RELAZIONE PER IL MODULO DI ANALISI DI DATI SAVINI RUDY

INTRODUZIONE:

1. Spiegazione

Per il modulo di "**Python - Data Analisi**" del corso **ITS 2021/2022** si è scelto di far eseguire ai propri studenti un progetto che riguardasse l'analisi dei dati di uno o più DataSet presi dal sito di Keggle.
Il linguaggio di programmazione scelto per questo modulo è **Python** con le seguenti librerie:

- Standard Library Python
- Numpy
- Pandas
- Matplotlib
- Seaborn
- SQLAlchemy

Abbinato a Python, è stato scelto di usare Jupyter Notebook per la prova rapida del codice, in particolare, integrato con VSCode e PyCharm Professional.
Il progetto dev'essere eseguito seguendo alcune linee guida dettate dal Prof. **Grandi**.

2. Scelta del Data Set

Il tema sul quale ho deciso di lavorare per questo progetto riguarda una passione che ho fin da quando ero bambino: i Pokémon. Ho deciso di cercare, tramite Keggle, un dataset che mi potesse soddisfare, quindi ho scelto "*Complete Pokémon Dataset (Updated 16.04.21)*" di *Mario Tormo Romero* che ho integrato successivamente con "*All Pokémon Dataset*" di *Mario Hernandez*.

3. Operazioni da condurre

Per quanto riguarda la manipolazione del dataset, oltre che prendere spunto da un Notebook relativo al primo dataset (credits to BryanB), mi sono anche chiesto, per pura curiosità, quali sarebbero stati i dati che avrei voluto visualizzare. Di seguito ci sono tutte le operazioni che sono andato ad eseguire sul Dataset.

- Quanti Pokémon sono stati rilasciati per Generazione.
- Confronto dell'incremento di Pokémon in ogni generazione rispetto a quella precedente.
- Confrontare l'incremento totale di Pokémon con l'incremento in ogni Generazione.
- Ricerca del tipo primario e secondario di Pokémon più comune.
- Ricerca Pokémon di tipo morfologico (leggero, pesante, veloce, alto, basso).
- Confronto dei Punti Totali dei Pokémon più forti a con i più deboli per ogni generazione.

- Confronto fra il Pokémon più forte in assoluto e quello più debole in assoluto.
- Confronto del miglior Pokémon “Leggendario”, “Normale”, “Mitico”, con quello. peggiore.
- Comparazione per Generazione di tutte le prime evoluzioni degli starter.
- Quantità di Pokémon per Esperienza totale al raggiungimento del livello 100.
- Quanti Pokémon hanno abilità nascoste.
- Confronto tra due Pokémon per vedere qual è il più forte.

Per molte di queste operazioni, ho prodotto dei grafici tramite la libreria di Matplotlib, che variano dai più classici grafici a torta a quelli più complessi come grafici a Radar (o Spider).

PREMESSA:

Premettendo che i dati presi dai due Datasets utilizzati siano giusti, e che si abbia un minimo di dimestichezza col “mondo Pokémon”, passiamo alla spiegazione delle operazioni che ho eseguito per questo progetto:

SVOLGIMENTO:

Per rendere la relazione più snella, ho omesso il codice che, eventualmente, può essere recuperato nel file di Jupyter, riordinato come in relazione. La maggior parte dei tentativi fatti durante la scrittura del codice, o di eventuali parti che hanno subito refactor, sono nel file di jupyter sottoforma di commento, per far capire il processo logico.

1. Inizializzazione del progetto

Come sopracitato, mi sono servito di Jupyter integrato con VSCode (o Codium) e PyCharm Professional. In parallelo alla creazione dei primi file, ho anche inizializzato una repository per avere più flessibilità con diversi dispositivi, più controllo per quanto riguarda il workflow del progetto e, soprattutto, per tener traccia di tutti i cambiamenti eseguiti, cosicché in caso di errori, potessi tornare alle versioni precedenti del codice. Come prima cosa, ho importato le librerie basi per analisi dati in Python.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Fruizione dei primi dati

Come secondo passo, ho iniziato installando le librerie necessarie per questo progetto.
Come prima cosa, ho iniziato a leggere il mio Dataset in formato CSV e secondariamente l'ho convertito in un DataFrame.
Essendo abbastanza lungo e con qualche colonna che non mi serviva, ho filtrato solo le colonne che mi interessavano creando un nuovo DataFrame, un po' più compatto.

```
col_names = [
    'pokedex_number', 'name', 'generation',
    'status', 'type_number', 'type_1',
    'type_2', 'height_m', 'weight_kg',
    'abilities_number', 'ability_1', 'ability_2',
    'ability_hidden', 'total_points', 'hp',
    'attack', 'defense', 'sp_attack',
    'sp_defense', 'speed', 'base_experience',
]
filtered_pokemon_data = pd.DataFrame(df, columns=col_names)
filtered_pokemon_data.head(5)
```

✓ 0.9s — Python

	pokedex_number	name	generation	status	type_number	type_1	type_2	height_m	weight_kg	abilities_number	...	ability_2	ability_hidden	total_points	hp	attack	defense	sp_attack	sp_defense	speed	base_experience
0	1	Bulbasaur	1	Normal	2	Grass	Poison	0.7	6.9	2	...	NaN	Chlorophyll	318.0	45.0	49.0	49.0	65.0	65.0	45.0	64.0
1	2	Ivysaur	1	Normal	2	Grass	Poison	1.0	13.0	2	...	NaN	Chlorophyll	405.0	60.0	62.0	63.0	80.0	80.0	60.0	142.0
2	3	Venusaur	1	Normal	2	Grass	Poison	2.0	100.0	2	...	NaN	Chlorophyll	525.0	80.0	82.0	83.0	100.0	100.0	80.0	236.0
3	3	Mega Venusaur	1	Normal	2	Grass	Poison	2.4	155.5	1	...	NaN	NaN	625.0	80.0	100.0	123.0	122.0	120.0	80.0	281.0
4	4	Charmander	1	Normal	1	Fire	NaN	0.6	8.5	2	...	NaN	Solar Power	309.0	39.0	52.0	43.0	60.0	50.0	65.0	62.0

```
df_2 = pd.read_csv('CSV/All_Pokemon.csv', index_col=0)
df_2.reset_index(inplace=True)
df_3 = pd.DataFrame(df_2['Experience to level 100'])
filtered_pokemon_data = pd.merge(filtered_pokemon_data, df_3, left_on=filtered_pokemon_data.index, right_index=True)

# --- TRY FATTI E FALLITI ---
# filtered_pokemon_data
# filtered_pokemon_data.insert(22, column='experience_to_100', value=experience_to_100[''])
# -- END TRY FATTI E FALLITI --
filtered_pokemon_data.to_csv('CSV/filtered_pokemon_data_years.csv', index=False)
```

