prog_datasci_4_scilib_ejerciciosResueltos

October 26, 2020

1 Programación para *Data Science*

1.1 Unidad 4: Librerías científicas 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

Calcula la norma y el determinante de la siguiente matriz: `[[1, 0], [2, -1]]`.

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

1.2.2 Ejercicio 2

Genera una lista de 100 valores enteros aleatorios de 0-9. Realiza los siguientes cálculos utilizando métodos de *numpy* : - Media y desviación estándar de los valores de la lista - Valor máximo y mínimo - Suma todos los valores de la lista - Consigue una lista de valores únicos

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

1.2.3 Ejercicio 3

Crea una matriz 10x10 que se ajuste a matriz identidad utilizando generadores básicos de arrays. Crea una matriz identidad 10x10 (esta vez utilizando generadores específicos de matrices identidad) y comprueba que ambas matrices son iguales.

Consideraciones: - La primera matriz debe crearse utilizando constructores básicos de arrays, como los presentados a los Notebooks de teoría. * La segunda matriz debe generarse utilizando el generador de matrices identidad de NumPy. * La comparación debe devolver True si las matrices son iguales (un único True), False en caso contrario.

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

1.2.4 Ejercicio 4

Ordena la matriz bidimensional `[[5,1,7], [0,7,4], [7,23,1]]` por columnas utilizando como algoritmo de ordenación el Heapsort.

Nota: Para resolver este ejercicio os puede ser de utilidad la función suerte de NumPy que nos permite ordenar los elementos de un array N-dimensional.

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

1.2.5 Ejercicio 5

Crea una matriz, I, de tamaño 10x10 donde todos sus elementos tendrán el valor `0` utilizando generadores de matrices de NumPy. Modifica la matriz anterior de tal forma que `Y [i, j] = y * j` por todo`i` y `j` pares, es decir, para todas las posiciones donde tanto `i` como `j` son pares.

[6]:
```
# Respuesta
```

1.2.6 Ejercicio 6

Considera la matriz [[1, 2, 3], [4, 5, 6]]. Transformala en la matriz 1D [1, 2, 3, 4, 5, 6].

Nota: Para resolver este ejercicio te pueden ser de utilidad las funciones ravel o flatten de Numpy que nos permite transformar un array 2D numpy a 1D.

[7]:
```
# Respuesta
```

1.2.7 Ejercicio 7

Crea una matriz, Z, de tamaño 8x8 que vaya alternando los valores 0 y 1, siguiendo un patrón de tablero de damas.

[8]:
```
# Respuesta
```

1.2.8 Ejercicio 8

El archivo `us_births.csv` contiene datos de los nacimientos en Estados Unidos durante los años 1994 a 2003. Carga el archivo representa de forma gráfica los nacimientos agrupados por mes y año. Utiliza un gráfico 2D de tipo *scatter*.

Consideraciones: * La figura debe contener 10 gráficos (1 por año) * El valor de cada punto debe corresponder con el total de nacimientos por año y mes. * Incluye una leyenda que contenga los años. Puedes diferenciar cada gráfico como mejor te parezca (diferentes símbolos / colores / combinación de ambas)

[9]:
```
# Respuesta
```

1.2.9 Ejercicio 9

Carga los datos del archivo `company_sales.csv`, que puedes encontrar en la carpeta `data`, en forma de *dataframe*. Este conjunto de datos recoge información sobre las ventas mensuales de una empresa de cosmética y productos de higiene.

Parte A:

Muestra un gráfico de líneas de puntos de las ventas totales (*total_profit*) por cada mes (los meses están numerados 1 a 12).

Consideraciones:

- La línea debe ser con puntos (*dotted line*) de color rojo y de tamaño 3.
- Se debe mostrar la leyenda en la parte inferior derecha.

Parte B:

Muestra un gráfico circular (*pie chart*) con el número de unidades vendidas este año por producto en porcentaje.

Consideraciones:

- Debéis incluir el porcentaje (%) de cada producto dentro de su respectivo espacio del gráfico.
- Se debe mostrar la leyenda en la parte derecha.

[10]:
```
# Respuesta
```

1.2.10 Ejercicio 10

Carga los datos del archivo `got.csv` a un *dataframe*. Este conjunto de datos recoge información de la Guerra de los Cinco Reyes de las novelas de Canción de hielo y de fuego de George RR Martin.

Muestra el número de filas del *dataframe* y las etiquetas de los ejes.

[11]:
```
# Respuesta
```

1.2.11 Ejercicio 11

Agrupa los datos cargados en el ejercicio 1 por el principal bando atacante (*attacker _1*). *Para cada posición, muestra el número de batallas y el resultado de la batalla (el resultado se encuentra en el campo (*attacker_outcome*).

[12]:
```
# Respuesta
```

1.2.12 Ejercicio 12

Muestra los datos de las batallas donde el número de participantes supera los 15000 "hombres" (contando los dos bandos: *attacker_size* y *defender_size*), el resultado haya sido desfavorable por el atacante, y la batalla haya estado en invierno (*summer = 0*).

