accidentes_en_madrid_enunciados

May 6, 2022

Estadísticas de Accidentes en Madrid

Ivonne Yanez Mendoza

17 de mayo, 2022

0.1 Introducción

En este proyecto se desarrolla en Python un análisis básico de datos sobre accidentes de tráfico de la Comunidad de Madrid. Los datos provienen de ***. A partir de ellos he realizado … He conseguido … pero no he podido …

Completa tú el breve apartado anterior. Elimina este párrafo en verde. A partir de ahora, pon en azul los comentarios tuyos, dejando en negro los míos, del enunciado.

0.2 Librerías

Pongamos todas las librerías necesarias al principio, tal como propone el estilo `pep-8`. Ej.: PEP 8 – Style Guide for Python Code.

[1]:
```
from collections import defaultdict, Counter
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
# %load_ext pycodestyle_magic
# %flake8_off
# %flake8_on --ignore E225,E265,E501
```

0.3 a) Algunas operaciones sencillas [2 puntos]

Vamos a trabajar con una tabla del INE que contiene información sobre el paro en España. Abriendo el archivo, vemos algo así:

Si miramos una línea de esta tabla (salvo la primera, que es la cabecera), encontamos lo siguiente:

2020S000073

01/01/2020

18:48

AVDA. PIO XII

81

CHAMARTÍN

Atropello a persona

Despejado

Turismo

Conductor

DE 55 A 59 AÑOS

Hombre

14

Pero si inspeccionamos el archivo con un editor de texto, vemos que esa línea es como sigue:

```
2020S000073;01/01/2020;18:48;AVDA. PIO XII;81;CHAMARTÍN;Atropello a persona;Despejado;Turismo;C
```

Una cadena de caracteres. Hagamos en Python algunas operaciones básicas con los algunos de los datos anteriores y con la cadena en sí.

a.1 Redondeo de la hora

La primera operación consiste en redondear una hora, simplemente despreciando los minutos y dando lugar al entero correspondiente, entre 0 y 23.

[3]:
```python
# Esta celda debe ser completada por el estudiante
def redondeo_hora(hora):
    """Dada una cadena de texto que representa la hora con minutos,
    devuelve la un numero que representa la hora.
    """
    return hora.split(":")[0]
```

[4]:
```python
# Pruebas de funcionamiento:

print(redondeo_hora('12:48'))
```

```
12
```

Es bastante habitual hacer varias pruebas a la vez:

[5]:
```python
# Pruebas de funcionamiento:

for h in ['15:00', '23:15', '14:22', '9:34']:
    print(redondeo_hora(h))
```

```
15
23
14
9
```

a.2 Rangos de edad

Ahora, deseamos codificar los rangos de edad, asignando a cada rango descrito un intervalo de dos enteros. El ejemplo de funcionamiento te aclarará lo que se pide exactamente:

[6]:
```python
# Esta celda debe ser completada por el estudiante

def rango_edad(e):
    """Dada una cierta condicion de edad,
    se entrega el rango numerico que pertenece.
    """
    rangos = {
        'DE 0 A 5 AÑOS': (0, 5),
        'DE 6 A 9 AÑOS': (6, 9),
        'DE 10 A 14 AÑOS': (10, 14),
        'DE 15 A 17 AÑOS': (15, 17),
        'DE 18 A 20 AÑOS': (18, 20),
        'DE 21 A 24 AÑOS': (21, 24),
        'DE 25 A 29 AÑOS': (25, 29),
        'DE 30 A 34 AÑOS': (30, 34),
        'DE 35 A 39 AÑOS': (35, 39),
        'DE 40 A 44 AÑOS': (40, 44),
        'DE 45 A 49 AÑOS': (45, 49),
        'DE 50 A 54 AÑOS': (50, 54),
        'DE 55 A 59 AÑOS': (55, 59),
        'DE 60 A 64 AÑOS': (60, 64),
        'DE 65 A 69 AÑOS': (65, 69),
        'DE 70 A 74 AÑOS': (70, 74),
        'MAYOR DE 74 AÑOS': (75, 100)
    }
    return rangos.get(e, (-1, 1))
```

[7]:
```python
# Pruebas de funcionamiento:

for c in ['DE 25 A 29 AÑOS', 'DESCONOCIDA', 'MAYOR DE 74 AÑOS']:
    print(c, " -> ", rango_edad(c))
```

```
DE 25 A 29 AÑOS  ->  (25, 29)
DESCONOCIDA  ->  (-1, 1)
MAYOR DE 74 AÑOS  ->  (75, 100)
```

a.3 Lesividad: datos en blanco

El dato de lesividad viene codificado con un entero:

```
01 Atención en urgencias sin posterior ingreso. - LEVE
02 Ingreso inferior o igual a 24 horas - LEVE
...
77 Se desconoce
En blanco Sin asistencia sanitaria
```

Deseamos convertir este dato en un número entero. Cuando no se requiere asistencia sanitaria vamos a codificar esto con el entero 0 por homogeneidad. Cuando la lesividad no se conoce (un dato missing por ejemplo), también la consignaremos con un cero.