A questo punto, per avere un ordine cronologico più comprensibile, inserisco con la funzione **.insert()** gli anni di pubblicazione dei Pokémon in base alla loro generazione, facendo una operazione vettoriale su tutto il Data Frame tramite il metodo **.map()**. Inserisco la colonna, chiamata "date_published" passando come valore, al metodo **.map()**, il mapping dato dal dizionario creato con enumerate (con lista di anni e index).

```
# Linking genreration number to publish date (JP)
years = [1996, 1999, 2002, 2006, 2010, 2013, 2016, 2019]

# inserisco la colonna (sarà la 3°) che si chiamerà date_published passandoli come valore il mapping dato dal
# dizionario creato con enumerate (con lista di anni e index)
filtered_pokemon_data.insert(2, column='date_published', value=filtered_pokemon_data.generation.map(dict(enumerate(years, start=1))))
```

A questo punto, prendo la colonna che mi interessa dal secondo Dataset, ovvero "Experience to level 100" e la unisco al DataFrame già esistente. Dal merge ricavo il Dataset completo sul quale lavorerò.
Ora il DataFrame (filtered_Pokémon_data), si mostra in questo modo (alcune colonne sono nascoste per questioni di spazio):

	key_0	pokedex_number	name	date_published	generation	status	type_number	type_1	type_2	height_m	...	ability_hidden	total_points	hp	attack	defense	sp_attack	sp_defense	speed	base_experience	Experience to level 100
0	0	1	Bulbasaur	1996	1	Normal	2	Grass	Poison	0.7	...	Chlorophyll	318.0	45.0	49.0	49.0	65.0	65.0	45.0	64.0	1059860
1	1	2	Ivysaur	1996	1	Normal	2	Grass	Poison	1.0	...	Chlorophyll	405.0	60.0	62.0	63.0	80.0	80.0	60.0	142.0	1059860
2	2	3	Venusaur	1996	1	Normal	2	Grass	Poison	2.0	...	Chlorophyll	525.0	80.0	82.0	83.0	100.0	100.0	80.0	236.0	1059860
3	3	3	Mega Venusaur	1996	1	Normal	2	Grass	Poison	2.4	...	NaN	625.0	80.0	100.0	123.0	122.0	120.0	80.0	281.0	1059860
4	4	4	Charmander	1996	1	Normal	1	Fire	NaN	0.6	...	Solar Power	309.0	39.0	52.0	43.0	60.0	50.0	65.0	62.0	1059860
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1023	1023	888	Zacian Hero of Many Battles	2019	8	Legendary	1	Fairy	NaN	2.8	...	NaN	670.0	92.0	130.0	115.0	80.0	115.0	138.0	NaN	1250000
1024	1024	889	Zamazenta Crowned Shield	2019	8	Legendary	2	Fighting	Steel	2.9	...	NaN	720.0	92.0	130.0	145.0	80.0	145.0	128.0	NaN	1250000
1025	1025	889	Zamazenta Hero of Many Battles	2019	8	Legendary	1	Fighting	NaN	2.9	...	NaN	670.0	92.0	130.0	115.0	80.0	115.0	138.0	NaN	1250000
1026	1026	890	Eternatus	2019	8	Legendary	2	Poison	Dragon	20.0	...	NaN	690.0	140.0	85.0	95.0	145.0	95.0	130.0	NaN	1250000
1027	1027	890	Eternatus Eternamax	2019	8	Legendary	2	Poison	Dragon	100.0	...	NaN	1125.0	255.0	115.0	250.0	125.0	250.0	130.0	NaN	1250000

3. Analisi dei Dati Senza Grafici

Iniziamo con l'analisi morfologica ricercando i Pokémon **più leggeri**, **più pesanti**, **più veloci**, **più alti** e **più bassi**:

Pokémon più **Leggero**:

*Il peso minimo imposto da The Pokémon Company è di 0.1 KG.

Immagine presa da: https://www.pngitem.com

Pokédex Number	Name	Peso KG
776	Flabébé	0.1

Pokémon più **Pesante**:

*Il peso massimo imposto da The Pokémon Company è di 999.9 KG.

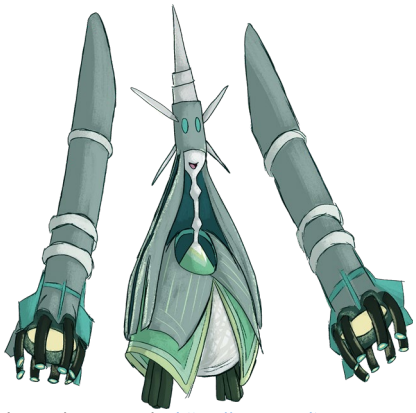


Immagine presa da: https://www.pngitem.com

Pokédex Number	Name	Peso KG
924	Celesteela	999.9

Pokémon più **Alto**:

*l'altezza massima imposta da The Pokémon Company è di 100.0 m.

Immagine presa da: https://www.pngitem.com

Pokédex Number	Name	Altezza m
1027	Eternatus Eternamx	100.0

Pokémon più Basso:

*l'altezza minima imposta da The Pokémon Company è di 0.1m.

Pokédex Number	Name	Altezza m
864	Cutiefly	0.1

Immagine presa da: https://www.pngitem.com

Pokémon più Veloce:

Pokédex Number	Name	Velocità
463	Deoxys Speed Forme	1.80

Immagine presa da: https://www.pngitem.com

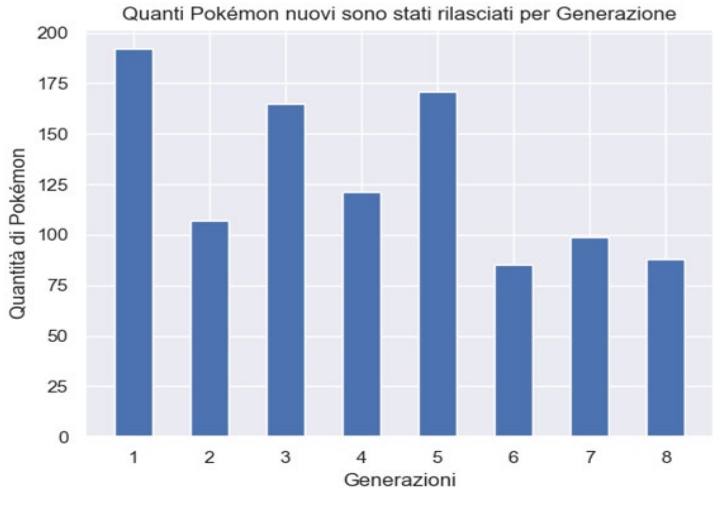
4. Analisi dei Dati con Grafici

In questo capitolo, andrò ad elencare tutte le manipolazioni dei dati che portano con loro anche la visualizzazione grafica degli stessi.

