accidentes_en_madrid_enunciados

May 17, 2022

Estadísticas de Accidentes en Madrid

Ivonne Yañez 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 un archivo de tipo CSV el cual contiene datos asociados a día, hora, distrito, tipo de accidente, sexo y grado de lesividad, entre otros campos, el archivo contiene información que no serán utilizados o que se deben ordenar y limpiar para realizar cálculos posteriores o su presentación como gráficos y tablas. Este proyecto pretende comenzar con Python en estado "puro" para después ir integrando librerías como pandas, NumPy o MrJob, con esto se busca extraer cierta información relevante, manipularla y entregar cálculos y resultados como tablas, tablas pivote y gráficos, terminando en el ejercicio libre que es donde se puede revisar por iniciativa propia que más cálculos o inferencias se pueden efectuar a partir de este set de datos. Ha sido interesante abordar un problema en común, empleando diferentes técnicas y librerías para perseguir el objetivo común de analizar y presentar la información de una forma interesante y elegante, tomando nota de sus ventajas y desventajas a la hora de elegir por cuál técnica o librería se desea utilizar.

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
from pylab import plot
import collections
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from numpy import array
import pandas as pd
from mrjob.job import MRJob
```

El próximo apartado es una librería que ayuda a comprobar que el estilo del código sea lo más pitónico posible. Fuente: https://github.com/mattijn/pycodestyle_magic

[2]:
```
# %load_ext pycodestyle_magic
# %flake8_on
# %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

0.4

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;(
```

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]:
```
#Esta celda debe ser completada por el estudiante
def redondeo_hora(hora):
    """
    Toma un formato hh:mm y devuelve solo con lo que
    corresponde a la hora.
```

```
    Parameters:
    -----------
    area: string
        La hora representada en formato hh:mm

    Precondition:
    -------------
    hora == string

    Returns:
    --------
    string
        el primer caracter del string ya separado

    Example:
    --------
    >>> redondeo_hora('12:48')
    12
    """
    return hora.split(":")[0]
```

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

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

```
12
```

Es bastante habitual hacer varias pruebas a la vez:

[5]:
```
# 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:

IYM: En esta funcion he llegado a dos respuestas posibles, la primera la dejo aqui comentada y es la que utilizo en el apartado con la libreria MrJob.

```
def rango_edad(edad):
"""Dado un string que representa una edad
calcula su rango etario correspondiente.
```

```
    Args:
      edad: string que contiene descripcion de edades.

    Returns:
      Tupla numerica con las edades en numeros enteros.
    """
    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(edad, (-1, -1))
```

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

def rango_edad(edad):
    """
    Entrega un rango de edad a partir de una lista de cadenas de texto.

    Parameters:
    -----------
    edad: lista
        Los strings que contiene los rangos de edades a evaluar.

    Precondition:
    -------------
    edad == string

    Returns:
    --------
    tupla
        La cual epresenta en números, las edades evaluadas.
```

```
    Example:
    --------
    >>> rango_edad(['MAYOR DE 74 AÑOS'])
    (75, 100)
    """

    if edad != 'DESCONOCIDA':
        if edad == 'MAYOR DE 74 AÑOS':
            mayor = 75, 100
            return mayor
        else:
            edades = tuple([int(s) for s in edad.split() if s.isdigit()])
            return edades
    desconocido = -1, -1
    return desconocido


rango_edad('DE 25 A 29 AÑOS')
```

[6]: (25, 29)

[7]:
```
# 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(codigo):
    """
    Entrega una cadena que representa un código de
```

```
    accidente convertido a entero.

    Parameters:
    -----------
    codigo:
        string que debe ser convertido a entero

    Precondition:
    -------------
    codigo > 0

    Returns:
    --------
    int
        entero que representa un codigo de accidentalidad

    Example:
    --------
    >>> lesividad("01")
    1
    """
    if len(codigo) != 0:
        return int(codigo)
    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(texto):
    """
    Extrae información seleccionada de una cadena de texto
    separada por puntos y comas.

    Parameters:
    -----------
    texto: string
        Cadena de texto de la cual se extraerá solo ciertos elementos

    Precondition:
    -------------
    len(texto) > 0

    Returns:
    --------
    string
        Imprime solo partes seleccionadas de la cadena con el formato adecuado

    Example:
    --------
    >>> presentar_operaciones_basicas("2020S000073;01/01/2020;18:48;AVDA.
    PIO XII;81;CHAMARTÍN;Atropello a persona;Despejado;Turismo;
    Conductor;DE 55 A 59 AÑOS;Hombre;14;;")

    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)
    """
    print("La cadena de entrada:")
    print(texto, '\n')
    print("Piezas: ")
    texto = texto.split(";")  # Convierte el texto en una lista separada
    print(texto, '\n')
    print("Distrito")
    print(texto[5], '\n')
    print("La hora, sin y con redondeo:")
    hora = texto[2]
    redondeo = redondeo_hora(texto[2])
    print(hora, redondeo, '\n', sep='\n')
    print("La edad, tal como viene y en su rango:")
    print(texto[10], '\n', rango_edad(texto[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]:
```python
# Esta celda debe ser completada por el estudiante
def extraer_datos(cadena):
    """
    De una cadena de texto extrae información con el formato correspondiente
    Los formatos de salida provienen de las funciones redondeo_hora(),
    rango_edad() y lesividad()

    Parameters:
    -----------
    cadena: string
        Cadena de texto de la cual se extraerá solo ciertos elementos y
        se les dará formato.

    Precondition:
    -------------
    len(cadena) > 0

    Returns:
    --------
    list
        Imprime solo partes seleccionadas de la cadena con el formato adecuado

    Example:
    --------
    >>> extraer_datos(linea_9)
    ['18', 'CHAMARTÍN', 'Despejado', (55, 59), 14]
    """
    cadena = cadena.split(";")

    return [redondeo_hora(cadena[2]), cadena[5], cadena[7],
            rango_edad(cadena[10]), lesividad(cadena[12])]
```

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

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

```
15
```

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

0.5 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]:
```python
# Esta celda debe ser completada por el estudiante
def cargar_cabecera(archivo):
    """
    De un archivo csv extrae la cabecera que contiene los nombres
    de las columnas.

    Parameters:
    -----------
    archivo: nombre archivo csv
        Archivo separado por puntos y comas del cual se extraerá la cabecera

    Precondition:
    -------------
    archivo == nombre_archivo.csv

    Returns:
    --------
    list
        Contiene la cabecera del archivo csv

    Example:
    --------
    >>> cargar_cabecera("2020_Accidentalidad.csv")
    ['Nº  EXPEDIENTE', 'FECHA', 'HORA', 'CALLE', 'NÚMERO', 'DISTRITO',
    'TIPO ACCIDENTE','ESTADO METEREOLÓGICO', 'TIPO VEHÍCULO', 'TIPO PERSONA',
    'RANGO DE EDAD', 'SEXO', 'LESIVIDAD*', '', '']
    """
    with open(archivo, 'r', encoding="ISO-8859-1") as a:
        a = a.readline().replace("\n", "")
        return a.split(";")
```

