COVID-19 Situation in Spain

Libraries

3 cells hidden

Data loading and general overview

First, we will load the csv file from the offical Spanish goverment site which as we said is updated every 24 hours.

Spanish government website (updated every 24 hours) The data can be automatically downloaded from the link below and will be saved in our dataframe called data.

14 cells hidden

Plotting

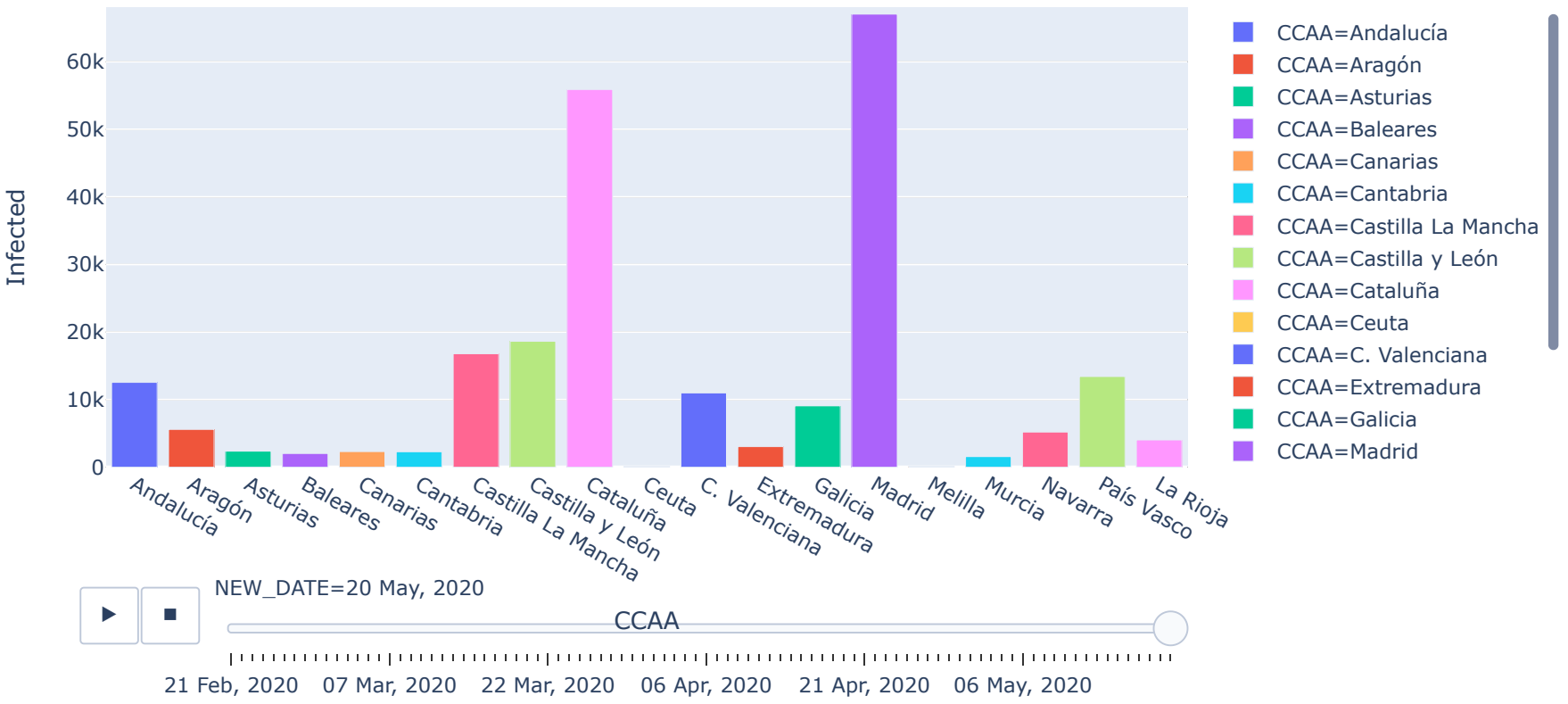
```
aux = total_s.melt(id_vars="Date", value_vars=("Cases","Infected","TestAc+","Deaths","ICU","Hospitalized"), value_name="Cc
aux.head()
```

	Date	Description	Count
0	2020-02-20	Cases	0.0
1	2020-02-21	Cases	0.0
2	2020-02-22	Cases	0.0
3	2020-02-23	Cases	0.0
4	2020-02-24	Cases	0.0

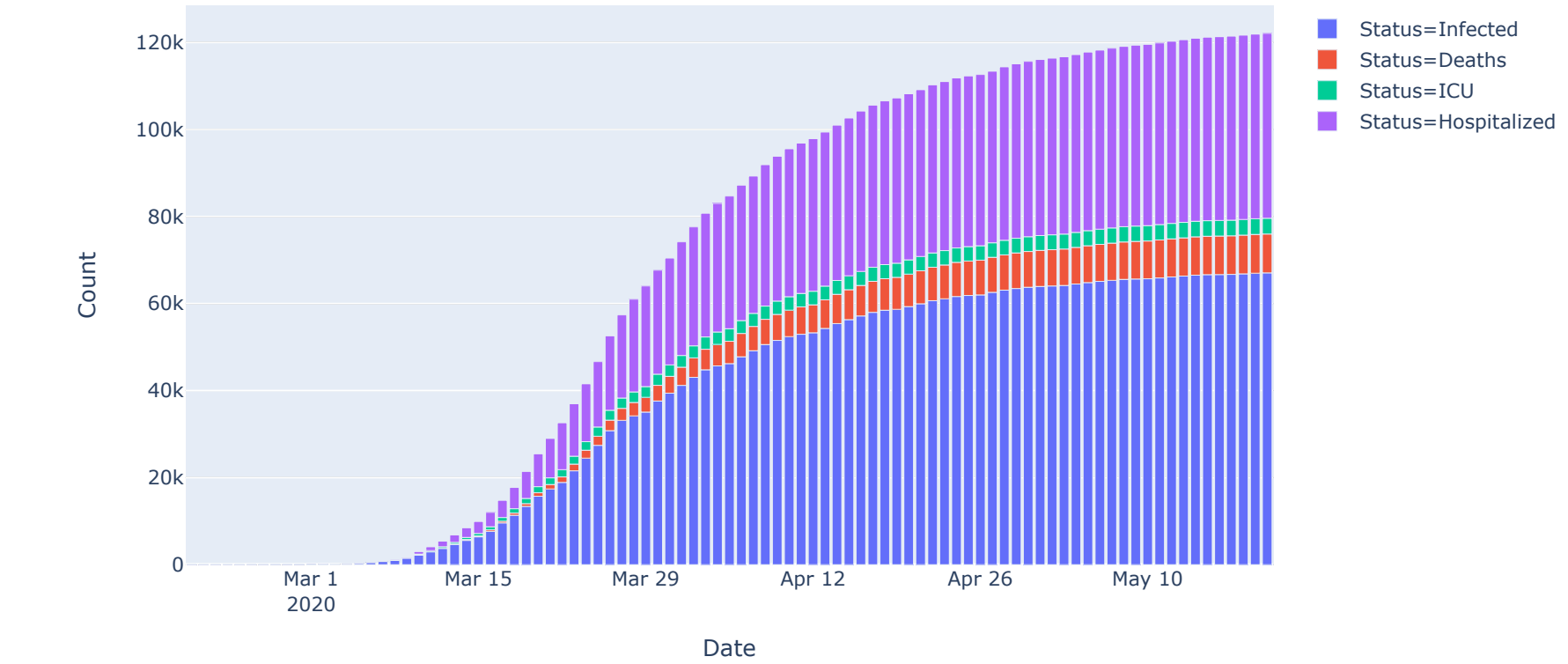
```
data_infected = data[data.Date>"20-02-2020"]
```

Infections over time

```
fig = px.bar(data_infected, x="CCAA", y="Infected", color="CCAA",
             animation_frame="NEW_DATE", animation_group="CCAA", range_y=[0,data.Infected.max()+1000],title= "Infections
fig.show()
```



```
total_madrid = data[data.CCAA=="Madrid"].groupby("Date")["Date","Infected","Deaths","Hospitalized","ICU"].sum().reset_inde

aux_m = total_madrid.melt(id_vars="Date", value_vars=("Infected","Deaths","ICU","Hospitalized"), value_name="Count" , var_

fig = px.bar(aux_m, x= "Date", y = "Count", color="Status", title= "Actual situation in Madrid")
fig.show()
```

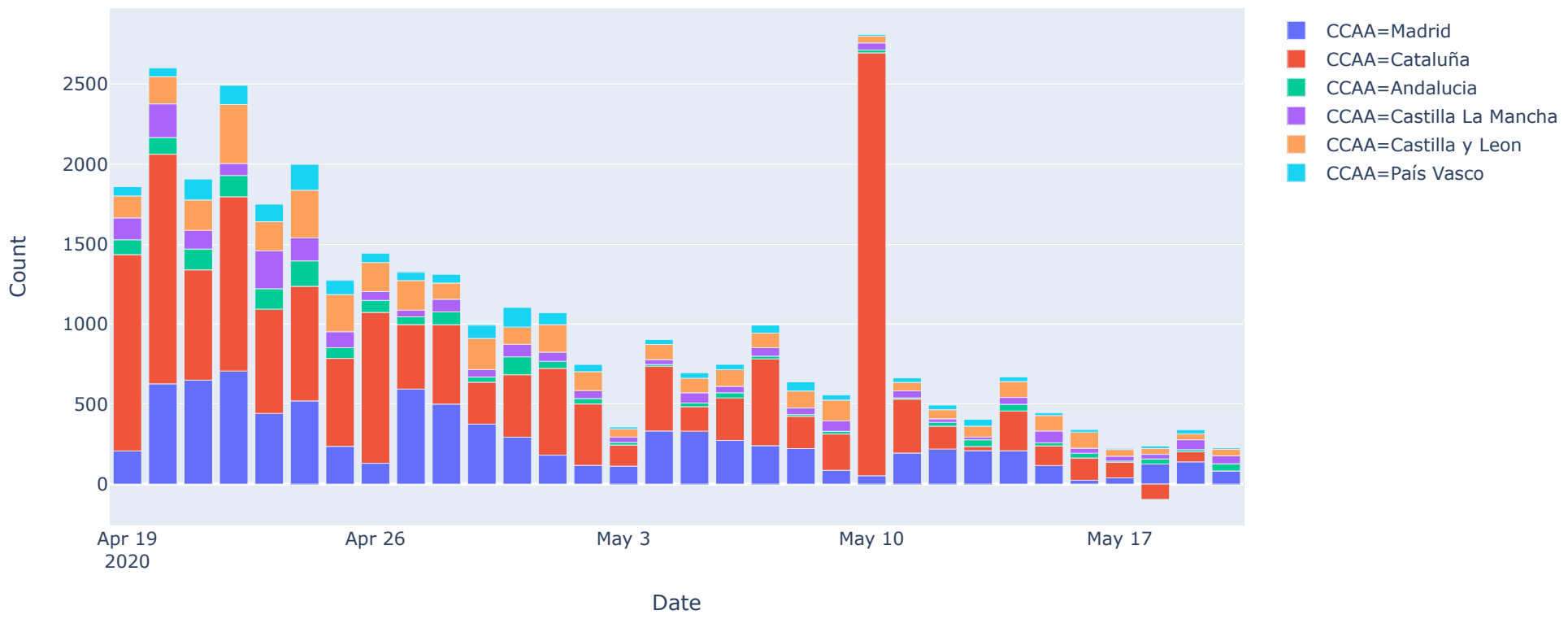



```
for i in data.CCAA.unique():

    a = i.replace(".","")
    a = a.replace(" ","_")

    exec('df_{}=data[data.CCAA == i]'.format(a))

    exec('aux_a = df_{}.Infected.to_list()'.format(a))


    daily=[]
    for i in range(len(aux_a)-1):
        b = aux_a[i+1] - aux_a[i]
        daily.append(b)

    daily.insert(0,0)

    exec('df_{}["Daily_infected"] = daily'.format(a))

    exec('aux_d = df_{}.Deaths.to_list()'.format(a))

    daily=[]
    for i in range(len(aux_d)-1):
        b = aux_d[i+1] - aux_d[i]
        daily.append(b)

    daily.insert(0,0)

    exec('df_{}["Daily_deaths"] = daily'.format(a))



df_daily_infected = pd.DataFrame({"Date":data.Date.unique(),
                                  "Madrid":df_Madrid["Daily_infected"].values,
                                  "Cataluña":df_Cataluña["Daily_infected"].values,
                                  "Andalucia":df_Andalucía["Daily_infected"].values,
                                  "Castilla La Mancha":df_Castilla_La_Mancha["Daily_infected"].values,
                                  "Castilla y Leon":df_Castilla_y_León["Daily_infected"].values,
                                  "País Vasco":df_País_Vasco["Daily_infected"].values})


aux_i = df_daily_infected.melt(id_vars="Date", value_vars=("Madrid","Cataluña","Andalucia","Castilla La Mancha","Castilla
```

Daily infections in Top 6 countries

```
aux 1=aux i[aux i.Date>"20-02-2020"]
```