4.1. Quanti Pokémon sono stati rilasciati per ogni generazione:

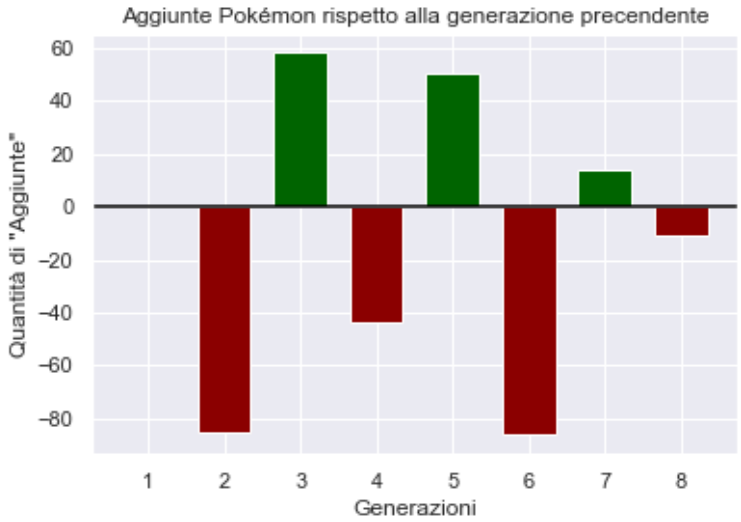
In questo grafico voglio analizzare quanti Pokémon sono stati rilasciati per generazione. Importo da *collections* la classe *Counter*, prendo tutti i valori della colonna "generation" e li metto in una lista. A seguire, passo la lista alla classe Counter che mi restituirà un dizionario avente le mie generazioni come chiavi (da 1 a 8) e come valore il totale di Pokémon per quella generazione, ordinati per valore.

```
Counter({1: 192, 5: 171, 3: 165, 4: 121, 2: 107, 7: 99, 8: 88, 6: 85})
```



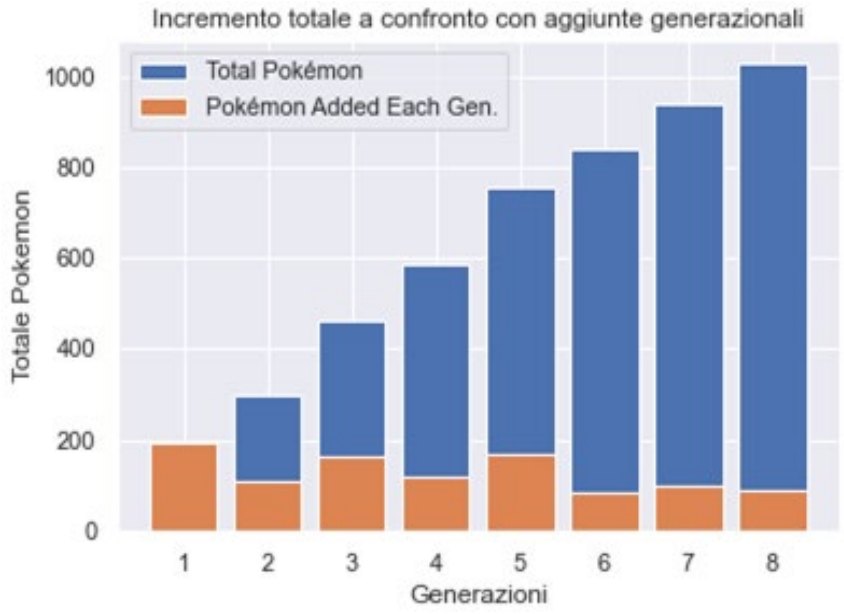
4.2. Confrontiamo l'incremento generazionale in relazione al totale dei Pokémon:

In questo punto voglio analizzare quante aggiunte ci sono state in ogni generazione rispetto a quella precedente, prendendo come punto zero la prima generazione.
Da notare la colorazione condizionale: verde se sono stati aggiunti più pokémon rispetto alla generazione precedente, rosso se ne sono stati aggiunti di meno.



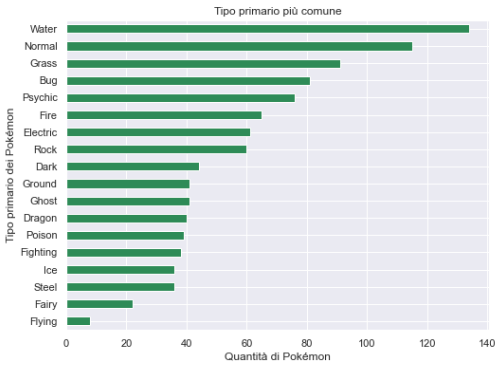
4.3. Confrontiamo l'incremento di Pokémon, con quanti ne sono stati aggiunti:

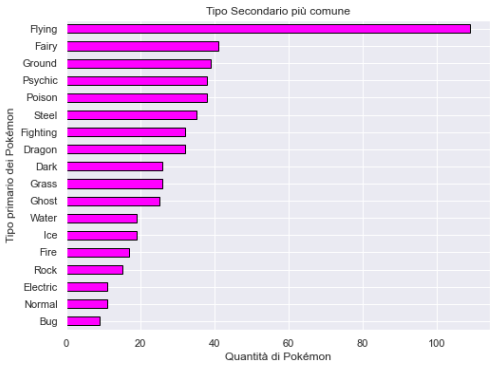
In questo caso, voglio andare a visualizzare, sovrapponendo due barre, i pokémon aggiunti in relazione alla quantità totale per generazione, quindi ho deciso di usare un grafico con barre sovrapposte.