[15]:
```python
# 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):
    """
    De un archivo csv extrae cierto número de líneas sin considerar la cabecera
    del archivo y extrae datos con el formato adecuado con la función
    extraer_datos().

    Parameters:
    -----------
    file: archivo csv
        Archivo separado por puntos y comas del cual se extraerán líneas

    Precondition:
    -------------
    archivo == file.csv

    Returns:
    --------
    list
        Contiene las lineas leidas del archivo

    Example:
    --------
    >>> cargar_lineas("2020_Accidentalidad.csv")
    ['23', 'RETIRO', 'Despejado', (25, 29), 0]
    """
    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(file):
    """
    De un archivo csv extrae todas las lineas del archivo sin incluir cabecera
    y entrega un formato adecuado segun la función extraer_datos()

    Parameters:
    -----------
    file: nombre de archivo csv
        Archivo separado por puntos y comas del cual se extraerán líneas

    Precondition:
    -------------
    file == file.csv

    Returns:
    --------
    list
```

```
        Contiene todas las líneas leídas del archivo

    Example:
    --------
    >>> cargar_datos("2020_Accidentalidad.csv") solo muestra
    ['23', 'RETIRO', 'Despejado', (25, 29), 0]
    """
    with open(file, 'r', encoding="ISO-8859-1") as a:
        next(a)  # Omite la cabecera del archivo
        lines = a.readlines()
        lineas = []
        for indice, linea in enumerate(lines):
            lineas.append(extraer_datos(linea))
        return lineas
```

En esta prueba de funcionamiento he limitado las salidas para los 10 primeros elementos y los 10 ultimos y asi no generar un pdf con todas las lineas del archivo.

[19]:
```
# Pruebas de funcionamiento:

datos_lista = cargar_datos("2020_Accidentalidad.csv")

for linea in datos_lista[:10] + datos_lista[-10:]:
    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]
['01', 'USERA', '', (35, 39), 0]
['01', 'USERA', '', (21, 24), 0]
['01', 'USERA', '', (25, 29), 0]
['00', 'RETIRO', '', (25, 29), 0]
['00', 'CHAMARTÍN', 'Despejado', (25, 29), 0]
['00', 'CHAMARTÍN', 'Despejado', (35, 39), 0]
['00', 'CHAMARTÍN', 'Despejado', (35, 39), 0]
['00', 'CHAMARTÍN', 'Despejado', (35, 39), 0]
['00', 'CHAMARTÍN', 'Despejado', (35, 39), 0]
['00', 'CHAMARTÍN', 'Despejado', (35, 39), 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]
```

0.6 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
def totales(lista):
    """
    Dada una lista que contiene rangos de edades y accidentes,
    se deben agrupar y contar los accidentes para cada grupo etario.

    Parameters:
    -----------
    lista: list
        Lista con datos extraídos de los accidentes en Madrid

    Precondition:
    -------------
    lista == list

    Returns:
    --------
    dict
        Por cada rango de edad, la cuenta de accidentes

    Example:
    --------
    primeras lineas de muestra:
    >>> totales(datos_lista[:3])
    {(25, 29): 3437,
```

```
     (21, 24): 2226,
     (45, 49): 3084}
    """
    cuenta_edades = defaultdict(int)
    # Con *rest descarto las columnas anteriores
    for index, (*rest, edades, accidentes) in enumerate(lista):
        cuenta_edades[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]:
```python
# Esta celda debe ser completada por el estudiante
def totales_mortales(datos):
    """
    Dada una lista que contiene rangos de edades y accidentes,
    se deben contar los accidentes totales y los accidentes con lesividad 4
    para cada grupo etario.

    Parameters:
    -----------
    datos: list
        Lista con datos extraídos y con formato de los accidentes en Madrid

    Precondition:
    -------------
    datos == list

    Returns:
    --------
    list
        Lista de tuplas con los rangos de edad más la cuenta de accidentes
        y cuenta de accidentes con resultado de muerte

    Example:
    --------
    Solo las primeras salidas de muestra:
    >>> totales_mortales(datos_lista)
    [((25, 29), (3437, 3)),
     ((21, 24), (2226, 2)),
     ((45, 49), (3084, 4))]
    """
    cuenta_accidentes = defaultdict(int)
    total_accidentes = totales(datos)
```

```
    for index, (*rest, edades, accidentes) in enumerate(datos):
        if accidentes == 4:
            cuenta_accidentes[edades] += 1
        else:
            cuenta_accidentes[edades] == 0

    # Union de los diccionarios generados en esta funcion y la funcion totales()
    final = {**total_accidentes, **cuenta_accidentes}
    for key, value in final.items():
        if key in total_accidentes and key in cuenta_accidentes:
            final[key] = (total_accidentes[key], value)
        else:
            value == 0
    return list(final.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)
```

0.7 d) Algunas gráficas [1.5 puntos]

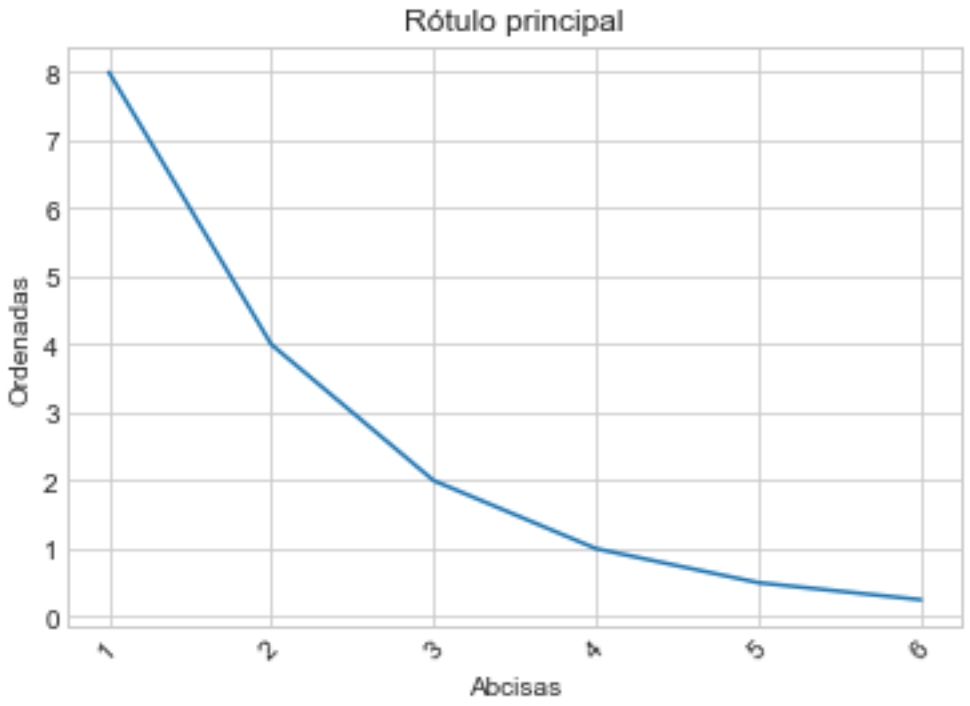
d.1 Un modelo de gráfica. Vamos a diseñar un modelo de gráfica sencillo que nos sirva para las siguientes representaciones. Tomará como parámetro una lista de pares (x, y), y opcionalmente los tres rótulos explicativos que necesitamos incluir. Además, queremos que las etiquetas de las abcisas aparezcan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