[8]:
```
# Esta celda debe ser completada por el estudiante
def lesividad(dato):
    """Evalua una cadena y si el largo de esta es superior a 0
    convierte la cadena a un numero entero,
    en caso contrario asigna el numero 0.
    """
    if len(dato) != 0:
        return int(dato)
    return 0
```

[9]:
```
# Pruebas de funcionamiento:

for c in ['01', '02', '14', '', '77']:
    print(c, " -> ", lesividad(c))
```

```
01  ->  1
02  ->  2
14  ->  14
  ->  0
77  ->  77
```

a.4 Operaciones con una línea de datos

Si ahora abres el archivo de datos con un editor de texto, podrás ver algo parecido a lo siguiente:

La línea novena es la que poníamos antes como ejemplo. Vista como una cadena de caracteres, podemos almacenarla en una variable para procesarla:

```
linea_9 = "2020S000073;01/01/2020;18:48;AVDA. PIO XII;81;CHAMARTÍN;Atropello a persona;Despeja
```

Y luego, podríamos hacer con ella algunas operaciones básicas, separando sus piezas (con el método `split`), extrayendo alguna que nos interese (accediendo a la componente adecuada con el corchete `[...]`, y estas piezas se pueden manejar con las funciones definicas antes, `redondeo_hora` y `rango_edad`.

En una primera versión, esto puede hacerse con una función que va imprimiendo las cosas, así:

```
def presentar_operaciones_basicas(cadena):
    print("La cadena de entrada: ")
    print(cadena)
    print()
    print("Piezas: ")
    ...
```

[10]:
```
# Esta celda debe ser completada por el estudiante
def presentar_operaciones_basicas(cadena):
    """R.
```

```
    """
    print("La cadena de entrada:")
    print(cadena, '\n')
    print("Piezas: ")
    cadena = cadena.split(";")
    print(cadena, '\n')
    print("Distrito")
    print(cadena[5], '\n')
    print("La hora, sin y con redondeo:")
    hora = cadena[2]
    redondeo = redondeo_hora(cadena[2])
    print(hora, redondeo, '\n', sep='\n')
    print("La edad, tal como viene y en su rango:")
    print(cadena[10], '\n', rango_edad(cadena[10]), '\n')
```

[11]:
```
# Ejemplo de funcionamiento:

linea_9 = "2020S000073;01/01/2020;18:48;AVDA. PIO XII;81;CHAMARTÍN;Atropello a␣
 ↪persona;Despejado;Turismo;Conductor;DE 55 A 59 AÑOS;Hombre;14;;"
presentar_operaciones_basicas(linea_9)
```

```
La cadena de entrada:
2020S000073;01/01/2020;18:48;AVDA. PIO XII;81;CHAMARTÍN;Atropello a
persona;Despejado;Turismo;Conductor;DE 55 A 59 AÑOS;Hombre;14;;

Piezas:
['2020S000073', '01/01/2020', '18:48', 'AVDA. PIO XII', '81', 'CHAMARTÍN',
'Atropello a persona', 'Despejado', 'Turismo', 'Conductor', 'DE 55 A 59 AÑOS',
'Hombre', '14', '', '']

Distrito
CHAMARTÍN

La hora, sin y con redondeo:
18:48
18


La edad, tal como viene y en su rango:
DE 55 A 59 AÑOS
 (55, 59)
```

Decíamos que, en una primera versión, esto puede hacerse con una función que va imprimiendo las cosas. Pero realmente, no es el estilo deseable. Preferimos una función que no escriba nada, que devuelva su resultado con `return`. Y de paso, que devuelva únicamente las piezas que nos interesan, por ejemplo: la hora (redondeada), el distrito, el estado meteorológico, el rango de edad y el nivel (entero) de lesividad del accidente.

(Lógicamente, según el objetivo que nos interese, podría ser necesario luego cargar unos campos u otros.)

[12]:
```
# Esta celda debe ser completada por el estudiante
def extraer_datos(data):
    """Permite leer una variable y despues de separar las cadenas
    segun el separador ;
    seleccionar los indices que entregan la
    informacion que se desea mostrar.
    """
    data = data.split(";")
    return [redondeo_hora(data[2]), data[5], data[7],
            rango_edad(data[10]), lesividad(data[12])]


extraer_datos(linea_9)
```

[12]:
```
['18', 'CHAMARTÍN', 'Despejado', (55, 59), 14]
```

[13]:
```
# Pruebas de funcionamiento:

print(len(linea_9.split(";")))
print(extraer_datos(linea_9))
```

```
15
['18', 'CHAMARTÍN', 'Despejado', (55, 59), 14]
```

0.4 b) Lectura de datos del archivo [2 puntos]

En este apartado te planteo diseñar tres funciones de lectura de datos.

b.1. Cabecera La primera función leerá la cabecera del archivo de datos, esto es, su primera línea, y la descompondrá en los rótulos correspondientes a cada columna. Además de abrir el archivo (preferiblemente con la instrucción `with open...`), bastará con un único `readline`.