4.4. Ricerchiamo i tipi primari e secondari più comuni:

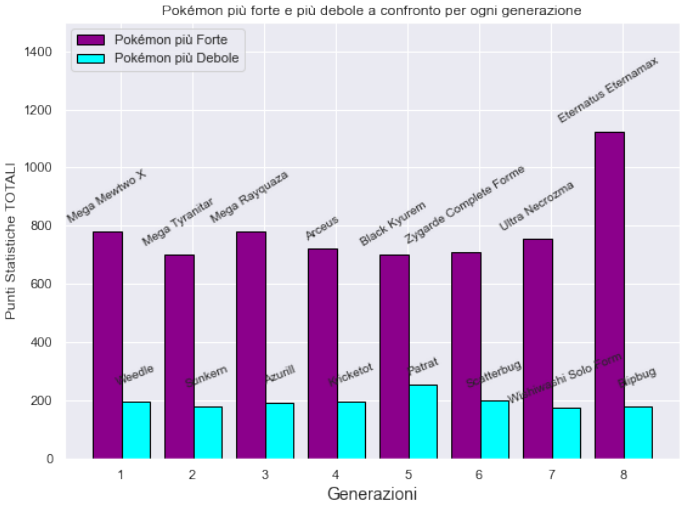
In questo caso voglio andare a visualizzare, su tutti i tipi disponibili, quali siano i tipi primari e secondari più comuni. Ho preso il mio DataFrame e ho eseguito un **groupby** per tipo (primario e secondario), dopodiché ho usato **sort_values** per ordinare i dati. È bene ricordare che non tutti i Pokémon hanno un tipo secondario.





4.5. Confronto Pokémon più forte e più debole per ogni generazione:

In questo grafico voglio visualizzare, in base ai "total_points", quali sono i Pokémon più forti e quelli più deboli per ogni generazione. Ho scelto di usare un grafico a barre raggruppate. Ci sono state molte modifiche al codice per renderlo più snello (i commenti sono nel codice), alla fine il risultato finale è questo:




```
from itertools import chain
# Ordino i pokemon per generazione e per punteggio totale
sorted_by_gen_and_total_points = filtered_pokemon_data.sort_values(by=['generation', 'total_points'], ascending=[True, False])

# creo un dizionario con chiavi che vanno da 1 a 8 (generazioni),
# i valori sono una lista con il migliore e il peggiore per ogni gen
my_dict = dict((k,
                [
                    *list(sorted_by_gen_and_total_points[sorted_by_gen_and_total_points['generation'] == k]['total_points'].head(1).values),
                    *list(sorted_by_gen_and_total_points[sorted_by_gen_and_total_points['generation'] == k]['total_points'].tail(1).values)
                ]
                ) for k in sorted_by_gen_and_total_points['generation'].unique())

# creo sempre un dizionario con chiavi (generazioni),
# i valori una lista con i nomi del + forte e il + debole
my_labels = dict((k,
                [
                    *list(sorted_by_gen_and_total_points[sorted_by_gen_and_total_points['generation'] == k]['name'].head(1).values),
                    *list(sorted_by_gen_and_total_points[sorted_by_gen_and_total_points['generation'] == k]['name'].tail(1).values)
                ]
                ) for k in sorted_by_gen_and_total_points['generation'].unique())

# Creo una serie per solo i valori e i nome a partire dai miei 2 dizionari
my_s = pd.Series(my_dict)
my_l = pd.Series(my_labels)

# inizializzo delle liste vuote d'appoggio per best - worst e le relative label
temp_b = []
temp_w = []
temp_b_l = []
temp_w_l = []
# aggiungo ai valori della lista temporanea, tutti i valori iterando sulle chiavi del mio dizionario
# inizializzando una nuova lista
best1 = [temp_b + [my_s[i][0]] for i in my_dict.keys()]
worst1 = [temp_w + [my_s[i][1]] for i in my_dict.keys()]
best_labels = [temp_b_l + [my_l[i][0]] for i in my_dict.keys()]
worst_labels = [temp_w_l + [my_l[i][1]] for i in my_dict.keys()]
```

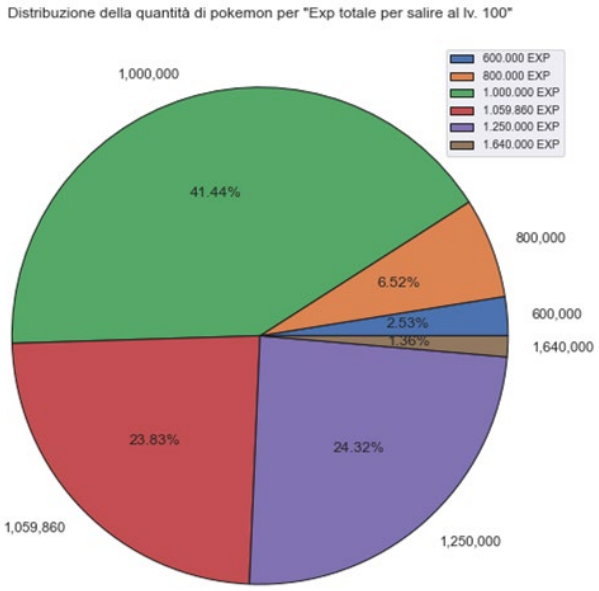
```
best_label_unpacked = list(chain(*best_labels))
worst_label_unpacked = list(chain(*worst_labels))

# -----GROUPED BAR CHART-----
col_width = 0.4

# prima barra è l'asse x (con le generazioni)(prendo la lunghezza di best 1 (0,7) +1
bar1 = np.arange(len(best1))+1
# da dove parte la barra 2 --> sommi alla posizione della prima colonna la larghezza della stessa e diventa
# la posizione dove disegnare la colonna 2
bar2 = [i + col_width for i in bar1]
ax.set_ylabel('Punti Statistiche TOTALI')
ax.set_xlabel('Generazioni', fontsize=14)
ax.set_title('Pokémon più forte e più debole a confronto per ogni generazione')
ax.set_xticks(bar1 + col_width/2.07, bar1)
ax.legend(["Pokémon più Forte", "Pokémon più Debole"], loc='upper left')
ax.set_ylim(0,1500)
ax.bar_label(rects1, labels=best_label_unpacked, padding=5, fontsize=10, rotation=32)
ax.bar_label(rects2, labels=worst_label_unpacked,
             padding=45, fontsize=10, label_type='center', rotation=22)
fig.tight_layout()
plt.savefig('images/weakest_strongest_pokemon_generation.png', dpi='figure', format='png')
plt.show()
```

