

prog_datasci_8_vis_ejerciciosResueltos

December 28, 2020

1 Programación para *Data Science*

1.1 Unidad 8: Visualización de datos en Python - Ejercicios resueltos

En este Notebook hay un conjunto de ejercicios para practicar. Estos ejercicios no puntúan para la PEC, pero os recomendamos que los intentéis resolver como parte del proceso de aprendizaje. Encontraréis ejemplos de posibles soluciones a los ejercicios en el propio notebook, pero es importante que intentéis resolverlos vosotros antes de consultar las soluciones. Las soluciones os permitirán validar vuestras respuestas, así como ver alternativas de resolución de las actividades. También os animamos a preguntar cualquier duda que surja sobre la resolución de las actividades para practicar en el foro del aula.

1.2 Preguntas y ejercicios para practicar

1.2.1 Ejercicio 1

Pokemon es una saga RPG muy famosa a nivel mundial. La misión de estos juegos es capturar y entrenar a los pokemon, unas criaturas que habitan en todo el continente, para hacerse con el título de maestro de la liga Pokemon. Los Pokemons son criaturas muy diversas, hay muchos tipos diferentes y algunos de ellos pueden evolucionar.

En este ejercicio exploraremos los Pokemons de la primera generación con el dataset `pokemon.csv`.

1. ¿Cuántos Pokemons legendarios hay? ¿Cuántos Pokemons hay de cada tipo de evolución (*variable Stage*)?
2. ¿Cuál es el tipo más frecuente? Y el menos?
3. Un nuevo entrenador tiene que escoger entre 3 Pokemons iniciales (Bulbasaur, Charmander y Squirtle) y nos ha pedido que le ayudemos a decidir basándonos en las estadísticas de los pokemons de estos tres tipos (Grass, Fire y Water) . Si nos centramos sólo en las características de ataque y defensa, qué tipo deberíamos recomendar?

Representa las respuestas gráficamente.

Pista: En la pregunta 3, podéis utilizar la función `jointplot` que hemos visto en el Notebook de teoría. Considerad qué tipo de gráfica, de entre las que ofrece `jointplot`, se ajusta a los requerimientos del enunciado.

```
[1]: # Respuesta
```

1.2.2 Ejercicio 2

Cargad el conjunto de datos `pulitzer-circulation-data.csv` en un dataframe de Pandas. La fuente original de este conjunto de datos es el repositorio de datos de [FiveThirtyEight] (<https://github.com/fivethirtyeight/data>).

Cread un diagrama de dispersión que permita visualizar las muestras del conjunto de datos de pulitzer según las variables `Daily Circulation 2004` y `Daily Circulation 2013`. Incluid una recta con el ajuste lineal entre las dos variables. En cuanto a los detalles de visualización, limitad la visualización de ambos ejes en el intervalo (0, 1000000), utilice el estilo `whitegrid` de Seaborn, y dibuja los puntos y la línea en color verde.

Pista: podéis utilizar la función `jointplot` que hemos visto en el Notebook de explicación. Considerad qué tipo de gráfica, [de entre las que ofrece `jointplot`] (<http://seaborn.pydata.org/generated/seaborn.jointplot.html>), se ajusta a los requerimientos del enunciado.

```
[2]: # Respuesta
```

1.2.3 Ejercicio 3

Juego de Tronos es una serie de televisión basada en la famosa saga *Una Canción de Fuego y Hielo* de George RR Martin. Esta serie es conocida por sus complejos escenarios políticos y bélicos, así como las numerosas muertes de personajes.

En este ejercicio trabajamos con el dataset `battles.csv` que nos da información sobre las batallas que han tenido lugar a lo largo de *Juego de Tronos*.

Cread un diagrama de puntos (*scatter*) que permita comparar el tamaño (`attacker_size` vs `defender_size`) de los dos principales ejércitos involucrados en cada batalla. Utilizad la columna `attacker_outcome` para mostrar con diferentes colores el resultado de la batalla por el atacante.

Podéis utilizar la función [`lmplot`] (<https://seaborn.pydata.org/generated/seaborn.lmplot.html>) de Seaborn para generar la gráfica. Considerad qué atributos nos permiten crear este tipo de visualización.

Viendo la gráfica, podemos afirmar que el tamaño del ejército es determinante en el *outcome* de la batalla?

Nota: utilizad el estilo `whitegrid` de Seaborn.

```
[3]: # Respuesta
```

1.2.4 Ejercicio 4

En los reinos de Poniente, los Grandes Maestros consideran que todo el mundo debería tener acceso a una biblioteca. Desgraciadamente, después de una gran guerra la red de carreteras que comunican estos reinos ha quedado gravemente afectada. Los Grandes Maestros nos han contratado para resolver las siguientes cuestiones:

1. ¿Qué reinos han quedado sin poder acceder a una biblioteca?
2. ¿Cuál es el número mínimo de bibliotecas que se deberían construir para que todo el mundo vuelva a tener acceso a una biblioteca? (Si no podemos re-construir ninguna carretera)

3. ¿Cuál es el coste mínimo de re-construcción de carreteras a fin de que todo el mundo vuelva a tener acceso a una biblioteca? (Si no podemos construir ninguna biblioteca)
4. Viendo la respuesta de la pregunta 2 y 3, que sería lo más óptimo teniendo en cuenta que queremos minimizar los costes?

Para poder responder a estas preguntas, los Grandes Maestros nos facilitan la siguiente información:

- Disponemos de un grafo con el estado de la red de carreteras entre los reinos.
- Disponemos de un listado con las ciudades que disponen de una biblioteca.
- Una ciudad se considera que tiene acceso a una biblioteca si existe una biblioteca en la misma ciudad o si los habitantes pueden viajar por carretera hasta otra ciudad con biblioteca.
- El coste de construir una biblioteca es de 150.000 €.
- El coste de re-construir una carretera es de 80.000 €.

Nota: para resolver el ejercicio, **tenéis que generar una visualización del grafo que os permita responder a las cuatro preguntas planteadas**. Después, recuerda responder a las cuatro preguntas!

```
[4]: # Datos disponibles para el ejercicio

import networkx as nx

# Lista de carreteras transitables
carreteras_ok = [(1,3), (3,4), (2, 7), (2, 8), (5, 6), (9, 10)]

# Lista de carreteras cortadas
carreteras_ko = [(1, 2), (2, 4), (2, 5), (8, 9)]

# Graf G, que representa la red de carreteras entre las ciudades
G = nx.Graph()

G.add_edges_from(carreteras_ok)
G.add_edges_from(carreteras_ko)

# Lista de ciudades con biblioteca
ciudades_con_biblioteca = [3, 6, 9]
```

```
[5]: # Respuesta
```

1.2.5 Ejercicio 5

El año 2018 en la ciudad de Barcelona se han producido, de media, 27.22 accidentes con vehículos implicados. Es por esto que el ayuntamiento de Barcelona requiere de nuestro servicio para construir un mapa **interactivo** que muestre los puntos donde se han producido estos accidentes.

Nos piden crear un mapa utilizando la librería *geoplotlib* que nos permitirá ver la localización de cada accidente y el número de víctimas implicadas. De momento sólo nos piden visualizar a los accidentes producidos en el mes de **junio del año 2018** en **fin de semana** (sábados y domingos).

Nota 1: Podéis obtener los datos de los accidentes gestionados por la Guardia Urbana en la ciudad

de Barcelona en el portal [Open Data BCN](#) y [Cargar los datos](#) a partir de 1 diccionario o dataframe.

[6]: `# Respuesta`

1.2.6 Ejercicio 6

En este ejercicio utilizaremos el `dataset covid19_US_cases.csv`, que muestra la evolución de los casos de Covid-19 en los diferentes estados desde finales de enero de 2020.

- 1.Cuál es la distribución de casos en Estados Unidos el último mes (abril)? **Muestra gráficamente los resultados para cada grupo (activos, recuperados y muertos), pero sin separar por estados.**
2. Adicionalmente, se nos pide generar una gráfica que muestre la evolución de los casos confirmados en el estado de Nueva York, Massachusetts, Washington, California y Texas desde el inicio de la pandemia. Qué región es la más afectada?

Nota 1: En el apartado 1, para sumar todos los casos de US se puede utilizar la función `groupby` de pandas que hemos visto en lecciones anteriores.

Nota 2: La visualización de los datos debería ser atractiva y clara a fin de que se puedan sacar conclusiones de ellas.

1.2.7 Ejercicio 7

A partir de los datos de twitter se pueden crear redes de relaciones entre sus usuarios. En este ejercicio trabajaremos con el conjunto de datos `politics_twitter.csv`, que constituye una red de las relaciones entre diferentes políticos de España generada a partir de extracciones de twitter.

Cada fila representa un link entre 2 políticos (P1 y P2) y el valor (*Weight*) indica la fuerza de la conexión, en este caso 1 ya que sólo hace referencia a la presencia de esta relación.

1. Cuántas comunidades se pueden detectar en el grafo?
2. Representa gráficamente el grafo y pinta de diferente color cada comunidad detectada.

