Nettoyage des données

(note : l'export html/pdf rend assez mal, le fichier notebook est plus joli dans la globalité. il est conseillé de lire ces notes depuis jupyter (ou autre éditeur notebook))

On commence par importer pandas puis importer les données pour nettoyer le dataset.

On se chargera uniquement de ***mettre en forme les données*** et non de les afficher dans ce fichier.

Les données nettoyées seront exportées dans un fichier csv -> "./Datasets/MP-24-25_Cleaned.csv"

In [2]:
```
import pandas as pd
dfRawPokemonData = pd.read_csv('./Datasets/MP-24-25.csv')
```

In [3]:
```
dfRawPokemonData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1045 entries, 0 to 1044
Data columns (total 55 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0.1       1045 non-null   int64  
 1   Unnamed: 0         1045 non-null   int64  
 2   pokedex_number     1045 non-null   int64  
 3   name               1045 non-null   object 
 4   german_name        1045 non-null   object 
 5   japanese_name      1045 non-null   object 
 6   is_generation_1    1045 non-null   bool   
 7   is_generation_2    1045 non-null   bool   
 8   is_generation_3    1045 non-null   bool   
 9   is_generation_4    1045 non-null   bool   
 10  is_generation_5    1045 non-null   bool   
 11  is_generation_6    1045 non-null   bool   
 12  is_generation_7    1045 non-null   bool   
 13  is_generation_8    1045 non-null   bool   
 14  status             1045 non-null   object 
 15  species            1045 non-null   object 
 16  types              1045 non-null   object 
 17  height             1045 non-null   object 
 18  weight             1045 non-null   object 
 19  abilities_number   1045 non-null   int64  
 20  ability_1          1042 non-null   object 
 21  ability_2          516 non-null    object 
 22  ability_hidden     813 non-null    object 
 23  total_points       1045 non-null   int64  
 24  hp                 1045 non-null   int64  
 25  attack             1045 non-null   int64  
 26  defense            1045 non-null   int64  
 27  sp_attack          1045 non-null   int64  
 28  sp_defense         1045 non-null   int64  
 29  speed              1045 non-null   int64  
 30  catch_rate         1027 non-null   float64
 31  base_friendship    930 non-null    float64
 32  base_experience    925 non-null    float64
 33  growth_rate        1044 non-null   object 
 34  egg_types          1042 non-null   object 
 35  percentage_male    872 non-null    float64
 36  egg_cycles         1044 non-null   float64
 37  against_normal     1045 non-null   float64
 38  against_fire       1045 non-null   float64
 39  against_water      1045 non-null   float64
 40  against_electric   1045 non-null   float64
 41  against_grass      1045 non-null   float64
 42  against_ice        1045 non-null   float64
 43  against_fight      1045 non-null   float64
 44  against_poison     1045 non-null   float64
 45  against_ground     1045 non-null   float64
 46  against_flying     1045 non-null   float64
 47  against_psychic    1045 non-null   float64
 48  against_bug        1045 non-null   float64
 49  against_rock       1045 non-null   float64
 50  against_ghost      1045 non-null   float64
 51  against_dragon     1045 non-null   float64
 52  against_dark       1045 non-null   float64
 53  against_steel      1045 non-null   float64
 54  against_fairy      1045 non-null   float64
```

```
dtypes: bool(8), float64(23), int64(11), object(13)
memory usage: 392.0+ KB
```

On remarque que la plupart du set est cohérent (au niveau des nulls), mais il faut cependant retirer des colonnes qui ne nous serviront pas :

- Le nom Japonais des Pokémons
- Le nom Allemand des Pokémons
- les deux colonnes Unnamed placées au début qui servaient sûrement à la base d'index
- On pourrait aussi retirer la colonne "Base_Frienship", comme c'est une statistique qui n'est quasiment pas implémentée dans le jeu vidéo, et qui ne nous permettera donc pas d'émettre beaucoup de conclusions...

C'est parti !

In [5]:
```
dfPokemonList = dfRawPokemonData.drop(columns=['german_name','japanese_name','Un
```

In [6]:
```
dfPokemonList.columns
```

Out[6]:
```
Index(['pokedex_number', 'name', 'is_generation_1', 'is_generation_2',
       'is_generation_3', 'is_generation_4', 'is_generation_5',
       'is_generation_6', 'is_generation_7', 'is_generation_8', 'status',
       'species', 'types', 'height', 'weight', 'abilities_number', 'ability_1',
       'ability_2', 'ability_hidden', 'total_points', 'hp', 'attack',
       'defense', 'sp_attack', 'sp_defense', 'speed', 'catch_rate',
       'base_friendship', 'base_experience', 'growth_rate', 'egg_types',
       'percentage_male', 'egg_cycles', 'against_normal', 'against_fire',
       'against_water', 'against_electric', 'against_grass', 'against_ice',
       'against_fight', 'against_poison', 'against_ground', 'against_flying',
       'against_psychic', 'against_bug', 'against_rock', 'against_ghost',
       'against_dragon', 'against_dark', 'against_steel', 'against_fairy'],
      dtype='object')
```