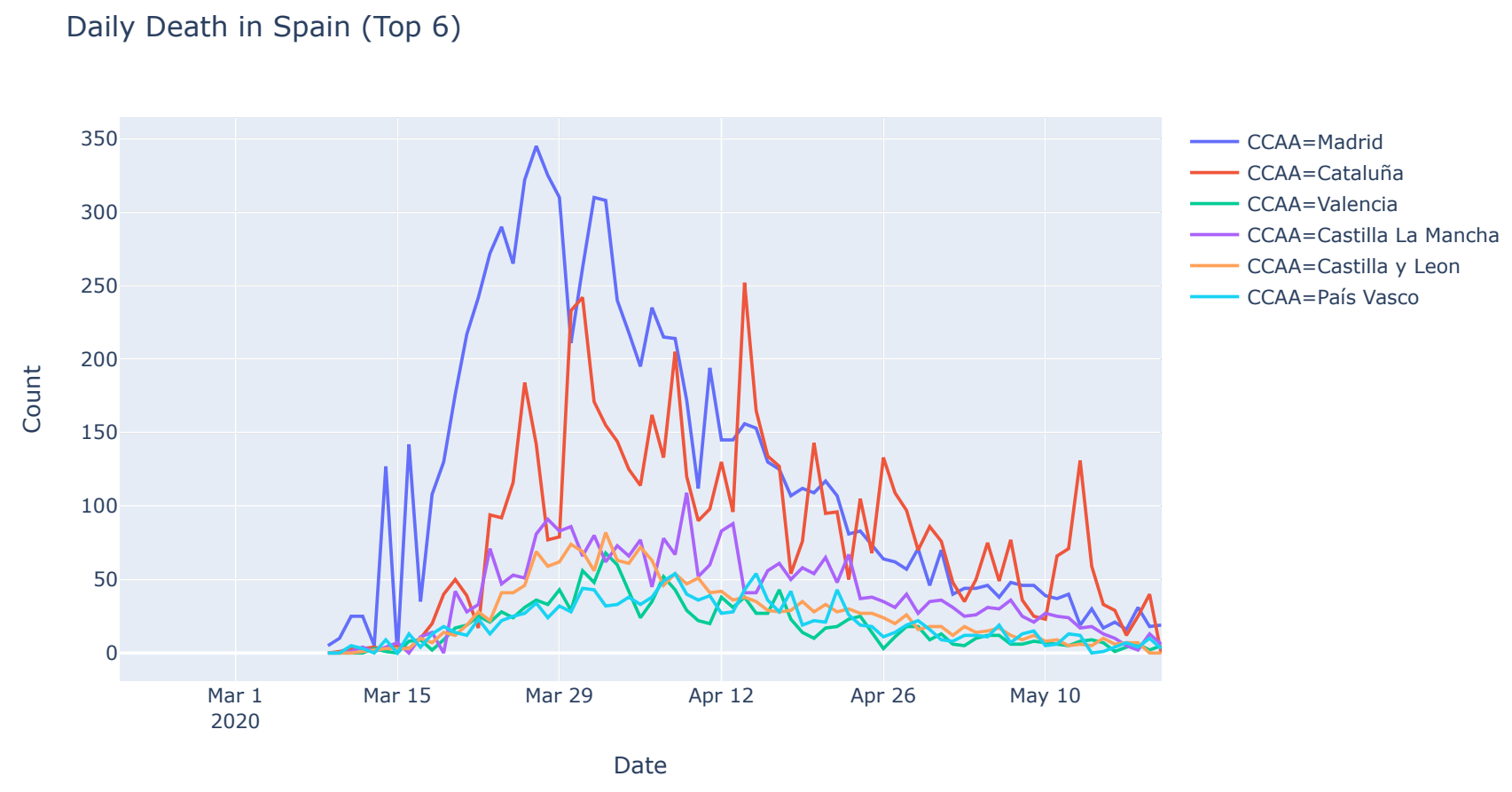
```
aux_2=aux_i[aux_i.Date>"18-04-2020"]
fig = px.bar (aux_2, x= "Date", y = "Count", color="CCAA", title= "Daily infections in Spain (Top 6)")
fig.show()
```



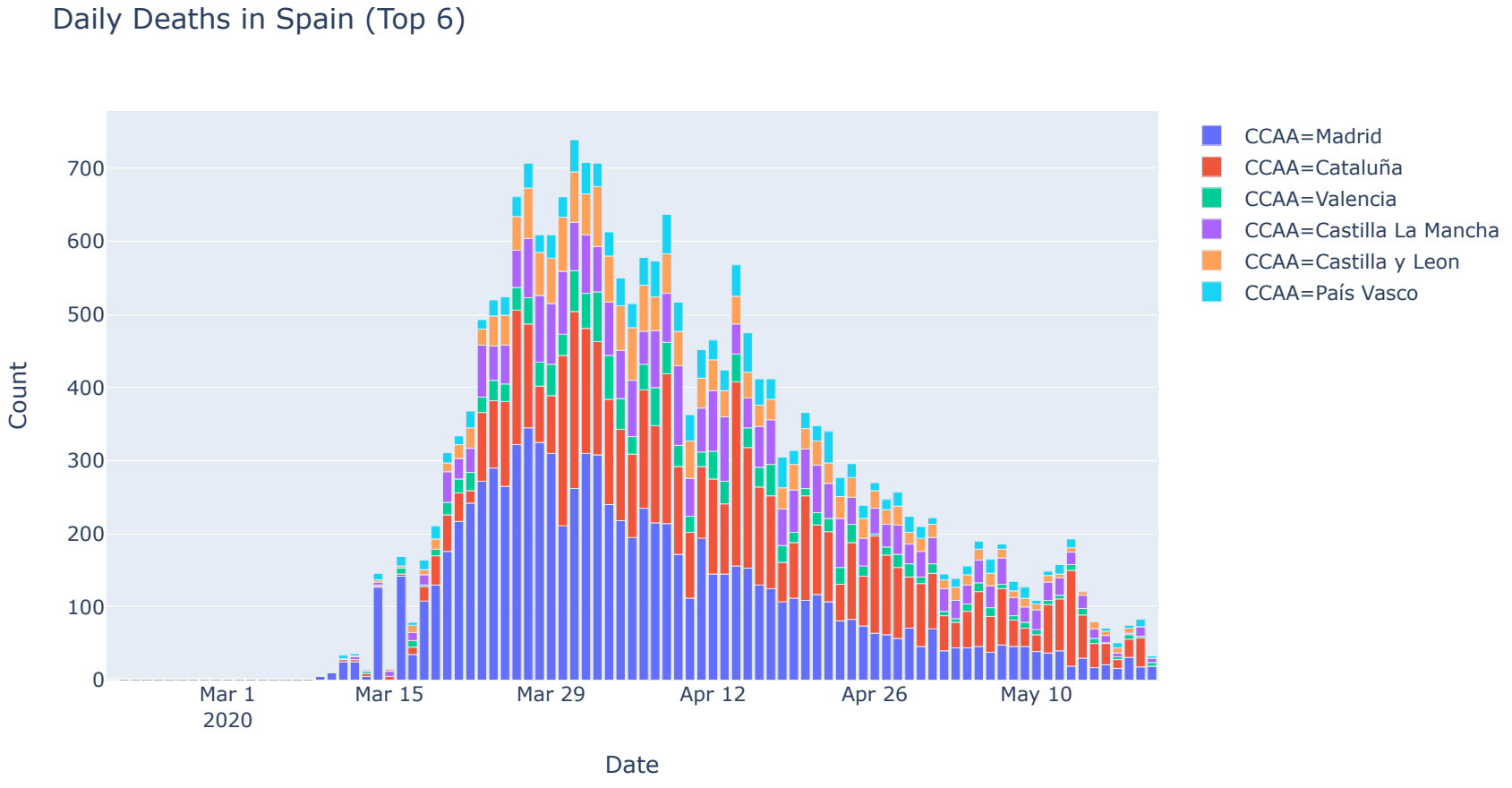
Daily deaths in Top 6 countries

```
df_daily_fatalities = pd.DataFrame({"Date":data.Date.unique(),
                                    "Madrid":df_Madrid["Daily_deaths"].values,
                                    "Cataluña":df_Cataluña["Daily_deaths"].values,
                                    "Valencia":df_C_Valenciana["Daily_deaths"].values,
                                    "Castilla La Mancha":df_Castilla_La_Mancha["Daily_deaths"].values,
                                    "Castilla y Leon":df_Castilla_y_León["Daily_deaths"].values,
                                    "País Vasco":df_País_Vasco["Daily_deaths"].values})


aux_f = df_daily_fatalities.melt(id_vars="Date", value_vars=("Madrid","Cataluña","Valencia","Castilla La Mancha","Castilla
```

```
fig = px.line (aux_f, x= "Date", y = "Count", color="CCAA", title= "Daily Death in Spain (Top 6)")
fig.show()
```




```
fig = px.bar (aux_f, x= "Date", y = "Count", color="CCAA", title= "Daily Deaths in Spain (Top 6)")
fig.show()
```



Mapping with chloropleth

```
d_name = {
'AN':'Andalucía',
'AR':'Aragón',
'AS':'Asturias',
'IB':'Baleares',
'CN':'Canarias',
'CB':'Cantabria',
'CM':'Castilla La Mancha',
'CL':'Castilla y León',
'CT':'Cataluña',
'CE':'Ceuta',
'VC':'C. Valenciana',
'EX':'Extremadura',
'GA':'Galicia',
'MD':'Madrid',
'ML':'Melilla',
'MC':'Murcia',
'NC':'Navarra',
'PV':'País Vasco',
'RI':'La Rioja'
}


d_ccaa = {
'Andalucía': 'Andalucía',
'Aragón': 'Aragón',
'Asturias': 'Principado de Asturias',
'Baleares': 'Islas Baleares',
'Canarias': 'Islas Canarias',
'Cantabria':'Cantabria',
'Castilla La Mancha': 'Castilla-La Mancha',
'Castilla y León': 'Castilla y León',
'Cataluña': 'Cataluña',
'Ceuta': 'Ceuta y Melilla',
'C. Valenciana': 'Comunidad Valenciana',
'Extremadura': 'Extremadura',
'Galicia': 'Galicia',
'Madrid': 'Comunidad de Madrid',
'Melilla': 'Ceuta y Melilla',
'Murcia': 'Región de Murcia',
'Navarra': 'Comunidad Foral de Navarra',
'País Vasco': 'País Vasco',
'La Rioja': 'La Rioja'
}


d_ccaa_id = {
'Andalucía': "1",
'Aragón' : "2",
'Principado de Asturias': "3",
'Islas Baleares': "4",
'Islas Canarias': "5",
'Cantabria': "6",
'Castilla-La Mancha': "7",
'Castilla y León': "8",
```

```
    'Asturias': 'Principado de Asturias',
    'Baleares': 'Islas Baleares',
    'Canarias': 'Islas Canarias',
    'Cantabria':'Cantabria',
    'Castilla La Mancha': 'Castilla-La Mancha',
    'Castilla y León': 'Castilla y León',
    'Cataluña': 'Cataluña',
    'Ceuta': 'Ceuta y Melilla',
    'C. Valenciana': 'Comunidad Valenciana',
    'Extremadura': 'Extremadura',
    'Galicia': 'Galicia',
    'Madrid': 'Comunidad de Madrid',
    'Melilla': 'Ceuta y Melilla',
    'Murcia': 'Región de Murcia',
    'Navarra': 'Comunidad Foral de Navarra',
    'País Vasco': 'País Vasco',
    'La Rioja': 'La Rioja'
    }


    d_ccaa_id = {
    'Andalucía': "1",
    'Aragón' : "2",
    'Principado de Asturias': "3",
    'Islas Baleares': "4",
    'Islas Canarias': "5",
    'Cantabria': "6",
    'Castilla-La Mancha': "7",
    'Castilla y León': "8",
    'Cataluña': "9",
    'Ceuta y Melilla': "10",
    'Comunidad Valenciana': "11",
    'Extremadura': "12",
    'Galicia': "13",
    'Comunidad de Madrid' : "14",
    'Ceuta y Melilla': "15",
    'Región de Murcia': "16",
    'Comunidad Foral de Navarra': "17",
    'País Vasco': "18",
    'La Rioja': "19"
    }


    d_ccaa_population = {
    'Andalucía': 8414240,
    'Aragón' : 1319291,
    'Principado de Asturias': 1022800,
    'Islas Baleares': 1149460,
    'Islas Canarias': 2153389,
    'Cantabria': 581078,
    'Castilla-La Mancha': 2032863,
    'Castilla y León': 2399548,
    'Cataluña': 7675217,
    'Ceuta y Melilla': 171264,
    'Comunidad Valenciana': 5003769,
    'Extremadura': 1067710,
    'Galicia': 2699499,
    'Comunidad de Madrid' : 6663394,
    'Ceuta y Melilla': 171264,
    'Región de Murcia': 1493898,
    'Comunidad Foral de Navarra': 654214,
    'País Vasco': 2207776,
    'La Rioja': 316798
    }


    def get_hex_colors(df, data_to_color, cmap = matplotlib.cm.Reds, log = False):

        '''
        This function takes the following arguments
            1. df:pandas DataFrame with the data.
            2. data_to_color: the column name with data based on which we want to create the color scale.
            3. cmap: colors you want to plot. You can use this to communicate different messages. For example: greens --> gooc
                    default is matplotlib.cm.Reds
                    more about colormaps: https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html
            3. log: if data has huge outliers, we can create the color map with a logarithic normalization. This way, the outl
                    default is False.

        '''
```

```
    cmap = cmap # define the color pallete you want. You can use Reds, Blues, Greens etc
    my_values = df[data_to_color] # get the value you wan to convert to colors

    mini = min(my_values) # get the min to normalize
    maxi= max(my_values) # get the max to normalize

    LOGMIN = 0.01 # arbitrary lower bound for log scale

    if log:
        norm = matplotlib.colors.LogNorm(vmin=max(mini,LOGMIN), vmax=maxi) # normalize log data
    else:
        norm = matplotlib.colors.Normalize(vmin=mini, vmax=maxi) # create a color range

    colors = {value:matplotlib.colors.rgb2hex(cmap(norm(value))[:3]) for value in sorted(list(set(my_values)))} # create a

    return colors


def get_hex_colors_2(value, cats):
    '''
    Color paletter used from this website:

    https://colorbrewer2.org/#type=sequential&scheme=Reds&n=9

    The color selection will be based on the percentile each value is in.
    '''
    if value == 0:
        return "#FFFFFF"
    elif value in cats[0]:
        return "#fff5f0"
    elif value in cats[1]:
        return "#fee0d2"
    elif value in cats[2]:
        return "#fcbba1"
    elif value in cats[3]:
        return "#fc9272"
    elif value in cats[4]:
        return "#fb6a4a"
    elif value in cats[5]:
        return "#ef3b2c"
    elif value in cats[6]:
        return "#cb181d"
    elif value in cats[7]:
        return "#a50f15"
    elif value in cats[8]:
        return "#67000d"
    else:
        return "#000000"


df = pd.read_csv("https://covid19.isciii.es/resources/serie_historica_acumulados.csv",delimiter=",", encoding="latin1", sk


df.rename(columns = {"FECHA":"DATE",
                     "CASOS":"CASES",
                      "PCR+":"TOTAL_INFECTED",
                     "Hospitalizados":"REQUIERED_HOSPITALIZATION",
                     "UCI":"REQUIERED_ADVANCED_CARE",
                     "Fallecidos":"TOTAL_DEATHS"}, inplace = True)

df.fillna(0, inplace = True)
df["CCAA"] = df["CCAA"].map(d_name)
df["CCAA_for_Folium"] = df["CCAA"].map(d_ccaa)
df["id"] = df["CCAA_for_Folium"].map(d_ccaa_id)

df["Population"] = df["CCAA_for_Folium"].map(d_ccaa_population)



df["CCAA"].isnull().sum()
```

