Sum  DEF Sum  Height  \
0      1     bulbasaur      False         318      159      159       7
1      2       ivysaur      False         405      202      203      10
2      3      venusaur      False         525      262      263      20
3      4    charmander      False         309      177      132       6
..   ...           ...        ...         ...      ...      ...     ...
803  804     naganadel      False         540      321      219      36
804  805     stakataka      False         570      197      373      55
805  806   blacephalon      False         570      385      185      18
806  807       zeraora       True         600      357      243      15

     Weight    Type 1  Type 2
0        69    poison   grass
1       130    poison   grass
2      1000    poison   grass
3        85      fire     NaN
..      ...       ...     ...
803    1500    dragon  poison
804    8200     steel    rock
805     130     ghost    fire
806     445  electric     NaN

[807 rows x 10 columns]
```

9.7 Obtain List of Pokémon Types

[10]:
```
# scrape list of pokemon types from bulbapedia
page = requests.get("https://bulbapedia.bulbagarden.net/wiki/Type")
soup = BeautifulSoup(page.content, 'html.parser')

# list of <a> tags of types
a_list = soup.select('td a[title*="(type)"]')
```

```
# extract text from <a> tags
type_list = [lp.get_text() for lp in a_list]

# ignore ??? type
type_list = type_list[:-1]
print(type_list)
```

```
['Normal', 'Fire', 'Fighting', 'Water', 'Flying', 'Grass', 'Poison', 'Electric',
'Ground', 'Psychic', 'Rock', 'Ice', 'Bug', 'Dragon', 'Ghost', 'Dark', 'Steel',
'Fairy']
```

9.8 Match Legendary Names to Smogon Names

[11]:
```
legend_list_smogon.extend(['deoxys-attack', 'deoxys-defense', 'deoxys-speed'])
legend_list_smogon.extend(['giratina-origin'])
legend_list_smogon.extend(['shaymin-sky'])
legend_list_smogon.extend(['tornadus-therian'])
legend_list_smogon.extend(['thundurus-therian'])
legend_list_smogon.extend(['landorus-therian'])

try:
    legend_list_smogon.remove('meltan')
except ValueError:
    print('Error')

try:
    legend_list_smogon.remove('melmetal')
except ValueError:
    print('Error')

# smogon pokemon names are captialized
legend_list_smogon = [str.title(name) for name in legend_list_smogon]

try:
    legend_list_smogon.remove('Tapu-Koko')
    legend_list_smogon.extend(['TapuKoko'])
except ValueError:
    print('Error')

try:
    legend_list_smogon.remove('Tapu-Lele')
    legend_list_smogon.extend(['TapuLele'])
except ValueError:
    print('Error')

try:
    legend_list_smogon.remove('Tapu-Bulu')
    legend_list_smogon.extend(['TapuBulu'])
except ValueError:
    print('Error')

try:
    legend_list_smogon.remove('Tapu-Fini')
    legend_list_smogon.extend(['TapuFini'])
except ValueError:
    print('Error')

try:
    legend_list_smogon.remove('Type-Null')
    legend_list_smogon.extend(['Type:Null'])
except ValueError:
    print('Error')

print(legend_list_smogon)
```

```
['Articuno', 'Zapdos', 'Moltres', 'Mewtwo', 'Mew', 'Raikou', 'Entei', 'Suicune',
'Lugia', 'Ho-Oh', 'Celebi', 'Regirock', 'Regice', 'Registeel', 'Latias',
'Latios', 'Kyogre', 'Groudon', 'Rayquaza', 'Jirachi', 'Deoxys', 'Uxie',
'Mesprit', 'Azelf', 'Dialga', 'Palkia', 'Heatran', 'Regigigas', 'Giratina',
'Cresselia', 'Phione', 'Manaphy', 'Darkrai', 'Shaymin', 'Arceus', 'Victini',
'Cobalion', 'Terrakion', 'Virizion', 'Tornadus', 'Thundurus', 'Reshiram',
'Zekrom', 'Landorus', 'Kyurem', 'Keldeo', 'Meloetta', 'Genesect', 'Xerneas',
'Yveltal', 'Zygarde', 'Diancie', 'Hoopa', 'Volcanion', 'Silvally', 'Cosmog',
'Cosmoem', 'Solgaleo', 'Lunala', 'Necrozma', 'Magearna', 'Marshadow', 'Zeraora',
'Zacian', 'Zamazenta', 'Deoxys-Attack', 'Deoxys-Defense', 'Deoxys-Speed',
'Giratina-Origin', 'Shaymin-Sky', 'Tornadus-Therian', 'Thundurus-Therian',
'Landorus-Therian', 'TapuKoko', 'TapuLele', 'TapuBulu', 'TapuFini', 'Type:Null']
```

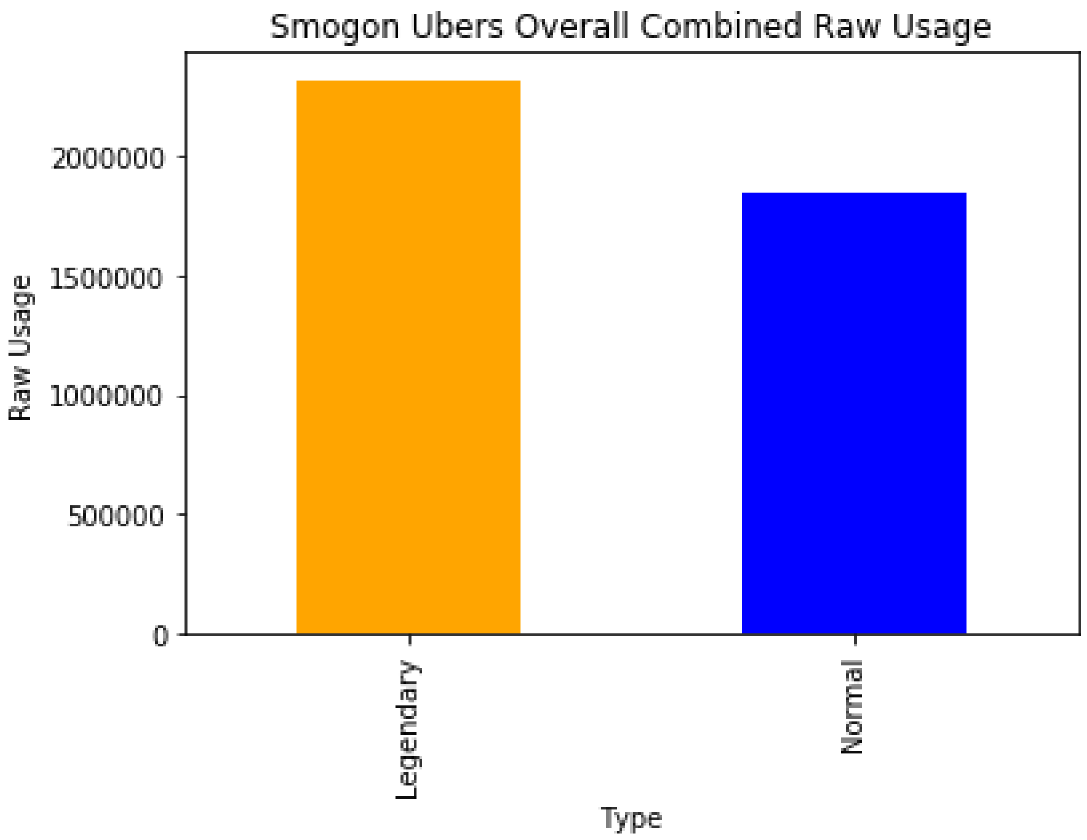
Smogon's name formatting is somewhat different than the format we use elsewhere in this notebook. We are doing some manual editing for entries that differ to match the naming used by Smogon.

10 Part 2: Smogon Competitive Usage Data Analysis

10.1 Visualizing Smogon Usage Data

[12]:
```
# convert string to number
smogon_df['Raw'] = smogon_df['Raw'].astype(int)

# using legend_list_smogon, split dataframe
smogon_legend_df = smogon_df[smogon_df['Pokemon'].str.contains('|'.
 ↪join(legend_list_smogon))].copy()
smogon_legend_df['Rank'] = smogon_legend_df['Rank'].astype(int)
```

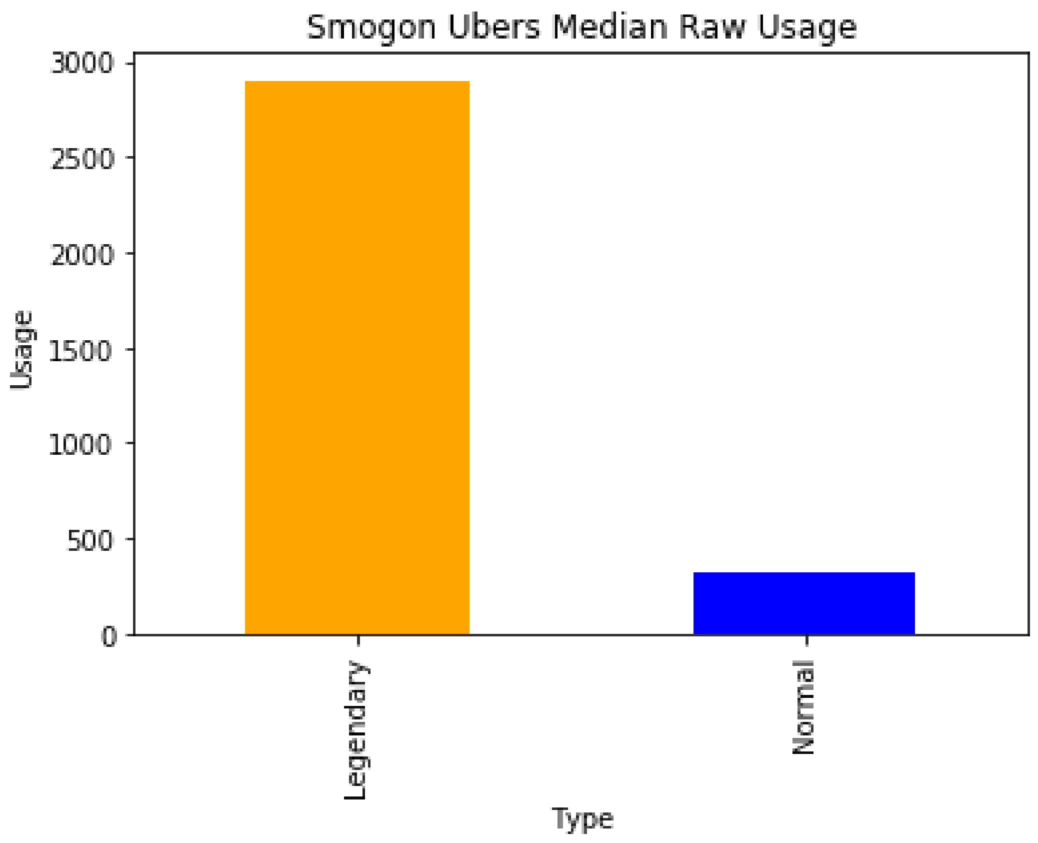
```
smogon_normal_df = smogon_df[-smogon_df['Pokemon'].str.contains('|'.
 ↪join(legend_list_smogon))].copy()
smogon_normal_df['Rank'] = smogon_normal_df['Rank'].astype(int)

