Programación para Data Science

Unidad 8: Visualización de datos en Python

En este Notebook encontraréis dos conjuntos de ejercicios: un primer conjunto de ejercicios para practicar y que no puntuan, pero que recomendamos intentar resolver y un segundo conjunto que evaluaremos como PEC.

Ejercicios para practicar

Los siguientes 3 ejercicios no puntuan para la PEC, pero os recomendamos que los intentéis resolver antes de pasar a los ejercicios propios de la PEC. También podéis encontrar las soluciones a estos ejercicios al final del Notebook.

Ejercicio 1

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).

Generad 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 ambas variables. Por lo que respecta a los detalles de visualización, limitad la visualización de ambos ejes al intervalo (0, 1000000), usad el estilo `whitegrid` de seaborn, y dibujad los puntos y la línea en color verde.

Pista: Podéis usar la función `jointplot` que hemos visto en el notebook de explicación. Pensad en 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.

In [ ]:
```
# Respuesta
```

Ejercicio 2

Generad una gráfica que muestre el número de votos y el número de escaños obtenidos de los 10 partidos más votados en las elecciones generales en España de 2016 para el congreso. Podéis generar el tipo de gráfica que consideréis más oportuno, siempre y cuando permita visualizar en **una única gráfica** ambos resultados (votos y escaños), de manera que sea fácil comparar ambos resultados. Recordad incluir toda la información necesaria para poder interpretar luego la gráfica adecuadamente.

Nota: para realizar el ejercicio, en primer lugar necesitaréis buscar y descargar los datos de interés. Después, habrá que preprocesarlos, para finalmente generar la visualización que se pide en el ejercicio.

In [ ]:
```
# Respuesta
```

Ejercicio 3

Aplicad un clasificador basado en un árbol de decisión (https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n) para predecir los tipos de especie de iris utilizando todos los atributos del dataset. Dibujad el árbol de decisión aprendido utilizando la librería `networkx`.

In [ ]:
```
# Respuesta
```

Ejercicios y preguntas teóricas para la PEC

A continuación encontraréis los **ejercicios y preguntas teóricas que debéis completar en esta PEC** y que forman parte de la evaluación de esta unidad.

Ejercicio 1

Recuperad el fichero `got.csv` utilizado durante la PAC 4 con información sobre las batallas que han tenido lugar a lo largo de la serie de televisión **Juego de Tronos**. Generad un diagrama de puntos (**scatter**) que permita comparar el tamaño (`attacker_size` vs `defender_size`) de los dos principales ejercitos involucrados en cada batalla. Utilizad la columna `attacker_outcome` para mostrar con diferentes colores el resultado de la batalla para el atacante.

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

Viendo la gráfica, podemos afirmar que el tamaño del ejercito es determinante en el resultado de la batalla?

Nota: utilizar el estilo `whitegrid` de Seaborn.

(2.5 punts)

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

Ejercicio 2

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 comunicaban estos reinos ha quedado gravemente afectada. Los Grandes Maestros nos han contratado para resolver las siguientes preguntas:

1. ¿Qué reinos han quedado sin poder acceder a una biblioteca?
2. ¿Cuál es el nombre 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 para que todo el mundo vuelve a tener acceso a una biblioteca? (si no podemos construir ninguna biblioteca?
4. Viendo la respuesta de las preguntas 2 y 3, ¿qué seria lo más óptimo teniendo en cuenta que queremos minimizar costes?

Para poder dar respuestas a las anteriores 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 una 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, **debéis generar una visualización del grafo que os permita responder a las cuatro preguntas planteadas**. Después, **recordad contestar a las cuatro preguntas!**

(2.5 punts)

In [ ]:
```
# 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]
```

In [ ]:
```
# Respuesta
```

Ejercicio 3

El año 2018, a la ciudad de Barcelona se ha producido, de media, 27,22 accidentes con vehiculos implicados. Es por eso que el ayuntamiento de Barcelona requiere nuestros servicios para construir un mapa **interactivo** que muestre los puntos donde se han producido los accidentes.

Nos piden crear un mapa utilizando la librería **geoplotlib** que nos ha de permitir ver la localización de cada accidente y el número de victimas implicadas. Por ahora, únicamente nos piden visualizar los accidentes producidos el mes de **junio del 2018** en **fines de semana** (sábados y domingos).