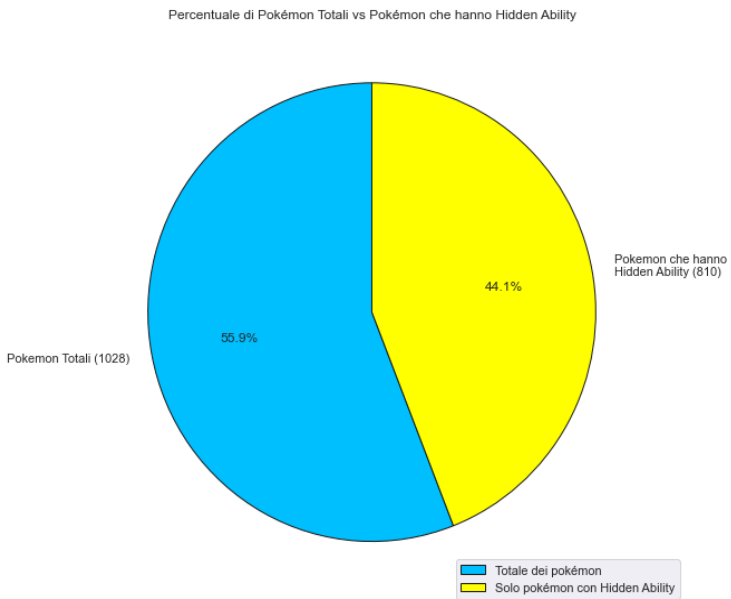
4.6. Quantità di Pokémon per Esperienza totale al livello 100:

Per questo punto, ho deciso di usare un grafico a torta. Sono partito dal mio DataFrame originario (filtered_Pokémon_data) e da lì ho contato i valori nella colonna “Experience to level 100”. Dopodiché ho fatto un reset degli indici e rinominato le colonne. Ho dato tutto in pasto a **plt.pie()** a cui ho passato il parametro **autopct** per formattare la percentuale.



4.7. Quantità di Pokémon con Abilità Nascosta

Per questo grafico ho scelto la tipologia “torta”.



Per questo punto mi sono servito della funzione **.dropna()**, visto che non tutti i pokémon hanno una “hidden ability” e, per avere il DataFrame un po’ più pulito e con meno entry, ho eliminato tutti i pokémon che non l’avevano, dopodiché ho proseguito a contare i valori con la funzione **.count()** e messo tutto nel **plt.pie()**.

4.8. Confrontare due Pokémon per vedere qual è il più forte

In questo punto voglio comparare due pokémon per vedere qual è il più forte. Per fare questo mi sono servito della universal function .greater()

```
lugia = filtered_pokemon_data.loc[filtered_pokemon_data['name'].str.contains(
    'Lugia')].total_points

bellsprout = filtered_pokemon_data.loc[filtered_pokemon_data['name'].str.contains(
    'Bellsprout')].total_points

lugia_only = np.array(lugia)
bellsprout_only = np.array(bellsprout)

print("Bellsprout è più debole di Lugia") if np.greater([bellsprout_only], [
    lugia_only]) == False else print("Bellsprout è più forte di Lugia")
```

```
Bellsprout è più debole di Lugia
```

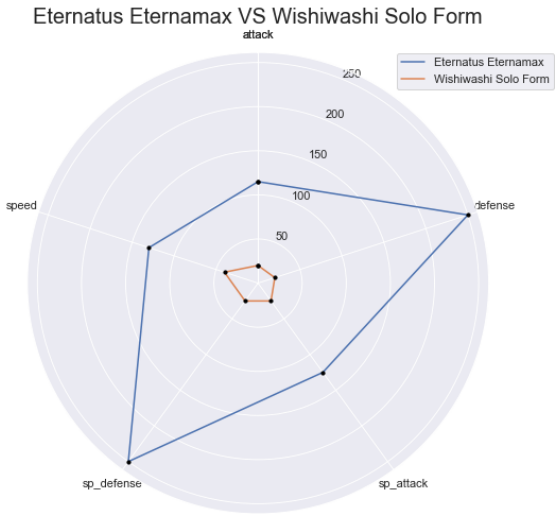
5. Elaborazioni con grafici più complessi:

Per l'elaborazione di questi diagrammi mi sono servito di alcuni snippet di codice per poter disegnare grafici Spider, o anche detti a "Radar".

Il procedimento, per il disegno del grafico (senza dati), comprende:

- Inizializzare una lista dove metterò le categorie da visualizzare.
- Duplicare il primo elemento della lista e metterlo all'ultimo posto della lista (per poter chiudere il cerchio del grafico).
- Creare le linee del grafico.
- Creare le linee di demarcazione per le unità (10, 20, 30…).
- Parametrizzare tutti i dati, dal titolo alle statistiche.

5.1. Migliore e peggiore Pokémon in assoluto a confronto:



Mi sono ispirato ai classici grafici nei videogiochi per mostrare le statistiche del personaggio.

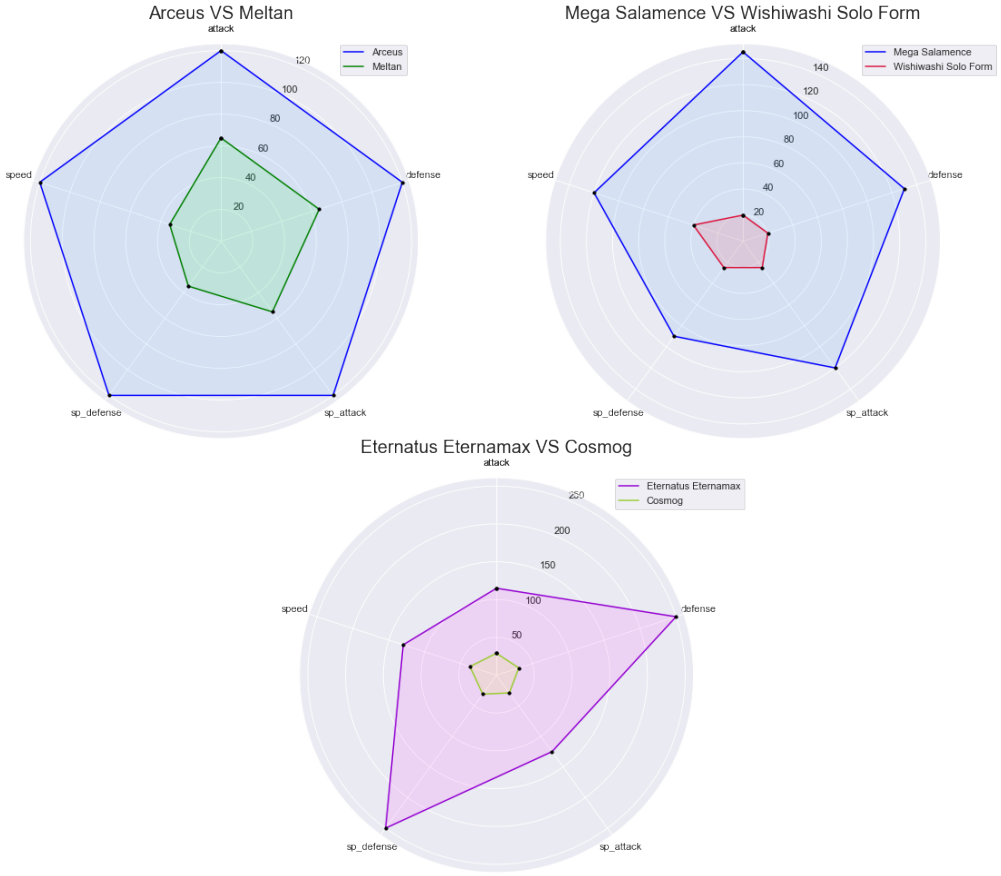
Ho confrontato le statistiche (punti base) del pokémon più forte in questo Dataset con quello più debole in termini di total_points, per rappresentare graficamente le loro 5 statistiche del gioco.