[14]:
```
# Esta celda debe ser completada por el estudiante
def cargar_cabecera(archivo):
    """Permite leer un archivo de tipo csv, para despues limpiar
    la cadena de texto generada
    (;, saltos de linea) y retornar una lista que representa la cabecera del
    documento.
    """
    # Se agrega el encoding segun esta solucion https://tinyurl.com/2p82tns4
    with open(archivo, 'r', encoding="ISO-8859-1") as a:
        a = a.readline().replace("\n", "")
        return a.split(";")
```

[15]:
```
# Pruebas de funcionamiento:

cabecera = cargar_cabecera("2020_Accidentalidad.csv")
print(cabecera)
```

```
['Nº  EXPEDIENTE', 'FECHA', 'HORA', 'CALLE', 'NÚMERO', 'DISTRITO', 'TIPO
ACCIDENTE', 'ESTADO METEREOLÓGICO', 'TIPO VEHÍCULO', 'TIPO PERSONA', 'RANGO DE
EDAD', 'SEXO', 'LESIVIDAD*', '', '']
```

b.2 Lectura de algunas líneas del archivo

Ahora, nos interesa leer justamente los datos a partir de la cabecera, esto es algunas de las demás líneas. Una forma de saltarnos esa primera línea es usar la instrucción `next`. Pongamos que queremos leer desde la línea 17 hasta la 23. Podemos leer (sin procesar) $17 - 1$ líneas y luego, podemos leer y retener $23 - 17 + 1$ líneas.

[16]:
```
# Esta celda debe ser completada por el estudiante
def cargar_lineas(archivo, desde=1, hasta=10):
    """Funcion que lee un archivo segun la cantidad de lineas indicadas,
    omite la cabecera y en base a lo extraido, genera lineas con
    informacion depurada.
    """
    # Se agrega el encoding segun esta solucion https://tinyurl.com/2p82tns4
    with open(archivo, 'r', encoding="ISO-8859-1") as a:
        next(a)  # Omite la cabecera del archivo
        lines = a.readlines()
        lines = lines[desde-1 : hasta]
        return [extraer_datos(linea) for indice, linea in enumerate(lines)]
```

[17]:
```
lineas_lista = cargar_lineas("2020_Accidentalidad.csv", 1, 4)

for linea in lineas_lista:
    print(linea)

# Si no decimos qué líneas nos interesa, se cargarán las diez primeras.
# (Esto puede hacerse con dos parámetros por defecto.)

print()

lineas_lista = cargar_lineas("2020_Accidentalidad.csv")

for linea in lineas_lista:
    print(linea)
```

```
['23', 'RETIRO', 'Despejado', (25, 29), 0]
['22', 'MONCLOA-ARAVACA', 'Despejado', (21, 24), 6]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (45, 49), 14]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (25, 29), 7]
```

```
['23', 'RETIRO', 'Despejado', (25, 29), 0]
['22', 'MONCLOA-ARAVACA', 'Despejado', (21, 24), 6]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (45, 49), 14]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (25, 29), 7]
['19', 'CENTRO', 'Despejado', (-1, 1), 0]
['19', 'CARABANCHEL', 'Despejado', (-1, 1), 14]
['19', 'CARABANCHEL', 'Despejado', (21, 24), 2]
['18', 'CHAMARTÍN', 'Despejado', (55, 59), 14]
['18', 'CHAMARTÍN', 'Despejado', (18, 20), 7]
['18', 'ARGANZUELA', '', (55, 59), 14]
```

b.3 Lectura de todas las líneas del archivo

Lo normal es desear cargar **todos** los datos de un archivo, y no sólo unas pocas líneas, excluyendo la cabecera. Al igual que en la función anterior, te pido que el resultado se dé en una lista, donde cada elemento recoge la información de una línea del archivo de datos, salvo la cabecera, pero incluyendo ahora **todas** esas líneas, sin dar opción a cuáles nos interesa, aunque luego deseemos mostrar tan solo unas pocas. Véanse ambas pruebas de funcionamiento.

[18]:
```
# Esta celda debe ser completada por el estudiante
def cargar_datos(archivo):
    with open(archivo, 'r', encoding = "ISO-8859-1" ) as a: #Se agrega el␣
 ↪encoding segun esta solucion https://tinyurl.com/2p82tns4
        next(a) #omite la cabecera del archivo
        lines = a.readlines()
        lineas = []
        for indice, linea in enumerate(lines):
                lineas.append(extraer_datos(linea))
        return lineas
```