[13]:
```
# Respuesta
```

1.2.13 Ejercicio 13

Cuenta cuantos sitios aparecen más de una vez en el *dataframe* (campo *location*), utilizando las funciones de la librería *pandas*. Existe algún lugar donde haya habido más de una batalla? Comprueba qué bandos estaban implicados.

[14]:
```
# Respuesta
```

1.2.14 Ejercicio 14

Carga los datos desde el archivo `netflixtitles.csv`, que puedes encontrar en la carpeta `data`, en un *dataframe*. Este conjunto de datos recoje información sobre películas y séries de TV de Netflix hasta el 2019.

Muestra las películas estrenadas el *2019* que tienen una duración superior a *100* minutos.

Nota: En el Notebook de teoría hemos visto operaciones básicas de filtrado. Para resolver este ejercicio necesitarás investigar cómo aplicar condiciones más complejas.

Nota 2: Los valores de la columna de duración (*duration*) son strings, ya que combinan números y letras. Por lo tanto, se tienen que transformar a int una vez se haya construido el subset de datos de películas. Te puede ser de utilidad la función astype de pandas.

[15]:
```
# Respuesta
```

1.3 Soluciones a los ejercicios para practicar

1.3.1 Ejercicio 1

Calcula la norma y el determinante de la siguiente matriz: `[[1, 0], [2, -1]]`.

[16]:
```
# Cargamos las librerías necesarias
import numpy as np

# Creamos la matriz como array bidimensional
m = np.array([[1, 0], [2, -1]])

# Calculamos la normal y el determinante utilizando el módulo linalg
norm = np.linalg.norm(m)
det = np.linalg.det(m)

print("La matriz m es:", m)
print("La norma de m es: ", norm)
print("El determinante de m es:", det)
```

```
La matriz m es: [[ 1  0]
 [ 2 -1]]
La norma de m es:  2.449489742783178
El determinante de m es: -1.0
```

1.3.2 Ejercicio 2

Genera una lista de 100 valores enteros aleatorios de 0-9. Realiza los siguientes cálculos utilizando métodos de_numpy_ : - Media y desviación estándar de los valores de la lista - Valor máximo y mínimo - Suma todos los valores de la lista - Consigue una lista de valores únicos

[17]:
```
# Cargamos las librerías necesarias
from random import randint
import numpy as np

# Utilizamos la función rand del módulo random y list comprensión para generar␣
 ↪la lista,
# como ya habíamos visto en unidades anteriores

rand_ints = [randint(0, 9) for _ in range(100)]

# Para calcular la media y la desviación estándar utilizamos las funciones mean␣
 ↪y std (standard deviation)
# respectivamente
m = np.mean(rand_ints)
std = np.std(rand_ints)
```

```
print("Nuestra lista de valores aleatorios es:", rand_ints)
print("Su media es %f, y su desviación estándar es %f" %(m, std))

# Para los valores máximo y mínimo, utilizamos max y min
max_val = np.max(rand_ints)
min_val = np.min(rand_ints)

print("Sus valores máximo y mínimo son, respectivamente: %d y %d" %(max_val,␣
 ↪min_val))

# Para sumar todos los elementos de la lista, la función sum
sum_values = np.sum(rand_ints)

print("El resultado de la suma de todos los valores de la lista es", sum_values)

# Y finalmente para conseguir una lista de valores únicos, la función unique
unique_values = np.unique(rand_ints)

print("Y la lista de valores únicos es", unique_values)
```

```
Nuestra lista de valores aleatorios es: [9, 2, 9, 0, 7, 8, 6, 6, 2, 4, 5, 7, 9,
3, 5, 7, 4, 3, 2, 4, 9, 5, 5, 4, 3, 1, 9, 3, 4, 8, 2, 7, 9, 8, 6, 2, 9, 4, 4, 7,
4, 1, 0, 3, 1, 8, 3, 1, 1, 8, 3, 0, 3, 3, 7, 0, 9, 8, 3, 8, 0, 1, 4, 8, 2, 2, 5,
5, 8, 7, 4, 1, 8, 1, 2, 6, 4, 2, 4, 9, 1, 0, 3, 6, 0, 4, 1, 3, 3, 1, 7, 4, 9, 3,
6, 6, 4, 9, 5, 4]
Su media es 4.470000, y su desviación estándar es 2.794477
Sus valores máximo y mínimo son, respectivamente: 9 y 0
El resultado de la suma de todos los valores de la lista es 447
Y la lista de valores únicos es [0 1 2 3 4 5 6 7 8 9]
```

1.3.3 Ejercicio 3

Crea una matriz 10x10 que se ajuste a matriz identidad utilizando generadores básicos de arrays. Crea una matriz identidad 10x10 (esta vez utilizando generadores específicos de matrices identidad) y comprueba que ambas matrices son iguales.

Consideraciones: - La primera matriz debe crearse utilizando constructores básicos de arrays, como los presentados a los Notebooks de teoría. * La segunda matriz debe generarse utilizando el generador de matrices identidad de NumPy. * La comparación debe devolver True si las matrices son iguales (un único True), False en caso contrario.

