Momo

February 9, 2021

1 Informes de mortalidad

Actualizado diariamente, este documento se visualiza mejor aquí.

Datos del Sistema de Monitorización de la Mortalidad diaria, que incluye las defunciones por todas las causas procedentes de 3.929 registros civiles informatizados, que representan el 92% de la población española.

[1]:
```
# Cargamos datos
import Loading_data
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from IPython.display import display, HTML
```

```
/root/scripts/COVID-19/jupyter/Loading_data.py:22: FutureWarning: Sorting
because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  df = pd.concat([df,this_df])
```

[2]:
```
df = pd.read_csv('https://momo.isciii.es/public/momo/data')
df.to_csv('/tmp/momo.csv')
df.head()
```

[2]:

	ambito	cod_ambito	cod_ine_ambito	nombre_ambito	cod_sexo	nombre_sexo	cod_gedad	nombre_gedad	fecha_defuncion	defunciones_observadas
0	nacional	NaN	NaN	NaN	all	todos	all	todos	2018-12-31	1111
1	nacional	NaN	NaN	NaN	all	todos				
2	nacional	NaN	NaN	NaN	all	todos				
3	nacional	NaN	NaN	NaN	all	todos				
4	nacional	NaN	NaN	NaN	all	todos				

```
1       all        todos      2019-01-01                    1166
2       all        todos      2019-01-02                    1233
3       all        todos      2019-01-03                    1270
4       all        todos      2019-01-04                    1308

   defunciones_observadas_lim_inf  defunciones_observadas_lim_sup  \
0                          1111.0                          1111.0
1                          1166.0                          1166.0
2                          1233.0                          1233.0
3                          1270.0                          1270.0
4                          1308.0                          1308.0

   defunciones_esperadas  defunciones_esperadas_q01  defunciones_esperadas_q99
0                 1267.0                   1108.025                     1514.35
1                 1269.0                   1108.025                     1563.25
2                 1302.0                   1108.025                     1635.40
3                 1302.0                   1108.025                     1635.40
4                 1312.0                   1087.900                     1635.40
```

[3]:
```
import janitor
import datetime



def pipeline_basic_with_query(df,query):
    ''' Basic filtering, using janitor
        Carga de datos, enriquecimiento de fechas y filtro por query␣
 ↪configurable
    '''
    LISTA_COLUMNAS_A_BORRAR = ['Unnamed: 0',
                           'defunciones_observadas_lim_inf',
                           'defunciones_observadas_lim_sup',
                           'defunciones_esperadas',
                           'defunciones_esperadas_q01',
                           'defunciones_esperadas_q99']
    return (
        df
        # Quitar: columnas
        .remove_columns(LISTA_COLUMNAS_A_BORRAR)
        .clean_names()
        # Enriquecer: fechas con columnas de años, mes y año-mes
        .rename_column( "fecha_defuncion", "date")
        .to_datetime('date')
        .join_apply(lambda x: x['date'].strftime('%Y')    ,␣
 ↪new_column_name="date_year"       )
        .join_apply(lambda x: x['date'].strftime('%m')    ,␣
 ↪new_column_name="date_month"      )
```