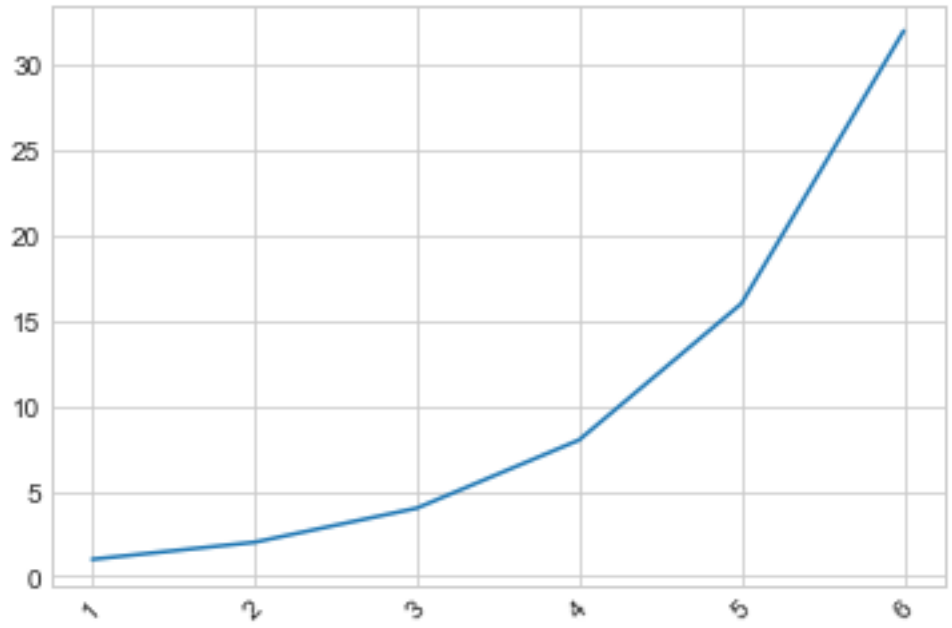
Las pruebas de funcionamiento te darán más información que las explicaciones que pueda yo dar aquí.

[25]:
```
# Esta celda debe ser completada por el estudiante
def representar_xxx_yyy(datos, etiquetas=None):
    """
    Dada una lista de pares y una lista con rótulos (strings)
    se genera el gráfico apropiado incluyendo las abcisas inclinadas
    para mejor lectura.

    Parameters:
    -----------
    datos: list
    etiquetas: list, opcional
        Dos listas, la primera contiene los ejes x e y
        la segunda los rótulos del gráfico

    Precondition:
    -------------
    datos == list

    Returns:
    --------
    list
        Lista de tuplas con los rangos de edad más la cuenta de accidentes y
        cuenta de accidentes con resultado de muerte

    Example:
    --------
    >>> representar_xxx_yyy([(1, 1), (2, 2), (3, 4), (4, 8),
    (5, 16), (6, 32)])
    genera grafico sin rotulos

    >>> representar_xxx_yyy([(1, 8), (2, 4), (3, 2), (4, 1),
    (5, 0.5), (6, 0.25)],
    ["Rótulo principal", "Ordenadas", "Abcisas"])
    genera grafico con rotulos
    """
    plt.style.use('seaborn-whitegrid')  # Para mostrar la grilla del grafico
    eje_x = sorted(datos)[1:]
    eje_x = [str(i[0]) for i in datos]  # Transforma la tupla numerica a string
```

```
    eje_y = [i[1] for i in datos]
    plot(eje_x, eje_y, color='tab:blue')
    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.

[28]:
```
# Esta celda debe ser completada por el estudiante
tasa_accidentes_mortales_por_mil = sorted(tasa_accidentes_mortales_por_mil)[1:]
tasa_accidentes_mortales_por_mil
```

[28]:
```
[((0, 5), 3.4602076124567476),
 ((6, 9), 0.0),
 ((10, 14), 0.0),
 ((15, 17), 0.0),
 ((18, 20), 0.0),
 ((21, 24), 0.8984725965858041),
 ((25, 29), 0.8728542333430317),
 ((30, 34), 0.594883997620464),
 ((35, 39), 2.4009603841536613),
 ((40, 44), 1.7652250661959399),
 ((45, 49), 1.297016861219196),
 ((50, 54), 0.39261876717707106),
 ((55, 59), 0.4814636494944632),
 ((60, 64), 0.7861635220125787),
 ((65, 69), 1.5600624024960998),
```

```
 ((70, 74), 2.34192037470726),
 ((75, 100), 6.0882800608828)]
```

[29]:
```
# 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.

[30]:
```
# Esta celda debe ser completada por el estudiante
def totales_mortales_por_horario(datos):
    """
    Por cada hora cuenta el número de accidentes en total
    y el número de los accidentes con resultado de muerte.
```

```
    Una vez obtenido esto, por cada horario 0, 1, 2, 3
    se deben reorganizar segun la hora actual mas la hora siguiente
    0, contiene datos de la hora 0 y 1
    2, contiene datos de la hora 2 y 3
    4, contiene datos de la hora 4, 5
    y en sucesivo.
    Y por último calcular y retornar la tasa de mortalidad.

    Parameters:
    -----------
    datos: list
        Lista que contiene los datos de accidentes en Madrid
        para esta función se trabaja con las columnas horario
        y accidente.

    Precondition:
    -------------
    datos == list

    Returns:
    --------
    list
        Lista de tuplas con los horarios re ordenado en pares más
        el cálculo de la tasa de mortalidad.

    Example:
    --------
    Solo las primeras salidas de muestra:
    >>>totales_mortales_por_horario(datos_lista)
    [(0, 1.9230769230769231),
     (2, 4.178272980501393),
     (4, 1.949317738791423),...
    """

    d1 = defaultdict(int)
    d2 = defaultdict(int)
    union = [d1, d2]
    total_muertes = {}

    # Accidentes por horario
    for index, (horario, *res, accidente) in enumerate(datos):
        horario = int(horario)
        d1[horario] += 1

    # Accidentes por horario con lesividad 4
    for index, (horario, *res, accidente) in enumerate(datos):
        horario = int(horario)
```

```
        if accidente == 4:
            d2[horario] += 1
        else:
            d2[accidente] == 0

    # Union de los diccionarios
    for k in d1.keys():
        total_muertes[k] = tuple(total_muertes[k] for total_muertes in union)

    # Orden del diccionario
    base_calculos = sorted(total_muertes.items())

    # Sumar los accidentes y muertes del indice actual y del siguiente
    horas = iter(base_calculos)
    final = []
    for hora in horas:
        hora_siguiente = next(horas)
        sum_accidentes = hora[1][0] + hora_siguiente[1][0]
        sum_muertes = hora[1][1] + hora_siguiente[1][1]
        final.append((hora[0], (sum_accidentes, sum_muertes)))

    # Calculo de la tasa de mortalidad
    tasa_mortalidad = [(k, m*1000/n) for k, (n, m) in final]
    return tasa_mortalidad


def emparejar_abcisas(base_calculos):
    """
    Convierte la tasa de mortalidad que contiene un número que
    representa un horario a una tupla con su correspondiente par.

    Parameters:
    -----------
    base_calculos:
        Lista de tuplas con la tasa de mortalidad para cada horario

    Precondition:
    -------------
    base_calculos == list

    Returns:
    --------
    list
        Lista de tuplas con los horarios re ordenado en pares mas
        el calculo de la tasa de mortalidad.

    Example:
```

```
    --------
    >>>emparejar_abcisas(totales_mortales_por_horario(datos_lista))
    [('(0, 2)', 1.9230769230769231),
    ('(2, 4)', 4.178272980501393),
    ('(4, 6)', 1.949317738791423)]..
    solo se muestran las primeras salidas
    """
    return [(str((hora, hora + 2)), calculo) for index, (hora, calculo)
            in enumerate(base_calculos)]


print(totales_mortales_por_horario(datos_lista), "\n")
print(emparejar_abcisas(totales_mortales_por_horario(datos_lista)))
```

```
[(0, 1.9230769230769231), (2, 4.178272980501393), (4, 1.949317738791423), (6,
0.8635578583765112), (8, 1.1415525114155252), (10, 1.5337423312883436), (12,
0.8234971177600878), (14, 1.112099644128114), (16, 0.5351886540005352), (18,
0.4287245444801715), (20, 1.1999040076793857), (22, 1.187178472497032)]

[('(0, 2)', 1.9230769230769231), ('(2, 4)', 4.178272980501393), ('(4, 6)',
1.949317738791423), ('(6, 8)', 0.8635578583765112), ('(8, 10)',
1.1415525114155252), ('(10, 12)', 1.5337423312883436), ('(12, 14)',
0.8234971177600878), ('(14, 16)', 1.112099644128114), ('(16, 18)',
0.5351886540005352), ('(18, 20)', 0.4287245444801715), ('(20, 22)',
1.1999040076793857), ('(22, 24)', 1.187178472497032)]
```

