Predicción Meteorológica

Proyecto 3 - Master IABD

Víctor Angulo | Marcos de Castro | Joshua Winn

1. Informe de origen de los datos

a. Opendata AEMET (https://opendata.aemet.es/)

Esta fuente la hemos utilizado para obtener los datos diarios de los últimos años para las ciudades de Madrid y Barcelona.
La página te obliga a pedir un API KEY para poder utilizarla aunque sea desde el propio navegador, para obtener los datos hay que rellenar los campos que se quieran en la página y esta te devuelve un link que te lleva a un archivo en formato JSON.
Desde la página web los datos solo se podían obtener en intervalos de 6 meses, más tiempo y la API devuelve un código de error.
La página te deja elegir entre muchas diferentes estaciones meteorológicas, la mayoría de ellas (si no todas) con su distintivo para reconocerlas, esto luego lo usaremos para crear el sistema de archivos para guardar los datos.

b. Open-Meteo (https://open-meteo.com/)

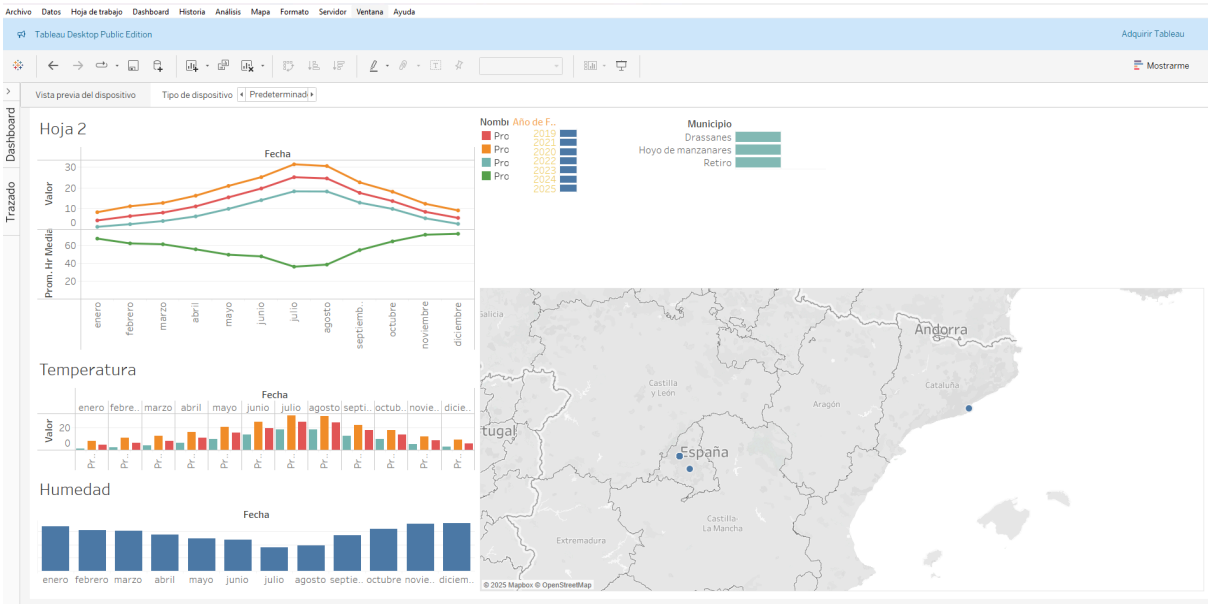
Open-Meteo nos daba tanto la opción de obtener datos por días como por horas, así que lo hemos usado para ambos casos. En el caso de los días hemos podido obtener datos de Hoyo de Manzanares en Madrid y para los horarios datos de Barcelona.
La página es sencilla de usar, primero eliges la latitud y longitud del sitio del que quieres obtener los datos, añades desde que fecha hasta cual quieres que devuelva datos y marcas en las checkboxes los datos específicos que quieres que te devuelva.
La página devolverá o un link con un archivo JSON o una llamada a la API para que puedas lanzar tu mismo. Los datos de esta página contienen pocos valores nulos lo cual ha hecho bastante agradable el trabajo con estos.
Al contrario de los datos de la AEMET, Open-Meteo no devuelve el nombre de la estación de la que obtiene los datos así que la hemos tenido que buscar por otras páginas que la contienen.

c. Datos.Madrid (https://datos.madrid.es/)

Datos Madrid es la página web oficial del ayuntamiento de la comunidad de Madrid para obtener los datos de meteorología de varias estaciones repartidas a lo largo de la comunidad autónoma.
Para extraer los datos simplemente hay que clickar en el link que devuelve los datos desde 2019 hasta la fecha actual.
Estos datos no tienen muchas variables comparados con los de AEMET y Open-Meteo pero aún así nos han sido útiles al poder obtener datos por horas.

2. Informe del cuadro de mandos y visualización de los datos (Explicación de lo que está expuesto en PowerBI o Tableau y como visualizarlo)

Para la creación del cuadro de mandos hemos decidido usar Tableau.
Dentro del cuadro de mandos tenemos expuesto parte de los datos como si fuese el EDA, mostrando graficos de barras, el mapa con las localizaciones y las graficas con las series temporales.



En esta imagen se puede observar que hemos añadido 4 fuentes de datos, 3 datos de temperatura, la máxima de color naranja, la media de color rojo/rosa y la minima de color azul claro.
Tambien hay graficos para los datos de humedad relativa que en este caso son de color verde.

Arriba a la izquierda de la imagen tenemos los datos puestos como serie temporal de un año, a primera vista ya se puede observar la curva que hacen las temperaturas dependiendo de la estación del año que nos encontremos, en invierno las temperaturas son menores y en verano llegamos a los picos más altos.
Inversamente tambien se puede ver a primera vista como cambian los datos de humedad durante el año, los meses frios, mas propensos a lluvias serán más humedos que los meses de calor.
A la derecha de estas graficas tenemos una pequeña leyenda para que sea más fácil saber que se está viendo.
Abajo a la izquierda tenemos unas graficas muy similares a las anteriormente mencionadas pero en este caso con barras verticales y con más separación entre los meses para desglosar mejor la visualización de los datos.
En la parte derecha de la pantalla hay un mapa de España con varios puntos marcados en los diferentes municipios de los cuales tenemos datos que en este caso son, Barcelona, Drassanes; Madrid, Retiro; Madrid, Hoyo de Manzanares.

3. Informe de análisis de datos

Para esta parte de la documentación hemos decidido separar los datos por diarios y horarios ya que siguen diferentes métricas y de esta forma se trabajaba de forma más cómoda y eficiente.

a. Análisis de los datos diarios (verDataDiario.ipynb)

Como se ha explicado brevemente en el punto 1 de la documentación, los 3 municipios que hemos usado para los datos diarios son: Madrid, Retiro (AEMET), Barcelona, Drassanes (AEMET) y Hoyo de Manzanares (Open-Meteo).
Las 4 variables que hemos elegido para predecir son: Temperatura Máxima, Temperatura Media, Temperatura Mínima y la Humedad Relativa.
El análisis de los datos se hará sobre los datos raw para así ver que hemos descargado y las posibles transformaciones que tendremos que hacer en un futuro a los datos.

Los pasos que hemos seguido en el EDA de datos diarios han sido los siguientes:
Lo primero es pasar los datos de csv a dataframe de pandas para poder trabajar con ellos, y posteriormente se hace un **.describe()** de cada

dataframe para ver en una tabla un análisis rápido de los datos numéricos que tenemos.
Aquí nos damos cuenta del primer problema, tantos en los datos de Madrid como los de Barcelona solo aparecen 4 variables como numéricas: altitud, hrMedia (humedad relativa), hrMax y hrMin. Las variables de temperatura que tenemos que usar no aparecen.
Para ver qué tipos de datos son nuestras variables lanzamos un **.dtypes** sobre el dataframe. Con esto vemos que solo las 4 variables mencionadas arriba están con el tipo de datos float o int y todas las demás son de tipo object.
Al hacer un **.head(3)** también observamos que los datos de temperatura están con números decimales pero están separados por ‘,’«coma» en vez de ‘.’«punto» lo cual hace que no sean del tipo de dato numérico.
Para arreglar esto usamos el código que hay en la imagen siguiente que básicamente cambia las comas por puntos y al mismo tiempo cambiamos el tipo de dato a float.

```
df_mad["tmed"] = df_mad["tmed"].str.replace(',', '.').astype(float)
df_mad["tmin"] = df_mad["tmin"].str.replace(',', '.').astype(float)
df_mad["tmax"] = df_mad["tmax"].str.replace(',', '.').astype(float)

df_bar["tmed"] = df_bar["tmed"].str.replace(',', '.').astype(float)
df_bar["tmin"] = df_bar["tmin"].str.replace(',', '.').astype(float)
df_bar["tmax"] = df_bar["tmax"].str.replace(',', '.').astype(float)
```

5] ✓ 0.0s

```
df_bar["fecha"] = pd.to_datetime(df_bar["fecha"])
df_mad["fecha"] = pd.to_datetime(df_mad["fecha"])
```

6] ✓ 0.0s

Para el tipo de dato de fecha solo tenemos que pasar la columna por un **.to_datetime** para cambiar el tipo de dato al deseado ya que el formato de fecha era el que buscábamos desde el comienzo (yyyy-MM-dd). Esto último también se hará sobre los datos de hoyo de manzanares que estaban también con fechas object.

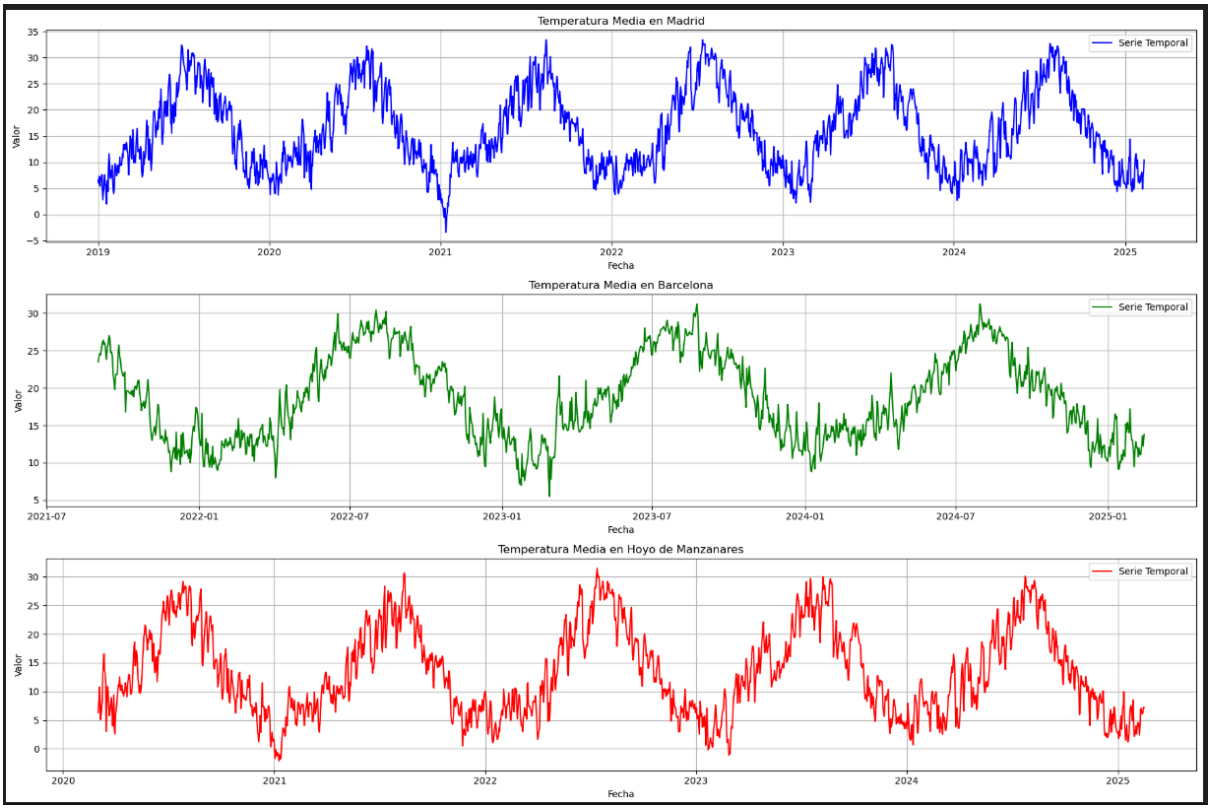
Lo siguiente que vamos a comprobar es el número de datos nulos que hay en cada variable que vamos a utilizar, para ello simplemente usamos el **.isna().sum** sobre las columnas y lo sacamos por pantalla con un print.

```
    print(f"Nulos en humedad relativa Barcelona: {df_bar['hrMedia'].isna().sum()}")
11]  ✓ 0.0s

..  Nulos en temperatura media Hoyo de Manzanares: 0
    Nulos en temperatura media Madrid: 20
    Nulos en temperatura media Barcelona: 4
    -----------------------------------
    Nulos en temperatura maxima Hoyo de Manzanares: 0
    Nulos en temperatura maxima Madrid: 20
    Nulos en temperatura maxima Barcelona: 4
    -----------------------------------
    Nulos en temperatura minima Hoyo de Manzanares: 0
    Nulos en temperatura minima Madrid: 20
    Nulos en temperatura minima Barcelona: 4
    -----------------------------------
    Nulos en humedad relativa Hoyo de Manzanares: 0
    Nulos en humedad relativa Madrid: 54
    Nulos en humedad relativa Barcelona: 7
```

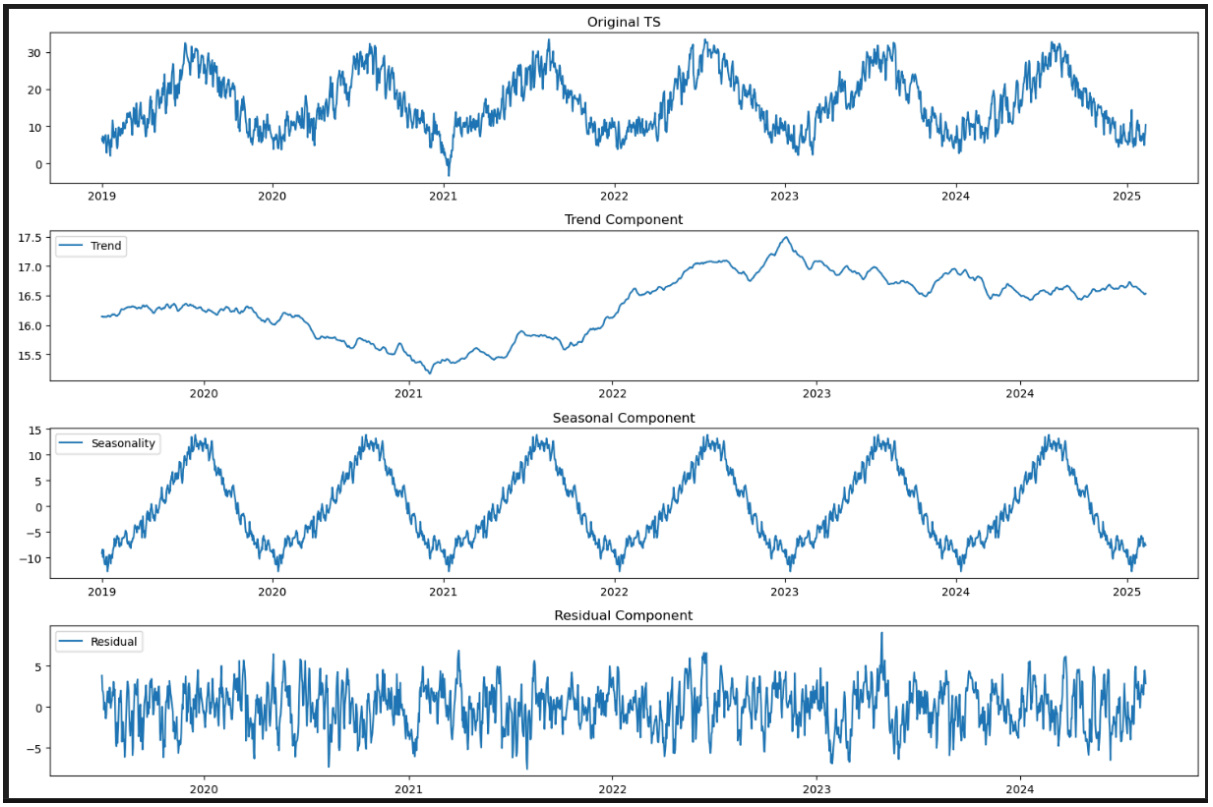
Como se puede apreciar no tenemos muchos datos nulos teniendo en cuenta que la cantidad de datos totales son de media unos 1500.
Para eliminar estos nulos vamos a usar el método **.ffill.** Hemos elegido este método ya que hemos comprobado que los datos nulos están separados y no hay un número elevado de estos juntos.