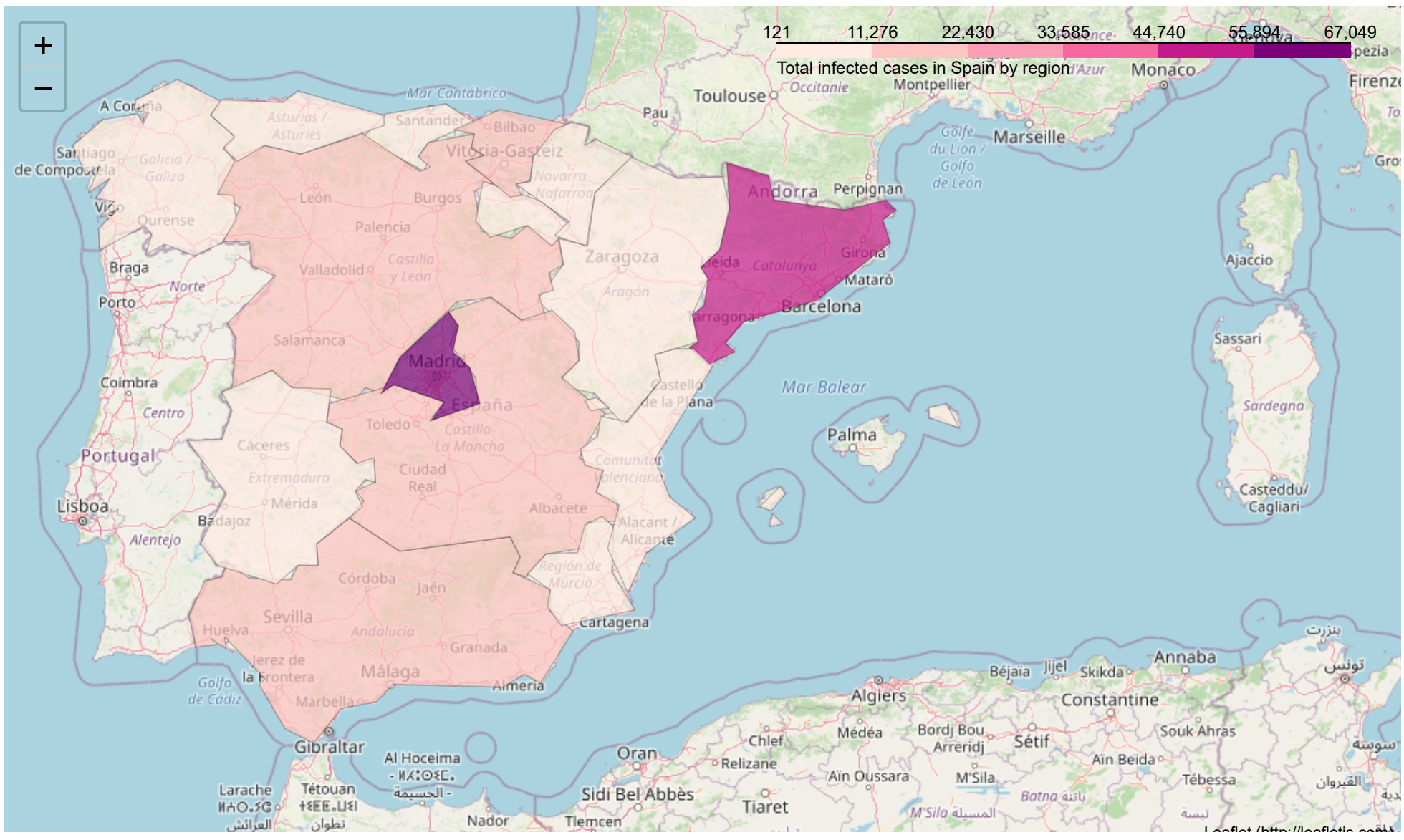
```
0
```

```
def correct_date(date_str):
    list_dates = date_str.split("/")
    day = list_dates[0]
    month = list_dates[1]
    year = list_dates[2]

    if len(day) == 1:
```

```
            day = "0" + day
        if len(month) == 1:
            month = "0" + month

        return "/".join([day, month, year])


    df["NEW_DATE"] = df["DATE"].apply(correct_date)


    df["DATE"] = pd.to_datetime(df["NEW_DATE"], format='%d/%m/%Y')

    df["DATE_for_Folium"] = (df["DATE"].astype(int)// 10**9).astype('U10')

    df = df[["id", "CCAA", "CCAA_for_Folium", "DATE", "DATE_for_Folium", "TOTAL_INFECTED", "REQUIERED_HOSPITALIZATION", "REQU
    df["id"].astype(np.int16)
    df.head()


    PATH_GEO_JSON = 'shapefiles_ccaa_espana.geojson'
    gdf = gpd.read_file(PATH_GEO_JSON)
    gdf["id"] = gdf["name_1"].map(d_ccaa_id) # create a numerical id for each ccaa
    gdf = gdf[["id", "shape_leng","shape_area","geometry"]] # extract the id and the geometry (coordinates of each ccaa)
    gdf["geometry"] = gdf["geometry"].simplify(0.1, preserve_topology = False)
    gdf["id"].astype(int)
    gdf.head()
```

Infections

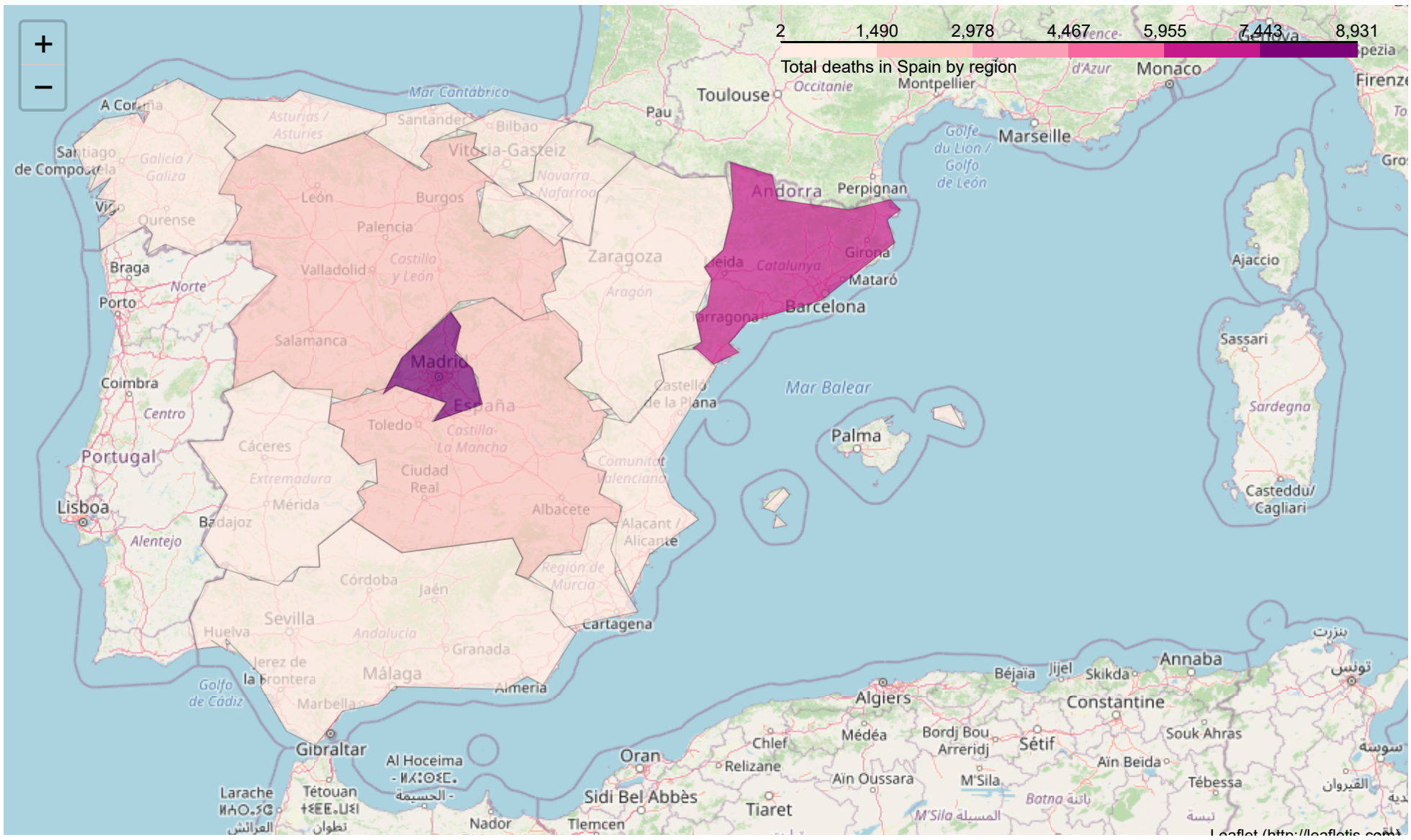
```
m = folium.Map(location = (40, 0), zoom_start = 5.5)

folium.Choropleth(
    geo_data = gdf,
    name = 'choropleth',
    data = df[df["DATE"] == max(df["DATE"])],
    columns = ['id', 'TOTAL_INFECTED'],
    key_on='feature.properties.id',
    fill_color='RdPu',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name = 'Total infected cases in Spain by region'
).add_to(m)

m
```



```
g = TimeSliderChoropleth(
    gdf.set_index("id").to_json(), # get's the coordinates for each id
    styledict = styledict # styledict contains for each id the timestamp and the color to plot.
)

m.add_child(g)

#-------------------------------------------------------------------------------------
# Let's create a legend for folium
# https://nbviewer.jupyter.org/gist/talbertc-usgs/18f8901fc98f109f2b71156cf3ac81cd

from branca.element import Template, MacroElement

template = """
{% macro html(this, kwargs) %}

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>jQuery UI Draggable - Default functionality</title>
  <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">

  <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

  <script>
  $( function() {
    $( "#maplegend" ).draggable({
                    start: function (event, ui) {
                        $(this).css({
                            right: "auto",
                            top: "auto",
                            bottom: "auto"
                        });
                    }
                });
});

  </script>
</head>
<body>


<div id='maplegend' class='maplegend'
    style='position: absolute; z-index:9999; border:2px solid grey; background-color:rgba(255, 255, 255, 0.8);
     border-radius:6px; padding: 10px; font-size:14px; right: 20px; bottom: 20px;'>

<div class='legend-title'>Legend</div>
<div class='legend-scale'>
  <ul class='legend-labels'>
    <li><span style='background:#FFFFFF;opacity:0.6;'></span>No cases</li>
    <li><span style='background:#fff5f0;opacity:0.6;'></span>1 Quantile</li>
    <li><span style='background:#fee0d2;opacity:0.6;'></span>2 Quantile</li>
    <li><span style='background:#fcbba1;opacity:0.6;'></span>3 Quantile</li>
    <li><span style='background:#fc9272;opacity:0.6;'></span>4 Quantile</li>
    <li><span style='background:#fb6a4a;opacity:0.6;'></span>5 Quantile</li>
    <li><span style='background:#ef3b2c;opacity:0.6;'></span>6 Quantile</li>
    <li><span style='background:#cb181d;opacity:0.6;'></span>7 Quantile</li>
    <li><span style='background:#a50f15;opacity:0.6;'></span>8 Quantile</li>
    <li><span style='background:#67000d;opacity:0.6;'></span>9 Quantile</li>
    <li><span style='background:#000000;opacity:0.6;'></span>Other</li>
  </ul>
</div>
</div>

</body>
</html>

<style type='text/css'>
  .maplegend .legend-title {
    text-align: left;
    margin-bottom: 5px;
    font-weight: bold;
    font-size: 90%;
    }
  .maplegend .legend-scale ul {
    margin: 0;
```