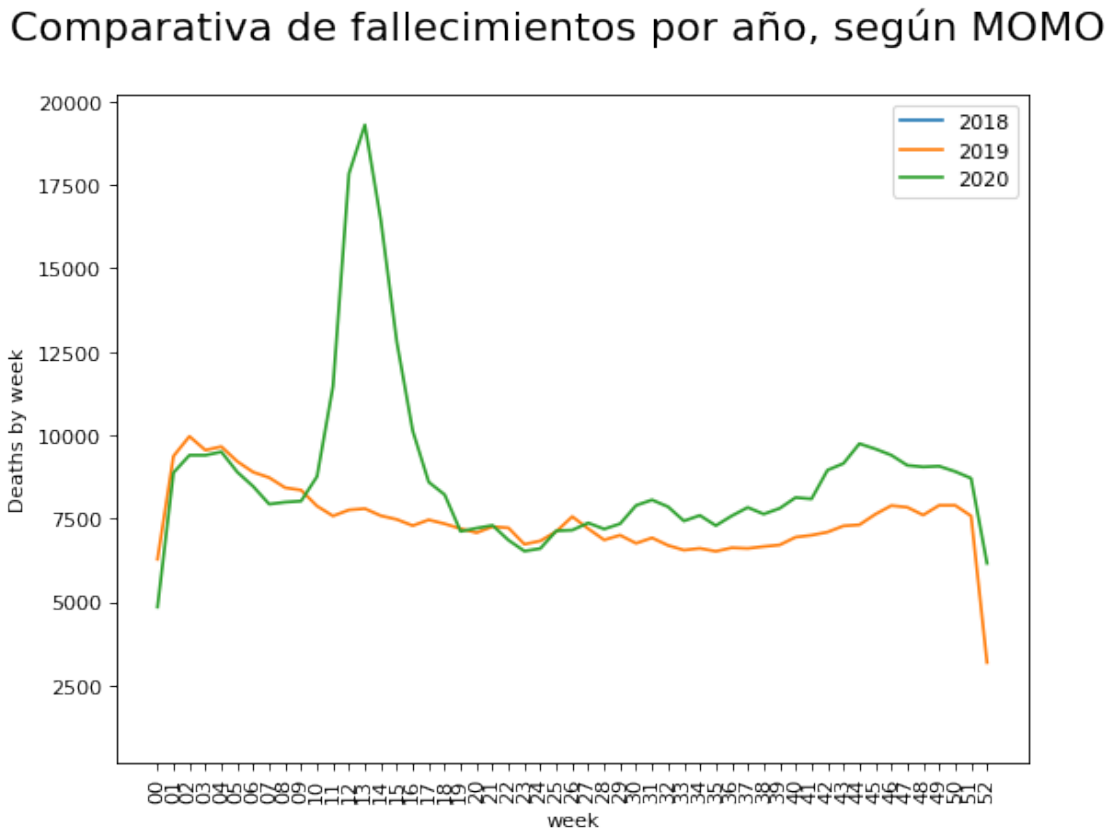
```
        .join_apply(lambda x: x['date'].strftime('%m-%d') ,␣
↪new_column_name="date_month_day"  )
        .join_apply(lambda x: x['date'].strftime('%U')    ,␣
↪new_column_name="date_week"       )
        .join_apply(lambda x: x['date'].strftime('%Y-%m') ,␣
↪new_column_name="date_year_month" )
        .join_apply(lambda x: x['date'].strftime('%Y-%U') ,␣
↪new_column_name="date_year_week"  )
        # Filtrar:por query
        .filter_on( query )
        .set_index('date')
    )

def pipeline_basic(df):
    query = 'ambito       == "nacional" & nombre_gedad == "todos"  &␣
↪nombre_sexo  == "todos"  '
    return pipeline_basic_with_query(df,query)




def extraer_defunciones_anuales_por_periodo(periodo_de_tiempo,query):
    '''Extrae el cuadro de comparativa por week, or year '''

    def pipeline_agregado_anual(periodo_de_tiempo,df,year):
        ''' Saca un dataframe de los datos agrupados por año'''
        return (
            df
            .filter_on('date_year == "'+year+'"'  )
            .groupby_agg( by='date_'+periodo_de_tiempo, agg='sum',␣
↪agg_column_name="defunciones_observadas", new_column_name="agregados")
            .rename_column( "agregados", year)
            .join_apply(lambda x: x['date_'+periodo_de_tiempo]  ,␣
↪new_column_name=periodo_de_tiempo )
            .set_index('date_'+periodo_de_tiempo)
            [[periodo_de_tiempo,year]]
            .drop_duplicates()
        )

    def pipeline_comparativa_anual(periodo_de_tiempo,df_2018,df_2019,df_2020):
        ''' Mergea tres dataframes de año, por periodo de tiempo'''
        return  (
        df_2018
        .merge( df_2019, on=periodo_de_tiempo, how='right')
        .merge( df_2020, on=periodo_de_tiempo, how='left')
        .sort_naturally(periodo_de_tiempo)
```

```
        .set_index(periodo_de_tiempo)
        .join_apply(lambda x: x['2020'] - x['2019'] , new_column_name="resta␣
↪2020 y 2019"  )
        )

    # Sacamos los datos y limpiamos
    df       = pd.read_csv('/tmp/momo.csv')
    df_basic = pipeline_basic_with_query(df,query)

    # Sacamos los datos agrupados por años
    muertes_2018  =␣
↪pipeline_agregado_anual(periodo_de_tiempo,df=df_basic,year='2018')
    muertes_2019  =␣
↪pipeline_agregado_anual(periodo_de_tiempo,df=df_basic,year='2019')
    muertes_2020  =␣
↪pipeline_agregado_anual(periodo_de_tiempo,df=df_basic,year='2020')

    # Generamos un solo cuadro, con columna por año
    df_comparativa_años  =␣
↪pipeline_comparativa_anual(periodo_de_tiempo,muertes_2018,muertes_2019,muertes_2020)
    return df_comparativa_años

def debug_extraer_defunciones_anuales_por_periodo():
    """ Solo para depurar"""
    query = 'ambito       == "nacional" & nombre_gedad == "todos"  &␣
↪nombre_sexo  == "todos"  '
    df_muertes_anuales_por_semana =␣
↪extraer_defunciones_anuales_por_periodo("week",query)
    df_muertes_anuales_por_mes    =␣
↪extraer_defunciones_anuales_por_periodo("month",query)
    return  df_muertes_anuales_por_semana , df_muertes_anuales_por_mes


#df1, df2 = debug_extraer_defunciones_anuales_por_periodo()
#df1
```