1.3 Soluciones a los ejercicios para practicar

1.3.1 Ejercicio 1

Pokemon es una saga RPG muy famosa a nivel mundial. La misión de estos juegos es capturar y entrenar a los pokemon, unas criaturas que habitan en todo el continente, para hacerse con el título de maestro de la liga Pokemon. Los Pokemons son criaturas muy diversas, hay muchos tipos diferentes y algunos de ellos pueden evolucionar.

En este ejercicio exploraremos los Pokemons de la primera generación con el dataset `pokemon.csv`.

1. ¿Cuántos Pokemons legendarios hay? ¿Cuántos Pokemons hay de cada tipo de evolución (*variable Stage*)?
2. ¿Cuál es el tipo más frecuente? Y el menos?

- Un nuevo entrenador tiene que escoger entre 3 Pokemons iniciales (Bulbasaur, Charmander y Squirtle) y nos ha pedido que le ayudemos a decidir basándonos en las estadísticas de los pokemons de estos tres tipos (Grass, Fire y Water) . Si nos centramos sólo en las características de ataque y defensa, qué tipo deberíamos recomendar?

Representa las respuestas gráficamente.

Pista: En la pregunta 3, podéis utilizar la función jointplot que hemos visto en el Notebook de teoría. Considerad qué tipo de gráfica, de entre las que ofrece jointplot, se ajusta a los requerimientos del enunciado.

```
[7]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Mostramos las gráficas en el notebook
%matplotlib inline

# Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set_style("whitegrid")

# Desactivamos unos warnings porque no nos aparezcan
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)

# Cargamos los datos del fichero pokemon.csv en un dataframe
pkmn = pd.read_csv('data/pokemon.csv')

# Visualizamos las primeras filas del dataset
pkmn.head()
```

```
[7]:
```

	Number	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	\
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	
2	3	Venusaur	Grass	Poison	525	80	82	83	100	
3	4	Charmander	Fire	NaN	309	39	52	43	60	
4	5	Charmeleon	Fire	NaN	405	58	64	58	80	

	Sp. Def	Speed	Stage	Legendary
0	65	45	1	False
1	80	60	2	False
2	100	80	3	False
3	50	65	1	False
4	65	80	2	False

```
[8]: ##1

# Cuántos Pokemons legendarios hay?

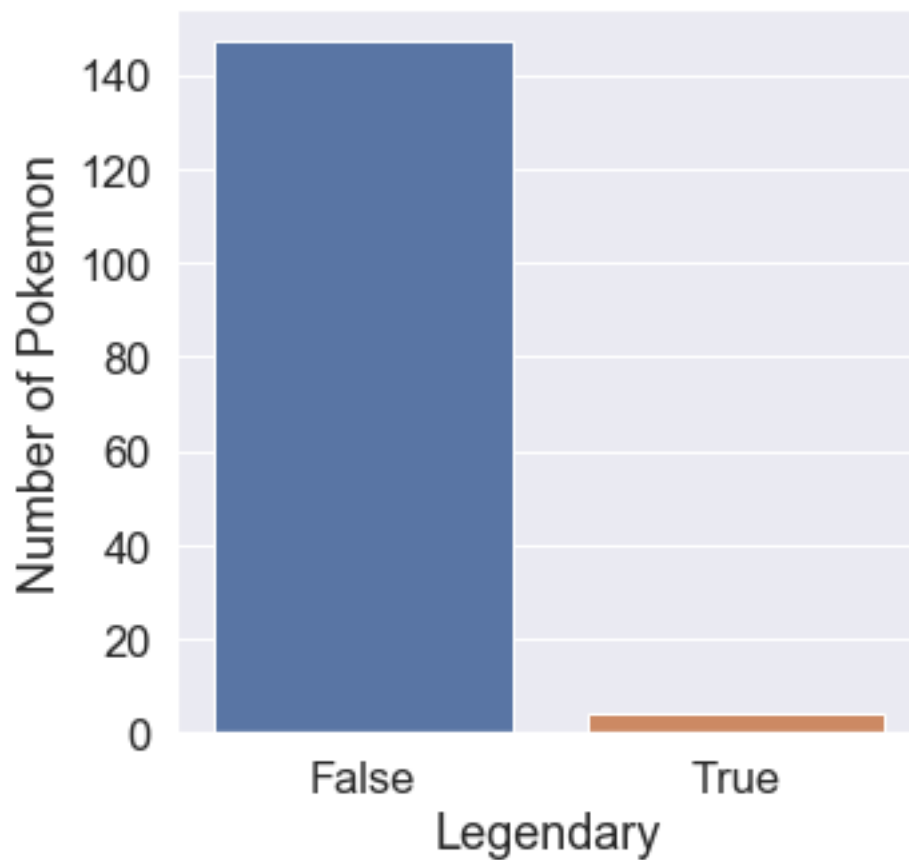
pkmn_no_leg=np.sum(pkmn['Legendary']==0)
pkmn_leg=np.sum(pkmn['Legendary']==1)

print(f"Número de Pokemons no legendarios: {pkmn_no_leg}")
print(f"Número de Pokemons legendarios: {pkmn_leg}")

#Graficamos la respuesta
plt.figure(figsize=(5,5))
sns.set(font_scale = 1.5)
legend_ = sns.countplot(x="Legendary", data=pkmn)
legend_ = plt.ylabel('Number of Pokemon')
```

Número de Pokemons no legendarios: 147

Número de Pokemons legendarios: 4



```
[9]: #Cuántos pokemons hay de cada evolución?

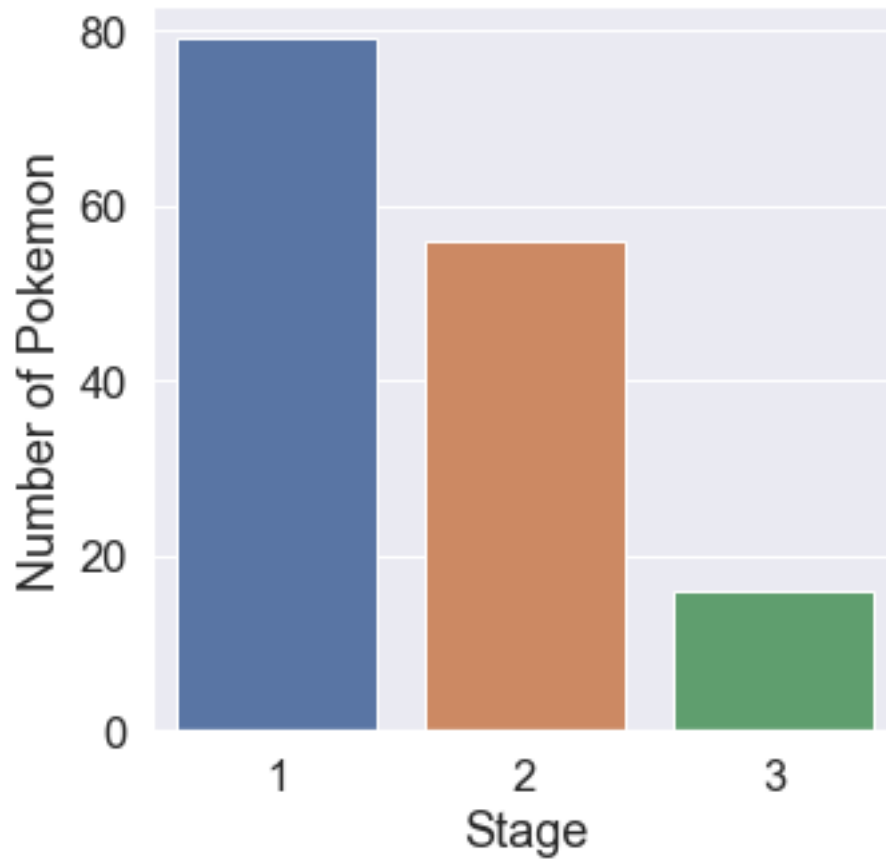
#OPCIÓN A
pkmn_S1=np.sum(pkmn['Stage']==1)
pkmn_S2=np.sum(pkmn['Stage']==2)
pkmn_S3=np.sum(pkmn['Stage']==3)

print(f"Número de Pokemons Stage 1: {pkmn_S1}")
print(f"Número de Pokemons Stage 2: {pkmn_S2}")
print(f"Número de Pokemons Stage 3: {pkmn_S3}")

#OPCIÓN B
pkmn_ = pkmn.groupby('Stage').count().Number

#Graficamos la respuesta
plt.figure(figsize=(5,5))
sns.set(font_scale = 1.5)
stage_ = sns.countplot(x="Stage", data=pkmn)
stage_ = plt.ylabel('Number of Pokemon')
```

```
Número de Pokemons Stage 1: 79
Número de Pokemons Stage 2: 56
Número de Pokemons Stage 3: 16
```



```
[10]: ##2

#Cuál es el tipo más frecuente? Y el menos?

#Mostramos el número de pokemons para cada tipo
print(pkmn.groupby('Type 1')['Number'].count())

print("\n")

#Mostramos cuál es el tipo más y menos frecuente

print(f"El tipo más frecuente es el de " + str(pkmn.groupby('Type 1')['Number'].
    ↳count().argmax()) +
      " con " + str(pkmn.groupby('Type 1')['Number'].count().max()) + "
    ↳pokemons")
print(f"El tipo menos frecuente es el de " + str(pkmn.groupby('Type 1')['Number'].count().idxmin()) +
      " con " + str(pkmn.groupby('Type 1')['Number'].count().min()) + "
    ↳pokemons")
```



```

#Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set_style("whitegrid")

#Graficamos la respuesta
plt.figure(figsize=(10,7))
sns.set(font_scale = 1.5)
type_ = sns.countplot(x="Type 1", data=pkmn)
type_ = plt.xticks(rotation=45)
type_ = plt.ylabel('Number of Pokemon')

