data_analysis

December 29, 2019

1 Pokemon

1.1 Context

Pokémon (, Pokemon) és un dels videojocs que Satoshi Tajiri va crear per a diverses plataformes, especialment la Game Boy, i que -arran de la seva popularitat- va aconseguir expandir-se a altres mitjans d'entreteniment, com ara sèries de televisió, jocs de cartes i roba, convertint-se, així, en una marca comercial reconeguda al mercat mundial. Fins al dia 1 de desembre de 2006 havien arribat a 175 milions d'exemplars venuts (inclosa la versió Pikachu de la consola Nintendo 64), arribant a ocupar el segon lloc de les nissagues de videojocs més venudes de Nintendo.

La saga Pokémon fou creada el 27 de febrer de 1996 al Japó. És desenvolupada per la companyia programadora de software japonesa Game Freak, amb els personatges creats per Satoshi Tajiri per a la companyia de joguines Creatures Inc., i alhora distribuïda i/o publicada per Nintendo. La missió dels protagonistes d'aquests jocs és capturar i entrenar els Pokémons, que actualment arriben a 806 tipus diferents. La possibilitat d'intercanviar-los amb altres usuaris va fer que la popularitat dels jocs de Pokémon augmentés i va provocar un èxit en les vendes de jocs de Pokémon, de televisió, de pel · lícules i marxandatges.

1.2 [1] Introducció

En aquesta pràctica es volen analitzar les dades dels *Pokemons* per tal d'extreure informació característica que ens permeti amplicar el coneixement i entendre millor la relació que hi ha entre ells.

Per això s'utilitzaràn diversos datasets (obtinguts de la plataforma Kaggle) que es complementen i que tenen les dades necessàries per realitzar l'anàlisi que es vol dur a terme.

Els *datasets* utilitzats són: * Informació dels pokemons * ***Pokemon_info.csv:*** Fitxer que conté les dades dels *Pokemons* amb els camps: * ***abilities***: Llista d'algunes de habilitats que pot adquirir. (Categòrica) * ***against_?***: Debilitat respecte a un tipus concret (against_fire, against_electric, etc). (Numèrica) * ***attack***: Punts d'atac. (Numèrica) * ***base_egg_steps***: Nombre de passos requerits per a que l'ou del *Pokemon* eclosioni. (Numèrica) * ***base_happiness***: Felicitat base. (Numèrica) * ***capture_rate***: Probabilitat de captura. (Numèrica) * ***classification***: Classificació del *Pokemon* segons la descripció de la *Pokedex* del joc Sol/Luna. (Categòrica) * ***defense***: Punts defensa. (Numèrica) * ***experience_growth***: Creixement d'experiència. (Numèrica) * ***height_m***: Alçada en metres. (Numèrica) * ***hp***: Punts de vida. (Numèrica) * ***japanese_name***: Nom original Japonès. (Categòrica) * ***name***: Nom del *Pokemon*. (Categòrica) * ***percentage_male***: Percentatge de mascles. (Numèrica) * ***pokedex_number***: Número de l'entrada en la *Pokedex*. (Numèrica) * ***sp_attack***: Atac especial. (Numèrica) * ***sp_defense***: Defensa especial. (Numèrica) * ***speed***: Velocitat. (Numèrica) * ***type1***: Tipus primari. (Categòrica) * ***type2***: Tipus secundari. (Categòrica) * ***weight_kg***: Pes en quilograms. (Numèrica) * ***generation***: Primera generació en que va apareixer el *Pokemon*. (Categòrica) * ***is_legendary***: Si és o no llegendari. (Categòrica)

- Informació de combats
  - ***combats.csv:*** Fitxer que conté informació sobre combats hipotètics
    - ***First_pokemon***: Identificador *Pokedex* del primer *Pokemon* del combat.
    - ***Second_pokemon***: Identificador *Pokedex* del segon *Pokemon* del combat.
    - ***Winner***: Identificador *Pokedex* del *Pokemon* guanyador.

1.3 [2] Integració i selecció

1.3.1 Imports

En aquesta pràctica s'utilitzaran les llibreries:

- *pandas*: Per treballar amb *DataFrames* (internament usa *numpy*).
- *matplotlib* i *seaborn*: Per fer els diagrames.
- *missingno*: Per fer gràfics de valors mancants.
- *scipy*: Per fer testos estadístics.

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
#conda install -c conda-forge missingno
```

```
import missingno as msno
import scipy as sp

path_folder = './datasets'
```

1.3.2 Carregar les dades

Pokemon_info dataset

[258]:
```
pokemon_info_df = pd.read_csv(path_folder+'/data/pokemon.csv')

#Dimensions del DF (files, columnes)
print(pokemon_info_df.shape)
```

```
(801, 42)
```

Hi ha **42 variables** i **801 registres**.

Quins són els diferents tipus de variable?

[259]:
```
print(pokemon_info_df.dtypes.unique())
```

```
[dtype('int64') dtype('O') dtype('float64')]
```

Hi ha variables de tipus:

- ***O***: Categòrica.
- ***float64***: Real.
- ***int64***: Enter.

De quin tipus és cada variable?

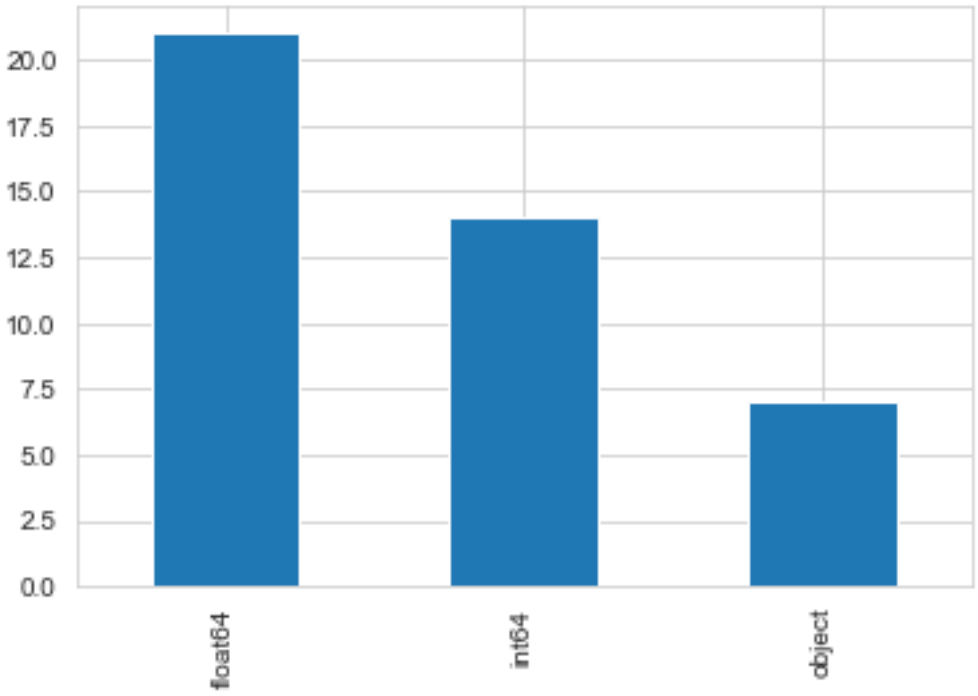
[260]:
```
#Variables
print(pokemon_info_df.dtypes)
```

```
Unnamed: 0            int64
abilities            object
against_bug         float64
against_dark        float64
against_dragon      float64
against_electric    float64
against_fairy       float64
against_fight       float64
against_fire        float64
against_flying      float64
against_ghost       float64
against_grass       float64
against_ground      float64
against_ice         float64
against_normal      float64
```

```
against_poison       float64
against_psychic      float64
against_rock         float64
against_steel        float64
against_water        float64
attack                 int64
base_egg_steps         int64
base_happiness         int64
base_total             int64
capture_rate          object
classfication         object
defense                int64
experience_growth      int64
height_m             float64
hp                     int64
japanese_name         object
name                  object
percentage_male      float64
pokedex_number         int64
sp_attack              int64
sp_defense             int64
speed                  int64
type1                 object
type2                 object
weight_kg            float64
generation             int64
is_legendary           int64
dtype: object
```

[261]:
```
pd.value_counts(pokemon_info_df.dtypes).plot.bar()
```

[261]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa61f07110>
```



Nota: Com es pot veure, hi ha moltes variables de tipus ***float64*** i ***int64***, es probable que donat el domini d'aquestes variables, es pogués canviar el tipus i així reduir la quantitat de memòria utilitzada.

1.3.3 *pokemon_battles dataset*

[268]:
```
pokemon_battles_df = pd.read_csv(path_folder+'/data/combats.csv')

print(pokemon_battles_df.shape)
```

```
(38743, 3)
```

Hi ha **38,743 registres** i **3 variables**

De quin tipus són?

[269]:
```
print(pokemon_battles_df.dtypes.unique())
```

```
[dtype('int64')]
```

[270]:
```
print(pokemon_battles_df.dtypes)
```

```
First_pokemon     int64
Second_pokemon    int64
```

```
Winner            int64
dtype: object
```

Totes les variables són enteres.

1.3.4 Selecció de les variables

1.4 [3] Neteja de les dades

Un cop es coneixen les variables de les quals es disposa i el seu tipus, és important explorar quines d'aquestes variables tenen valors mancants i si això fa que deixin de ser útils.

[6]:
```
#Hi ha algún camp en tot el DF que tingui un valor mancant?
pokemon_info_df.isnull().values.any()
```

[6]:
```
True
```

Quins camps tenen valors mancants?

[7]:
```
pokemon_info_mv_list = pokemon_info_df.columns[pokemon_info_df.isnull().any()].
↪tolist()
print(pokemon_info_mv_list)
```

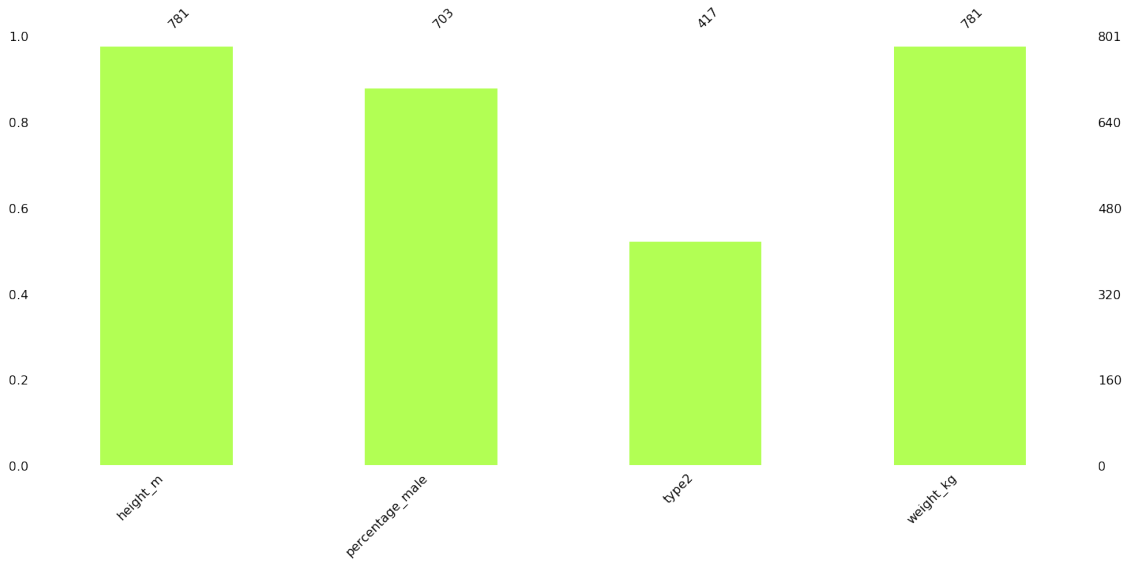
```
['height_m', 'percentage_male', 'type2', 'weight_kg']
```

Les variables: **height_m**, **percentage_male**, **type2**, **weight_kg** tenen valors mancants, però quants registres estan afectats?

[8]:
```
def missing_values(df, fields):
    n_rows = df.shape[0]
    for field in fields:
        n_missing_values = df[field].isnull().sum()
        print("%s: %d (%.3f)" % (field, n_missing_values, n_missing_values/
↪n_rows))
```

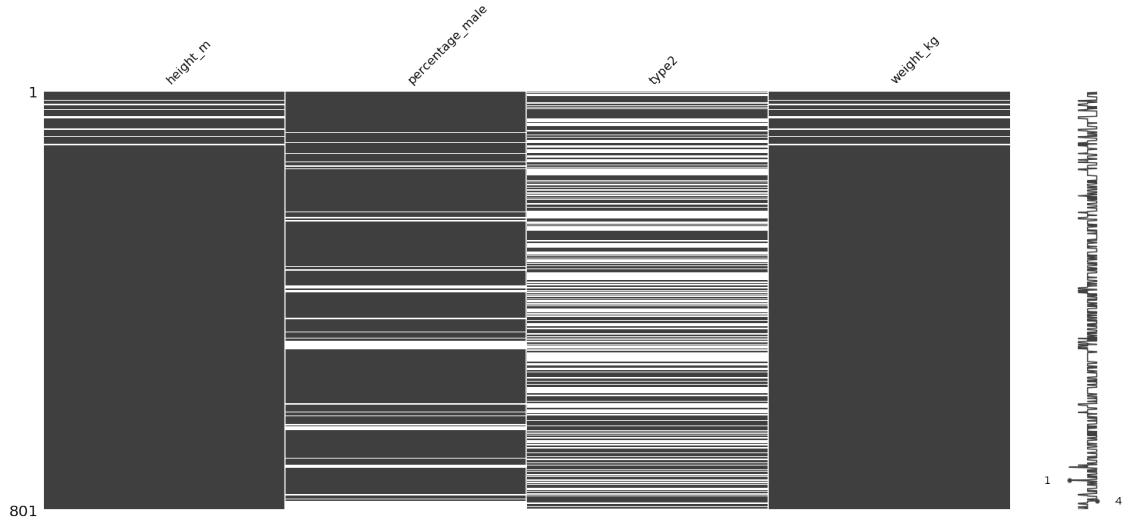
[9]:
```
msno.bar(pokemon_info_df[pokemon_info_mv_list], color="#b2ff54", labels=True)
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa7a457250>
```



[10]:
```
msno.matrix(pokemon_info_df[pokemon_info_mv_list])
```

[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffa7a3033d0>



La variable **height_m** té 20 registres sense valor (2.5%), **percentage_male** en té 98 (12.2%), **type2** 384 (48%) i **weight_kg** 20 (2.5%)

1.4.1 Imputar els valors perduts

Per tal d'imputar correctament els valors perduts, cal primer observar els altres valors per cada una d'aquestes variables. Així que anem a veure quins valors diferents hi ha per cada variable.

type2

[11]:
```
pokemon_info_df[pokemon_info_df['type2'].notnull()]['type2'].unique()
```

[11]:
```
array(['poison', 'flying', 'dark', 'electric', 'ice', 'ground', 'fairy',
       'grass', 'fight', 'psychic', 'steel', 'fire', 'rock', 'water',
       'dragon', 'ghost', 'bug', 'normal'], dtype=object)
```

Com es pot veure, hi ha 18 tipus de Pokemon diferents en la variable **type2**. Com que es tracta d'una variable categòrica, es pot assignar una etiqueta arbitrària (*unknown*) per distingir els valors mancants.