1.1 Sacamos el grafico comparativo de fallecimiento, para los años 2019 y 2020, por semana

[4]:
```
from matplotlib import pyplot as plt
from IPython.display import display, HTML
import pandas as pd

import numpy as np
```

```
periodo_de_tiempo="week"
query = 'ambito       == "nacional" & nombre_gedad == "todos"  & nombre_sexo ␣
 ↪== "todos"  '

df = extraer_defunciones_anuales_por_periodo(periodo_de_tiempo,query)

fig = plt.figure(figsize=(8, 6), dpi=80)
plt.xticks(rotation=90)

for ca in ['2018','2019','2020']:
    plt.plot(df[ca])
    plt.legend(df.columns)
    plt.xlabel(periodo_de_tiempo)
    plt.ylabel("Deaths by " + periodo_de_tiempo)
    fig.suptitle('Comparativa de fallecimientos por año, según MOMO',␣
 ↪fontsize=20)
plt.show()



periodo_de_tiempo="week"
query = 'ambito       == "nacional" & nombre_gedad == "todos"  & nombre_sexo ␣
 ↪== "todos"  '
df = extraer_defunciones_anuales_por_periodo(periodo_de_tiempo,query)

df.style.format({"2020": "{:20,.0f}",
                          "2018": "{:20,.0f}",
                          "2019":  "{:20,.0f}",
                          "resta 2020 y 2019": "{:20,.0f}", }).
 ↪background_gradient(cmap='Wistia',subset=['resta 2020 y 2019'])
```