[18]:
```
# Cargamos las librerías necesarias
import numpy as np

# Podemos definir la matriz de forma manual como una lista de listas
m0 = np.array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0,], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0,],
               [0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
```

```
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
               [0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])

# O empezar por una 10x10 con todo ceros y asignar unos en la diagonal
# recorriendo la matriz por filas y columnas
m1 = np.zeros((10,10), dtype=int)
for i, r in enumerate(m1):
    r[i] = 1

# Por otra parte, definimos la identidad 10x10 utilizando el constructor␣
 ↪"built-in"
m2 = np.identity(10, dtype=int)

# Y realizamos la comparación. Para ello podemos utilizar el método array_equal␣
 ↪de NumPy
# que nos compara si dos arrays son iguales
print(np.array_equal(m0, m2))
print(np.array_equal(m1, m2))

# También podemos comprobar que la comparación falla si modificamos cualquier␣
 ↪valor de la matriz
m0[1, 2] = 1
print(np.array_equal(m0, m2))
```

```
True
True
False
```

1.3.4 Ejercicio 4

Ordena la matriz bidimensional `[[5,1,7], [0,7,4], [7,23,1]]` por columnas utilizando como algoritmo de ordenación el Heapsort.

Nota: Para resolver este ejercicio os puede ser de utilidad la función suerte de NumPy que nos permite ordenar los elementos de un array N-dimensional.

[19]:
```
# Cargamos las librerías necesarias
import numpy as np

# Creamos la matriz indicada en el enunciado
matriz_ej3 = [[5,1,7], [0,7,4], [7,23,1]]

# Utilizamos la función suerte de NumPy para ordenar la matriz anterior.
# El parámetro axis = 0 nos indica que queremos ordenar por columnas
# Y el parámetro kind nos permite especificar el algoritmo de ordenación, en␣
 ↪este caso 'heapsort'.
```

```
np.sort(matriz_ej3, axis=0, kind='heapsort')
```

[19]:
```
array([[ 0,  1,  1],
       [ 5,  7,  4],
       [ 7, 23,  7]])
```

1.3.5 Ejercicio 5

Crea una matriz, I, de tamaño 10x10 donde todos sus elementos tendrán el valor `0` utilizando generadores de matrices de NumPy. Modifica la matriz anterior de tal forma que `Y [i, j] = y * j` por todo`i` y `j` pares, es decir, para todas las posiciones donde tanto `i` como `j` son pares.

[20]:
```
# Cargamos las librerías necesarias
import numpy as np

# Definimos un par de constantes por el número de filas y columnas
NUM_FILAS = 10
NUM_COLUMNAS = 10

# Creamos una matriz de 10x10 donde todos los valores serán 0.
# Para ello utilizamos la función ceros que hemos visto en el Notebook de␣
 ↪teoría.
matriz_ej4 = np.zeros((NUM_FILAS, NUM_COLUMNAS))

# Mostramos la matriz original por pantalla
print(matriz_ej4)

# Iteramos sobre todas las filas y columnas de la matriz generada utilizando un␣
 ↪doble bucle for.
# El primer bucle nos recorrerá el índice de las filas, mientras que el segundo␣
 ↪bucle nos recurrirá
# El índice de las columnas.
for i in range(0, NUM_FILAS):
    for j in range(0, NUM_COLUMNAS):
        # Comprobamos que el índice de la fila y el índice de la columna sean␣
 ↪ambos pares.
        if i % 2 == 0 and j % 2 == 0:
        # En caso afirmativo, pasamos a realizar la asignación indicada.
            matriz_ej4[i, j] = i * j

# Mostramos el resultado final por pantalla
print(matriz_ej4)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  4.  0.  8.  0. 12.  0. 16.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  8.  0. 16.  0. 24.  0. 32.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 12.  0. 24.  0. 36.  0. 48.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 16.  0. 32.  0. 48.  0. 64.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

1.3.6 Ejercicio 6

Considera la matriz [[1, 2, 3], [4, 5, 6]]. Transformala en la matriz 1D [1, 2, 3, 4, 5, 6].

Nota: Para resolver este ejercicio te pueden ser de utilidad las funciones ravel o flatten de Numpy que nos permite transformar un array 2D numpy a 1D.

[21]:
```python
# Cargamos las librerías necesarias
import numpy as np

# Creamos la matriz indicada en el anunciado
matriz_ej6 =  np.array([[1, 2, 3], [4, 5, 6]])

#Opción 1: ravel
out1 = np.ravel(matriz_ej6)
print('Opción 1: ravel\n',out1)

#Opción 2: flatten
out2 = matriz_ej6.flatten()
print('Opción 2: flatten\n',out1)
```

```
Opción 1: ravel
 [1 2 3 4 5 6]
Opción 2: flatten
 [1 2 3 4 5 6]
```

1.3.7 Ejercicio 7

Crea una matriz, Z, de tamaño 8x8 que vaya alternando los valores 0 y 1, siguiendo un patrón de tablero de damas.

[22]:
```
# Cargamos las librerías necesarias
import numpy as np
import matplotlib.pyplot as plt