Nos colonnes désormais supprimées, on chercher maintenant à identifier les valeurs qui pourraient être gênantes à la manipulation des autres tableaux

on assume que toute la colonne est déjà censée avoir le même format de données

In [8]:
```
dfPokemonList[dfPokemonList['height'].str[1] != "'"]['height']
```

Out[8]:
```
31       11'06''
126      28'10''
135      35'09''
167      21'04''
168      21'04''
191      13'01''
255      30'02''
256      34'05''
302      17'01''
303      12'06''
386      47'07''
417      20'04''
459      14'09''
460      32'02''
461      11'06''
462      16'05''
463      23'00''
464      35'05''
578      17'09''
579      13'09''
581      12'02''
582      14'09''
583      22'08''
590      10'06''
594      10'10''
750      10'06''
755      10'10''
756      11'10''
838      19'00''
839      16'05''
841      14'09''
845      21'04''
878      26'11''
914      12'10''
924      11'02''
925      13'01''
929      12'06''
930      30'02''
932      18'01''
934      12'06''
935      13'09''
936      24'07''
940      11'10''
941      18'01''
980      12'06''
1032     65'07''
1033    328'01''
Name: height, dtype: object
```

Même si l'argument est peu faible (malgré que suffisant), on remarque que les seuls objets de la liste qui n'ont pas d'apostrophe "'" sont les valeurs avec une taille en pieds en dizaine ou centaine, ce qui confirme (en plus des 0 null dans la première étape info) que la colonne ne contient pas de valeurs erronnées

En plus, les pouces sont exprimés avec une double apostrophe, on peut donc facilement le split de la façon suivante :

In [10]:
```
sSplittedHeight = dfPokemonList['height'].str.split("'")
```

Je le convertis en dataframe pour faire des calculs par la suite

In [12]:
```
dfSplittedHeight = pd.DataFrame(sSplittedHeight.values.tolist(), index=sSplitted
dfSplittedHeight
```

Out[12]:

	0	1	2	3
0	2	04		
1	3	03		
2	6	07		
3	7	10		
4	2	00		
...	...	...	...	...
1040	7	03		
1041	6	07		
1042	3	07		
1043	7	10		
1044	7	10		

1045 rows × 4 columns

On supprime les colonnes vides et on renomme les restantes...

In [14]:
```
dfSplittedHeight = dfSplittedHeight.drop(columns=[2,3])
```

In [15]:
```
dfSplittedHeight = dfSplittedHeight.rename(columns={0:'feet',1:'inch'})
dfSplittedHeight
```

Out[15]:

	feet	inch
0	2	04
1	3	03
2	6	07
3	7	10
4	2	00
...	...	...
1040	7	03
1041	6	07
1042	3	07
1043	7	10
1044	7	10

1045 rows × 2 columns

Convertir le type des colonnes pour faire les calculs

In [17]:
```
dfSplittedHeight['feet'] = dfSplittedHeight['feet'].astype('float')
dfSplittedHeight['inch'] = dfSplittedHeight['inch'].astype('float')

dfSplittedHeight.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1045 entries, 0 to 1044
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   feet    1045 non-null   float64
 1   inch    1045 non-null   float64
dtypes: float64(2)
memory usage: 16.5 KB
```

et on termine ce dataset en ajoutant une colonne qui contient la taille en mètres

In [19]:
```
dfSplittedHeight.insert(2, 'meters' , value= dfSplittedHeight['feet'] * 0.3048 +
```

In [20]:
```
dfSplittedHeight
```

Out[20]:

	feet	inch	meters
0	2.0	4.0	0.7112
1	3.0	3.0	0.9906
2	6.0	7.0	2.0066
3	7.0	10.0	2.3876
4	2.0	0.0	0.6096
...	...	...	...
1040	7.0	3.0	2.2098
1041	6.0	7.0	2.0066
1042	3.0	7.0	1.0922
1043	7.0	10.0	2.3876
1044	7.0	10.0	2.3876

1045 rows × 3 columns

Victoire ! on a notre taille en mètres, on remplace donc nos données dans le dataset initial.