Una vez tenemos todos los datos disponibles podemos empezar a visualizarlos, lo primero que se hará es ver las series temporales sin modificaciones ni nada extra, solo ver como están nuestros datos.



(A partir de aquí solo hablaré de los datos de humedad relativa y de temperatura media, ya que los datos de temperatura son similares entre sí)

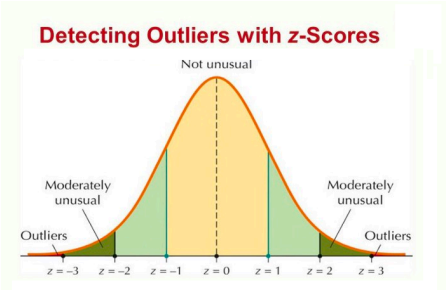
Ahora podemos ver el **seasonal_decompose()** de nuestra serie temporal, este nos desglosa en 3 gráficas diferentes la tendencia, estacionalidad y el ruido que hay en nuestros datos.



A primera vista podríamos deducir que nuestra serie temporal muy probablemente sea estacional, lo cual tendría bastante sentido ya que tenemos datos diarios sobre temperatura que no fluctúan demasiado de un dia a otro.

A continuación podemos ver si en nuestros datos hay algún valor que sea outlier, para esto tenemos que sacar la media móvil (moving average) de nuestros datos y la desviación típica de los mismos. Siguiendo la imagen a continuación decidimos poner el threshold de desviaciones típicas en 3 para

detectar los datos que verdaderamente son outliers.