# Creamos una matriz de zeros 8x8
Z = np.zeros((8,8),dtype=int)

# Comenzando en la fila 1 y cada 2, ponemos 1 cada 2 columnas
Z[1::2,::2] = 1

# Comenzando en columna 1 y cada 2, ponemos 1 cada 2 filas
Z[::2,1::2] = 1
print(Z)
```

```
[[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
```

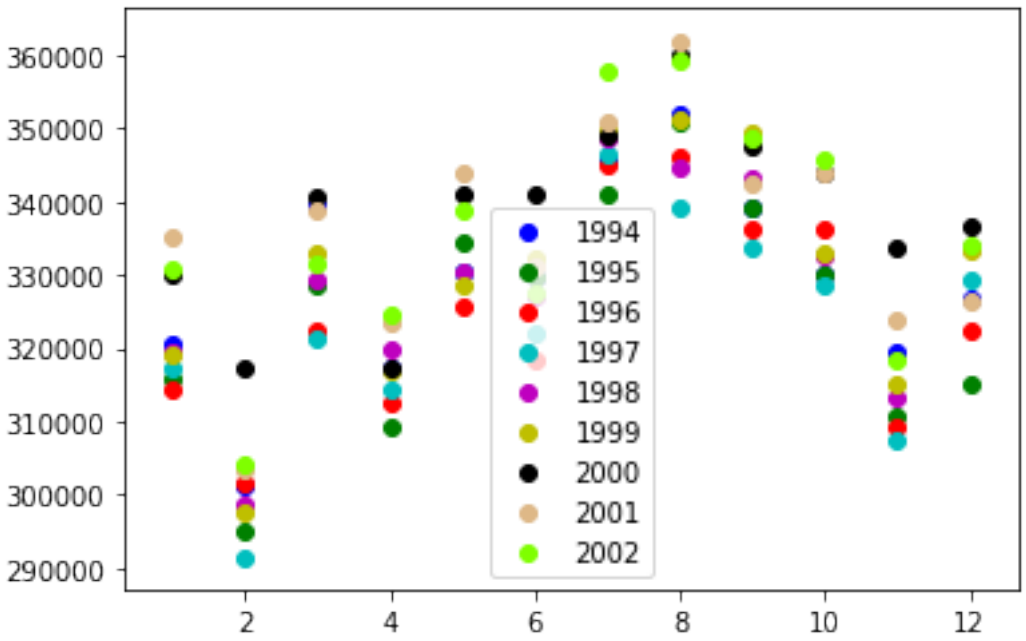
1.3.8 Ejercicio 8

El archivo `us_births.csv` contiene datos de los nacimientos en Estados Unidos durante los años 1994 a 2003. Carga el archivo representa de forma gráfica los nacimientos agrupados por mes y año. Utiliza un gráfico 2D de tipo *scatter*.

Consideraciones: * La figura debe contener 10 gráficos (1 por año) * El valor de cada punto debe corresponder con el total de nacimientos por año y mes. * Incluye una leyenda que contenga los años. Puedes diferenciar cada gráfico como mejor te parezca (diferentes símbolos / colores / combinación de ambas)

[23]:
```
# Cargamos las librerías necesarias
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Cargamos los datos del archivo
data = pd.read_csv('data/us_births.csv')
```

```
# Las agrupamos por año y mes
grouped = data.groupby(['year', 'month'])

# Grouped contiene un conjunto de dataframes identificados por año y mes, pero␣
 ↪los datos
# referentes al nacimientos cada dataframe aún están separadas por día.
# Sumamos los nacimientos de cada mes.
parsed_years = grouped['births'].sum()

# Separamos los datos en una matriz de 10x12 (10 años con 12 meses por año)
parsed_years = np.array(parsed_years).reshape(10, 12)


# Llegados a este punto ya tenemos los datos listos para ser mostrados, ahora␣
 ↪nos toca
# definir los atributos del gráfico (ejes, colores, etc.)

# Representaremos los meses con una lista de 1 a 12
months = range(1, 13, 1)

# Y los años (para la leyenda) con su valor. Podemos generar la lista a mano o␣
 ↪obtenerla
# Como los valores únicos de la columna years
years = data['year'].unique()

# Finalmente definimos una lista con 10 colores, uno para cada año
colours = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'burlywood', 'chartreuse']

# Una vez tenemos los ejes, colores y datos definidas, podemos generar un␣
 ↪scatter por cada año
# Cada punto en la gráfica (x, y) vendrá definido por los elementos de las␣
 ↪listas months y year_data
# Label definirá qué etiqueta se le dará al scatter a la leyenda, y␣
 ↪corresponderá con el año
for year_data, y, c in zip(parsed_years, years, colours):
    plt.scatter(months, year_data, color=c, label=y)