In [22]:
```
dfPokemonList['height'] = dfSplittedHeight['meters']
```

In [23]:
```
dfPokemonList[['name','height']]
```

Out[23]:

	name	height
0	Bulbasaur	0.7112
1	Ivysaur	0.9906
2	Venusaur	2.0066
3	Mega Venusaur	2.3876
4	Charmander	0.6096
...	...	...
1040	Glastrier	2.2098
1041	Spectrier	2.0066
1042	Calyrex	1.0922
1043	Calyrex Ice Rider	2.3876
1044	Calyrex Shadow Rider	2.3876

1045 rows × 2 columns

on s'attaque au poids

In [25]:
```
sSplittedWeight = dfPokemonList['weight'].str.split("lbs")
```

même stratégie que precedemment, on le convertit en df

In [27]:
```
dfSplittedWeight = pd.DataFrame(sSplittedWeight.values.tolist(), index=sSplitted
dfSplittedWeight
```

Out[27]:

	0	1
0	15.21	
1	28.66	
2	220.46	
3	342.82	
4	18.74	
...	...	...
1040	1763.7	
1041	98.11	
1042	16.98	
1043	1783.76	
1044	118.17	

1045 rows × 2 columns

In [28]:
```
dfSplittedWeight = dfSplittedWeight.drop(columns=[1])
```

In [29]:
```
dfSplittedWeight = dfSplittedWeight.rename(columns={0:'lbs'})
dfSplittedWeight
```

Out[29]:

	lbs
0	15.21
1	28.66
2	220.46
3	342.82
4	18.74
...	...
1040	1763.7
1041	98.11
1042	16.98
1043	1783.76
1044	118.17

1045 rows × 1 columns

Cette fois-ci, pour vérifier que la colonne ne contienne pas de valeurs eronnées, et comme on ne divise qu'en une seule partie j'utilise :

In [31]:
```
dfSplittedWeight[pd.to_numeric(dfSplittedWeight['lbs'],errors="coerce").isna() =
```

Out[31]:

	lbs
1033	nan

Une seule valeur null dans la liste, on regarde quel est le pokemon concerné

In [33]:
```
dfPokemonList.iloc[1033]
```

Out[33]:
```
pokedex_number                    890
name                Eternatus Eternamax
is_generation_1                   False
is_generation_2                   False
is_generation_3                   False
is_generation_4                   False
is_generation_5                   False
is_generation_6                   False
is_generation_7                   False
is_generation_8                    True
status                       Legendary
species                Gigantic Pokémon
types                    Poison, Dragon
height                          99.9998
weight                          nan lbs
abilities_number                      0
ability_1                           NaN
ability_2                           NaN
ability_hidden                      NaN
total_points                       1125
hp                                  255
attack                              115
defense                             250
sp_attack                           125
sp_defense                          250
speed                               130
catch_rate                          NaN
base_friendship                     NaN
base_experience                     NaN
growth_rate                        Slow
egg_types                 Undiscovered
percentage_male                     NaN
egg_cycles                        120.0
against_normal                      1.0
against_fire                        0.5
against_water                       0.5
against_electric                    0.5
against_grass                      0.25
against_ice                         2.0
against_fight                       0.5
against_poison                      0.5
against_ground                      2.0
against_flying                      1.0
against_psychic                     2.0
against_bug                         0.5
against_rock                        1.0
against_ghost                       1.0
against_dragon                      2.0
against_dark                        1.0
against_steel                       1.0
against_fairy                       1.0
Name: 1033, dtype: object
```

D'après poképedia, ce pokemon ne possède en effet pas de poids. pour des raisons statistiques et de simplicité, on considèrera donc que le poids du pokemon est la moyenne des autres...

In [35]:
```
EternatusEtermaxWeight = dfSplittedWeight['lbs'].drop(index=[1033]).astype('floa
```

In [36]:
```
print(EternatusEtermaxWeight)
```

```
157.00548850574714
```

Sa valeur attribuée est donc 157.00548850574714.

In [38]:
```
dfSplittedWeight.loc[1033, 'lbs'] = EternatusEtermaxWeight
```

Convertir le type des colonnes pour faire les calculs

In [40]:
```
dfSplittedWeight['lbs'] = dfSplittedWeight['lbs'].astype('float')

dfSplittedWeight.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1045 entries, 0 to 1044
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   lbs     1045 non-null   float64
dtypes: float64(1)
memory usage: 8.3 KB
```

on créée donc notre conversion dans le df du poids

In [42]:
```
dfSplittedWeight.insert(1, 'kg' , value= 0.454 * dfSplittedWeight['lbs'], allow_
```

In [43]:
```
dfSplittedWeight
```

Out[43]:

	lbs	kg
0	15.21	6.90534
1	28.66	13.01164
2	220.46	100.08884
3	342.82	155.64028
4	18.74	8.50796
...	...	...
1040	1763.70	800.71980
1041	98.11	44.54194
1042	16.98	7.70892
1043	1783.76	809.82704
1044	118.17	53.64918

1045 rows × 2 columns

Le résultat n'est pas arrondi mais on peut quand même le mettre dans le dataset initial.