[29]:
```
# Pruebas de funcionamiento:

datos_lista = cargar_datos("2020_Accidentalidad.csv")

for linea in datos_lista[:50]:
    print(linea)
```

```
['23', 'RETIRO', 'Despejado', (25, 29), 0]
['22', 'MONCLOA-ARAVACA', 'Despejado', (21, 24), 6]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (45, 49), 14]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (25, 29), 7]
['19', 'CENTRO', 'Despejado', (-1, 1), 0]
['19', 'CARABANCHEL', 'Despejado', (-1, 1), 14]
['19', 'CARABANCHEL', 'Despejado', (21, 24), 2]
['18', 'CHAMARTÍN', 'Despejado', (55, 59), 14]
['18', 'CHAMARTÍN', 'Despejado', (18, 20), 7]
['18', 'ARGANZUELA', '', (55, 59), 14]
['18', 'ARGANZUELA', '', (35, 39), 14]
['16', 'HORTALEZA', 'Despejado', (40, 44), 0]
```

```
['16', 'HORTALEZA', 'Despejado', (30, 34), 0]
['16', 'HORTALEZA', 'Despejado', (-1, 1), 0]
['16', 'PUENTE DE VALLECAS', 'Despejado', (45, 49), 14]
['16', 'PUENTE DE VALLECAS', 'Despejado', (35, 39), 2]
['16', 'PUENTE DE VALLECAS', 'Despejado', (40, 44), 2]
['16', 'PUENTE DE VALLECAS', 'Despejado', (45, 49), 2]
['15', 'HORTALEZA', 'Despejado', (21, 24), 14]
['15', 'HORTALEZA', 'Despejado', (18, 20), 0]
['15', 'HORTALEZA', 'Despejado', (-1, 1), 0]
['15', 'HORTALEZA', 'Despejado', (-1, 1), 0]
['15', 'HORTALEZA', 'Despejado', (-1, 1), 0]
['15', 'CARABANCHEL', 'Despejado', (50, 54), 6]
['15', 'CARABANCHEL', 'Despejado', (-1, 1), 0]
['15', 'CARABANCHEL', 'Despejado', (-1, 1), 0]
['15', 'CARABANCHEL', 'Despejado', (-1, 1), 0]
['15', 'CARABANCHEL', 'Despejado', (-1, 1), 0]
['15', 'CARABANCHEL', 'Despejado', (-1, 1), 0]
['14', 'PUENTE DE VALLECAS', 'Despejado', (55, 59), 0]
['14', 'PUENTE DE VALLECAS', 'Despejado', (-1, 1), 0]
['14', 'HORTALEZA', 'Despejado', (35, 39), 1]
['14', 'PUENTE DE VALLECAS', 'Despejado', (40, 44), 14]
['14', 'PUENTE DE VALLECAS', 'Despejado', (50, 54), 1]
['14', 'PUENTE DE VALLECAS', 'Despejado', (30, 34), 14]
['14', 'PUENTE DE VALLECAS', 'Despejado', (60, 64), 14]
['14', 'PUENTE DE VALLECAS', 'Despejado', (45, 49), 14]
['14', 'PUENTE DE VALLECAS', 'Despejado', (45, 49), 14]
['14', 'SAN BLAS-CANILLEJAS', 'Despejado', (45, 49), 0]
['14', 'SAN BLAS-CANILLEJAS', 'Despejado', (45, 49), 0]
['14', 'SAN BLAS-CANILLEJAS', 'Despejado', (-1, 1), 0]
['14', 'SAN BLAS-CANILLEJAS', 'Despejado', (-1, 1), 0]
['14', 'SAN BLAS-CANILLEJAS', 'Despejado', (-1, 1), 0]
['13', 'PUENTE DE VALLECAS', '', (40, 44), 0]
['12', 'LATINA', 'Despejado', (18, 20), 14]
['12', 'RETIRO', 'Despejado', (65, 69), 7]
['12', 'RETIRO', 'Despejado', (35, 39), 7]
['12', 'FUENCARRAL-EL PARDO', 'Despejado', (45, 49), 0]
['12', 'FUENCARRAL-EL PARDO', 'Despejado', (18, 20), 0]
['12', 'FUENCARRAL-EL PARDO', 'Despejado', (15, 17), 0]
```

[20]:
```
# Pruebas de funcionamiento:

for linea in datos_lista[0:4]:
    print(linea)
```

```
['23', 'RETIRO', 'Despejado', (25, 29), 0]
['22', 'MONCLOA-ARAVACA', 'Despejado', (21, 24), 6]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (45, 49), 14]
['20', 'FUENCARRAL-EL PARDO', 'Despejado', (25, 29), 7]
```

0.5 c) Accidentalidad y mortalidad por edad [2 puntos]

c.1. Accidentalidad. Cómputo básico

Deseamos totalizar el número de accidentes de nuestra tabla por cada rango de edad. Para ello, te pido que uses un diccionario en el que la clave es el rango de edad y el valor, el total de accidentes para dicho rango de edad. Ahora, las posibilidades son dos:

1. Cada accidente actualiza el diccionario así: si ese rango de edad no está en el diccionario, se añade con un total de un accidente; si ya está, se añade una unidad más al tltal de accidentes de dicho rango de edad
2. Con un diccionario con el valor 0 por defecto.

[21]:
```
# Esta celda debe ser completada por el estudiante
# Un hint para este ejercicio se encuentra aqui: https://realpython.com/
 ↪python-defaultdict/
def totales(datos):
    cuenta_edades = defaultdict(int)
    for i in datos:
        rango_edades = i[3]
        cuenta_edades[rango_edades] += 1
    return dict(cuenta_edades)

totales(datos_lista)
```

[21]:
```
{(25, 29): 3437,
 (21, 24): 2226,
 (45, 49): 3084,
 (-1, 1): 3962,
 (55, 59): 2077,
 (18, 20): 978,
 (35, 39): 3332,
 (40, 44): 3399,
 (30, 34): 3362,
 (50, 54): 2547,
 (60, 64): 1272,
 (65, 69): 641,
 (15, 17): 250,
 (70, 74): 427,
 (75, 100): 657,
 (0, 5): 289,
 (6, 9): 175,
 (10, 14): 305}
```

[22]:
```
# Prueba de funcionamiento:

total_accidentes_por_edades = totales(datos_lista)
```

```
for k, e in total_accidentes_por_edades.items():
    print(k, e)
```

```
(25, 29) 3437
(21, 24) 2226
(45, 49) 3084
(-1, 1) 3962
(55, 59) 2077
(18, 20) 978
(35, 39) 3332
(40, 44) 3399
(30, 34) 3362
(50, 54) 2547
(60, 64) 1272
(65, 69) 641
(15, 17) 250
(70, 74) 427
(75, 100) 657
(0, 5) 289
(6, 9) 175
(10, 14) 305
```

c.2. Accidentalidad con mortalidad

Deseamos recopilar, para cada rango de edad, el total de accidentes registrados en nuestra tabla, junto con el número de dichos accidentes que han resultado ser mortales. El cociente (multiplicado por mil) nos dará la tasa de accidentes mortales por cada mil accidentes.