[31]:
```
# 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, 1.9230769230769231), (2, 4.178272980501393), (4, 1.949317738791423), (6,
0.8635578583765112), (8, 1.1415525114155252), (10, 1.5337423312883436), (12,
0.8234971177600878), (14, 1.112099644128114), (16, 0.5351886540005352), (18,
0.4287245444801715), (20, 1.1999040076793857), (22, 1.187178472497032)]
```

```
[('(0, 2)', 1.9230769230769231), ('(2, 4)', 4.178272980501393), ('(4, 6)',
1.949317738791423), ('(6, 8)', 0.8635578583765112), ('(8, 10)',
1.1415525114155252), ('(10, 12)', 1.5337423312883436), ('(12, 14)',
0.8234971177600878), ('(14, 16)', 1.112099644128114), ('(16, 18)',
0.5351886540005352), ('(18, 20)', 0.4287245444801715), ('(20, 22)',
1.1999040076793857), ('(22, 24)', 1.187178472497032)]
```

0.8 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.

[32]:
```python
# Esta celda debe ser completada por el estudiante
def cargar_dataframe_v0(archivo):
    """
    Convierte un archivo csv en un data frame que contiene
    solo columnas seleccionadas del csv.

    Parameters:
    -----------
```

```
    archivo:
        csv file

    Precondition:
    -------------
    archivo == csv file

    Returns:
    --------
    pd.DataFrame
    Dataframe con las columnas detalladas a continuación

        ==========    ====================================================
        HORA          object que representa la hora
        DISTRITO      object que representa el distrito
        RANGO DE EDAD object que indica entre que edades se clasifican
        LESIVIDAD*    float64 que asigna un codigo por cada accidente
        ==========    ====================================================

    Example:
    --------
    >>>cargar_dataframe_v0("2020_Accidentalidad.csv").head(2)
        HORA             DISTRITO    RANGO DE EDAD  LESIVIDAD*
    0  23:15               RETIRO  DE 25 A 29 AÑOS         NaN
    1  22:35      MONCLOA-ARAVACA  DE 21 A 24 AÑOS         6.0
    """
    base = pd.read_csv(archivo, encoding='unicode_escape',
                       on_bad_lines='skip', delimiter=";")
    base = base[['HORA', 'DISTRITO', 'RANGO DE EDAD', 'LESIVIDAD*']]
    return base
```

[33]:
```
tabla_pre = cargar_dataframe_v0("2020_Accidentalidad.csv")
print(tabla_pre)
```

```
        HORA             DISTRITO    RANGO DE EDAD  LESIVIDAD*
0      23:15               RETIRO  DE 25 A 29 AÑOS         NaN
1      22:35      MONCLOA-ARAVACA  DE 21 A 24 AÑOS         6.0
2      20:15  FUENCARRAL-EL PARDO  DE 45 A 49 AÑOS        14.0
3      20:15  FUENCARRAL-EL PARDO  DE 25 A 29 AÑOS         7.0
4      19:45               CENTRO      DESCONOCIDA         NaN
…      …                     …               …          …
32415  00:18            CHAMARTÍN  DE 35 A 39 AÑOS         NaN
32416  00:18            CHAMARTÍN  DE 35 A 39 AÑOS         NaN
32417  00:18            CHAMARTÍN  DE 35 A 39 AÑOS         NaN
32418  00:18            CHAMARTÍN  DE 35 A 39 AÑOS         NaN
32419  00:18            CHAMARTÍN  DE 35 A 39 AÑOS         NaN
```

```
[32420 rows x 4 columns]
```

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.

[34]:
```
# Esta celda debe ser completada por el estudiante
def cargar_dataframe(archivo):
    """
    A partir de un data frame creado desde un csv
    con la función cargar_dataframe_v0, se realiza lo siguiente:
    Reemplaza valores NaN con 0
    Reeemplaza valores (77,14) en columna lesividad con 0
    Reemplaza todos los valores qque no sean 1 con 0 en la
    columna lesividad
    Aplica funcion redondeo_hora() para dar formato a columna
    hora.
    Aplica funcion rango_edad() para tranformarla a una tupla
    numerica con los rangos (75,100) ej

    Parameters:
    -----------
    archivo:
        csv file

    Precondition:
    -------------
    archivo == csv file

    Returns:
    --------
    pd.DataFrame
    Dataframe con las columnas detalladas a continuación

        ==========    ==================================================
        HORA          object que representa la hora
        DISTRITO      object que representa el distrito
        RANGO DE EDAD object que indica entre que edades se clasifican
        LESIVIDAD*    float64 que asigna un codigo por cada accidente
        ==========    ==================================================

    Example:
    --------
    >>>cargar_dataframe("2020_Accidentalidad.csv").head(2)
    HORA         DISTRITO RANGO DE EDAD  LESIVIDAD*
```

```
    0   23           RETIRO      (25, 29)           0
    1   22  MONCLOA-ARAVACA      (21, 24)           1
    """
    df = cargar_dataframe_v0(archivo)
    # Reemplaza los NaN con un 0
    df['LESIVIDAD*'] = df['LESIVIDAD*'].fillna(0)

    # Localiza la columna 3 que corresponde a lesividad
    les = df.iloc[:, 3].name

    # Localiza la columna 0 que corresponde a la hora
    hora = df.iloc[:, 0].name

    # Localiza la columna 2 que representa las edades
    edad = df.iloc[:, 2].name
    # Reemplaza valores en columna lesividad que sean 77,14 con 0
    df[les] = df[les].replace([77, 14], [0, 0]).astype(int)

    # Reemplaza todo lo distinto a 0 con 1
    df[les] = np.where(df[les] != 0, 1, df[les])

    #Transforma hora y edad al formato adecuado
    df[hora] = df[hora].apply(redondeo_hora)
    df[edad] = df[edad].apply(rango_edad)
    return df
```

[35]:
```
tabla = cargar_dataframe("2020_Accidentalidad.csv")
tabla
```

[35]:

	HORA	DISTRITO	RANGO DE EDAD	LESIVIDAD*
0	23	RETIRO	(25, 29)	0
1	22	MONCLOA-ARAVACA	(21, 24)	1
2	20	FUENCARRAL-EL PARDO	(45, 49)	0
3	20	FUENCARRAL-EL PARDO	(25, 29)	1
4	19	CENTRO	(-1, -1)	0
…	…	…	…	…
32415	00	CHAMARTÍN	(35, 39)	0
32416	00	CHAMARTÍN	(35, 39)	0
32417	00	CHAMARTÍN	(35, 39)	0
32418	00	CHAMARTÍN	(35, 39)	0
32419	00	CHAMARTÍN	(35, 39)	0

[32420 rows x 4 columns]

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.