In [45]:
```
dfPokemonList['weight'] = dfSplittedWeight['kg']
```

In [46]:
```
dfPokemonList[['name','height','weight']]
```

Out[46]:

	name	height	weight
0	Bulbasaur	0.7112	6.90534
1	Ivysaur	0.9906	13.01164
2	Venusaur	2.0066	100.08884
3	Mega Venusaur	2.3876	155.64028
4	Charmander	0.6096	8.50796
...	...	...	...
1040	Glastrier	2.2098	800.71980
1041	Spectrier	2.0066	44.54194
1042	Calyrex	1.0922	7.70892
1043	Calyrex Ice Rider	2.3876	809.82704
1044	Calyrex Shadow Rider	2.3876	53.64918

1045 rows × 3 columns

Maintenant, on cherche à remplacer nos colonnes générations par une colonne unique qui représente la génération par un entier plutôt que 8 booléens

(la solution présentée suivante a été trouvée sur stackoverflow : Pandas multiple boolean columns to new single column with mapping dict)

In [48]:
```
dfGenerationColumns = dfPokemonList.filter(like='generation_')
```

In [49]:
```
dfGenerationColumns
```

Out[49]:

	is_generation_1	is_generation_2	is_generation_3	is_generation_4	is_generation_5
0	True	False	False	False	False
1	True	False	False	False	False
2	True	False	False	False	False
3	True	False	False	False	False
4	True	False	False	False	False
...	...	...	...	...	...
1040	False	False	False	False	False
1041	False	False	False	False	False
1042	False	False	False	False	False
1043	False	False	False	False	False
1044	False	False	False	False	False

1045 rows × 8 columns

In [50]:
```
mapping_dict = {False: "none", "is_generation_1": "1", "is_generation_2": "2", "
```

On utilise ce sous ensemble et ce dictionnaire pour mapper nos colonnes en une seule nouvelle colonne :

In [52]:
```
dfGenerationColumns.insert(allow_duplicates=True,column='generation',loc=8,value
```

In [53]:
```
dfGenerationColumns['generation'] = (dfGenerationColumns.idxmax(axis=1).map(mapp
```

```
C:\Users\Aubin\AppData\Local\Temp\ipykernel_11108\3306450642.py:1: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfGenerationColumns['generation'] = (dfGenerationColumns.idxmax(axis=1).map(map
ping_dict).where(dfGenerationColumns.any(axis=1), mapping_dict[False]))
```

On insère notre colonne calculée à notre dataset initial...

In [55]:
```
dfPokemonList.insert(column='generation', loc=2, value=dfGenerationColumns['gene
```

In [56]:
```
dfPokemonList[dfPokemonList['generation'] == 0]
```

Out[56]:

	pokedex_number	name	generation	is_generation_1	is_generation_2	is_generation_3

0 rows × 52 columns

On voit que notre dataset ne contient pas de lignes avec generation = 0, donc nos générations on été mappées sans erreurs, on prend aussi un exemple pour faire une

double vérification :

In [58]:
```
dfPokemonList.iloc[430] # nombre choisi aléatoirement
```

Out[58]:
```
pokedex_number                  359
name                          Absol
generation                        3
is_generation_1               False
is_generation_2               False
is_generation_3                True
is_generation_4               False
is_generation_5               False
is_generation_6               False
is_generation_7               False
is_generation_8               False
status                       Normal
species             Disaster Pokémon
types                         Dark
height                      1.1938
weight                     47.04348
abilities_number                 3
ability_1                 Pressure
ability_2                Super Luck
ability_hidden            Justified
total_points                    465
hp                               65
attack                          130
defense                          60
sp_attack                        75
sp_defense                       60
speed                            75
catch_rate                     30.0
base_friendship                35.0
base_experience               163.0
growth_rate              Medium Slow
egg_types                      Field
percentage_male                50.0
egg_cycles                     25.0
against_normal                  1.0
against_fire                    1.0
against_water                   1.0
against_electric                1.0
against_grass                   1.0
against_ice                     1.0
against_fight                   2.0
against_poison                  1.0
against_ground                  1.0
against_flying                  1.0
against_psychic                 0.0
against_bug                     2.0
against_rock                    1.0
against_ghost                   0.5
against_dragon                  1.0
against_dark                    0.5
against_steel                   1.0
against_fairy                   2.0
Name: 430, dtype: object
```

Nos données correspondent, OK.

On va retirer les anciennes colonnes

In [60]:
```
dfPokemonList = dfPokemonList.drop(columns=['is_generation_1','is_generation_2',
```

In [61]:
```
dfPokemonList
```

Out[61]:

	pokedex_number	name	generation	status	species	types	height
0	1	Bulbasaur	1	Normal	Seed Pokémon	Grass, Poison	0.7112
1	2	Ivysaur	1	Normal	Seed Pokémon	Grass, Poison	0.9906
2	3	Venusaur	1	Normal	Seed Pokémon	Grass, Poison	2.0066
3	3	Mega Venusaur	1	Normal	Seed Pokémon	Grass, Poison	2.3876
4	4	Charmander	1	Normal	Lizard Pokémon	Fire	0.6096
...	...	...	...	...	...	...	...
1040	896	Glastrier	8	Sub Legendary	Wild Horse Pokémon	Ice	2.2098
1041	897	Spectrier	8	Sub Legendary	Swift Horse Pokémon	Ghost	2.0066
1042	898	Calyrex	8	Legendary	King Pokémon	Psychic, Grass	1.0922
1043	898	Calyrex Ice Rider	8	Legendary	High King Pokémon	Psychic, Ice	2.3876
1044	898	Calyrex Shadow Rider	8	Legendary	High King Pokémon	Psychic, Ghost	2.3876