# plot raw usage of legendaries vs normal
smogon_usage_overall_graph_data = [
  ['Legendary', smogon_legend_df['Raw'].sum()],
  ['Normal', smogon_normal_df['Raw'].sum()]
]
smogon_usage_overall_graph_df = pd.DataFrame(smogon_usage_overall_graph_data,␣
 ↪columns = ['Type', 'Raw Usage'])
ax_smogon_usage_overall = smogon_usage_overall_graph_df.plot.bar(x = 'Type', y␣
 ↪= 'Raw Usage',
  title = 'Smogon Ubers Overall Combined Raw Usage', legend = False, color =␣
 ↪['orange', 'blue'])
ax_smogon_usage_overall.set_xlabel('Type')
ax_smogon_usage_overall.set_ylabel('Raw Usage')
plt.show()
smogon_usage_overall_graph_df
```



[12]:
```
         Type  Raw Usage
0  Legendary    2319636
1     Normal    1848922
```

Comparing the overall combined raw usage of legendary against normal Pokémon, we can see that legendary Pokémon are somewhat more popular in team compositions by a difference of 22.58%.

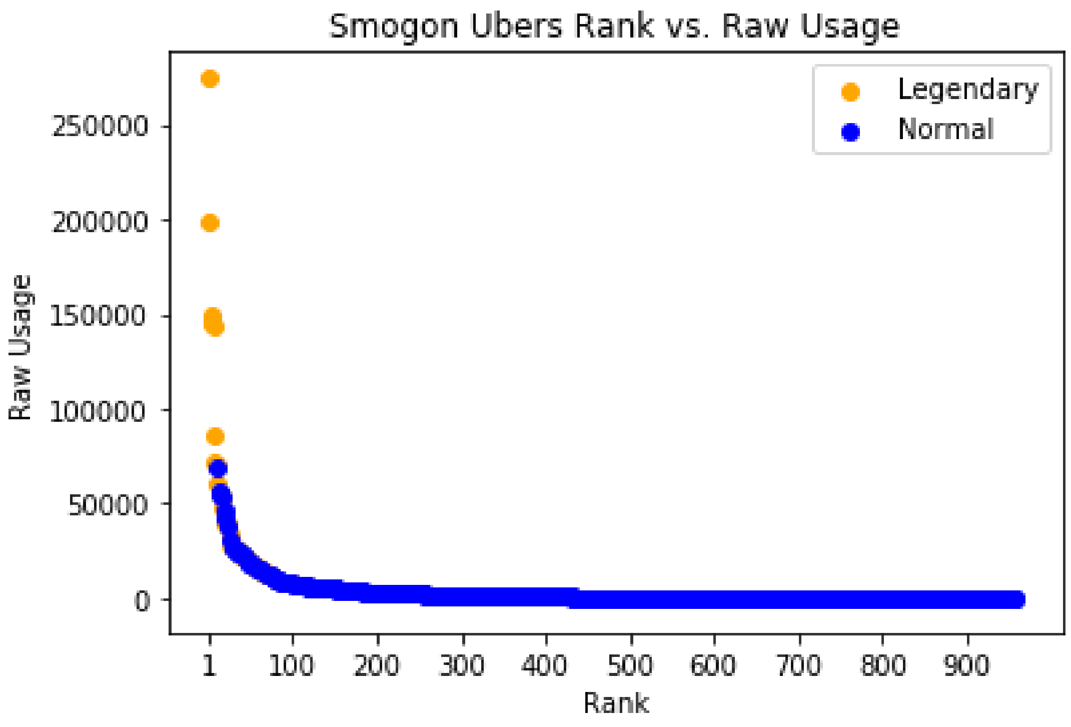
[13]:
```
smogon_graph_data_usage = [
    ['Legendary', smogon_legend_df['Raw'].median()],
    ['Normal', smogon_normal_df['Raw'].median()]
]
smogon_usage_graph_df = pd.DataFrame(smogon_graph_data_usage, columns =␣
 ↪['Type', 'Usage'])
ax_smogon_usage = smogon_usage_graph_df.plot.bar(x = 'Type', y = 'Usage', title␣
 ↪= 'Smogon Ubers Median Raw Usage',
    legend = False, color = ['orange', 'blue'])
ax_smogon_usage.set_xlabel('Type')
ax_smogon_usage.set_ylabel('Usage')
plt.show()
smogon_usage_graph_df
```



[13]:
```
        Type   Usage
0  Legendary  2900.0
1     Normal   321.0
```

Here, we graph the median usage. The median has been chosen over the mean as it is more robust to potential outliers. Comparing the two usage medians, legendary usage appears to be ahead by a staggering amount. We should graph each individual usage to see what is happening.

[14]:
```
# plot raw usage against rank
plt.scatter(smogon_legend_df['Rank'], smogon_legend_df['Raw'], color = 'orange')
plt.scatter(smogon_normal_df['Rank'], smogon_normal_df['Raw'], color = 'blue')
plt.title('Smogon Ubers Rank vs. Raw Usage')
plt.xlabel('Rank')
plt.ylabel('Raw Usage')
plt.xticks([1, *range(100, len(smogon_df), 100)])
plt.legend(['Legendary', 'Normal'])
plt.show()
```



Here, we plot the raw usage of each Pokémon against their rank. It is apparent that usage is exponential, meaning only a certain amount of Pokémon are widely used and the rest quickly fall off in usage. In addition, at first glance it would appear that legendary Pokémon seem to dominate the upper ranks. It is hard to tell for sure without a further breakdown since the dots overlap each other.

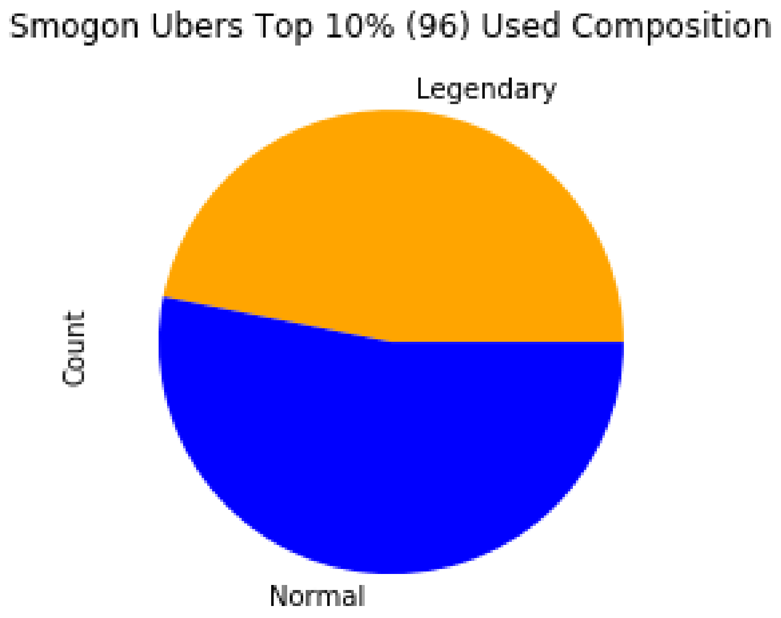
[15]:
```
print('Legendary:\n', smogon_legend_df['Raw'].describe(), '\n\nNormal:\n',␣
↪smogon_normal_df['Raw'].describe())
```

```
Legendary:
 count       123.000000
mean       18858.829268
std        40850.295869
min           25.000000
25%          818.000000
50%         2900.000000
75%        17505.500000
max       274400.000000
Name: Raw, dtype: float64

Normal:
 count      834.000000
mean       2216.932854
std        6011.370972
min           1.000000
25%          63.000000
50%         321.000000
75%        1701.750000
max       68805.000000
Name: Raw, dtype: float64
```

Looking at the statistics and graph of all legendary and normal Pokémon, our data is subject to extreme skew due to the amount of Pokémon that remain relatively unused. Thus, we will visualize the most used Pokémon to make it easier to analyze. We decided on the top 10% usage of all Pokémon, which rounds up to 96.

10.1.1 Visualizing Just the Top 10%

[16]:
```
# get top x% used
smogon_top_count = math.ceil(len(smogon_df) * 0.1) # used for the cutoffs
smogon_legend_top_df = (smogon_legend_df[smogon_legend_df['Rank'] <=␣
 ↪smogon_top_count])
smogon_normal_top_df = (smogon_normal_df[smogon_normal_df['Rank'] <=␣
 ↪smogon_top_count])

smogon_num_legend_top = smogon_legend_top_df['Rank'].size
smogon_num_normal_top = smogon_normal_top_df['Rank'].size
smogon_num_graph_data = [
  ['Legendary', smogon_num_legend_top],
  ['Normal', smogon_num_normal_top]
]
smogon_num_graph_top_df = pd.DataFrame(smogon_num_graph_data, columns =␣
 ↪['Type', 'Count'])
ax_smogon_num_top = smogon_num_graph_top_df.plot.pie(x = 'Type', y = 'Count',␣
 ↪title = 'Smogon Ubers Top 10% ({}) Used Composition'.
 ↪format(smogon_top_count),
  legend = False, colors = ['orange', 'blue'], labels = ['Legendary', 'Normal'])
plt.show()
smogon_num_graph_top_df
```



[16]:
```
        Type  Count
0  Legendary     45
1     Normal     51
```

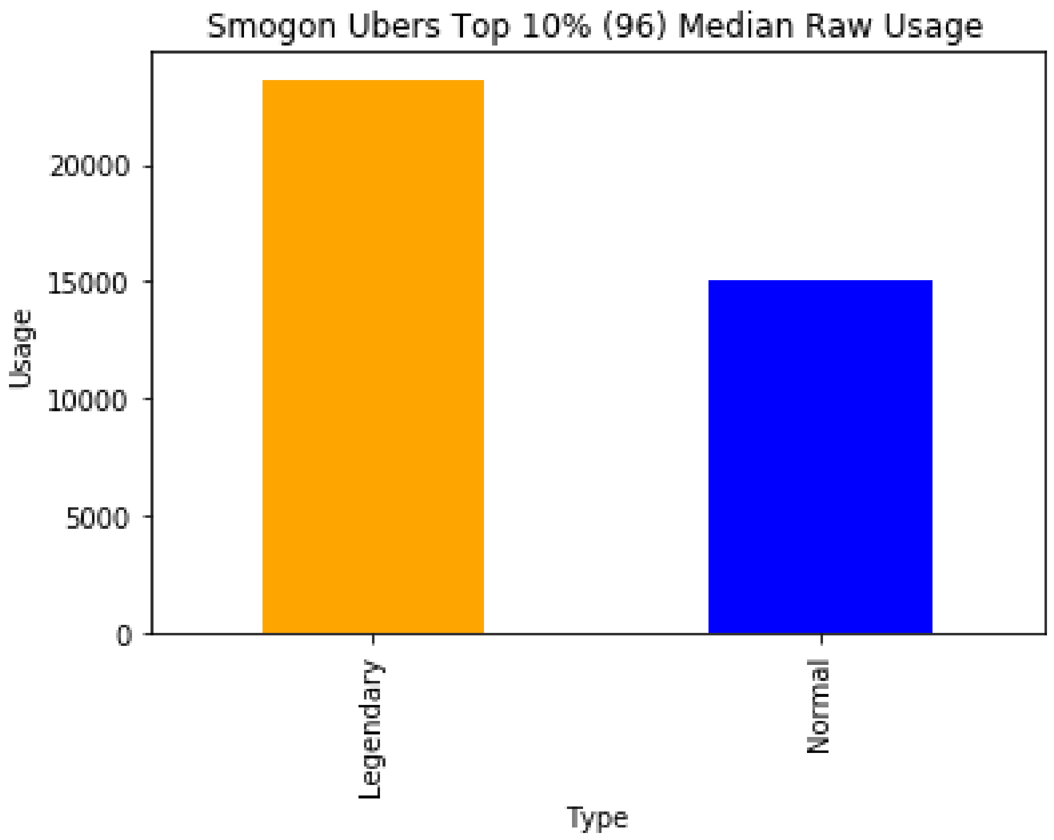
[17]:
```
print('Legendary in Top 10%:\n', smogon_legend_top_df['Raw'].describe(), \
      '\n\nNormal in Top 10%:\n', smogon_normal_top_df['Raw'].describe())
```