# Y finalmente incluimos también la leyenda (loc = 0 la posiciona "en el mejor␣
 ↪lugar" dentro
# Del límite del gráfico). Y mostramos la figura.
plt.legend(loc=0)
plt.show()
```



1.3.9 Ejercicio 9

Carga los datos del archivo `company_sales.csv`, que puedes encontrar en la carpeta `data`, en forma de *dataframe*. Este conjunto de datos recoge información sobre las ventas mensuales de una empresa de cosmética y productos de higiene.

Parte A:

Muestra un gráfico de líneas de puntos de las ventas totales (*total_profit*) por cada mes (los meses están numerados 1 a 12).

Consideraciones:

- La línea debe ser con puntos (*dotted line*) de color rojo y de tamaño 3.
- Se debe mostrar la leyenda en la parte inferior derecha.

Parte B:

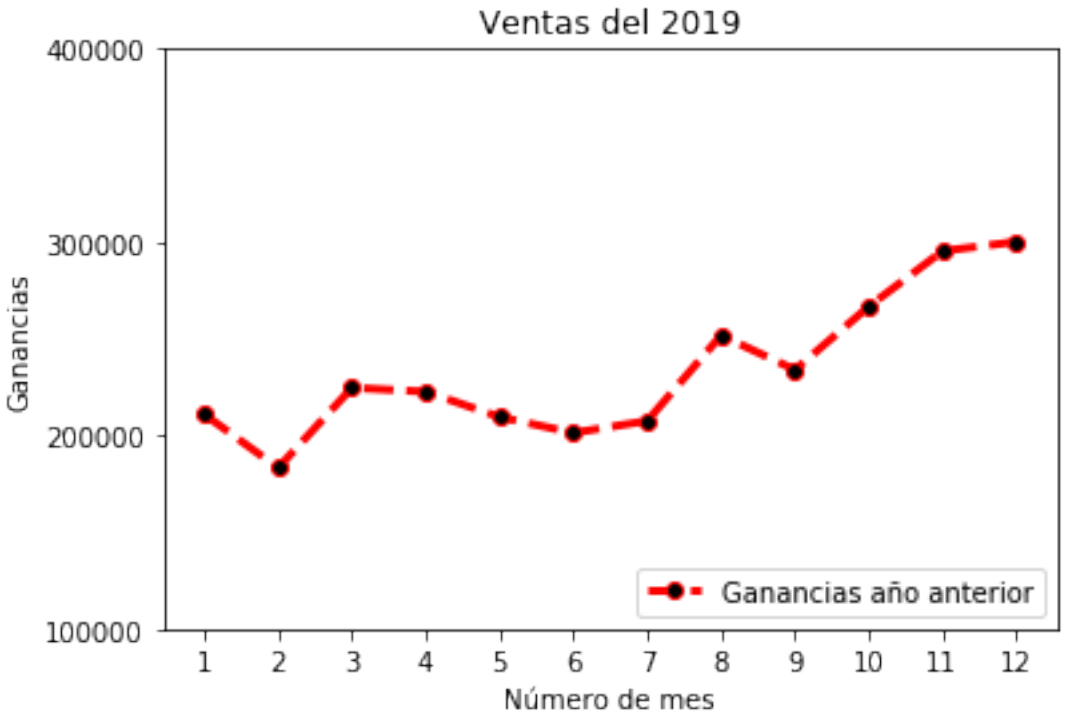
Muestra un gráfico circular (*pie chart*) con el número de unidades vendidas este año por producto en porcentaje.

Consideraciones:

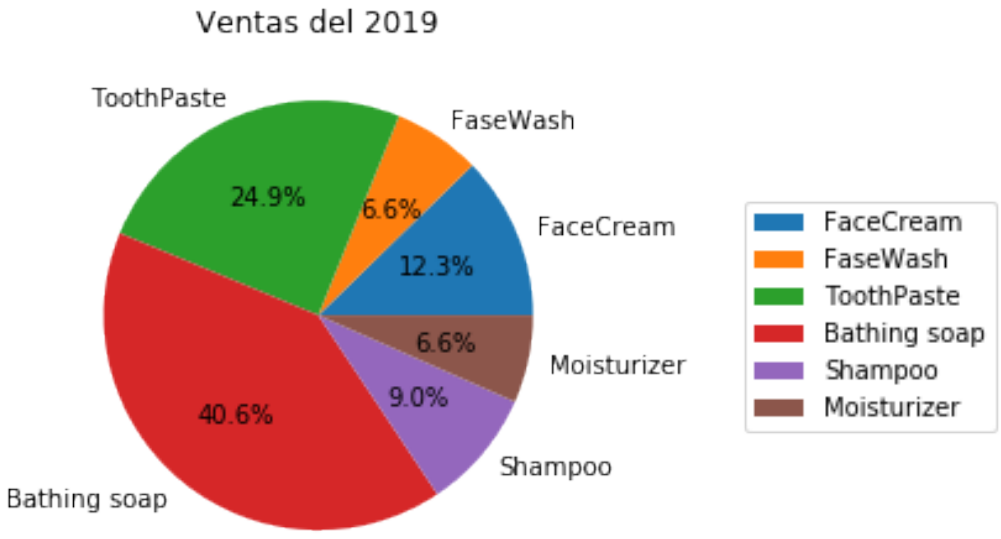
- Debéis incluir el porcentaje (%) de cada producto dentro de su respectivo espacio del gráfico.
- Se debe mostrar la leyenda en la parte derecha.

[24]:
```
# Cargamos las librerías necesarias
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Importamos el fichero a un dataframe de pandas
df = pd.read_csv("data/company_sales.csv")