```

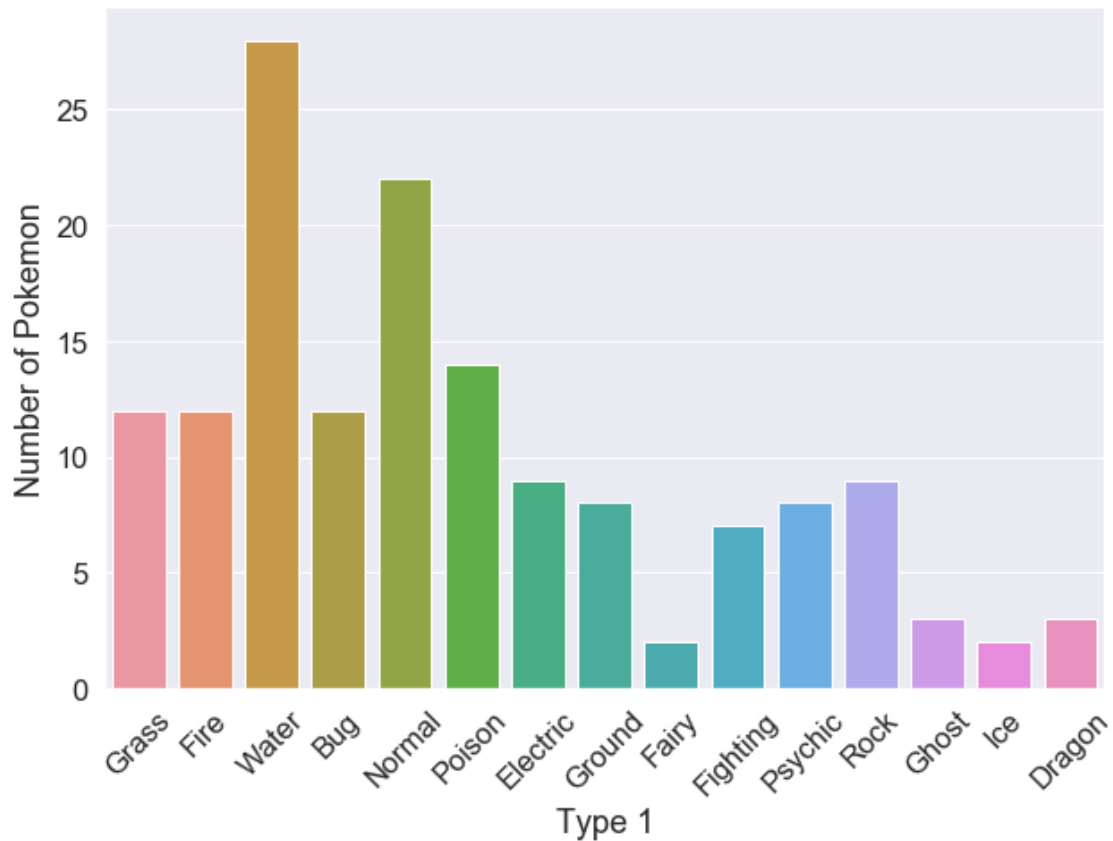
```

Type 1
Bug      12
Dragon   3
Electric 9
Fairy     2
Fighting 7
Fire     12
Ghost     3
Grass     12
Ground    8
Ice       2
Normal   22
Poison    14
Psychic   8
Rock      9
Water    28
Name: Number, dtype: int64

```

El tipo más frecuente es el de 14 con 28 pokemons

El tipo menos frecuente es el de Fairy con 2 pokemons



```
[11]: ##3

#Que pokemon inicial debemos escoger?

#Seleccionamos los pokemons de tipo agua, fuego y planta
pkmn_iniciales_tipos = pkmn[pkmn["Type 1"].isin(["Grass", "Fire", "Water"])]

#Seleccionamos las características que queremos evaluar
pkmn_iniciales_tipos_stats=pkmn_iniciales_tipos[["Type_
↳1", "HP", "Attack", "Defense"]]

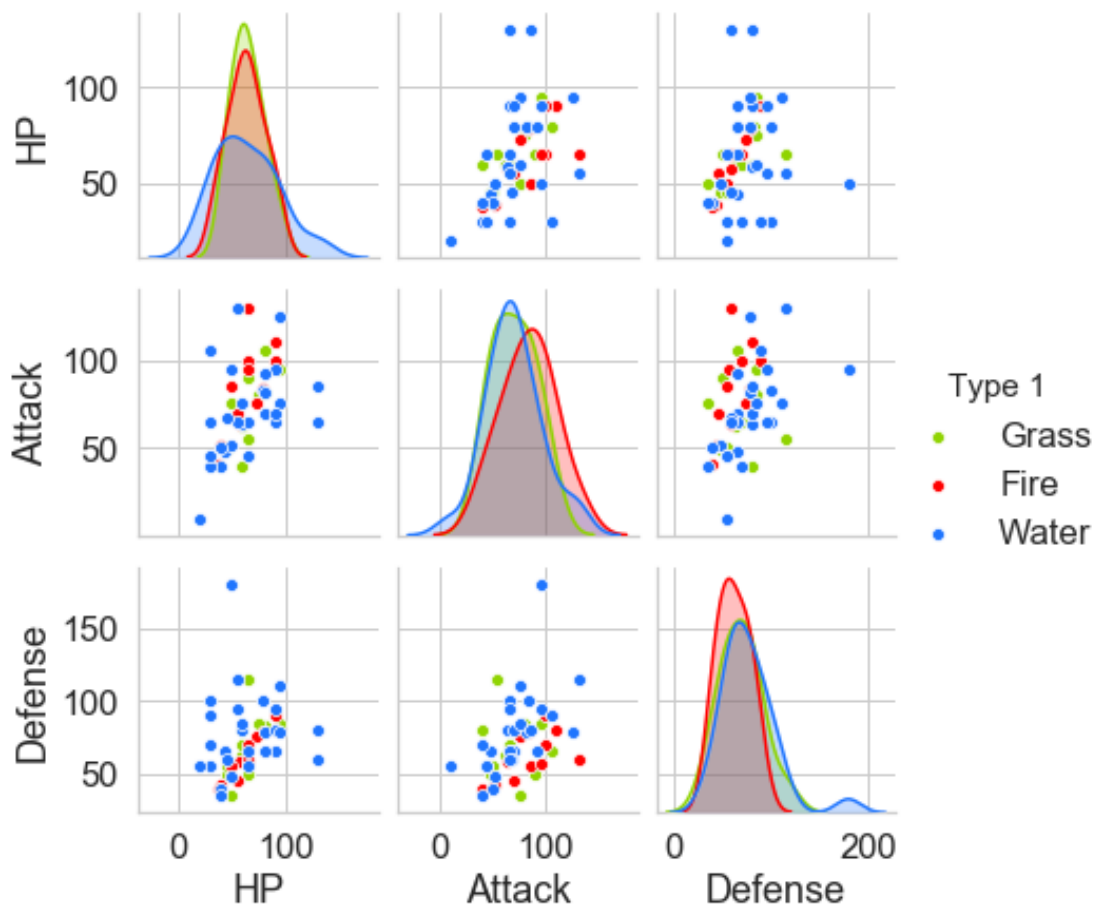
#Especificamos unos colores concretos para cada tipo de pokemon en los plots_
↳(verde = planta, azul = agua, rojo = fuego)
colors_pkm=["#8fd400", "#ff0000", "#1f75fe"]
```

```
[12]: #OPCIÓN A: pairplot

#Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set_style("whitegrid")
```

```
#Graficamos la respuesta
sns.pairplot(pkmn_iniciales_tipos_stats, hue="Type 1",height=2,
→palette=colors_pkm)
```

[12]: <seaborn.axisgrid.PairGrid at 0x1a25b383d0>

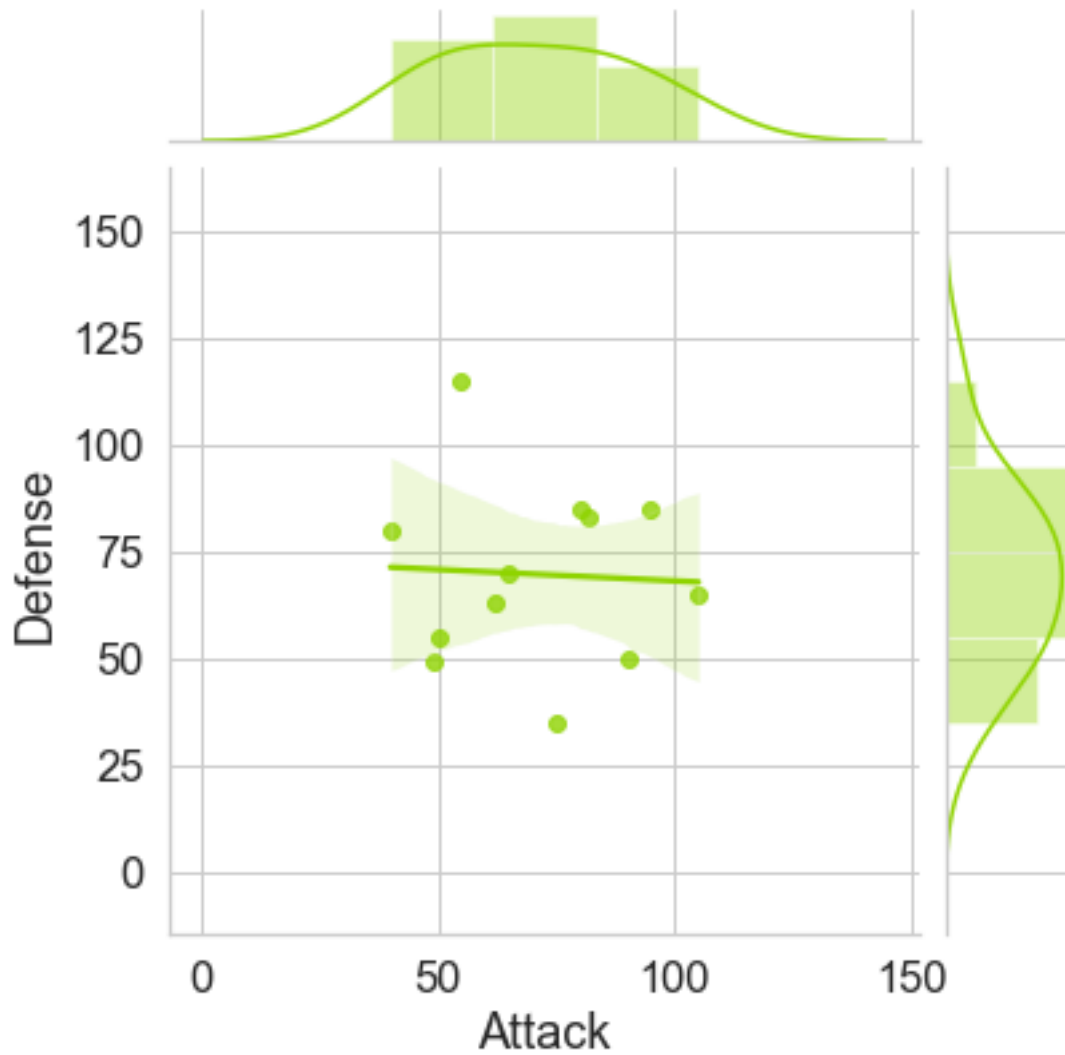


[13]: #OPCIÓN B: jointplot

```
#Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set_style("whitegrid")

# Joint Distribution Plot
sns.jointplot(x='Attack', y='Defense',
→data=pkmn_iniciales_tipos_stats[pkmn_iniciales_tipos_stats['Type_
→1']=='Grass'],
→color="#8fd400",kind='reg')
```

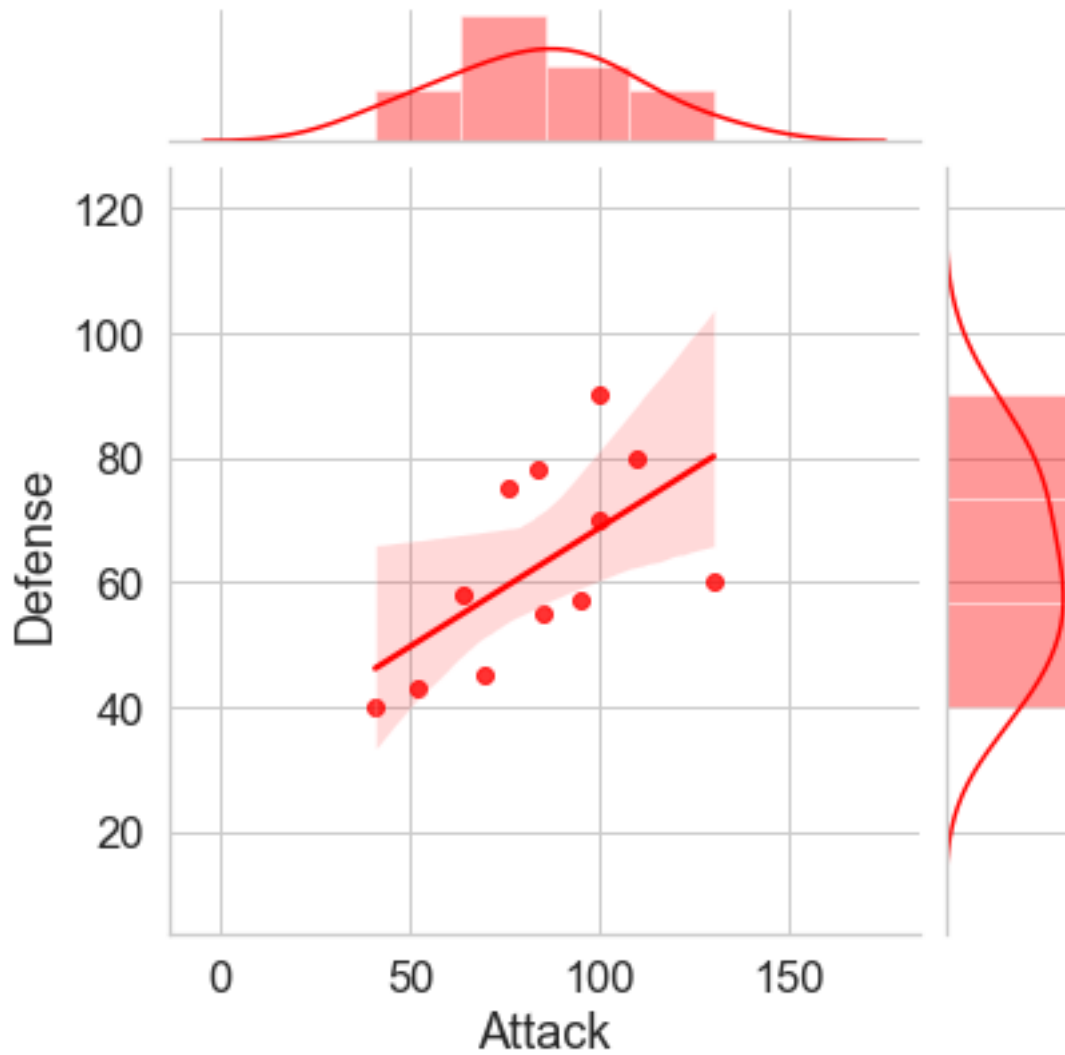
[13]: <seaborn.axisgrid.JointGrid at 0x1a2474c990>



```
[14]: #Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set_style("whitegrid")

sns.jointplot(x='Attack', y='Defense',
             data=pkmn_iniciales_tipos_stats[pkmn_iniciales_tipos_stats['Type_1']=='Fire'],
             color="#ff0000", kind='reg')
```

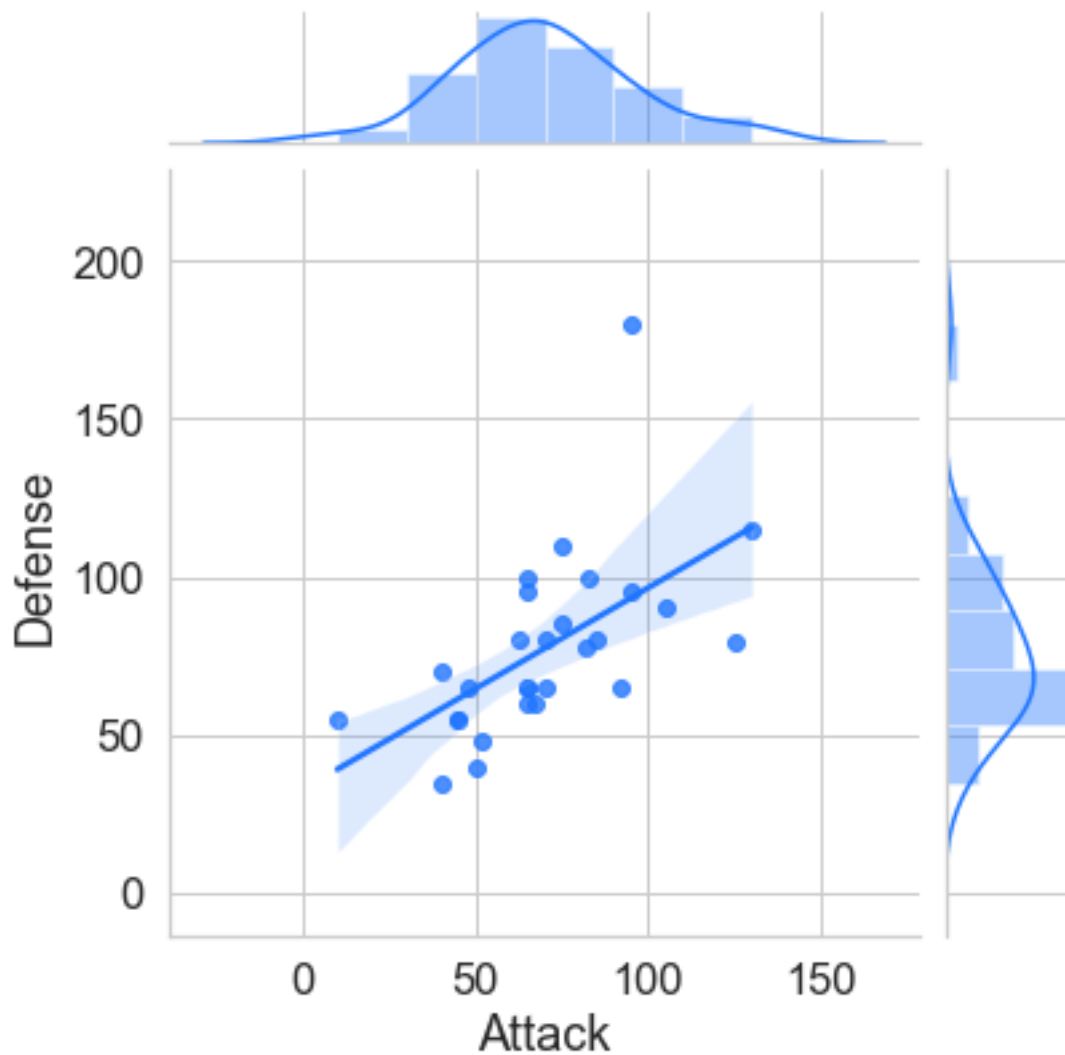
```
[14]: <seaborn.axisgrid.JointGrid at 0x1a264b1990>
```



```
[15]: #Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set_style("whitegrid")

sns.jointplot(x='Attack', y='Defense',
             data=pkmn_iniciales_tipos_stats[pkmn_iniciales_tipos_stats['Type_1']=='Water'],
             color="#1f75fe", kind='reg')
```

```
[15]: <seaborn.axisgrid.JointGrid at 0x11096f850>
```



Respuesta

No hay una sola solución correcta en este ejercicio ;) Una buena respuesta sería recomendar un pokemon inicial de agua o fuego, ya que tienen una correlación positiva ente ataque y defensa, por tanto, los pokemons evolucionados con ataque alto tendrán también una defensa alta.

1.3.2 Ejercicio 2

Cargad el conjunto de datos `pulitzer-circulation-data.csv` en un dataframe de Pandas. La fuente original de este conjunto de datos es el repositorio de datos de [FiveThirtyEight] (<https://github.com/fivethirtyeight/data>).

Cread un diagrama de dispersión que permita visualizar las muestras del conjunto de datos de Pulitzer según las variables `Daily Circulation 2004` y `Daily Circulation 2013`. Incluid una recta con el ajuste lineal entre las dos variables. En cuanto a los detalles de visualización, limitad

la visualización de ambos ejes en el intervalo (0, 1000000), utilice el estilo `whitegrid` de Seaborn, y dibuja los puntos y la línea en color verde.

Pista: podéis utilizar la función `jointplot` que hemos visto en el Notebook de explicación. Considerad qué tipo de gráfica, [de entre las que ofrece `jointplot`] (<http://seaborn.pydata.org/generated/seaborn.jointplot.html>), se ajusta a los requerimientos del enunciado.

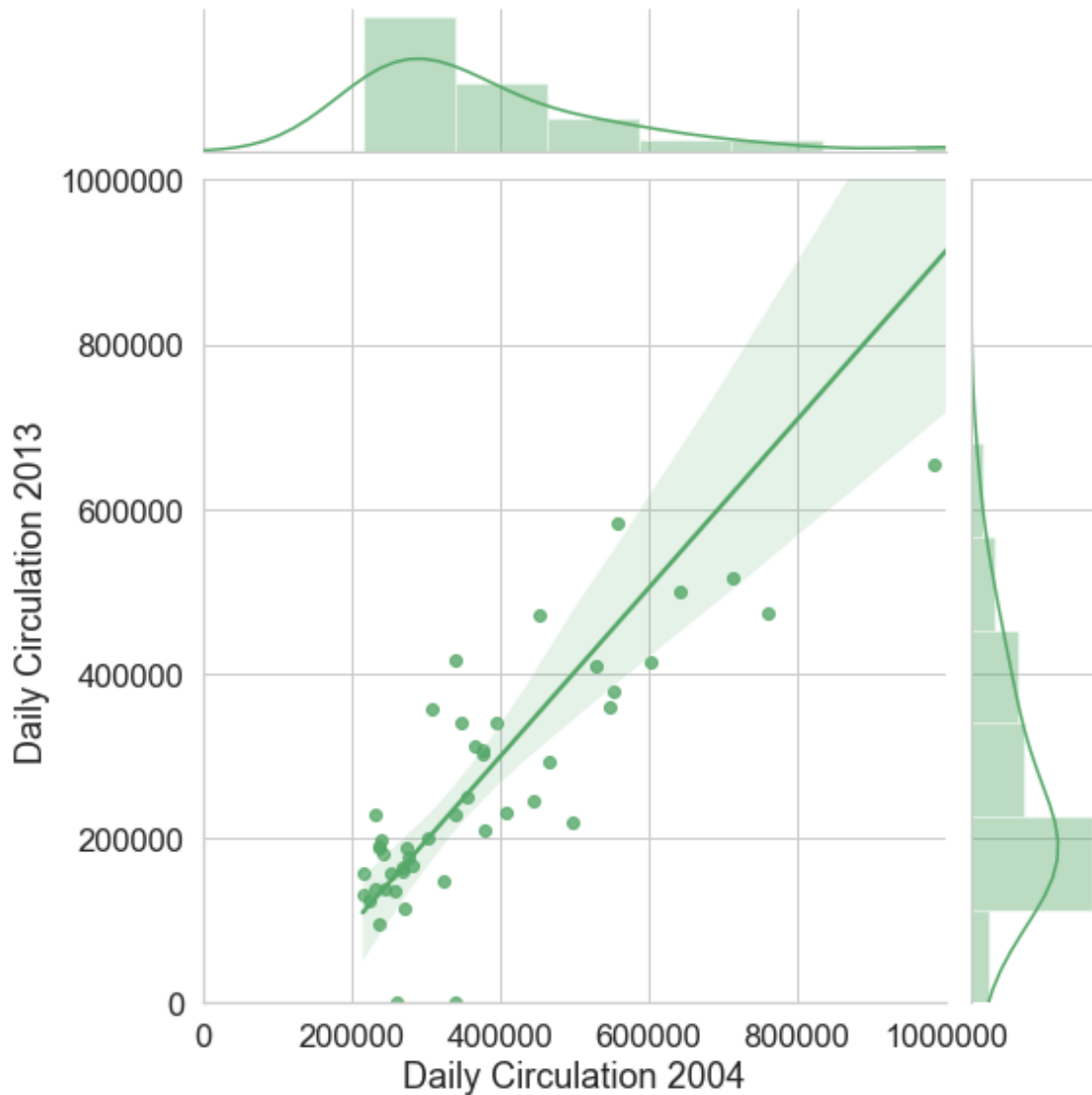
```
[16]: # Importamos las librerías
import pandas as pd
import seaborn as sns

# Mostramos las gráficas en el notebook
%matplotlib inline

# Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set_style("whitegrid")

# Cargamos los datos del fichero Pulitzer-circulation-data.csv en un dataframe
data = pd.read_csv('data/pulitzer-circulation-data.csv')

# Generamos la gráfica de tipo "reg"
g = sns.jointplot("Daily Circulation 2004", "Daily Circulation 2013", data=data,
                  xlim=[0, 1000000], ylim=[0, 1000000],
                  kind="reg",
                  color="g", height=8)
```



1.3.3 Ejercicio 3

Juego de Tronos es una serie de televisión basada en la famosa saga *Una Canción de Fuego y Hielo* de George RR Martin. Esta serie es conocida por sus complejos escenarios políticos y bélicos, así como las numerosas muertes de personajes.

En este ejercicio trabajamos con el dataset `battles.csv` que nos da información sobre las batallas que han tenido lugar a lo largo de *Juego de Tronos*.

Cread un diagrama de puntos (*scatter*) que permita comparar el tamaño (`attacker_size` vs `defender_size`) de los dos principales ejércitos involucrados en cada batalla. Utilizad la columna `attacker_outcome` para mostrar con diferentes colores el resultado de la batalla por el atacante.

Podéis utilizar la función `[lmploot]` (<https://seaborn.pydata.org/generated/seaborn.lmploot.html>) de Seaborn para generar la gráfica. Considerad qué atributos nos permiten crear este tipo de

visualización.

Viendo la gráfica, podemos afirmar que el tamaño del ejército es determinante en el *outcome* de la batalla?

Nota: utilizad el estilo `whitegrid` de Seaborn.

```
[17]: # Importamos las librerías
import pandas as pd
import seaborn as sns

# Mostramos las gráficas en el notebook
%matplotlib inline

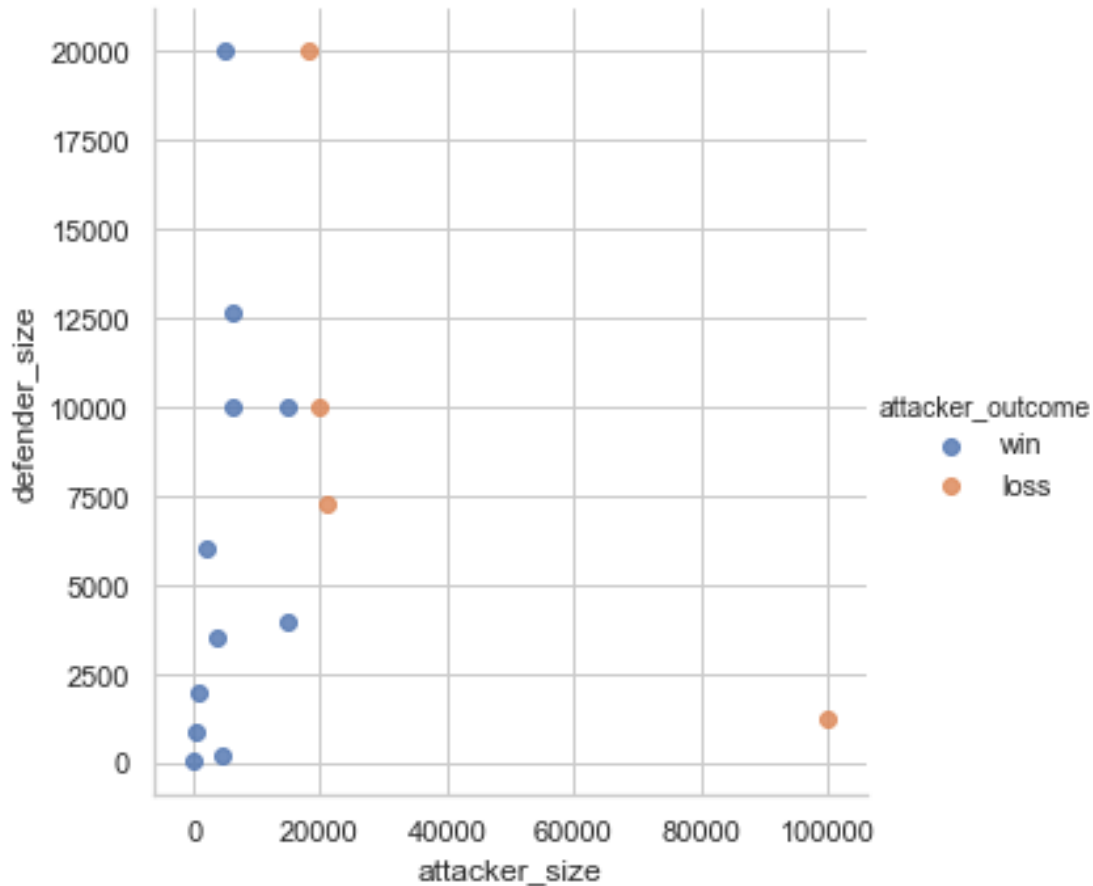
# Indicamos que queremos utilizar el estilo "whitegrid" de Seaborn
sns.set(style="whitegrid")

# Cargamos los datos del fichero got.csv en un dataframe
got_data = pd.read_csv('data/battles.csv')

# Eliminamos las muestras con valores NaN en las columnas attacker_size o
↳ defender_size
got_data = got_data.dropna(subset=["attacker_size", "defender_size"])

# Mostramos la gráfica
sns.lmplot(x="attacker_size", y="defender_size", hue="attacker_outcome",
↳ scatter=True, fit_reg=False, data=got_data)
```

```
[17]: <seaborn.axisgrid.FacetGrid at 0x1a26c01790>
```



Resposta

Veient la gràfica anterior, no podem afirmar que la mida de l'exèrcit és determinant en l'outcome de la batalla. Com podem observar, hi ha casos on l'atacant perd la batalla tot i la superioritat numèrica de les tropes i a l'inrevés, l'atacant guanya la batalla tot i la superioritat numèrica del bàndol contrari.

1.3.4 Ejercicio 4

En los reinos de Poniente, los Grandes Maestros consideran que todo el mundo debería tener acceso a una biblioteca. Desgraciadamente, después de una gran guerra la red de carreteras que comunican estos reinos ha quedado gravemente afectada. Los Grandes Maestros nos han contratado para resolver las siguientes cuestiones:

1. ¿Qué reinos han quedado sin poder acceder a una biblioteca?
2. ¿Cuál es el número mínimo de bibliotecas que se deberían construir para que todo el mundo vuelva a tener acceso a una biblioteca? (Si no podemos re-construir ninguna carretera)
3. ¿Cuál es el coste mínimo de re-construcción de carreteras a fin de que todo el mundo vuelva a tener acceso a una biblioteca? (Si no podemos construir ninguna biblioteca)
4. Viendo la respuesta de la pregunta 2 y 3, que sería lo más óptimo teniendo en cuenta que queremos minimizar los costes?

Para poder responder a estas preguntas, los Grandes Maestros nos facilitan la siguiente información:

- Disponemos de un grafo con el estado de la red de carreteras entre los reinos.
- Disponemos de un listado con las ciudades que disponen de una biblioteca.
- Una ciudad se considera que tiene acceso a una biblioteca si existe una biblioteca en la misma ciudad o si los habitantes pueden viajar por carretera hasta otra ciudad con biblioteca.
- El coste de construir una biblioteca es de 150.000 €.
- El coste de re-construir una carretera es de 80.000 €.

Nota: para resolver el ejercicio, **tenéis que generar una visualización del grafo que os permita responder a las cuatro preguntas planteadas**. Después, recuerda responder a las cuatro preguntas!

```
[18]: # Datos disponibles para el ejercicio

import networkx as nx

# Lista de carreteras transitables
carreteras_ok = [(1,3), (3,4), (2, 7), (2, 8), (5, 6), (9, 10)]

# Lista de carreteras cortadas
carreteras_ko = [(1, 2), (2, 4), (2, 5), (8, 9)]

# Graf G, que representa la red de carreteras entre las ciudades
G = nx.Graph()

G.add_edges_from(carreteras_ok)
G.add_edges_from(carreteras_ko)

# Lista de ciudades con biblioteca
ciudades_con_biblioteca = [3, 6, 9]
```

```
[19]: %matplotlib inline

# Calculamos la posición de los nodos usando el algoritmo spring
graph_pos = nx.spring_layout(G)

# Mostramos los nodos del grafo. Mostramos con un tamaño más grande y en azul
↳ las ciudades que tienen biblioteca
nx.draw_networkx_nodes(G, graph_pos, ciudades_con_biblioteca, node_size=700,
↳ node_color='blue', alpha = 0.6)

# Mostramos con un tamaño más pequeño y en naranja los otros nodos del grafo
ciudades_sin_biblioteca = list(set(G.nodes()) - set(ciudades_con_biblioteca))
nx.draw_networkx_nodes(G, graph_pos, ciudades_sin_biblioteca, node_size=300,
↳ node_color='orange', alpha = 0.6)

# Mostramos las etiquetas del nodos, especificando el tamaño y la fuente
```

```

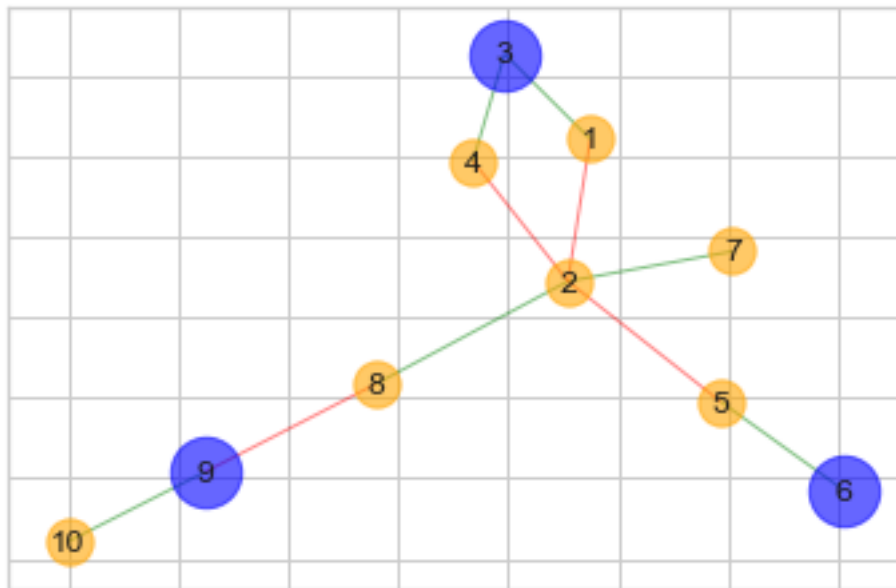
nx.draw_networkx_labels(G, graph_pos, font_size=12, font_family='sans-serif')

# Mostramos las aristas del grafo
# Primero, mostramos con color verde las aristas que corresponden a las
→ carreteras que aún son transitables
nx.draw_networkx_edges(G, graph_pos, edgelist=carreteras_ok,
→ edge_color='green', alpha=0.5)

# Mostramos con color rojo las aristas que corresponden a las carreteras
→ cortadas
nx.draw_networkx_edges(G, graph_pos, edgelist=carreteras_ko, edge_color='red',
→ alpha=0.5)

```

[19]: <matplotlib.collections.LineCollection at 0x1a27182250>



Respuesta

Una vez construido el grafo, podemos contestar las preguntas que nos han hecho los Grandes Maestros.