[12]:
```
pokemon_info_df['type2'].fillna('unknown', inplace=True)
```

percentage_male

Com que aquesta variable és de típu *float64* i el seu domini va de 0 a 1, es pot utilizar el valor -1 per indicar un valor mancant.

[13]:
```
pokemon_info_df['percentage_male'].fillna(np.int(-1), inplace=True)
```

height_m

Aquesta variable és de tipus *float64* i com que no té sentit un Pokemon amb alçada: 0 metres, es pot utilitzar el valor 0 com a indicador de valor mancant.

[14]:
```
pokemon_info_df['height_m'].fillna(np.int(0), inplace=True)
```

weight_kg

Igual que amb la variable **height_m**

[15]:
```
pokemon_info_df['weight_kg'].fillna(np.int(0), inplace=True)
```

Ara es pot comprovar que no hi ha cap valor *na* en tot el *dataset*

[16]:
```
pokemon_info_df.columns[pokemon_info_df.isnull().any()].tolist() == []
```

[16]:
```
True
```

1.5 Anàlisi descriptiu bàsic

1.5.1 Generacions

Quantes generacions hi ha?

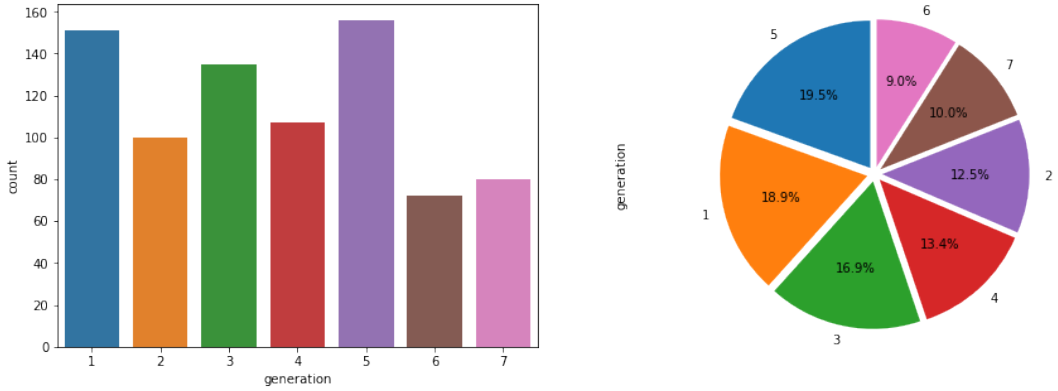
[17]:
```
print("Hi ha %d generacions de Pokemons" %(pokemon_info_df["generation"].
 ↪nunique()))
```

```
Hi ha 7 generacions de Pokemons
```

Com es distribueixen els Pokemons en base a la primera generació en que van apareixre?

[18]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

#Diagrama de barres
sns.countplot(x="generation", data=pokemon_info_df, ax=ax1)

#Diagrama de sectors
sector_diagram = pd.value_counts(pokemon_info_df.generation)
sector_diagram.plot.pie(startangle=90, autopct='%1.1f%%', shadow=False,
                        explode=(0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05),␣
 ↪ax=ax2)
plt.axis("equal")
```

[18]:
```
(-1.172795083539883, 1.157629806230198, -1.169802928367887, 1.1536302074244398)
```



Quines són les tres generacions on van apareixer més Pokemons?

[19]:
```
print("5na generació -> %d␣
 ↪Pokemons"%(len(pokemon_info_df[pokemon_info_df["generation"] == 5])))
print("1ra generació -> %d␣
 ↪Pokemons"%(len(pokemon_info_df[pokemon_info_df["generation"] == 1])))
print("3era generació -> %d␣
 ↪Pokemons"%(len(pokemon_info_df[pokemon_info_df["generation"] == 3])))
```

```
5na generació -> 156 Pokemons
1ra generació -> 151 Pokemons
3era generació -> 135 Pokemons
```

La generació amb més Pokemons és la **5na** amb **156 (19,5%)**, seguidament de la **1era** generació amb **151 (18,9%)** i finalment la **3era** generació amb **135 Pokemons (16,9%)**. Entre aquestes tres generacions hi ha el **55,3%** del total de Pokemons.

1.5.2 Pokemons llegendaris

Hi ha Pokemons que destaquen per sobre de la resta degut a les seves característiques, aquests Pokemons es coneixen com a llegendàris. Què podem dir al respecte d'aquests Pokemons?

Quants Pokemons llegendaris hi ha?

[20]:
```
print(len(pokemon_info_df[pokemon_info_df["is_legendary"] == True]))
```

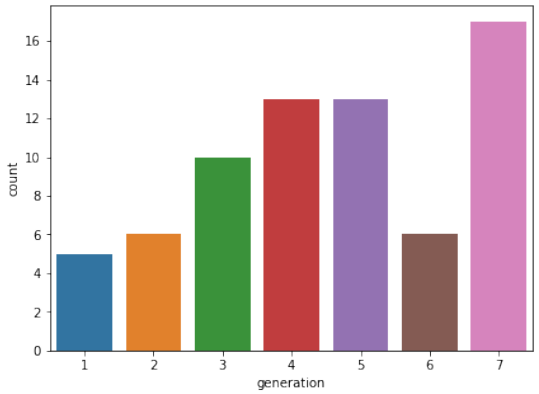
70

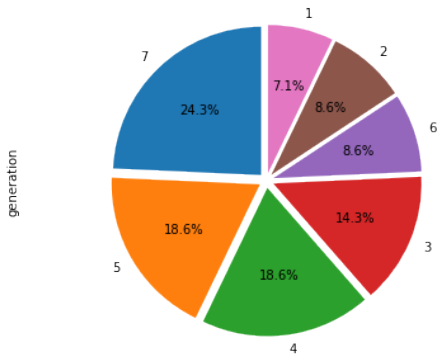
En total hi ha **70 Pokemons llegendaris.**

En quines edicions apareixen aquests Pokemons?

[21]:
```
pokemon_legendary_df = pokemon_info_df[pokemon_info_df["is_legendary"] == True]
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

#Diagrama de barres
sns.countplot(x="generation", data=pokemon_legendary_df, ax=ax1)

#Diagrama de sectors
sector_diagram = pd.value_counts(pokemon_legendary_df.generation)
sector_diagram.plot.pie(startangle=90, autopct='%1.1f%%', shadow=False,
                        explode=(0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05),␣
 ↪ax=ax2)
plt.axis("equal")
```

[21]:
```
(-1.155822766529171,
 1.1568298393693945,
 -1.1725632198765732,
 1.1545230565022295)
```





[22]:
```

```

```
print("7na generació -> %d␣
 ↪Pokemons"%(len(pokemon_legendary_df[pokemon_legendary_df["generation"] ==␣
 ↪7])))
print("4rta generació -> %d␣
 ↪Pokemons"%(len(pokemon_legendary_df[pokemon_legendary_df["generation"] ==␣
 ↪4])))
print("5na generació -> %d␣
 ↪Pokemons"%(len(pokemon_legendary_df[pokemon_legendary_df["generation"] ==␣
 ↪5])))
```

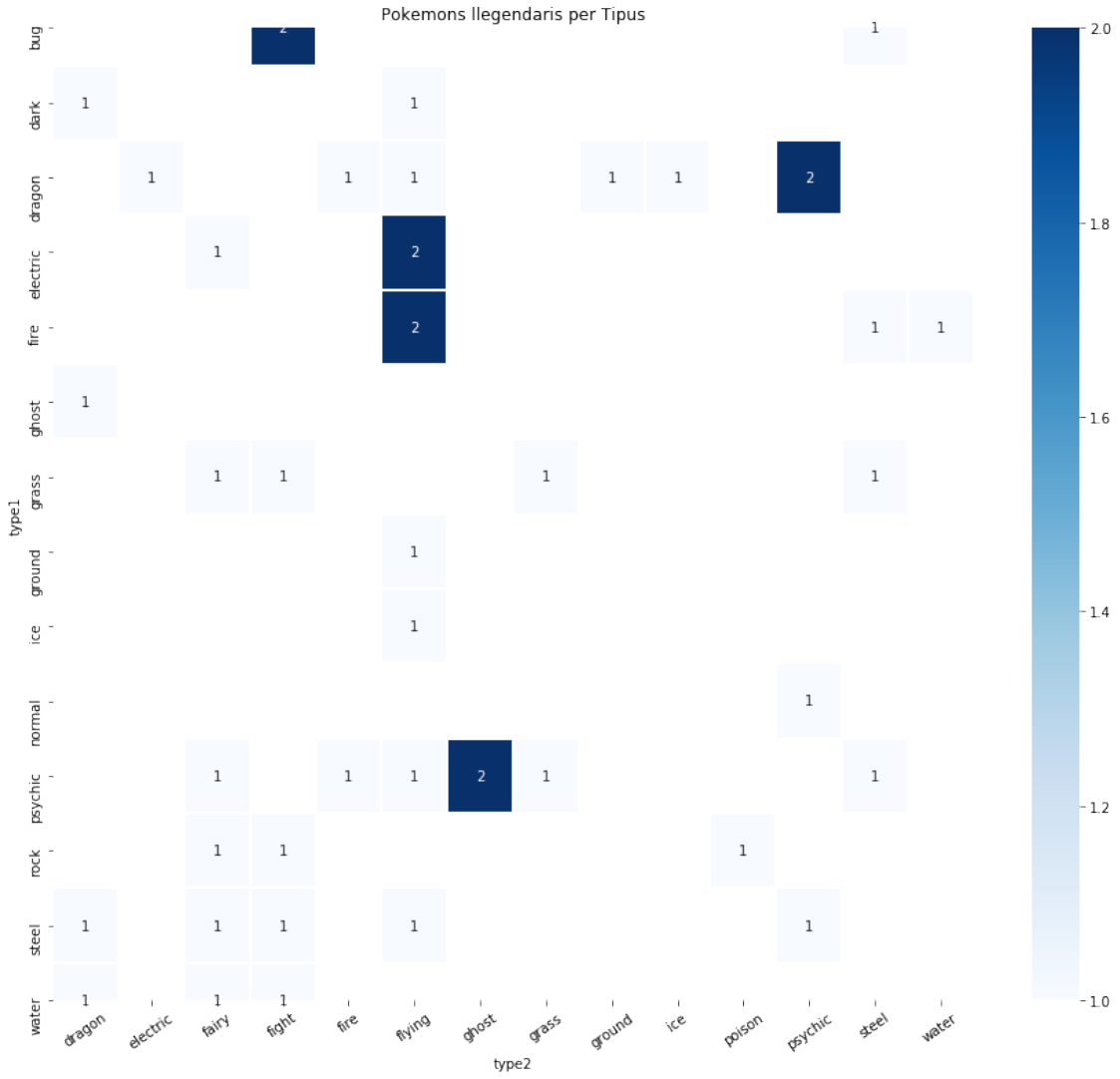
```
7na generació -> 17 Pokemons
4rta generació -> 13 Pokemons
5na generació -> 13 Pokemons
```

La **7na generació** té **17 Pokemons llegendaris (24,3%)**, la **4rta** en té **13 (18,6%)** i la **5na 13**. Entre **aquestes tres generacions** hi ha un **61,5% de Pokemons llegendaris**.

Quins és el tipus (*type1* i *type2*) amb més Pokemons llegendaris?

[23]:
```
def plot_by_type(dataFrame, title):
    plt.subplots(figsize=(15, 13))

    sns.heatmap(
        dataFrame[dataFrame["type2"] != "unknown"].groupby(["type1", "type2"]).
 ↪size().unstack(),
        cmap="Blues",
        linewidths=1,
        annot=True
    )

    plt.xticks(rotation=35)
    plt.title(title)
    plt.show()
```

[24]:
```
plot_by_type(pokemon_legendary_df, "Pokemons llegendaris per Tipus")
```



Els tipus **psíquic/fantasma**, **foc/volador**, **elèctric/volador**, **bitxo/lluita** i **drac/psíquic** són els tipus amb més Pokemons llegendaris, tots ells amb 2 exemplars.

Quin és el Pokemon llegendari amb més atac (attack), defensa (defense), vida (hp) i velocitat (velocity) mitjana?

[25]:
```
legendary_with_more_attack = max(pokemon_legendary_df['attack'])
legendary_with_less_attack = min(pokemon_legendary_df['attack'])

legendary_with_more_defense = max(pokemon_legendary_df['defense'])
legendary_with_less_defense = min(pokemon_legendary_df['defense'])

legendary_with_more_hp = max(pokemon_legendary_df['hp'])
legendary_with_less_hp = min(pokemon_legendary_df['hp'])
```

```
legendary_with_more_speed = max(pokemon_legendary_df['speed'])
legendary_with_less_speed = min(pokemon_legendary_df['speed'])


pokemon_legendary_df["strong"] = ((pokemon_legendary_df['attack'] -␣
↪legendary_with_less_attack)/
↪(legendary_with_more_attack-legendary_with_less_attack) +
                                  (pokemon_legendary_df['defense'] -␣
↪legendary_with_less_defense)/
↪(legendary_with_more_defense-legendary_with_less_defense) +
                                  (pokemon_legendary_df['hp'] -␣
↪legendary_with_less_hp)/(legendary_with_more_hp-legendary_with_less_hp) +
                                  (pokemon_legendary_df['speed'] -␣
↪legendary_with_less_speed)/
↪(legendary_with_more_speed-legendary_with_less_speed))
```

```
/home/oscar/.conda/envs/uoc/lib/python3.7/site-
packages/ipykernel_launcher.py:17: SettingWithCopyWarning:
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: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[26]:
```
pokemon_legendary_df["strong"]
```

[26]:
```
143    1.373481
144    1.422513
145    1.447958
149    2.097102
150    1.632615
         …
796    1.367552
797    2.184101
798    1.645820
799    1.521065
800    1.332612
Name: strong, Length: 70, dtype: float64
```