# Seleccionamos las dos columnas que nos interesan
profitList = df ['total_profit']
monthList  = df ['month_number']

# Graficamos las ganancias (y) en función del mes (x), con las características␣
 ↪solicitadas en el enunciado
plt.plot(monthList, profitList, label = 'Ganancias año anterior',
      color='r', marker='o', markerfacecolor='k',
      linestyle='--', linewidth=3)

# Especificamos la información de los ejes
plt.xlabel('Número de mes')
plt.ylabel('Ganancias')
plt.legend(loc='lower right')
plt.title('Ventas del 2019')
plt.xticks(monthList)
plt.yticks([100000, 200000, 300000, 400000])
plt.show()
```



[25]:
```
# Cargamos las librerías necesarias
import pandas as pd
import matplotlib.pyplot as plt

# Importamos el fichero a un dataframe de pandas
df = pd.read_csv("data/company_sales.csv")


# Creamos un vector con la suma de cada uno de los productos
salesData   = [df ['facecream'].sum(), df ['facewash'].sum(), df ['toothpaste'].
 ↪sum(),
         df ['bathingsoap'].sum(), df ['shampoo'].sum(), df ['moisturizer'].
 ↪sum()]

# Creamos la información para la leyenda
labels = ['FaceCream', 'FaseWash', 'ToothPaste', 'Bathing soap', 'Shampoo',␣
 ↪'Moisturizer']

# Especificamos la información de los ejes
plt.axis("equal")
plt.pie(salesData, labels=labels, autopct='%1.1f%%') # autopct muestra el␣
 ↪porcentaje (%)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.title('Ventas del 2019')
plt.show()
```



1.3.10 Ejercicio 10

Carga los datos del archivo `got.csv` a un *dataframe*. Este conjunto de datos recoge información de la Guerra de los Cinco Reyes de las novelas de Canción de hielo y de fuego de George RR Martin.

Muestra el número de filas del *dataframe* y las etiquetas de los ejes.

[26]:
```
# Cargamos las librerías necesarias
import pandas as pd

# Leemos el archivo csv con el método read_csv
data = pd.read_csv('data/got.csv')

# Mostramos el numero de filas
print("Número de filas:", len(data))

# Mostramos las etiquetas de los ejes
print("Etiquetas:", data.axes)
```

```
Número de filas: 38
Etiquetas: [RangeIndex(start=0, stop=38, step=1), Index(['name', 'year',
'battle_number', 'attacker_king', 'defender_king',
       'attacker_1', 'attacker_2', 'attacker_3', 'attacker_4', 'defender_1',
       'defender_2', 'defender_3', 'defender_4', 'attacker_outcome',
       'battle_type', 'major_death', 'major_capture', 'attacker_size',
       'defender_size', 'attacker_commander', 'defender_commander', 'summer',
       'location', 'region', 'note'],
      dtype='object')]
```

1.3.11 Ejercicio 11

Agrupa los datos cargados en el ejercicio 1 por el principal bando atacante (*attacker _1*). *Para cada posición, muestra el número de batallas y el resultado de la batalla (el resultado se encuentra en el campo (*attacker_outcome*).*

[27]:
```
# Tal y como estaba definido el enunciado se podía interpretar de dos formas:
# 1 - que se mostrara el número de batallas ganadas y perdidas por cada bando
# 2 - que se mostrara el número de batalla (identificador) y el resultado de␣
 ↪ésta por cada
# Bando.

# Resolveremos ambos casos.

# Cargamos las librerías necesarias
import pandas as pd
import numpy as np
```

```
# En cualquiera de los casos tenemos que cargar los datos y agruparlos por el␣
 ↪bando atacante
data = pd.read_csv('data/got.csv')
grouped = data.groupby('attacker_1')


# Caso 1. Cada grupo contiene como etiqueta el bando (primer dato que nos␣
 ↪piden) y como
# datos un dataframe. Podemos acceder a cada una de las columnas como si de un␣
 ↪diccionario
# Se tratara, y extraer sus valores.
print("CASO 1:\n")
# Para cada agrupación
for name, g in grouped:
    # Mostramos el bando
    print(name)
    # Mostramos el numero de batalla y el resultado para cada batalla
    for battle, outcome in zip(g['battle_number'].values, g['attacker_outcome'].
 ↪values):
        print("battle: %s, outcome: %s" % (battle, outcome))


# Caso 2. La estructura es la misma que en el caso anterior, pero ahora tenemos␣
 ↪que contar cuántas
# batallas se han ganado o perdido.
print("\n--------------------\n")
print("CASO 2: \n")
for name, g in grouped:
    # Empezamos contando el número de batallas extrayendo la información del␣
 ↪dataframe
    outcome = g['attacker_outcome'].values
    s = len(outcome)