- Qué reinos han quedado sin poder acceder a una biblioteca?

Los reinos que han quedado sin acceso a una biblioteca son: [2, 7, 8].

- Cuál es el número mínimo de bibliotecas que se deberían construir para que todo el mundo vuelva a tener acceso a una biblioteca? (Si no podemos re-construir ninguna carretera)

Dado que los tres reinos que han quedado sin acceso a una biblioteca están conectados entre sí (aunque quedan carreteras que los unen), podemos construir una biblioteca en cualquiera de los

tres para que los ciudadanos de estos reinos puedan volver a disfrutar de una biblioteca. Por tanto, el número mínimo de bibliotecas sería **1**.

- Cuál es el coste mínimo de re-construcción de carreteras a fin de que todo el mundo vuelva a tener acceso a una biblioteca? (Si no podemos construir ninguna biblioteca)

Siguiendo el mismo razonamiento anterior, si re-construimos la carretera que conectaba el reino 8 con el reino 9 (que tiene biblioteca), también estaremos habilitando el acceso a los reinos 2 y 7. Por tanto, el coste mínimo para re-construir carreteras sería **150.000 €**.

- Viendo la respuesta de la pregunta 2 y 3, que sería lo más óptimo teniendo en cuenta que queremos minimizar los costes?

En este caso, lo más óptimo sería re-construir una carretera (por ejemplo la que une los reinos 8 y 9). Esto costaría 80.000 €, mientras que construir una biblioteca nos implica un coste de 150.000 €.

1.3.5 Ejercicio 5

El año 2018 en la ciudad de Barcelona se han producido, de media, 27.22 accidentes con vehículos implicados. Es por esto que el ayuntamiento de Barcelona requiere de nuestro servicio para construir un mapa **interactivo** que muestre los puntos donde se han producido estos accidentes.

Nos piden crear un mapa utilizando la librería *geoplotlib* que nos permitirá ver la localización de cada accidente y el número de víctimas implicadas. De momento sólo nos piden visualizar a los accidentes producidos en el mes de **junio del año 2018** en **fin de semana** (sábados y domingos).

Nota 1: Podéis obtener los datos de los accidentes gestionados por la Guardia Urbana en la ciudad de Barcelona en el portal [Open Data BCN](#) y [Cargar los datos](#) a partir de 1 diccionario o dataframe.

```
[20]: # Primero, nos descargamos el archivo csv con los datos del portal de Open Data
      ↪BCN
      # (https://opendata-ajuntament.barcelona.cat/data/ca/dataset/accidents-gu-bcn)
      # Y lo guardamos en la carpeta fecha para poder cargarlo a posteriori
      import pandas as pd
      import geoplotlib

      # Cargamos los datos en un dataframe
      accidentes_data = pd.read_csv('data/2018_accidents_gu_bcn.csv')

      # Inspeccionamos los datos que hemos cargado
      accidentes_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9936 entries, 0 to 9935
Data columns (total 27 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Numero_expedient      9936 non-null   object
1   Codi_districte        9936 non-null   int64
2   Nom_districte         9936 non-null   object
```

3	Codi_barri	9936	non-null	int64
4	Nom_barri	9936	non-null	object
5	Codi_carrer	9936	non-null	int64
6	Nom_carrer	9935	non-null	object
7	Num_postal	5138	non-null	object
8	Descripcio_dia_setmana	9936	non-null	object
9	Dia_setmana	9936	non-null	object
10	Descripcio_tipus_dia	9936	non-null	object
11	Any	9936	non-null	int64
12	Mes_any	9936	non-null	int64
13	Nom_mes	9936	non-null	object
14	Dia_mes	9936	non-null	int64
15	Hora_dia	9936	non-null	int64
16	Descripcio_torn	9936	non-null	object
17	Descripcio_causa_vianant	9936	non-null	object
18	Numero_morts	9936	non-null	int64
19	Numero_lesionats_lleus	9936	non-null	int64
20	Numero_lesionats_greus	9936	non-null	int64
21	Numero_victimes	9936	non-null	int64
22	Numero_vehicles_implicats	9936	non-null	int64
23	Coordenada_UTM_X	9936	non-null	float64
24	Coordenada_UTM_Y	9936	non-null	float64
25	Longitud	9936	non-null	float64
26	Latitud	9936	non-null	float64

dtypes: float64(4), int64(12), object(11)
memory usage: 2.0+ MB

```
[21]: # Inspeccionamos los valores de la columna Mes_any y Dia_setmana
# Ya que nos piden filtrar el mes de junio y fines de semana
print(accidentes_data['Mes_any'].unique())
print(accidentes_data['Dia_setmana'].unique())
```

```
[ 1  7  2  9  3 10 12  5  4  6 11  8]
['Dl' 'Dg' 'Dc' 'Dv' 'Dj' 'Ds' 'Dm']
```

```
[22]: # Geoplotlib necesita dos columnas con el nombre lat y lon para poder dibujar
# los puntos sobre el mapa.
# Renombramos las columnas Longitud y Latitud
accidentes_data.rename(index=str, columns={'Longitud': 'lon', 'Latitud': 'lat'}, inplace=True)
```

```
[23]: # Filtramos por los datos correspondientes al mes de junio 2018 y fines de
# semana
accidentes_junio = accidentes_data.loc[(accidentes_data['Mes_any'] == 6)
& (accidentes_data['Dia_setmana'].isin(['Ds', 'Dg']))]
```

```
# Cargamos con geoplotlib los datos del dataframe
gdata = geoplotlib.utils.DataAccessObject.from_dataframe(accidentes_junio)

# Mostramos los puntos del archivo
geoplotlib.dot(gdata, color='b', point_size=1.75, f_tooltip=lambda x :
↳str(x['Numero_victimes']))

# Mostramos el mapa
geoplotlib.show()
```

1.3.6 Ejercicio 6

En este ejercicio utilizaremos el *dataset covid19_US_cases.csv*, que muestra la evolución de los casos de Covid-19 en los diferentes estados desde finales de enero de 2020.

1. Cuál es la distribución de casos en Estados Unidos el último mes (abril)? **Muestra gráficamente los resultados para cada grupo (activos, recuperados y muertos), pero sin separar por estados.**
2. Adicionalmente, se nos pide generar una gráfica que muestre la evolución de los casos confirmados en el estado de Nueva York, Massachusetts, Washington, California y Texas desde el inicio de la pandemia. Qué región es la más afectada?

Nota 1: En el apartado 1, para sumar todos los casos de US se puede utilizar la función `groupby` de pandas que hemos visto en lecciones anteriores.

Nota 2: La visualización de los datos debería ser atractiva y clara a fin de que se puedan sacar conclusiones de ellas.

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Mostramos las gráficas en el notebook
%matplotlib inline

# Indicamos que queremos utilizar el estilo "whitegrid" de seaborn
sns.set_style("whitegrid")

# Cargamos los datos del fichero covid19_US.csv en un dataframe
covid_US_cases = pd.read_csv('data/covid19_US_cases.csv')

# Visualizamos las primeras filas
covid_US_cases.head()
```

```
[1]:  ObservationDate  Province/State  Country/Region  Confirmed  Deaths  Recovered  \
0      03/13/2020      Alabama                US           5         0           0
1      03/14/2020      Alabama                US           6         0           0
```

2	03/15/2020	Alabama	US	12	0	0
3	03/16/2020	Alabama	US	29	0	0
4	03/17/2020	Alabama	US	39	0	0

	Active
0	5
1	6
2	12
3	29
4	39

```
[2]: ##1

# Cuál es la distribución de casos en Estados Unidos?

# Agrupamos los datos por fecha y sumamos los casos.

covid_US_all_cases=covid_US_cases.groupby('ObservationDate').sum()

# Creamos dos columnas nuevas con la información del mes y del día de la
↳ observación

covid_US_all_cases['Month']=pd.DatetimeIndex(covid_US_all_cases.index).month
covid_US_all_cases['Day']=pd.DatetimeIndex(covid_US_all_cases.index).day