Così come per i punti, anche i nomi inseriti nel titolo del grafico sono stati presi dinamicamente dal codice, in modo da aggiornarli dinamicamente in caso vengano aggiornati e/o cambiati.

5.2. Migliore e peggiore per status:

In questo punto, raggruppo 3 grafici che mettono a confronto i Pokémon più forti e più deboli per "status" ovvero: **Mitico, Normale e Leggendario**.

Ho sempre utilizzato un grafico a Radar per poter visualizzare i dati, inoltre tutti i dati, compreso il titolo, sono parametrizzati, quindi non sono stati "hardcodati"



5.3. Comparazione per Generazione di tutte le prime evoluzioni degli starter:

Indubbiamente, questa è la parte che mi ha richiesto decisamente più tempo, quindi inserirò qualche immagine del codice che ho scritto (documentato tutto sul file di jupyter, compresi i tentativi fatti). L'intenzione originale era quella di effettuare, per ogni generazione, il confronto di tutte le evoluzioni di tutti e tre gli starter. Quest'idea l'ho abbandonata poco dopo, realizzando che sarebbe stato un lavoro troppo grande e dispendioso in termini di tempo. Passata l'idea della comparazione fra le evoluzioni, ho deciso di ridimensionare i calcoli e comparare tutte le "forme base" per ogni generazione.

Sono partito creandomi una lista di tutte le colonne del mio DataFrame che mi sarebbero servite. Successivamente ho creato un dizionario avente 8 chiavi e, come valori, la lista degli starter della generazione riferita alla chiave, e infine ho creato un nuovo DataFrame partendo dal dizionario.

```
categories_to_query_type = [
    'name', 'type_1', 'total_points',
    'attack', 'defense',
    'sp_attack', 'sp_defense',
    'speed'
]

# Creo un dizionario dove aggiungo per ogni generazione gli started (1st evolution)
my_generation_starter = {
    1: ['Bulbasaur', 'Squirtle', 'Charmander'],
    2: ['Chikorita', 'Totodile', 'Cyndaquil'],
    3: ['Treecko', 'Mudkip', 'Torchic'],
    4: ['Turtwig', 'Piplup', 'Chimchar'],
    5: ['Snivy', 'Oshawott', 'Tepig'],
    6: ['Chespin','Fennekin','Froakie'],
    7: ['Rowlet', 'Litten', 'Popplio'],
    8: ['Grookey', 'Scorbunny', 'Sobble']
}

starter_df= pd.DataFrame(my_generation_starter)
starter_df = starter_df.T

query_grass_starter = starter_df[0]
query_water_starter = starter_df[1]
query_fire_starter = starter_df[2]
```

Successivamente ho raggruppato tutti gli starter per genere in delle variabili e ho creato un nuovo DataFrame con l'unione delle 3 variabili.

```
# Mi creo un DataFrame con una maschera dove metto le mie categorie che voglio nel nuovo DF
my_filtered_df = filtered_pokemon_data[categories_to_query_type]


# Raggruppo tutti gli starters per tipo in delle variabili
my_grass_starters = my_filtered_df.loc[(my_filtered_df['name'].isin(query_grass_starter))]
my_fire_startes = my_filtered_df.loc[(my_filtered_df['name'].isin(query_fire_starter))]
my_water_startes = my_filtered_df.loc[(my_filtered_df['name'].isin(query_water_starter))]

# Creo un nuovo DF che sarà l'unione delle 3 variabili sopracitate
all_starters_merged = pd.concat([my_grass_starters, my_water_startes,
                  my_fire_startes]).sort_index(kind='mergesort')
```

Il risultato è un DataFrame ordinato come nel pokédex, escludendo varie forme alternative dei Pokémon.

	name	type_1	total_points	attack	defense	sp_attack	sp_defense	speed
0	Bulbasaur	Grass	318.0	49.0	49.0	65.0	65.0	45.0
4	Charmander	Fire	309.0	52.0	43.0	60.0	50.0	65.0
9	Squirtle	Water	314.0	48.0	65.0	50.0	64.0	43.0
192	Chikorita	Grass	318.0	49.0	65.0	49.0	65.0	45.0
195	Cyndaquil	Fire	309.0	52.0	43.0	60.0	50.0	65.0
198	Totodile	Water	314.0	65.0	64.0	44.0	48.0	43.0
299	Treecko	Grass	310.0	45.0	35.0	65.0	55.0	70.0
303	Torchic	Fire	310.0	60.0	40.0	70.0	50.0	45.0
307	Mudkip	Water	310.0	70.0	50.0	50.0	50.0	40.0

464	Turtwig	Grass	318.0	68.0	64.0	45.0	55.0	31.0
467	Chimchar	Fire	309.0	58.0	44.0	58.0	44.0	61.0
470	Piplup	Water	314.0	51.0	53.0	61.0	56.0	40.0
586	Snivy	Grass	308.0	45.0	55.0	45.0	55.0	63.0
589	Tepig	Fire	308.0	63.0	45.0	45.0	45.0	45.0
592	Oshawott	Water	308.0	55.0	45.0	63.0	45.0	45.0
756	Chespin	Grass	313.0	61.0	65.0	48.0	45.0	38.0
759	Fennekin	Fire	307.0	45.0	40.0	62.0	60.0	60.0
762	Froakie	Water	314.0	56.0	40.0	62.0	44.0	71.0
841	Rowlet	Grass	320.0	55.0	55.0	50.0	50.0	42.0
844	Litten	Fire	320.0	65.0	40.0	60.0	40.0	70.0
847	Popplio	Water	320.0	54.0	54.0	66.0	56.0	40.0
940	Grookey	Grass	310.0	65.0	50.0	40.0	40.0	65.0
943	Scorbunny	Fire	310.0	71.0	40.0	40.0	40.0	69.0
946	Sobble	Water	310.0	40.0	40.0	70.0	40.0	70.0

Successivamente, ho calcolato per ogni generazione:

- La generazione corrente.
- I nomi dei tre Pokémon della generazione corrente.
- Le statistiche base (5 + 1) di ogni Pokémon.