[4]: <pandas.io.formats.style.Styler at 0x7f1695d4ec88>

[5]:
```
def get_current_year_comparison(query):
    """Saca muertos del año en curso en el ambito como argumento"""
    df          = pd.read_csv('/tmp/momo.csv')
    df = pipeline_basic_with_query(df,query)

    semana_actual = df.tail(1).date_week.values[0]
    year_actual = df.tail(1).date_year.values[0]
    date_month_day_actual= df.tail(1).date_month_day.values[0]
    year_last = str(int(year_actual)-1)

    death_this_year_today = df.query( f"date_year == '{year_actual}' ").
↪defunciones_observadas.sum()
    deaht_last_year_today = df.query( f"date_year == '{year_last}' and␣
↪date_month_day <= '{date_month_day_actual}' ").defunciones_observadas.sum()
    deaths_this_year_excess = death_this_year_today - deaht_last_year_today
    return deaths_this_year_excess
```

```
query = f""" ambito == "nacional" & nombre_gedad == "todos" & nombre_sexo  ==␣
 ↪"todos"  """
deaths_this_year_excess = get_current_year_comparison(query)

display(HTML(f"<h4 id='excedente'>Excdente de muertes de este año, respecto al␣
 ↪año anterior:</h4><h2>{deaths_this_year_excess:,.0f} </h2>"))
```

```
<IPython.core.display.HTML object>
```

[6]:
```
query = f""" nombre_ambito == "Madrid, Comunidad de" & nombre_gedad == "todos"␣
 ↪& nombre_sexo  == "todos"  """
deaths_this_year_excess = get_current_year_comparison(query)
display(HTML(f"<h4 id='excedentemadrid'>Excedente de muertes de este año en␣
 ↪Madrid, respecto al año anterior:</h4><h2>{deaths_this_year_excess:,.0f} </
 ↪h2>"))
```

```
<IPython.core.display.HTML object>
```

[7]:
```
# Sacamos las muertes en madrid de hombres y de mujeres

import numpy as np
import seaborn as sns


def pipeline_comparativa_semestral_diaria(df):
    return  (
        df
        .filter_on(" defunciones_observadas > 0")
        .
 ↪remove_columns(['nombre_gedad','ambito','cod_ambito','cod_ine_ambito','nombre_ambito','cod_
        .rename_column( "nombre_sexo"    , "sexo")
        .rename_column( "date_year_month",  "mes")
        )

# Sacamos los datos de 2019
df          = pd.read_csv('/tmp/momo.csv')
query       = '  date_year == "2019" & nombre_ambito       == "Madrid,␣
 ↪Comunidad de" & nombre_gedad == "todos"  & nombre_sexo  != "todos" & ␣
 ↪date_month < "13" '
df_madrid_2019 =  pipeline_basic_with_query(df,query)
df_madrid_2019 =  pipeline_comparativa_semestral_diaria(df_madrid_2019)

# Sacamos los datos de 2020
df       = pd.read_csv('/tmp/momo.csv')
query = ' date_year == "2020"  &  nombre_ambito       == "Madrid, Comunidad de"␣
 ↪& nombre_gedad == "todos"  & nombre_sexo  != "todos" &  date_month < "13" '
```

```
df_madrid_2020 =  pipeline_basic_with_query(df,query)
df_madrid_2020 =  pipeline_comparativa_semestral_diaria(df_madrid_2020)



df_madrid_2019
```

[7]:

date	sexo	defunciones_observadas	date_month_day	mes
2019-01-01	hombres	60	01-01	2019-01
2019-01-02	hombres	55	01-02	2019-01
2019-01-03	hombres	63	01-03	2019-01
2019-01-04	hombres	62	01-04	2019-01
2019-01-05	hombres	55	01-05	2019-01
…	…	…	…	…
2019-12-27	mujeres	61	12-27	2019-12
2019-12-28	mujeres	58	12-28	2019-12
2019-12-29	mujeres	57	12-29	2019-12
2019-12-30	mujeres	54	12-30	2019-12
2019-12-31	mujeres	68	12-31	2019-12