Deaths

```
m = folium.Map(location = (40, 0), zoom_start = 5.5)

folium.Choropleth(
    geo_data = gdf,
    name = 'choropleth',
    data = df,
    columns = ['id', 'TOTAL_DEATHS'],
    key_on='feature.properties.id',
    fill_color='RdPu',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name = 'Total deaths in Spain by region'
).add_to(m)

m
```

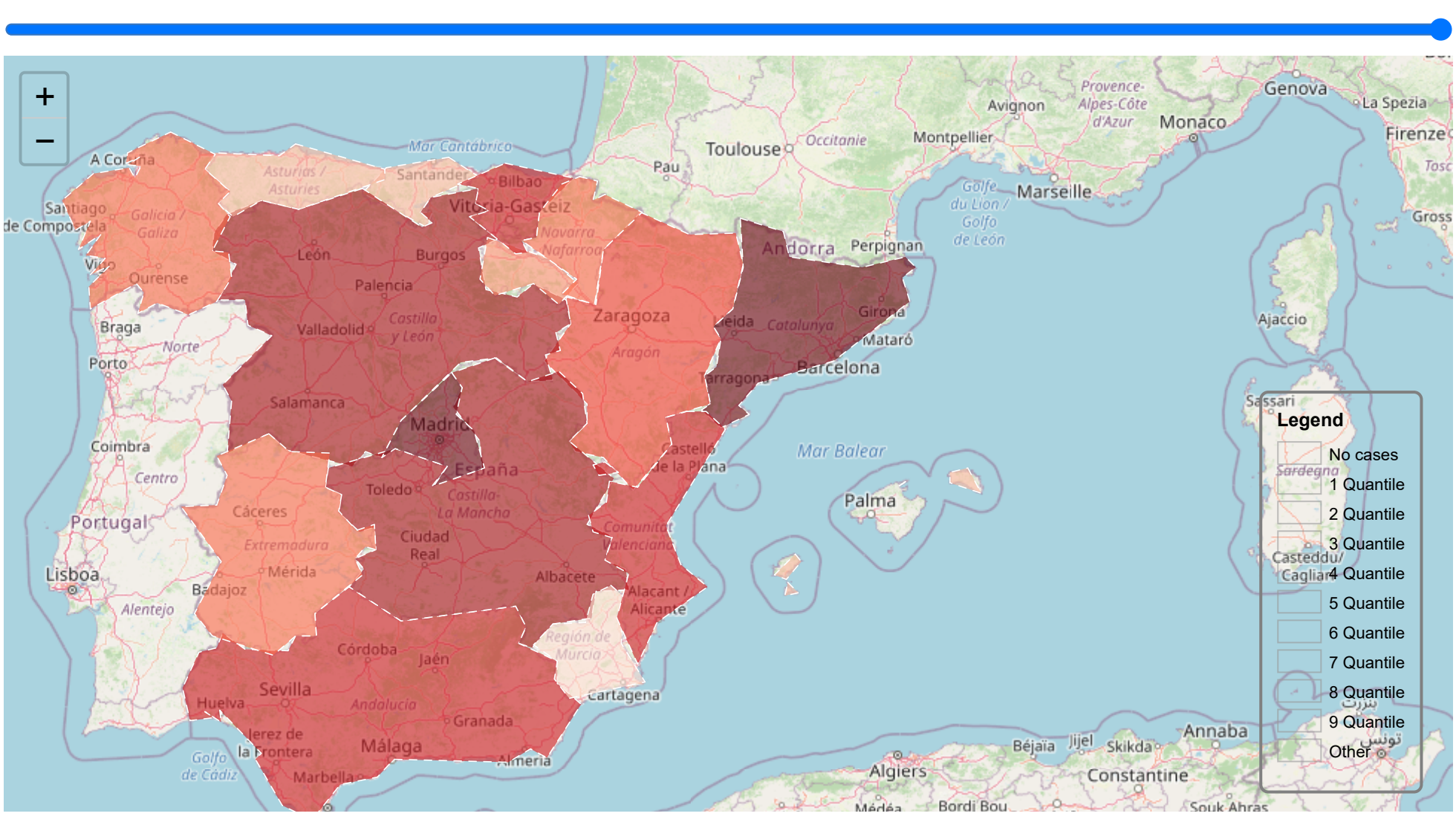


Deaths time slider

```
data_to_color = "TOTAL_DEATHS"
cats, bins =  pd.qcut(df[data_to_color].unique()[np.argsort(df[data_to_color].unique())], q = 9, retbins = True)
cats = cats.unique()

#--------------------------------------------------------------------------------

# value we will iterate in order to create the styledict
ccaas = list(df["id"].unique())
dates = list(df["DATE_for_Folium"].unique())

# create the color dict and color column
df["COLORS"] = df[data_to_color].apply(get_hex_colors_2, args = [cats]) # we create a colum in the df so that we can itera

# creates the styledict for the map
styledict = {}

# iterate the populate the styledict
for ccaa in ccaas:
    styledict[str(ccaa)] = {date: {'color': df[(df["id"] == ccaa) & (df["DATE_for_Folium"] == date)]["COLORS"].values[0],
                                   'opacity': 0.6} for date in dates}

# creates and renders the Folium map
m = folium.Map(location=(40, 0), tiles='OpenStreetMap', zoom_start=6)
```

```
    margin-bottom: 5px;
    padding: 0;
    float: left;
    list-style: none;
    }
  .maplegend .legend-scale ul li {
    font-size: 80%;
    list-style: none;
    margin-left: 0;
    line-height: 18px;
    margin-bottom: 2px;
    }
  .maplegend ul.legend-labels li span {
    display: block;
    float: left;
    height: 16px;
    width: 30px;
    margin-right: 5px;
    margin-left: 0;
    border: 1px solid #999;
    }
  .maplegend .legend-source {
    font-size: 80%;
    color: #777;
    clear: both;
    }
  .maplegend a {
    color: #777;
    }
</style>
{% endmacro %}"""

macro = MacroElement()
macro._template = Template(template)

m.get_root().add_child(macro)
```

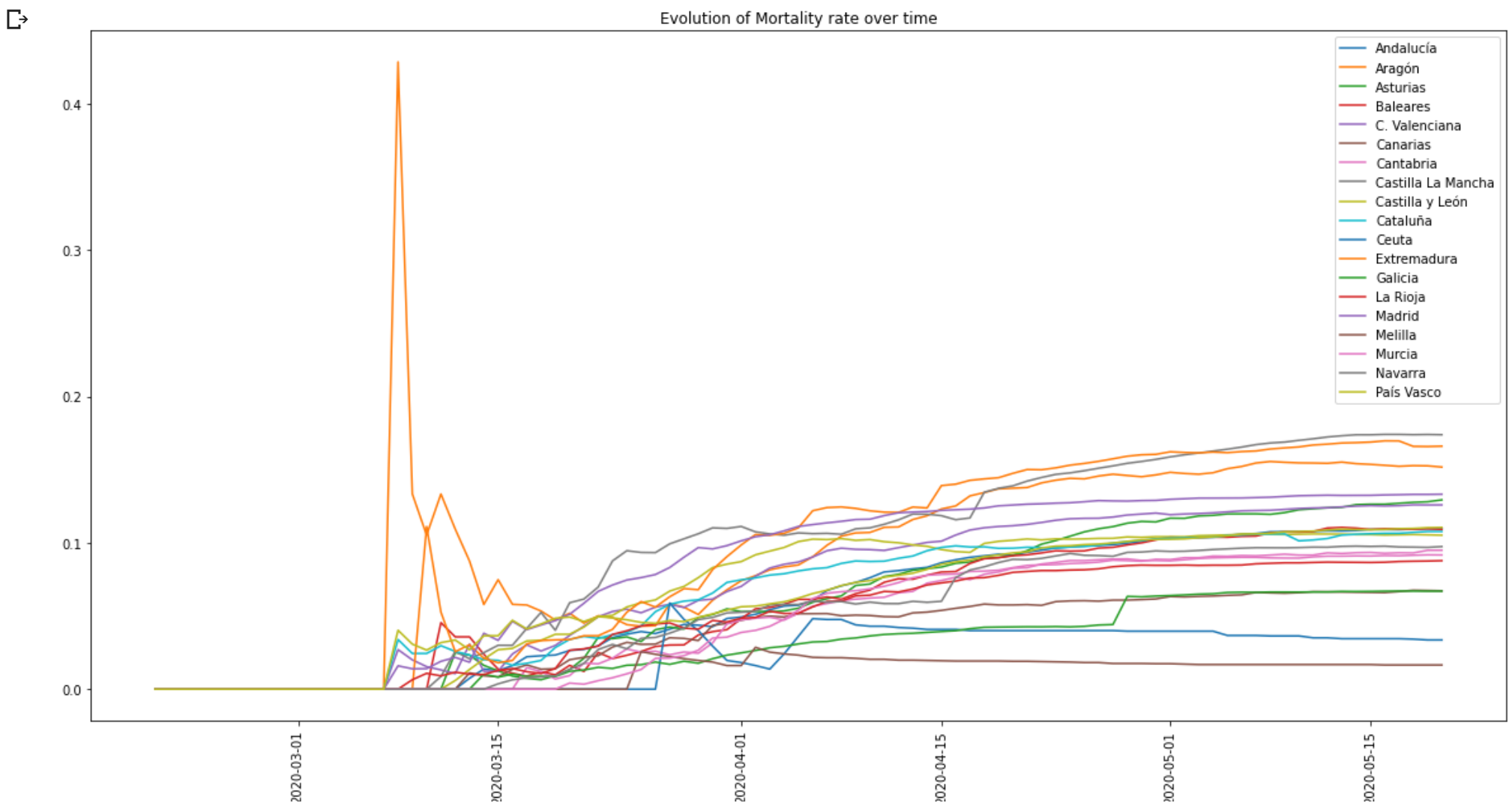
Make this Notebook Trusted to load map: File -> Trust Notebook