[27]:
```
pokemon_legendary_df[pokemon_legendary_df["strong"] ==␣
↪max(pokemon_legendary_df["strong"])]["name"]
```

[27]:
```
382    Groudon
Name: name, dtype: object
```

1.5.3 Type1 i type2

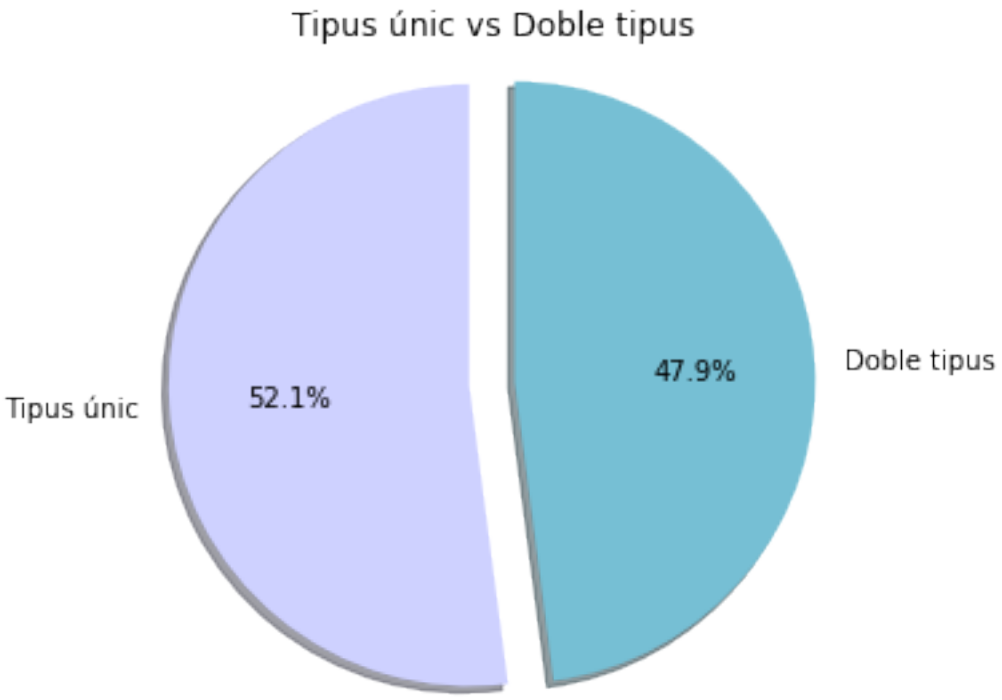
Cada Pokemon és d'un tipus concret **type1** o és una combinació de **type1** i **type2**, per aquest motiu, alguns dels Pokemons no tenen **type2** (com s'ha vist en l'apartat anterior).

[28]:
```
single_type_pokemons = []
dual_type_pokemons = []

for i in pokemon_info_df.index:
    if(pokemon_info_df.type2[i] != "unknown"):
        single_type_pokemons.append(pokemon_info_df.name[i])
    else:
        dual_type_pokemons.append(pokemon_info_df.name[i])

print("Nombre de Pokemons amb un únic tipus %d: " % len(single_type_pokemons))
print("Nombre de Pokemons amb dos tipus %d: " % len(dual_type_pokemons))
```

```
Nombre de Pokemons amb un únic tipus 417:
Nombre de Pokemons amb dos tipus 384:
```

Hi ha **417** Pokemons d'un únic tipus i **384** amb doble tipus, això es representa en el següent diagrama de sectors.

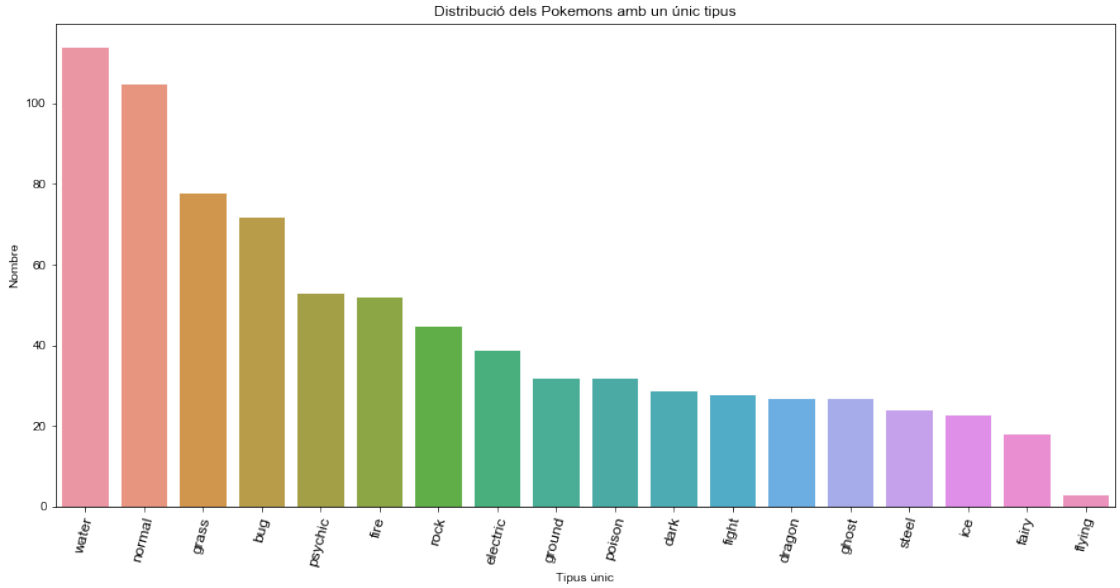
[29]:
```
data= [len(single_type_pokemons), len(dual_type_pokemons)]
colors= ["#ced1ff","#76bfd4"]

plt.pie(data, labels=["Tipus únic","Doble tipus"],
        startangle=90, explode=(0, 0.15),
        shadow=True, colors=colors, autopct='%1.1f%%')

plt.axis("equal")

plt.title("Tipus únic vs Doble tipus")

plt.tight_layout()

plt.show()
```



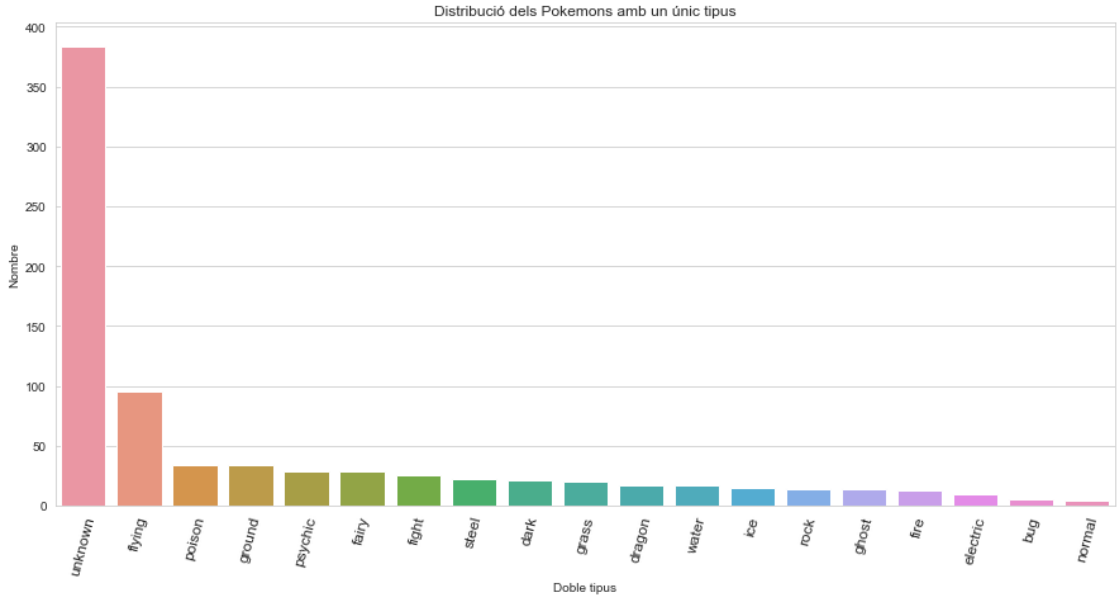
En els següents diagrames de barres es mostra la distribució de Pokemons per **type1** i per **type2**

[30]:
```
def plot_distribution(data, col, xlabel, ylabel, title):
    types = pd.value_counts(data[col])

    fig, ax = plt.subplots()
    fig.set_size_inches(15,7)
    sns.set_style("whitegrid")

    ax = sns.barplot(x=types.index, y=types, data=data)
    ax.set_xticklabels(ax.get_xticklabels(), rotation=75, fontsize=12)
    ax.set(xlabel=xlabel, ylabel=ylabel)
    ax.set_title(title)
```

[31]:
```
plot_distribution(pokemon_info_df, "type1", "Tipus únic", "Nombre",
                  "Distribució dels Pokemons amb un únic tipus")
```



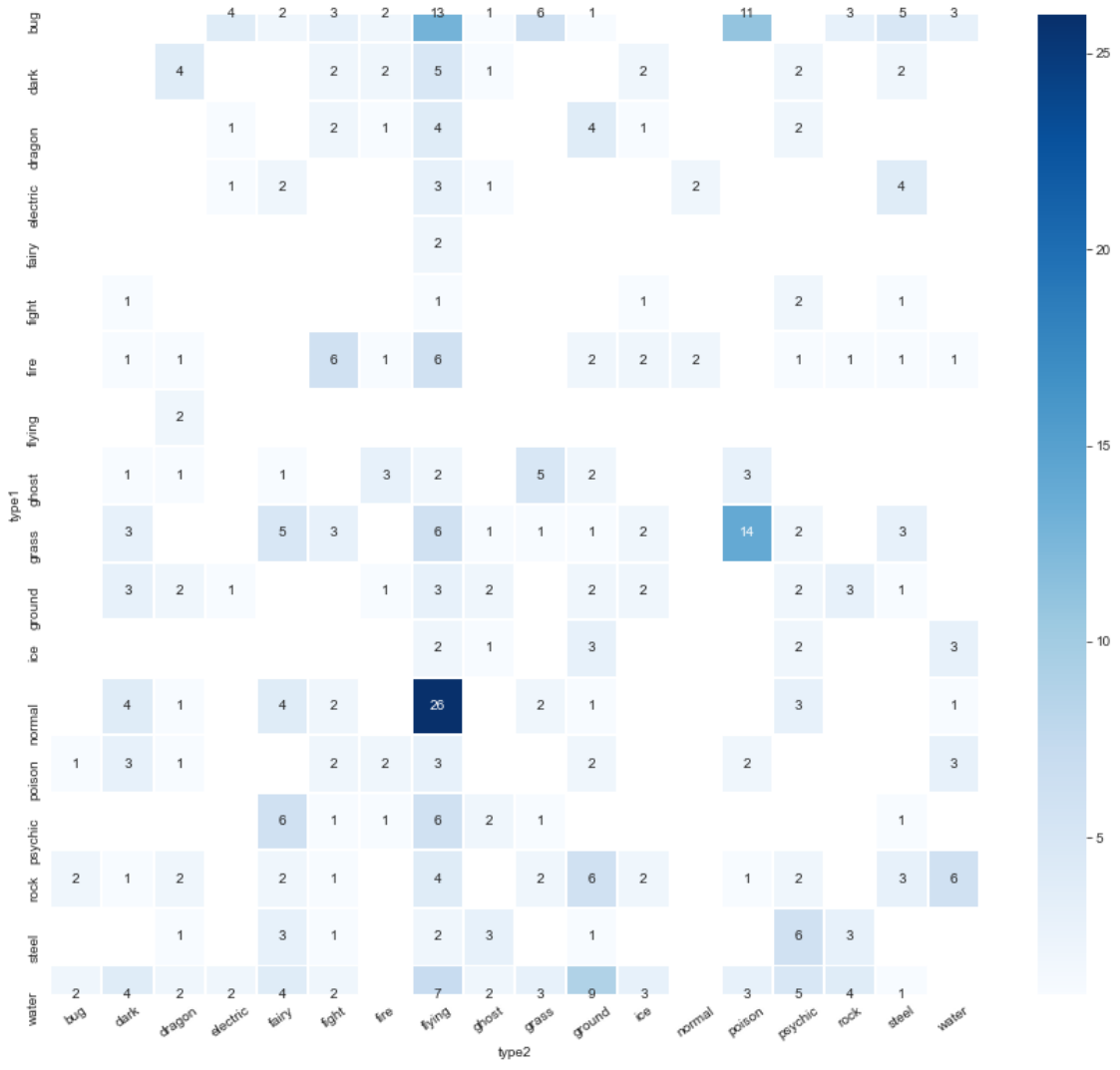
[32]:
```
plot_distribution(pokemon_info_df, "type2", "Doble tipus", "Nombre",
                  "Distribució dels Pokemons amb un únic tipus")
```



1.5.4 Combinació de tipus

Ara volem saber quina combinació de tipus **type1** i **type2** hi ha entre tots els Pokemons.

[33]:
```
plt.subplots(figsize=(15, 13))

sns.heatmap(
    pokemon_info_df[pokemon_info_df["type2"] != "unknown"].groupby(["type1",␣
 ↪"type2"]).size().unstack(),
    cmap="Blues",
    linewidths=1,
    annot=True
)

plt.xticks(rotation=35)
plt.show()
```



Com es pot veure, la combinació de tipus més comuna és **normal/volador** amb **26 Pokemons** seguida per la combinació **planta/verí** i **bitxo/volador** amb **14** i **13 Pokemons** respectivament.

Nota: En aquest mapa de calor s'han filtrat tots aquells Pokemons sense segon tipus.

1.5.5 Pes i alçada

La variable **height_m** conté l'alçada en metres, mentre que la variable **weight_kg** conté el pes en kilgorams. Així que, quin és el Pokemon amb major i menor pes? I el pokemon amb major i menor alçada?