Nella prima versione, i calcoli erano abbastanza prolissi, visto che ripetevo le stesse operazioni per ben otto volte, quindi ho creato una funzione parametrizzata che mi facesse i calcoli e restituisse la lista di nomi e la lista delle statistiche.

```
# grazie anche a Martino :)
def get_gen_name_and_points(df, start=0, end=0):
    current_gen = df[start:end]
    # faccio una lista dei valori nella colonna 'name'
    current_gen_name = list(current_gen['name'])
    # droppo le colonne che non mi interessano
    current_gen = current_gen.drop(['name', 'total_points', 'type_1'], axis=1)
    # metto in una lista i valori di 'attack' 'defense' 'sp_Attack' 'sp_defense' 'speed'
    current_gen = list(current_gen.values)
    # casto a lista i valori all'interno dell' array di NP
    current_gen_points = [[*l] for l in current_gen]
    # per chiudere il grafico devo duplicare il primo valore
    for l in current_gen_points:
        l.append(l[0])

    return (current_gen_name, current_gen_points)
```

Definiti i calcoli, il prossimo step è quello di disegnare i grafici con i dati raccolti:

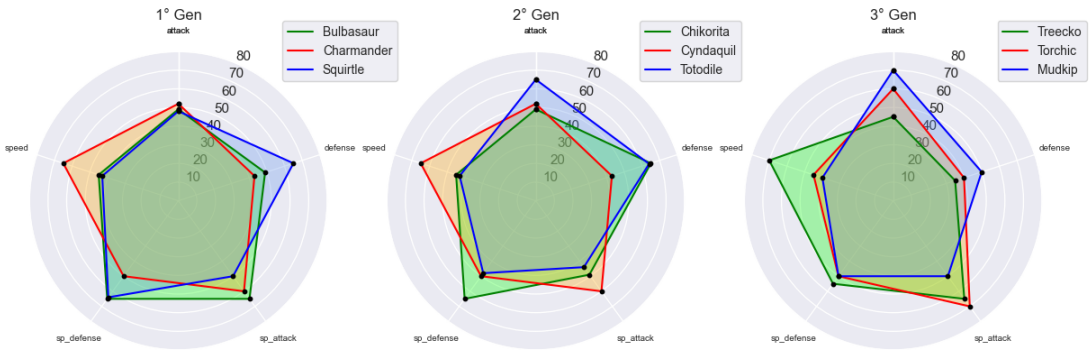
Passando alla realizzazione del grafico vi era, anche qui, diverso codice ripetuto per creare gli otto grafici totali (ben 150+ righe!). Prendendo atto di questo, ho provato a fare una funzione che mi disegnasse in automatico tutti i grafici in un subplot 3x3. ***Non sono riuscito in questo***, ma sono comunque arrivato alla soluzione che consiste nel disegnare una riga alla volta.

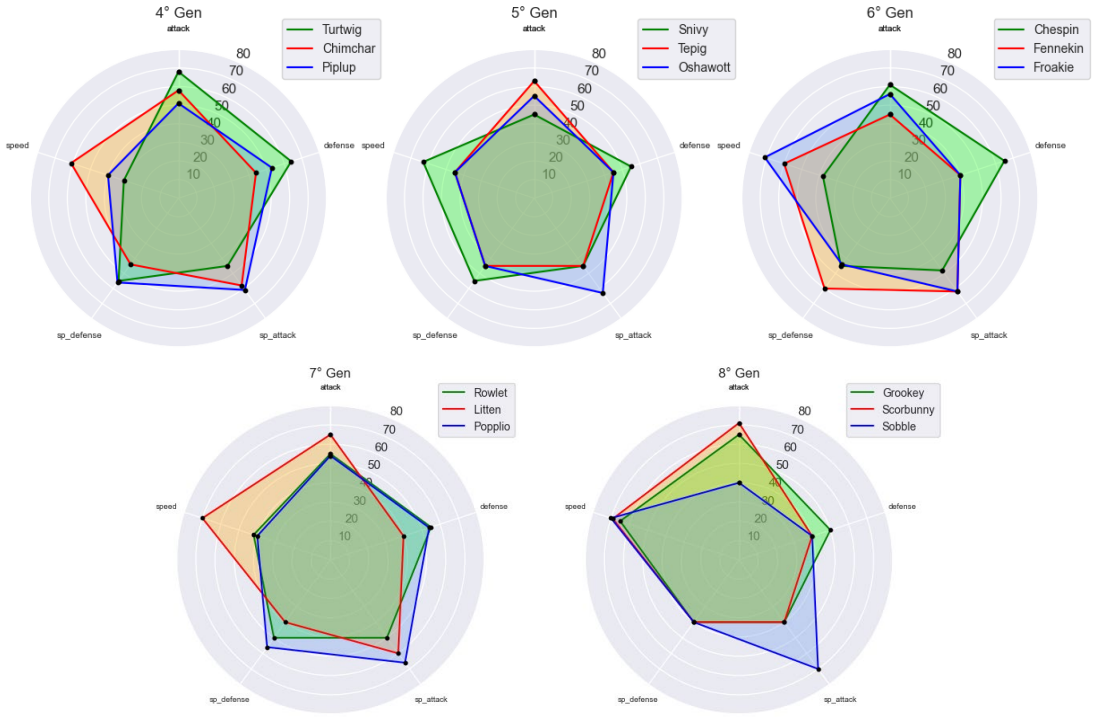
```python
# Faccio un dizionario per le prime 3 entry del primo gruppo di plot
one_to_three = {
    1: [*first_gen, first_gen_name],
    2: [*second_gen, second_gen_name],
    3: [*third_gen, third_gen_name],
}
# Faccio un dizionario per le prime 3 entry del secondo gruppo di plot
fourth_to_sixth = {
    1: [*fourth_gen, fourth_gen_name],
    2: [*fifth_gen, fifth_gen_name],
    3: [*sixth_gen, sixth_gen_name],
}
# Faccio un dizionario per le prime 2 entry del terzo gruppo di plot
seventh_to_eighth = {
    1: [*seventh_gen, seventh_gen_name],
    2: [*eighth_gen, eighth_gen_name],
}
```

```python
# Definisco una funzione che preso il dizionario, il titolo, e di default una tupla per colori e colori di riempimento che
# riempie un subplot con massimo 3 grafici (per renderlo leggibile)
def draw(my_dic, title=(), color=color, filler=fill_color):
    for i in range(len(my_dic)):
        for j in range(3):
            ax[i].plot(label_loc, my_dic[i+1][j], label=my_dic[i+1][3][j], color=color[j])
            ax[i].fill(label_loc, my_dic[i+1][j], filler[j], alpha=0.3)
            ax[i].scatter(label_loc, my_dic[i+1][j],s=10, color='black', zorder=10)
            ax[i].set_thetagrids(np.degrees(label_loc),labels=categories_to_show, fontsize=7)
            ax[i].set_theta_offset(np.pi / 2)
            ax[i].set_theta_direction(-1)
            ax[i].set_ylim(0, 80)
            ax[i].legend(loc=(0.85, 0.9), fontsize=10)
            fig.subplots_adjust(right=2)
        ax[i].set_title(f'{title[i]}° Gen')
    return(ax)
```

```python
# Creo il primo subplot con 1 riga e 3 colonne (dove andranno le prime 3 generazioni (1,2,3))
fig, (ax) = plt.subplots(1, 3, figsize=(6, 6),subplot_kw=dict(projection='polar'), facecolor='white')
first_to_thrid = draw(one_to_three, ('1', '2', '3'))
plt.savefig('images/first_radar.png', dpi='figure', format='png', bbox_inches='tight')
# Creo il secondo subplot con 1 riga e 3 colonne(dove andranno le seconde 3 generazioni (4,5,6))
fig, (ax) = plt.subplots(1, 3, figsize=(6, 6),subplot_kw=dict(projection='polar'),facecolor='white')
fourth_to_sixth = draw(fourth_to_sixth, ('4', '5', '6'))
plt.savefig('images/second_radar.png', dpi='figure', format='png', bbox_inches='tight')
# Creo il terzo subplot con 1 riga e 2 colonne(dove andranno le ultime 2 generazioni (7,8))
fig, (ax) = plt.subplots(1, 2, figsize=(5, 5),subplot_kw=dict(projection='polar'),facecolor='white')
seventh_and_eighth = draw(seventh_to_eighth, ('7', '8'))
plt.savefig('images/third_radar.png', dpi='figure', format='png', bbox_inches='tight')
plt.show()
```