```
Legendary in Top 10%:
 count        45.000000
mean      48429.933333
std       56661.093534
min        8907.000000
25%       15831.000000
50%       23644.000000
75%       53916.000000
max      274400.000000
Name: Raw, dtype: float64

Normal in Top 10%:
 count       51.000000
mean     20206.274510
std      14432.625164
```

```
min       8051.000000
25%       9928.000000
50%      15080.000000
75%      24082.500000
max      68805.000000
Name: Raw, dtype: float64
```

Looking at the statistcs of the top 10%, we can see this is somewhat more reasonable than analyzing them all. In the top 10%, there are more normal Pokémon than legendary Pokémon by a difference of 12.50%. However, the difference in usage is still quite large.

[18]:
```
# get the makeup of the top x% used
smogon_num_graph_data_usage = [
  ['Legendary', smogon_legend_top_df['Raw'].median()],
  ['Normal', smogon_normal_top_df['Raw'].median()]
]
smogon_usage_top_graph_df = pd.DataFrame(smogon_num_graph_data_usage, columns =␣
 ↪['Type', 'Usage'])
ax_smogon_usage_top = smogon_usage_top_graph_df.plot.bar(x = 'Type', y =␣
 ↪'Usage', title = 'Smogon Ubers Top 10% ({}) Median Raw Usage'.
 ↪format(smogon_top_count),
  legend = False, color = ['orange', 'blue'])
ax_smogon_usage_top.set_xlabel('Type')
ax_smogon_usage_top.set_ylabel('Usage')
plt.show()
smogon_usage_top_graph_df
```



[18]:
```
        Type    Usage
0  Legendary  23644.0
1     Normal  15080.0
```

Again, the medians have been graphed instead of the mean as the maximum legendary usage is quite large, and the median is more robust in this scenario. Looking at just the top 10% medians of raw usage, legendary Pokémon are still used far more than normal Pokémon by a difference of 50.84%.

10.2 Analysis of Usage Results

The usage of legendary Pokémon compared to normal Pokémon is undoubtedly larger. This is especially true at the very top - the difference between the most used legendary Pokémon far surpass the normal Pokémon. With this usage difference, the fact that there are ten more normal Pokémon than legendary in the top 10% may be surprising. However, it is important to note that there are vastly more normal Pokémon than legendary. Smogon has 838 entries that are classified as normal, compared to 119 that are classified as legendary. With that many more normal Pokémon, there are bound to be quite a few normal Pokémon that can be considered a valuable addition to the team. Regardless, this large difference in usage between legendary and normal Pokémon is significant and must be due to some kind of attribute differences legendary Pokémon possess. In the next part, we will look at the objective data of legendary Pokémon against normal Pokemon.

11 Part 3: Physical Attributes Analysis

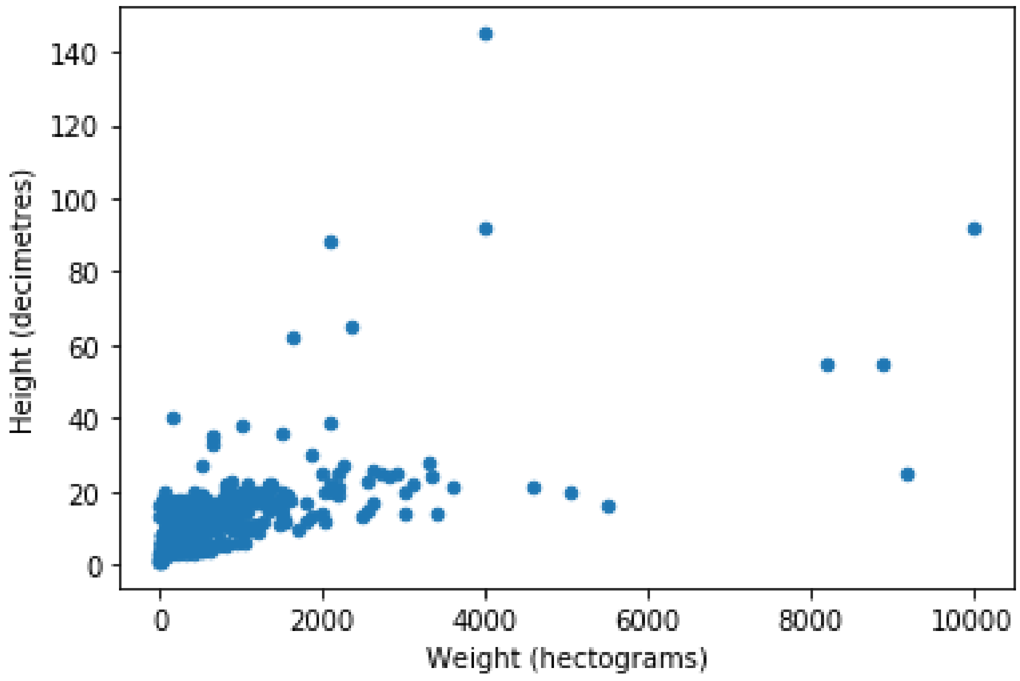
11.1 3.1 Weight vs. Height

Using a scatter plots, we can examine the relationship between the height and weight of all Pokémon. The plots below show how most Pokémon's measurements concentrate toward the bottom left corner, whereas there are only a few outliers.

11.1.1 Non-legendaries

[19]:
```
df[~df['Pokemon'].isin(legend_list)][['Height', 'Weight']].plot.
↪scatter('Weight', 'Height').set(xlabel='Weight (hectograms)', ylabel='Height␣
↪(decimetres)')
```

[19]:
```
[Text(0, 0.5, 'Height (decimetres)'), Text(0.5, 0, 'Weight (hectograms)')]
```

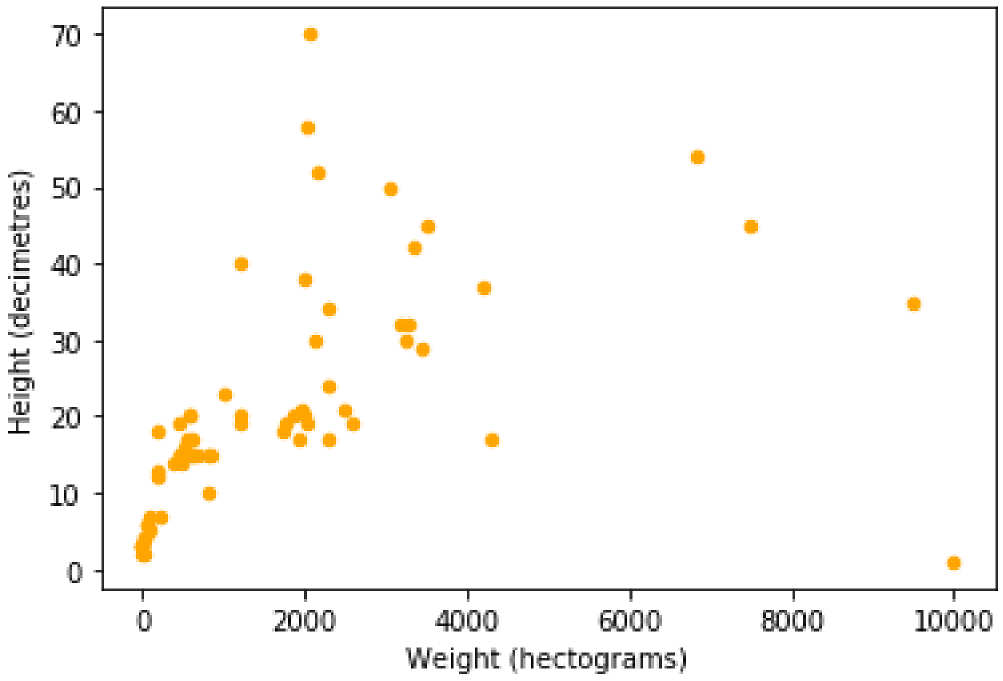


At first glance, we can see in the scatter plot that the bulk of Non-Legendary Pokémons' Height and Weight are concentrated in the bottom left. This indicates that Non-Legendaries tend to be smaller in size.

11.1.2 Legendaries

[20]:
```
df[df['Pokemon'].isin(legend_list)][['Height', 'Weight']].plot.
↪scatter('Weight', 'Height', c='orange').set(xlabel='Weight (hectograms)',␣
↪ylabel='Height (decimetres)')
```

[20]:
```
[Text(0, 0.5, 'Height (decimetres)'), Text(0.5, 0, 'Weight (hectograms)')]
```

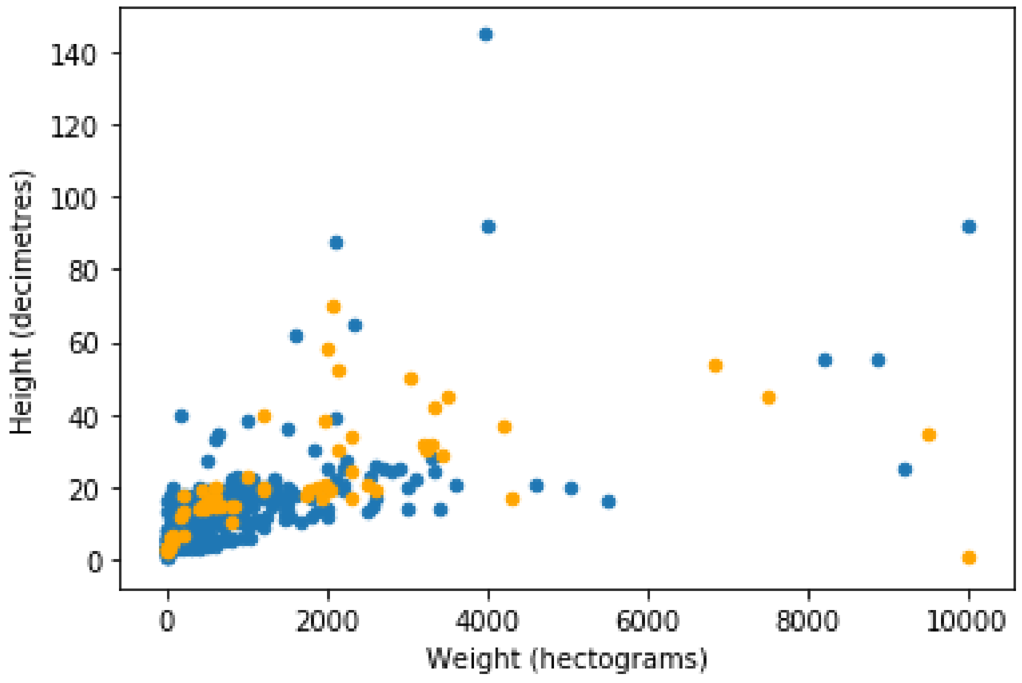


For this scatterplot, the first thing to note is that there is significantly less Legendary Pokémons than Non-Legendaries. Additionally, we can see that the Height and Weight for Legendaries vary extremely - with most being in the larger range.

11.1.3 Combined

[21]:
```
ax = df[~df['Pokemon'].isin(legend_list)][['Height', 'Weight']].plot('Weight',␣
 ↪'Height', kind='scatter')
df[df['Pokemon'].isin(legend_list)][['Height', 'Weight']].plot('Weight',␣
 ↪'Height', kind='scatter', c='orange', ax=ax).set(xlabel='Weight␣
 ↪(hectograms)', ylabel='Height (decimetres)')
```

[21]:
```
[Text(0, 0.5, 'Height (decimetres)'), Text(0.5, 0, 'Weight (hectograms)')]
```



Combining both scatter plots shows that although Legendary Pokémons are larger in size, there are exceptions to this case. An outlier example being that there is a Non-Legendary that is the tallest out of all Pokémon. Another irrational outlier is that there is a Legendary that is short but is the heaviest Pokémon overall.

11.1.4 Means of Weight and Height

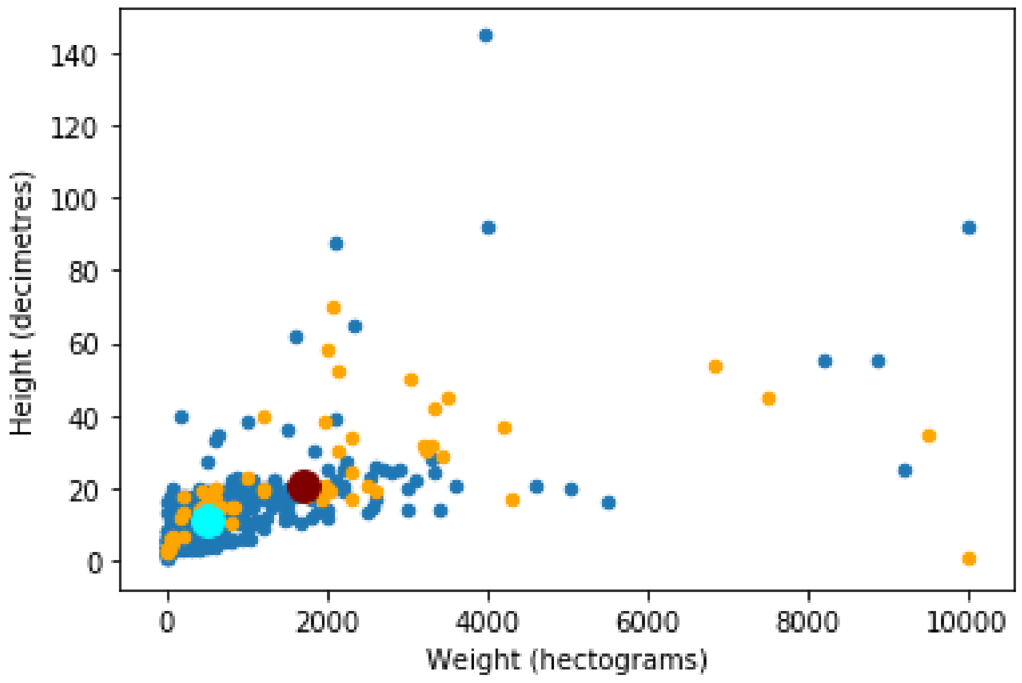
Legendary mean: maroon
Regular mean: cyan

[22]:
```
# legendary average height
l_h_mean = df[df['Pokemon'].isin(legend_list)]['Height'].mean()