[34]:
```
tallest_m = max(pokemon_info_df['height_m'])
shortest_m = tallest_m
for i in pokemon_info_df.index:
    if pokemon_info_df.height_m[i] > 0 and pokemon_info_df.height_m[i] <␣
 ↪shortest_m:
        shortest_m = pokemon_info_df.height_m[i]

tallest_pokemon = pokemon_info_df[pokemon_info_df['height_m'] == tallest_m]
shortest_pokemon = pokemon_info_df[pokemon_info_df['height_m'] == shortest_m]

print("Els Pokemons més alts són:")
for i in tallest_pokemon.index:
    print("\t%s amb %.2f metres" % (tallest_pokemon.name[i], tallest_pokemon.
 ↪height_m[i]))

print("\nEls Pokemons més petits són:")
for i in shortest_pokemon.index:
    print("\t%s amb %.2f metres" % (shortest_pokemon.name[i], shortest_pokemon.
 ↪height_m[i]))
```

```
Els Pokemons més alts són:
        Wailord amb 14.50 metres

Els Pokemons més petits són:
        Joltik amb 0.10 metres
        Flabébé amb 0.10 metres
        Cutiefly amb 0.10 metres
        Comfey amb 0.10 metres
        Cosmoem amb 0.10 metres
```

[35]:
```
max_weight = max(pokemon_info_df['weight_kg'])
light_kg = max_weight
for i in pokemon_info_df.index:
    if pokemon_info_df.weight_kg[i] > 0 and pokemon_info_df.weight_kg[i] <␣
 ↪light_kg:
        light_kg = pokemon_info_df.weight_kg[i]

heviest_pokemon = pokemon_info_df[pokemon_info_df['weight_kg'] == max_weight]
lightest_pokemon = pokemon_info_df[pokemon_info_df['weight_kg'] == light_kg]
```

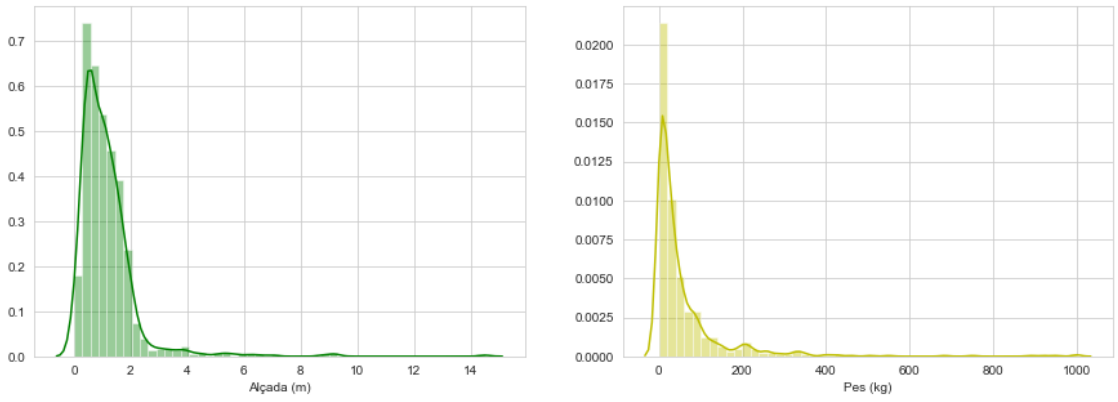
```
print("Els Pokemons amb més pes són:")
for i in heviest_pokemon.index:
    print("\t%s amb %.2f kilograms" % (heviest_pokemon.name[i], heviest_pokemon.
 ↪weight_kg[i]))

print("\nEls Pokemons amb menys pes són:")
for i in lightest_pokemon.index:
    print("\t%s amb %.2f kilograms" % (lightest_pokemon.name[i],␣
 ↪lightest_pokemon.weight_kg[i]))
```

```
Els Pokemons amb més pes són:
        Cosmoem amb 999.90 kilograms
        Celesteela amb 999.90 kilograms

Els Pokemons amb menys pes són:
        Gastly amb 0.10 kilograms
        Haunter amb 0.10 kilograms
        Flabébé amb 0.10 kilograms
        Cosmog amb 0.10 kilograms
        Kartana amb 0.10 kilograms
```

Ara es vol veure quina és la distribució de l'alçada i pes dels Pokemons, per això es pot utilitzar histogrames i diagrames de caixa.

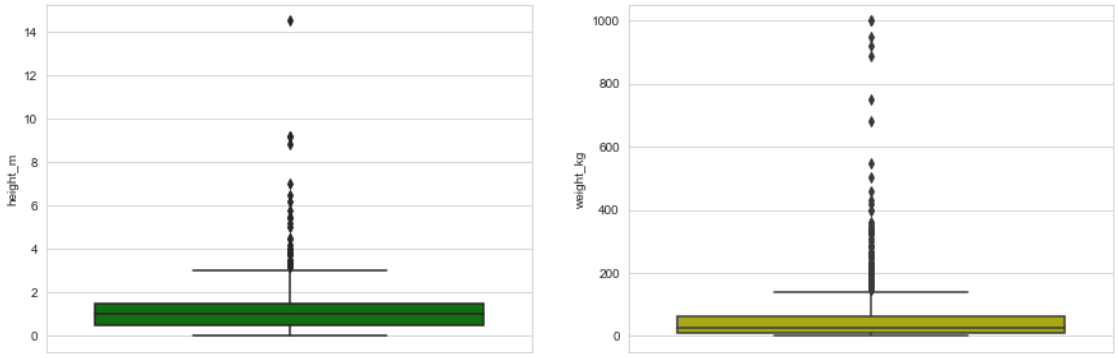
[36]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))

sns.distplot(pokemon_info_df['height_m'], color='g', axlabel="Alçada (m)",␣
 ↪ax=ax1)
sns.distplot(pokemon_info_df['weight_kg'], color='y', axlabel="Pes (kg)",␣
 ↪ax=ax2)
```

[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa79f191d0>
```



[37]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

sns.boxplot(x=pokemon_info_df["height_m"], color="g", orient="v", ax=ax1)
sns.boxplot(x=pokemon_info_df["weight_kg"], color="y", orient="v", ax=ax2)
```

[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa797cb2d0>
```

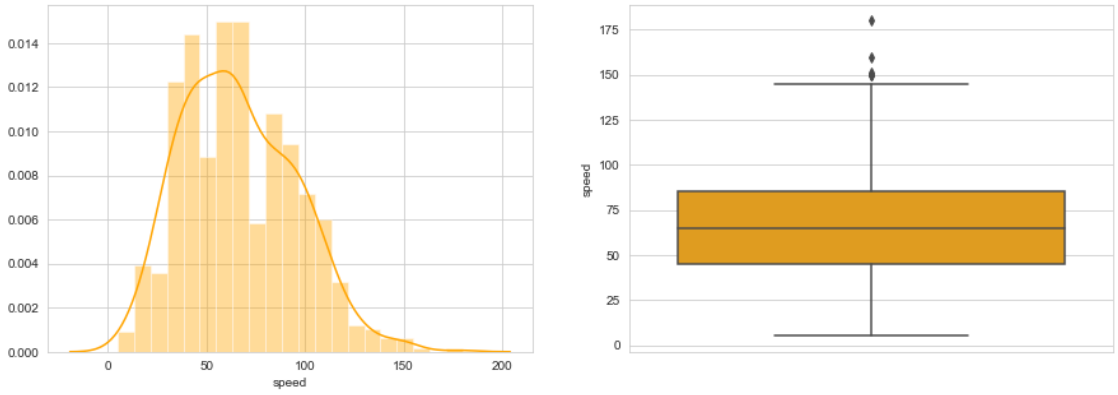


Tots aquells Pokemons amb una alçada inferior a Com es pot veure, hi ha Pokemons molt dispersos a la resta, es con

1.5.6 Velocitat

[38]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
sns.distplot(pokemon_info_df['speed'], color="orange", ax=ax1)
sns.boxplot(pokemon_info_df['speed'], color="orange", orient="v", ax=ax2)
```

[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa7970d490>
```



[39]:
```
fast_value = max(pokemon_info_df['speed'])
slow_value = min(pokemon_info_df[pokemon_info_df['speed'] != 0]['speed'])

fastest_pokemon = pokemon_info_df[pokemon_info_df['speed'] ==␣
↪max(pokemon_info_df['speed'])]
slowest_pokemon = pokemon_info_df[pokemon_info_df['speed'] == slow_value]

print("Els Pokemons més ràpids són:")
for i in fastest_pokemon.index:
    print("\t%s amb una velocitat de %.f punts" %(fastest_pokemon.name[i],␣
↪fastest_pokemon.speed[i]))

print("Els Pokemons més lents són:")
for i in slowest_pokemon.index:
    print("\t%s amb una velocitat de %.f punts" %(slowest_pokemon.name[i],␣
↪slowest_pokemon.speed[i]))
```

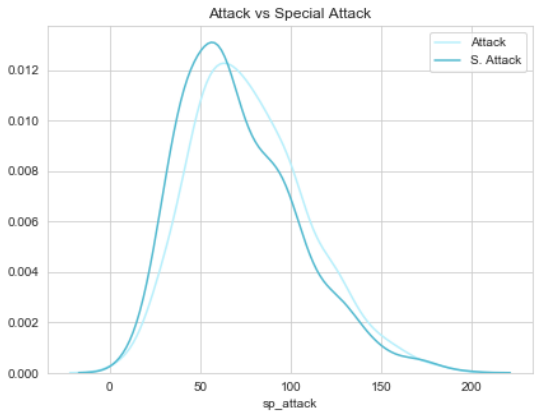
```
Els Pokemons més ràpids són:
        Deoxys amb una velocitat de 180 punts
Els Pokemons més lents són:
        Shuckle amb una velocitat de 5 punts
        Munchlax amb una velocitat de 5 punts
        Pyukumuku amb una velocitat de 5 punts
```

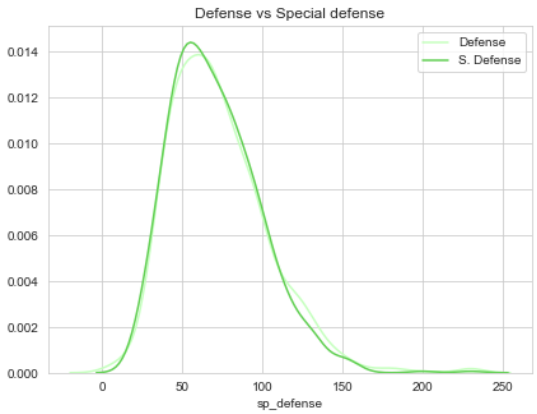
1.5.7 Atac i defensa

Comparar l'atac i la defensa base dels Pokemons.

[40]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

sns.distplot(pokemon_info_df['attack'], color="#B8F0FC", hist=False, ax=ax1,␣
↪label="Attack")
sns.distplot(pokemon_info_df["sp_attack"], color="#52BAD0", hist=False, ax=ax1,␣
↪label="S. Attack")
ax1.title.set_text("Attack vs Special Attack")

ax2.title.set_text("Defense vs Special defense")
sns.distplot(pokemon_info_df['defense'], color="#C6FFBF", hist=False, ax=ax2,␣
↪label="Defense")
sns.distplot(pokemon_info_df["sp_defense"], color="#61D052", hist=False,␣
↪ax=ax2, label="S. Defense")
```

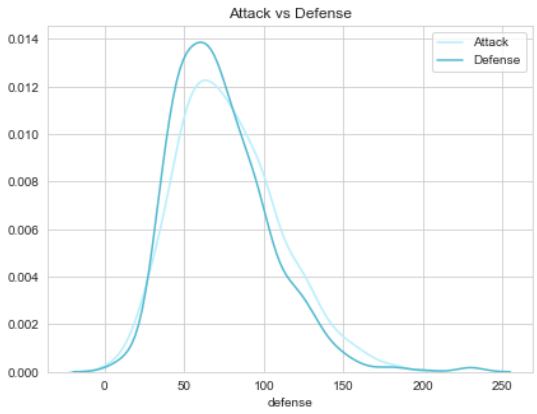
[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa795a99d0>
```

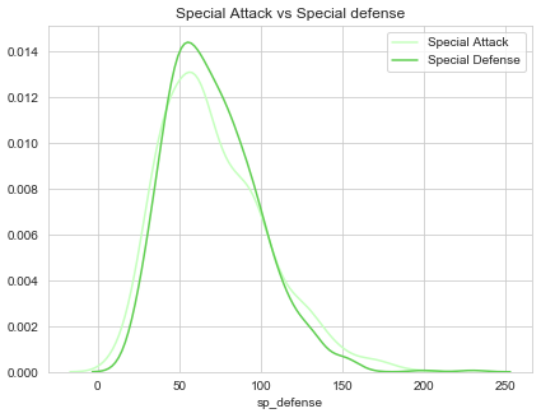




[41]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

ax1.title.set_text("Attack vs Defense")
sns.distplot(pokemon_info_df['attack'], color="#B8F0FC", hist=False, ax=ax1,␣
↪label="Attack")
sns.distplot(pokemon_info_df["defense"], color="#52BAD0", hist=False, ax=ax1,␣
↪label="Defense")