Mortality rate

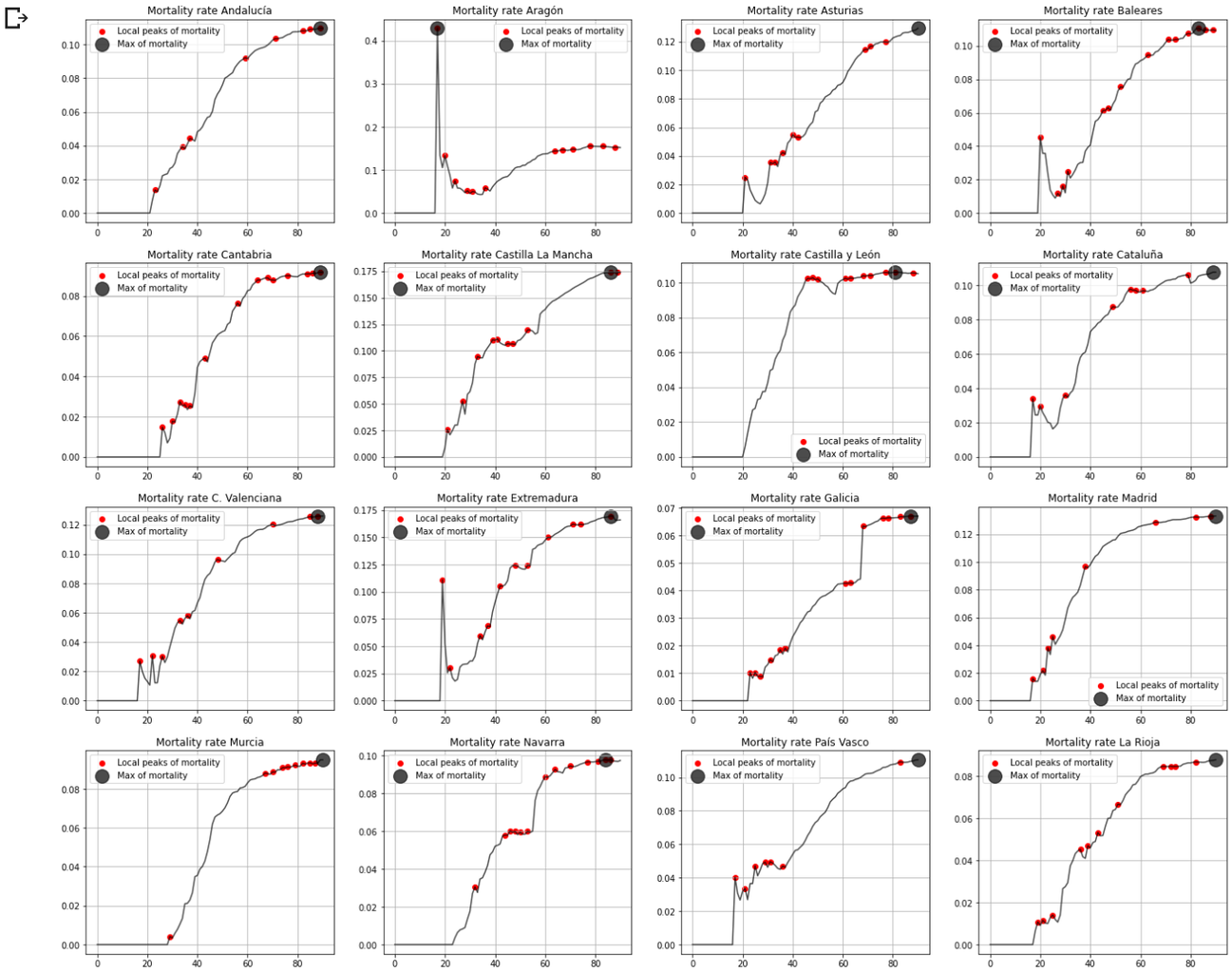
```
df["Infected_1000h"] = df["TOTAL_INFECTED"]/(df["Population"]/1000)
df["Mortality_rate"] = df["TOTAL_DEATHS"] / df["TOTAL_INFECTED"]
df.fillna(0, inplace = True)
df.tail()
```

```
plt.figure(figsize = (20, 10))

for ccaa in sorted(list(df["CCAA"].unique())):

    x = df["DATE"].unique()
    y = df[df["CCAA"] == ccaa]["Mortality_rate"]

    plt.plot(x, y, label = ccaa)
    plt.title("Evolution of Mortality rate over time")
    plt.legend()
    plt.xticks(rotation=90)
```




```
x = [day for day in range(len(df["DATE"].unique()))]

fig, axes = plt.subplots(nrows=4, ncols=5, figsize=(30,20))
# plt.setp(axes, ylim=(0 ,max(df["Mortality rate"])))
ccaas = list(df["CCAA"].unique())

i = 0

for col_axes in axes:
    for ax in col_axes:
        if i < len(ccaas):
            ccaa = ccaas[i]
            y = df[df["CCAA"] == ccaa]["Mortality_rate"].values
            ipeaks, _ = find_peaks(y)

            ax.plot(x, y, color = "k", alpha = 0.7)
            ax.scatter(ipeaks, np.array(y)[ipeaks], color = "red", label = "Local peaks of mortality")
            ax.scatter(x[list(y).index(np.max(y))], np.max(y), color = "k", marker = "o", alpha = 0.7, s = 250, label = "M

            ax.set_title("Mortality rate {}".format(ccaa))
            ax.legend()
            ax.grid()
            i += 1

fig.delaxes(axes[3, 4])
```

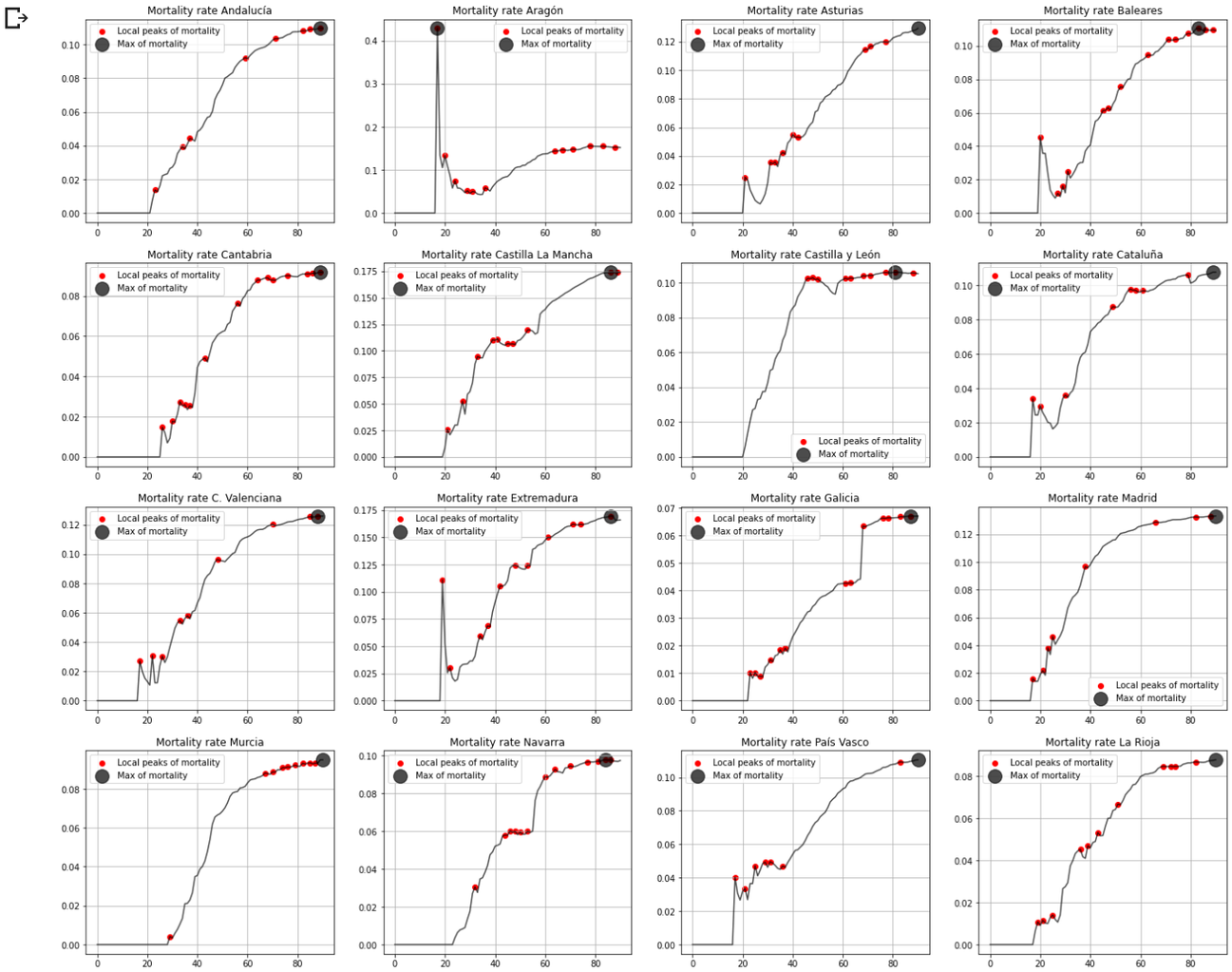
```
            ax.plot(x, y, color = "k", alpha = 0.7)
            ax.scatter(ipeaks, np.array(y)[ipeaks], color = "red", label = "Local peaks of mortalit
            ax.scatter(x[list(y).index(np.max(y))], np.max(y), color = "k", marker = "o", alpha = 0

            ax.set_title("Mortality rate {}".format(ccaa))
            ax.legend()
            ax.grid()
            i += 1

fig.delaxes(axes[3, 4])
```



create the pivot table of total cases

```
total_df = df.set_index("DATE").resample("D")[["TOTAL_INFECTED", "REQUIERED_HOSPITALIZATION", "REQU
total_df = total_df[total_df["Population"] > 0]
total_df["TOTAL_INFECTED_1000H"] = total_df["TOTAL_INFECTED"]/(total_df["Population"]/1000)
total_df["TOTAL_DEATHS_1000H"] = total_df["TOTAL_DEATHS"]/(total_df["Population"]/1000)

# get the data
x = list(total_df.index)
y_1 = list(total_df["TOTAL_INFECTED_1000H"]) # 1 axis
y_2 = list(total_df["TOTAL_DEATHS_1000H"]) # 2 axis

# create the figures
fig, ax = plt.subplots(figsize = (15, 7))
plot1 = ax.plot(x, y_1, color = "r", label = "Total infected per 1000 habitants") # plot the first
plt.xticks(rotation=90) # rotate the date

ax2 = ax.twinx() # create a secondary axis
plot2 = ax2.plot(x, y_2, color = "k", label = "Total deaths per 1000 habitants") # plot the second
fig.tight_layout()
plt.title("Evolution of total infected cases and total deaths per 1000 habitants")

# create a common legend
lns = plot1 + plot2
labs = [l.get_label() for l in lns]
```

```
        ax.plot(x, y, color = "k", alpha = 0.7)
        ax.scatter(ipeaks, np.array(y)[ipeaks], color = "red", label = "Local peaks of mortalit
        ax.scatter(x[list(y).index(np.max(y))], np.max(y), color = "k", marker = "o", alpha = 0

        ax.set_title("Mortality rate {}".format(ccaa))
        ax.legend()
        ax.grid()
        i += 1

fig.delaxes(axes[3, 4])
```

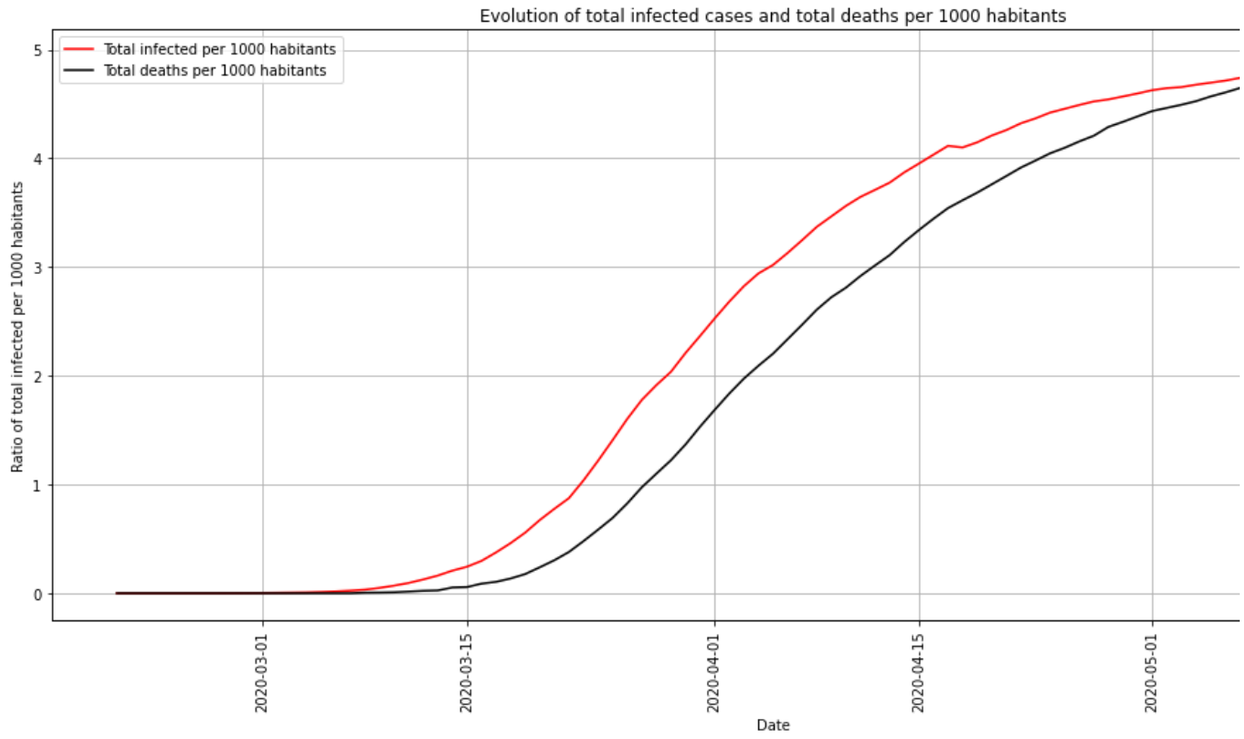


create the pivot table of total cases

```
total_df = df.set_index("DATE").resample("D")[["TOTAL_INFECTED", "REQUIERED_HOSPITALIZATION", "REQU
total_df = total_df[total_df["Population"] > 0]
total_df["TOTAL_INFECTED_1000H"] = total_df["TOTAL_INFECTED"]/(total_df["Population"]/1000)
total_df["TOTAL_DEATHS_1000H"] = total_df["TOTAL_DEATHS"]/(total_df["Population"]/1000)

# get the data
x = list(total_df.index)
y_1 = list(total_df["TOTAL_INFECTED_1000H"]) # 1 axis
y_2 = list(total_df["TOTAL_DEATHS_1000H"]) # 2 axis

# create the figures
fig, ax = plt.subplots(figsize = (15, 7))
plot1 = ax.plot(x, y_1, color = "r", label = "Total infected per 1000 habitants") # plot the first
plt.xticks(rotation=90) # rotate the date

ax2 = ax.twinx() # create a secondary axis
plot2 = ax2.plot(x, y_2, color = "k", label = "Total deaths per 1000 habitants") # plot the second
fig.tight_layout()
plt.title("Evolution of total infected cases and total deaths per 1000 habitants")

# create a common legend
lns = plot1 + plot2
labs = [l.get_label() for l in lns]
```

```
ax.legend(lns, labs, loc=0)

# prettify
ax.grid()
ax.set_xlabel("Date")
ax.set_ylabel("Ratio of total infected per 1000 habitants")
ax2.set_ylabel("Ratio of total deaths per 1000 habitants")
```

```
Text(998.875, 0.5, 'Ratio of total deaths per 1000 habitants')
```



8 of March manifestation

On 8 of March, the goverment allowed a massive manifestationa all over the country for the international w

By that time, there where already some informed cases, but, we only know what we know. Now, since the vi days to incubate in your body, this means that a person who was on the manifestation and begin to have sy likely that was already infected or got infected. Let's make this shift in infected cases and see the result.