[36]:
```
# Esta celda debe ser completada por el estudiante

def accidentes_edad(set_datos):
    """
    A partir de un data frame que contiene datos de accidentes
    en Madrid, se deben calcular el número de accidentes para
    cada rango etario.

    Parameters:
    -----------
    set_datos:
        pandas.core.frame.DataFrame

    Precondition:
    -------------
    set_datos == pd.DataFrame

    Returns:
    --------
    pd.DataFrame
    Dataframe con las columnas detalladas a continuación

        ==========    ========================================
        Edad          tupla con rangos de edad
        NumAccs       int, cuenta de accidentes para cada edad
        ==========    ========================================

    Example:
    --------
    >>>accidentes_edad(tabla).head(2)
       Edad         NumAccs
    0  (-1, -1)     3962
    1    (0, 5)      289

    """
    tabla_nueva = set_datos
    nuevo = tabla_nueva.filter(['RANGO DE EDAD'])
    nuevo = nuevo['RANGO DE EDAD'].value_counts().reset_index()
    nuevo.columns = ['Edad', 'NumAccs']
    base = nuevo.sort_values(by='Edad').reset_index(drop=True)
    return base


accidentes_edad(tabla)
```

[36]:

	Edad	NumAccs
0	(-1, -1)	3962

```
1       (0, 5)       289
2       (6, 9)       175
3     (10, 14)       305
4     (15, 17)       250
5     (18, 20)       978
6     (21, 24)      2226
7     (25, 29)      3437
8     (30, 34)      3362
9     (35, 39)      3332
10    (40, 44)      3399
11    (45, 49)      3084
12    (50, 54)      2547
13    (55, 59)      2077
14    (60, 64)      1272
15    (65, 69)       641
16    (70, 74)       427
17  (75, 100)        657
```

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

[37]:
```python
# Esta celda debe ser completada por el estudiante

def accidentes_edades(datos):
    """
    Quita indices (-1, -1) de un data frame
    que no se utilizaran en cálculos posteriores.

    Parameters:
    -----------
    datos:
        pandas.core.frame.DataFrame

    Precondition:
    -------------
    datos == pd.DataFrame

    Returns:
    --------
    pd.DataFrame
    Dataframe con las columnas detalladas a continuacion

        ==========    ============================================
        Edad          tupla con rangos de edad sin -1, -1
        NumAccs       int, cuenta de accidentes para cada edad
        ==========    ============================================
```

```
    Example:
    --------
    >>>accidentes_edades(tabla).head(2)
        Edad  NumAccs
    0  (0, 5)      289
    1  (6, 9)      175


    """
    df = accidentes_edad(datos)
    # con drop True no agrega una nueva columna al indice
    acc_edad = df.iloc[1:].reset_index(drop=True)
    return acc_edad


accidentes_edades(tabla)
```

[37]:

	Edad	NumAccs
0	(0, 5)	289
1	(6, 9)	175
2	(10, 14)	305
3	(15, 17)	250
4	(18, 20)	978
5	(21, 24)	2226
6	(25, 29)	3437
7	(30, 34)	3362
8	(35, 39)	3332
9	(40, 44)	3399
10	(45, 49)	3084
11	(50, 54)	2547
12	(55, 59)	2077
13	(60, 64)	1272
14	(65, 69)	641
15	(70, 74)	427
16	(75, 100)	657

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.

[38]:
```
# Esta celda debe ser completada por el estudiante

def accidentes_medicos(base):
    """
    Totaliza los accidentes que requieren atencion médica
    o que resultan en muerte por cada rango de edad.
```

```
    Parameters:
    -----------
    base:
        pandas.core.frame.DataFrame

    Precondition:
    -------------
    base == pd.DataFrame

    Returns:
    --------
    pd.DataFrame
    Dataframe con las columnas detalladas a continuación

        ==========          ================================================
        Edad                Una tupla con rangos de edad
        NumAccsConLesiones  int, cuenta de accidentes con lesiones x edad
        ==========          ================================================

    Example:
    --------
    >>>accidentes_medicos(tabla).head(2)
    Edad         NumAccsConLesiones
    0  (0, 5)                  107
    1  (6, 9)                   73
    """
    archivo_base = base.filter(['RANGO DE EDAD', 'LESIVIDAD*'])

    # Ordena las columnas por edades
    df = archivo_base.sort_values(by=['RANGO DE EDAD', 'LESIVIDAD*'])

    filtros = df[(df['LESIVIDAD*'] >= 1)]
    df = filtros.groupby(['RANGO DE EDAD']).size().reset_index(name='count')

    # Quita el indice 0 y reordena
    accidentes_lesiones = df.iloc[1:].reset_index(drop=True)
    accidentes_lesiones.columns = ['Edad', 'NumAccsConLesiones']

    return accidentes_lesiones


accidentes_medicos(tabla)
```

[38]:

	Edad	NumAccsConLesiones
0	(0, 5)	107
1	(6, 9)	73
2	(10, 14)	135

```
3     (15, 17)                 114
4     (18, 20)                 325
5     (21, 24)                 720
6     (25, 29)                1184
7     (30, 34)                1098
8     (35, 39)                 944
9     (40, 44)                 899
10    (45, 49)                 735
11    (50, 54)                 587
12    (55, 59)                 493
13    (60, 64)                 281
14    (65, 69)                 153
15    (70, 74)                 129
16  (75, 100)                  260
```

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.

[39]:
```python
# Esta celda debe ser completada por el estudiante

def union_tablas(tablas):
    """
    Une dos dataframes accidentes_edades() y
    accidentes_medicos() para crear un data frame único que
    contenga los accidentes y lesiones por cada rango de edad

    Parameters:
    -----------
    tablas:
        pandas.core.frame.DataFrame

    Precondition:
    -------------
    tablas == pd.DataFrame

    Returns:
    --------
    pd.DataFrame
    Dataframe con las columnas detalladas a continuación

        ==========          ===============================================
        Edad                object que contiene tupla con rangos de edad
        NumAccs             int, cuenta de accidentes totales por edad
        NumAccsConLesiones  int, cuenta de accidentes con lesiones p/ edad
        ==========          ===============================================
```

```
    Example:
    --------
    >>>union_tablas(tabla).head(2)
        Edad  NumAccs   NumAccsConLesiones
    0  (0, 5)      289                 107
    1  (6, 9)      175                  73

    """
    return pd.merge(accidentes_edades(tablas), accidentes_medicos(tablas))


union_tablas(tabla)
```

[39]:

	Edad	NumAccs	NumAccsConLesiones
0	(0, 5)	289	107
1	(6, 9)	175	73
2	(10, 14)	305	135
3	(15, 17)	250	114
4	(18, 20)	978	325
5	(21, 24)	2226	720
6	(25, 29)	3437	1184
7	(30, 34)	3362	1098
8	(35, 39)	3332	944
9	(40, 44)	3399	899
10	(45, 49)	3084	735
11	(50, 54)	2547	587
12	(55, 59)	2077	493
13	(60, 64)	1272	281
14	(65, 69)	641	153
15	(70, 74)	427	129
16	(75, 100)	657	260

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.