    # Después contamos cuántas han sido ganadas y perdidas
    # count_nonzero evalúa una expresión booleana y devuelve el número de␣
 ↪coincidencias
    # que no son cero
    w = np.count_nonzero(outcome == 'win')
    l = np.count_nonzero(outcome == 'loss')

    # Podemos hacer lo mismo utilizando count si transformamos outcome a una␣
 ↪lista
    outcome = list(outcome)
    w = outcome.count('win')
    l = outcome.count('loss')
```

```
    # Finalmente mostramos el resultado
    print(name)
    print("total: %s, won: %s, lost: %s" % (s, w, l))
```

```
CASO 1:

Baratheon
battle: 16, outcome: win
battle: 20, outcome: loss
battle: 31, outcome: win
battle: 34, outcome: win
battle: 35, outcome: win
battle: 38, outcome: nan
Bolton
battle: 14, outcome: win
battle: 29, outcome: win
Bracken
battle: 37, outcome: win
Brave Companions
battle: 30, outcome: win
Brotherhood without Banners
battle: 23, outcome: win
Darry
battle: 21, outcome: win
Free folk
battle: 28, outcome: loss
Frey
battle: 26, outcome: win
battle: 27, outcome: win
Greyjoy
battle: 8, outcome: win
battle: 9, outcome: win
battle: 10, outcome: win
battle: 12, outcome: win
battle: 13, outcome: win
battle: 32, outcome: win
battle: 33, outcome: win
Lannister
battle: 1, outcome: win
battle: 2, outcome: win
battle: 3, outcome: win
battle: 7, outcome: win
battle: 17, outcome: loss
battle: 24, outcome: win
battle: 25, outcome: win
battle: 36, outcome: win
Stark
```

```
battle: 4, outcome: loss
battle: 5, outcome: win
battle: 6, outcome: win
battle: 11, outcome: win
battle: 15, outcome: win
battle: 18, outcome: win
battle: 19, outcome: win
battle: 22, outcome: loss

---------------------

CASO 2:

Baratheon
total: 6, won: 4, lost: 1
Bolton
total: 2, won: 2, lost: 0
Bracken
total: 1, won: 1, lost: 0
Brave Companions
total: 1, won: 1, lost: 0
Brotherhood without Banners
total: 1, won: 1, lost: 0
Darry
total: 1, won: 1, lost: 0
Free folk
total: 1, won: 0, lost: 1
Frey
total: 2, won: 2, lost: 0
Greyjoy
total: 7, won: 7, lost: 0
Lannister
total: 8, won: 7, lost: 1
Stark
total: 8, won: 6, lost: 2
```

1.3.12 Ejercicio 12

Muestra los datos de las batallas donde el número de participantes supera los 15000 "hombres" (contando los dos bandos: *attacker_size* y *defender_size*), el resultado haya sido desfavorable por el atacante, y la batalla haya estado en invierno (*summer = 0*).

[28]:
```python
# Cargamos las librerías necesarias
import pandas as pd

# Cargamos los datos
data = pd.read_csv('data/got.csv')
```

```
# Definimos las condiciones
# La suma de las tropas atacantes y defensoras debe ser superior a 15000
size_cond = (data['attacker_size'] + data['defender_size'] > 15000)

# El resultado debe ser desfavorable para el atacante
outcome_cond = data['attacker_outcome'] == 'loss'

# Y la batalla debe haber sucedido en invierno
seasson_cond = data['summer'] == 0

# Accedemos a los datos combinando todas las condiciones y las mostramos por␣
↪pantalla
print(data[size_cond & outcome_cond & seasson_cond])

# Podemos ver que se trata de una única batalla (battle_number = 28),␣
↪correspondiente a
# Battle of Castle Black
```

```
                    name  year  battle_number      attacker_king  \
27  Battle of Castle Black   300             28  Stannis Baratheon

   defender_king attacker_1 attacker_2 attacker_3 attacker_4     defender_1  \
27  Mance Rayder  Free folk     Thenns     Giants        NaN  Night's Watch

    … major_death  major_capture  attacker_size defender_size  \
27  …         1.0            1.0       100000.0        1240.0

                                  attacker_commander  \
27  Mance Rayder, Tormund Giantsbane, Harma Dogshe…

                                  defender_commander  summer      location  \
27  Stannis Baratheon, Jon Snow, Donal Noye, Cotte…     0.0  Castle Black

            region note
27  Beyond the Wall  NaN

[1 rows x 25 columns]
```

1.3.13 Ejercicio 13

Cuenta cuantos sitios aparecen más de una vez en el *dataframe* (campo *location*), utilizando las funciones de la librería *pandas*. Existe algún lugar donde haya habido más de una batalla? Comprueba qué bandos estaban implicados.

[29]:
```
# Cargamos las librerías necesarias
import pandas as pd