As you can see, when the manifestation was held, it is likely that there was around between 10k and 30k n than the official 1006 cases.

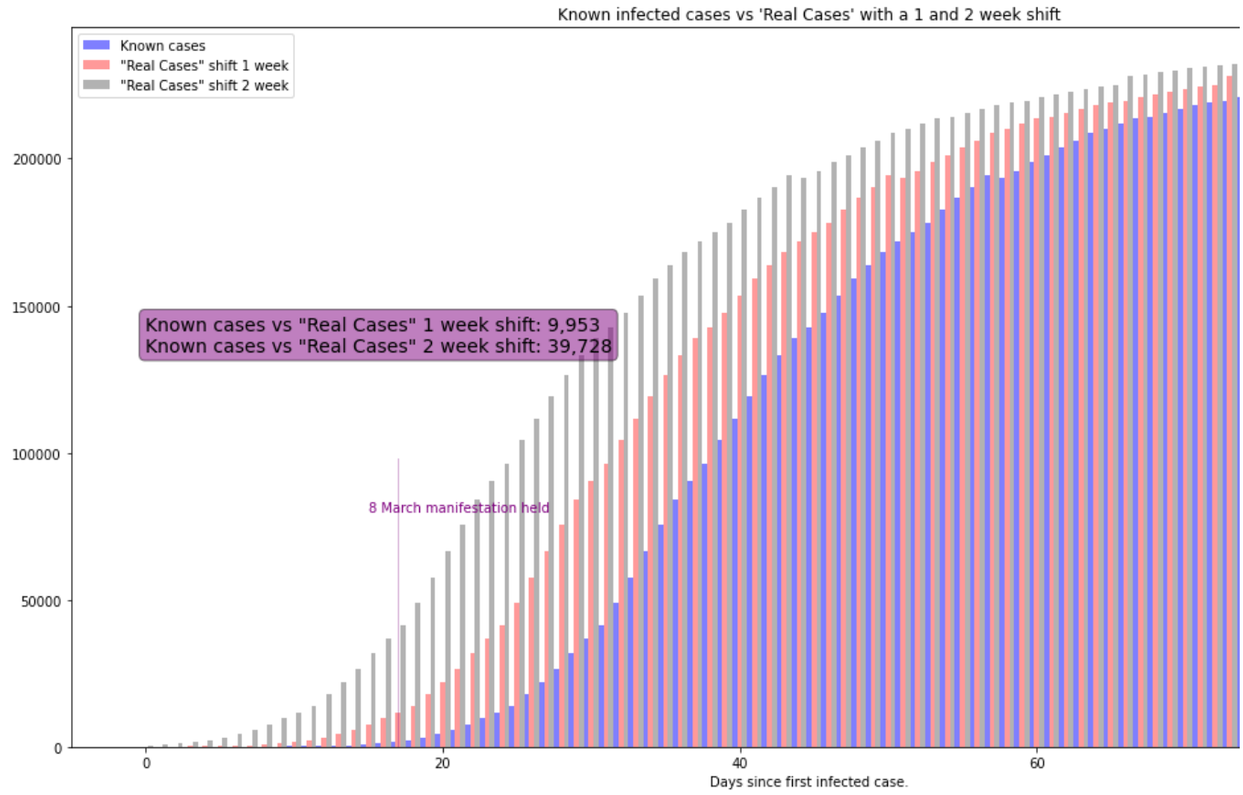
```
total_df["SHIFT_7_DAYS"] = total_df["TOTAL_INFECTED"].shift(-7)
total_df["SHIFT_14_DAYS"] = total_df["TOTAL_INFECTED"].shift(-14)
```

Known infected cases vs 'Real Cases' with a 1 and 2 week shift

```
x = np.array([x for x in range(len(total_df.index))])
y_informed = total_df["TOTAL_INFECTED"]
y_real_7_days = total_df["SHIFT_7_DAYS"]
y_real_14_days = total_df["SHIFT_14_DAYS"]
width = np.min(np.diff(x))/3

fig = plt.figure(figsize = (20, 10))

ax = fig.add_subplot(111)
ax.bar(x - width, y_informed, width, color = 'b', label = 'Known cases', alpha = 0.5)
ax.bar(x, y_real_7_days, width, color = 'r', label = '"Real Cases" shift 1 week', alpha = 0.4)
ax.bar(x + width, y_real_14_days, width, color='k', label = '"Real Cases" shift 2 week', alpha = 0.
ax.set_xlabel('Days since first infected case.')

plt.title("Known infected cases vs 'Real Cases' with a 1 and 2 week shift")
plt.axvline(x=17, lw = 1, alpha = 0.3, ymax = 0.4, color = "purple")
```

```
plt.annotate("8 March manifestation held", xy= (15, 80000), color = "purple")

textstr = '\n'.join((
    r'Known cases vs "Real Cases" 1 week shift: {:,.0f}'.format(total_df.iloc[17]["SHIFT_7_DAYS"] -
    r'Known cases vs "Real Cases" 2 week shift: {:,.0f}'.format(total_df.iloc[17]["SHIFT_14_DAYS"]

props = dict(boxstyle='round', facecolor='purple', alpha=0.5)

# place a text box in upper left in axes coords
ax.text(0.05, 0.6, textstr, transform=ax.transAxes, fontsize=14,
        verticalalignment='top', bbox=props)

plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa53a6eaeb8>
```

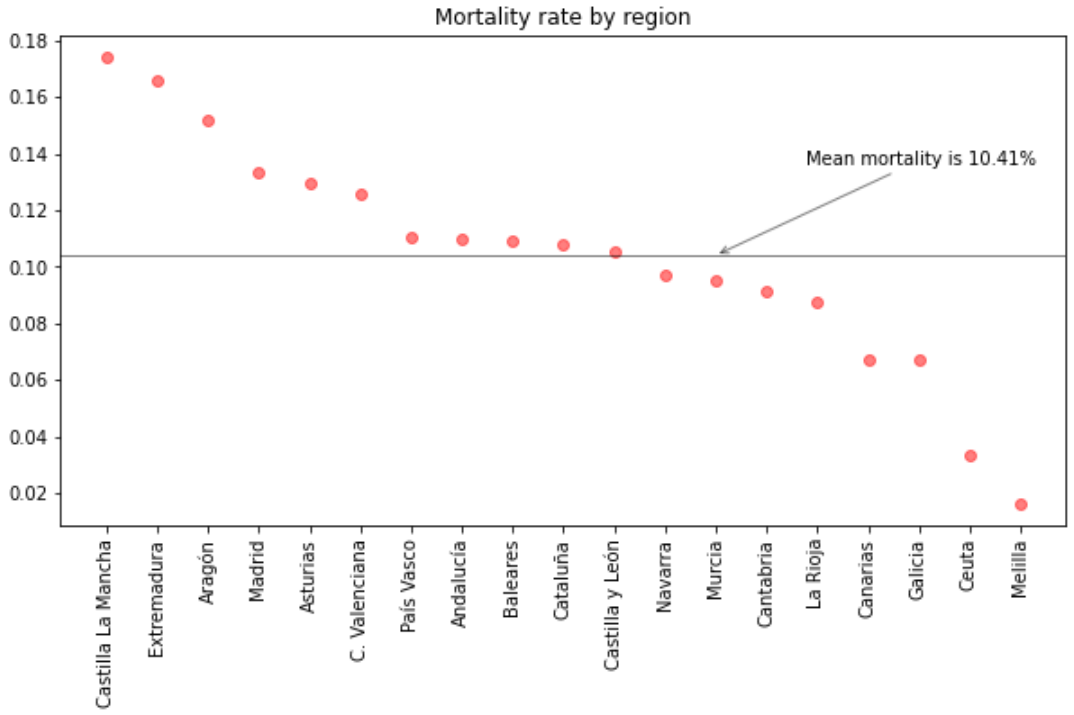


Mortality rate by region

```
short_df = df[df["DATE"] == max(df["DATE"])][["CCAA", "Mortality_rate"]].sort_values("Mortality_rat
x = short_df["CCAA"]
y = short_df["Mortality_rate"]

mean_y = np.mean(y)
mean_y

plt.figure(figsize = (10, 5))
plt.scatter(x, y, c= "red", alpha = 0.5)
plt.title("Mortality rate by region")

plt.xticks(rotation=90)
plt.axhline(mean_y, c = "k", alpha = 0.5, lw = 1)
plt.annotate('Mean mortality is {}%'.format(round(mean_y * 100, 2)),
             xy=(12, mean_y),
             xycoords='data',
             xytext=(50, 50),
             textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color = "k", alpha = 0.5),
             color = "k")
```

```
x = short_df["CCAA"]
y = short_df["Mortality_rate"]

mean_y = np.mean(y)
mean_y

plt.figure(figsize = (10, 5))
plt.scatter(x, y, c= "red", alpha = 0.5)
plt.title("Mortality rate by region")

plt.xticks(rotation=90)
plt.axhline(mean_y, c = "k", alpha = 0.5, lw = 1)
plt.annotate('Mean mortality is {}%'.format(round(mean_y * 100, 2)),
             xy=(12, mean_y),
             xycoords='data',
             xytext=(50, 50),
             textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color = "k", alpha = 0.5),
             color = "k")
```

```
Text(50, 50, 'Mean mortality is 10.41%')
```



General Analysis over the time

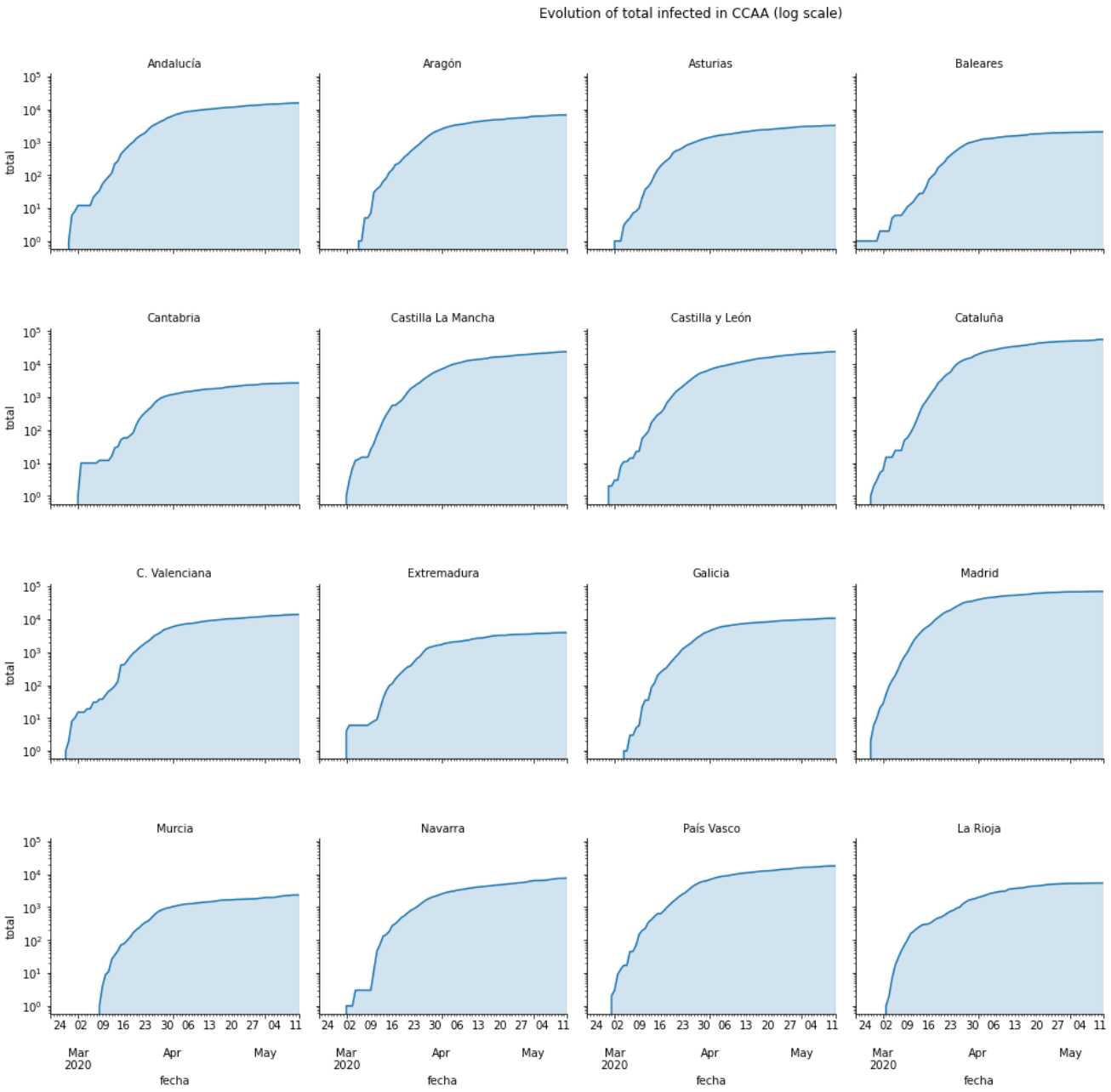
Data Load

6 cells hidden

Infected over time

```
infected = infected[infected['CCAA']!= 'Total']
g = sns.FacetGrid(infected, col="CCAA", col_wrap=5, height=3.5)
g = g.map_dataframe(dateplot, "fecha", "total").set(yscale='log')
g = g.map(plt.fill_between, 'fecha', 'total', alpha=0.2).set_titles("{col_name} CCAA")
g = g.set_titles("{col_name}")
```

```
plt.subplots_adjust(top=0.92)
g = g.fig.suptitle('Evolution of total infected in CCAA (log scale)')
```