[40]:
```
# Esta celda debe ser completada por el estudiante
def acc_les_prop(datos):
    """
    Entrega la proporción de accidentes en que se producen
    lesiones

    Parameters:
    -----------
    datos:
        pandas.core.frame.DataFrame
```

```
    Precondition:
    -------------
    datos == pd.DataFrame

    Returns:
    --------
    pd.DataFrame
    Dataframe con las columnas detalladas a continuación

        ==========          ================================================
        Edad                tupla con rangos de edad
        NumAccs             int, cuenta de accidentes totales por edad
        NumAccsConLesiones  int, cuenta de accidentes con lesiones x edad
        PropLesiones        float64, cálculo de proporción de accidentes
                            c/ lesiones
        ==========          ================================================

    Example:
    --------
    >>>acc_les_prop(tabla).head(2)
        Edad  NumAccs   NumAccsConLesiones
    0  (0, 5)      289                 107
    1  (6, 9)      175                  73
    """

    tabla_prop = union_tablas(datos)
    tabla_prop['PropLesiones'] = (tabla_prop['NumAccsConLesiones'] * 100 /
                                  tabla_prop['NumAccs'])
    return tabla_prop


acc_les_prop(tabla)
```

[40]:

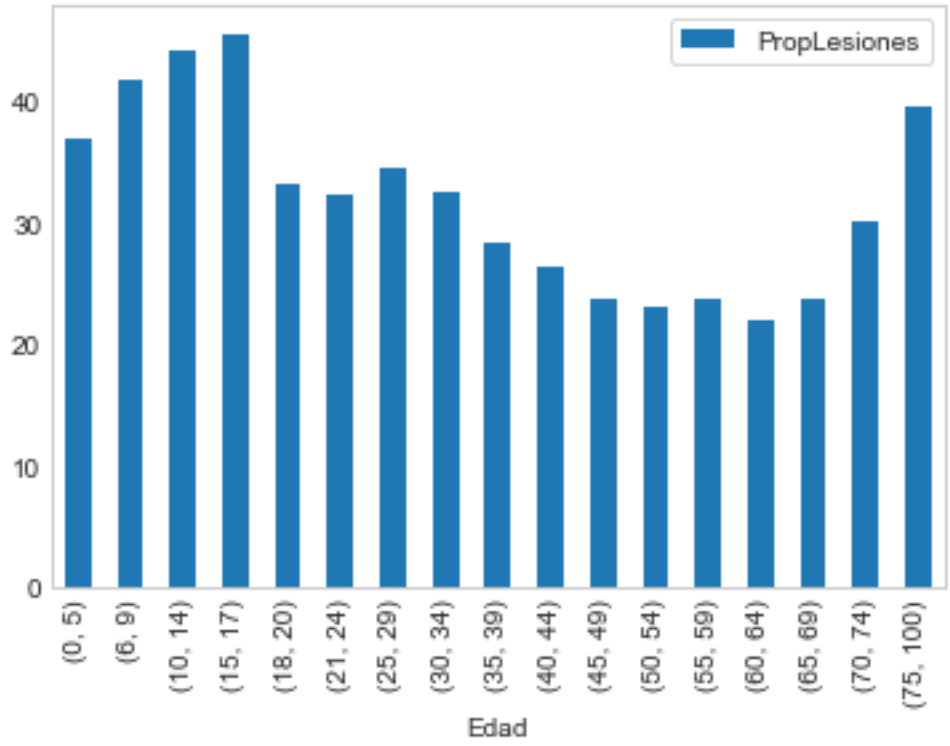
	Edad	NumAccs	NumAccsConLesiones	PropLesiones
0	(0, 5)	289	107	37.024221
1	(6, 9)	175	73	41.714286
2	(10, 14)	305	135	44.262295
3	(15, 17)	250	114	45.600000
4	(18, 20)	978	325	33.231084
5	(21, 24)	2226	720	32.345013
6	(25, 29)	3437	1184	34.448647
7	(30, 34)	3362	1098	32.659131
8	(35, 39)	3332	944	28.331333
9	(40, 44)	3399	899	26.448956
10	(45, 49)	3084	735	23.832685
11	(50, 54)	2547	587	23.046722
12	(55, 59)	2077	493	23.736158

```
13   (60, 64)     1272                281     22.091195
14   (65, 69)      641                153     23.868955
15   (70, 74)      427                129     30.210773
16  (75, 100)      657                260     39.573820
```

e6. Gráfico.

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

[41]:
```python
# Esta celda debe ser completada por el estudiante
def grafico(datos):
    """
    Crea grafico a partir del data frame generado en la
    funcion acc_les_prop() que calcula la proporción de accidentes.

    Parameters:
    -----------
    datos:
        pandas.core.frame.DataFrame

    Precondition:
    -------------
    datos == pd.DataFrame

    Returns:
    --------
        plot

    Example:
    --------
    >>>grafico(tabla)
    plt.show()
    """
    plotdata = acc_les_prop(datos)
    plotdata = plotdata[['Edad', 'PropLesiones']]
    plotdata.plot(kind='bar', x='Edad', label='aaa', color='tab:blue')
    plt.grid(False)
    plt.legend(frameon=True)
    plt.show()


grafico(tabla)
```



0.9 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:

[42]:
```
# Hagamos una llamada al programa de consola desde aquí:

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

```
[70,74] [427,1]
[75,100]        [657,4]
[40,44] [3399,6]
[25,29] [3437,3]
[30,34] [3362,2]
[35,39] [3332,8]
[55,59] [2077,1]
[21,24] [2226,2]
[45,49] [3084,4]
```

```
[50,54] [2547,1]
[0,5]   [289,1]
[10,14] [305,0]
[15,17] [250,0]
[18,20] [978,0]
[6,9]   [175,0]
[60,64] [1272,1]
[65,69] [641,1]
```

[43]:
```
# 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`.