ax2.title.set_text("Special Attack vs Special defense")
sns.distplot(pokemon_info_df['sp_attack'], color="#C6FFBF", hist=False, ax=ax2,␣
↪label="Special Attack")
sns.distplot(pokemon_info_df["sp_defense"], color="#61D052", hist=False,␣
↪ax=ax2, label="Special Defense")
```

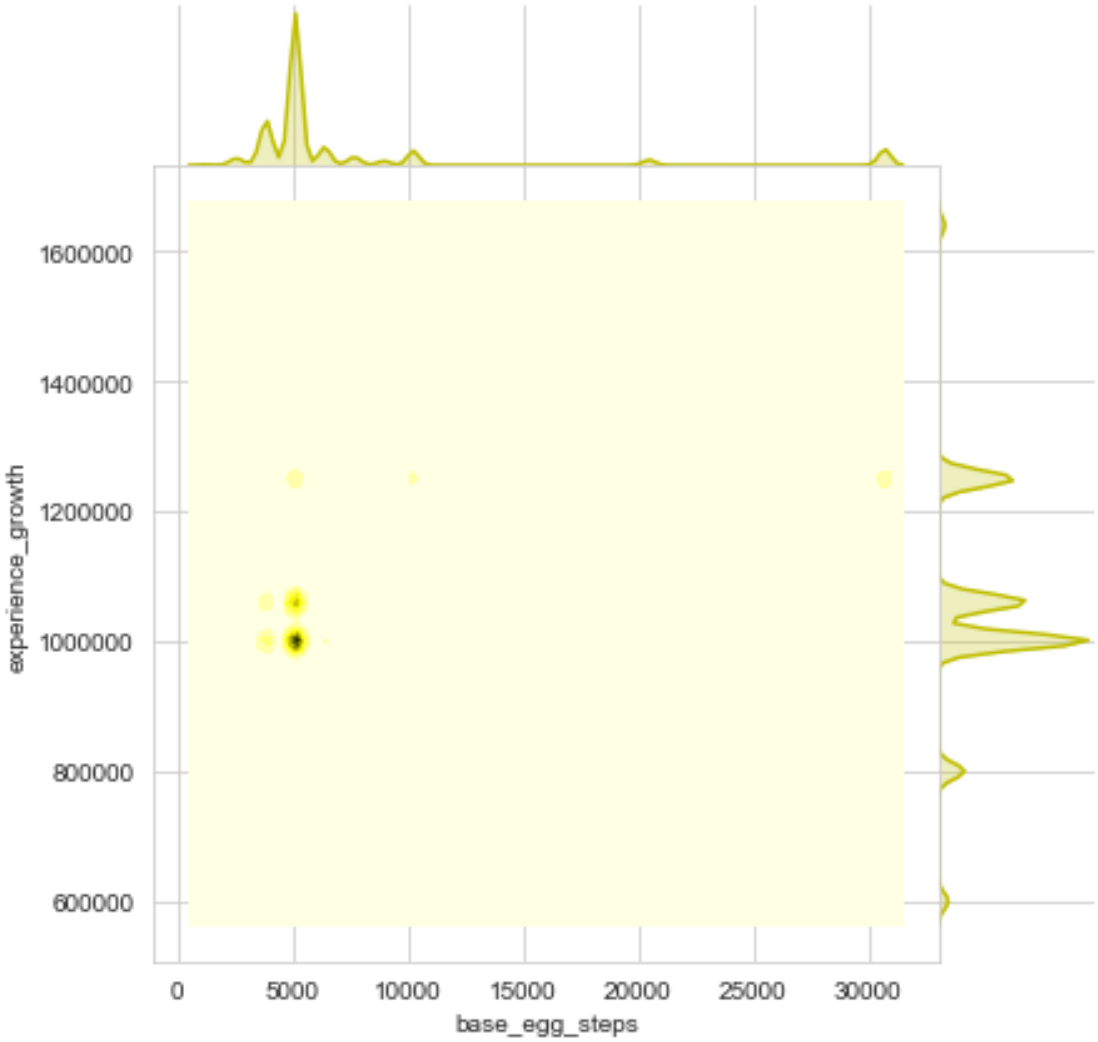
[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffa79faf450>





1.6 Ampliació de les dades

[42]:
```
g = sns.jointplot("base_egg_steps", "experience_growth", data=pokemon_info_df,␣
↪kind="kde", space=0, color="y")
```



1.7 Anàlisi de dades

En aquest apartat s'estudiarà la distribució que segueixen algunes de les variables i s'aplicaran contrastos de hipòtesi amb la finalitat d'extreure conclusions en base al tipus dels Pokemons.

S'ha decidit estudiar les variables *atac*, *hp*, *defensa* i *velocitat*.

1.7.1 Normalitat en la distribució

S'aplica un test de normalitat *shapiro-wilks* per veure si segueixen una distribució normal.

[43]:
```
sp.stats.shapiro(pokemon_info_df['attack'].to_numpy())
```

[43]:
```
(0.9794801473617554, 3.578010998595005e-09)
```

[44]:
```
sp.stats.shapiro(pokemon_info_df['hp'].to_numpy())
```

[44]:
```
(0.892400860786438, 3.323253268086282e-23)
```

[45]:
```
sp.stats.shapiro(pokemon_info_df['defense'].to_numpy())
```

[45]:
```
(0.9398431181907654, 1.7829034379355707e-17)
```

[46]:
```
sp.stats.shapiro(pokemon_info_df['speed'].to_numpy())
```

[46]:
```
(0.9813277721405029, 1.3825084188567871e-08)
```

[47]:
```
sp.stats.shapiro(pokemon_info_df['height_m'].to_numpy())
```

[47]:
```
(0.6370251774787903, 5.111117879034381e-38)
```

[48]:
```
sp.stats.shapiro(pokemon_info_df['weight_kg'].to_numpy())
```

[48]:
```
(0.5102710723876953, 2.9525358643323896e-42)
```

Els testos per les variables *attack*, *hp*, *defense* i *speed* han obtingut un *p-value* inferior al nivell de significació ($\alpha = 0.05$), i per tant hi ha evidències estadístiques suficients per rebutjar la hipòtesi nul · la i acceptar que no segueixen una distribució normal.

Per altre banda, les variables *height_m* i *weight_kg* han obtingut un *p-value* de 1, i per tant, es pot assumir que si segueixen una distribucció normal.

1.7.2 Homocedasticitat

Pes en els Pokemons de tipus **roca** i **foc**

Ara es vol saber si hi ha diferència en la variancia (heterocedasticitat) o no (homocedasticitat) per la variable *weight_kg* en base a si el seu primer tipus és roca (*rock*) o foc (*fire*). Com que presuntament la variable *weight_kg* segueix una distribució normal, es pot aplicar un test de *Levene*.

[49]:
```
rock_pokemons_array = pokemon_info_df[(pokemon_info_df['type1'] == 'rock') \
                                      & (pokemon_info_df['weight_kg'] !=␣
↪0)]['weight_kg'].to_numpy()
fire_pokemons_array = pokemon_info_df[(pokemon_info_df['type1'] == 'fire') \
```

```
                                    & (pokemon_info_df['weight_kg'] !=␣
↪0)]['weight_kg'].to_numpy()
sp.stats.levene(rock_pokemons_array, fire_pokemons_array)
```

[49]:
```
LeveneResult(statistic=1.689997161528986, pvalue=0.19695810336375189)
```

Com que s'ha obtingut un ***p-value*** de **0,197**, **no hi ha suficients evidències estadístiques per rebutjar la hipótesi nul · la**, i per tant, s'accepta amb un **nivell de confiança del 95%** que **no hi ha diferències entre les variancies dels Pokemons de tipus roca i els de tipus foc.**

1.7.3 Contrast

Pes dels Pokemons de tipus roca i foc

Es sap que la variable **weight_kg** segueix una distribució normal i que no hi ha diferència entre la variancia dels pokemons de tipus roca i tipus foc. Ara es vol contestar a la pregunta: Es pot considerar que els Pokemons de tipus roca i foc tenen la mateixa mitja de pes? Per això es pot aplicar un **t-test** on el contrast d'hipòtesis és:

H_0 : La mitja de **weight_kg** és igual pels Pokemons de tipus roca i foc.

H_1 : La mitja de **weight_kg** no és igual pels Pokemons de tipus roca i foc.

[50]:
```
sp.stats.ttest_ind(a = rock_pokemons_array, b = fire_pokemons_array)
```

[50]:
```
Ttest_indResult(statistic=1.4997248629874218, pvalue=0.13722476811641718)
```

Com que s'ha obtingut un ***p-value*** de **0,137**, **no hi ha evidències estadístiques suficients per rebutjar la hipòtesi nul · la**, i per tant **es pot considerar que la mitja de pes entre els Pokemons de tipus roca i de tipus foc és el mateix.**

1.8 Model predictiu

TODO: Descripció del model predictiu que es vol crear...

1.8.1 Pokemon_battles_df

Com que el *dataset* utilitzat fins el moment no conté la informació relacionada als combats, cal carregar el fitxer *combats.csv* que conté la següènt informació

- ***First_pokemon***: Index de la Pokedex pel primer Pokemon involucrat en el combat.
- ***Second_pokemon***: Índex de la Pokedex pel segon Pokemon involucrat en el combat.
- ***Winner***: Índex de la Pokedex pel Pokemon guanyador.

[52]:
```
pokemon_battles_df
```

[52]:

	First_pokemon	Second_pokemon	Winner
0	1	617	617

```
1                 31              617      617
2                 44              617      617
3                 72              617      617
4                 76              617       76
…                 …               …         …
38738            650              283      283
38739            659              283      283
38740            660              283      660
38741            700              283      283
38742            708              283      708

[38743 rows x 3 columns]
```

El primer que cal fer es relacionar el *dataset* que conté la informació dels Pokemons (*pokemon_info_df*) amb el *dataset* dels combats (*pokemon_battles_df*).

Per això apliquem dos *joins*, el primer que relaciona aquests dos datasets per obtenir les dades del primer Pokemon i el segon *join* on es tornen a relacionar aquest dos *datasets* però aquesta vegada per obtenir la informació del segón *Pokemon* implicat.

[53]:
```
pokemon_battles_info_df = pokemon_battles_df.merge(pokemon_info_df, \
                                                   left_on='First_pokemon', \
                                                   right_on='pokedex_number'␣
↪\
                                                  ).merge(pokemon_info_df, \
                                                          ␣
↪left_on='Second_pokemon', \
                                                          ␣
↪right_on='pokedex_number' \
                                                          )[['First_pokemon',␣
↪'Second_pokemon', \
                                                             'Winner',␣
↪'name_x', 'attack_x', \
                                                             'sp_attack_x',␣
↪'defense_x', 'sp_defense_x', \
                                                             'hp_x',␣
↪'speed_x', 'type1_x','is_legendary_x', \
                                                             'name_y',␣
↪'attack_y', \
                                                             'sp_attack_y',␣
↪'defense_y', 'sp_defense_y', \
                                                             'hp_y',␣
↪'speed_y', 'type1_y','is_legendary_y']]
```

El *dataset* resultat d'aplicar els *joins* conté per nom *field_x* pel resultat del primer join i *field_y* pel resultat del segon join. Per això apliquem un rename on els camps del primer Pokemon comencen per *First_pokemon* i els camps del segón Pokemon per *Second_pokemon* s'han

[54]:
```
pokemon_battles_info_df.rename(columns={'name_x': 'First_pokemon_name',␣
↪'attack_x': 'First_pokemon_attack', \
                                        'sp_attack_x':␣
↪'First_pokemon_sp_attack', 'defense_x': 'First_pokemon_defense', \
                                        'sp_defense_x':␣
↪'First_pokemon_sp_defense', 'hp_x': 'First_pokemon_hp', \
                                        'speed_x': 'First_pokemon_speed',␣
↪'type1_x': 'First_pokemon_type1', \
                                        'is_legendary_x':␣
↪'First_pokemon_is_legendary', 'name_y': 'Second_pokemon_name', \
                                        'attack_y': 'Second_pokemon_attack',␣
↪'sp_attack_y': 'Second_pokemon_sp_attack', \
                                        'defense_y': 'Second_pokemon_defense',␣
↪'sp_defense_y': 'Second_pokemon_sp_defense', \
                                        'hp_y': 'Second_pokemon_hp', 'speed_y':␣
↪'Second_pokemon_speed', \
                                        'type1_y': 'Second_pokemon_type1',␣
↪'is_legendary_y': 'Second_pokemon_is_legendary'}, \
                                inplace=True)
```

[55]:
```
#https://www.kaggle.com/jonathanbouchet/pokemon-battles
```

1.8.2 Camps *diff_?*

Per construir el model predictiu cal calcular els camps amb les diferències entre les propietats implicades, aquestes diferències s'anomenaran *Diff_?*. Per exemple, la diferència d'atac es dirà: *Diff_attack*

[56]:
```
pokemon_battles_info_df['Diff_attack'] =␣
↪pokemon_battles_info_df['First_pokemon_attack'] -␣
↪pokemon_battles_info_df['Second_pokemon_attack']

pokemon_battles_info_df['Diff_sp_attack'] =␣
↪pokemon_battles_info_df['First_pokemon_sp_attack'] -␣
↪pokemon_battles_info_df['Second_pokemon_sp_attack']

pokemon_battles_info_df['Diff_defense'] =␣
↪pokemon_battles_info_df['First_pokemon_defense'] -␣
↪pokemon_battles_info_df['Second_pokemon_defense']

pokemon_battles_info_df['Diff_sp_defense'] =␣
↪pokemon_battles_info_df['First_pokemon_sp_defense'] -␣
↪pokemon_battles_info_df['Second_pokemon_sp_defense']
```

```
pokemon_battles_info_df['Diff_hp'] =␣
↪pokemon_battles_info_df['First_pokemon_hp'] -␣
↪pokemon_battles_info_df['Second_pokemon_hp']

pokemon_battles_info_df['Diff_speed'] =␣
↪pokemon_battles_info_df['First_pokemon_speed'] -␣
↪pokemon_battles_info_df['Second_pokemon_speed']
```

1.8.3 Camp *winner_result*

Com que l'objectiu d'aquest model predictiu és fer una classificació on el resultat sigui 0 si guanya el primer Pokemon o 1 en cas contrari. Afegim el camp ***Winner_result*** amb aquest càlcul.

[57]:
```
pokemon_battles_info_df['Winner_result'] = np.where(\
                                                    ␣
↪pokemon_battles_info_df['First_pokemon'] == \
                                                    ␣
↪pokemon_battles_info_df['Winner'], 0, 1)
```

1.8.4 Seleccionar els camps del model

Ara creem el *dataset* ***pokemon_battles_pred_df*** amb els camps que s'usaran com a predictors, que són:

- ***Diff_attack***
- ***Diff_sp_attack***
- ***Diff_defense***
- ***Diff_sp_defense***
- ***Diff_hp***
- ***Diff_speed***
- ***First_pokemon_is_legendary***
- ***Second_pokemon_is_legendary***

I el *dataset* ***pokemon_battles_res_df*** amb el camp resultat que és *Winner_result*

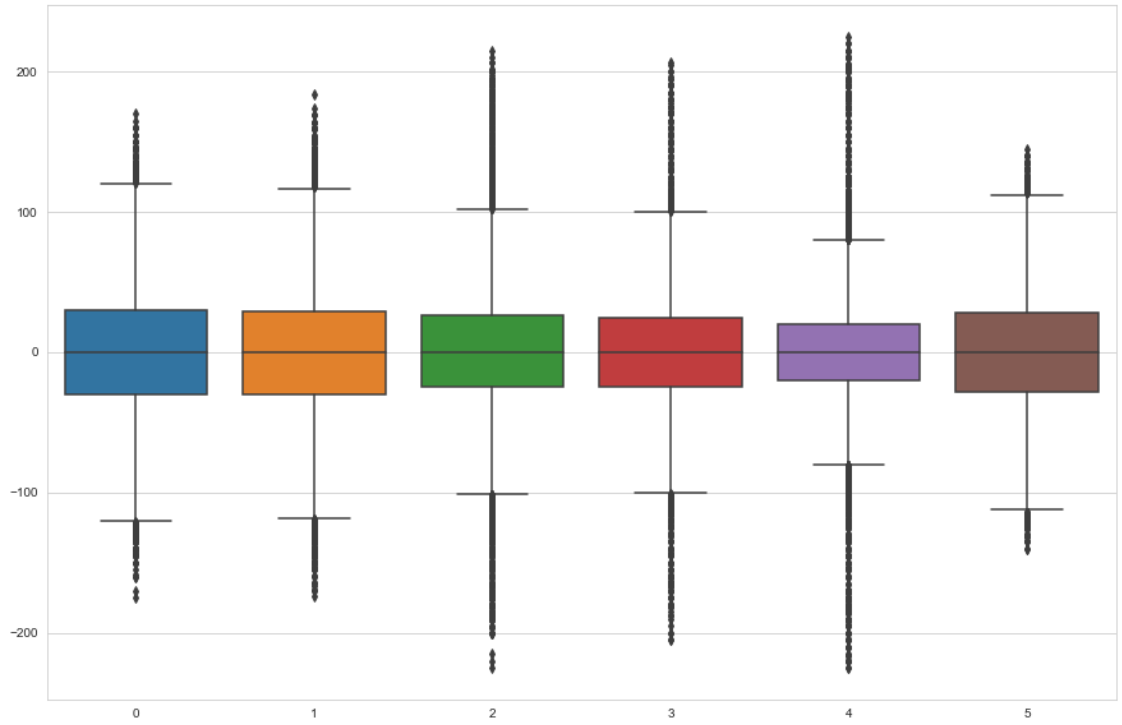
[58]:
```
pokemon_battles_pred = pokemon_battles_info_df[['Diff_attack',␣
↪'Diff_sp_attack', \
                                                'Diff_defense',␣
↪'Diff_sp_defense', \
                                                'Diff_hp', 'Diff_speed', \
                                                'First_pokemon_is_legendary',
                                                'Second_pokemon_is_legendary']].
↪values

pokemon_battles_res = pokemon_battles_info_df['Winner_result'].values
```

1.8.5 Escalar les dades

Si els rangs de valors per les variables utilitzades en el model és considerablment diferent, poden causar distorsions en els resultats obtinguts. Per mostrar la seva distribució es pot utilitzar un *boxplot*.

[59]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(data=pokemon_battles_pred[:,0:6], orient='v')
```