Nota: Podéis obtener los datos de los accidentes gestionados por la Guàrdia Urbana a la ciudad de Barcelona al portal Open Data BCN (https://opendata-ajuntament.barcelona.cat/data/ca/dataset) y cargar los datos (https://github.com/andrea-cuttone/geoplotlib/wiki/user-Guide#loading-data) a partir de un diccionario o un dataframe.

(2.5 punts)

In [4]:
```
# Respuesta
```

Ejercicio 4

Teniendo en cuenta los datos obtenidos en el ejercicio anterior, el ayuntamiento de Barcelona también nos pide realizar un análisis de las horas cuando se producen los accidentes para intentar prevenirlos.

Se nos pide generar una gráfica que muestre la **distribución** de las horas en las que se han producido los accidentes para cada día de la semana. La misma gráfica debería mostrar también un mapa de puntos de los valores de la hora **sin solapar**.

Nota 1: La visualización de los datos debería ser atractiva y clara para que los responsables del ayuntamiento puedan identificar fácilmente cuales son las franjas horarias donde hay más probabilidad que succeda un accidente.

Nota 2: Por ahora, nos limitarems a estudiar únicamente los datos del mes de **junio del 2018**

(2.5 punts)

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

Soluciones ejercicios para practicar

Ejercicio 1

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).

Generad 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 ambas variables. Por lo que respecta a los detalles de visualización, limitad la visualización de ambos ejes al intervalo (0, 1000000), usad el estilo `whitegrid` de seaborn, y dibujad los puntos y la línea en color verde.

Pista: Podéis usar la función `jointplot` que hemos visto en el notebook de explicación. Pensad en 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.

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

# Mostramos las gráficas en el notebook
%matplotlib inline

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

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

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

Ejercicio 2

Generad una gráfica que muestre el número de votos y el número de escaños obtenidos de los 10 partidos más votados en las elecciones generales en España de 2016 para el congreso. Podéis generar el tipo de gráfica que consideréis más oportuno, siempre y cuando permita visualizar en **una única gráfica** ambos resultados (votos y escaños), de manera que sea fácil comparar ambos resultados. Recordad incluir toda la información necesaria para poder interpretar luego la gráfica adecuadamente.

Nota: para realizar el ejercicio, en primer lugar necesitaréis buscar y descargar los datos de interés. Después, habrá que preprocesarlos, para finalmente generar la visualización que se pide en el ejercicio.

In [ ]:
```
# Cargamos los datos desde un csv
data = pd.read_csv("data/elecciones.csv", decimal=",", thousands=".")
```

In [ ]:
```
# Visualizamos el contenido del DataFrame cargado
data
```

In [ ]:
```
import numpy as np

# Seleccionamos únicamente los 10 partidos más votados
n = 10
most_voted = data.nlargest(n, "Votos")

# Configuramos el plot
bar_width = 0.4
fig, ax1 = plt.subplots(figsize=(12, 8))

# Mostramos los votos en el axis ax1
ax1.bar(np.arange(n), most_voted['Votos'], bar_width, label='Votos')
ax1.grid(False)
ax1.legend()

# Mostramos los escaños en el axis ax2
ax2 = ax1.twinx()
ax2.bar(np.arange(n) + bar_width, most_voted['Escaños'], bar_width, lab
el=u'Escaños', color="green")
ax2.grid(False)
ax2.legend(loc=[0.88, 0.87])

# Etiquetamos las barras
ax1.set_xticks(np.arange(n) + bar_width/2)
ax1.set_xticklabels(['PP', 'PSOE', 'Podemos-IU', 'Cs', 'ECP', 'Podem-EU
PV', 'ERC', 'PSC', 'CDC', 'Podemos-Marea'])

plt.show()
```

Ejercicio 3

Aplicad un clasificador basado en un árbol de decisión (https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n) para predecir los tipos de especie de iris utilizando todos los atributos del dataset. Dibujad el árbol de decisión aprendido utilizando la librería `networkx`.