[44]:
```
! cat total_accs_edad.py
```

```
from mrjob.job import MRJob


def rango_edad(edad):
    """
    Dado un string que representa una edad
    calcula su rango etario correspondiente.

    Parameters:
    -----------
    edad:
      string que contiene descripcion de edades.

    Precondition:
    -------------
    edad == string

    Returns:
    --------
        tuple
        Tupla numerica con las edades en numeros enteros.

    Example:
    --------
    >>>rango_edad('DE 0 A 5 AÑOS')
```

```
    (0,5)
    """
    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(edad, (-1, -1))


def entero(cadena):
    """
    Funcion auxiliar que toma un string que
    representa la lesividad y la convierte en entero.

    Parameters:
    -----------
    cadena:
      string que representa un codigo de accidente

    Returns:
    --------
        int
        Entero que representa el codigo de accidente.

    Example:
    --------
    >>>entero('4')
    4
    """
    try:
        return int(cadena)
    except:
        return 0
```

```
def arma_lista(lista):
    """
    Funcion auxliar que guarda los accidentes,
    las muertes y retorna la suma para cada lista
    generada, esta funcion se usa en el reducer.

    Parameters:
    -----------
    lista:
      lista que contiene la cuenta de accidentes
      y muertes.

    Returns:
    --------
        tuple
        Tupla con la suma de accidentes y suma de muertes.

    Example:
    --------
    >>>arma_lista(values)
    (3437, 3)
    """
    accidentes = []
    muertes = []
    for i, (a, m) in enumerate(lista):
        accidentes.append(a)
        muertes.append(m)
    return (sum(accidentes), sum(muertes))


class Cuenta(MRJob):
    """
    La clase Cuenta llama a MrJob para en un inicio
    mapear linea a linea los accidentes y despues
    contar los accidentes con lesividad igual a 4

    Attributes:
        linea: cada linea leida que sera mapeada
    """

    def mapper(self, _, linea):
        accidentes, muertes = 0, 0
        row = linea.split(';')
        edad = rango_edad(row[10])
        lesividad = entero(row[12])
        if edad != (-1, -1):
```

```
            accidentes = 1
            if lesividad == 4:
                muertes = 1
            yield edad, [accidentes, muertes]

    def reducer(self, key, values):
        yield key, arma_lista(values)


if __name__ == '__main__':
    Cuenta.run()
```

0.10 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.

En este apartado libre he querido saber la cantidad de accidentes contabilizados segun su tipo, pero separados por sexo y generando un grafico que muestre visualmente los datos recolectados mas la tabla pivote generada en el codigo

[45]:
```
# Este apartado debe ser completado por el estudiante
def c_dataframe(archivo, columnas=None):
    """
    Crea un dataframe a partir de un archivo csv y extrae
    las columnas requeridas.

    Parameters:
    -----------
    archivo:
        csv file
    columns (opcional):
```

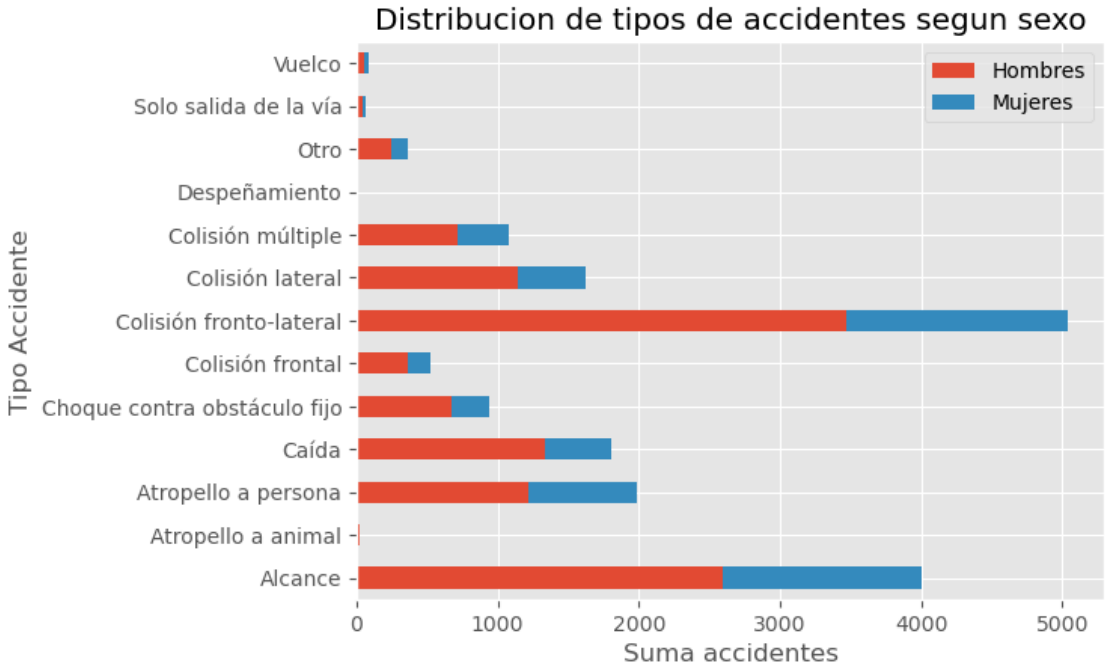
```
        lista que contiene las columnas que se requieren extraer
        si no se especifican se incluyen todas las columnas
        en el data frame a crear

    Precondition:
    -------------
    archivo == csv file
    columns >= len(0)

    Returns:
    --------
        plot

    Example:
    --------
    >>>c_dataframe("2020_Accidentalidad.csv",
        columnas=['SEXO', 'TIPO ACCIDENTE',
        'RANGO DE EDAD', 'LESIVIDAD*']).head(2)

    TIPO ACCIDENTE                    RANGO DE EDAD    SEXO  LESIVIDAD*
    0  Choque contra obstáculo fijo  DE 25 A 29 AÑOS  Hombre         NaN
    1  Caída                         DE 21 A 24 AÑOS   Mujer         6.0
    """
    base = pd.read_csv(archivo, encoding='unicode_escape', on_bad_lines='skip',
                       delimiter=";", usecols=columnas)
    return base


c_dataframe("2020_Accidentalidad.csv",
            columnas=['SEXO', 'TIPO ACCIDENTE', 'RANGO DE EDAD', 'LESIVIDAD*'])
```

[45]:

```
                      TIPO ACCIDENTE    RANGO DE EDAD    SEXO  LESIVIDAD*
0      Choque contra obstáculo fijo  DE 25 A 29 AÑOS  Hombre         NaN
1                             Caída  DE 21 A 24 AÑOS   Mujer         6.0
2                             Caída  DE 45 A 49 AÑOS  Hombre        14.0
3                             Caída  DE 25 A 29 AÑOS  Hombre         7.0
4      Choque contra obstáculo fijo      DESCONOCIDA     NaN         NaN
…                                 …              …     …           …
32415              Colisión lateral  DE 35 A 39 AÑOS  Hombre         NaN
32416              Colisión lateral  DE 35 A 39 AÑOS  Hombre         NaN
32417              Colisión lateral  DE 35 A 39 AÑOS  Hombre         NaN
32418              Colisión lateral  DE 35 A 39 AÑOS  Hombre         NaN
32419              Colisión lateral  DE 35 A 39 AÑOS  Hombre         NaN

[32420 rows x 4 columns]
```