[23]:
```
# Esta celda debe ser completada por el estudiante
def totales_mortales(datos):
    cuenta_edades = defaultdict(int)
    a = totales(datos)
    comun = defaultdict(int)
    for i in datos:
        rango_edades = i[3]
        if i[4] == 4:
            cuenta_edades[rango_edades] += 1
        else:
            cuenta_edades[rango_edades] == 0


    dict_3 = {**a, **cuenta_edades}
    for key, value in dict_3.items():
        if key in a and key in cuenta_edades:
            dict_3[key] = (a[key], value)
        else:
            value == 0
    return list(dict_3.items())
```

```
totales_mortales(datos_lista)
```

[23]:
```
[((25, 29), (3437, 3)),
 ((21, 24), (2226, 2)),
 ((45, 49), (3084, 4)),
 ((-1, 1), (3962, 0)),
 ((55, 59), (2077, 1)),
 ((18, 20), (978, 0)),
 ((35, 39), (3332, 8)),
 ((40, 44), (3399, 6)),
 ((30, 34), (3362, 2)),
 ((50, 54), (2547, 1)),
 ((60, 64), (1272, 1)),
 ((65, 69), (641, 1)),
 ((15, 17), (250, 0)),
 ((70, 74), (427, 1)),
 ((75, 100), (657, 4)),
 ((0, 5), (289, 1)),
 ((6, 9), (175, 0)),
 ((10, 14), (305, 0))]
```

Observa que la función `totales_mortales` devuelve una lista, y no un diccionario.

[24]:
```
# Prueba de funcionamiento:

total_accidentes_y_muertes_por_edades = totales_mortales(datos_lista)

for edades, dos_totales in total_accidentes_y_muertes_por_edades:
    print(edades, dos_totales)

print()

# Total accidentes mortales / 1000 accidentes, por rangos de edad:

tasa_accidentes_mortales_por_mil = [(k, m*1000/n) for k, (n, m) in␣
 ↪total_accidentes_y_muertes_por_edades]

for k_tasa  in tasa_accidentes_mortales_por_mil:
    print(k_tasa)
```

```
(25, 29) (3437, 3)
(21, 24) (2226, 2)
(45, 49) (3084, 4)
(-1, 1) (3962, 0)
(55, 59) (2077, 1)
(18, 20) (978, 0)
(35, 39) (3332, 8)
```

```
(40, 44) (3399, 6)
(30, 34) (3362, 2)
(50, 54) (2547, 1)
(60, 64) (1272, 1)
(65, 69) (641, 1)
(15, 17) (250, 0)
(70, 74) (427, 1)
(75, 100) (657, 4)
(0, 5) (289, 1)
(6, 9) (175, 0)
(10, 14) (305, 0)

((25, 29), 0.8728542333430317)
((21, 24), 0.8984725965858041)
((45, 49), 1.297016861219196)
((-1, 1), 0.0)
((55, 59), 0.4814636494944632)
((18, 20), 0.0)
((35, 39), 2.4009603841536613)
((40, 44), 1.7652250661959399)
((30, 34), 0.594883997620464)
((50, 54), 0.39261876717707106)
((60, 64), 0.7861635220125787)
((65, 69), 1.5600624024960998)
((15, 17), 0.0)
((70, 74), 2.34192037470726)
((75, 100), 6.0882800608828)
((0, 5), 3.4602076124567476)
((6, 9), 0.0)
((10, 14), 0.0)
```

0.6 d) Algunas gráficas [1.5 puntos]

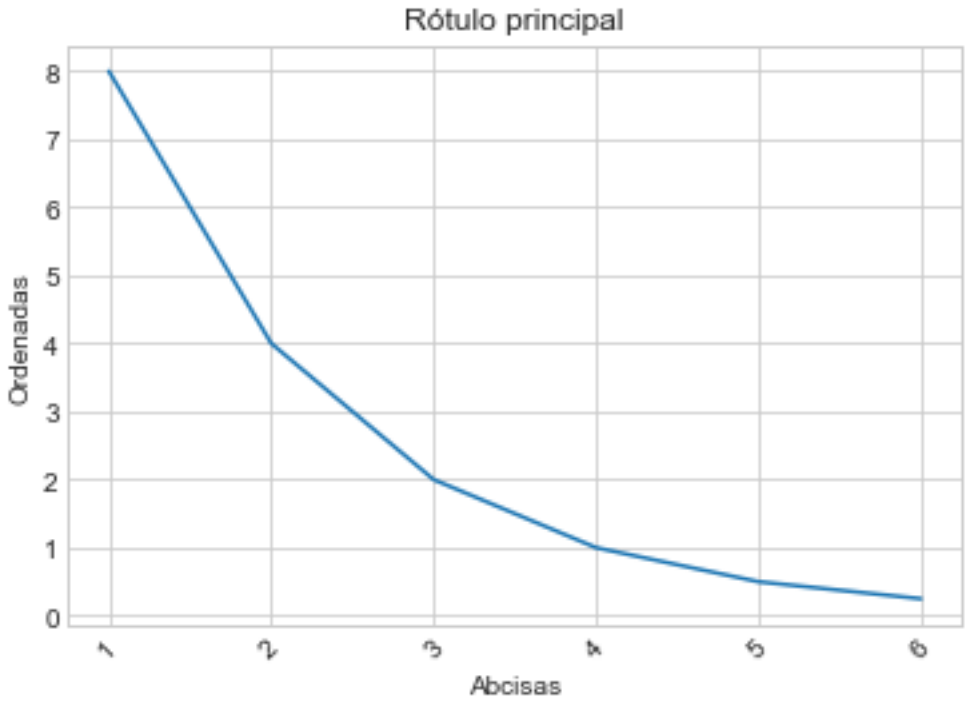
d.1 Un modelo de gráfica. Vamos a diseñar un modelo de gráfica sencillo que nos sirva para las siguientes representaciones. Tomará como parámetro una lista de pares (x, y), y opcionalmente los tres rótulos explicativos que necesitamos incluir. Además, queremos que las etiquetas de las abcisas aparezcan inclinadas, para poder luego mostrar intervalos de edad.