[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa7a087950>
```



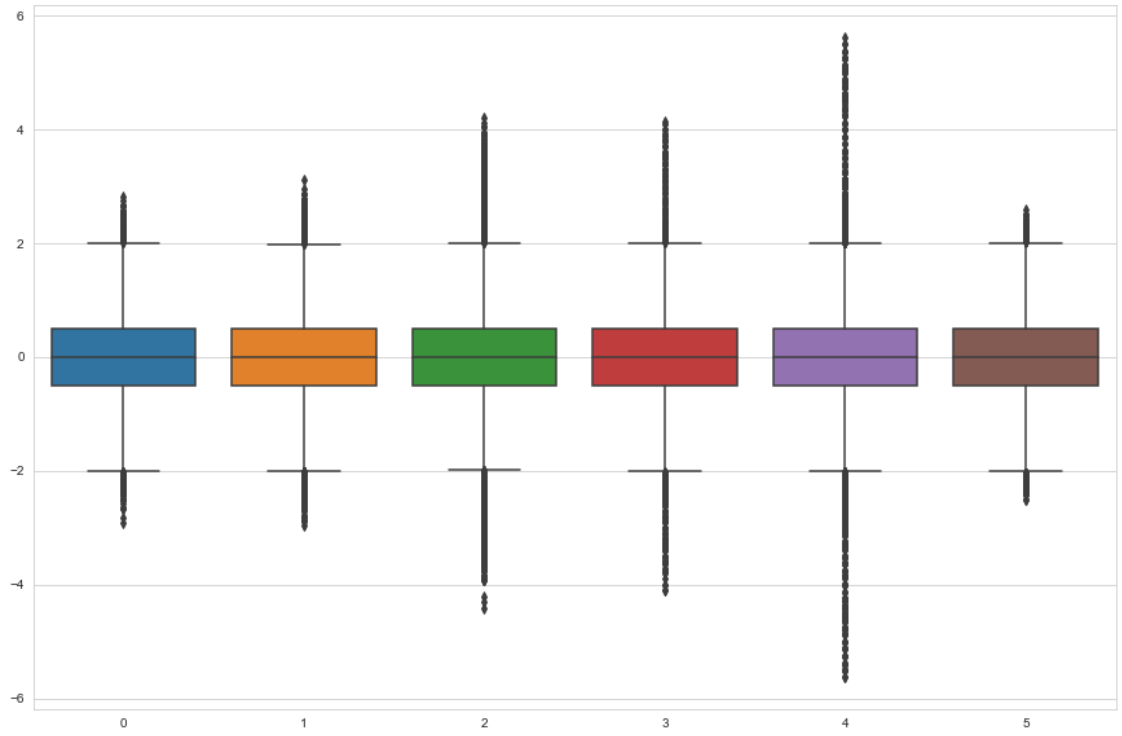
Com es pot observar hi ha forçes valors atípics, per això es pot aplicar una estandardització basada en una transofrmació que redueixi el nombre de valors utiliers. Aquesta estandardització es basa en:

$z = \frac{x_i - x}{s}$

[60]:
```
from sklearn.preprocessing import RobustScaler
rs = RobustScaler()

rs.fit(pokemon_battles_pred)

pokemon_battles_pred = rs.transform(pokemon_battles_pred)
```

[61]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(data=pokemon_battles_pred[:,0:6], orient='v')
```

[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa79d59fd0>
```



1.8.6 Separar les dades en *dades d'entrenament* i *dades de prova*

Com que és un **model supervisat**, cal separar les dades en dades d'entrenament i dades de prova. El model utilitzarà les dades d'entrenament per aprendre (fase d'entrenament) i les dades de prova per comprovar si el que ha aprés és o no correcte (fase de test).

Com que **hi ha una quantitat relativament alta de registres** (50,000), s'ha decidit utilitzar un **80% de les dades per a l'entrenament** (40,000 registres) i **un 20% pel test** (10,000 registres).

[62]:
```
from sklearn.model_selection import train_test_split

#S'ha decidit assignar el valor 23 a la llavor per així obtenir sempre el␣
 ↪mateix resultat.
pokemon_battle_pred_train, pokemon_battle_pred_test, \
pokemon_battle_res_train, pokemon_battle_res_test = train_test_split(\
                                                                    ␣
 ↪pokemon_battles_pred, \
```

```
↪pokemon_battles_res, \
↪test_size=0.2, random_state = 23)
```

Article sobre l'escalat, estandarització i normalització

1.8.7 Crear el model de regressió logística

[63]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(pokemon_battle_pred_train, pokemon_battle_res_train)
```

[63]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[64]:
```
pokemon_battle_results = classifier.predict(pokemon_battle_pred_test)
```

[65]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(pokemon_battle_res_test, pokemon_battle_results)
```

[66]:
```
cm
```

[66]:
```
array([[3167,  438],
       [ 456, 3688]])
```

[67]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = pokemon_battles_pred,␣
↪y = pokemon_battles_res, cv = 10, scoring='accuracy')
print('Mean: {}, standard deviation: {}'.format(accuracies.mean(), accuracies.
↪std()))
```

```
Mean: 0.8796945659233601, standard deviation: 0.010370394451956502
```

1.8.8 K nearest Neighbours (*Knn*)

[68]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
knn_classifier.fit(X=pokemon_battle_pred_train, y=pokemon_battle_res_train)
```

[68]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
```

```
                    weights='uniform')
```

[69]:
```
knn_pokemon_battle_results = knn_classifier.predict(pokemon_battle_pred_test)
```

[70]:
```
knn_cm = confusion_matrix(pokemon_battle_res_test, knn_pokemon_battle_results)
```

[71]:
```
knn_cm
```

[71]:
```
array([[3182,  423],
       [ 458, 3686]])
```

[72]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = knn_classifier, X =␣
 ↪pokemon_battles_pred, y = pokemon_battles_res, cv = 10, scoring='accuracy')
print('Mean: {}, standard deviation: {}'.format(accuracies.mean(), accuracies.
 ↪std()))
```

```
Mean: 0.8758741252685397, standard deviation: 0.008856867734178363
```

1.8.9 Support Vector Machine - SVM

[73]:
```
from sklearn.svm import SVC
svm_classifier = SVC(kernel='rbf', random_state=0)
```

[74]:
```
svm_classifier = svm_classifier.fit(X=pokemon_battle_pred_train,␣
 ↪y=pokemon_battle_res_train)
```

[75]:
```
svm_pokemon_battle_results = svm_classifier.predict(X=pokemon_battle_pred_test)
svm_cm = confusion_matrix(pokemon_battle_res_test, svm_pokemon_battle_results)
```

[76]:
```
svm_cm
```

[76]:
```
array([[3317,  288],
       [ 356, 3788]])
```

[77]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = svm_classifier, X =␣
 ↪pokemon_battles_pred, y = pokemon_battles_res, cv = 10, scoring='accuracy')
print('Mean: {}, standard deviation: {}'.format(accuracies.mean(), accuracies.
 ↪std()))
```

```
Mean: 0.9092222359151998, standard deviation: 0.007967037350509649
```

1.8.10 Classificació per xarxa bayesiana (*Naive bayes*)

[78]:
```
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
```

[79]:
```
nb_classifier = nb_classifier.fit(X=pokemon_battle_pred_train,␣
↪y=pokemon_battle_res_train)
nb_pokemon_battle_results = nb_classifier.predict(X=pokemon_battle_pred_test)
```

[80]:
```
nb_cm = confusion_matrix(pokemon_battle_res_test, nb_pokemon_battle_results)
nb_cm
```

[80]:
```
array([[2946,  659],
       [ 817, 3327]])
```

[81]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = nb_classifier, X =␣
↪pokemon_battles_pred, y = pokemon_battles_res, cv = 10, scoring='accuracy')
print('Mean: {}, standard deviation: {}'.format(accuracies.mean(), accuracies.
↪std()))
```

```
Mean: 0.7994998584442187, standard deviation: 0.01358469466986609
```

1.8.11 Random Forest Classifier (RFC)

[82]:
```
from sklearn.ensemble import RandomForestClassifier
```

[83]:
```
rfc_classifier = RandomForestClassifier(n_estimators=10, criterion='entropy',␣
↪random_state=0)
rfc_classifier = rfc_classifier.fit(X=pokemon_battle_pred_train,␣
↪y=pokemon_battle_res_train)
```

[84]:
```
rfc_pokemon_battle_results = rfc_classifier.predict(X=pokemon_battle_pred_test)
rfc_cm = confusion_matrix(pokemon_battle_res_test, rfc_pokemon_battle_results)
```

[85]:
```
rfc_cm
```

[85]:
```
array([[3415,  190],
       [ 326, 3818]])
```

[86]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = rfc_classifier, X =␣
↪pokemon_battles_pred, y = pokemon_battles_res, cv = 10, scoring='accuracy')
print('Mean: {}, standard deviation: {}'.format(accuracies.mean(), accuracies.
↪std()))
```

```
Mean: 0.9251219411461022, standard deviation: 0.007323388880667111
```

1.9 Millorar el model (afegir el tipus dels *Pokemons*)

Com s'ha mostrat en apartats anteriors, cada Pokemon té un tipus base i pot tenir un segon tipus. Evidentment, aquestes propietats influeixen alhora de determinar el guanyador en un combat, per exemple, un Pokemon d'aigua més debil (menys atac, defensa, vida, etc.) pot guanyar amb més facilitat a un Pokemon de foc que a un Pokemon de planta.

Per això, anem a calcular una nova propietat que determini l'eficacia en base al tipus de Pokemon. Aquesta propietat vindrà definida en funció del primer i segon tipus del Pokemon (*type1* i *type2*) i la seva devilitat en vers als altres tipus (*against_?*).

D'aquesta manera, si comparem els Pokemons *Pikachu* (elèctric/elèctric) i *Onix* (roca/terra), té avantatge l'Ònix perquè no té debilitat en vers a l'electricitat (*against_electric = 0*) i en canvi, en Pikachu té debilitat per la roca (*against_rock = 1*) i per la terra (*against_ground = 2*).

Per obtenir un valor numèric, s'aplica la formula:

$f(p1, p2) = g(p1, p2) - g(p2, p1)$

On:

- $g(p1, p2) = dbt1(p1, p2) * ft1 + dbt2(p1, p2) * ft2$
- $dbt1(p1, p2)$ = Debilitat del Pokemon p2 en vers al primer tipus del Pokemon p1.
- $dbt2(p1, p2)$ = Debilitat del Pokemon p2 en vers el segon tipus del Pokemon p1.
- $ft1$ = Factor arbitrari per ponderar el tipus 1
- $ft2$ = Factor arbitrari per ponderar el tipus 2

D'aquesta manera, per l'exemple de l'*Onix* vs *Pikachu* donat:

- *Onix*: *type1* = rock, *type2* = ground, *against_electric* = 0
- *Pikachu*: *type1* = eletric, *type2* = eletric, *against_rock* = 1, *against_ground*= 2
- $ft1 = 1$
- $ft2 = 0.3$

tenim:

$f(Onix, Pikachu) = (1 * 1 + 2 * 0.3) - (0 * 1 + 0 * 0.3) = 1.6$

Com era d'esperar, degut a que els *Pokemons* de tipus roca i terra tenen avantatge davant dels *Pokemons* de tipus elèctric, s'ha obtingut un valor positiu.