Primero, creamos el clasificador y exportamos los resultados al formato graphviz, indicando que el reultado no se guarde en un fichero sino que se imprima por pantalla.

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn import tree
%matplotlib inline

# Cargamos el dataset iris
iris = load_iris()

# Creamos un clasificador basado en un árboñ de decisión y exportamos e
l resultado al formato graphviz
dtree = tree.DecisionTreeClassifier()
dtree = dtree.fit(iris.data, iris.target)
tree_str = tree.export_graphviz(dtree, out_file=None)
```

A continuación, parseamos el árbol generado en formato graphviz, y creamos el grafo correspondientes.

In [ ]:
```
import networkx as nx
import math
import matplotlib.pyplot as plt

IDX_INICIO_ARBOL = 14 # El fichero empieza con "digraph Tree", que hemo
s de ignorar

"""
Función que, dada una cadena de texto con formato graphviz, devuelve un
grafo
de networkx conservando los nodos y arristas.
"""
def graphviz_str_to_networkx_graph(tree_str):
    g = nx.Graph()
    nodes, edges = [], []

    # Iteramos sobre todos los elementos del árbol generado.
    # Estos elementos pueden ser nodos o arristas.
    for element in tree_str[IDX_INICIO_ARBOL:-1].split(";"):
        if is_node(element):  # A node
            # Comprovamos si el nodo es un número de dos cifras(> 9).
            # En caso afirmativo, la etiqueta del nodo la podemos encon
trar a partir
            # de la posición 12 de la línia. En caso contrario, esta et
iqueta la
            # podemos encontrar a patir de la posición 11.
            if (int(element[1:3]) > 9):
                nodes.append((int(element[1:3]), {"label": element[12:-
3]}))
            else:
                nodes.append((int(element[1:3]), {"label": element[11:-
3]}))

        elif is_edge(element):  # An edge
            # Eliminamos cualquier espacio en blanco al principio o fin
al de la línia
            element = element.strip()

            # Si encontramos el atributo labeldistance en la línea, lo
 ignoramos
            # y nos quedamos únicamente con cadenas con formato X -> Y
            element = remove_labeldistance_if_present(element)

            # Separamos los dos nodos que conforman la arrista
            edge_left_node, edge_right_node = element.split("->")

            # Añadimos la arrista a la lista de arristas del graf
            edges.append((int(edge_left_node), int(edge_right_node)))

    g.add_nodes_from(nodes)
    g.add_edges_from(edges)

    return g

def is_node(element):
    return len(element) > 2 and (element[3] == "[" or element[4] == "["
)

def is_edge(element):
    return len(element) > 3 and (element [3] == "-" or element[4] == "-
")

def remove_labeldistance_if_present(element):
```

```
    parsed_element = element
    if "labeldistance" in element:
        parsed_element = element.split("[")[0]

    return parsed_element


"""
Función que calcula la posición de los nodos a mostrar en el grafo en f
unción
de la profundidad que presentan en el árbol.
"""
def tree_positions(g):
    pos = {}
    max_depth = get_max_depth(g)
    depths = {i: 1.0 for i in range(max_depth+1)}
    for node in g.nodes():
        d = get_depth(g, node)
        pos[node] = ( depths[d] / (get_num_nodes_in_depth(g, node) + 1
), max_depth - d)
        depths[d] = depths[d] + 1
    return pos

def get_max_depth(g):
    return max([get_depth(g, n) for n in g.nodes()])

def get_depth(g, node_id):
    return nx.shortest_path_length(g, source=0, target=node_id)

def get_num_nodes_in_depth(g, node_id):
    return 2**get_depth(g, node_id)
```

In [ ]:
```
plt.figure(1, figsize=(20, 20))

# Creamos el grafo
g = graphviz_str_to_networkx_graph(tree_str)

# Calculamos las posiciones de los nodos
pos = tree_positions(g)

# Sustituimos los caracteres "\\n" en las etiquetas para mantener los s
altos de línea
l = nx.get_node_attributes(g, "label")
l = {k: v.replace("\\n", "\n") for k, v in l.items()}

# Dibujamos el grafo
nx.draw_networkx(g, pos=pos, labels=l, node_shape="s", node_size=4000,
node_color='#0fafda', font_size=9)
```