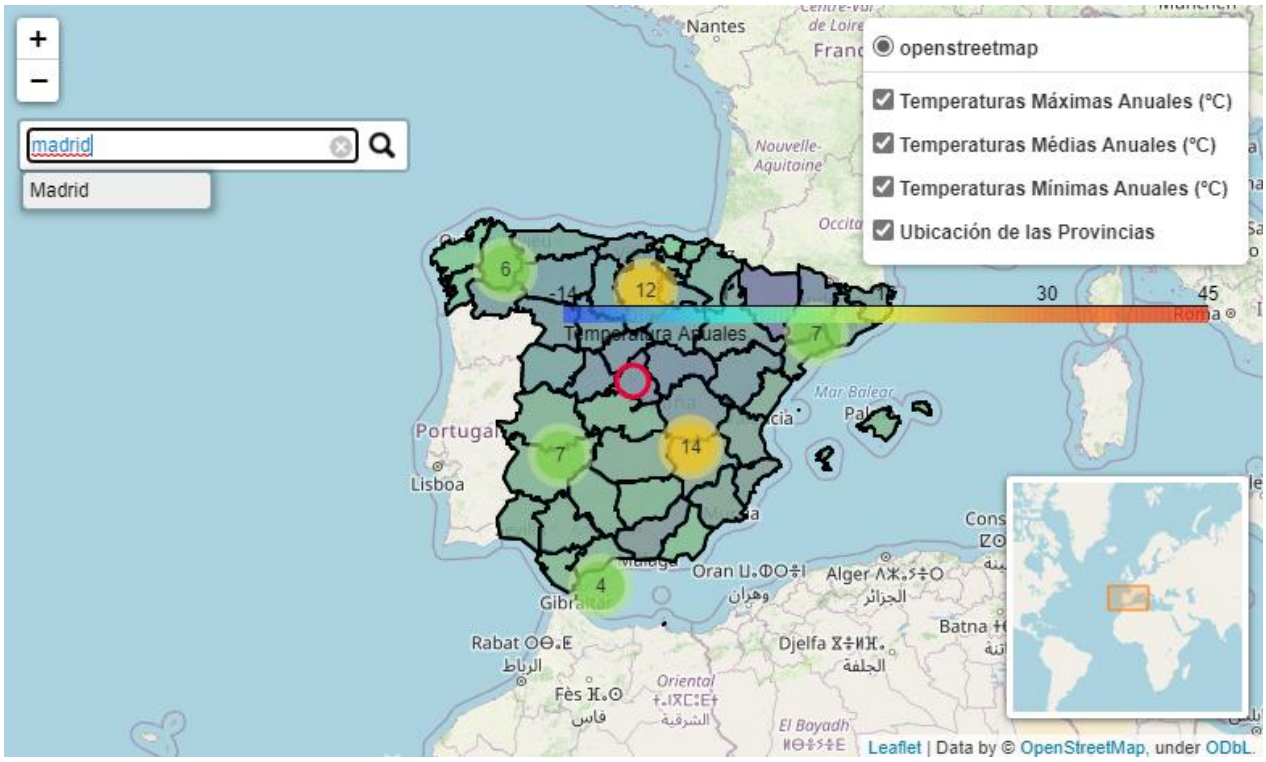
Python MIAR 2021

```
from IPython import display
from datetime import datetime
print(datetime.now().strftime("%d-%m-%Y %H:%M:%S"))
url_path="https://github.com/davidlima/01MIAR-2021/raw/main/"#data/"
path0="data/"
```

Objetivo del trabajo

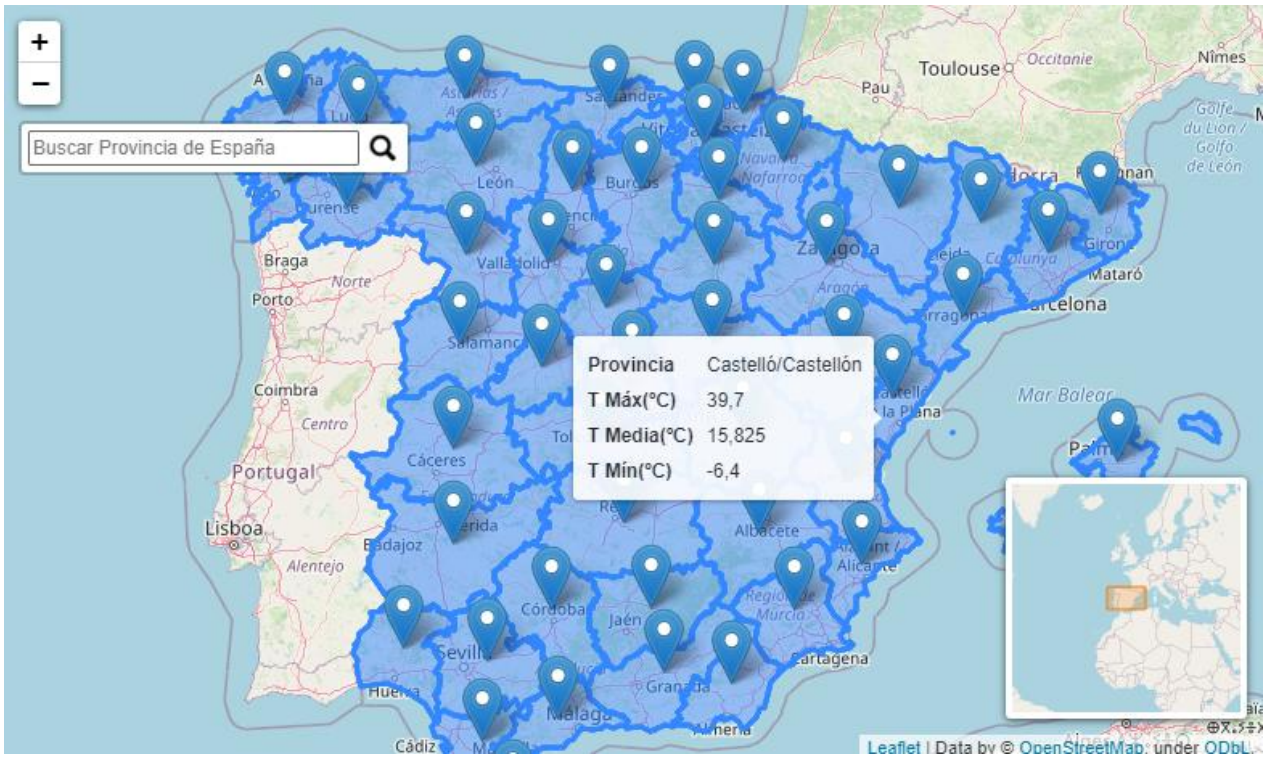
- Mediante la descarga de datos procedentes de Aemet se han obtenido los datos sobre temperaturas, velocidad del viento y precipitaciones historicas en todas las estaciones metereológicas de España que se guardan en dicha base de datos y que se han centrado en el año 2020.
- Una vez procesados, unidos y limpiados todos los datos se ha procedido a mostrarlos en una primera parte a través de un mapa interactivo como el de la foto.

```
display.Image(url_path+"img/target.png")
```



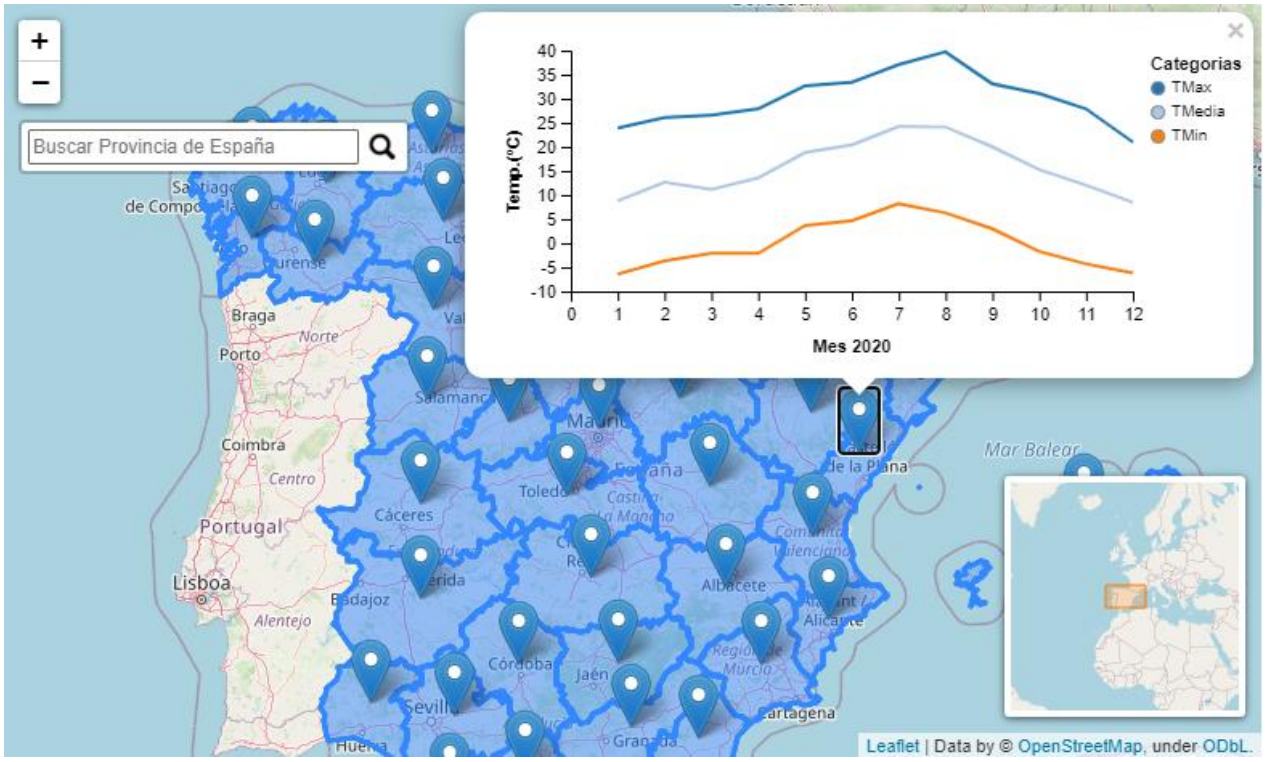
- Este mapa generado a través folium y con datos GeoJson los límites de cada provincia.
- Mediante la manipulación de los DataFrame obtenidos en Aemet y los datos de GeoJson se asocian por ubicación geográfica las medias, máximas y mínimas de temperaturas.
- Toda la información se muestra en un único plano de España donde moviendose con el ratón se pueden observar cualquiera de estos datos previamente clasificados.
- La información de cada provincia se puede acceder haciendo click, pasando el ratón por encima o buscando el nombre en un buscador insertado en el mapa.
- Tambien se ha generado 4 capas de mapas para poder separar informaciones por código de colores según temperaturas máx,min, media,...

```
1 display.Image(url_path+"img/provincias_temperatura_anuales.png")
```

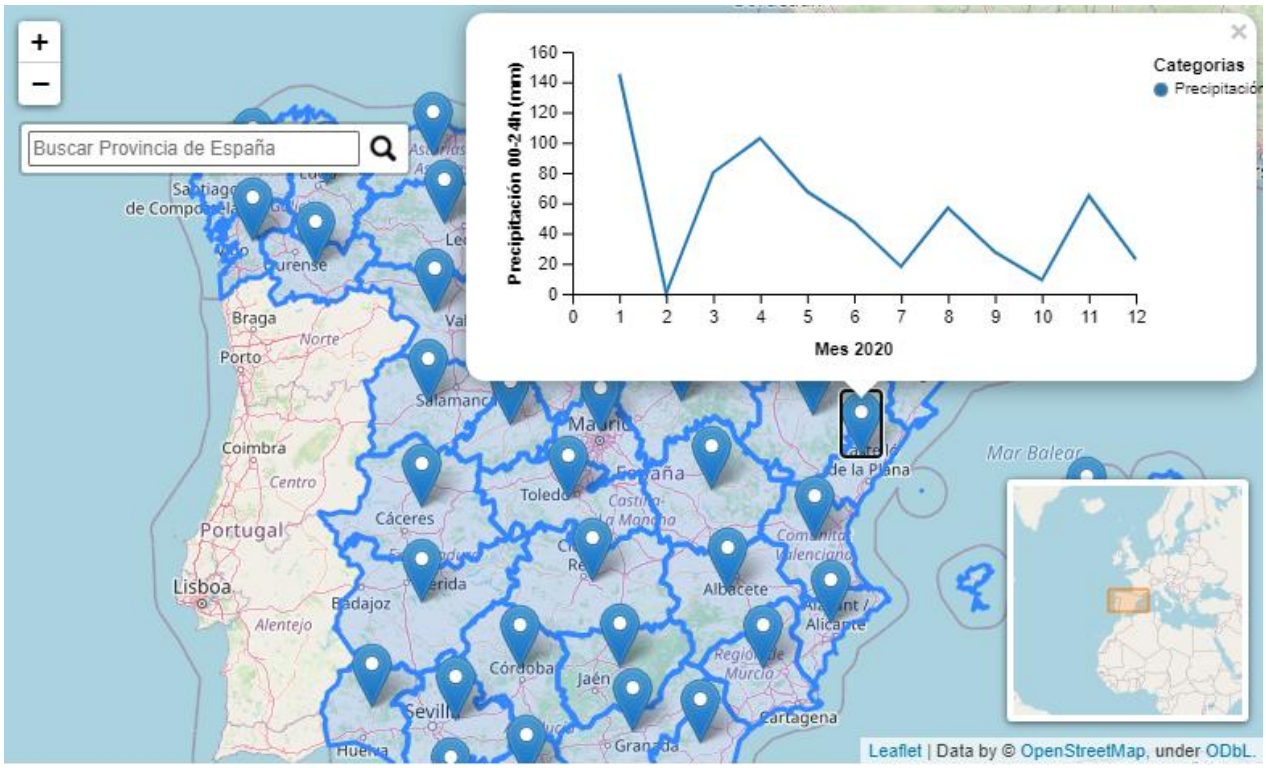


- El siguiente paso ha sido dar algo más de información visual añadiendo a cada una de las provincias, mapas independientes y tambien interactivos. De manera que haciendo click en cada una de las provincias se pueda acceder a una gráfica popup con los valores mensuales según la gráfica escogida.

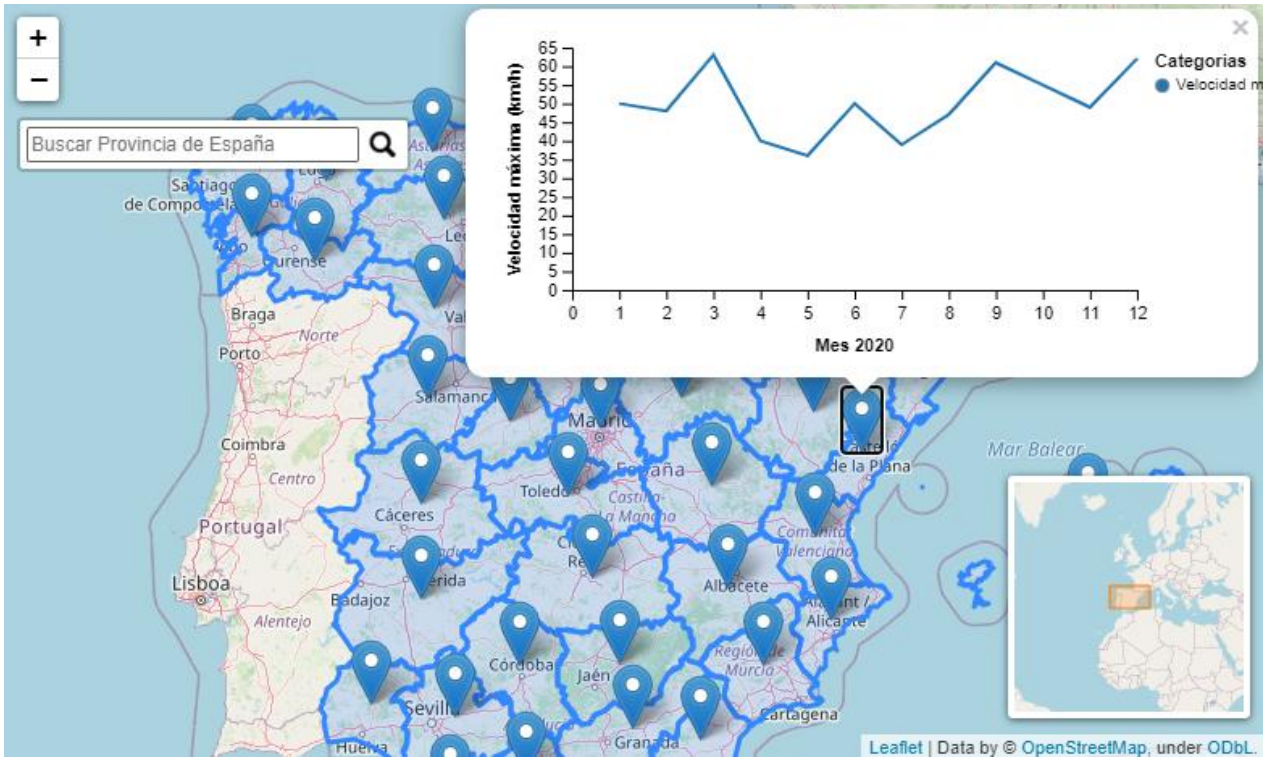
```
1 display.Image(url_path+"img/grafica_temperatura_mensual_por_provincias.png")
```




```
1 display.Image(url_path+"img/grafica_precipitacion_mensual_por_provincias.png")
```




```
1 display.Image(url_path+"img/grafica_viento_mensual_por_provincias.png")
```



- El código se ha realizado integramente en Jupyter Notebook
  - En mi caso he tenido que actualizar e instalar algunas librerias diferentes a las clásicas como:
    - !pip install pyproj --upgrade
    - !pip install geopandas --upgrade
    - !pip install vincent

Indice

- f) Mapa Popup gráfico por provincia
- g) Función Generadora gráfico Popup por provincia (genérico)
- h) Función Mapa Popup gráfico por provincia
- i) Mapa gráfico velocidad del viento por provincia.
- j) Mapa gráfico precipitaciones por provincia.

Home

0. Librerias necesarias y variables inciales

```
import geopandas
import pandas as pd
import numpy as np
import branca
import json
import folium
from folium.plugins import Search,MiniMap
from folium import plugins,features
import vincent
import random
import time
import requests
from IPython import display
from datetime import datetime
year="2020"
```

1. Obtención de datos

- Archivos descargados de enlaces de Aemet web
- Obtenidos en xls y guardados en la carpeta github.com de acceso público... Aemet2020-yyyy-mm-dd.xls
- Cada día del año tiene asociado un único archivo.xls con todas las estacioens meteoriológicas de España
- En cada uno de estos archivos se guardan datos diarios de:
  - Estación de captura.
  - Provincia.
  - Temperatura máxima.
  - Temperatura mínima.
  - Temperatura média.
  - Velocidad del viento.
  - Precipitaciones.
  - ...

Home

1.a) Lectura de un archivo de muestra

- Procedo a leer un archivo de un día en concreto y observar que datos hay en ese fichero
- data\Aemet2020-01-01.xls

```
url =url_path+"data/Aemet2020-01-01.xls"
print(url)
xl = pd.ExcelFile(url)
xl.sheet_names
df = xl.parse(xl.sheet_names[0])
df.head(5)

```

https://github.com/davidlima/01MIAR-2021/raw/main/data/Aemet2020-01-01.xls

	España	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6
0	Actualizado: jueves, 02 enero 2020 a las 18:00...	NaN	NaN	NaN	NaN	NaN	NaN
1	Fecha: miércoles, 01 enero 2020	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Home

1.b) Limpieza y Eliminación de datos incorrectos

- Observo que las 3 primeras filas NO contienen información y los nombres de las columnas estan escritos en la cuarta fila
- Renombro las columnas con los nombres de la fila nº3
- Elimino las 4 primeras filas y reinicio los indices del DataFrame

```
#  Renombro las columnas con los nombres de la fila nº3
df=df.rename(columns = df.iloc[3])
# Elimino las 4 primeras filas y reinicio los indices del DataFrame
df=df[4:].reset_index(drop = True)
df.head(6)
```

	Estación	Provincia	Temperatura máxima (ºC)	Temperatura mínima (ºC)	Temperatura media (ºC)	Racha (km/h)	Velocidad máxima (km/h)
0	Estaca de Bares	A Coruña	10.6 (00:00)	8.0 (21:00)	9.3	23 (18:40)	18 (18:40)
1	Ferrol	A Coruña	NaN	NaN	NaN	NaN	NaN

- Observo que hay valores con datos incorrectos NaN, columnas no necesarias y horas como por ejemplo (15:50) que molestan para el tratamiento de los datos
- Procedo a corregirlo

A Coruña 30

```
#Eliminamos filas con datos NaN
df.dropna(inplace=True)
#Elimino columnas no necesarias
df=df.drop(['Estación','Racha (km/h)',\
            'Precipitación 00-06h (mm)','Precipitación 06-12h (mm)',\
            'Precipitación 12-18h (mm)','Precipitación 18-24h (mm)'],axis=1)

def elimina_anomalias(texto):
    try:
        index = texto.index('(')
        texto=texto[:index]
    except:
        pass
    try:texto=float(texto)
    except:pass
    return texto
#Elimino mediante la funcion elimina_anomalias caracteres no útiles
#y luego los convierto a float para poderlos tratar más adelante
for a in df.keys():
    df[a]=df[a].apply(elimina_anomalias)
df=df.reset_index(drop=True)
df.head(6)
```

	Provincia	Temperatura máxima (ºC)	Temperatura mínima (ºC)	Temperatura media (ºC)	Velocidad máxima (km/h)	Precipitación 00-24h (mm)
0	A Coruña	10.6	8.0	9.3	18.0	0.0
1	A Coruña	11.5	6.7	9.1	13.0	0.0
2	A Coruña	10.7	2.6	6.6	13.0	0.0
3	A Coruña	10.9	7.1	9.0	20.0	0.0
4	A Coruña	12.1	-0.6	5.8	9.0	0.0

Home

1.c) Añadir datos extra

- Añado datos necesarios para el posterior procesamiento como Mes, día y año.

- Posteriormente lo tendremos que hacer con cada uno de los 365 archivos diarios.

```
# añado datos necesarios para un posterior anlisis: día, mes y año del fichero leido
year_ = [int(2020)]*len(df)
mont_ = [int(1)]*len(df)
day_ = [int(1)]*len(df)
df_temp=pd.DataFrame({'Día':day_,'Mes':mont_,'Año':year_})
df=df_temp.join(df)
df.head(8)
```

	Día	Mes	Año	Provincia	Temperatura máxima (ºC)	Temperatura mínima (ºC)	Temperatura media (ºC)	Velocidad máxima (km/h)	Pr
0	1	1	2020	A Coruña	10.6	8.0	9.3	18.0	
1	1	1	2020	A Coruña	11.5	6.7	9.1	13.0	
2	1	1	2020	A Coruña	10.7	2.6	6.6	13.0	
3	1	1	2020	A Coruña	10.9	7.1	9.0	20.0	
4	1	1	2020	A Coruña	12.1	-0.6	5.8	9.0	
5	1	1	2020	A Coruña	8.4	2.2	5.3	3.0	
6	1	1	2020	A Coruña	11.7	2.3	7.0	5.0	

Home

1.d) Generar un único DataFrame Global con todos los archivos procesados

- Despues de analizar y limpiar un único archivo,
- Procedo a generar un único DataFrame con todos los archivos de cada uno de los días del año 2020 mediante la funcion Generar_df_2020
  - Generar_df_2020 utiliza todos los procedimientos utilizados anteriormente y unidos en una única función. Repitiendo el proceso 31 veces por días máximos posibles por cada mes y a su vez por 12 meses. Accediendo con estos datos al archivo de cada día y añadiendolo al DataFrame Global

```
def Generar_df_2020():
    cont=1
    df_final = None
    print("Generando DataFrame global (df_final). Por favor espere...")
    for mes_ in range(1,13):#máx 12 meses
        mes="{:02d}".format(mes_)
        for dia_ in range(1,32):# máx 31 días
            #Cuando el día es inferior a 2 dígitos le añado un 0 para que coincida con
            dia="{:02d}".format(dia_)
            #La fecha del archivo se tiene que abrir con nombre AemetYYYY-MM-DD
            url=url_path+"data/Aemet{}-{}-{}.xls".format(year,mes,dia)
```

```
            # el try está para evitar problemas cuando se intente buscar
            # un archivo con una fecha no válida por ejempo el 30 de febrero
            try:
                xl = pd.ExcelFile(url)
                xl.sheet_names
                #se carga el archivo inicial en el dataframe df
                df = xl.parse(xl.sheet_names[0])
                #  Renombro las columnas con los nombres de la fila nº3
                df=df.rename(columns = df.iloc[3])
                # Elimino las 4 primeras filas y reinicio los indices del DataFrame
                df=df[4:].reset_index(drop = True)
                #Eliminamos filas con datos NaN
                df.dropna(inplace=True)
                #Elimino columnas no necesarias
                df=df.drop(['Estación','Racha (km/h)',\
                            'Precipitación 00-06h (mm)','Precipitación 06-12h (mm)',\
                            'Precipitación 12-18h (mm)','Precipitación 18-24h (mm)'],ax
                #Elimino mediante la funcion elimina_anomalias caracteres no utiles
                #y luego los convierto a float para poderlos tratar más adelante
                for a in df.keys():
                    df[a]=df[a].apply(elimina_anomalias)
                df=df.reset_index(drop=True)
                # añado datos necesarios para un posterior anlisis: día, mes y año del
                # añado tantos años, mese y dias como líneas tiene el DataFrame descarg
                year_ = [int(2020)]*len(df)
                month_ = [int(mes_)]*len(df)
                day_ = [int(dia_)]*len(df)
                df_temp=pd.DataFrame({'Día':day_,'Mes':month_,'Año':year_})
                # uno el dataframe descargado con las columnas nuevas añadidas
                df=df_temp.join(df)

                # en cada iteración voy uniendo el nuevo archivo con los anteriores
                try:
                    df_final=pd.concat([df_final,df])

                except:
                    df_final=df.copy()
            except:
                pass
        print("({}/12) {}".format(cont,url),end="\r")
        cont+=1
    df_final.sample(5)
    print("\nDataFrame df_final finalizado!      \t\t\t        \t\t  \t\t\t     \t\t\t
    #procedo a resetear el indice ya que cada DataFrame concatenado lleva sus propios í
    df_final=df_final.reset_index(drop=True).sort_values(by=['Provincia'])
    df_final.sample(5)
    return df_final
```

- Descargar los 365 archivos de Github es más lento que hacerlo desde el propio pc...
- Si tarda mucho se puede parar u omitir la siguiente ejecución y pasar a la siguiente celda donde descargará el archivo ya procesado de GitGub!!

```
df_final=Generar_df_2020()
df_final=df_final.sort_values(by=['Provincia']).reset_index(drop=True)

# Guarda  copia de seguridad en pc
df_final.to_csv("Aemet_Total{}.csv".format(year))#Guardar DataFrame completo limpio
df_final.tail(5)
```

```
Generando DataFrame global (df_final). Por favor espere...
(12/12) https://github.com/davidlima/01MIAR-2021/raw/main/data/Aemet2020-12-31.xls
DataFrame df_final finalizado!
```

	Día	Mes	Año	Provincia	Temperatura máxima (ºC)	Temperatura mínima (ºC)	Temperatura media (ºC)	Velocida máxi (km/h
225614	27	4	2020	Ávila	14.2	6.5	10.3	32
225615	1	6	2020	Ávila	27.3	13.0	20.1	26
225616	1	6	2020	Ávila	26.3	10.6	18.4	17
225617	18	1	2020	Ávila	11.6	3.0	7.3	24

- Recuperar archivo procesado desde copia de seguridad de GitHub

```
# Recupera copia de seguridad del archivo generado por Generar_df_2020
df_final=pd.read_csv(url_path+"Aemet_Total{}.csv".format(year),index_col=0)#Leer DataFr
```

Home

2. Mostrar información

- Muestro una pequeño resumen estadístico del 2020

```
print("Número de registros generados en fichero de 2020 = {:0,.0f}".format(float(len(df
print("\nNúmero de provincias = {}".format(len(df_final['Provincia'].unique())))
temp=df_final['Temperatura máxima (ºC)'].max()
p_temp=df_final['Temperatura máxima (ºC)'].idxmax()
print("\nTemperatura máxima registrada en 2020 = {}ºC registrado en {} ".\
      format(temp,df_final.iloc[p_temp]['Provincia']))
temp=df_final['Temperatura mínima (ºC)'].min()
p_temp=df_final['Temperatura mínima (ºC)'].idxmin()
print("\nTemperatura mínima registrada en 2020 = {}ºC registrado en {} ".\
      format(temp,df_final.iloc[p_temp]['Provincia']))
temp=df_final['Temperatura mínima (ºC)'].max()
p_temp=df_final['Velocidad máxima (km/h)'].idxmax()
print("\nVelocidad máxima registrada en 2020 = {}km/h registrado en {} ".\
      format(temp,df_final.iloc[p_temp]['Provincia']))
temp=df_final['Precipitación 00-24h (mm)'].max()
p_temp=df_final['Precipitación 00-24h (mm)'].idxmax()
print("\nPrecipitación 00-24h máxima registrada en 2020 = {}mm registrado en {} ".\
      format(temp,df_final.iloc[p_temp]['Provincia']))
```

```
Número de registros generados en fichero de 2020 = 225,619
```

```
Número de provincias = 52

Temperatura máxima registrada en 2020 = 44.8ºC registrado en Málaga

Temperatura mínima registrada en 2020 = -13.7ºC registrado en Huesca

Velocidad máxima registrada en 2020 = 32.9km/h registrado en Bizkaia

Precipitación 00-24h máxima registrada en 2020 = 301.0mm registrado en Ávila
```

Home

2.a) Preparación del mapa

- Genero colormap para poder mostrar posteriormente información visual según sus valores y colores

```
colormap = branca.colormap.LinearColormap(
    colors=["#3151EE", "#31EEE5", "#E2EE31", "#F26B13", "#F23513"],
    vmin=df_final['Temperatura mínima (ºC)'].min(),
    vmax=df_final['Temperatura máxima (ºC)'].max(),)
colormap.caption = "Temperatura Anuales"
print("Variación de colores según la temperatura:")
colormap
```

```
Variación de colores según la temperatura:
```

- Genero 3 estilos de funciones de colormap para relaciónar posteriormente las capas de planos a colores relativos

```
def style_function_TMax(x):
    return {
        "fillColor": colormap(x["properties"]["Temperatura máxima (ºC)"]),
        "color": "black",
        "weight": 2,
        "fillOpacity": 0.5,
    }
def style_function_TMin(x):
    return {
        "fillColor": colormap(x["properties"]["Temperatura mínima (ºC)"]),
        "color": "black",
        "weight": 2,
        "fillOpacity": 0.5,
    }
def style_function_TMedia(x):
    return {
        "fillColor": colormap(x["properties"]["Temperatura media (ºC)"]),
        "color": "black",
```

```
        "weight": 2,
        "fillOpacity": 0.5,
    }
```

Home

2.b) Generar estadísticas de (Aemet) dentro del plano (GeoJSON)

- Cargo archivo provincias con las coordenadas de todas las provincias en formato GeoJSON

```
json_url = requests.get(url_path+'provincias-espanolas.geojson')
datos = json_url.json()


file_=r'provincias-espanolas.geojson'
provincias = geopandas.read_file(url_path+file_,driver="GeoJSON",)
```

```
#realizo copia del GeoJSON en DataFrame a procesar df_Provincia
df_Provincia=provincias.copy().sort_values(by=['provincia'])

#UNIFICA LOS NOMBRES DE LAS 2 TABLAS (Aemet y GeoJSON)
#PARA QUE EL NOMBRE DE LAS PROVINCIAS SEAN IGUALES
tmp=df_final['Provincia'].unique()
tmp=np.sort(tmp)
df_Provincia[['provincia']]=tmp

#CALCULA MAX, MIN, MEDIAS Y LAS AÑADE AL DATAFRAME df_Provincia (GeoJSON)
nombre_de_provincias=df_final['Provincia'].unique()
nombre_de_provincias=np.sort(tmp)
max_=[]
min_=[]
mean_=[]
veloc_=[]
precip_=[]
#Recorro todas las provincias filtrando el DataFrame de Aemet (df_final)
#por provincias y los valores de los que quiero buscar sus máximos, mínimos o medios
#añadiendo cada valo de cada provincia a una lista
for _ in nombre_de_provincias:
    max_.append(df_final[df_final['Provincia']==_]['Temperatura máxima (ºC)'].max())
    min_.append(df_final[df_final['Provincia']==_]['Temperatura mínima (ºC)'].min())
    mean_.append(df_final[df_final['Provincia']==_]['Temperatura media (ºC)'].mean())
    veloc_.append(df_final[df_final['Provincia']==_]['Velocidad máxima (km/h)'].max())
    precip_.append(df_final[df_final['Provincia']==_]['Precipitación 00-24h (mm)'].max(

# inserto en cada provincia (GeoJSON) sus valores encontrados en (Aemet) df_final
df_Provincia.insert(2,'Precipitación 00-24h (mm)',precip_)
df_Provincia.insert(2,'Velocidad máxima (km/h)',veloc_)
df_Provincia.insert(2,'Temperatura media (ºC)',mean_)
df_Provincia.insert(2,'Temperatura mínima (ºC)',min_)
df_Provincia.insert(2,'Temperatura máxima (ºC)',max_)
df_Provincia.head(2)
```

	provincia	texto	Temperatura máxima (ºC)	Temperatura mínima (ºC)	Temperatura media (ºC)	Velocidad máxima (km/h)	Prec 00
15	A Coruña	La Coruña	36.5	-2.4	15.048957	109.0	

Home

2.c) Generar Tooltip (iconos)

- Con las coordenadas de todas las ciudades de provincias a partir de las coordenadas obtenidas del archivo provincias-espanolas.json diferente al (GeoJSON) previo. En donde se ha podido obtener las coordenadas de las 52 provincias y añadirlas a las base de datos.

```
#leer datos json con coordenadas de todas las ciudades de provincia
json_url = requests.get(url_path+'provincias-espanolas.json')
datos = json_url.json()

#convierte el json en dos listas con nombres y coordenadas de provincias
Pos_Provincias_2d=[]
pos_Provincia_name=[]
df=pd.DataFrame(columns=['geo_point_2d'])

for datos_ in datos:
    Pos_Provincias_2d.append(datos_['fields']['geo_point_2d'])
    pos_Provincia_name.append(datos_['fields']['provincia'])
#Genero en una lista los datos de provincias y coordenadas de dichas provincias
lista_Pos_Provincias_2d=list(dict(sorted(zip(pos_Provincia_name,Pos_Provincias_2d))).va
```

Home

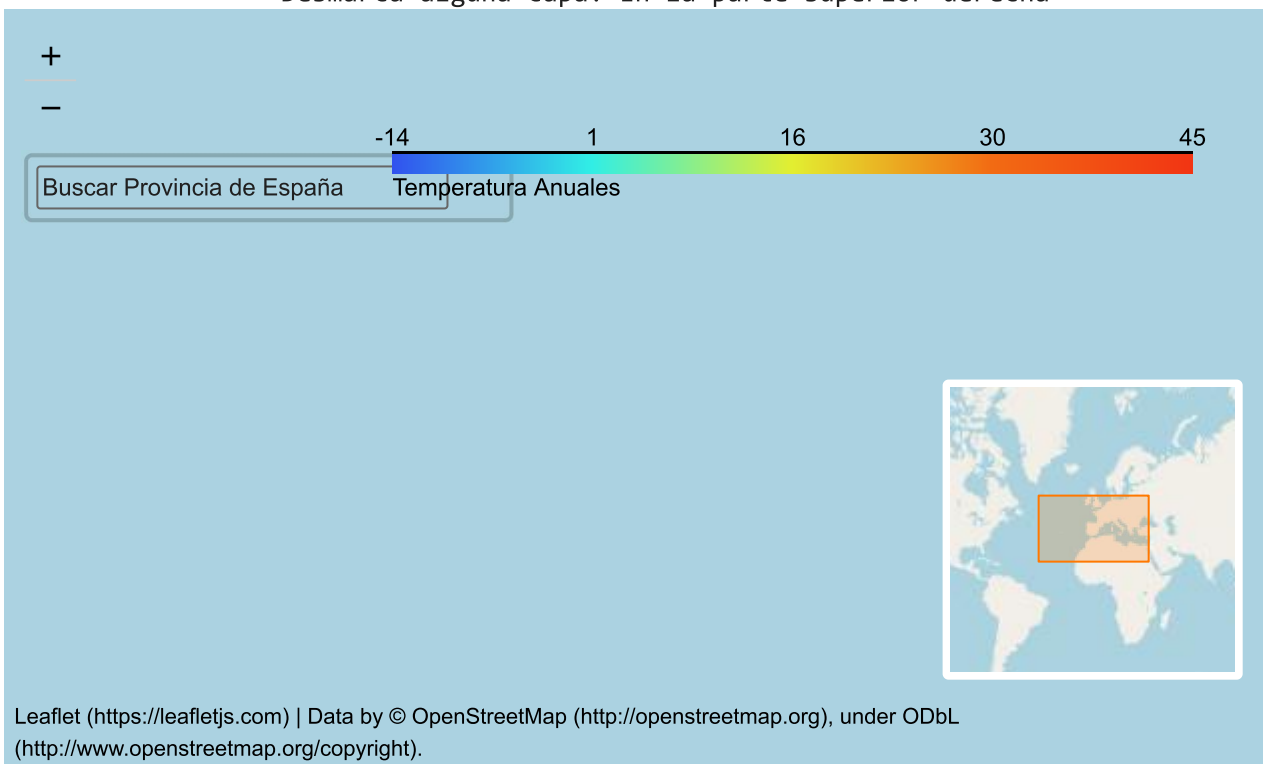
2.d) Muestra Mapa por capas.

- Capa 1. Temperaturas Máximas anuales.
- Capa 2. Temperaturas Medias anuales.
- Capa 3. Temperaturas Mínimas anuales.
- Capa 4. Iconso de provincias.

```
print("Procesando! Por favor espere...",end="\r")
m = folium.Map(location=[40.416775, -3.703790],  zoom_start=5)
fields_=["provincia", "Temperatura máxima (ºC)","Temperatura media (ºC)","Temperatura m
aliases_=["Provincia", "T Máx(ºC)","T Media(ºC)","T Mín(ºC)"]
#Añade capa temperatura máxima por provincia/anual
t_max_ = folium.GeoJson(
    df_Provincia,
```

```
    name="Temperaturas Máximas Anuales (ºC)",
    style_function=style_function_TMax,
    tooltip=folium.GeoJsonTooltip(
        fields=fields_, aliases=aliases_, localize=True
    ),
).add_to(m)
#Añade capa temperatura media por provincia/anual
folium.GeoJson(
    df_Provincia,
    name="Temperaturas Médias Anuales (ºC)",
    style_function=style_function_TMedia,
    tooltip=folium.GeoJsonTooltip(
        fields=fields_, aliases=aliases_, localize=True
    ),
).add_to(m)
#Añade capa temperatura mínima por provincia/anual
folium.GeoJson(
    df_Provincia,
    name="Temperaturas Mínimas Anuales (ºC)",
    style_function=style_function_TMin,
    tooltip=folium.GeoJsonTooltip(
        fields=fields_, aliases=aliases_, localize=True
    ),
).add_to(m)

#Añade capa de los Tooltips de todas las provincias
pos_Provincia_name.sort()
plugins.MarkerCluster(lista_Pos_Provincias_2d, popups=pos_Provincia_name,
                      name="Ubicación de las Provincias").add_to(m)
folium.LayerControl().add_to(m)
colormap.add_to(m)

#inserta barra de busqueda de provincia
Search(
    layer=t_max_,
    geom_type="Point",
    placeholder="Buscar Provincia de España",
    collapsed=False,
    search_label="provincia",
).add_to(m)

#Generar un mapa en miniatura en la parte inferior derecha
minimap = MiniMap()
m.add_child(minimap)

print("                 Temperatura Máximas/Mínimas/Medias (anuales 2020) por Provinci
print("                 Desmarca alguna capa! En la parte superior derecha         ")
m #muestra mapa
```

```
Temperatura Máximas/Mínimas/Medias (anuales 2020) por Provincias
Desmarca alguna capa! En la parte superior derecha
```



Home

2.e) Genera gráfico Popup por provincia.

- Genera gráfico independiente de temperaturas para cada una de las ciudades de los meses de 2020.

```
tmp=df_final['Provincia'].unique()#tmp= nº total de provincias (52)
tmp=np.sort(tmp)
Y= ['TMín (ºC)', 'TMedia (ºC)', 'TMáx (ºC)']
X = range(1, 13, 1)
grafica0={}
for provincia_ in tmp:
    #grafica comienza añadiendo un dict con el eje X que será igual al nº de meses (12)
    grafica = {'index': X}
    #añade al diccionario una lista TMin que será igual al filtrado de cada una de las
    #(provincia_) y cada uno de los meses (mes_)
    grafica['TMin']=[df_final[(df_final['Provincia']==provincia_)&(df_final['Mes']==mes
        ['Temperatura mínima (ºC)'].min() for mes_ in X]
    grafica['TMedia']=[df_final[(df_final['Provincia']==provincia_)&(df_final['Mes']==m
        ['Temperatura media (ºC)'].mean() for mes_ in X]
    grafica['TMax']=[df_final[(df_final['Provincia']==provincia_)&(df_final['Mes']==mes
        ['Temperatura máxima (ºC)'].max() for mes_ in X]
    time.sleep(0.01) #Para evitar problemas de visualización genero un retardo de 10ms
    print( "Procesando gráfico para (",provincia_,")                  ",end="\r")
    #realiza una conversión mediante vincent para poder lanzar posteriormente el Popup
    #con folium
    line = vincent.Line(grafica, iter_idx='index',width=350,height=150)
    line.axis_titles(x='Mes 2020', y='Temp.(ºC)')
    line.legend(title='Categorias')
    data = json.loads(line.to_json())
    #añade los datos convertidos a grafica0 para añadirlo en folium cuando se genere
    #el mapa con los tooltips y este gráfico popup
    grafica0[provincia_]=data
print( "Procesando gráfico finalizado!                 ",end="\r")
```

Home

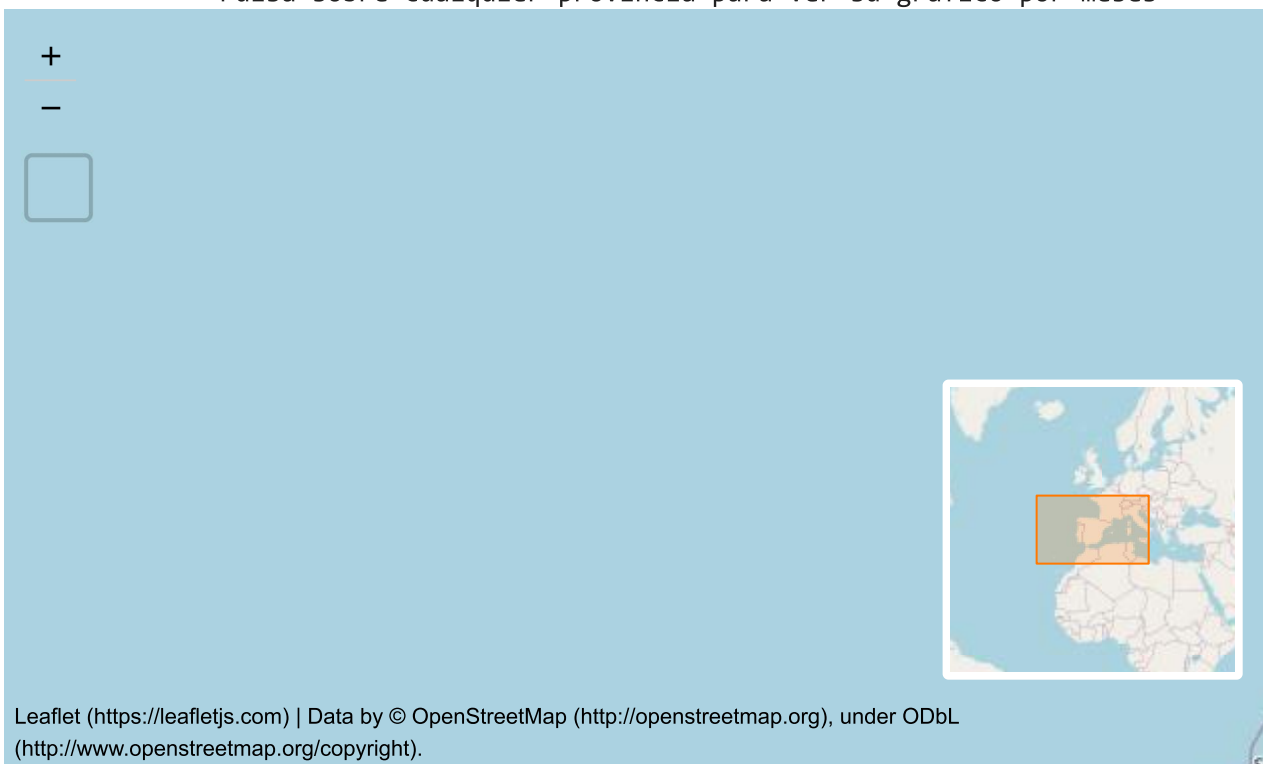
2.f) Mapa Popup gráfico por provincia.

- Muestra cada uno de los graficos generados anteriormente cuando se hace click sobre cualquiera de los iconos de las provincias
    - Se abre un Popup con una triple gráfica de los 12 meses y sus temperaturas máx,med y mím

```
print("Procesando! Por favor espere...",end="\r")
m2 = folium.Map(location=[40.41, -3.70],  zoom_start=6)
fields_=["provincia", "Temperatura máxima (ºC)","Temperatura media (ºC)","Temperatura m
aliases_=["Provincia", "T Máx(ºC)","T Media(ºC)","T Mín(ºC)"]
# #Añade capa temperatura máxima por provincia/anual
```

```
t_max_ = folium.GeoJson(
    df_Provincia,
    #style_function=style_function_TMax,
    tooltip=folium.GeoJsonTooltip(
        fields=fields_, aliases=aliases_, localize=True
    ),
).add_to(m2)
#Añade capa temperatura mínima por provincia/anual
folium.GeoJson(
    df_Provincia,
    #style_function=style_function_TMedia,
    tooltip=folium.GeoJsonTooltip(
        fields=fields_, aliases=aliases_, localize=True
    ),
).add_to(m2)
#Añade capa temperatura mínima por provincia/anual
folium.GeoJson(
    df_Provincia,
    #style_function=style_function_TMin,
    tooltip=folium.GeoJsonTooltip(
        fields=fields_, aliases=aliases_, localize=True
    ),
).add_to(m2)
#Añade gráficos Popup las provincias
pos_Provincia_name.sort()
for num,provincia_ in enumerate(tmp):
    folium.Marker(
        location=lista_Pos_Provincias_2d[num],
        popup=folium.Popup(max_width=500).add_child(
            folium.Vega(grafica0[provincia_], width=450, height=200)
        ),
    ).add_to(m2)

#inserta barra de busqueda de provincia
Search(
    layer=t_max_,
    geom_type="Point",
    placeholder="Buscar Provincia de España",
    collapsed=True,
    search_label="provincia",
).add_to(m2)

#Para generar un mapa en miniatura en la parte inferior derecha
minimap = MiniMap()
m2.add_child(minimap)


print("              Gráfica de Temperatura por Provincias 2020
print("              Pulsa sobre cualquier provincia para ver su gráfico por meses
m2
```

Gráfica de Temperatura por Provincias 2020
Pulsa sobre cualquier provincia para ver su gráfico por meses



Home

2.g) Función genera gráfico Popup por provincia.

- Genera gráfico independiente generico para usarlo posteriormente para mostrar velocidad del viento y precipitaciones para cada una de las ciudades de los meses de 2020.
    - Funciomiento parecido al 2.e)

```
def genera_grafico_por_provincias(columna):
    tmp=df_final['Provincia'].unique()
    tmp=np.sort(tmp)

    Y= ['columna']
    X = range(1, 13, 1)
    grafica0={}
    for provincia_ in tmp:
        grafica = {'index': X}
        grafica[columna]=[df_final[(df_final['Provincia']==provincia_)&(df_final['Mes']
            [columna].max() for mes_ in X]
        time.sleep(0.01) #Para evitar problemas de visualización genero un retardo de 1
        print( "Procesando gráfico",columna," para (",provincia_,")                  ",
        line = vincent.Line(grafica, iter_idx='index',width=300,height=150)
        line.axis_titles(x='Mes 2020', y=columna)
        line.legend(title='Categorias')
        data = json.loads(line.to_json())
        grafica0[provincia_]=data
        #display(grafica)
    print( "Generador gráfico finalizado!                                  ",end="\r")
    return grafica0
```

Home

2.h) Función mapa Popup gráfico por provincia.

- Muestra cada uno de los graficos generados anteriormente cuando se hace click sobre cualquiera de los iconos de las provincias
- Se abre un Popup con una triple gráfica de los 12 meses y sus temperaturas máx,med y mím.
    - Funcionamiento igual al 2.f)

```
def muestra_mapa_pantalla(columna):
    print("Procesando! Por favor espere...",end="\r")
    m3 = folium.Map(location=[40.41, -3.70],  zoom_start=6)
    fields_=["provincia", columna]
    aliases_=["Provincia", columna]
    # #Añade capa temperatura máxima por provincia/anual
    t_max_ = folium.GeoJson(
        df_Provincia,
```

```
        #style_function=style_function_TMax,
        tooltip=folium.GeoJsonTooltip(
            fields=fields_, aliases=aliases_, localize=True
        ),
    ).add_to(m3)


    pos_Provincia_name.sort()
    for num,provincia_ in enumerate(tmp):
        folium.Marker(
            location=lista_Pos_Provincias_2d[num],
            popup=folium.Popup(max_width=500).add_child(
                folium.Vega(grafica0[provincia_], width=550, height=200)
            ),
        ).add_to(m3)

    Search(
        layer=t_max_,
        geom_type="Point",
        placeholder="Buscar Provincia de España",
        collapsed=False,
        search_label="provincia",
    ).add_to(m3)

    print("              Gráfica de {}  2020                                        ".for
    print("              Pulsa sobre cualquier provincia para ver su gráfico por meses
    return m3
```

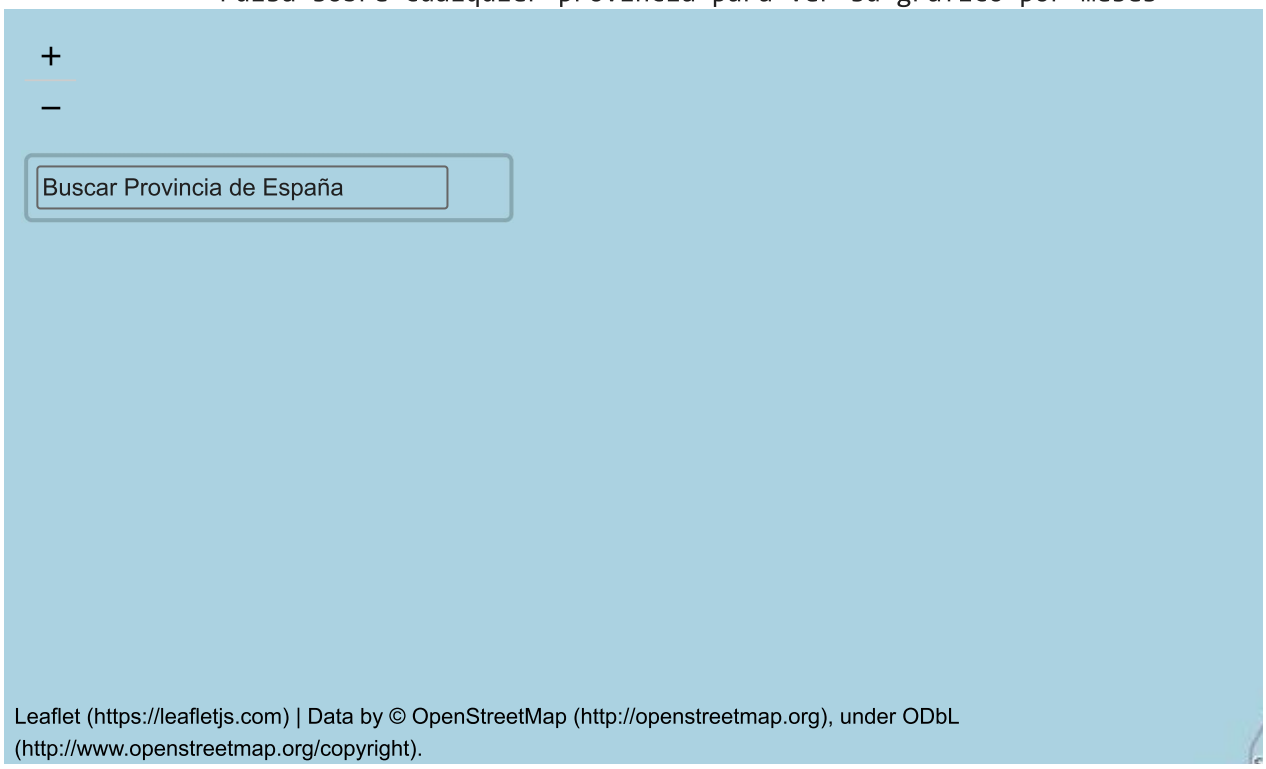
Home

2.i) Mapa gráfico velocidad del viento por provincia.

```
grafica0=genera_grafico_por_provincias('Velocidad máxima (km/h)')
```

```
m3=muestra_mapa_pantalla('Velocidad máxima (km/h)')
m3
```

```
Gráfica de Velocidad máxima (km/h)  2020
Pulsa sobre cualquier provincia para ver su gráfico por meses
```



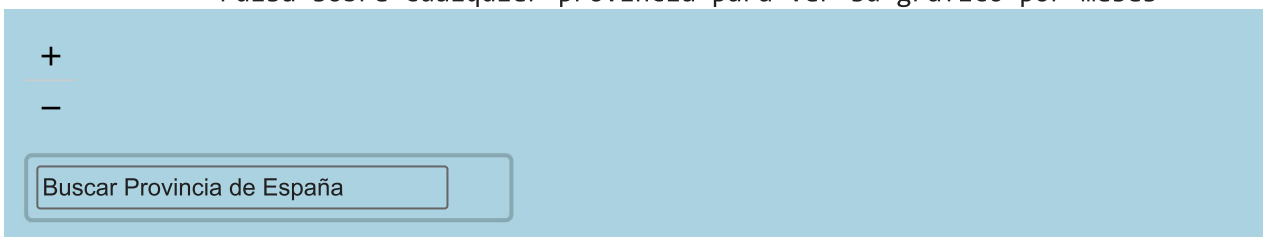
[Home](#)

2.j) Mapa gráfico precipitaciones por provincia.

```
1 grafica0=genera_grafico_por_provincias('Precipitación 00-24h (mm)')
```

```
m3=muestra_mapa_pantalla('Precipitación 00-24h (mm)')
m3
```

```
Gráfica de Precipitación 00-24h (mm)  2020
Pulsa sobre cualquier provincia para ver su gráfico por meses
```



Conclusión

- Aunque se pueden seguir generando más gráficos y estadisticas en 3D, por estaciones, etc ... creo que quedan plasmados en este notebook las peticiones iniciales del trabajo.
- Me ha generado muchísimo más tiempo el buscar y aprender como utilizar los recursos utilizados que la propia realización del trabajo solicitado.
- Aunque he de reconocer que es la mejor forma de aprender. Utilizar los recursos aprendidos, indagar, probar y errar. Para poder crecer.

```
1
```