[87]:
```
def effectivity_against(pokemon1, pokemon2, effectivity_type1,␣
↪effectivity_type2):
    type1 = pokemon1['type1'].iloc[0]
    type2 = pokemon1['type2'].iloc[0]
    against_type1 = pokemon2['against_'+type1].iloc[0]
    if type2 == 'unknown':
        return against_type1 * effectivity_type1
    else:
        against_type2 = pokemon2['against_'+type2].iloc[0]
```

```
        return (against_type1 * effectivity_type1) + (against_type2 *␣
↪effectivity_type2)
```

[88]:
```
def balance_effectivity_against(pokemon1, pokemon2, effectivity_type1 = 1,␣
↪effectivity_type2 = 0.3):
    return effectivity_against(pokemon1, pokemon2, \
                               effectivity_type1, effectivity_type2) -␣
↪effectivity_against(pokemon2, pokemon1, \
                               effectivity_type1, effectivity_type2)
```

[89]:
```
def balance_effectivity_against_by_pokedex_number(pokemon_number1,␣
↪pokemon_number2, \
                                                  effectivity_type1 = 1,␣
↪effectivity_type2 = 0.3):
    pokemon1 = pokemon_info_df[pokemon_info_df['pokedex_number'] ==␣
↪pokemon_number1]
    pokemon2 = pokemon_info_df[pokemon_info_df['pokedex_number'] ==␣
↪pokemon_number2]
    return balance_effectivity_against(pokemon1, pokemon2, effectivity_type1,␣
↪effectivity_type2)
```

Ara cal afegir la propietat *balance_effectivity* al *dataframe pokemon_battles_info_df*

[90]:
```
pokemon_battles_info_df['balance_effectivity'] = [\
                                                  ␣
↪balance_effectivity_against_by_pokedex_number(\
                                                      row['First_pokemon'], \
                                                      row['Second_pokemon']) \
                                                  for index, row in␣
↪pokemon_battles_df.iterrows()\
                                                  ]
```

S'afegeix la columna *balance_effectivity* al *dataframe pokemon_battles_pred*

[91]:
```
pokemon_battles_improved_pred = pokemon_battles_info_df[['Diff_attack',␣
↪'Diff_sp_attack', \
                                                         'Diff_defense',␣
↪'Diff_sp_defense', \
                                                         'Diff_hp',␣
↪'Diff_speed', \
                                                        ␣
↪'First_pokemon_is_legendary',
                                                        ␣
↪'Second_pokemon_is_legendary',
                                                        ␣
↪'balance_effectivity']].values
```

En base a tota la informació que tenim, ara podem questionar-nos quin és el combat més desigual que hi ha en el *dataset pokemon_battles_df*. Per això busquem el mínim i màxim del sumatori de totes les variables *Diff_?* afegint la variable *First_pokemon_is_legendary* en positiu, *Second_pokemon_is_legendary* en negatiu i balance_effectivity

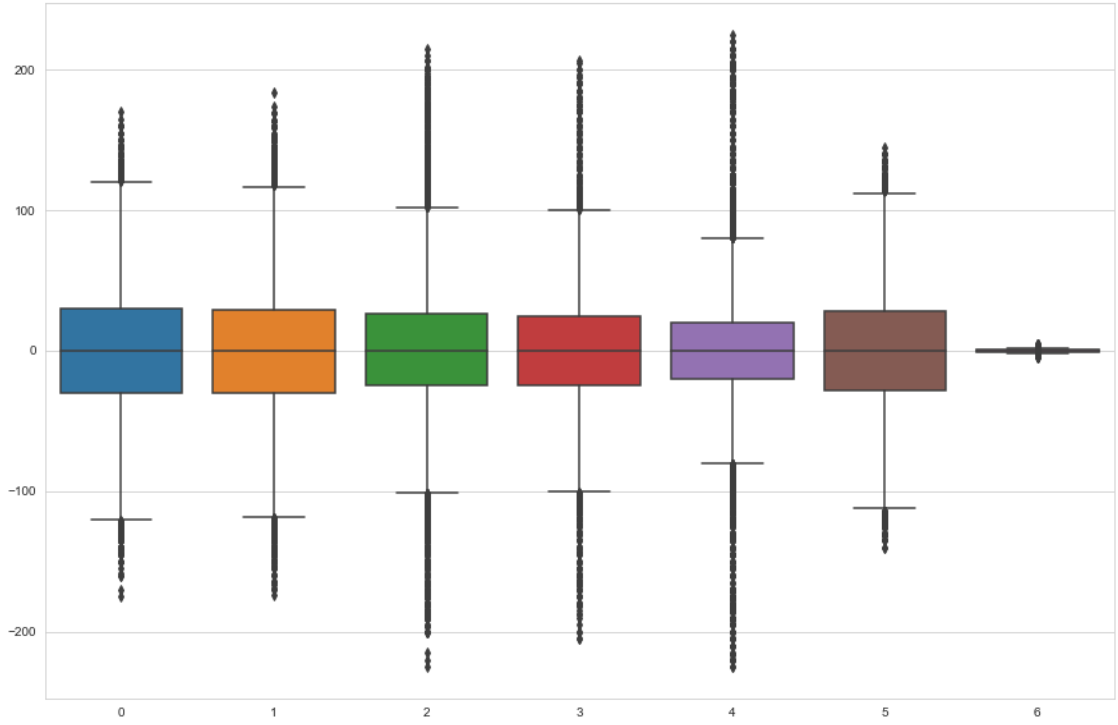
[92]:
```
pokemon_battles_pred[0]
```

[92]:
```
array([-0.35      , -0.59322034,  0.17647059,  0.1       , -0.875     ,
       -1.78571429,  0.        ,  0.        ])
```

Distribució de les variables

[93]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(data=pokemon_battles_improved_pred[:,[0,1,2,3,4,5,8]], orient='v')
```

[93]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa7654d9d0>
```

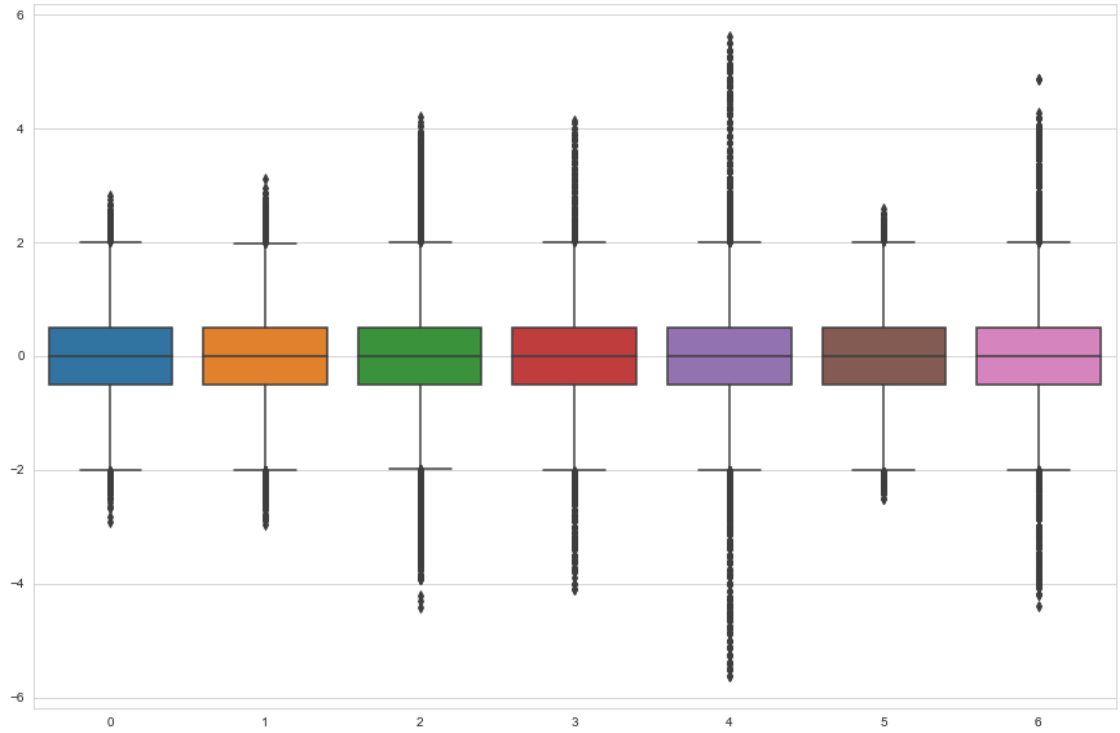


Es normalitzen altre vegada les variables numèriques.

[95]:
```
rs = RobustScaler()

rs.fit(pokemon_battles_improved_pred)

pokemon_battles_improved_pred = rs.transform(pokemon_battles_improved_pred)
```

```
plt.subplots(figsize=(15,10))
sns.boxplot(data=pokemon_battles_improved_pred[:,[0,1,2,3,4,5,8]], orient='v')
```

[95]: <matplotlib.axes._subplots.AxesSubplot at 0x7ffa66fad690>



Un cop escalades les dades tornem a separar-les en un conjunt d'entrenament i un de prova.

[96]:
```
pokemon_battles_improved_pred_train, pokemon_battles_improved_pred_test, \
pokemon_battles_improved_res_train, pokemon_battles_improved_res_test =␣
↪train_test_split(\
                                                                        ␣
↪pokemon_battles_improved_pred, \
                                                                        ␣
↪pokemon_battles_res, \
                                                                        ␣
↪test_size=0.2, random_state = 23)
```

***Random forest* millorat** Calculat l'atribut *balance_effectivity* que té en compte el tipus dels Pokemons involucrats en el combat, tornem a crear el model basat en *random forest* (ja que és amb el que hem obtingut un major *accuracy*) per veure si millorem els resultats.

[98]:
```
improved_rfc_classifier = RandomForestClassifier(n_estimators=10,␣
↪criterion='entropy', random_state=0)
improved_rfc_classifier = improved_rfc_classifier.fit(\
                                                      ␣
↪X=pokemon_battles_improved_pred_train, \
                                                      ␣
↪y=pokemon_battles_improved_res_train)
```

[100]:
```
improved_rfc_pokemon_battle_results = improved_rfc_classifier.
↪predict(X=pokemon_battles_improved_pred_test)
improved_rfc_cm = confusion_matrix(pokemon_battles_improved_res_test,␣
↪improved_rfc_pokemon_battle_results)
```

[101]:
```
improved_rfc_cm
```

[101]:
```
array([[3405,  200],
       [ 332, 3812]])
```

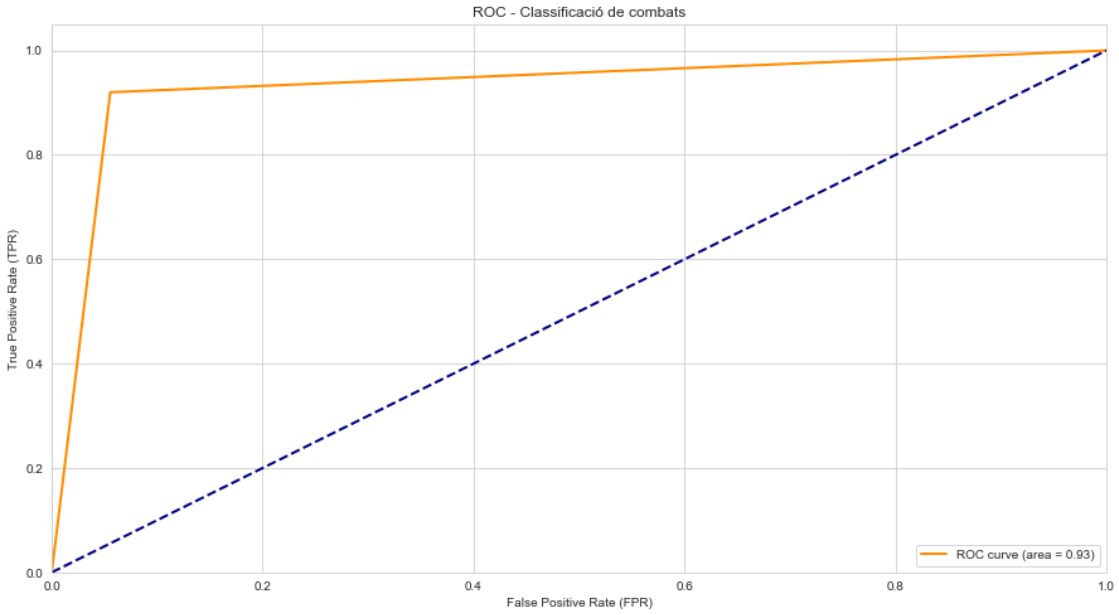
[104]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = improved_rfc_classifier, X =␣
↪pokemon_battles_improved_pred, y = pokemon_battles_res, cv = 10,␣
↪scoring='accuracy')
print('Mean: {}, standard deviation: {}'.format(accuracies.mean(), accuracies.
↪std()))
```

```
Mean: 0.9256382700218161, standard deviation: 0.006970821084828071
```

1.10 Corba ROC

[105]:
```
from sklearn.metrics import roc_curve, auc

fpr, tpr, _ = roc_curve(y_true=pokemon_battles_improved_res_test ,␣
↪y_score=improved_rfc_pokemon_battle_results)
auc = auc(fpr, tpr)
```

[106]:
```
plt.subplots(figsize=(15, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
↪auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC - Classificació de combats')
plt.legend(loc="lower right")
plt.show()
```



2 Torneig *Pokemon*

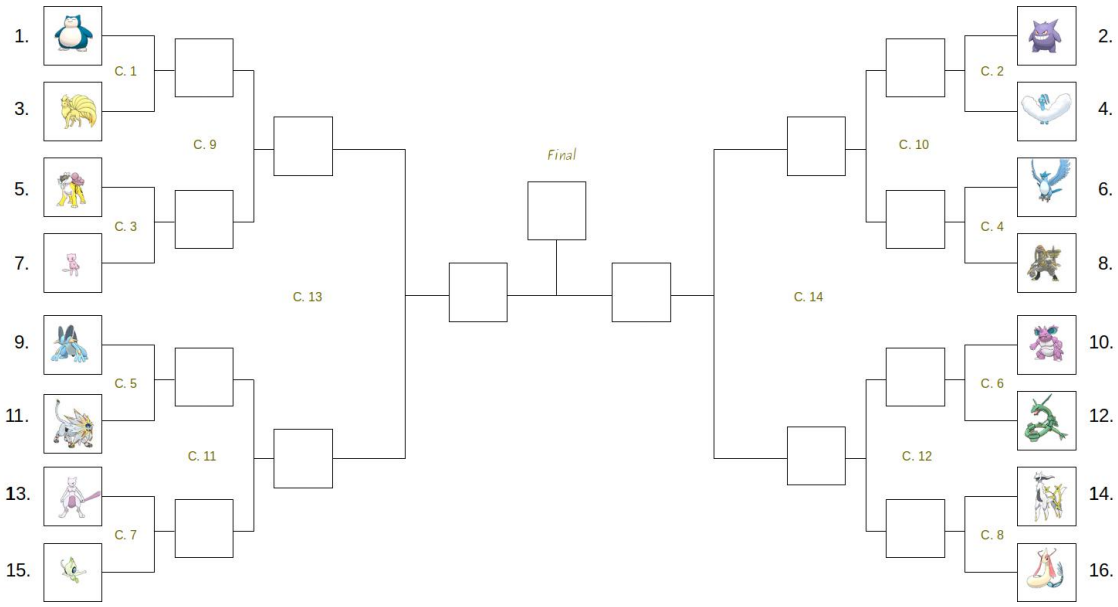
Per comprovar l'efectivitat del model de predicció creat s'ha decidit realitzar un Torneig *Pokemon*, on hi participen **16** ***Pokemons***, **8** dels quals **són llegendaris**. El Torneig consta de **8 combats** dividits en **4 fases**.

[139]:
```
# Construeix les dades del combat que enfronta el pokemon1 contra el pokemon2,
#les dades retornades ja estan normalitzades.
def build_fight(name_pokemon1, name_pokemon2):
    pokemon1 = pokemon_info_df[pokemon_info_df['name'] == name_pokemon1].iloc[0]
    pokemon2 = pokemon_info_df[pokemon_info_df['name'] == name_pokemon2].iloc[0]
    return rs.transform(pd.DataFrame.from_dict({'Diff_attack':␣
 ↪[pokemon1['attack']-pokemon2['attack']],\
        'Diff_sp_attack': [pokemon1['sp_attack']-pokemon2['sp_attack']],\
        'Diff_defense': [pokemon1['defense']-pokemon2['defense']],\
        'Diff_sp_defense': [pokemon1['sp_defense']-pokemon2['sp_defense']],\
        'Diff_hp': [pokemon1['hp']-pokemon2['hp']],\
        'Diff_speed': [pokemon1['speed']-pokemon2['speed']],\
        'First_pokemon_is_legendary': [pokemon1['is_legendary']],\
        'Second_pokemon_is_legendary': [pokemon2['is_legendary']],\
        'balance_effectivity': [balance_effectivity_against_by_pokedex_number(\
            pokemon1['pokedex_number'], pokemon2['pokedex_number'])]}))
```

[217]:
```
# Realitza una lluita que enfronta el Pokemon1 contra el Pokemon2 i
# fa la predicció del guanyador amb el classifier
```