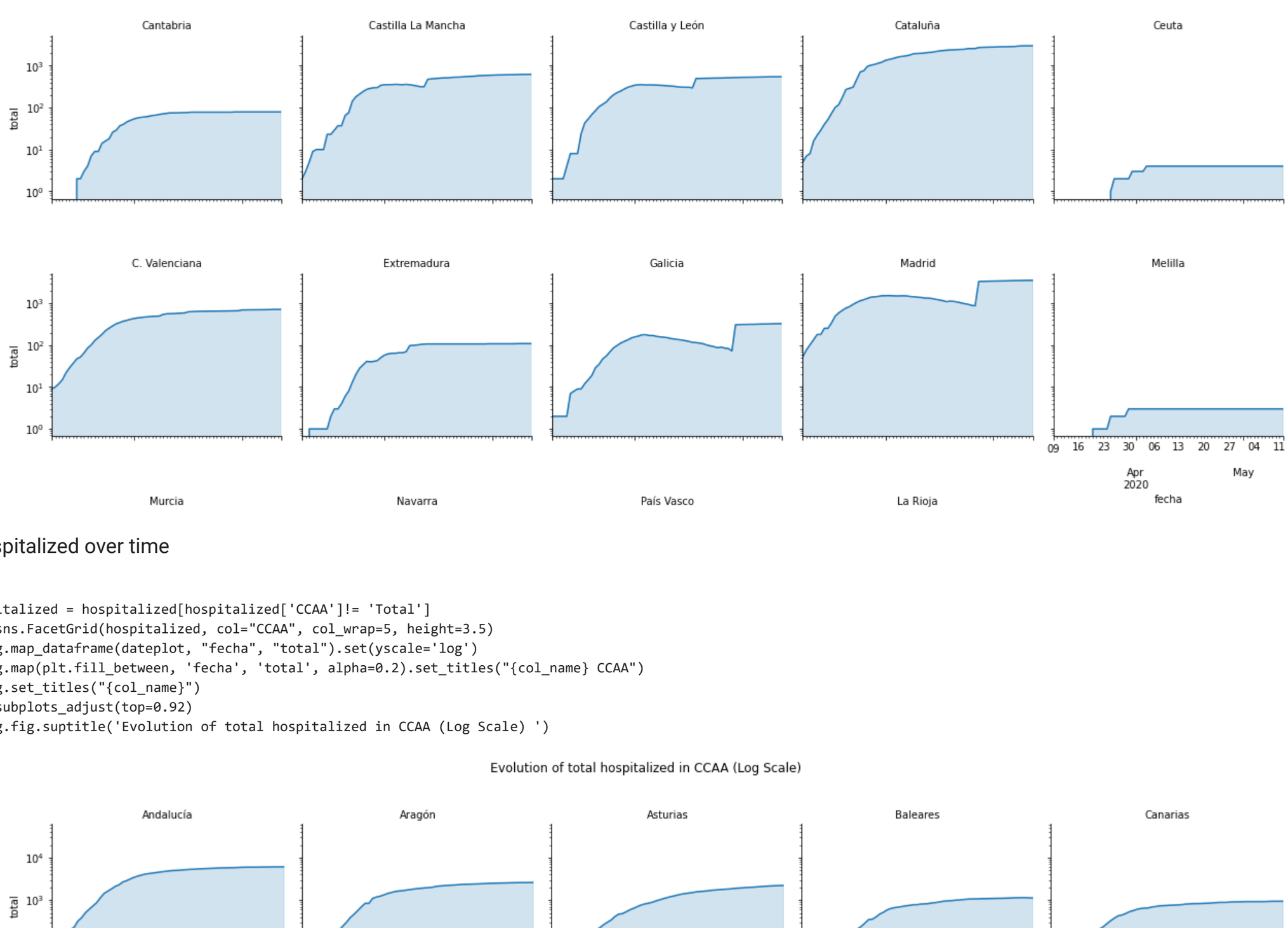


ICU over time

```
uci = uci[uci['CCAA']!= 'Total']
g = sns.FacetGrid(uci, col="CCAA", col_wrap=5, height=3.5)
g = g.map_dataframe(dateplot, "fecha", "total").set(yscale='log')
g = g.map(plt.fill_between, 'fecha', 'total', alpha=0.2).set_titles("{col_name} CCAA")
g = g.set_titles("{col_name}")
plt.subplots_adjust(top=0.92)
g = g.fig.suptitle('Evolution of total UCI patients in CCAA (log scale)')
```

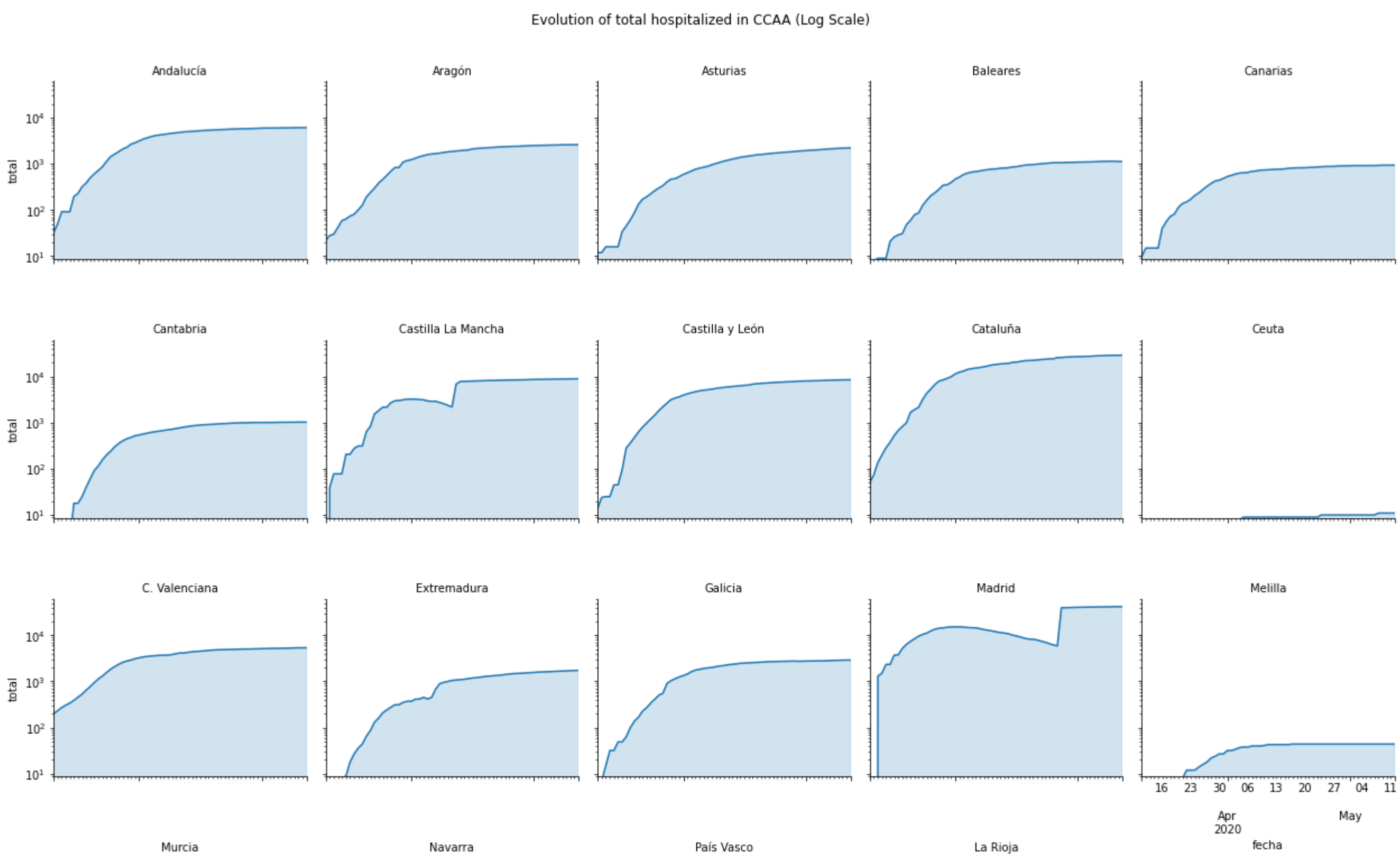
ICU over time

```
uci = uci[uci['CCAA']!= 'Total']
g = sns.FacetGrid(uci, col="CCAA", col_wrap=5, height=3.5)
g = g.map_dataframe(dateplot, "fecha", "total").set(yscale='log')
g = g.map(plt.fill_between, 'fecha', 'total', alpha=0.2).set_titles("{col_name} CCAA")
g = g.set_titles("{col_name}")
plt.subplots_adjust(top=0.92)
g = g.fig.suptitle('Evolution of total UCI patients in CCAA (log scale)')
```



Hospitalized over time

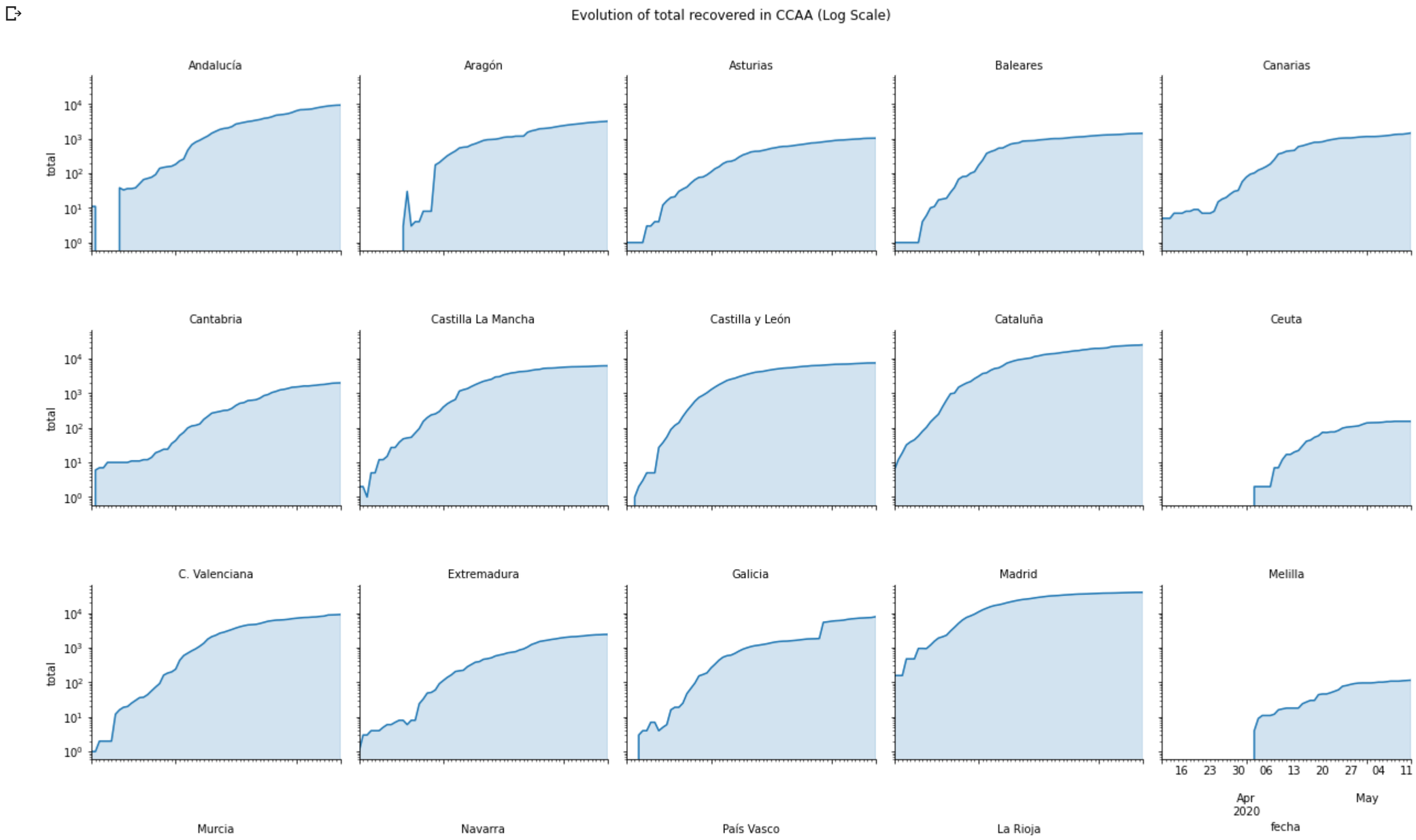
```
hospitalized = hospitalized[hospitalized['CCAA']!= 'Total']
g = sns.FacetGrid(hospitalized, col="CCAA", col_wrap=5, height=3.5)
g = g.map_dataframe(dateplot, "fecha", "total").set(yscale='log')
g = g.map(plt.fill_between, 'fecha', 'total', alpha=0.2).set_titles("{col_name} CCAA")
g = g.set_titles("{col_name}")
plt.subplots_adjust(top=0.92)
g = g.fig.suptitle('Evolution of total hospitalized in CCAA (Log Scale) ')
```