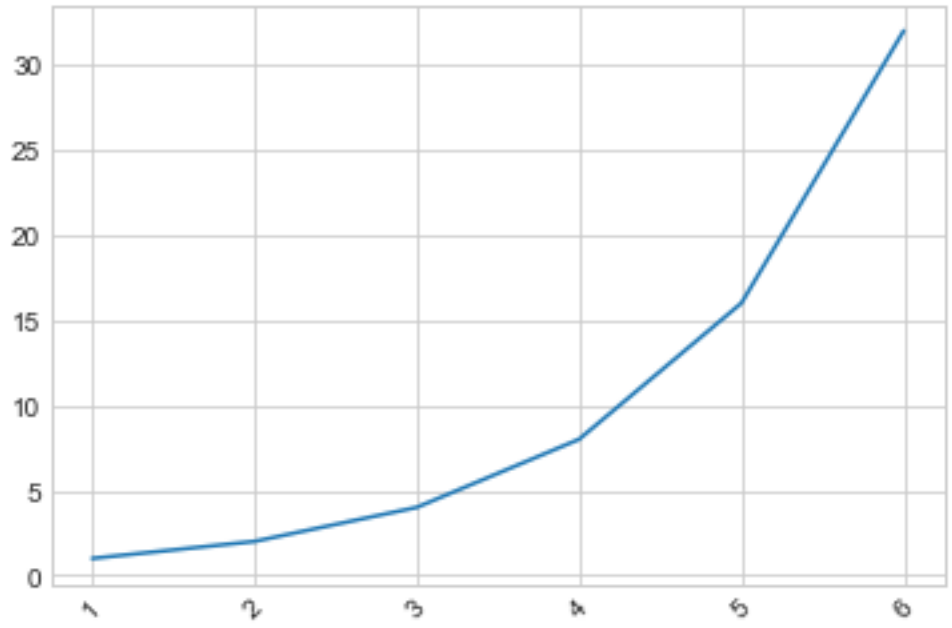
Las pruebas de funcionamiento te darán más información que las explicaciones que pueda yo dar aquí.

[25]:
```python
# Esta celda debe ser completada por el estudiante
def representar_xxx_yyy(datos, etiquetas = None):
    """Crea un grafico a partir de un conjunto de pares(x,y).

    Args:
      datos: Lista de tuplas de conjunto de pares que representa los ejes x e y
```

```
    etiquetas: Opcional, lista de caracteres que se utilizaran para etiquetar␣
↪el grafico y los ejes

  Returns:
    Grafico de linea que incluye grilla, si la etiqueta viene como parametro,␣
↪el grafico
    incluye las leyendas para los ejes x e y mas un titulo.
  """
  #datos = np.asarray(datos, dtype=object)
  plt.style.use('seaborn-whitegrid')#Para mostrar la grilla del grafico
  plt.plot(*zip(*datos)) # https://stackoverflow.com/questions/18458734/
↪how-do-i-plot-list-of-tuples-in-python
  plt.xticks(rotation = 45) # Rota las etiquetas en 45 grados
  if etiquetas:
      plt.xlabel(etiquetas[2])# Extrae la etiqueta del eje x
      plt.ylabel(etiquetas[1])# Extrae la etiqueta del eje y
      plt.title(etiquetas[0])
  plt.show()
```

[26]:
```
# Pruebas de funcionamiento:

representar_xxx_yyy([(1, 8), (2, 4), (3, 2), (4, 1), (5, 0.5), (6, 0.25)],␣
↪["Rótulo principal", "Ordenadas", "Abcisas"])

representar_xxx_yyy([(1, 1), (2, 2), (3, 4), (4, 8), (5, 16), (6, 32)])
```

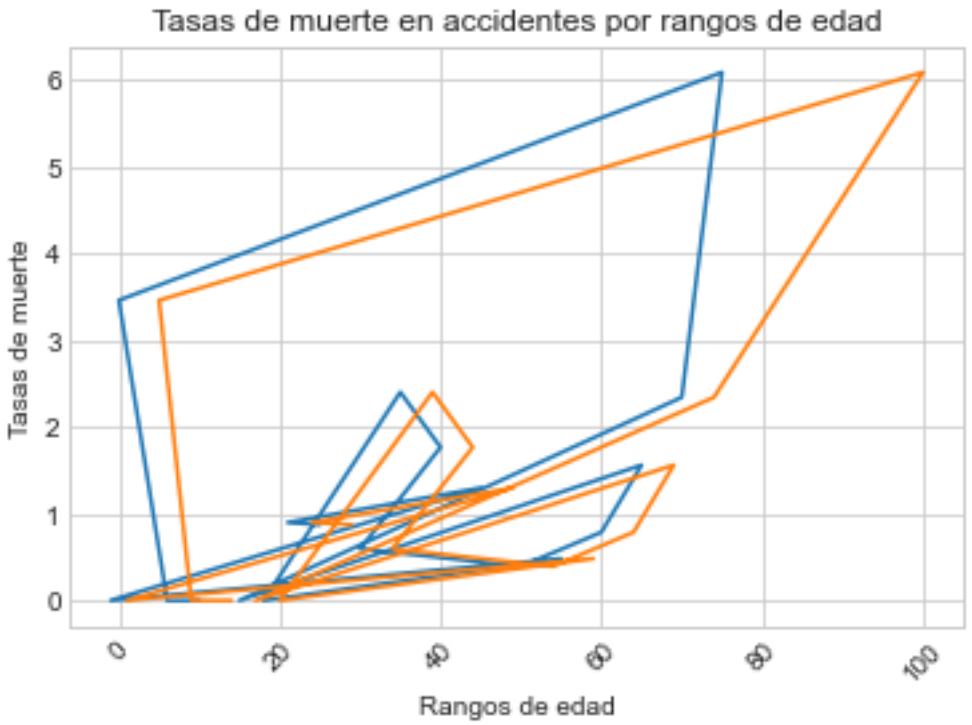




d.2. Tasas de muerte por edades

Queremos aplicar nuestro modelo de gráfica a la representación de las tasas de accidentes mortales por edad, que hemos calculado un poco antes. Pero obtenemos una gráfica poco adecuada, porque las edades (las abcisas) están en un orden arbitrario.

[27]:
```
# Intento de representación:

rotulos = "Tasas de muerte en accidentes por rangos de edad", "Tasas de␣
 ↪muerte", "Rangos de edad"
representar_xxx_yyy(tasa_accidentes_mortales_por_mil, rotulos)
```



Para remediar esto se ha de reordenar la lista de pares absisa-ordenada, atendiendo a las abcisas. También, el elemento de abcisa (-1, -1) se ha de suprimir. Esto es lo que te pido.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

[ ]:
```
# Prueba de funcionamiento:

representar_xxx_yyy(tasa_accidentes_mortales_por_mil, rotulos)
```

d.3. Tasas de muerte por rangos horarios

De forma similar a lo resuelto en los apartados anteriores, deseamos preparar los datos y un gráfico con la tasa de muerte por rangos horarios. En lugar de tratar los rangos por horas enteras (las 4 representa el intervalo entre las 4:00 y las 4:59), deseamos representar de dos en dos horas (las 4 representa el intervalo entre las 4:00 y las 5:59, las 6, entre las 6:00 y las 7:59, etc.)

Observa que se necesitan dos funciones, una para recopilar los datos, calcular las tasas, dar una lista ordenada, etc., y otra para preparar las abcisas, cadenas de caracteres con las horas de dos en dos, junto con sus tasas respectivas.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

[ ]:
```
# Prueba de funcionamiento:
```

```
tasas_accidentes_y_muertes_por_horario =␣
↪totales_mortales_por_horario(datos_lista)

print(tasas_accidentes_y_muertes_por_horario)

print()

datos_para_grafica = emparejar_abcisas(tasas_accidentes_y_muertes_por_horario)

print(datos_para_grafica)

rotulos = "Tasas de muerte en accidentes por rangos horarios", "Tasas de␣
↪muerte", "Rangos horarios"
representar_xxx_yyy(datos_para_grafica, rotulos)
```

0.7 e) Operaciones con dataframes [1.5 puntos]

En este apartado, vamos a trabajar con tablas de la librería `pandas`, llamadas `dataframes`.

e1. Carga del dataframe. La primera operación que necesitamos es cargar el archivo de datos en una tabla, como se ve en el siguiente ejemplo.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

[ ]:
```
tabla_pre = cargar_dataframe_v0("2020_Accidentalidad.csv")
print(tabla_pre)
```

e2. Carga del dataframe, codificando rangos de edad y lesividad. Ahora, queremos modificar esta lectura para que los rangos de edad se conviertan en el intervalo correspondiente. Además, vemos que el nivel de lesividad se ha leído directamente como un real, y las cadenas en blanco se han traducido a `NaN` (*Not a Number*). Queremos ponerlo como un entero, consistente en un 1 cuando hay lesividad. Cuando no se conoce la lesividad, o no hay lesividad (casos codificados con un 0, un 77, un 14), anotamos un 0 en la tabla.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

[ ]:
```
tabla = cargar_dataframe("2020_Accidentalidad.csv")
tabla
```

e3. Tabla de número de accidentes por rangos de edad

Nos interesa quedarnos únicamente con dos columnas: el rango de edad y el número de accidentes, formando una tabla nueva. Esta tabla debe mostrarse en orden ascendente de rango de edad.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

Esta tabla contiene el rango de edad $(-1, -1)$, que no nos interesa. Por eso preferimos descartar esta fila.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

e4. Accidentes con consecuencias médicas.