# legendary average weight
l_w_mean = df[df['Pokemon'].isin(legend_list)]['Weight'].mean()

# regular average height
r_h_mean = df[~df['Pokemon'].isin(legend_list)]['Height'].mean()

# regular avaerage weight
r_w_mean = df[~df['Pokemon'].isin(legend_list)]['Weight'].mean()

# calculate means
df_l_mean = pd.DataFrame({"Weight":[l_w_mean], "Height":[l_h_mean]})
df_r_mean = pd.DataFrame({"Weight":[r_w_mean], "Height":[r_h_mean]})
```

```
print('Legendary Pokemon average weight: ', round(df_l_mean.loc[0]['Weight'],␣
↪2), '; average height: ', round(df_l_mean.loc[0]['Height'], 2))
print('Regular Pokemon average weight: ', round(df_r_mean.loc[0]['Weight'], 2),␣
↪'; average height: ', round(df_r_mean.loc[0]['Height'], 2))

# plot
ax = df[~df['Pokemon'].isin(legend_list)][['Height', 'Weight']].plot('Weight',␣
↪'Height', kind='scatter')
df[df['Pokemon'].isin(legend_list)][['Height', 'Weight']].plot('Weight',␣
↪'Height', kind='scatter', c='orange', ax=ax)
df_l_mean.plot.scatter('Weight', 'Height', c="maroon", s=140, ax = ax)
df_r_mean.plot.scatter('Weight', 'Height', c="cyan", s=140, ax = ax).
↪set(xlabel='Weight (hectograms)', ylabel='Height (decimetres)')
```

```
Legendary Pokemon average weight:  1716.66 ; average height:  20.66
Regular Pokemon average weight:  516.59 ; average height:  10.79
```

[22]:
```
[Text(0, 0.5, 'Height (decimetres)'), Text(0.5, 0, 'Weight (hectograms)')]
```



We plot the mean Height and Weight data to display the combined average Height & Weight of Legendary and Non-Legendary, individually, amongst all Pokémons. This data also includes outliers but they do not affect the data substantially, as there are many other data points we are analyzing.

Additionally from the plot above, we can conclude that the average legendary Pokémon is likely to be taller and heavier than the average non-legendary Pokémon.

11.2 3.2 Types

Lets identify what typings legendary Pokemon typically are. First we need to separate the legendary data from the overall dataframe

11.2.1 Make a list of Pokémon Types

[23]:
```
legendsDf = df[df['Pokemon'].isin(legend_list)]

legend_types = list(legendsDf['Type 1']) + list(legendsDf['Type 2'])
types = list(df['Type 1']) + list(df['Type 2'])

legendTypeCount = pd.Series(legend_types).value_counts().append(pd.
 ↪Series([0],['poison']))
regularTypeCount = pd.Series(types).value_counts()
```

[24]:
```
legendsDf
```

[24]:
```
      ID    Pokemon  Legendary  Stat Total  ATK Sum  DEF Sum  Height  Weight  \
143  144   articuno       True         580      265      315      17     554
144  145     zapdos       True         580      315      265      16     526
145  146    moltres       True         580      315      265      20     600
149  150     mewtwo       True         680      394      286      20    1220
..   ...        ...        ...         ...      ...      ...     ...     ...
799  800   necrozma       True         600      313      287      24    2300
800  801   magearna       True         600      290      310      10     805
801  802  marshadow       True         600      340      260       7     222
806  807    zeraora       True         600      357      243      15     445

      Type 1    Type 2
143   flying       ice
144   flying  electric
145   flying      fire
149  psychic       NaN
..       ...       ...
799  psychic       NaN
800    fairy     steel
801    ghost  fighting
806 electric       NaN

[68 rows x 10 columns]
```

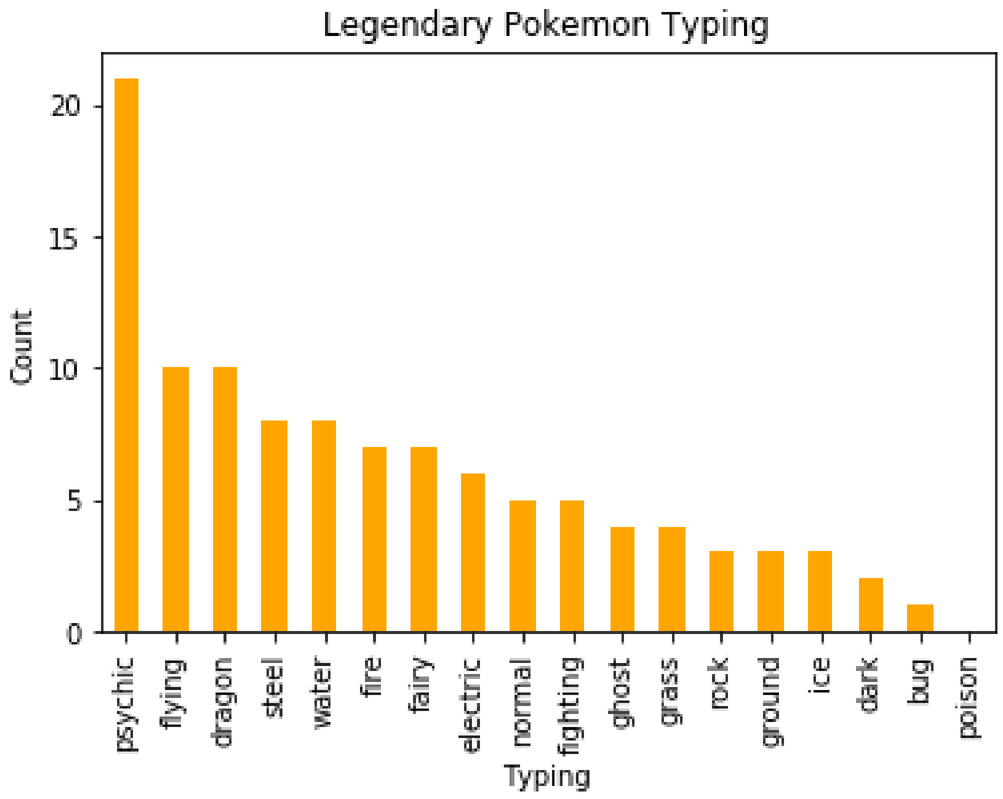
Kind of hard to read, so lets just get the counts instead of individual pokemon.

[25]:
```
legendTypeCount
```

[25]:
```
psychic     21
flying      10
dragon      10
steel        8
```