Recovered over time

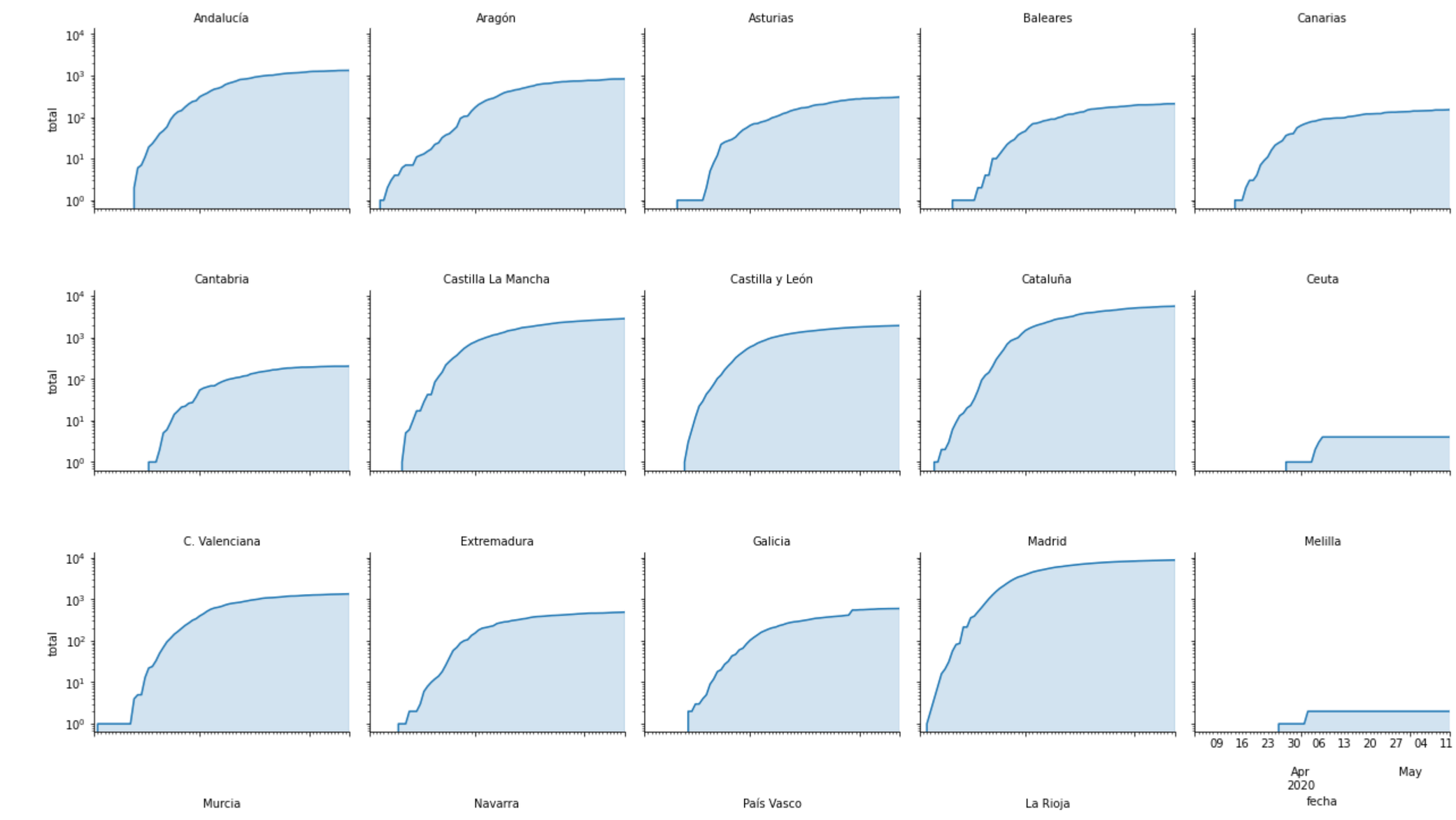
```
recovered = recovered[recovered['CCAA']!= 'Total']
g = sns.FacetGrid(recovered, col="CCAA", col_wrap=5, height=3.5)
g = g.map_dataframe(dateplot, "fecha", "total").set(yscale='log')
g = g.map(plt.fill_between, 'fecha', 'total', alpha=0.2).set_titles("{col_name} CCAA")
```

```
g = g.map(plt.fill_between, "fecha", "total", alpha=0.2).set_titles("{col_name} CCAA")
g = g.set_titles("{col_name}")
plt.subplots_adjust(top=0.92)
g = g.fig.suptitle('Evolution of total recovered in CCAA (Log Scale)')
```



Deaths over time

```
death = death[death['CCAA']!= 'Total']
g = sns.FacetGrid(death, col="CCAA", col_wrap=5, height=3.5)
g = g.map_dataframe(dateplot, "fecha", "total").set(yscale='log')
g = g.map(plt.fill_between, 'fecha', 'total', alpha=0.2).set_titles("{col_name} CCAA")
g = g.set_titles("{col_name}")
plt.subplots_adjust(top=0.92)
g = g.fig.suptitle('Evolution of total deaths in CCAA (log scale)')
```



Patient Analysis

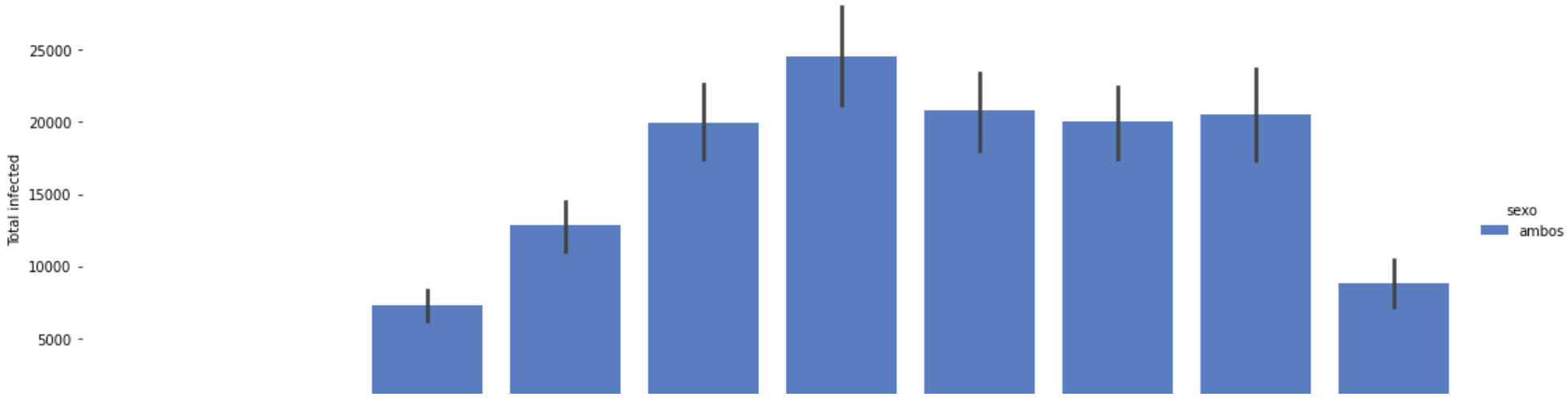
As a first aproach, I will compare the effects of COVID-19 in each age group without taking into account gender.

```
age_range= age_range[age_range['rango_edad']!='Total']
age_range= age_range[age_range['rango_edad']!='80 y +']
no_gender = age_range[age_range['sexo']=='ambos']
```

```
g = sns.catplot(x="rango_edad", y="casos_confirmados", hue="sexo", data=no_gender, kind="bar", height=5,aspect=3,palette='
g.despine(left=True)
g.set_ylabels("Total infected")
```

```
age_range= age_range[age_range['rango_edad']!='Total']
age_range= age_range[age_range['rango_edad']!='80 y +']
no_gender = age_range[age_range['sexo']=='ambos']


g = sns.catplot(x="rango_edad", y="casos_confirmados", hue="sexo", data=no_gender, kind="bar", height=5,aspect=3,palette='
g.despine(left=True)
g.set_ylabels("Total infected")
```

```
<seaborn.axisgrid.FacetGrid at 0x7fa53d6e27b8>
```




```
last = age_range[age_range.iloc[:,0]== age_range.iloc[:,0].max()]
```

In case dataframe format is wrong:

```
for i in range(last['ingresos_uci'].shape[0]):
    if last.iloc[i,5] == 'i':
        last.iloc[i,5] = 0

last['ingresos_uci']= last['ingresos_uci'].astype(int)
```

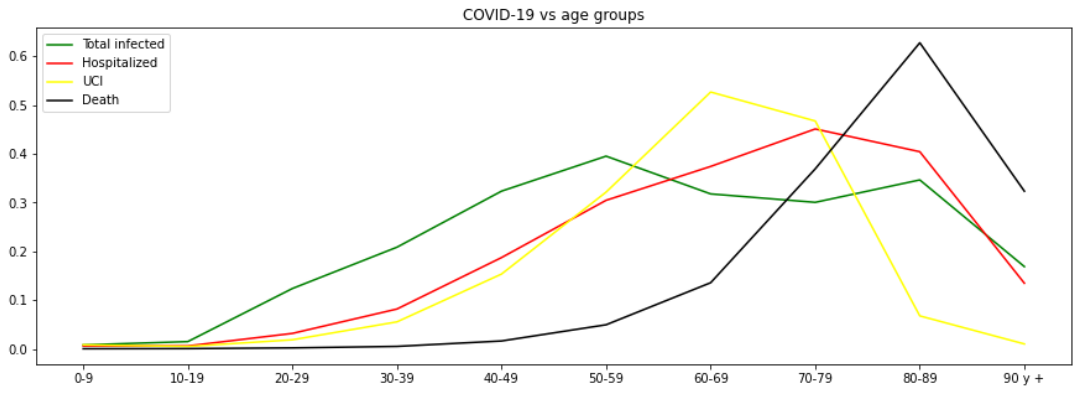
In order to compare between different categories, we should normalize the data:

```
last['casos_confirmados'] = last['casos_confirmados'] / np.linalg.norm(last['casos_confirmados'])
last['hospitalizados'] = last['hospitalizados'] / np.linalg.norm(last['hospitalizados'])
last['ingresos_uci'] = last['ingresos_uci'] / np.linalg.norm(last['ingresos_uci'])
last['fallecidos'] = last['fallecidos'] / np.linalg.norm(last['fallecidos'])


last_ambos = last[last['sexo']=='ambos']
last_gender = last[last['sexo']!='ambos']
```

COVID-19 vs age groups

```
plt.figure(figsize=(15,5))
plt.plot(last_ambos['rango_edad'], last_ambos['casos_confirmados'],color = 'green',label='Total infected')
plt.plot(last_ambos['rango_edad'], last_ambos['hospitalizados'],color = 'red',label='Hospitalized')
plt.plot( last_ambos['rango_edad'], last_ambos['ingresos_uci'],color = 'yellow',label='UCI')
plt.plot( last_ambos['rango_edad'], last_ambos['fallecidos'],color = 'black',label='Death')
plt.title('COVID-19 vs age groups')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa53efb6ef0>
```



The results are as expected. virus affects mostly older people. We can see that the death´s curve peak is around 80-89yo. The most surprising result is the "Total infected" line as the 'peak' goes from 40 to 80 years whick means the disease is present in most of the population.

COVID-19 vs Gender

```
plt.figure(figsize= (10,5))
sns.relplot(x='rango_edad',y ='casos_confirmados', hue = 'sexo',kind='line',data = last_gender,height=5,aspect=4)
plt.title('Comparison between men and women: Total infections')
```