1045 rows × 44 columns

Après avoir nettoyé la variable génération, on va maintenant s'occuper des types ; on prend la même approche que pour la taille et pour la masse :

on commence par découper les types par leur virgule (présente ou non)

In [63]:
```
sSplittedPokemonTypes = dfPokemonList['types'].str.split(",")
```

In [64]:
```
sSplittedPokemonTypes[sSplittedPokemonTypes.isna()]
```

Out[64]:
```
Series([], Name: types, dtype: object)
```

In [65]:
```
sSplittedPokemonTypes
```

Out[65]:
```
0        [Grass,  Poison]
1        [Grass,  Poison]
2        [Grass,  Poison]
3        [Grass,  Poison]
4                [Fire]
              ...
1040              [Ice]
1041            [Ghost]
1042    [Psychic,  Grass]
1043      [Psychic,  Ice]
1044    [Psychic,  Ghost]
Name: types, Length: 1045, dtype: object
```

Pas de NaN à l'horizon, on peut donc passer à la suite

On transforme la series en dataframe et on supprime les espaces qui sont venus se glisser dans les valeurs

In [67]:
```
dfPokemonTypes = pd.DataFrame(sSplittedPokemonTypes.values.tolist(), index=sSpli
dfPokemonTypes[1] = dfPokemonTypes[1].str.strip()
dfPokemonTypes
```

Out[67]:

	0	1
0	Grass	Poison
1	Grass	Poison
2	Grass	Poison
3	Grass	Poison
4	Fire	None
...	...	...
1040	Ice	None
1041	Ghost	None
1042	Psychic	Grass
1043	Psychic	Ice
1044	Psychic	Ghost

1045 rows × 2 columns

on compte le nombre de valeurs en l'ajoutant dans une colonne

In [69]:
```
dfPokemonTypes.insert(loc=0, column='type_number', value=dfPokemonTypes.count(ax
```

In [70]:
```
dfPokemonTypes
```

Out[70]:

	type_number	0	1
0	2	Grass	Poison
1	2	Grass	Poison
2	2	Grass	Poison
3	2	Grass	Poison
4	1	Fire	None
...	...	...	...
1040	1	Ice	None
1041	1	Ghost	None
1042	2	Psychic	Grass
1043	2	Psychic	Ice
1044	2	Psychic	Ghost

1045 rows × 3 columns

maintenant qu'on a toutes les valeurs qui nous intéressent, on va modifier les noms de colonne et les agréger à notre dataframe principal

In [72]:
```
dfPokemonTypes = dfPokemonTypes.rename(columns={0:'type_1',1:'type_2'})
```

In [73]:
```
dfPokemonTypes
```

Out[73]:

	type_number	type_1	type_2
0	2	Grass	Poison
1	2	Grass	Poison
2	2	Grass	Poison
3	2	Grass	Poison
4	1	Fire	None
...	...	...	...
1040	1	Ice	None
1041	1	Ghost	None
1042	2	Psychic	Grass
1043	2	Psychic	Ice
1044	2	Psychic	Ghost

1045 rows × 3 columns

In [74]:
```
dfPokemonList.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1045 entries, 0 to 1044
Data columns (total 44 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   pokedex_number    1045 non-null   int64
 1   name              1045 non-null   object
 2   generation        1045 non-null   int32
 3   status            1045 non-null   object
 4   species           1045 non-null   object
 5   types             1045 non-null   object
 6   height            1045 non-null   float64
 7   weight            1045 non-null   float64
 8   abilities_number  1045 non-null   int64
 9   ability_1         1042 non-null   object
 10  ability_2         516 non-null    object
 11  ability_hidden    813 non-null    object
 12  total_points      1045 non-null   int64
 13  hp                1045 non-null   int64
 14  attack            1045 non-null   int64
 15  defense           1045 non-null   int64
 16  sp_attack         1045 non-null   int64
 17  sp_defense        1045 non-null   int64
 18  speed             1045 non-null   int64
 19  catch_rate        1027 non-null   float64
 20  base_friendship   930 non-null    float64
 21  base_experience   925 non-null    float64
 22  growth_rate       1044 non-null   object
 23  egg_types         1042 non-null   object
 24  percentage_male   872 non-null    float64
 25  egg_cycles        1044 non-null   float64
 26  against_normal    1045 non-null   float64
 27  against_fire      1045 non-null   float64
 28  against_water     1045 non-null   float64
 29  against_electric  1045 non-null   float64
 30  against_grass     1045 non-null   float64
 31  against_ice       1045 non-null   float64
 32  against_fight     1045 non-null   float64
 33  against_poison    1045 non-null   float64
 34  against_ground    1045 non-null   float64
 35  against_flying    1045 non-null   float64
 36  against_psychic   1045 non-null   float64
 37  against_bug       1045 non-null   float64
 38  against_rock      1045 non-null   float64
 39  against_ghost     1045 non-null   float64
 40  against_dragon    1045 non-null   float64
 41  against_dark      1045 non-null   float64
 42  against_steel     1045 non-null   float64
 43  against_fairy     1045 non-null   float64
dtypes: float64(25), int32(1), int64(9), object(9)
memory usage: 355.3+ KB
```