Queremos totalizar ahora los accidentes que requieren algún tipo de atención sanitaria o con resultado de muerte por cada rango de edad.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

e5. Unión de dos tablas.

Deseamos ahora combinar las dos tablas generadas, usando la columna "Edad" como pivote, al estilo de la operación `inner join` de SQL en el mundo de las bases de datos.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

e6. Proporción de accidentes con lesiones.

Deseamos ahora ver las cifras de lesiones en términos relativos, esto es, como el porcentaje proporcióde accidentes en que se producen lesiones.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

e6. Gráfico.

Finalmente, deseamos presentar la proporción de accidentes con lesiones por edades, por si al ver esto pudiéramos extraer alguna conclusión útil.

[ ]:
```
# Esta celda debe ser completada por el estudiante
```

0.8 f) Un cálculo masivo con map-reduce [0.5 puntos]

En este apartado se ha de realizar un programa aparte que calcule, para cada rango de edad, un par de enteros con los totales de podría activarse así desde la consola:

```
C:\...> python total_accs_edad.py -q 2020_Accidentalidad.csv
```

El programa funcionará necesariamente con la técnica map-reduce, que podemos poner en juego con la librería `mrjob`.

El funcionamiento del mismo se puede activar también desde aquí:

[ ]:
```
# Hagamos una llamada al programa de consola desde aquí:

! python total_accs_edad.py -q 2020_Accidentalidad.csv
```

[ ]:
```
# Para que el resultado se almacene en un archivo:

! python total_accs_edad.py -q 2020_Accidentalidad.csv  >␣
↪accidentalidad_y_mortalidad_por_edades.txt
```

Para que pueda yo ver tu programa cómodamente desde aquí, también se puede mostrar con un comando de la consola, anteponiendo el símbolo `!`. Observaciones:

- La instrucción siguiente está comentada para ocultar una solución mía. Tú debes suprimir el símbolo `#` del comentario para mostrar tu solución aquí.
- Desde mac o linux, se ha de usar el comando `cat`, en vez de `type`.

[ ]:
```
! type total_accs_edad.py
```

0.9 g) Un apartado libre [0.5 puntos]

Dejo este apartado a tu voluntad. Inventa tú mismo el enunciado y resuélvelo, mostrando algún aspecto de programación en Python no contemplado o alguna técnica o librería que no has puesto en juego en los apartados anteriores, relacionado con el análisis de datos y con este proyecto. He aquí dos ejemplos posibles:

- Me he quedado un poco insatisfecho con el uso de pandas, un poco escaso: este apartado adicional podría usar dicha librería poniendo en juego algunas operaciones que no hemos visto.
- Tampoco me gusta mucho el acabado de las figuras: la librería Plotly puede ser quizá permitirte trazar figuras más profesionales, y una posibilidad sencilla es quizá importar los datos del archivo creado por el programa de map-reduce y representarlos gráficamente.

Estos ejemplos pueden servirte como pista, pero que no te limiten. Hay muchas otras posibilidades: geopandas, web scraping, etc.

En la evaluación, si este apartado está bien o muy bien, anota un 0.4. El 0,5 lo reservaremos para las situaciones en que se presente algo brillante, con alguna idea original o alguna técnica novedosa o complejidad especial o algún gráfico vistoso. Especialmente quien opta a un 9,5 o más, debe esmerarse en plantear este apartado a la altura de esa calificación.

[ ]:
```
# Este apartado debe ser completado por el estudiante
```

[ ]:
```
# Pruebas de funcionamiento:
```

0.9.1 Datos personales

- **Apellidos:** Yanez Mendoza
- **Nombre:** Ivonne Valeska
- **Email:** ivonne@imendoza.io

- **Fecha:** 17 de mayo de 2022

0.9.2 Ficha de autoevaluación

Comentarios sobre la autoevaluación:

Aquí vienen comentarios del estudiante. Lo siguiente es un ejemplo posible obviamente … elimina este párrafo y redacta el tuyo propio, en azul.

Apartado	Calificación	Comentario
a)	2.0 / 2.0	Completamente resuelto
b)	0.0 / 2.0	No lo he conseguido
c)	0.25 / 2.0	Sólo he conseguido una parte mínima
d)	0.25 / 1.5	Sólo he conseguido una parte mínima
e)	0.0 / 1.5	No lo he conseguido
f)	0.5 / 0.5	No lo he conseguido
g)	0.0 / 0.5	No he logrado el correcto funcionamiento
Total	**2.5** / 10.0	**Suspenso**

0.9.3 Ayuda recibida y fuentes utilizadas

`https://stackoverflow.com/questions/60208/replacements-for-switch-statement-in-python`

… comentarios del estudiante … Pon tú este párrafo con tus propias observaciones. Elimina este párrafo en verde.

0.9.4 Comentario adicional

… Este apartado es optativo. Si lo completas, ponlo en azul; si no, suprímelo con su título.

[ ]:
```
# Esta celda se ha de respetar: está aquí para comprobar el funcionamiento de␣
↪algunas funciones por parte de tu profesor
```

[ ]:
```

```

[ ]:
```

```