Il risultato, sono tre grafici di tre generazioni per riga. Con un software di editing di immagini le ho accorpate insieme ricalcando quello che sarebbe dovuto essere il risultato ottimale secondo la mia idea iniziale (si trova nella cartella con tutte le immagini).





6. Esportazione in SQL con SQLAlchemy

In questo punto, ho usato la libreria di SQLAlchemy per trasferire un DataFrame in un DataBase per poterci operare sopra.

Ho creato la connessione col database e poi l'ho trasferito in sql.

```
from sqlalchemy import create_engine
engine = create_engine('sqlite://', echo=False)

df_to_db = filtered_pokemon_data[['pokedex_number', 'name', 'date_published', 'generation', 'status', 'total_points', 'catch_rate']]

df_to_db.to_sql('Pokedex_Up_To_8th_Gen', con=engine)
```

Successivamente ho messo in un DataFrame il risultato della query e l'ho convertito in CSV.

```
df_db_query = pd.DataFrame(engine.execute("SELECT * FROM Pokedex_Up_To_8th_Gen WHERE catch_rate > 50").fetchall())
df_db_CSV = df_db_query.to_csv('CSV/sql_to_csv.csv', index=False)
```

CONCLUSIONI:

Dai grafici a barre riguardanti la quantità di Pokémon, si può evincere come l'incremento totale di generazione in generazione sia molto lineare e costante, però le aggiunte fatte vedono le generazioni "pari" avere un incremento minore rispetto a quelle "dispari", con generazione 3, 5 e 7 che aggiungono più Pokémon rispetto a 2, 4, 6 e 8.
Passando all'analisi dei tipi, possiamo notare come i primari siano distribuiti in maniera abbastanza omogenea eccezion fatta per il tipo "Volante" che, secondo questo dataset, ha un numero di Pokémon inferiore. Invece abbiamo una situazione diametralmente opposta per quanto riguarda il tipo secondario, dove c'è un alto quantitativo di Pokémon "Volante" che sbilancia il grafico, distaccando assai le altre tipologie secondarie.
Per quanto riguarda il confronto generazionale tra i Pokémon più forti con quelli più deboli, possiamo notare come, nelle generazioni prese in causa, le statistiche dei Pokémon più deboli siano pressappoco similari fra loro e in alcuni casi coincidenti. Lo stesso comportamento si denota anche fra i più forti, eccezion fatta per Eternatus Eternamax che ha delle statistiche nettamente più alte degli altri, nonostante non sia un Pokémon catturabile, ma solo un Pokémon "evento", che appare una volta sola nel gioco".
Per quanto riguarda la comparazione fra starters, possiamo notare come in ogni generazione, ogni Pokémon prediliga una statistica lasciandone scoperta un'altra, portando il giocatore a un completo controllo su come affrontare strategicamente il suo "cammino". Prendiamo ad esempio la Seconda Generazione: se un "allenatore" volesse avere un Pokémon con statistiche "fisiche" più accentuate, prenderebbe sicuramente Totodile, invece, se si volessero prediligere le statistiche "speciali", si sceglierebbe Cyndaquil o Chicorita in base alla volontà di voler aver un Pokémon più veloce e offensivo o uno un po' più difensivo per attacchi sia "fisici" che "speciali".

SVILUPPI FUTURI:

Sicuramente in questa relazione non sono stati toccati alcuni punti, come la percentuale di Pokémon maschi o femmine, oppure tutte le colonne riguardanti le debolezze di ogni Pokémon su ogni tipo, o anche la percentuale di cattura di ogni Pokémon (se catturabile). Per questo, si potrebbe ampliare lo studio focalizzandosi sugli aspetti sopracitati, magari mettendo a confronto, per ogni generazione, il Pokémon più difficile da catturare, piuttosto che trovare il tipo Pokémon "migliore" per ogni generazione. Infine, si potrebbe anche espandere il grafico degli "starters a confronto" con i loro secondi e terzi stadi evolutivi, come originariamente pensato.

BIBLIOGRAFIA e FONTI:

- https://www.kaggle.com/
- https://www.kaggle.com/mariotormo
- https://www.kaggle.com/datasets/mariotormo/complete-pokemon-dataset-updated-090420
- https://www.kaggle.com/datasets/maca11/all-pokemon-dataset
- https://www.kaggle.com/maca11
- https://www.kaggle.com/code/bryanb/pokemon-eda-with-plotly
- https://www.kaggle.com/bryanb
- https://bulbapedia.bulbagarden.net/wiki/Main_Page
- https://github.com/SavGRY/savini_its_data_analysis