```
          ..
ice         3
dark        2
bug         1
poison      0
Length: 18, dtype: int64
```

[26]:
```
legendPlot = legendTypeCount.plot(kind='bar', title='Legendary Pokemon Typing',␣
 ↪yticks=range(0,25,5), color="orange")
xstuff = legendPlot.set_xlabel("Typing")
ystuff = legendPlot.set_ylabel("Count")
```



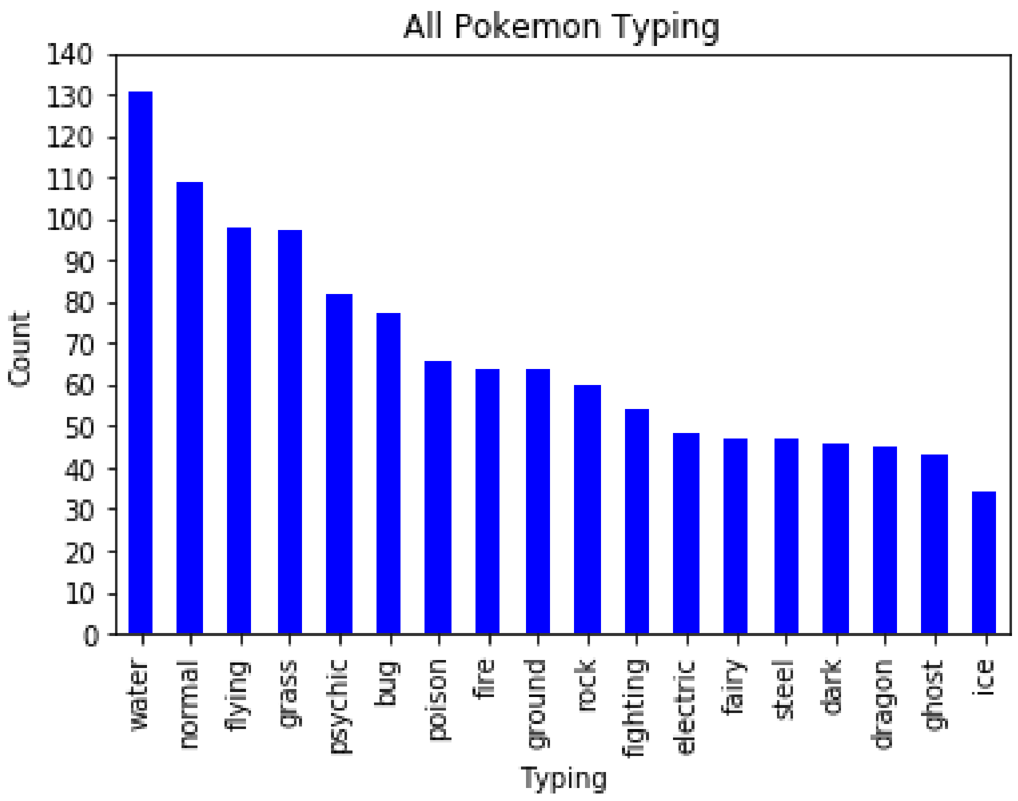
Here we can see that psychic dominate the typings for legendary Pokemon, doubling the counts for any other type. From there it slowly declines until we get to poison, where we can see 0 Pokemon have poison typing. Surprisingly, dragon isnt the most frequent typing, which somewhat subverted our expectations but is reasonable once we thought back onto what legendary Pokemon there were.

[27]:
```
regularTypeCount
```

[27]:
```
water      131
normal     109
flying      98
grass       97
```

```
           ...
dark        46
dragon      45
ghost       43
ice         34
Length: 18, dtype: int64
```

[28]:
```
normalPlot = regularTypeCount.plot(kind='bar', title='All Pokemon Typing',␣
 ↪yticks=range(0,150,10),  color="blue")
xstuff = normalPlot.set_xlabel("Typing")
ystuff = normalPlot.set_ylabel("Count")
```



Now with regular pokemon, we can see that its a lot more evenly spread out. Water has a plurality of the typings but is not as dominant as psychic was for legendaries. Poison finds itself near the middle of the type distribution, suggesting that the lack of a legendary Pokemon being poison typed may be intentional rather than through sheer chance.

[29]:
```
typeCompDf = legendTypeCount.to_frame()
typeCompDf = typeCompDf.merge(regularTypeCount.to_frame(), left_index=True,␣
 ↪right_index=True)
typeCompDf.columns = ['Legendary','Normal']
typeCompDf=typeCompDf.reindex(columns=['Normal','Legendary'])
normFirst = regularTypeCount.to_frame()
normFirst = normFirst.merge(legendTypeCount.to_frame(), left_index = True,␣
  ↪right_index = True)
normFirst.columns = ['Normal','Legendary']
```

Now lets compare the two side-by-side.

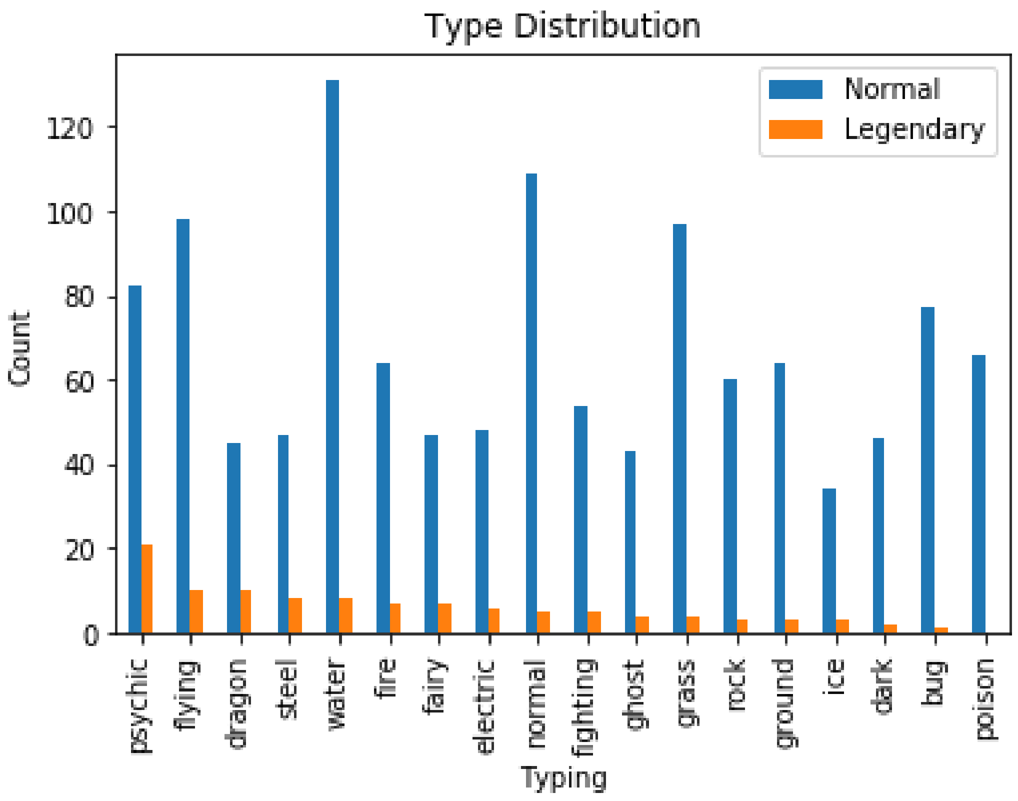
[30]:
```
typeCompDf
```

[30]:
```
         Normal  Legendary
psychic      82         21
flying       98         10
dragon       45         10
steel        47          8
...         ...        ...
ice          34          3
dark         46          2
bug          77          1
poison       66          0

[18 rows x 2 columns]
```

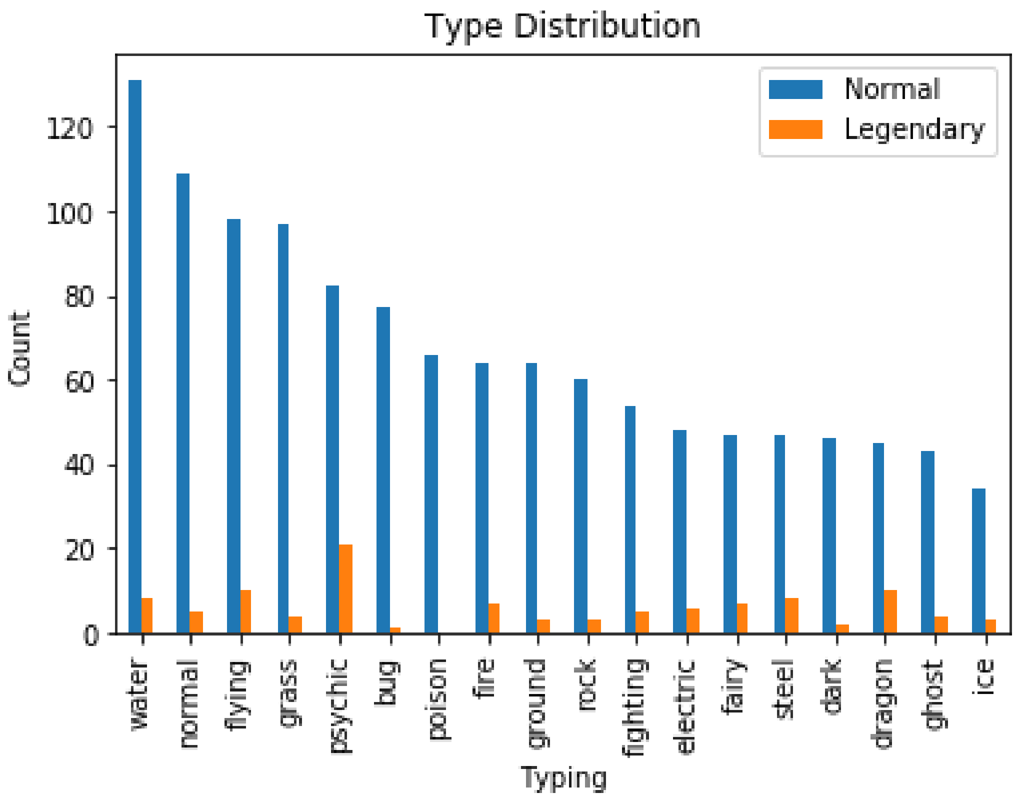
[31]:
```
normFirst
```

[31]:
```
        Normal  Legendary
water      131          8
normal     109          5
flying      98         10
grass       97          4
...        ...        ...
dark        46          2
dragon      45         10
ghost       43          4
ice         34          3

[18 rows x 2 columns]
```

[32]:
```
overallPlot = typeCompDf.plot(kind='bar', title ="Type Distribution")
xstuff = overallPlot.set_xlabel("Typing")
ystuff = overallPlot.set_ylabel("Count")
```



[33]:
```
normFirstPlot = normFirst.plot(kind='bar', title ="Type Distribution")
xstuff = normFirstPlot.set_xlabel("Typing")
ystuff = normFirstPlot.set_ylabel("Count")
```



Through these two graphs, we can see that there only seems to be a small correlation between the typing for all Pokemon in general, and the typing for legendary Pokemon (seen in flying, water, and psychic for both graphs). This suggests that legendary Pokemon are made without this distribution in mind, and more made to appeal to the abstract idea of legendary that most people view as dragons or psychic beings.

12 Part 4: Pokémon Stats Analysis

[34]:
```
# Display plots directly in the notebook instead of in a new window
%matplotlib inline

# Configure libraries
# The seaborn library makes plots look nicer
sns.set()
sns.set_context('talk')

# Don't display too many rows/cols of DataFrames
pd.options.display.max_rows = 9
pd.options.display.max_columns = 10

# Round decimals when displaying DataFrames
```

```
pd.set_option('precision', 2)
```

Reading the CSV

```
[35]: df_poke = pd.read_csv("statsDF.csv")
      df_poke
```

```
[35]:      ID      Pokemon  Legendary  Stat Total  ATK Sum  DEF Sum  Height  \
      0      1    bulbasaur      False         318      159      159       7
      1      2      ivysaur      False         405      202      203      10
      2      3     venusaur      False         525      262      263      20
      3      4   charmander      False         309      177      132       6
      ..   ...          ...        ...         ...      ...      ...     ...
      803  804    naganadel      False         540      321      219      36
      804  805    stakataka      False         570      197      373      55
      805  806  blacephalon      False         570      385      185      18
      806  807      zeraora       True         600      357      243      15

           Weight    Type 1  Type 2
      0        69    poison   grass
      1       130    poison   grass
      2      1000    poison   grass
      3        85      fire     NaN
      ..      ...       ...     ...
      803    1500    dragon  poison
      804    8200     steel    rock
      805     130     ghost    fire
      806     445  electric     NaN

      [807 rows x 10 columns]
```

Dropping columns that are irrelevant to this section and splitting the dataframe into two dataframes for Legendary and NonLegendary Pokemon:

```
[36]: df_AD = df_poke.drop(columns = ["Height","Weight","Stat Total","Type 1","Type␣
       ↪2"])
      df_LegsEnd = df_AD[df_AD["Legendary"] == True]
      df_LegsDontEnd = df_AD[df_AD["Legendary"] == False]
      df_LegsEnd
```

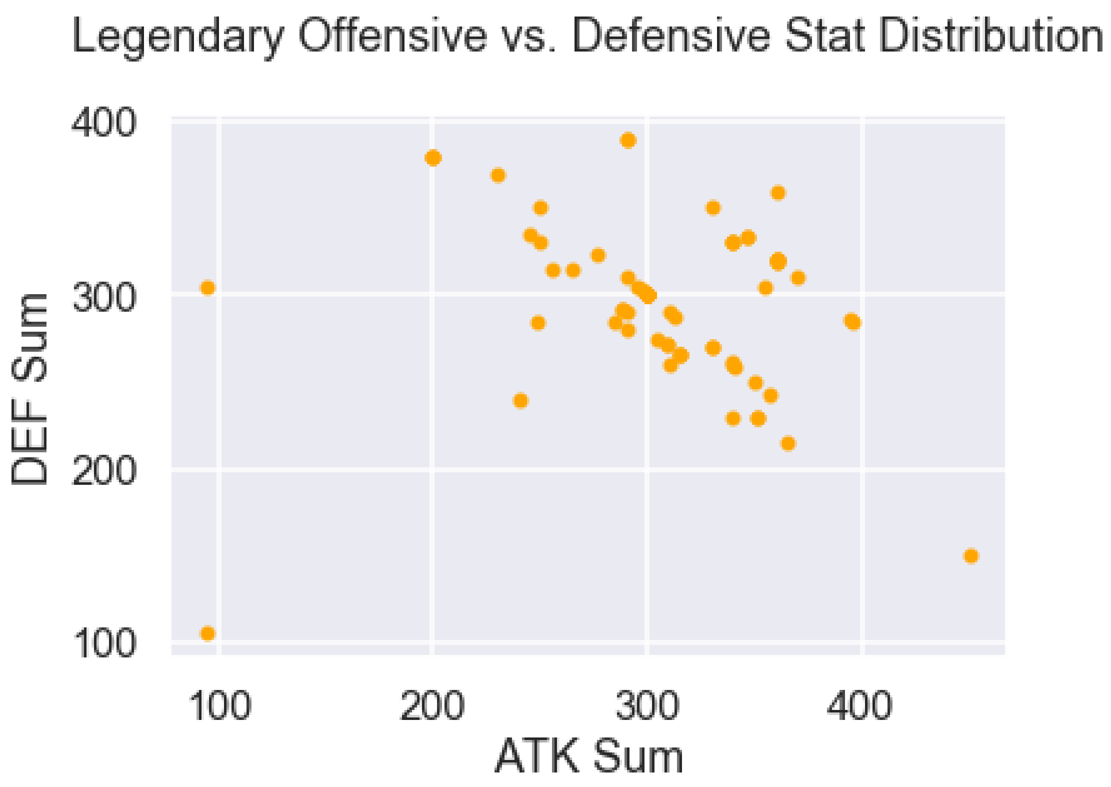
```
[36]:      ID    Pokemon  Legendary  ATK Sum  DEF Sum
      143  144   articuno       True      265      315
      144  145     zapdos       True      315      265
      145  146    moltres       True      315      265
      149  150     mewtwo       True      394      286
      ..   ...        ...        ...      ...      ...
      799  800   necrozma       True      313      287
      800  801   magearna       True      290      310
      801  802  marshadow       True      340      260
      806  807    zeraora       True      357      243
```

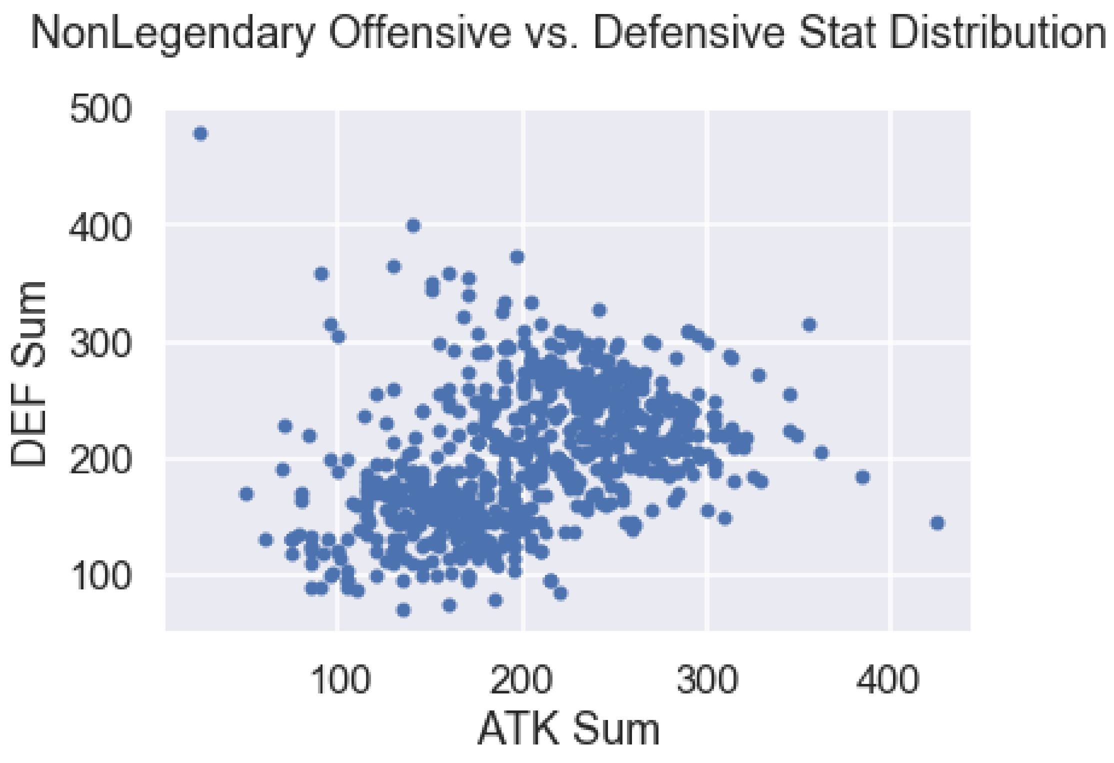
```
[68 rows x 5 columns]
```

12.0.1 Visualizing Pokemon Distribution by Stats

Now that the data is organized distinctly, let's visualize the pokemon in scatterplots with their offensive stat totals on the x-axis and their defensive stat totals on the y-axis.

[37]:
```
df_LegsEnd.plot.scatter(x="ATK Sum", y="DEF Sum", c="orange")
plt.title("Legendary Offensive vs. Defensive Stat Distribution", pad=(25))
df_LegsDontEnd.plot.scatter(x="ATK Sum", y="DEF Sum", c="b")
plt.title("NonLegendary Offensive vs. Defensive Stat Distribution", pad=(25))
plt.show()
```

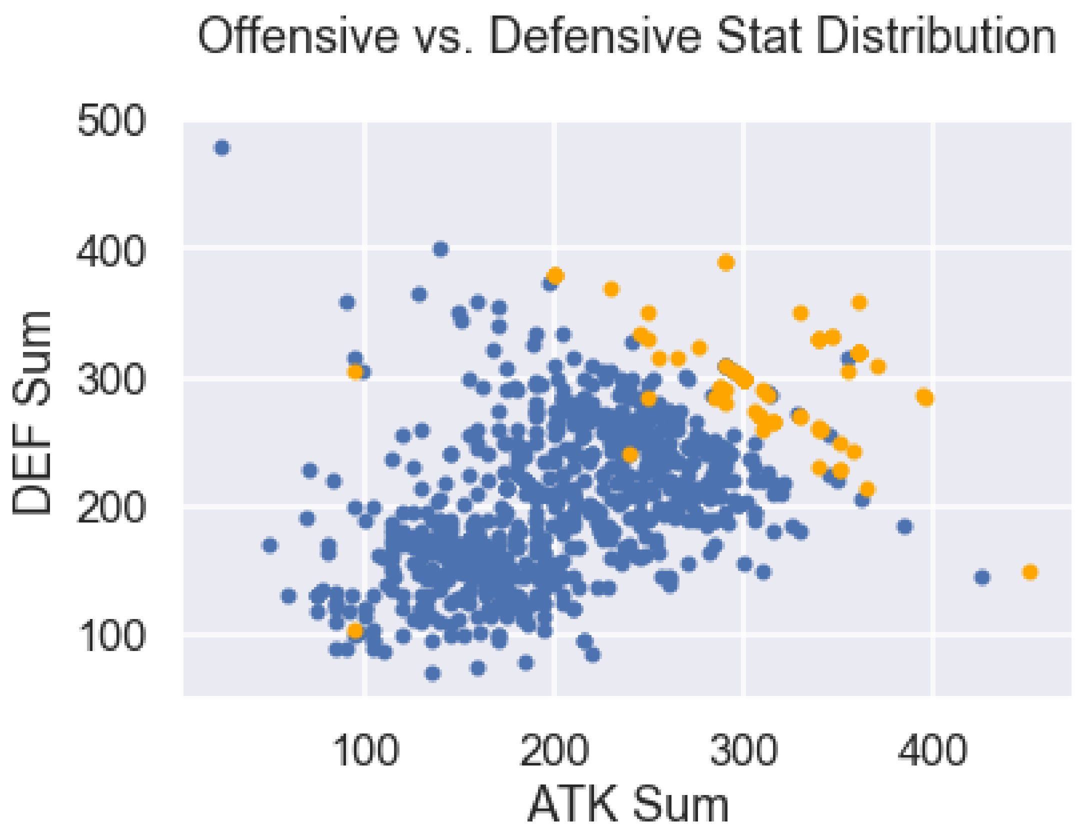




In the two scatterplots above, we can observe the distribution of offensive and defensive stats on the Legendary and NonLegendary Pokemon separately. In the first scatterplot, we can already see that Legendary Pokemon tend to lean in the upper range for both offsensive and defensive stats, with the exception of a couple of outliers. In the second scatterplot, we can observe that the majority of the NonLegendary Pokemon cluster in a lower range than the Legendary Pokemon.

Now let's see how the two groups look in the same scatterplot:

[38]:
```
a = df_LegsDontEnd.plot.scatter(x="ATK Sum", y="DEF Sum", c="b")
b = df_LegsEnd.plot.scatter(x="ATK Sum", y="DEF Sum", c="orange", ax = a)
plt.title("Offensive vs. Defensive Stat Distribution", pad=(25))
plt.show()
```



After stacking the two scatterplots together we can observe that the majority of the Legendary Pokemon (yellow) tend to have greater stats all around. Except for a few exceptions, almost all of the Legendary Pokemon data points are in the upper right quadrant of the scatterplot such that they either have great defensive stats or greater offensive stats or even both. Only a handful of the NonLegendary Pokemon data points are in the same class as the Legendary Pokemon.

However, despite that distinct difference between the two categories of Pokemon, they both seem to have populations that do not heavily skew towards either axis.

Now, to visualize their stats as a whole and comparing their distributions, let's create another data frame to handle the "Stat Total" data and drop irrelevant columns.

[39]:
```
df_total = df_poke.drop(columns = ["Height","Weight","DEF Sum","ATK Sum","Type␣
↪1","Type 2"])
df_total
```

[39]:
```
      ID     Pokemon  Legendary  Stat Total
0      1   bulbasaur      False         318
1      2     ivysaur      False         405
2      3    venusaur      False         525
3      4  charmander      False         309
..   ...         ...        ...         ...
803  804   naganadel      False         540
804  805   stakataka      False         570
```