# Cargamos los datos
data = pd.read_csv('data/got.csv')

# Hay diferentes formas de resolver el problema. Una de ellas sería utilizando␣
 ↪el método
# value_counts que dada una columna nos devuelve el número de ocurrencias de␣
 ↪los valores de esta
# Podemos buscar dentro del dataframe de forma similar al ejercicio anterior␣
 ↪utilizando> 1
locations = data['location'].value_counts()
rep_loc = locations[locations > 1]
print("Número de lugares que aparecen más de una vez al dataframe:",␣
 ↪len(rep_loc))

# De forma alternativa podemos agrupar los datos para localización y computar␣
 ↪el tamaño de los grupos
# (Cuántas veces aparece cada uno)
grouped = data.groupby('location')
locations = grouped.size()

# Podemos acceder al dataframe utilizando el método loc y pasando una lista␣
 ↪booleana de posiciones
# correspondientes si el contenido de la posición es mayor que 1. Por ello␣
 ↪podemos utilizar
# una lambda function
rep_loc2 = locations.loc[lambda x: x>1]
print("Número de sitios que aparecen más de una vez al dataframe (v2):",␣
 ↪len(rep_loc2))

# Para mostrar los bandos principales implicados en lugares donde ha habido más␣
 ↪de una batalla
# podemos acceder a las filas del dataframe que tienen como localización los␣
 ↪sitios repetidos
# que acabamos de obtener. Podemos utilizar cualquier identificativo de los␣
 ↪bandos:
# Attacker_1 vs defender_1, attacker_kings vs defender_king, etc

# Recorremos la lista de localizaciones repetidas
for loc, c in rep_loc.iteritems():
    # Mostramos la localitzación
    print("\n%s:" % loc)
    # Definimos el filtro para esta localización y filtramos las batallas que␣
 ↪han tenido lugar aquí
```

```
    loc_cond = data["location"] == loc
    battles = data[loc_cond]

    # Finalmente mostramos los bandos
    for a, d in zip(battles['attacker_1'], battles['defender_1']):
        print("%s vs %s" %(a,d))
```

```
Número de lugares que aparecen más de una vez al dataframe: 8
Número de sitios que aparecen más de una vez al dataframe (v2): 8

Riverrun:
Lannister vs Tully
Stark vs Lannister
Lannister vs Tully

Winterfell:
Greyjoy vs Stark
Bolton vs Stark
Baratheon vs Bolton

Deepwood Motte:
Greyjoy vs Stark
Baratheon vs Greyjoy

Harrenhal:
Stark vs Lannister
Lannister vs Brave Companions

Storm's End:
Baratheon vs Baratheon
Baratheon vs Baratheon

Darry:
Lannister vs Darry
Darry vs Lannister

Torrhen's Square:
Stark vs Greyjoy
Greyjoy vs Stark

Moat Cailin:
Greyjoy vs Stark
Bolton vs Greyjoy
```

1.3.14 Ejercicio 14

Carga los datos desde el archivo `netflixtitles.csv`, que puedes encontrar en la carpeta `data`, en un *dataframe*. Este conjunto de datos recoje información sobre películas y séries de TV de Netflix (https://www.kaggle.com/shivamb/netflix-shows) hasta el 2019.

Muestra las películas estrenadas el *2019* que tienen una duración superior a *100* minutos.

Nota: En el Notebook de teoría hemos visto operaciones básicas de filtrado. Para resolver este ejercicio necesitarás investigar cómo aplicar condiciones más complejas.

Nota 2: Los valores de la columna de duración (*duration*) son strings, ya que combinan números y letras. Por lo tanto, se tienen que transformar a int una vez se haya construido el subset de datos de películas. Te puede ser de utilidad la función astype de pandas.

[30]:
```
# Cargamos las librerías necesarias
import numpy as np
import pandas as pd

# Importamos el fichero a un dataframe de pandas
data = pd.read_csv("data/netflixtitles.csv")

# Filtramos el dataset por el tipo "Movie"
data_movies=data[data['type']=="Movie"]

# Podemos mostrar los 10 primeros valores únicos de la columna duration de la␣
 ↪siguiente manera.
print(data_movies['duration'][0:10])

# Vemos que los minutos los coge como string, los tenemos que convertir en␣
 ↪número.
data_movies['duration']= data_movies['duration'].astype(int)

# Extraemos los valores que nos interesan los valores mostrados
# El operador & nos permite aplicar un operador lógico AND
data_ex3 = data_movies[(data_movies["release_year"] == 2019) &␣
 ↪(data_movies["duration"] > 100)]


# Accedemos a la columna "title" del conjunto filtrado y mostramos sus valores␣
 ↪únicos.
data_ex3['title']
```

```
0       90
1       94
4       99
6      110
7       60
9       90
```

```
10     78
11     95
12     58
13     62
Name: duration, dtype: object
```

[30]:
```
35                            Article 15
57                    The World We Make
102           In the Shadow of the Moon
109                               Malaal
119                 Oh! Baby (Malayalam)
                      …
5347    Ek Ladki Ko Dekha Toh Aisa Laga
5359                      Music Teacher
5365                       The Ruthless
5385      HOMECOMING: A film by Beyoncé
5469                                706
Name: title, Length: 134, dtype: object
```