avec toutes nos modifications, on voit que le type est le numéro de colonne '5', on va donc insérer notre dataframe à la colonne 5, puis supprimer l'ancienne.

In [76]:
```
dfPokemonList.insert(loc=5, column='type_number', value= dfPokemonTypes['type_nu
dfPokemonList.insert(loc=6, column='type_1', value= dfPokemonTypes['type_1'], al
dfPokemonList.insert(loc=7, column='type_2', value= dfPokemonTypes['type_2'], al
```

In [77]:
```
dfPokemonList = dfPokemonList.drop(columns='types')
```

In [78]:
```
dfPokemonList
```

Out[78]:

	pokedex_number	name	generation	status	species	type_number	ty
0	1	Bulbasaur	1	Normal	Seed Pokémon	2	(
1	2	Ivysaur	1	Normal	Seed Pokémon	2	(
2	3	Venusaur	1	Normal	Seed Pokémon	2	(
3	3	Mega Venusaur	1	Normal	Seed Pokémon	2	(
4	4	Charmander	1	Normal	Lizard Pokémon	1	
...	...	...	...	...	...	...	
1040	896	Glastrier	8	Sub Legendary	Wild Horse Pokémon	1	
1041	897	Spectrier	8	Sub Legendary	Swift Horse Pokémon	1	(
1042	898	Calyrex	8	Legendary	King Pokémon	2	Ps
1043	898	Calyrex Ice Rider	8	Legendary	High King Pokémon	2	Ps
1044	898	Calyrex Shadow Rider	8	Legendary	High King Pokémon	2	Ps

1045 rows × 46 columns

On fait la même chose pour le types d'oeufs :

In [80]:
```
sSplittedEggTypes = dfPokemonList['egg_types'].str.split(",")
```

In [81]:
```
sSplittedEggTypes[sSplittedEggTypes.isna()]
```

Out[81]:
```
33     NaN
172    NaN
658    NaN
Name: egg_types, dtype: object
```

On remarque que certains pokemons n'ont pas de types d'oeufs. après vérification (non montrée ici), les trois pokemons concernés sont : Pikachu Partenaire, Evoli Partenaire, et Darumarond de galar (mode Transe). Pour ne pas se séparer de ces lignes, et comme ce sont des pokemons qui ont d'autres formes, on peut facilement remplacer ces valeurs par

la mode des autres.

On remplacera donc ici, par la valeur des pokémons dans leur version standard.

On cherche donc nos types d'oeufs

In [83]:
```
dfPokemonList[dfPokemonList['name'] == 'Pikachu'][['name','egg_types']]
```

Out[83]:

	name	egg_types
32	Pikachu	Fairy, Field

In [84]:
```
dfPokemonList[dfPokemonList['name'] == 'Eevee'][['name','egg_types']]
```

Out[84]:

	name	egg_types
171	Eevee	Field

In [85]:
```
dfPokemonList[dfPokemonList['name'] == 'Galarian Darmanitan Standard Mode'][['na
```

Out[85]:

	name	egg_types
657	Galarian Darmanitan Standard Mode	Field

d'après notre liste déjà éxistante, on sait donc que les types d'oeufs sont :

- Pikachu : Field, Fairy
- Eevee : Field
- Galarian Darmanitan Standard Mode : Field

on remplace nos valeurs manquantes.