```
805  806  blacephalon      False          570
806  807      zeraora       True          600

[807 rows x 4 columns]
```

Here we'll separate the data frame into two parts: Legendary and NonLegendary:

[40]:
```
df_legtotal = df_total[df_AD["Legendary"] == True]
df_legtotal
```

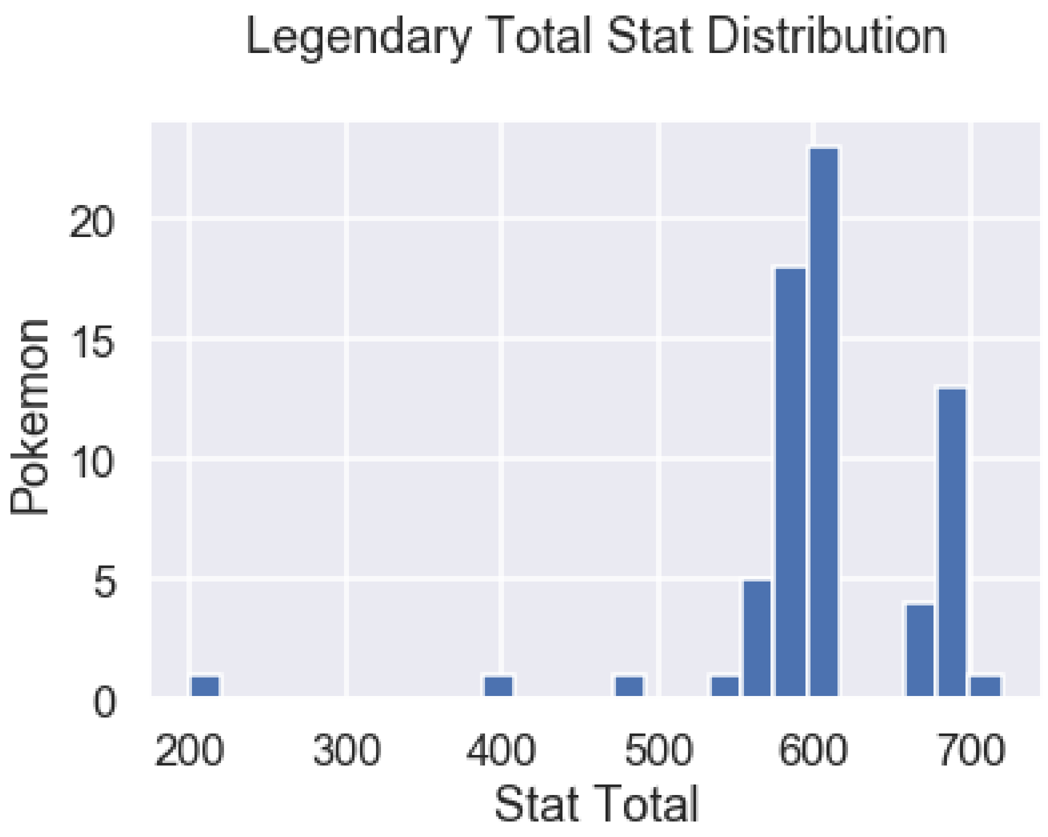
[40]:
```
      ID    Pokemon  Legendary  Stat Total
143  144   articuno       True         580
144  145     zapdos       True         580
145  146    moltres       True         580
149  150     mewtwo       True         680
..   ...        ...        ...         ...
799  800   necrozma       True         600
800  801   magearna       True         600
801  802  marshadow       True         600
806  807    zeraora       True         600

[68 rows x 4 columns]
```

[41]:
```
df_nontotal = df_total[df_AD["Legendary"] == False]
df_nontotal
```

[41]:
```
      ID      Pokemon  Legendary  Stat Total
0      1    bulbasaur      False         318
1      2      ivysaur      False         405
2      3     venusaur      False         525
3      4   charmander      False         309
..   ...          ...        ...         ...
802  803      poipole      False         420
803  804     naganadel      False         540
804  805     stakataka      False         570
805  806  blacephalon      False         570

[739 rows x 4 columns]
```

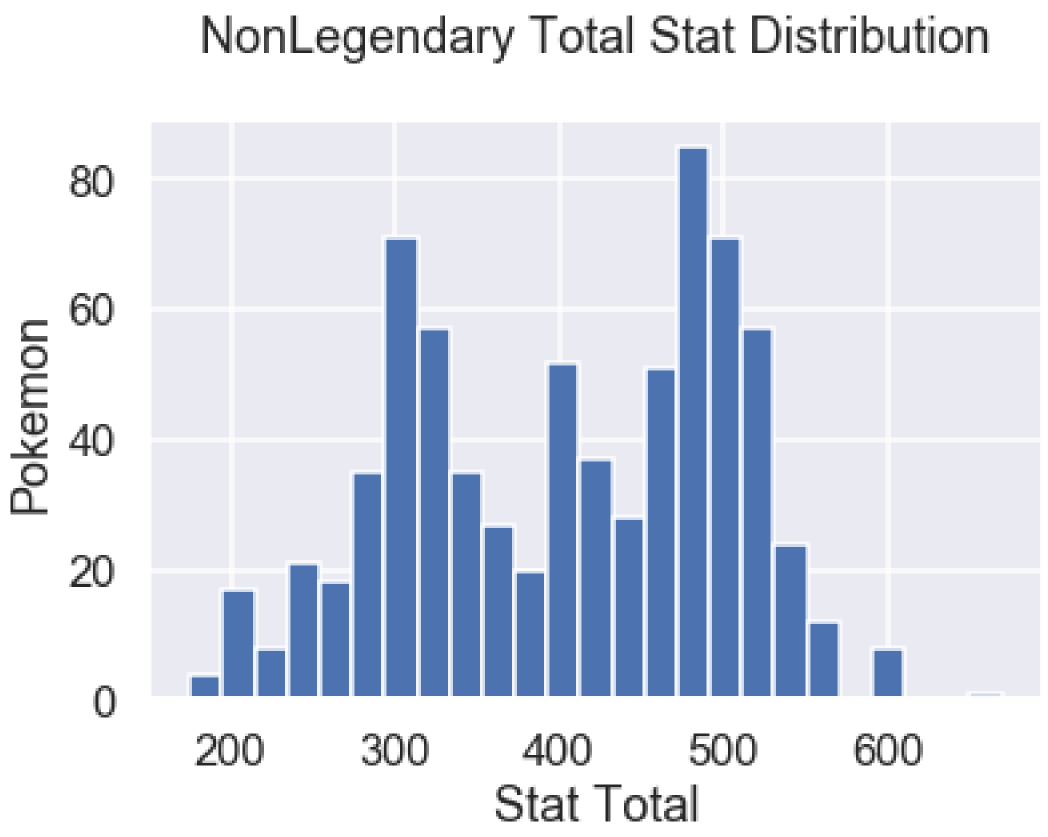
Now with the data separated, let's visualize the distributions using histograms:

[42]:
```
df_legtotal["Stat Total"].hist(bins = 25)
plt.xlabel("Stat Total")
plt.ylabel("Pokemon")
plt.title("Legendary Total Stat Distribution", pad=(25))
plt.show()
```



In this first histogram, we plotted the distribution of Legendary Pokemon with respect to the total sum of all their core stats. Most notably, we observe that a large number of Legendary Pokemon have stat totals of around 600 and even a significant number around 700. Furthermore, we can observe that there is an outlier data point with a stat total of around 200. Altogether, the histogram shows that Legendary Pokemon have stat totals that skews to the right of the histogram, with notably high stats.

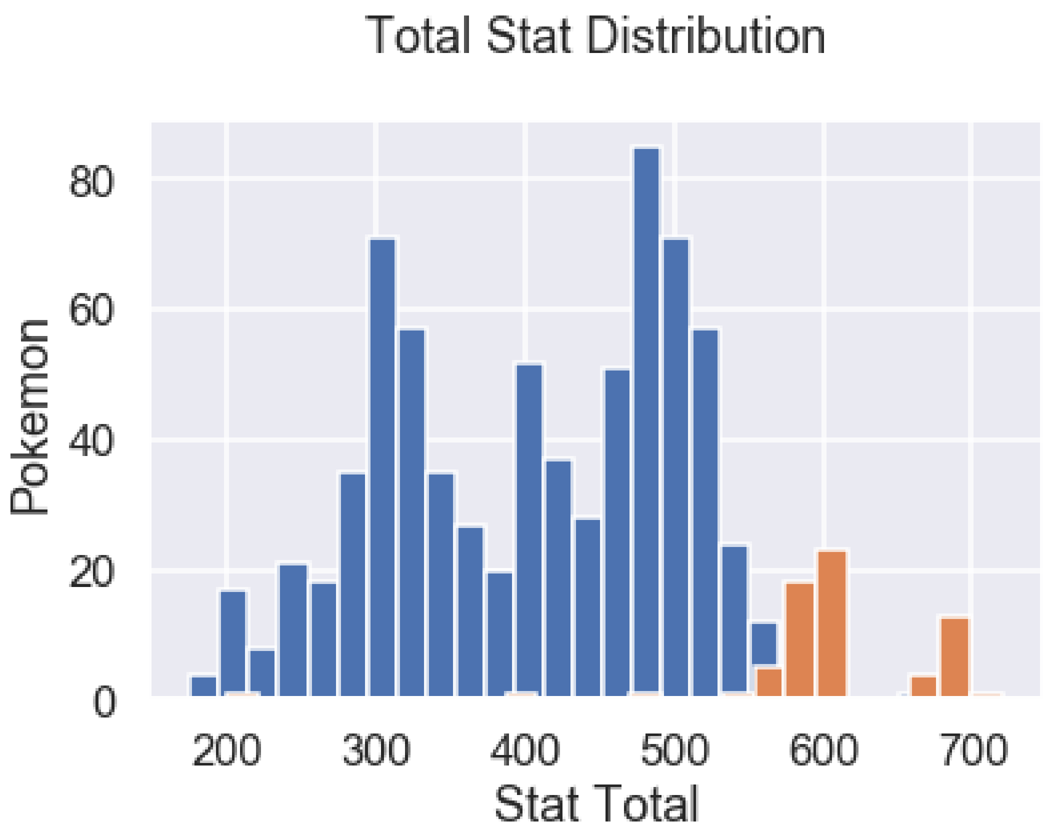
[43]:
```
df_nontotal["Stat Total"].hist(bins = 25)
plt.xlabel("Stat Total")
plt.ylabel("Pokemon")
plt.title("NonLegendary Total Stat Distribution", pad=(25))
plt.show()
```



In this second histogram, we plotted the distribution of NonLegendary Pokemon with respect to the total sum of all their core stats. Here, we can observe that the distribution is not quite normal with its two peaks at roughly 300 and 500 and a fall-off on either end that do not have any extremeley explicit outliers. The distribution seems to be wide and does not skew in either direction in particular.

Now let's see how they compare together on the same histogram:

[44]:
```
a1 = df_nontotal["Stat Total"].hist(bins = 25)
b1 = df_legtotal["Stat Total"].hist(bins = 25, ax = a1)
plt.title("Total Stat Distribution", pad=(25))
plt.xlabel("Stat Total")
plt.ylabel("Pokemon")
plt.show()
```



In this histogram, the two categories of Pokemon are plotted together with blue representing the NonLegendary Pokemon and orange representing the Legendary Pokemon. In this histogram, the data just barely overlaps and we chose to have the Legendary population at the front because it has a smaller population and infringes less in the overlap. Here we see that the vast majority of the Legendary Pokemon distribution is significantly right skewed are almost completely to the right of the NonLegendary Pokemon distribution.

To further examine the magnitude of their difference let's get their averages amost the entire population and visualize the comparison:

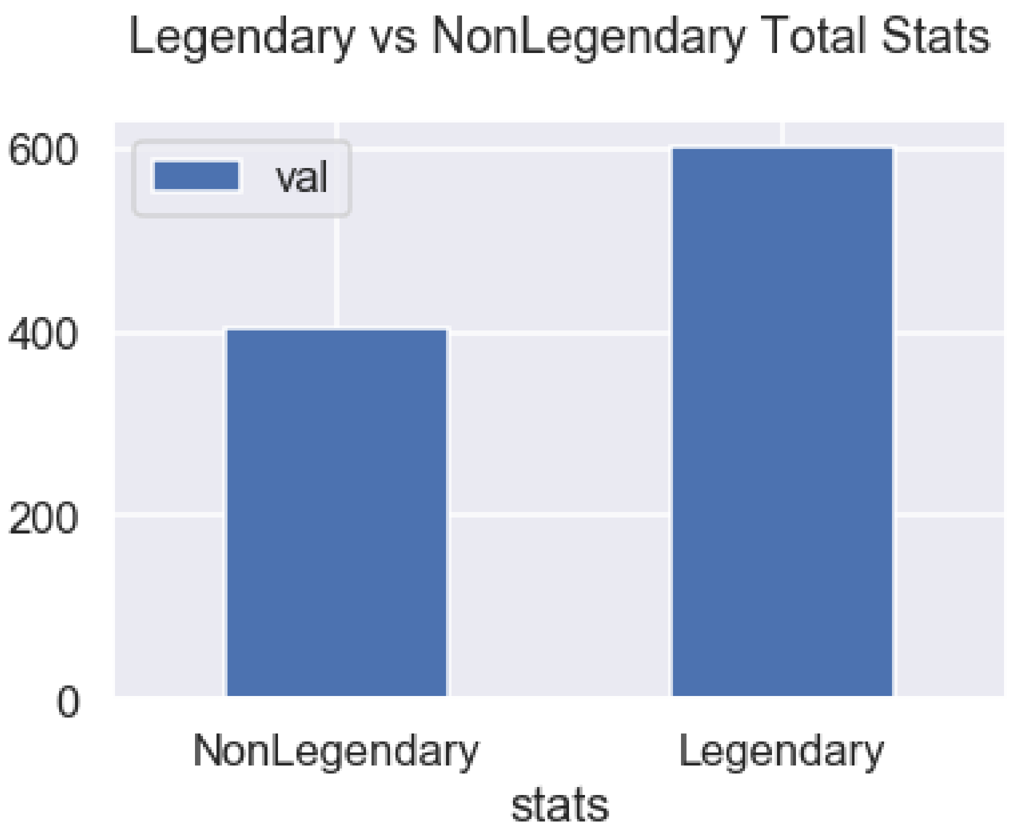
[45]:
```python
numNon = len(df_nontotal["Stat Total"])
numLeg = len(df_legtotal["Stat Total"])
sumNon = df_nontotal.sum(axis = 0)[3]
sumLeg = df_legtotal.sum(axis = 0)[3]
avgNon = sumNon / numNon
avgLeg = sumLeg / numLeg
print(avgNon)
print(avgLeg)
```