```
def fight(classifier, name_pokemon1, name_pokemon2):
    pokemon_fight = build_fight(name_pokemon1, name_pokemon2)

    #Make the prediction
    result = classifier.predict_proba(X=pokemon_fight)

    if result[0][0] > 0.5:
        print('The winner is: {} with a probability of: {}%'.
↪format(name_pokemon1, (result[0][0]*100)))
    else:
        print('The winner is: {} with a probability of: {}%'.
↪format(name_pokemon2, (result[0][1]*100)))
```

2.1 Round 1



[233]:
```
fight1 = fight(classifier=improved_rfc_classifier, name_pokemon1='Snorlax',␣
↪name_pokemon2='Ninetales')
```

```
The winner is: Snorlax with a probability of: 70.0%
```

[235]:
```
fight2 = fight(classifier=improved_rfc_classifier, name_pokemon1='Gengar',␣
↪name_pokemon2='Altaria')
```

```
The winner is: Altaria with a probability of: 60.0%
```

[236]:
```
fight3 = fight(classifier=improved_rfc_classifier, name_pokemon1='Raikou',␣
↪name_pokemon2='Mew')
```

```
The winner is: Raikou with a probability of: 70.0%
```

[237]:
```
fight4 = fight(classifier=improved_rfc_classifier, name_pokemon1='Articuno',␣
↪name_pokemon2='Kommo-o')
```

```
The winner is: Kommo-o with a probability of: 70.0%
```

[238]:
```
fight5 = fight(classifier=improved_rfc_classifier, name_pokemon1='Swampert',␣
↪name_pokemon2='Solgaleo')
```

```
The winner is: Swampert with a probability of: 60.0%
```

[239]:
```
fight6 = fight(classifier=improved_rfc_classifier, name_pokemon1='Nidoking',␣
↪name_pokemon2='Rayquaza')
```

```
The winner is: Rayquaza with a probability of: 100.0%
```

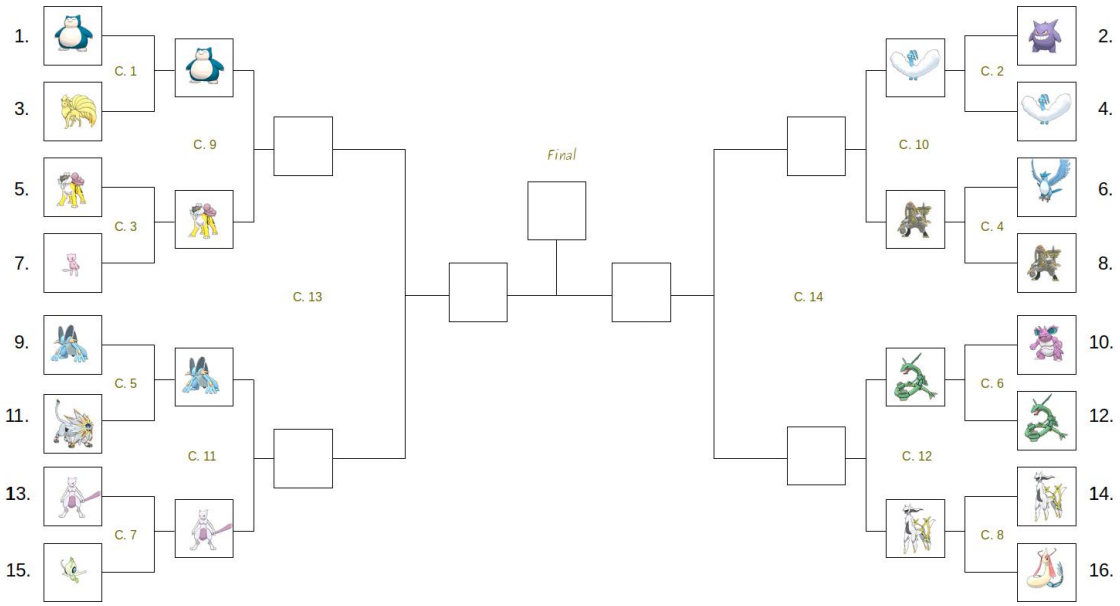
[240]:
```
fight7 = fight(classifier=improved_rfc_classifier, name_pokemon1='Mewtwo',␣
↪name_pokemon2='Celebi')
```

```
The winner is: Mewtwo with a probability of: 70.0%
```

[241]:
```
fight8 = fight(classifier=improved_rfc_classifier, name_pokemon1='Arceus',␣
↪name_pokemon2='Milotic')
```

```
The winner is: Arceus with a probability of: 90.0%
```

2.2 Round 2



[206]:
```
fight9 = fight(classifier=improved_rfc_classifier, name_pokemon1='Snorlax',␣
↪name_pokemon2='Raikou')
```

```
The winner is: Snorlax with a probability of: 70.0%
```

[243]:
```
fight10 = fight(classifier=improved_rfc_classifier, name_pokemon1='Altaria',␣
↪name_pokemon2='Kommo-o')
```

```
The winner is: Kommo-o with a probability of: 80.0%
```

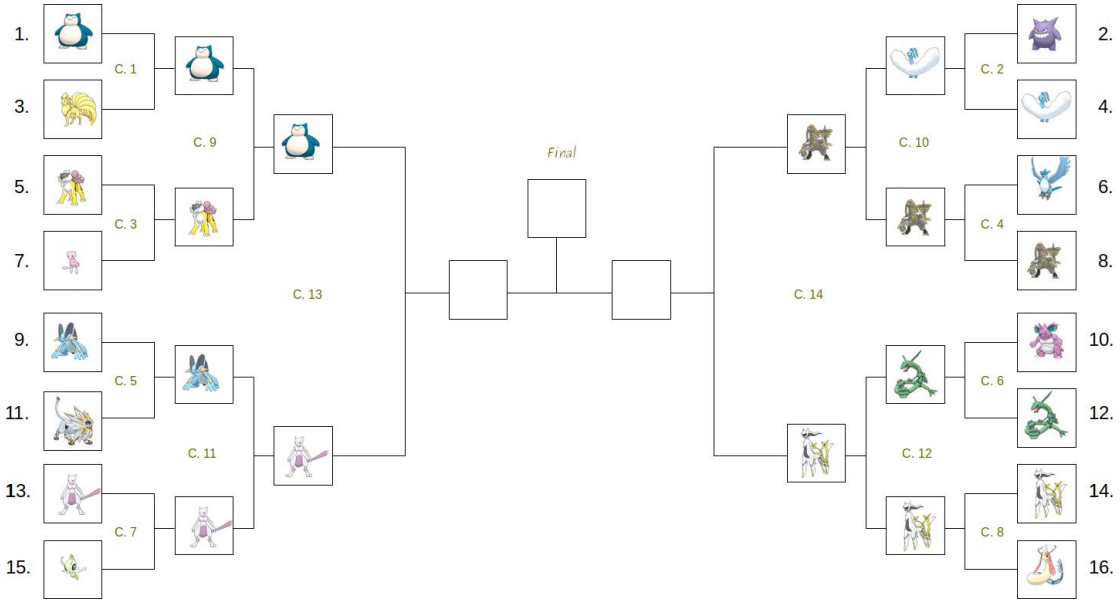
[242]:
```
fight11 = fight(classifier=improved_rfc_classifier, name_pokemon1='Swampert',␣
↪name_pokemon2='Mewtwo')
```

```
The winner is: Mewtwo with a probability of: 70.0%
```

[209]:
```
fight12 = fight(classifier=improved_rfc_classifier, name_pokemon1='Rayquaza',␣
↪name_pokemon2='Arceus')
```

```
The winner is: Arceus with a probability of: 70.0%
```

2.3 Round 3



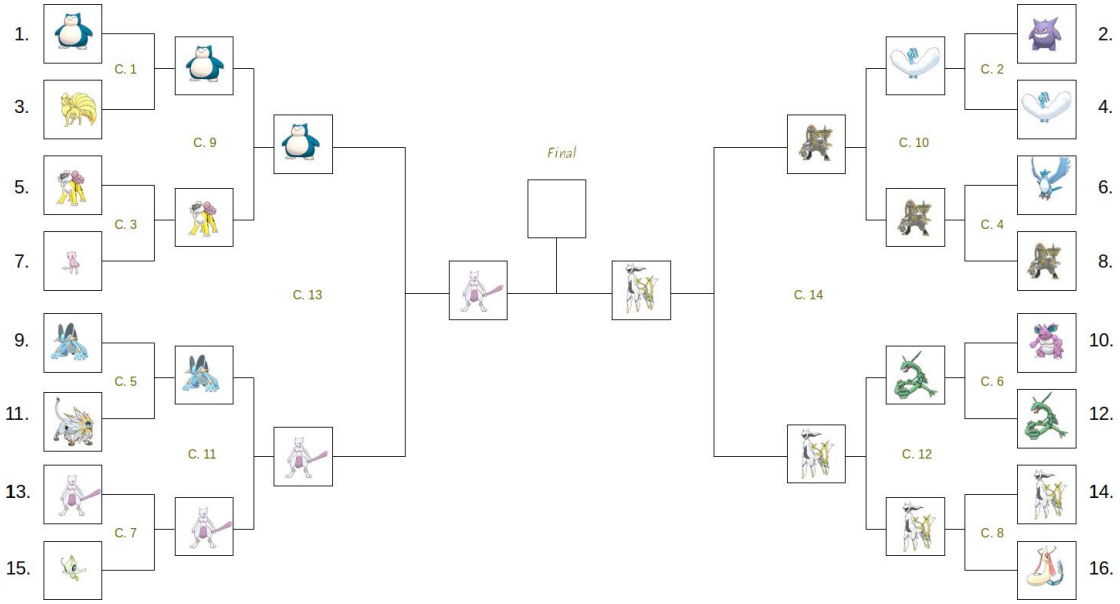
[210]:
```
fight9 = fight(classifier=improved_rfc_classifier, name_pokemon1='Snorlax',␣
↪name_pokemon2='Mewtwo')
```

```
The winner is: Mewtwo with a probability of: 90.0%
```

[244]:
```
fight10 = fight(classifier=improved_rfc_classifier, name_pokemon1='Kommo-o',␣
↪name_pokemon2='Arceus')
```

```
The winner is: Arceus with a probability of: 100.0%
```

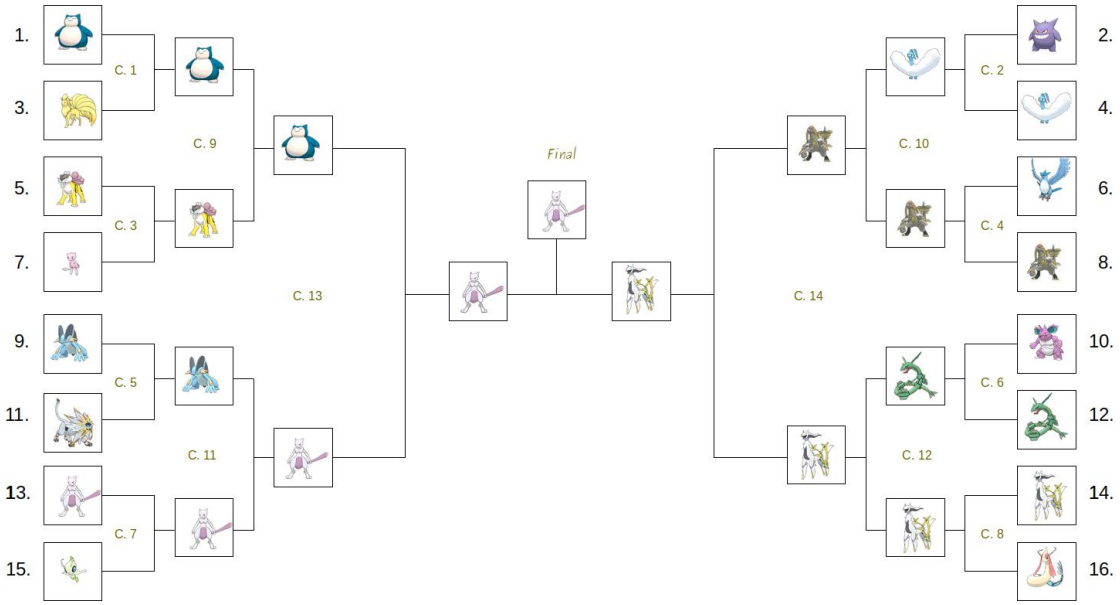
2.4 Round 4



[245]:
```
fight10 = fight(classifier=improved_rfc_classifier, name_pokemon1='Mewtwo',␣
↪name_pokemon2='Arceus')
```

```
The winner is: Mewtwo with a probability of: 60.0%
```

2.5 Resultat del torneig



2.5.1 Mewtwo

Aquest *Pokemon* es un dels primers creats per la ciència i es la conseqüència d'una producció genèticament realçada de Mew, donant com a resultat un *Pokemon* molt intel · ligent, de fet molt més intel · ligent que els humans. L'objectiu de la seva creació és crear el *Pokemon* més fort del món.

Les seves habilitats psíquiques li permeten volar a través de levitació, comunicar-se telepàticament, bloquejar les habilitats especials d'altres *Pokemons*, hipnotitzar altres éssers, entre moltes altres.

3 Conclusions