In [87]:
```
sSplittedEggTypes.iloc[33] = ['Fairy','Field'] # Pikachu
sSplittedEggTypes.iloc[172] = ['Field'] # Eevee
sSplittedEggTypes.iloc[658] = ['Field'] # Darmanitan
```

In [88]:
```
sSplittedEggTypes[sSplittedEggTypes.isna()]
```

Out[88]:
```
Series([], Name: egg_types, dtype: object)
```

Plus de NaN ! , on va pouvoir finir nos opérations

In [90]:
```
dfEggTypes = pd.DataFrame(sSplittedEggTypes.values.tolist(), index=sSplittedEggT
dfEggTypes
```

Out[90]:

	0	1
0	Grass	Monster
1	Grass	Monster
2	Grass	Monster
3	Grass	Monster
4	Dragon	Monster
...	...	...
1040	Undiscovered	None
1041	Undiscovered	None
1042	Undiscovered	None
1043	Undiscovered	None
1044	Undiscovered	None

1045 rows × 2 columns

In [91]:
```
dfEggTypes.insert(loc=0, column='egg_type_number', value=dfEggTypes.count(axis=1
```

In [92]:
```
dfEggTypes
```

Out[92]:

	egg_type_number	0	1
0	2	Grass	Monster
1	2	Grass	Monster
2	2	Grass	Monster
3	2	Grass	Monster
4	2	Dragon	Monster
...	...	...	...
1040	1	Undiscovered	None
1041	1	Undiscovered	None
1042	1	Undiscovered	None
1043	1	Undiscovered	None
1044	1	Undiscovered	None

1045 rows × 3 columns

On renomme nos colonnes :

In [94]:
```
dfEggTypes = dfEggTypes.rename(columns={0:'egg_type_1',1:'egg_type_2'})
```

Il existe des Pokemon qui n'ont pas de type d'oeufs découverts, on remplacera donc leur

nombre de type par '0' dans ces cas là.

In [96]:
```
dfEggTypes.loc[dfEggTypes['egg_type_1'] == 'Undiscovered', 'egg_type_number'] =
```

In [97]:
```
dfEggTypes[dfEggTypes['egg_type_1'] == 'Undiscovered']
```

Out[97]:

	egg_type_number	egg_type_1	egg_type_2
41	0	Undiscovered	None
42	0	Undiscovered	None
184	0	Undiscovered	None
185	0	Undiscovered	None
186	0	Undiscovered	None
...	...	...	...
1040	0	Undiscovered	None
1041	0	Undiscovered	None
1042	0	Undiscovered	None
1043	0	Undiscovered	None
1044	0	Undiscovered	None

152 rows × 3 columns

OK.

Comme pour les types de pokemon, on agrège à notre data frame principal. On cherche le numéro de colonne pour l'insérer dans le dataframe.

In [99]:
```
dfPokemonList.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1045 entries, 0 to 1044
Data columns (total 46 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   pokedex_number    1045 non-null   int64
 1   name              1045 non-null   object
 2   generation        1045 non-null   int32
 3   status            1045 non-null   object
 4   species           1045 non-null   object
 5   type_number       1045 non-null   int64
 6   type_1            1045 non-null   object
 7   type_2            553 non-null    object
 8   height            1045 non-null   float64
 9   weight            1045 non-null   float64
 10  abilities_number  1045 non-null   int64
 11  ability_1         1042 non-null   object
 12  ability_2         516 non-null    object
 13  ability_hidden    813 non-null    object
 14  total_points      1045 non-null   int64
 15  hp                1045 non-null   int64
 16  attack            1045 non-null   int64
 17  defense           1045 non-null   int64
 18  sp_attack         1045 non-null   int64
 19  sp_defense        1045 non-null   int64
 20  speed             1045 non-null   int64
 21  catch_rate        1027 non-null   float64
 22  base_friendship   930 non-null    float64
 23  base_experience   925 non-null    float64
 24  growth_rate       1044 non-null   object
 25  egg_types         1042 non-null   object
 26  percentage_male   872 non-null    float64
 27  egg_cycles        1044 non-null   float64
 28  against_normal    1045 non-null   float64
 29  against_fire      1045 non-null   float64
 30  against_water     1045 non-null   float64
 31  against_electric  1045 non-null   float64
 32  against_grass     1045 non-null   float64
 33  against_ice       1045 non-null   float64
 34  against_fight     1045 non-null   float64
 35  against_poison    1045 non-null   float64
 36  against_ground    1045 non-null   float64
 37  against_flying    1045 non-null   float64
 38  against_psychic   1045 non-null   float64
 39  against_bug       1045 non-null   float64
 40  against_rock      1045 non-null   float64
 41  against_ghost     1045 non-null   float64
 42  against_dragon    1045 non-null   float64
 43  against_dark      1045 non-null   float64
 44  against_steel     1045 non-null   float64
 45  against_fairy     1045 non-null   float64
dtypes: float64(25), int32(1), int64(10), object(10)
memory usage: 371.6+ KB
```

loc à insérer : 25

In [101…

```
dfPokemonList.insert(loc=25, column='egg_type_number', value= dfEggTypes['egg_ty
dfPokemonList.insert(loc=26, column='egg_type_1', value= dfEggTypes['egg_type_1'
dfPokemonList.insert(loc=27, column='egg_type_2', value= dfEggTypes['egg_type_2'
```

In [102…

```
dfPokemonList.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1045 entries, 0 to 1044
Data columns (total 49 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   pokedex_number    1045 non-null   int64  
 1   name              1045 non-null   object 
 2   generation        1045 non-null   int32  
 3   status            1045 non-null   object 
 4   species           1045 non-null   object 
 5   type_number       1045 non-null   int64  
 6   type_1            1045 non-null   object 
 7   type_2            553 non-null    object 
 8   height            1045 non-null   float64
 9   weight            1045 non-null   float64
 10  abilities_number  1045 non-null   int64  
 11  ability_1         1042 non-null   object 
 12  ability_2         516 non-null    object 
 13  ability_hidden    813 non-null    object 
 14  total_points      1045 non-null   int64  
 15  hp                1045 non-null   int64  
 16  attack            1045 non-null   int64  
 17  defense           1045 non-null   int64  
 18  sp_attack         1045 non-null   int64  
 19  sp_defense        1045 non-null   int64  
 20  speed             1045 non-null   int64  
 21  catch_rate        1027 non-null   float64
 22  base_friendship   930 non-null    float64
 23  base_experience   925 non-null    float64
 24  growth_rate       1044 non-null   object 
 25  egg_type_number   1045 non-null   int64  
 26  egg_type_1        1045 non-null   object 
 27  egg_type_2        286 non-null    object 
 28  egg_types         1042 non-null   object 
 29  percentage_male   872 non-null    float64
 30  egg_cycles        1044 non-null   float64
 31  against_normal    1045 non-null   float64
 32  against_fire      1045 non-null   float64
 33  against_water     1045 non-null   float64
 34  against_electric  1045 non-null   float64
 35  against_grass     1045 non-null   float64
 36  against_ice       1045 non-null   float64
 37  against_fight     1045 non-null   float64
 38  against_poison    1045 non-null   float64
 39  against_ground    1045 non-null   float64
 40  against_flying    1045 non-null   float64
 41  against_psychic   1045 non-null   float64
 42  against_bug       1045 non-null   float64
 43  against_rock      1045 non-null   float64
 44  against_ghost     1045 non-null   float64
 45  against_dragon    1045 non-null   float64
 46  against_dark      1045 non-null   float64
 47  against_steel     1045 non-null   float64
 48  against_fairy     1045 non-null   float64
dtypes: float64(25), int32(1), int64(11), object(12)
memory usage: 396.1+ KB
```

On supprime notre colonne egg_types du dataset initial

In [104…
```
dfPokemonList = dfPokemonList.drop(columns='egg_types')
```

In [105…
```
dfPokemonList
```

Out[105…

	pokedex_number	name	generation	status	species	type_number	ty
0	1	Bulbasaur	1	Normal	Seed Pokémon	2	
1	2	Ivysaur	1	Normal	Seed Pokémon	2	
2	3	Venusaur	1	Normal	Seed Pokémon	2	
3	3	Mega Venusaur	1	Normal	Seed Pokémon	2	
4	4	Charmander	1	Normal	Lizard Pokémon	1	
...	...	...	...	...	...	...	
1040	896	Glastrier	8	Sub Legendary	Wild Horse Pokémon	1	
1041	897	Spectrier	8	Sub Legendary	Swift Horse Pokémon	1	
1042	898	Calyrex	8	Legendary	King Pokémon	2	Ps
1043	898	Calyrex Ice Rider	8	Legendary	High King Pokémon	2	Ps
1044	898	Calyrex Shadow Rider	8	Legendary	High King Pokémon	2	Ps

1045 rows × 48 columns

Lors de notre analyse, on s'est aprçus que la colonne 'against_ice' contenait des valeurs mal rédigées, on va donc les régler ici.

En effet, les faiblesses de pokémon ne peuvent être situées qu'entre 0 et 4, or, on a découvert que des valeurs erronnées à 125 sont dans le dataset.

Ces valeurs ne sont pas impossibles, simplement mal formattées, au lieu de 125, la résistance est en réalité 0.125 (1/8, grâce à un talent).

> **On va d'abord identifier les pokémons touchés puis les modifier**

In [107…
```
print(dfPokemonList[dfPokemonList['against_ice'] > 4][['pokedex_number','name']])
dfPokemonList.loc[dfPokemonList['against_ice'] > 4, 'against_ice'] = 0.125
```

```
     pokedex_number     name
115              87  Dewgong
436             363   Spheal
437             364   Sealeo
438             365  Walrein
```

Terminé de ce côté !

On supprime le mot Pokémon de chaque espèce, pour simplement avoir le nom réel de l'espèce :

In [109…
```
dfPokemonList['species'] = dfPokemonList['species'].str.replace(r' Pokémon', '',
```

Maintenant que l'on a entièrement nettoyé notre dataset, on l'enregistre en csv à l'adresse du début !

In [111…
```
dfPokemonList.to_csv('./Datasets/MP-24-25_Cleaned.csv')
```

Le notebook pour le nettoyage se termine ici, pour la visualisation et l'analyse de données, on utilisera l'autre notebook, pour bien séparer les usages.