```
405.32205683355886
601.9705882352941
```

[46]:
```
sumNonA = df_LegsDontEnd.sum(axis = 0)[3]
sumNonD = df_LegsDontEnd.sum(axis = 0)[4]
sumLegA = df_LegsEnd.sum(axis = 0)[3]
sumLegD = df_LegsEnd.sum(axis = 0)[4]
avgNonA = sumNonA / numNon
avgNonD = sumNonD / numNon
avgLegA = sumLegA / numLeg
avgLegD = sumLegD / numLeg

print(avgNonA)
print(avgNonD)
print(avgLegA)
print(avgLegD)
```

```
202.68606224627877
202.6359945872801
306.1470588235294
295.8235294117647
```

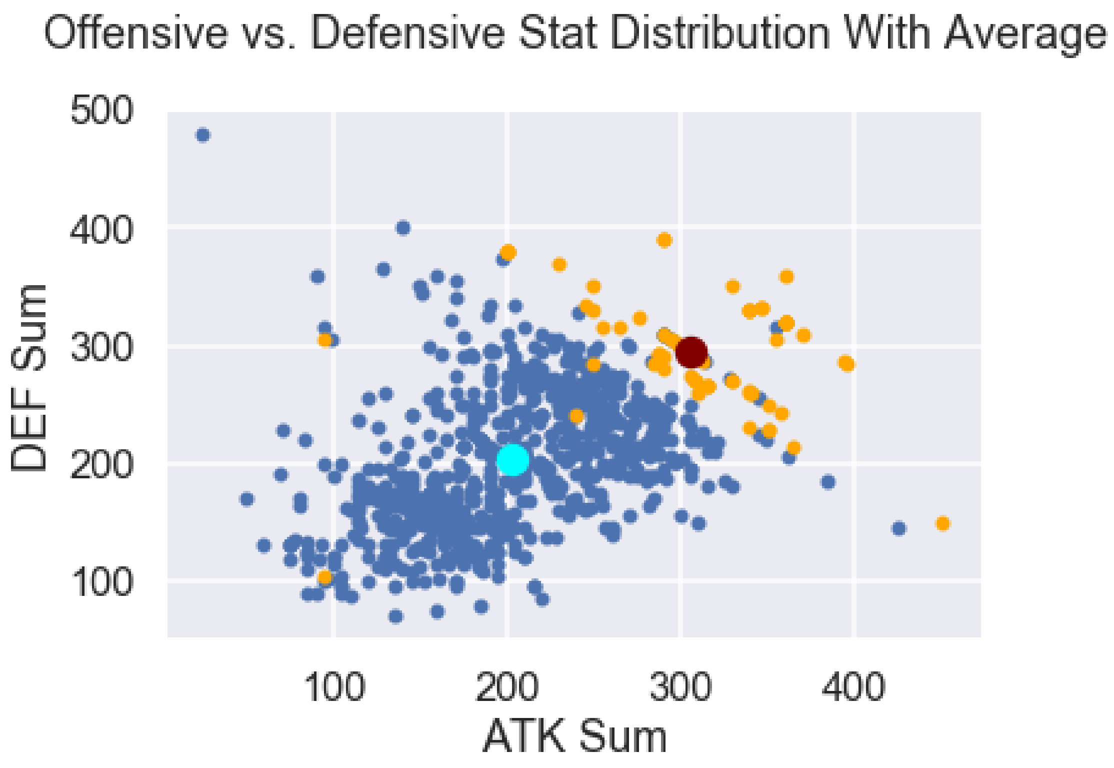
[47]:
```
df = pd.DataFrame({'stats':['NonLegendary', 'Legendary'], 'val':[avgNon,␣
↪avgLeg]})
ax = df.plot.bar(x='stats', y='val', rot=0)
plt.title("Legendary vs NonLegendary Total Stats", pad=(25))
plt.show()
```



We've calculated the average NonLegendary stat total to be 405 and the average Legendary stat total to be 602. For their offsensive and defensive breakdowns we have 203 and 203, and 306 and 296 respectively. As shown in the bar graph, Legendary Pokemon are found to have approximately 50% greater stats than NonLegendary Pokemon, truly earning them a "legendary" status.

Now with their average offensive and defensive stats, let's revisit the scatterplot and add in points to show where their averages lie:

[48]:
```
df_avgLeg = pd.DataFrame({"ID":["999"], "Pokemon":["Average Legendary"],␣
 ↪"Legendary":["True"], "ATK Sum":[avgLegA], "DEF Sum":[avgLegD]})
df_avgNon = pd.DataFrame({"ID":["999"], "Pokemon":["Average NonLegendary"],␣
 ↪"Legendary":["False"], "ATK Sum":[avgNonA], "DEF Sum":[avgNonD]})

aaa = df_LegsDontEnd.plot.scatter(x="ATK Sum", y="DEF Sum", c="b")
bbb = df_LegsEnd.plot.scatter(x="ATK Sum", y="DEF Sum", c="orange", ax = aaa)
ccc = df_avgNon.plot.scatter(x="ATK Sum", y="DEF Sum", c="cyan", s=140, ax =␣
 ↪aaa)
ddd = df_avgLeg.plot.scatter(x="ATK Sum", y="DEF Sum", c="maroon", s=140, ax =␣
 ↪aaa)
plt.title("Offensive vs. Defensive Stat Distribution With Average", pad=(25))
plt.show()
```



After adding a big cyan data point to represent the average NonLegendary Pokemon and a big maroon data point to represent the average Legendary Pokemon, we can easily see how Legendary Pokemon tend to be roughly 50% further from the origin (0,0) point in the scatterplot.

With such a well-defined difference in stats in favor of Legendary Pokemon, it's no surprise they are referred to as such and tend to be more limited and significant.

13 Conclusion

Usage-wise, legendary Pokémon are significantly more popular than other non-legendary Pokémon, despite being outnumbered in count. Overall, legendary Pokémon usage is significantly higher compared to the non-legendary counterparts, despite the fact that non-legendary Pokémon outnumber them in count. This can be seen due to their increased base stats that we analyzed next.

A limitation of our usage analysis is that the only data we have readily available to us is the usage data of Pokémon used in online battles against other players. The Pokémon franchise spans multiple mediums: anime, movies, manga, video games, card games, and more. Gauging the popularity of the 800+ Pokémon species across the millions of fans of the various mediums would be a huge task that would involve a lengthy data collection process that would be beyond the scope of this project. Analyzing only the competitive usage allows us to compare popularity with the objective statistics of legendary and non-legendary Pokémon.

In terms of height and weight, the average legendary Pokémon stands 2.1 metres tall and weighs 171.7 kilograms, while the average non-legendary Pokémon stands 1.1 metres tall and weighs 51.7 kilograms. We can conclude that the average legendary Pokémon is likely to be about 2 times taller and at least 3 times heavier than the average non-legendary Pokémon.

Typingwise, we see that while legendary Pokémon do appear often in the most populated typing, there are some significant outliers (Psychic and Poison). An interesting tidbit is that Psychic Pokémon were extremely strong in the early generations, only to grow progressively weaker as typing with strong matchups were released; Psychic still maintains a subjectively strong moveset however. Dragon is also weighted highly, being only weak to Ice and other Dragon types, and deals at least full damage to all types except Steel (and the newly released Fairy type). This lead us to believe that the developers tried to make legendary Pokémon relatively strong or above average versus the majority of Pokémon, and improve their image of being "legendary".

As we predicted, legendary Pokémon have higher average attack and defense stats than non-legendary Pokémon. Looking at the Pokémon's total stats, an amalgamation of all 6 of their stat distributions, we can see that legendary Pokémon are, on average, 50% stronger than non-legendary Pokémon. Another interesting observation is that across both non-legendary and legendary Pokémon, the total attack and defense are about equal. This indicates that both Pokémon categories are fairly evenly distributed among those that specialize in attack and those in defense. We are unsure if this is an intentional design choice by the creators, but it is an interesting result nonetheless.