[46]:
```
# Pruebas de funcionamiento:
def grafico_pivote(archivo, columnas):
    """
    Transforma un dataframe en una tabla pivote y posterior
    a esto se crea gráfico a partir de la tabla pivote.
    La tabla pivote sumariza los tipo de accidentes segun sexo
    del causante.

    Parameters:
    -----------
    archivo:
        csv file
    columnas:
        Lista que contiene las columnas a usar en la tabla pivote

    Precondition:
    -------------
    archivo == csv file
    len(columnas) > 0

    Returns:
    --------
        plot
        pd.Data.Frame
            Tabla pivote


    Example:
    --------
    >>>grafico_pivote('2020_Accidentalidad.csv',
            columnas=['SEXO', 'TIPO ACCIDENTE',
               'RANGO DE EDAD', 'LESIVIDAD*'])
    plt.show()

    Tabla pivote:
              TIPO ACCIDENTE      count
                                LESIVIDAD*
                     SEXO          Hombre Mujer
    0     Alcance                   2592  1414
    1     Atropello a animal          17     9

    """
    data = c_dataframe(archivo, columnas)
    # Convierte el df a una tabla pivote
    pivot = pd.pivot_table(data, values=['LESIVIDAD*'],
                           index=['TIPO ACCIDENTE'],
                           columns=['SEXO'],
```

```
                            fill_value=0,
                            aggfunc=['count'])
    df = pivot.reset_index()

    # Apartado de graficos
    plt.rcdefaults()
    plt.style.use('ggplot')
    pivot.plot.barh(stacked=True)
    plt.xlabel('Suma accidentes')
    plt.ylabel('Tipo Accidente')
    plt.legend(frameon=True)
    plt.legend(['Hombres', 'Mujeres'])
    plt.title('Distribucion de tipos de accidentes segun sexo')
    return plt.show(), df


resultado = grafico_pivote(
    '2020_Accidentalidad.csv',
    columnas=['SEXO', 'TIPO ACCIDENTE', 'RANGO DE EDAD', 'LESIVIDAD*'])
```



[47]:
```
# Apartado de tabla pivote
"""
De la funcion grafico extrae el segundo indice del return que contiene
la tabla pivote y la muestra
"""
```

```
df = resultado[1]
df
```

[47]:

TIPO ACCIDENTE	count	
	LESIVIDAD*	
SEXO	Hombre	Mujer
0 Alcance	2592	1414
1 Atropello a animal	17	9
2 Atropello a persona	1214	769
3 Caída	1337	470
4 Choque contra obstáculo fijo	668	269
5 Colisión frontal	364	155
6 Colisión fronto-lateral	3474	1570
7 Colisión lateral	1140	487
8 Colisión múltiple	717	362
9 Despeñamiento	0	1
10 Otro	247	116
11 Solo salida de la vía	40	22
12 Vuelco	56	25

0.10.1 Datos personales

- **Apellidos:** Yañez Mendoza
- **Nombre:** Ivonne Valeska
- **Email:** ivonne@imendoza.io
- **Fecha:** 17 de mayo de 2022

0.10.2 Ficha de autoevaluación

Comentarios sobre la autoevaluación:

Siendo honesta a mi las autoevaluaciones me cuestan. En general tiendo a pensar que lo realizado siempre puede quedar mejor y que está bien más no excepcional. Pero en este caso quiero ser un poco mas generosa conmigo y aún cuando creo que se puede mejorar el código, siento que he tenido un buen desempeño que se condensa en un trabajo prolijo y adecuadamente documentado, donde he ido adquieriendo más y mejor conocimiento del lenguaje y del estilo pitónico siendo lo más importante el haber podido practicar, mejorar, aprender y refrescar conocimientos, lo cual independientemente de la nota final para mi ha sido lo mas relevante y enriquecedor de esta tarea.

Apartado	Calificación	Comentario
a)	2.0 / 2.0	Completamente resuelto
b)	2.0 / 2.0	Completamente resuelto
c)	1.5 / 2.0	Completamente resuelto
d)	1.25 / 1.5	Completamente resuelto
e)	1.5 / 1.5	Completamente resuelto
f)	0.25 / 0.5	Completamente resuelto
g)	0.5 / 0.5	Completamente resuelto

Apartado	Calificación	Comentario
Total	**9** / 10.0	**Suspenso**

0.10.3 Ayuda recibida y fuentes utilizadas

```
Documentar
https://stackoverflow.com/questions/57225904/type-annotating-pandas-dataframes
https://stackoverflow.com/questions/53849585/how-to-specify-type-in-docstrings
https://google.github.io/styleguide/pyguide.html

Python
https://stackoverflow.com/questions/60208/replacements-for-switch-statement-in-python
https://favtutor.com/blogs/merge-dictionaries-python
https://realpython.com/python-enumerate/
https://realpython.com/python-defaultdict/

Obtener primer y ultimo elemento lista
https://tinyurl.com/mrymuzx8

Error de encoding en macOS
https://tinyurl.com/2p82tns4

En lo personal durante un tiempo hice varios ejercicios en esta plataforma y
he ido recordando ejercicios que han servido en este trabajo, adjunto mi user:
https://edabit.com/user/utyt9o5BYPWRbXf2X

Graficos
https://tinyurl.com/plot-tuple
https://towardsdatascience.com/pandas-tips-and-tricks-33bcc8a40bb9

Set color azul por default en plots
https://tinyurl.com/blue-plot

Pandas
https://stackoverflow.com/questions/57225904/type-annotating-pandas-dataframes
https://pandas.pydata.org/pandas-docs/stable/development/contributing_docstring.html
https://tinyurl.com/delete-first-row
https://tinyurl.com/storing-first-value

Extraer columnas especificas en pandas
https://tinyurl.com/bdd7zyux

Agrupar en columnas pandas
https://tinyurl.com/parrvhs5
https://tinyurl.com/columns-pandas

Map, reduce
```

https://realpython.com/python-map-function/ # Solo como consulta
https://computehustle.com/2019/09/02/getting-started-with-mapreduce-in-python/
https://stackoverflow.com/questions/65059930/mrjob-iterating-over-values
https://medium.com/datable/beginners-guide-for-mapreduce-with-mrjob-in-python-dbd2e7dd0f86
https://tinyurl.com/3yhputdz
https://tinyurl.com/MrJob

Ejercicio libre
https://towardsdatascience.com/pivot-tables-with-pandas-7d1078167d62

Libros:

VanderPlas, J. (2017), Python data science handbook.
Lubanovic, Bill.(2015), Introducing python
Hellmann, Doug. (2011), The Python Standard Library by Example

Otras fuentes consultadas:
Pandas: https://github.com/jvns/pandas-cookbook
Comprobar estilo: https://github.com/mattijn/pycodestyle_magic

Ayuda recibida:
Agradecimientos especiales a compañeros del Master con quienes intercambiamos impresiones, dudas y sugerencias:
Nelson Bonilla, Arturo Berrios y en especial a Gonzalo Cristobal Manchado quien me brindó guía para resolver el ejercicio de MRJob.

0.10.4 Comentario adicional

En lo personal me ha gustado este ejercicio, ha sido bastante trabajo, pero siento que condensa de forma correcta lo visto en clase y me ha permitido pasar de los ejercicios más básicos o abstractos a crear cosas que se pueden usar en el mundo real. En mi caso aprendí con java a programar, después pase a utilizar PeopleCode para cálculos de nóminas de salarios en Oracle Peoplesoft y he estudiado JavaScript para desarrollo web y por lejos Python es el mejor lenguaje que he utilizado pues se asemeja a como naturalmente daríamos instrucciones a una máquina, es agradable de aprender y usar. Además admiro el trabajo de la comunidad, del equipo de traductores al español (del cual he aportado con ayuda) y de establecer oportunidades, mentorías y un ambiente de respeto y buen trato sobre todo para mujeres y minorías en la industria desde mucho antes que esto se empezase a discutir de forma masiva. Lo cual es muy relevante pues aun la brecha es muy grande en la industria aunque cada vez somos más y se van creando lazos y amistad en las comunidades. Tambien dejo un comentario sobre la calidad de los materiales entregados, los ejercicios han sido muy completos de seguir y practicar.

[48]:
```
# Esta celda se ha de respetar: está aquí para comprobar el funcionamiento de␣
↪algunas funciones por parte de tu profesor
```