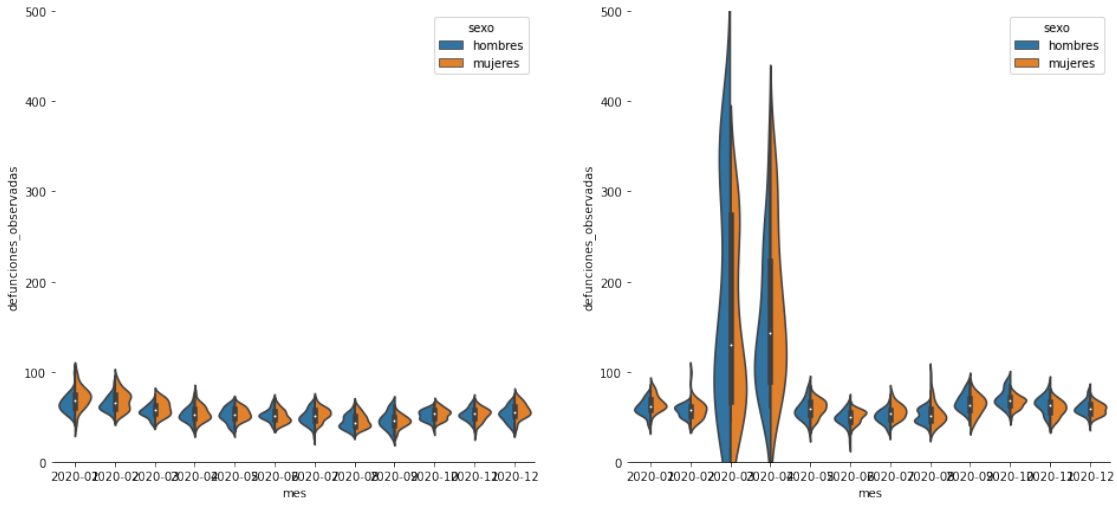
[730 rows x 4 columns]

[8]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

display(HTML("<h2>Distribucion muertes en Madrid  </h2>"))
display(HTML("<h3>Comparativa de defunciones, entre el primer semestre de 2019␣
 ↪y el del 2020</h3>"))

f, axes = plt.subplots(1 , 2 ,figsize=(16, 7), sharex=True)
sns.despine(left=True)

# Mismo limites, para poder comparar entre años
axes[0].set_ylim([0,500])
axes[1].set_ylim([0,500])


sns.violinplot(x="mes", y="defunciones_observadas", hue="sexo",
                    data=df_madrid_2019,  split=True, scale="count", ax=axes[0]␣
 ↪)


sns.violinplot(x="mes", y="defunciones_observadas", hue="sexo",
                    data=df_madrid_2020,  split=True, scale="count", ax=axes[1])
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1695d27390>
```



[9]:
```
# Aux functions
def print_categorical_variables(df):
    """ Get a dict with categorical variables"""
    my_dict = {}
    cols = df.columns
    num_cols = df._get_numeric_data().columns
    # Show categorical values
    categorical = list(set(cols) - set(num_cols))
    for i in categorical :
        if 'echa' not in i.lower() : my_dict[i] = df[i].unique()
    return my_dict
df       = pd.read_csv('/tmp/momo.csv')
my_dict = print_categorical_variables(df)
my_dict
```

[9]:
```
{'cod_ambito': array([nan, 'AN', 'AR', 'AS', 'IB', 'CN', 'CB', 'CL', 'CM', 'CT',
'VC',
        'EX', 'GA', 'MD', 'MC', 'NC', 'PV', 'RI', 'CE', 'ML'], dtype=object),
 'ambito': array(['nacional', 'ccaa'], dtype=object),
 'nombre_ambito': array([nan, 'Andalucía', 'Aragón', 'Asturias, Principado de',
        'Balears, Illes', 'Canarias', 'Cantabria', 'Castilla y León',
        'Castilla - La Mancha', 'Cataluña', 'Comunitat Valenciana',
        'Extremadura', 'Galicia', 'Madrid, Comunidad de',
```

```
       'Murcia, Región de', 'Navarra, Comunidad Foral de', 'País Vasco',
       'Rioja, La', 'Ceuta', 'Melilla'], dtype=object),
 'cod_sexo': array(['all', '1', '6'], dtype=object),
 'cod_gedad': array(['all', 'menos_65', '65_74', 'mas_74'], dtype=object),
 'nombre_gedad': array(['todos', 'edad < 65', 'edad 65-74', 'edad > 75'],
dtype=object),
 'nombre_sexo': array(['todos', 'hombres', 'mujeres'], dtype=object)}
```

[ ]: