10-07-22

Trabajo Final de Máster

MÁSTER DE PROGRAMACIÓN AVANZADA EN PYTHON PARA BIG DATA, HACKING Y MACHINE LEARNING

JAVIER BERMEJO

1 Contenido

2 Introducción

2.1 Contextualización

Este trabajo de fin de master lo realizo con la finalidad de crear un modelo predictivo utilizando el lenguaje de programación Phyton. El modelo debe permitir la estimación de las temperaturas máximas futuras en la ciudad de Getafe en la provincia de Madrid en los próximos meses utilizando técnicas estadísticas. He elegido esta población basándome en la proximidad con respecto a mi ubicación de una de las estaciones meteorológica de las que dispone la AEMET.

Actualmente, las técnicas de aprendizaje automático se utilizan ampliamente en muchos campos. El uso de técnicas de aprendizaje automático para la predicción del clima es una de esas aplicaciones. Estas técnicas han tenido éxito en la predicción de las condiciones climáticas futuras de una región.

En general, los algoritmos de aprendizaje automático se utilizan para predecir las condiciones climáticas futuras de una región analizando datos pasados y haciendo predicciones sobre la base de las relaciones que existen entre varias variables, como son: la temperatura, la humedad, la velocidad del viento, etc. La precisión de estas predicciones depende del número de observaciones que están disponibles para el análisis, con un mayor número de observaciones se produce un aumento de la precisión.

2.2 Motivación, objetivos, herramientas

Este TFM tiene como objetivo entender varios modelos de ML y aplicarlos a la predicción de la temperatura máxima de las localidades.

Se utilizarán diversas herramientas y librerías. Para el análisis inicial de los modelos se usará el entorno de trabajo de Jupyter notebook ejecutándose sobre un contenedor docker en lugar de hacerlo en el equipo local.

Las librerías principales que se usarán son:

- Pandas
- Numpy
- Matplotlib
- Folium
- Sklearn
- Pyaemet

La Agencia Estatal de Meteorología (AEMET) permite el libre acceso a sus datos climatológicos. Según la página web, el portal de datos abiertos de AEMET es un "Sistema de difusión y reutilización de la información de AEMET". La pantalla principal del portal contiene cuatro secciones:

- Una con información general sobre cómo acceder a los datos y qué datos están disponibles
- Otra sección en la que los usuarios pueden solicitar una clave API
- Dos secciones que muestran los datos propiamente dichos

Todos los datos climáticos, así como las estaciones necesarias para la realización del trabajo, los obtendré de fuentes abiertas facilitadas por AEMET.

2.3 Estructura del trabajo

El enfoque que se ha dado al trabajo es el de tratarlo como un problema de ML directamente, siguiendo con un análisis de los resultados obtenidos y de la eficiencia de cada uno de los modelos probados, estas pruebas se realizarán utilizando los estándares de análisis de errores utilizados en la estadística.

3 Desarrollo

3.1 Descripción

Este TFM es la creación y prueba de un modelo de predicción de las temperaturas máximas en la estación más cercana a mi lugar de residencia, para ello se usará un conjunto de datos seleccionados del portal AEMET OPENDATA.

He utilizado los valores de las distancias de las distintas estaciones de medición existentes cercanas a mi ubicación. De este modo, he seleccionado la estación situada en la población de Getafe ya que es la más próxima.

Los rangos de datos que se tomarán son desde el año 2015 hasta la fecha actual del TFM.

Dentro del amplio campo del ML, para este tipo de problemas se utiliza el enfoque de series temporales, que encaja perfectamente con este problema. En un principio realicé pruebas con muy buenos resultados, utilizando este sistema. Después descarté las series temporales pensando en un futuro desarrollo ligado a ventas en establecimientos de hostelería dependiendo de clima y ocupación hotelera.

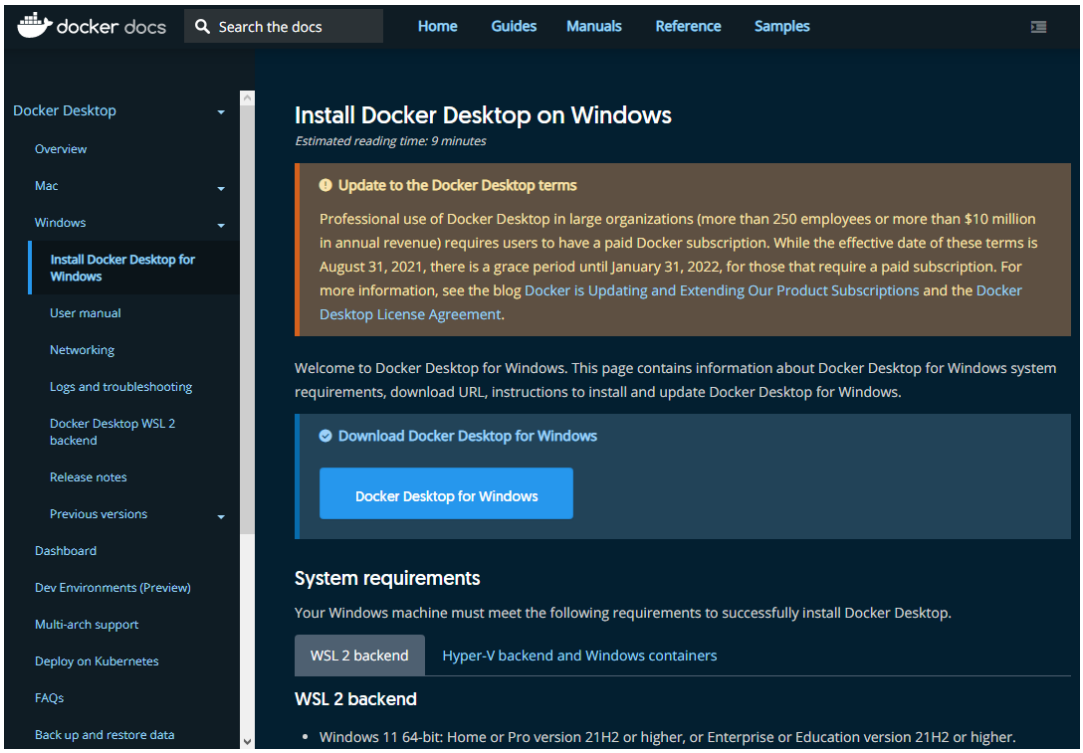
Los datos del clima se descargarán inicialmente y no se obtendrán de manera secuencial.

3.2 Preparación del entorno

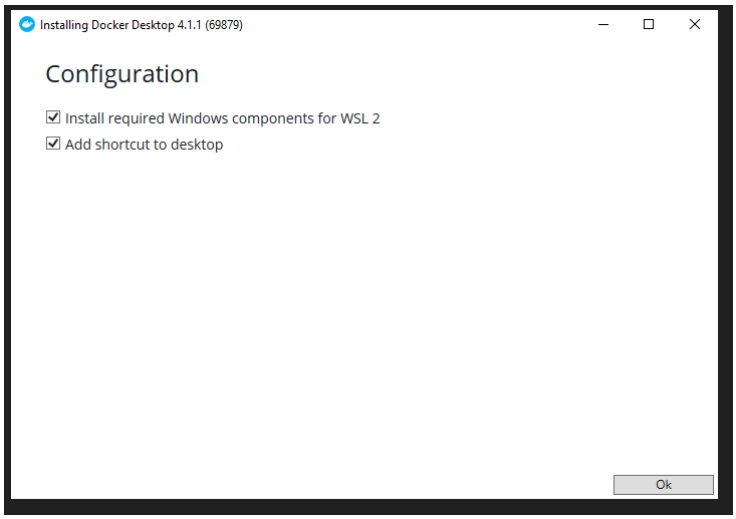
En una primera versión utilicé la herramienta de desarrollo de Google llamada Google Colab, es el Jutyter Notebook de Google en modelo SAAS, pero posteriormente he decidido aislar los entornos usando un contenedore y para ello he usado la tecnología docker.

3.2.1 Proceso de instalación de Docker Desktop

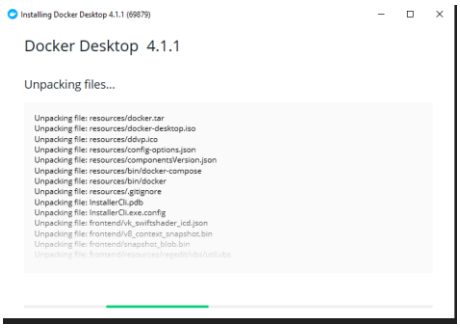
En la web de docker procedo a descargarlo para la versión de sistema operativo que utilizo, en mi caso Windows

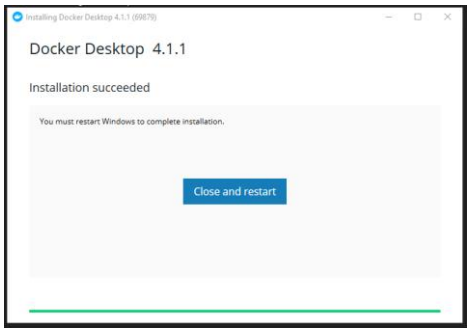


Una vez descargado procedemos con la ejecución del instalador, dejaremos por defecto los valores que se muestran en la imagen

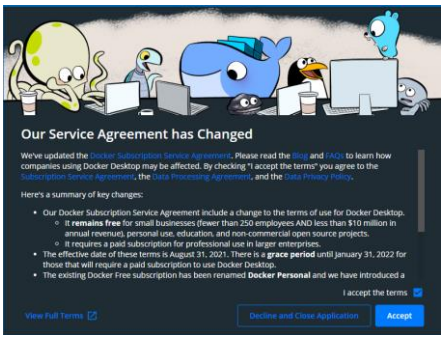


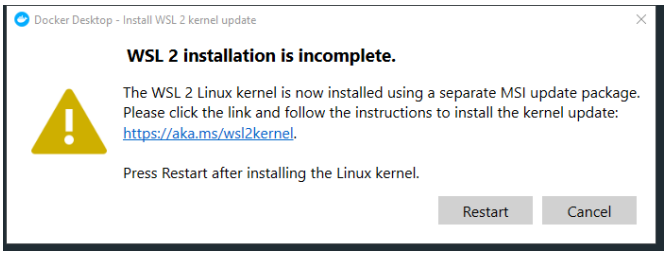
Esperando el proceso de instalación y pantalla de proceso completado



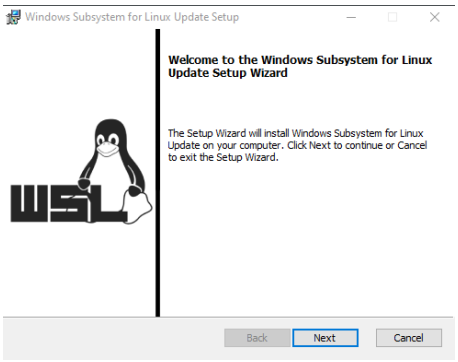


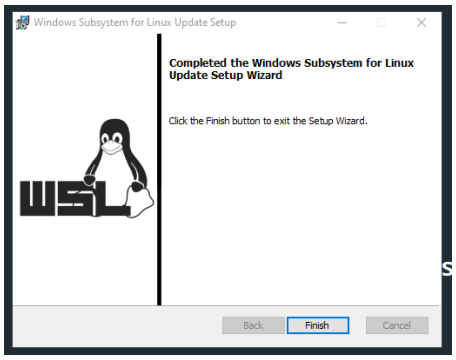
Pantallas de aceptación de términos del servicio y de instalación adicional



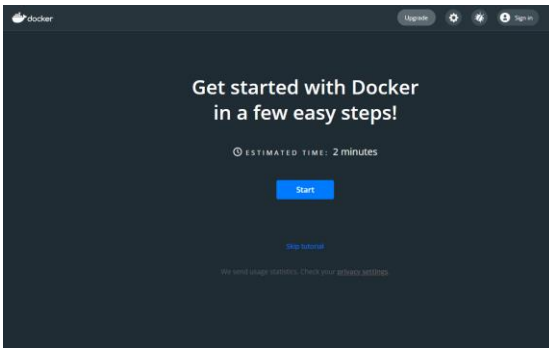


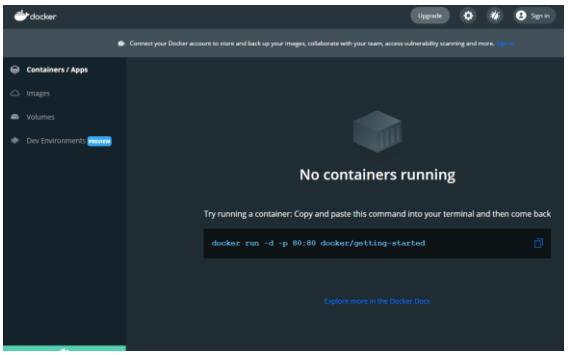
Proceso de instalación y pantalla de instalación finalizada



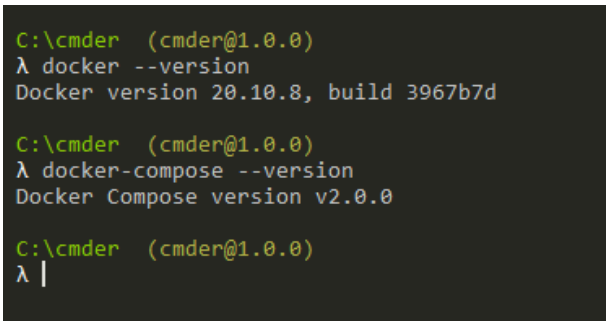


Inicio de aplicación Docker, finalización de la instalación y pantalla principal de la aplicación





Verificación mediante consola de la instalación de Docker y Docker-compose



3.2.2 Configuración del contenedor y puesta en marcha

Para realizar el TFM procederemos a configurar un contenedor para el uso de Jupiternotebook.

El primer paso es crear el fichero docker-compose.yml que contendrá los parámetros de configuración y montaje de sistema de almacenamiento.

```
docker-compose.yml (compose-spec.json)
version:  '3'
services:
  datascience-notebook:
    image:    jupyter/datascience-notebook
    volumes:
      - c:/TFM/work:/home/jovyan/work
      - c:/TFM/datasets:/home/jovyan/work/datasets
      - c:/TFM/models:/home/jovyan/work/models
    ports:
      - 8888:8888
    container_name:   jupyter_notebook
```

En mi caso las rutas son las que se muestran y que se montarán en el directorio c:/TFM/

Para lanzar el entorno, simplemente ejecutaremos el comando dentro del directorio donde tenemos el fichero docker-compose.yml

```
C:\TFM\container
λ docker compose up
[+] Running 1/0
 - Container jupyter_notebook  Created
Attaching to jupyter_notebook
jupyter_notebook  | Entered start.sh with args: jupyter lab
jupyter_notebook  | Executing the command: jupyter lab
jupyter_notebook  | [I 2022-07-01 15:15:22.475 ServerApp] jupyterlab | extension was successfully linked.
jupyter_notebook  | [W 2022-07-01 15:15:22.484 NotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp.
 your config before our next release.
jupyter_notebook  | [W 2022-07-01 15:15:22.484 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp
te your config before our next release.
jupyter_notebook  | [W 2022-07-01 15:15:22.484 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp
te your config before our next release.
jupyter_notebook  | [I 2022-07-01 15:15:22.500 ServerApp] nbclassic | extension was successfully linked.
jupyter_notebook  | [I 2022-07-01 15:15:23.114 ServerApp] notebook_shim | extension was successfully linked.
jupyter_notebook  | [I 2022-07-01 15:15:23.161 ServerApp] notebook_shim | extension was successfully loaded.
jupyter_notebook  | [I 2022-07-01 15:15:23.163 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.10/site-packages/jupyterlab
jupyter_notebook  | [I 2022-07-01 15:15:23.163 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
jupyter_notebook  | [I 2022-07-01 15:15:23.175 ServerApp] jupyterlab | extension was successfully loaded.
jupyter_notebook  | [I 2022-07-01 15:15:23.188 ServerApp] nbclassic | extension was successfully loaded.
jupyter_notebook  | [I 2022-07-01 15:15:23.189 ServerApp] Serving notebooks from local directory: /home/jovyan
jupyter_notebook  | [I 2022-07-01 15:15:23.189 ServerApp] Jupyter Server 1.17.1 is running at:
jupyter_notebook  | [I 2022-07-01 15:15:23.189 ServerApp] http://1173237cbfd2:8888/lab?token=f57495af4ee94cd2d3b034a51f9f298c6d78588794521487
jupyter_notebook  | [I 2022-07-01 15:15:23.189 ServerApp]  or http://127.0.0.1:8888/lab?token=f57495af4ee94cd2d3b034a51f9f298c6d78588794521487
jupyter_notebook  | [I 2022-07-01 15:15:23.190 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
jupyter_notebook  | [C 2022-07-01 15:15:23.198 ServerApp]
jupyter_notebook  |
jupyter_notebook  |     To access the server, open this file in a browser:
jupyter_notebook  |         file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html
jupyter_notebook  |     Or copy and paste one of these URLs:
jupyter_notebook  |         http://1173237cbfd2:8888/lab?token=f57495af4ee94cd2d3b034a51f9f298c6d78588794521487
jupyter_notebook  |      or http://127.0.0.1:8888/lab?token=f57495af4ee94cd2d3b034a51f9f298c6d78588794521487
```

Y tal y como se indica en la imagen tendremos disponible nuestro entorno y podemos acceder al él desde el navegador preferido poniendo una de las dos urls indicadas:

```
   http://1173237cbfd2:8888/lab?token=f57495af4ee94cd2d3b034a51f9f298c6d78588794521487
or http://127.0.0.1:8888/lab?token=f57495af4ee94cd2d3b034a51f9f298c6d78588794521487
```

3.3 EDA: Análisis exploratorio de datos

Para convertir los datos meteorológicos en información útil, comienzo importando el conjunto de datos descargados de AEMET. El conjunto de datos se llamará conjunto de datos climáticos.

Utilizo la librería pyaemet de Python para interactuar con OPENDATA de AEMET ya que dispone de funciones que me permite obtener la información organizada tal y como necesito.

3.3.1 Ingeniería y tratamiento de los datos

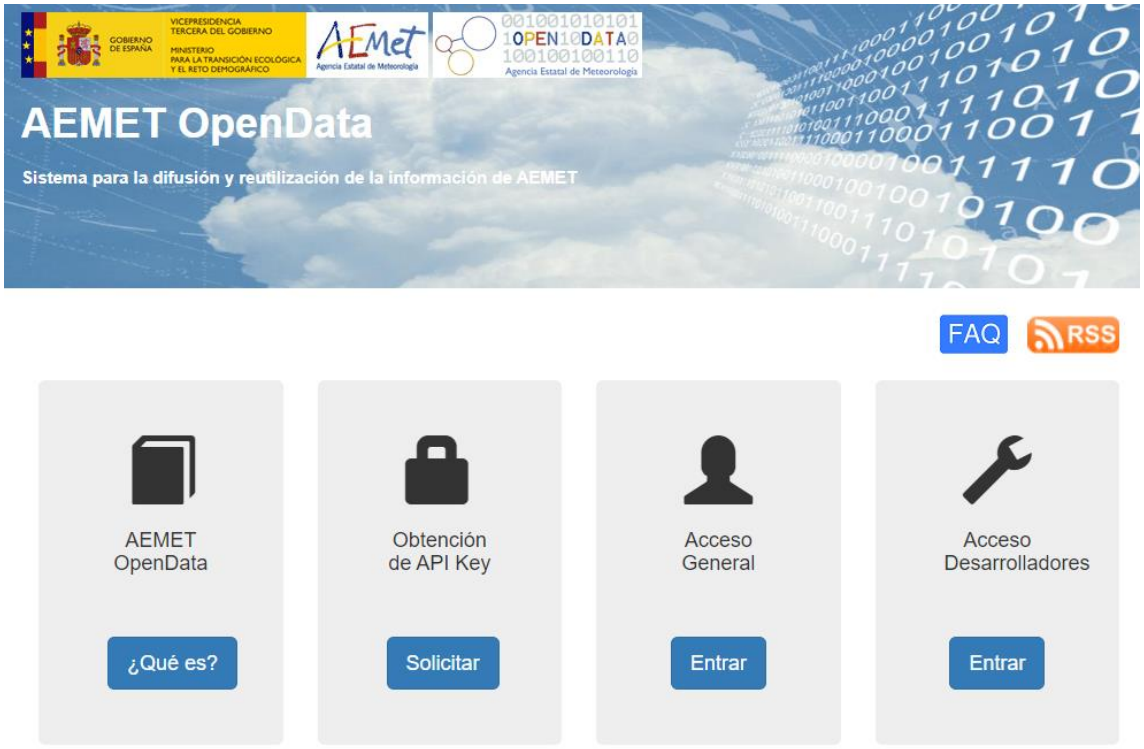
Los datos utilizados, como se menciona anteriormente, se recopilarán del servicio web API de nivel gratuito de AEMET OPENDATA que es un sistema para la difusión y reutilización de la información de la Agencia Estatal de Meteorología. (AEMET).

El funcionamiento de AEMET OpenData está indicado en su propia web https://opendata.aemet.es/centrodedescargas/inicio

Se trata de una API REST desarrollada por AEMET que permite la difusión y la reutilización de la información meteorológica y climatológica de la Agencia.

Para el uso de este API es necesario disponer de unas claves que se solicitan previamente en

https://opendata.aemet.es/centrodedescargas/inicio



Pulsamos en **Solicitar** y nos pide un correo, pasados unos minutos nos llegará la clave API a ese correo.

Para simplificar el proceso de extracción y obtención de los datos de Aemet, utilizo una librería que facilita el proceso, se llama pyaemet.

Jaimedgp/pyAEMET: Libreria de python para la descarga de datos climatologicos diarios de la AEMET mediante su API OpenData (github.com)

A continuación, aplico algunas transformaciones básicas a nuestros datos. Estas transformaciones incluyen:

- Limpiar los datos incompletos
- Eliminar los datos duplicados

También hay que ocuparse de los valores que faltan. Para tratar estos valores perdidos, utilizo técnicas de imputación, como la sustitución de la media y la imputación de los vecinos más cercanos.

Después de tratar estas situaciones, sólo se analizará el conjunto de datos de entrenamiento.

3.3.2 carga de datos

En el notebook, lo primero que hago es instalar la librería de acceso a los datos:

```
# Instalamos librería pyaemat (https://github.com/Jaimedgp/pyAEMET)
!pip install pyaemet
```

```
Collecting pyaemet
  Downloading pyaemet-1.0.2-py3-none-any.whl (22 kB)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-packages (from pyaemet) (2.28.0)
Collecting geocoder
  Downloading geocoder-1.38.1-py2.py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.6/98.6 kB 3.4 MB/s eta 0:00:00
Requirement already satisfied: setuptools>=42 in /opt/conda/lib/python3.10/site-packages (from pyaemet) (62.5.0
Requirement already satisfied: wheel in /opt/conda/lib/python3.10/site-packages (from pyaemet) (0.37.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages (from pyaemet) (1.22.4)
Requirement already satisfied: python-dateutil in /opt/conda/lib/python3.10/site-packages (from pyaemet) (2.8.2
Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-packages (from pyaemet) (1.4.3)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages (from geocoder->pyaemet) (1.16.0)
Collecting ratelim
  Downloading ratelim-0.1.6-py2.py3-none-any.whl (4.0 kB)
Requirement already satisfied: click in /opt/conda/lib/python3.10/site-packages (from geocoder->pyaemet) (8.1.3
Collecting future
  Downloading future-0.18.2.tar.gz (829 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 829.2/829.2 kB 5.0 MB/s eta 0:00:0000:0100:01
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-packages (from pandas->pyaemet) (
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests-
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests->pyaemet)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/conda/lib/python3.10/site-packages (from reque
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests->py
Requirement already satisfied: decorator in /opt/conda/lib/python3.10/site-packages (from ratelim->geocoder->py
Building wheels for collected packages: future
  Building wheel for future (setup.py) ... done
  Created wheel for future: filename=future-0.18.2-py3-none-any.whl size=491058 sha256=feb8534ad91e833cca65c934
  Stored in directory: /home/jovyan/.cache/pip/wheels/22/73/06/557dc4f4ef68179b9d763930d6eec26b88ed7c389b19588a
Successfully built future
Installing collected packages: ratelim, future, geocoder, pyaemet
Successfully installed future-0.18.2 geocoder-1.38.1 pyaemet-1.0.2 ratelim-0.1.6
```

Una vez finalizado el proceso ya tendré disponible la librería para su uso.

Tal y como se ha comentado anteriormente necesitamos un apikey para hacer las llamadas a OPENDATA , dado que el nivel de uso que voy a realizar de los datos es mínimo, no se requiere ninguna cuenta profesional, simplemente se necesita el nivel básico de acceso. Este apikey se obtiene en la web https://opendata.aemet.es/centrodedescargas/inicio poniendo nuestro correo electrónico, nos enviarán el api a nuestro email.

3.3.3 Interpretación, formato y unidades del dataset meteorológico

El primer conjunto de datos que uso es el de las estaciones de medición disponibles que contiene las siguientes características:

Característica	Descripción y formato
Latitud	Distancia de la estación en grados con formato decimal al ecuador
ProvinciaAemet	Provincia a la que pertenece la estación según AEMET
Altitud	Altitud de la estación sobre el nivel del mar en metros
Indicativo	Identificador de la estación de AEMET

Nombre	Nombre de la estación
Indsinop	Indicativo sinóptico (se utiliza para emitir los partes de observaciones meteorológicas)
Longitud	Distancia de la estación en grados con formato decimal al meridiano 0
Distrito	Distrito al que pertenece la estación
Ciudad	Ciudad a la que pertenece la estación
Provincia	Provincia de la estación según sus coordenadas
CA	Comunidad autónoma
País	País al que pertenece la estación

La descripción completa sobre las llamadas al API que se pueden hacer se puede consultar aquí: https://opendata.aemet.es/dist/index.html

En este caso me interesa saber las estaciones que se encuentran cerca de Illescas para posteriormente descargar el histórico de datos climáticos.

Por simplicidad no uso las llamadas al api directamente, lo hago a través de la librería pyament que ya dispone de unos métodos para filtrados muy útiles para este caso.

Como el objetivo es extraer todos los datos climáticos de una zona, necesito obtener un listado de localización de las distintas estaciones que se encuentren próximas a esa zona.

Para ello, el primer paso es obtener los datos de las estaciones disponibles, creando un dataframe que las almacene.

```
from pyaemet import AemetClima
aemet = AemetClima(apikey="eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqYXZpZXIuYmVybWVqb0BtZS5jb20iLCJqdGkiOiIzYzNlZGMwYi01OTAxLTQwOWEtOWU3MS04ZDg0M2M0OGFiMzAiL
# obtenemos la información de todas las estaciones
df_estaciones = aemet.estaciones_info()
df_estaciones.head()
```

```
Actualizando...
```

	latitud	provinciaAemet	altitud	indicativo	nombre	indsinop	longitud	distrito	ciudad	provincia	CA	pais
0	40.958056	TARRAGONA	32.0	0002I	VANDELLÒS	08169	0.871389		Vandellòs l'Hospitalet de l'Infant	Tarragona	Cataluña	ESP
1	41.145000	TARRAGONA	71.0	0016A	REUS AEROPUERTO	08175	1.163611		Reus	Tarragona	Catalunya	ESP
2	41.292778	BARCELONA	4.0	0076	BARCELONA AEROPUERTO	08181	2.070000		El Prat de Llobregat	Barcelona	Catalunya	ESP
3	41.720000	BARCELONA	291.0	0149X	MANRESA	08174	1.840278	Manresa	Manresa	Barcelona	Catalunya	ESP
4	41.418333	BARCELONA	408.0	0200E	BARCELONA, FABRA		2.124167	Barcelona	Barcelona	Barcelona	Catalunya	ESP

Se puede ver la cantidad de registros y columnas, así como las características (columnas) disponibles en el dataframe de las estaciones, son las siguientes:

```
print('Cantidad de Filas y columnas:',df_estaciones.shape)
print('Nombre columnas:',df_estaciones.columns)
```

```
Cantidad de Filas y columnas: (291, 12)
Nombre columnas: Index(['latitud', 'provinciaAemet', 'altitud', 'indicativo', 'nombre',
       'indsinop', 'longitud', 'distrito', 'ciudad', 'provincia', 'CA',
       'pais'],
      dtype='object')
```

Se puede ver que hay 291 estaciones disponibles repartidas por toda España.

Se puede representar en un mapa la localización de las estaciones por su latitud y longitud, para poder hacernos una idea de cómo se encuentran posicionadas en el mapa.

Para hacerlo de una forma sencilla podemos utilizar la librería “folium” que instalaré con el siguiente código:

```
!pip install folium
```

```
Collecting folium
  Downloading folium-0.12.1.post1-py2.py3-none-any.whl (95 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.0/95.0 kB 3.8 MB/s eta 0:00:00
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-packag
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages
Requirement already satisfied: jinja2>=2.9 in /opt/conda/lib/python3.10/site-pac
Collecting branca>=0.3.0
  Downloading branca-0.5.0-py3-none-any.whl (24 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.10/site
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/s
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-pa
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/conda/lib/pytho
Installing collected packages: branca, folium
Successfully installed branca-0.5.0 folium-0.12.1.post1
```

Ahora extraigo las características necesarias para la representación de su posición en el mapa, estas son las coordenadas de latitud y longitud junto con el identificador único de la estación

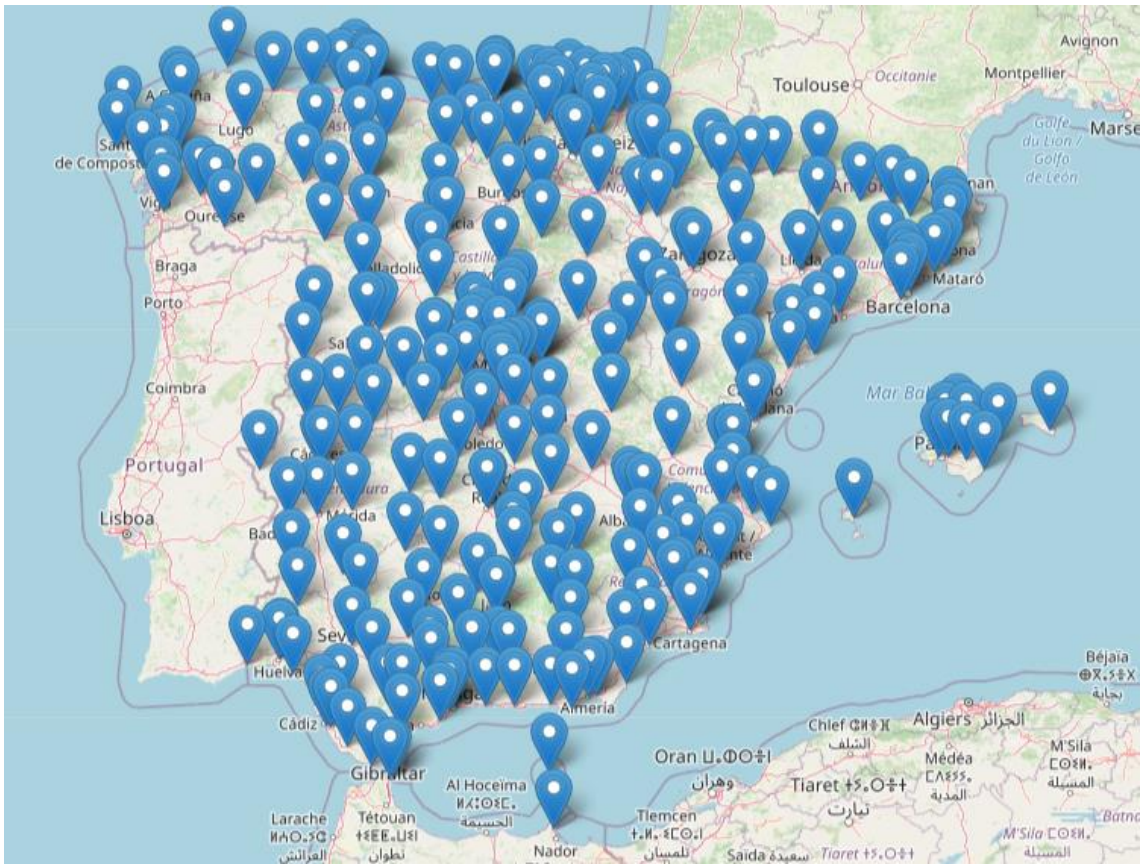
```
import pandas
import folium
# Necesitamos solamente latitud longitud y el indicativo
estaciones_locations = df_estaciones[['latitud','longitud','indicativo']]
estaciones_locations.head()
```

	latitud	longitud	indicativo
0	40.958056	0.871389	0002I
1	41.145000	1.163611	0016A
2	41.292778	2.070000	0076
3	41.720000	1.840278	0149X
4	41.418333	2.124167	0200E

Ahora procedo a crear el mapa y agrego cada estación al mismo

```
# creamos el mapa con las 291 estaciones marcadas
map = folium.Map(location=[estaciones_locations.latitud.mean(),
                           estaciones_locations.longitud.mean()],
                 zoom_start=6, control_scale=True)
#agregamos puntos al mapa
for index, location_info in estaciones_locations.iterrows():
    folium.Marker([location_info["latitud"], location_info["longitud"]],
                  popup=location_info["indicativo"]).add_to(map)
map
```

Esto genera el siguiente resultado:



De esta forma ya podremos desplazarnos por el mapa para visualizar la posición de las distintas estaciones.

Para la obtención de los datos climáticos que me interesan tengo que aplicar como filtro el indicativo de la estación ya que como veremos más adelante, necesito el indicativo de la estación para saber si se encuentra cerca de la posición de la zona que queremos. También necesitaré la latitud y la longitud, de esta forma tendré todos los datos necesarios para realizar un filtrado y obtener las estaciones cercanas a una latitud y longitud concreta.

Se podría realizar un filtro manualmente sobre el dataframe pero la librería pyaemet ya dispone de un método que nos permite realizar este filtro.

Ahora buscaré las 15 estaciones más cercanas a la localización de Illescas (latitud y longitud):

- latitud = 40.1358100
- longitud = -3.8571300

Para conseguir un dataframe con los datos de las estaciones cercanas podemos hacer uso de la librería pyaemet

```
# Estaciones cercanas a la población en la que resido
estaciones_cercanas = aemet.estaciones_cerca(latitud=40.1358100,
                                             longitud=-3.8571300,
                                             n_cercanas=15)
```

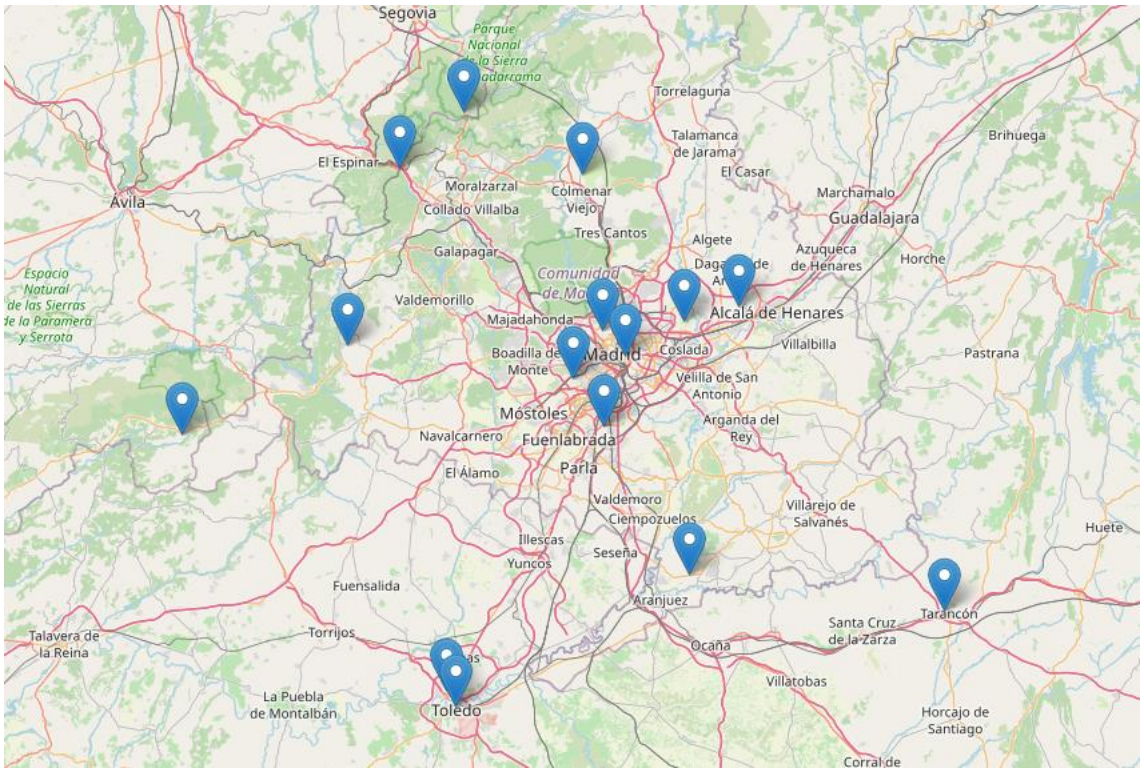
Este método devuelve los registros ordenados por distancia en orden ascendente:

```
estaciones_cercanas[['indicativo','nombre','distancia']]
```

	indicativo	nombre	distancia
109	3200	GETAFE	21.500724
108	3196	MADRID, CUATRO VIENTOS	27.331227
96	3100B	ARANJUEZ	27.530677
111	3260B	TOLEDO	32.191276
110	3259	TOLEDO, LORENZANA	33.914545
107	3195	MADRID, RETIRO	34.257622
106	3194U	MADRID, CIUDAD UNIVERSITARIA	36.887670
99	3129	MADRID AEROPUERTO	44.805384
114	3338	ROBLEDO DE CHAVELA	46.525564
104	3175	TORREJÓN DE ARDOZ	52.614665
105	3191E	COLMENAR VIEJO	62.788948
116	3391	SOTILLO DE LA ADRADA	64.577405
112	3266A	PUERTO ALTO DEL LEÓN	67.872081
95	3094B	TARANCÓN	72.444869
75	2462	PUERTO DE NAVACERRADA	74.225971

Si las posicionamos en el mapa como antes podemos ver la zona que se cubre

```
estaciones_cercanas_locations = estaciones_cercanas[['latitud','longitud','indicativo']]
# creamos el mapa con las estaciones cercanas
map = folium.Map(location=[estaciones_cercanas_locations.latitud.mean(),
                           estaciones_cercanas_locations.longitud.mean()],
                 zoom_start=10, control_scale=True)
#agregamos puntos al mapa
for index, location_info in estaciones_cercanas_locations.iterrows():
    folium.Marker([location_info["latitud"], location_info["longitud"]],
                  popup=location_info["indicativo"]).add_to(map)
map
```



La estación más cercana, es la de Getafe.

3.3.4 Interpretación, formato y unidades del dataset climático

Las características que usaremos para entrenamiento y el test de los modelos son las siguientes:

Característica	Descripción y formato
Fecha	Fecha de la lectura DD/MM/YYYY
Indicativo	Identificador de la estación de AEMET que realizó la lectura
Nombre	Nombre de la estación
Provincia	Provincia donde se encuentra la estación
Altitud	Altitud de la estación sobre el nivel del mar en metros
Tmed	Temperatura media del día en grados Celsius
Prec	Precipitaciones totales del día en mm de agua acumulada
Tmin	Temperatura mínima del día en grados Celsius
Horatmin	Hora en formato 24hs HH:MM en la que se registra la temperatura mínima del día
Tmax	Temperatura máxima del día en grados Celsius
Horatmax	Hora en formato 24hs HH:MM en la que se registra la temperatura máxima del día
Dir	Dirección del viento en grados
Velmedia	Velocidad media del viento en Km/h
Racha	Velocidad máxima del viento en Km/h
Horaracha	Hora en la que se registra la máxima velocidad del viento en formato 24h HH:MM
Sol	Horas de sol en el día

Presmax	Presión máxima registrada en bares
Horapresmax	Hora en formato 24hs HH:MM en la que se registra la máxima presión del día
Presmin	Presión mínima registrada en bares
Horapresmin	Hora en formato 24hs HH:MM en la que se registra la mínima presión del día

Para obtener los datos climáticos de esa estación uso el código:

```
from datetime import date

# Obtenemos los datos climáticos de la estación de Getafe
df_datos_clima = aemet.clima_diaria(estacion = 3200,
                                    fecha_ini=date(2015, 1, 1),
                                    fecha_fin=date(2022, 7, 1))
df_datos_clima.head()
```

	fecha	indicativo	nombre	provincia	altitud	tmed	prec	tmin	horatmin	tmax	horatmax	dir	velmedia	racha	horaracha	sol	presMax	horaPresMax	presMin	horaPresMin
0	2015-01-01	3200	GETAFE	MADRID	620	4.7	0.0	-3.4	7	12.8	15	24	0.0	3.1	18.0	8.8	964.0	23	958.8	1.0
1	2015-01-02	3200	GETAFE	MADRID	620	5.5	0.0	-3.0	8	14.0	14	99	0.6	2.5	-1.0	7.3	966.4	10	963.5	3.0
2	2015-01-03	3200	GETAFE	MADRID	620	6.1	0.0	-1.4	6	13.6	15	99	0.0	2.5	-1.0	8.9	966.6	10	963.3	16.0
3	2015-01-04	3200	GETAFE	MADRID	620	8.0	0.0	-1.0	7	17.0	15	4	0.0	2.5	2.0	8.9	963.8	0	958.5	24.0
4	2015-01-05	3200	GETAFE	MADRID	620	7.6	0.0	0.2	6	15.0	15	33	0.0	3.6	20.0	8.9	958.5	0	954.5	24.0

Se puede mostrar la cantidad de filas y columnas, así como el nombre de las mismas

```
print('Cantidad de Filas y columnas:',df_datos_clima.shape)
print('Nombre columnas:',df_datos_clima.columns)
```

```
Cantidad de Filas y columnas: (2740, 20)
Nombre columnas: Index(['fecha', 'indicativo', 'nombre', 'provincia', 'altitud', 'tmed', 'prec',
       'tmin', 'horatmin', 'tmax', 'horatmax', 'dir', 'velmedia', 'racha',
       'horaracha', 'sol', 'presMax', 'horaPresMax', 'presMin', 'horaPresMin'],
      dtype='object')
```

Entre las características de los datos obtenidos se puede ver que se muestra el indicativo de la estación que hemos seleccionado.

3.3.5 Tratamiento de datos incompletos

Para observar si hay valores nulos podemos usar el siguiente código

```
df_datos_clima.isnull().sum()
```

```
fecha          0
indicativo     0
nombre         0
provincia      0
altitud        0
tmed           0
prec           0
tmin           0
horatmin       0
tmax           0
horatmax       0
dir            0
velmedia       0
racha          0
horaracha      1
sol            0
presMax        0
horaPresMax    0
presMin        0
horaPresMin    1
dtype: int64
```

Se puede apreciar que hay 1 valor nulo en racha y otro en horaPresMin.

Dado que los valores faltantes son poco voy a utilizar el método “ffill” para rellenarlos con los valores anteriores con el código siguiente:

```
df_datos_clima.fillna(method="ffill", inplace=True)
```

```
df_datos_clima.isnull().sum()
```

```
fecha          0
indicativo     0
nombre         0
provincia      0
altitud        0
tmed           0
prec           0
tmin           0
horatmin       0
tmax           0
horatmax       0
dir            0
velmedia       0
racha          0
horaracha      0
sol            0
presMax        0
horaPresMax    0
presMin        0
horaPresMin    0
dtype: int64
```

Ahora ya no aparece ningún valor nulo, si bien, como se verá más a delante descartaré esas variables horas y no será necesario realizar el paso anterior.

3.3.6 Supresión de características con valores repetitivos y texto

En el dataframe climático podemos observar que hay varios campos en el que los valores están repetidos.

```
df_datos_clima.head()
```

	fecha	indicativo	nombre	provincia	altitud	tmed	prec	tmin	horatmin	tmax	horatmax	dir	velmedia	racha	horaracha	sol	presMax	horaPresMax	presMin	horaPresMin
0	2015-01-01	3200	GETAFE	MADRID	620	4.7	0.0	-3.4	7	12.8	15	24	0.0	3.1	18.0	8.8	964.0	23	958.8	1.0
1	2015-01-02	3200	GETAFE	MADRID	620	5.5	0.0	-3.0	8	14.0	14	99	0.6	2.5	-1.0	7.3	966.4	10	963.5	3.0
2	2015-01-03	3200	GETAFE	MADRID	620	6.1	0.0	-1.4	6	13.6	15	99	0.0	2.5	-1.0	8.9	966.6	10	963.3	16.0
3	2015-01-04	3200	GETAFE	MADRID	620	8.0	0.0	-1.0	7	17.0	15	4	0.0	2.5	2.0	8.9	963.8	0	958.5	24.0
4	2015-01-05	3200	GETAFE	MADRID	620	7.6	0.0	0.2	6	15.0	15	33	0.0	3.6	20.0	8.9	958.5	0	954.5	24.0

Se pueden eliminar las características indicativo, nombre, provincia y altitud ya que todos sus valores son iguales.

```
df_datos_clima.drop(['indicativo',
                     'nombre',
                     'provincia',
                     'altitud'
                     ], axis=1, inplace=True)
df_datos_clima.head()
```

	fecha	tmed	prec	tmin	horatmin	tmax	horatmax	dir	velmedia	racha	horaracha	sol	presMax	horaPresMax	presMin	horaPresMin
0	2015-01-01	4.7	0.0	-3.4	7	12.8	15	24	0.0	3.1	18.0	8.8	964.0	23	958.8	1.0
1	2015-01-02	5.5	0.0	-3.0	8	14.0	14	99	0.6	2.5	-1.0	7.3	966.4	10	963.5	3.0
2	2015-01-03	6.1	0.0	-1.4	6	13.6	15	99	0.0	2.5	-1.0	8.9	966.6	10	963.3	16.0
3	2015-01-04	8.0	0.0	-1.0	7	17.0	15	4	0.0	2.5	2.0	8.9	963.8	0	958.5	24.0
4	2015-01-05	7.6	0.0	0.2	6	15.0	15	33	0.0	3.6	20.0	8.9	958.5	0	954.5	24.0

HoraPresMax, HoraPresMin, horaracha, horatmin,horatmax son variables que hacen referencia a la hora del día en la que se producen esos fenómenos, dado que voy a tratar los datos a nivel diario considero que no son muy relevantes e igualmente las elimino.

```
df_datos_clima.drop(['horatmin',
                     'horatmax',
                     'horaracha',
                     'horaPresMax',
                     'horaPresMin',
                     ], axis=1, inplace=True)
df_datos_clima.head()
```

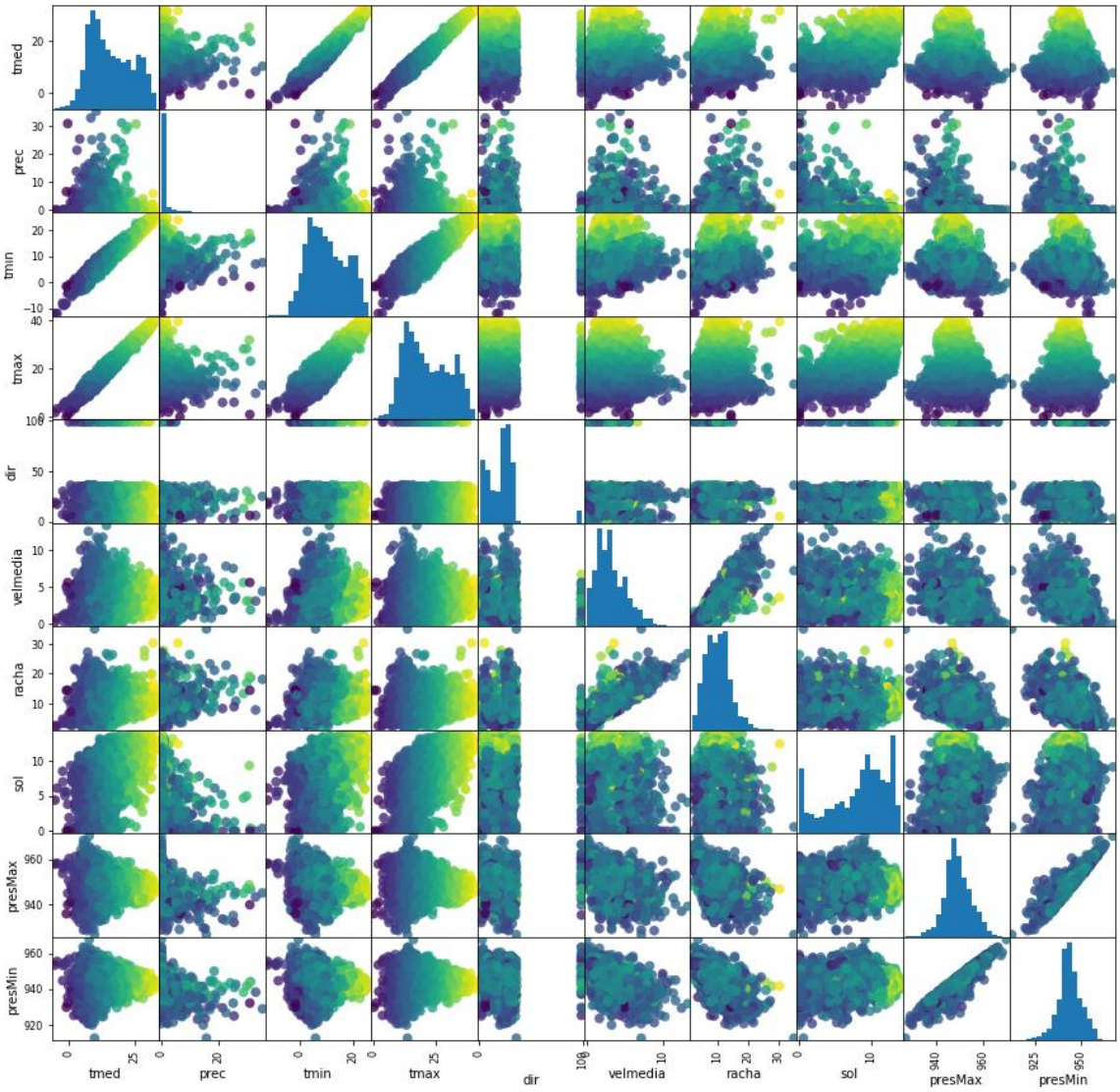
	fecha	tmed	prec	tmin	tmax	dir	velmedia	racha	sol	presMax	presMin
0	2015-01-01	4.7	0.0	-3.4	12.8	24	0.0	3.1	8.8	964.0	958.8
1	2015-01-02	5.5	0.0	-3.0	14.0	99	0.6	2.5	7.3	966.4	963.5
2	2015-01-03	6.1	0.0	-1.4	13.6	99	0.0	2.5	8.9	966.6	963.3
3	2015-01-04	8.0	0.0	-1.0	17.0	4	0.0	2.5	8.9	963.8	958.5
4	2015-01-05	7.6	0.0	0.2	15.0	33	0.0	3.6	8.9	958.5	954.5

3.3.7 Correlación de los datos para la selección de variables

Antes de empezar con el preprocesado de datos y la creación de los modelos realizo un análisis de calidad, distribución y correlación para detectar posibles anomalías.

Se puede realizar una gráfica sencilla como la siguiente:

```
import matplotlib.pyplot as plt

g = pd.plotting.scatter_matrix(df_datos_clima,c=round(df_datos_clima['tmax']), figsize=(15, 15), marker='o',
                               hist_kwds={'bins':20}, s=60, alpha=.8)

plt.show()
```



Igualmente uso los valores numéricos para obtener más información:

```
df_datos_clima.describe()
```

	tmed	prec	tmin	tmax	dir	velmedia	racha	sol	presMax	presMin
count	2740.000000	2740.000000	2740.000000	2740.000000	2740.000000	2740.000000	2740.000000	2740.000000	2740.000000	2740.000000
mean	16.151496	0.965036	10.433540	21.869854	22.090511	3.259197	10.259051	8.246642	948.912117	944.384307
std	7.889246	3.409625	7.180128	8.913713	15.632924	1.974873	4.032233	3.978257	5.697295	6.335752
min	-5.100000	0.000000	-12.000000	0.000000	1.000000	0.000000	1.900000	0.000000	926.400000	912.400000
25%	9.600000	0.000000	4.675000	14.200000	10.000000	1.900000	7.200000	5.600000	945.500000	940.900000
50%	14.900000	0.000000	9.600000	20.600000	24.000000	2.800000	10.300000	9.200000	948.400000	944.100000
75%	22.700000	0.000000	16.400000	29.400000	29.000000	4.400000	12.800000	11.400000	952.500000	948.100000
max	34.600000	41.400000	26.600000	42.600000	99.000000	13.300000	35.000000	14.000000	970.300000	967.800000

Variable precipitaciones: Se puede observar que los datos se agrupan entorno al valor 0, teniendo una media de 0,96.

Variable temperatura máxima no se observan grandes valores anómalos y se puede ver que hay una temperatura máxima de 42,6 grados.

Variable temperatura mínima con una media de 10,43 grados y un valor mínimo de -12 grados (coincidiendo con el temporal Filomena).

El análisis de la correlación entre variables es muy importante ya que algunos algoritmos se ven fuertemente afectados si existe una relación fuerte entre ellas.

El coeficiente de correlación de Pearson es una medida del grado de relación lineal entre dos variables. Indica si existe una relación positiva o negativa entre las dos variables y también puede utilizarse para cuantificar lo fuerte que es esa relación.

El método que la librería pandas utiliza por defecto, para calcular la correlación entre variables, es el método de Pearson, aunque se pueden indicar otros como el de kendall y spearman mediante el parámetro de method.

Para observar la correlación de los datos de una forma más sencilla se puede utilizar una gráfica como la siguiente:

```
corr.style.background_gradient(cmap='coolwarm')
```

	tmed	prec	tmin	tmax	dir	velmedia	racha	sol	presMax	presMin
tmed	1.000000	-0.124482	0.975616	0.984260	0.013075	0.038383	0.138372	0.591944	-0.212316	-0.111695
prec	-0.124482	1.000000	-0.056048	-0.175101	-0.019897	0.101190	0.211514	-0.390104	-0.277094	-0.315818
tmin	0.975616	-0.056048	1.000000	0.921477	0.011298	0.124730	0.210198	0.456927	-0.294694	-0.199824
tmax	0.984260	-0.175101	0.921477	1.000000	0.014056	-0.032520	0.075673	0.679713	-0.138451	-0.036796
dir	0.013075	-0.019897	0.011298	0.014056	1.000000	0.065999	0.026400	0.024800	-0.093060	-0.091891
velmedia	0.038383	0.101190	0.124730	-0.032520	0.065999	1.000000	0.816723	-0.085916	-0.443249	-0.503905
racha	0.138372	0.211514	0.210198	0.075673	0.026400	0.816723	1.000000	-0.041941	-0.493341	-0.563654
sol	0.591944	-0.390104	0.456927	0.679713	0.024800	-0.085916	-0.041941	1.000000	0.080628	0.147760
presMax	-0.212316	-0.277094	-0.294694	-0.138451	-0.093060	-0.443249	-0.493341	0.080628	1.000000	0.934069
presMin	-0.111695	-0.315818	-0.199824	-0.036796	-0.091891	-0.503905	-0.563654	0.147760	0.934069	1.000000

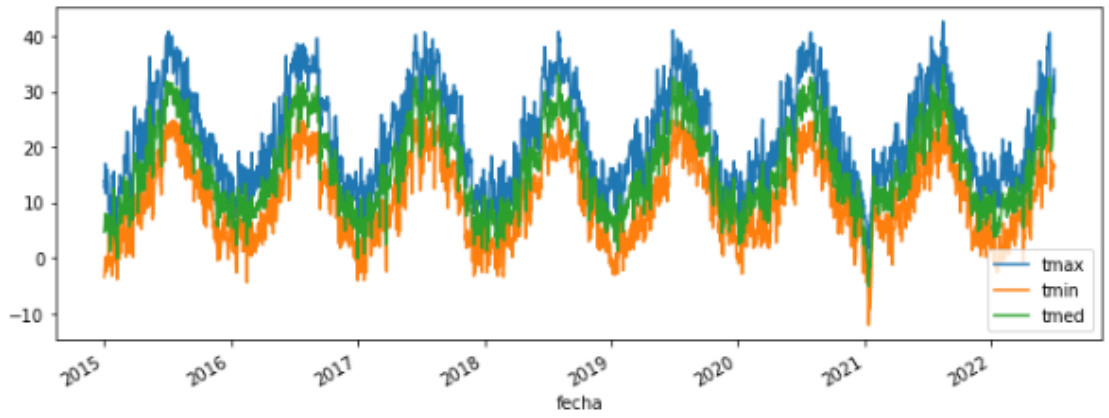
De esta forma se observa que hay una fuerte relación entre temperatura media (tmed), temperatura mínima (tmin) y temperatura máxima (tmax). Por otro lado, se ve también que

existe una relación entre la presión máxima (preMax) y presión mínima(preMin). Por último, se observa otra relación entre la racha de viento máxima(racha) y la velocidad media del viento (velmedia) con un valor de 0,81.

Se puede apreciar la fuerte correlación de las variables indicadas de forma visual

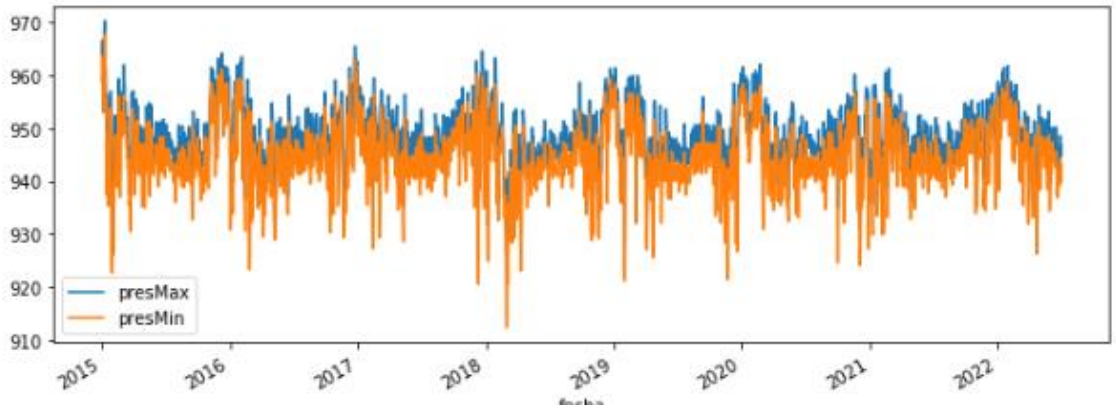
```
df_datos_clima.plot(x="fecha", y=["tmax","tmin", "tmed"], figsize=(11,4))
```

```
<AxesSubplot:xlabel='fecha'>
```



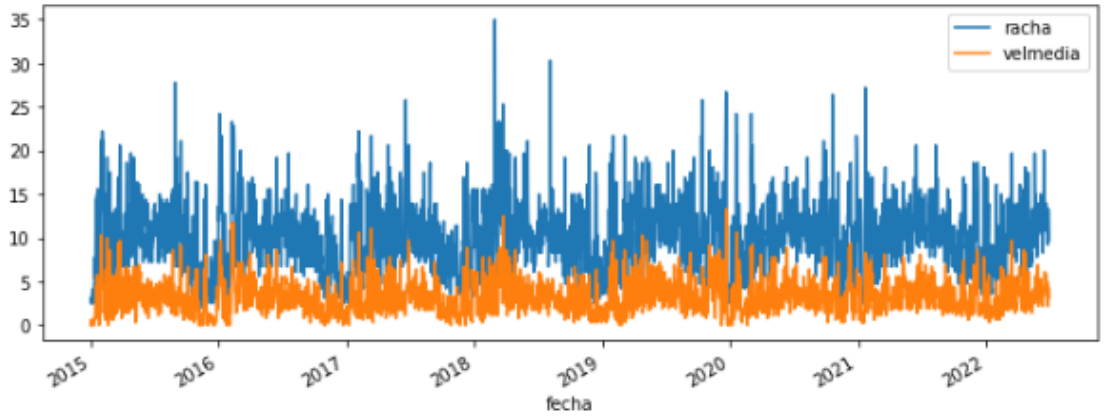

```
df_datos_clima.plot(x="fecha", y=["presMax","presMin"], figsize=(11,4))
```

```
<AxesSubplot:xlabel='fecha'>
```




```
df_datos_clima.plot(x="fecha", y=["racha","velmedia"], figsize=(11,4))
```

```
<AxesSubplot:xlabel='fecha'>
```



Dado que algunos algoritmos son especialmente sensibles a las relaciones fuertes, debería ver la posibilidad de eliminarlo y dejar solo uno de los valores relacionados. Voy a dejar uno de los valores en el caso de la temperatura, como la que se va a predecir es la temperatura máxima, dejaré esta y la temperatura media, eliminando la temperatura mínima.

Respecto a la presión, decido dejarla ya que, aunque existe una fuerte correlación, nos está dando el rango en el que se mueve la presión en un día.

```
df_datos_clima.drop(['tmed'], axis=1, inplace=True)
df_datos_clima.head()
```

	fecha	prec	tmin	tmax	dir	velmedia	racha	sol	presMax	presMin
0	2015-01-01	0.0	-3.4	12.8	24	0.0	3.1	8.8	964.0	958.8
1	2015-01-02	0.0	-3.0	14.0	99	0.6	2.5	7.3	966.4	963.5
2	2015-01-03	0.0	-1.4	13.6	99	0.0	2.5	8.9	966.6	963.3
3	2015-01-04	0.0	-1.0	17.0	4	0.0	2.5	8.9	963.8	958.5
4	2015-01-05	0.0	0.2	15.0	33	0.0	3.6	8.9	958.5	954.5

Ahora tendré unas correlaciones más adecuadas:

```
#mostramos la tabla de correlacción entre la variables
df_corr = df_datos_clima.drop(['fecha'], axis=1)
corr=df_corr.corr()
corr.style.background_gradient(cmap='coolwarm')
```

	prec	tmin	tmax	dir	velmedia	racha	sol	presMax	presMin
prec	1.000000	-0.056048	-0.175101	-0.019897	0.101190	0.211514	-0.390104	-0.277094	-0.315818
tmin	-0.056048	1.000000	0.921477	0.011298	0.124730	0.210198	0.456927	-0.294694	-0.199824
tmax	-0.175101	0.921477	1.000000	0.014056	-0.032520	0.075673	0.679713	-0.138451	-0.036796
dir	-0.019897	0.011298	0.014056	1.000000	0.065999	0.026400	0.024800	-0.093060	-0.091891
velmedia	0.101190	0.124730	-0.032520	0.065999	1.000000	0.816723	-0.085916	-0.443249	-0.503905
racha	0.211514	0.210198	0.075673	0.026400	0.816723	1.000000	-0.041941	-0.493341	-0.563654
sol	-0.390104	0.456927	0.679713	0.024800	-0.085916	-0.041941	1.000000	0.080628	0.147760
presMax	-0.277094	-0.294694	-0.138451	-0.093060	-0.443249	-0.493341	0.080628	1.000000	0.934069
presMin	-0.315818	-0.199824	-0.036796	-0.091891	-0.503905	-0.563654	0.147760	0.934069	1.000000

Al utilizar fechas y no plantear un modelo de series temporal, tengo que conseguir dar valores continuos a los valores de fechas, para ello lo realizaré mediante el uso de cosenos.

Dado que el enfoque que daré no será un modelo de serie temporal, se debe tener en cuenta que los valores de fecha no son continuos. Es decir, no tienen un valor específico asociado a ellos: son sólo un número que representa el día del mes.

Necesito que estos valores sean continuos (por ejemplo, algo como 0.25 para el día 25 del mes), entonces usar fechas no va a funcionar - necesitaré otro enfoque.

Una forma es utilizar cosenos en lugar de fechas para representar los datos de la serie temporal con valores enteros para cada día. Esto permite dar valores continuos para los días sin tener que preocuparnos de qué día concreto es (por ejemplo, el 1 o el 30). Tendré que modificar el campo fecha para conseguir dar valores continuos, en este caso usaremos los cosenos.

Este enfoque es el que más me ha gustado y lo he sacado de la guía “Stop One-Hot Encoding your Time-based Features” de Satyam Kumar, disponible en https://towardsdatascience.com/stop-one-hot-encoding-your-time-based-features-24c699face2f

Para realizar estas transformaciones creo las variables ”mes”,”dia”, además de las siguientes funciones que usaremos más adelante.

```
# Creamos campos de día y mes
df_datos_clima['mes']=df_datos_clima['fecha'].dt.month
df_datos_clima['dia']=df_datos_clima['fecha'].dt.day
```

```
def tranformar_fechas(df):
    #funcion para transformar fechas
    df['dia'] = np.cos(((2*np.pi)/31)*df['dia'])
    df['mes'] = np.cos(((2*np.pi)/12)*df['mes'])
```

Aplicamos la función:

```
tranformar_fechas(df_datos_clima)
df_datos_clima.head()
```

	fecha	prec	tmin	tmax	dir	velmedia	racha	sol	presMax	presMin	mes	dia
0	2015-01-01	0.0	-3.4	12.8	24	0.0	3.1	8.8	964.0	958.8	0.866025	0.979530
1	2015-01-02	0.0	-3.0	14.0	99	0.6	2.5	7.3	966.4	963.5	0.866025	0.918958
2	2015-01-03	0.0	-1.4	13.6	99	0.0	2.5	8.9	966.6	963.3	0.866025	0.820763
3	2015-01-04	0.0	-1.0	17.0	4	0.0	2.5	8.9	963.8	958.5	0.866025	0.688967
4	2015-01-05	0.0	0.2	15.0	33	0.0	3.6	8.9	958.5	954.5	0.866025	0.528964

```
tranformar_fechas(df_datos_clima)
df_datos_clima.head()
```

	fecha	prec	tmin	horatmin	tmax	horatmax	dir	velmedia	horaracha	sol	presMax	horaPresMax	horaPresMin	mes	dia
0	2015-01-01	0.0	-3.4	7	12.8	15	24	0.0	18.0	8.8	964.0	23	1.0	0.866025	0.979530
1	2015-01-02	0.0	-3.0	8	14.0	14	99	0.6	-1.0	7.3	966.4	10	3.0	0.866025	0.918958
2	2015-01-03	0.0	-1.4	6	13.6	15	99	0.0	-1.0	8.9	966.6	10	16.0	0.866025	0.820763
3	2015-01-04	0.0	-1.0	7	17.0	15	4	0.0	2.0	8.9	963.8	0	24.0	0.866025	0.688967
4	2015-01-05	0.0	0.2	6	15.0	15	33	0.0	20.0	8.9	958.5	0	24.0	0.866025	0.528964

3.3.8 Particionado de los datos

Para realizar la separación entre los datos de entrenamiento y los datos de test, he creado una función que separa los datos según la fecha y las variables pasadas como argumentos "features" y "target".

El 85% de los datos serán utilizados para entrenar el modelo y el 15% para probarlo.

La función también devuelve un conjunto para la validación

```
def train_test_val_split(df, features, targets, percentage_test, percentage_val=0.0):
    '''
    Funcion para particionar los datos manteniendo fechas
    :argumento df: Dataframe
    :argumento features: lista de variables X
    :argumento targets: lista de variables y
    :argumento percentage_test: porcentaje de registros para test
    :argumento percentage_val: porcentaje de registros para validacion, por defecto 0.0
    :devuelve: X_train, X_test, y_train, y_test, X_validation, y_validation
    '''
    # Establecemos porcentajes
    total = len(df.index)
    row_test = np.round(total-(total*(percentage_test+percentage_val)), 0).astype(int)
    row_valalidation = np.round(total-(total*percentage_val), 0).astype(int)

    train = df.iloc[:row_test]
    test = df.iloc[row_test:row_validation]
    validation = df.iloc[row_validation:]

    # Separamos X e y para cada conjunto de datos
    X_train = train[features]
    y_train = train[targets]

    X_test = test[features]
    y_test = test[targets]

    X_validation = validation[features]
    y_validation = validation[targets]

    return X_train, X_test, y_train, y_test, X_validation, y_validation
```

Defino las características que voy a utilizar

```
# Definimos variables X
features_list=['mes',
               'dia',
               'presMax',
               'presMin',
               'sol',
               'tmin',
               'prec',
               'velmedia',
               'racha']
# Definimos la variable a predecir y
target_list=['tmax']
```

Ahora se pueden generar los distintos conjuntos de datos usando

```
X_train,X_test,y_train,y_test,X_validation,y_validacion=train_test_val_split(df_datos_clima,
                                                                             features_list,
                                                                             target_list,
                                                                             .15,
                                                                             0)
```

Podemos ver cómo queda el dataframe del x_train que será utilizado para el entrenamiento de los modelos

```
X_train.head()
```

	mes	dia	presMax	presMin	sol	tmin	prec	velmedia	racha
0	0.866025	0.979530	964.0	958.8	8.8	-3.4	0.0	0.0	3.1
1	0.866025	0.918958	966.4	963.5	7.3	-3.0	0.0	0.6	2.5
2	0.866025	0.820763	966.6	963.3	8.9	-1.4	0.0	0.0	2.5
3	0.866025	0.688967	963.8	958.5	8.9	-1.0	0.0	0.0	2.5
4	0.866025	0.528964	958.5	954.5	8.9	0.2	0.0	0.0	3.6

3.3.9 Escalado de datos

Ahora todas las variables de las que se dispone en nuestro dataframe son numéricas.

Para evitar que ciertos valores tengan mucho peso dentro del modelo y que los valores tengan un peso relativo en su variable, se realiza un escalado de las variables que se van a utilizar como entrada del modelo.

Para ello utilizaré la librería de skalearn y procederé a instalarlo con este comando:

```
!pip install sklearn
```

Las variables 'presMax','presMin','sol','tmin','prec','velmedia','racha' se escalan para obtener puntos de datos más homogéneos, en este caso usando un escalado normal con la función standardScaler.

```
#Escalar
from sklearn.preprocessing import StandardScaler

columnas_scalar=['presMax','presMin','sol','tmin','prec','velmedia','racha']

# Entrenamos scaler solo con los datos del conjunto de entrenamiento
scaler=StandardScaler().fit(X_train[columnas_scalar])

X_train[columnas_scalar]=scaler.transform(X_train[columnas_scalar])
X_test[columnas_scalar]=scaler.transform(X_test[columnas_scalar])
X_train.head()
```

	mes	dia	presMax	presMin	sol	tmin	prec	velmedia	racha
0	0.866025	0.979530	2.520970	2.175551	0.144319	-1.906353	-0.291576	-1.566068	-1.698132
1	0.866025	0.918958	2.924279	2.886523	-0.235769	-1.850594	-0.291576	-1.275033	-1.842418
2	0.866025	0.820763	2.957888	2.856269	0.169658	-1.627558	-0.291576	-1.566068	-1.842418
3	0.866025	0.688967	2.487361	2.130170	0.169658	-1.571799	-0.291576	-1.566068	-1.842418
4	0.866025	0.528964	1.596721	1.525088	0.169658	-1.404521	-0.291576	-1.566068	-1.577894

Para entrenar scaler utilizo los datos del conjunto de entrenamiento.

3.4 Predicción mediante aprendizaje automático

3.4.1 Evaluación de los modelos

Voy a realizar las pruebas sobre los siguientes regresores:

- Regresión Lineal (que se usará como base)
- k-neighbors
- Árbol de decisión
- Bagging K-Neighbors
- XGBoost (Boosting)
- LightGBM (Boosting)

Respecto a los sistemas de métricas que usaré son:

```
# Métricas que parasemos al scoring
scoring_metrics=['neg_root_mean_squared_error','r2','neg_mean_absolute_error']
```

Entreno los modelos ajustando los parámetros de entrada intentando obtener las mejores métricas posibles y para ello voy a utilizar la función GridSearchCV de sklearn. Como anteriormente he indicado, voy a utilizar varias métricas, he puesto la opción refit, para entrenar de nuevo el mejor modelo usando la métrica RMSE

3.4.1.1 Regresión lineal

Primero empiezo con el modelo de regresión lineal que utilizo como modelo base

```
# regresión linear
from sklearn.linear_model import LinearRegression

reg_ln=LinearRegression()
reg_ln.fit(X_train,y_train['tmax'])
y_pred_ln=reg_ln.predict(X_test)
```

3.4.1.2 Modelo K-Neighbors

Ahora empiezo con modelos sencillos, Modelo K-Neighbors

```
# K-neighbors
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV

param_KN={"n_neighbors":np.arange(3,100)}

reg_KN=GridSearchCV(KNeighborsRegressor(),
                    param_grid=param_KN,
                    refit='neg_root_mean_squared_error',
                    scoring=scoring_metrics)

reg_KN.fit(X_train,y_train['tmax'])

y_pred_KN=reg_KN.predict(X_test)
reg_KN.best_params_
```

```
{'n_neighbors': 8}
```

3.4.1.3 Modelo de árbol de decisión

```
# árbol de decisión
from sklearn.tree import DecisionTreeRegressor

params_DT={'max_depth':range(3,10),'min_samples_leaf':range(1,5)}

reg_DT=GridSearchCV(DecisionTreeRegressor(random_state=7),
                    param_grid=params_DT,
                    refit='neg_root_mean_squared_error',
                    scoring=scoring_metrics)

reg_DT.fit(X_train,y_train['tmax'])

y_pred_DT=reg_DT.predict(X_test)
reg_DT.best_params_
```

```
{'max_depth': 7, 'min_samples_leaf': 4}
```

3.4.1.4 Modelo Bagging

También se prueba un Modelo Bagging basado en el algoritmo KNeighbor ya probado, para ello uso la función BaggingRegressor de sklearn.

```
# Modelo Bagging K-Neighbors
from sklearn.ensemble import BaggingRegressor

params_BKN={
    "base_estimator__n_neighbors":np.arange(3,20),
    "n_estimators":np.arange(3,20),
    "max_features":np.arange(.5,1,.1)
}

reg_BKN=GridSearchCV(BaggingRegressor(KNeighborsRegressor()),
                     param_grid=params_BKN,
                     refit='neg_root_mean_squared_error',
                     scoring=scoring_metrics)

reg_BKN.fit(X_train,y_train['tmax'])
y_pred_BKN=reg_BKN.predict(X_test)
reg_BKN.best_params_
```

```
{'base_estimator__n_neighbors': 5,
 'max_features': 0.8999999999999999,
 'n_estimators': 19}
```

3.4.1.5 Modelos Boosting

Ahora los Modelos Boosting, los utilizados son XGBoost y LigthGBM, ambos se basan en árboles de decisión.

3.4.1.5.1 Modelo XGBoost

Instalo la librería

```
!pip install xgboost
```

```
# XGBoost
from xgboost import XGBRegressor

params_xgb={
    'max_depth':range(3,6),
    'n_estimators':range(60,80,1),
    'learning_rate':np.arange(.11,.13,.005),
    'subsample':np.arange(.4,0.6,0.05),
    'colsample_bytree':np.arange(.5,1,0.1)
    }

reg_XGB=GridSearchCV(XGBRegressor(objective='reg:squarederror',random_state=7),
                     param_grid=params_xgb,
                     refit='neg_root_mean_squared_error',
                     scoring=scoring_metrics)

reg_XGB.fit(X_train,y_train['tmax'])
y_pred_XGB=reg_XGB.predict(X_test)

print(reg_XGB.best_params_)
print(reg_XGB.best_estimator_)
```

```
{'colsample_bytree': 0.8999999999999999, 'learning_rate': 0.115, 'max_depth': 4, 'n_estimators': 65, 'subsample': 0.55}
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1,
             colsample_bytree=0.8999999999999999, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, gamma=0, gpu_id=-1,
             grow_policy='depthwise', importance_type=None,
             interaction_constraints='', learning_rate=0.115, max_bin=256,
             max_cat_to_onehot=4, max_delta_step=0, max_depth=4, max_leaves=0,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=65, n_jobs=0, num_parallel_tree=1, predictor='auto',
             random_state=7, reg_alpha=0, reg_lambda=1, ...)
```

3.4.1.5.2 Modelo LightGBM

Instalo la librería

```
!pip install lightgbm
```

```
# LightGBM
from lightgbm import LGBMRegressor

params_LGBM={
    'max_depth':range(3,6),
    'n_estimators':range(110,130,1),
    'learning_rate':np.arange(.09,.11,.005),
    'subsample':np.arange(.3,0.6,0.05),
    'colsample_bytree':np.arange(.5,1,0.1)
    }

reg_LGBM=GridSearchCV(LGBMRegressor(random_state=7),
                      param_grid=params_LGBM,
                      refit='neg_root_mean_squared_error',
                      scoring=scoring_metrics)

reg_LGBM.fit(X_train,y_train['tmax'])
y_pred_LGBM=reg_LGBM.predict(X_test)

print(reg_LGBM.best_params_)
print(reg_LGBM.best_estimator_)
```

```
{'colsample_bytree': 0.7999999999999999, 'learning_rate': 0.09, 'max_depth': 5, 'n_estimators': 112, 'subsample': 0.3}
LGBMRegressor(colsample_bytree=0.7999999999999999, learning_rate=0.09,
              max_depth=5, n_estimators=112, random_state=7, subsample=0.3)
```

4 Resultados

Para la evaluación de los resultados creo una función

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def resultados(y_real, y_pred, model):
  results = {}
  results['Modelo'] = [model]

  # calculamos MAE
  result_mae = mean_absolute_error(y_real, y_pred)
  results['MAE'] = [result_mae]

  # Calculamos RMSE
  result_rmse = mean_squared_error(y_real, y_pred, squared=False)
  results['RMSE'] = [result_rmse]

  # calculamos R2
  results['R2'] = [r2_score(y_real, y_pred)]

  return pd.DataFrame.from_dict(data=results)
```

Realizo la evaluación de cada modelo

```
#Evaluación
from sklearn.metrics import mean_absolute_error,mean_squared_error
import pandas as pd
pd_ln=resultados(y_test['tmax'],y_pred_ln,'Regresión Linear (Modelo base)')
pd_kn=resultados(y_test['tmax'],y_pred_KN,'KNeighbors')
pd_bkn=resultados(y_test['tmax'],y_pred_BKN,'Bagging KNeighbors')
pd_DT=resultados(y_test['tmax'],y_pred_DT,'DecissionTree')
pd_XGB=resultados(y_test['tmax'],y_pred_XGB,'XGBoost')
pd_LGBM=resultados(y_test['tmax'],y_pred_LGBM,'LightGBM')
```

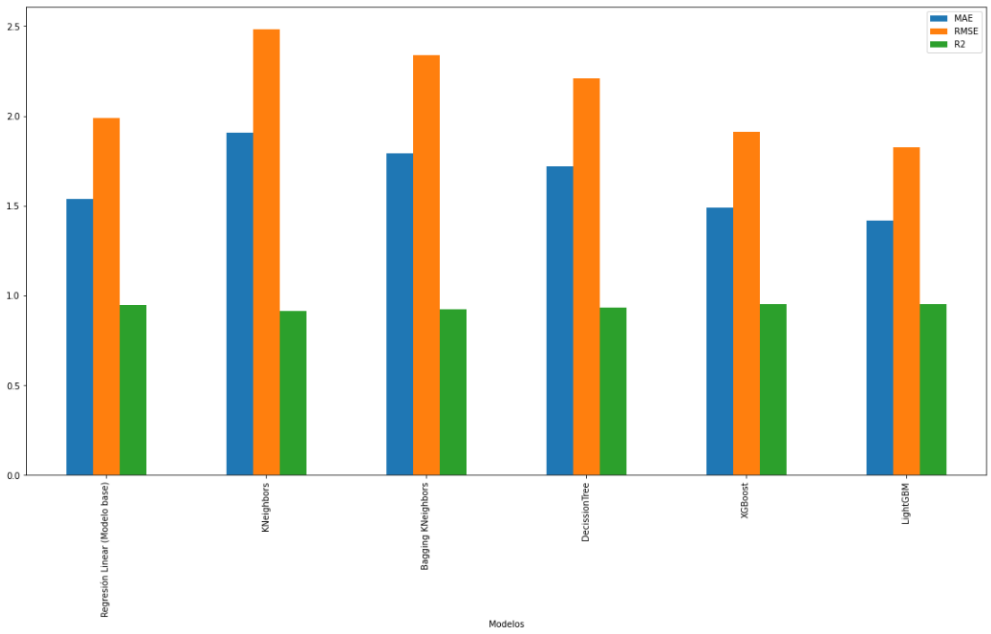
Muestro los resultados en tabla

```
total = pd.concat([pd_ln, pd_kn,pd_bkn,pd_DT,pd_XGB,pd_LGBM]).set_index(['Modelo'])
total
```

Modelo	MAE	RMSE	R2
Regresión Linear (Modelo base)	1.538424	1.986712	0.945830
KNeighbors	1.905687	2.483712	0.915337
Bagging KNeighbors	1.789581	2.340120	0.924843
DecissionTree	1.720511	2.208189	0.933079
XGBoost	1.489172	1.909120	0.949978
LightGBM	1.419295	1.826951	0.954191

Muestro los resultados en gráfica en una gráfica simple

```
total.plot.bar(xlabel="Modelos",rot=90,figsize=(20, 10))
```



Se observa que los modelos con mejor resultado son el XGBoost y LGBM

4.1 Generación del mejor modelo

El modelo con mejores resultados ha sido LightGBM.

Para tener disponible el modelo final creo un pipeline pasándole los mejores valores que he obtenido anteriormente, y se entrena con todos los datos

```
import joblib
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer

# Definimos variables X
features_list=['mes',
               'dia',
               'presMax',
               'presMin',
               'sol',
               'tmin',
               'prec',
               'velmedia',
               'racha']
# Definimos la variable a predecir y
target_list=['tmax']

# Métricas que parasemos al scoring
scoring_metrics=['neg_root_mean_squared_error','r2','neg_mean_absolute_error']

# Transformamos con standar scaler
ct=ColumnTransformer(
    [
     ('sc',StandardScaler(),['presMax','presMin','sol','tmin','prec','velmedia','racha'])
    ]
)

# crear canalización con los pasos de ingeniería de características y el mejor modelo
pipe_best_total=Pipeline(steps=[
                                ('ct',ct),
                                ('LGBM',LGBMRegressor(random_state=7))
                               ])

# Se establecen los parámetros para el modelo
params_LGBM={
    'LGBM__max_depth':[5],
    'LGBM__n_estimators':[112],
    'LGBM__learning_rate':[.09],
    'LGBM__subsample':[.3],
    'LGBM__colsample_bytree':[.8]
    }

best_LGBM=GridSearchCV(pipe_best_total,
                       param_grid=params_LGBM,
                       refit='neg_root_mean_squared_error',
                       scoring=scoring_metrics)

# entrenamos con todos los datos
best_LGBM.fit(df_datos_clima[columnas_scalar],df_datos_clima['tmax'])

# Guardamos el modelo para usarlo desde una app
model_path='models/'
joblib.dump(best_LGBM, model_path+'modelo_mejor.sav')
```

El modelo se guarda en la carpeta models con el nombre modelo_mejor.sav

5 Prueba de concepto

Para realizar una prueba creo una aplicación con streamlit con una interface simple que me permite introducir valores de prueba y realizar las predicciones cargando el modelo guardado.

En este caso no se ha creado ningún entorno Docker y se ejecuta en mi equipo local.

Instalo la librería con:

```
PS C:\TFM\streamlit> pip install streamlit
```

Una vez terminada la instalación puedo editar las rutas del fichero app.py para que coincidan con las rutas donde está guardado el modelo.

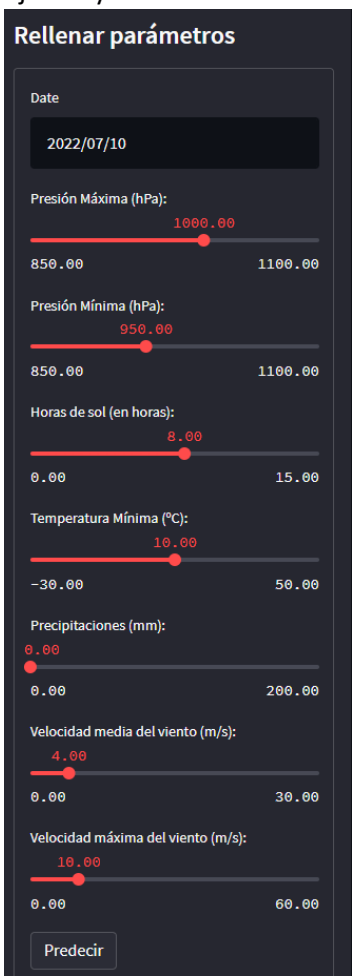
Para ejecutar la app uso:

```
PS C:\TFM\streamlit> streamlit run c:/TFM/streamlit/app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.124:8501
```

Abro en el navegador la url indicada para ver la app.

El código de la misma se puede encontrar en el apartado código.

Se muestra un menú lateral con los valores necesarios para la predicción los cuales se pueden ajustar y de esta forma se realizará la predicción de la temperatura máxima:



Una vez se pulsa el botón predecir se nos muestra el resultado solicitado:

6 Conclusiones

El principal objetivo de este trabajo se ha podido desarrollar sin utilizar la visión de series temporales, si bien son algo más precisas, sus aplicaciones futuras son menores para este caso.

En un principio utilizaba Colab de Google, pero dado que las máquinas virtuales se ejecutan en el cloud de Google y se desconectaban si dejaba mucho tiempo el pc sin atención, terminé implementando el Jupyter con Docker, lo que resulta muy sencillo, disponiendo de distintas distribuciones con las librerías ya instaladas, incluso he realizado pruebas con algunas librerías de pycaret con la que también se obtienen muy buenos resultados y sobre todo, de forma rápida y automática.

Me ha sorprendido gratamente, la facilidad para la obtención de datos climáticos por parte de la AEMET, que dispone de una información muy completa y con poca falta de datos.

Por otro lado, me ha resultado interesante la cantidad de librerías desarrolladas para la obtención de los datos de OPENDATA, simplificando la descarga de los datos.

A la hora de realizar gráficas he revisado distintas librerías, pero al final me he decantado por lo simple, con gráficos de calidad normal pero que muestran los resultados que se buscan como matplotlib o folium.

Bajo mi punto de vista, ha quedado muy clara la necesidad de aplicaciones del ML dentro de la predicción meteorológica y la facilidad de implementar un sistema de regresión lineal que se comporta bastante bien.

Con un buen estudio de los parámetros y un escalado correcto de los mismos, una simple regresión puede realizar unas predicciones muy válidas.

Me parece muy interesante también, además de muy simple, el uso de herramientas como streamlit para la generación de interfaces simples, con muy pocas líneas de código se puede realizar una prueba de concepto muy atractiva.

6.1 Mejoras, aplicaciones y futuras líneas de investigación

Al comienzo de este trabajo, realicé una serie temporal ya que es lo más obvio y el resultado para la obtención de la temperatura máxima fue muy, pero me surgió la idea de combinar los valores de temperaturas junto con valores de otro tipo.

Por ejemplo, ocupación hotelera o facturación de hostelería, lo que permitiría a los dueños de establecimientos hacer unas previsiones de ventas y reservas de sus restaurantes de una forma óptima.

Por ello realicé la eliminación de la serie y preparé un conjunto de datos más relevante que contenía además de los datos climáticos otros campos como, reservas, compras, ventas, día de la semana, ocupación hotelera en la zona, comensales, días festivos, etc.

Estos datos son fáciles de obtener al estar presentes en cualquier aplicación de TPV, se pueden realizar modelos más complejos.

Pero, los resultados que obtuve no fueron nada buenos dada la dificultad que ha presentado el covid en estas fechas pasadas, teniendo limitaciones de aforo, horarios de cierre e incluso fechas en las que no se podían abrir los establecimientos, todo ello me ha hecho pensar en atrasar dicha línea de investigación para un futuro.

De ahí la importancia de los datos.

7 Referencias

Docker y Docker compose - https://docs.docker.com/compose/

Streamlit - https://docs.streamlit.io/

Google Colab - https://colab.research.google.com/?hl=es

Stop One-Hot Encoding your Time-based Features - https://towardsdatascience.com/stop-one-hot-encoding-your-time-based-features-24c699face2f?gi=adb9e5c302ab

AEMET OPENDATA - https://opendata.aemet.es/centrodedescargas/inicio

Pyaemet - https://pypi.org/project/pyaemet/

Folium - https://pypi.org/project/folium/

Pycaret - https://pycaret.org/

8 Código

Organización del repositorio

El proyecto se puede encontrar en mi Github público:

https://github.com/jbermejog/MLFTM

Dentro de este repositorio lo he organizado con la siguiente estructura

Doc

- Predicion_AEMET.pdf - Este documento del TFM (sin datos personales)

Python

- TFM-Notebook – Fichero que contiene el código utilizado para generar y probar los modelos
- App.py – código de aplicación de streamlit

Container

- Docker-compose.yml – fichero para iniciar el contenedor de Jupyter

Modelo

- Mejor_modelo.sav – modelo de predicción generado con mejor resultado