# Filtramos el dataset para coger sólo las observaciones de abril

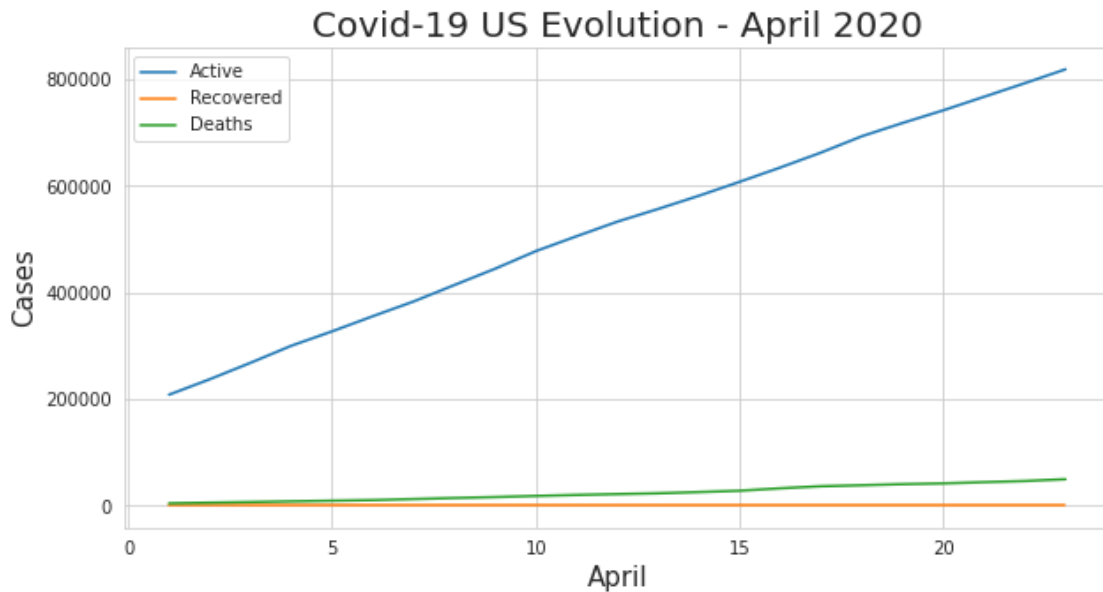
covid_US_all_cases_april=covid_US_all_cases[covid_US_all_cases['Month']==4]
```

```
[3]: # Graficamos el resultado

plt.figure(figsize = (10,5))

ax = sns.lineplot(x='Day', y="Active", data=covid_US_all_cases_april)
ax = sns.lineplot(x='Day', y="Recovered", data=covid_US_all_cases_april)
ax = sns.lineplot(x='Day', y="Deaths", data=covid_US_all_cases_april)

plt.legend(loc='upper left', labels=['Active', 'Recovered', 'Deaths'])
plt.title("Covid-19 US Evolution - April 2020", fontsize = 20)
plt.xlabel("April", fontsize = 15)
plt.ylabel("Cases", fontsize = 15)
plt.show()
```

[4]: ##2

```
# Evolución de los casos confirmados en el estado de Nueva York, Massachusetts,
↳ Washington, California y Texas

# Hacemos una lista con los estados que queremos estudiar
list_subset=['New York','Massachusetts','Washington','California','Texas']

# Creamos el subset con estos estados
covid_US_cases_subset=covid_US_cases[covid_US_cases['Province/State'].
↳ isin(list_subset)]

# Especificamos el tamaño y el formato del gráfico
plt.figure(figsize = (14,7))
sns.set_style("whitegrid", {'axes.grid' : False})

# Hacemos el gráfico indicando que los resultados se mostrarán en función del
↳ estado
sns.lineplot(x = "ObservationDate", y = "Confirmed", data =
↳ covid_US_cases_subset, hue = "Province/State",
style = "Province/State", dashes = False,
markers = ["o", "<", ">", "^", "*"], legend="brief")

# Inclínamos las fechas para que se puedan leer bien
plt.xticks(
```

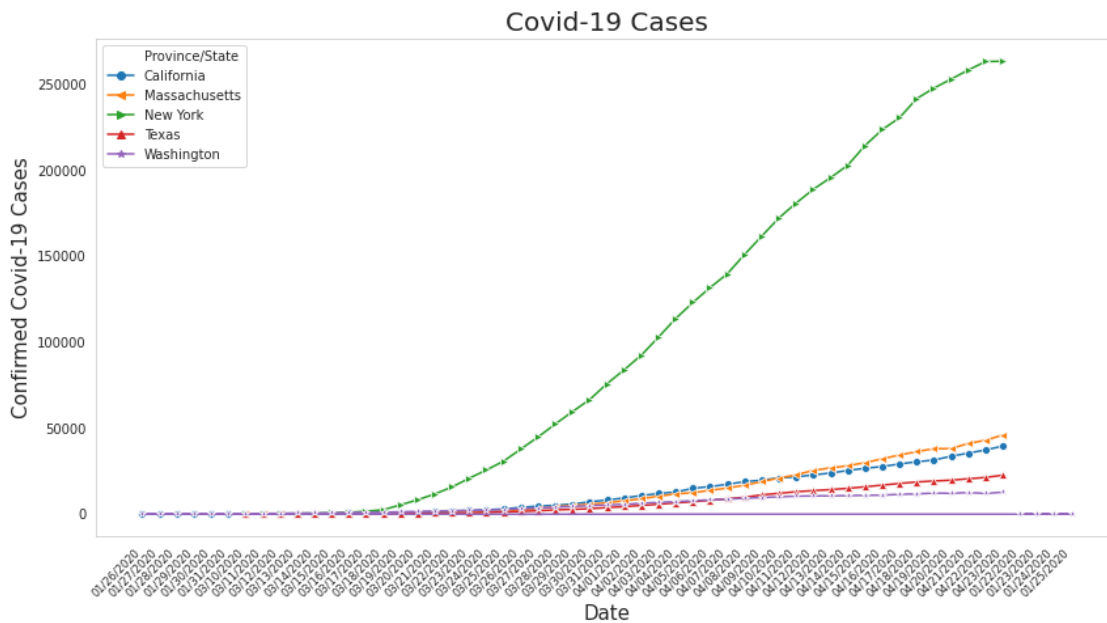
```

rotation=45,
horizontalalignment='right',
fontweight='light',
fontsize='small'
)

plt.title("Covid-19 Cases", fontsize = 20)
plt.xlabel("Date", fontsize = 15)
plt.ylabel("Confirmed Covid-19 Cases", fontsize = 15)

plt.show()

```



Respuesta

Con este gráfico se puede ver claramente que el estado más afectado por Covid-19 es Nueva York.

1.3.7 Ejercicio 7

A partir de los datos de twitter se pueden crear redes de relaciones entre sus usuarios. En este ejercicio trabajaremos con el conjunto de datos `politics_twitter.csv`, que constituye una red de las relaciones entre diferentes políticos de España generada a partir de extracciones de twitter.

Cada fila representa un link entre 2 políticos (P1 y P2) y el valor (*Weight*) indica la fuerza de la conexión, en este caso 1 ya que sólo hace referencia a la presencia de esta relación.

1. Cuántas comunidades se pueden detectar en el grafo?
2. Representa gráficamente el grafo y pinta de diferente color cada comunidad detectada.

```
[5]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx

# Cargamos los datos del fichero book1.csv en un dataframe
twitter = pd.read_csv('data/politics_twitter.csv')

# Visualizamos las primeras filas del dataset
twitter.head()
```

```
[5]:
```

	P1	P2	Weight
0	KRLS	IreneMontero	1
1	LauraBorras	IreneMontero	1
2	gabrielrufian	IreneMontero	1
3	AdaColau	IreneMontero	1
4	Jaumeasens	IreneMontero	1

```
[6]: # Datos disponibles para el ejercicio

# Creamos un objeto grafo vacío
G_twitter= nx.Graph()

# Iteramos para todas las filas del dataset
for _, edge in twitter.iterrows():
    G_twitter.add_edge(edge['P1'], edge['P2'], weight=edge['Weight'])
```

```
[7]: ## 1

# Cuántas comunidades pueden detectar en el grafo?

# Importamos la librería de detección de comunidades
from community import community_louvain

# Detectamos las comunidades existentes en el grafo
# Hay 3
partition = community_louvain.best_partition(G_twitter)
```

```
[8]: ## 2

# Representa gráficamente el grafo y pinta de diferente color cada comunidad
↳ detectada

# Importamos la librería de detección de comunidades
from community import community_louvain

# Generamos una nueva figura
```

```

plt.figure(1, figsize=(12, 10))

# Calculamos las posiciones de los nodos del grafo en el plano con el algoritmo
↳spring
graph_pos = nx.spring_layout(G_twitter)

# Definimos los colores que utilizaremos para pintar las 3 comunidades
colors = ['green', 'blue', 'red']

# Para cada comunidad detectada, mostramos los nodos
for count, com in enumerate(set(partition.values())):

    # Seleccionamos los nodos de la comunidad que estamos procesando
    list_nodes = [nodes for nodes in partition.keys()
                   if partition[nodes] == com]

    # Mostramos los nodos, coloreados según la comunidad a la que pertenecen
    nx.draw_networkx_nodes(G_twitter, graph_pos, list_nodes,
                           node_size = 800,
                           node_color = colors[count],
                           alpha = 0.5)

# Mostramos las aristas del grafo, especificando la posición,
# el color y la transparencia
nx.draw_networkx_edges(G_twitter, graph_pos, edge_color='k', alpha=0.3)

# Mostramos las etiquetas, indicando la fuente y el tamaño
a = nx.draw_networkx_labels(G_twitter, graph_pos, font_size=12,
↳font_family='sans-serif')

```