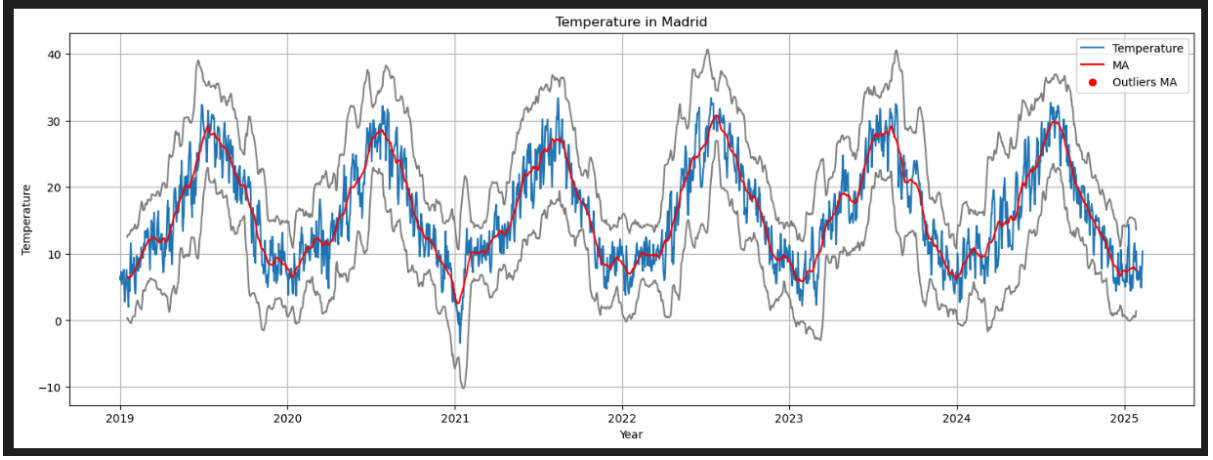


Cuando visualizamos la serie temporal junto con los thresholds de detección de outliers vemos que no hay valores atípicos que modificar, ni siquiera los datos de enero de 2021 en Madrid que corresponden a la borrasca Filomena que sale muy marcada en la serie temporal.

```
plt.figure(figsize=(18, 6))

plt.plot(df_mad['fecha'], df_mad['tmed'], label='Temperature')
plt.plot(df_mad['fecha'], ma, label='MA', color='red')
plt.plot(df_mad['fecha'], threshold_min, color='gray')
plt.plot(df_mad['fecha'], threshold_max, color='gray')
plt.scatter(df_mad["fecha"][df_mad["outlier_ma"]], df_mad["tmed"][df_mad["outlier_ma"]], color='red', label="Outliers MA")
plt.xlabel('Year')
plt.ylabel('Temperature')
plt.title('Temperature in Madrid')
plt.legend()
plt.grid(True)
plt.show()
```

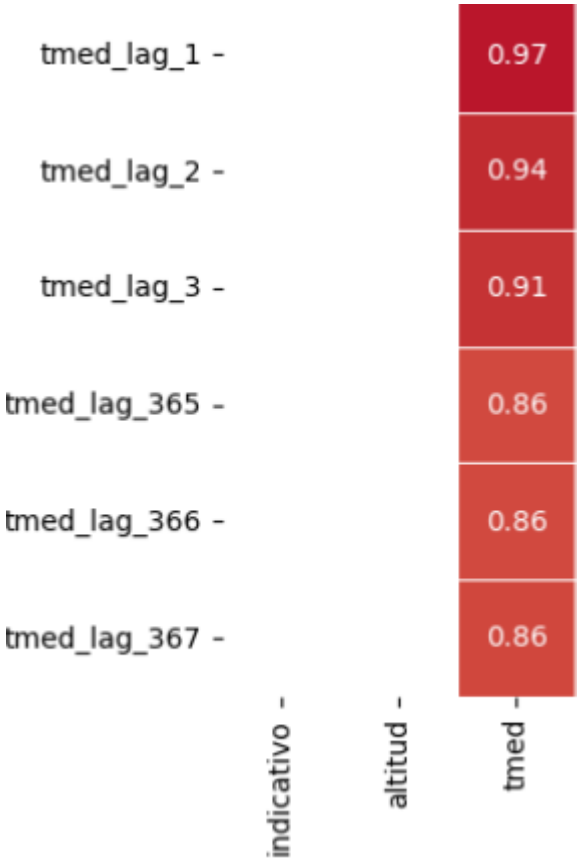
✓ 0.3s



También nos gustaría saber la correlación que tienen los datos entre ellos, en específico con los datos de días anteriores y los datos del mismo día pero del año anterior (lags). Para esto hemos sacado 6 variables de lags y usando el **heatmap de seaborn** podemos apreciar la correlación entre ellos.

En el caso de los valores de temperatura la correlación del dato con los datos de sus 3 días anteriores es superior al 0.95, y con los datos del año anterior superior al 0.85. Estos números son demasiado altos y si los incluyéramos en la predicción el modelo daría la respuesta basando casi exclusivamente en estos datos por lo cual no usaremos los lags para el modelo predictivo.

En el caso de los valores de Humedad Relativa la correlación de los valores de los 3 días anteriores como los del año pasado es casi nula, 0.10 o menos. Justo al contrario que con los valores de temperatura, como la correlación de los lags es tan cercana a 0 que no influirá en el resultado final de la predicción, así que decidimos tampoco usar ese dato.



Por último mediremos la estacionalidad de nuestras series temporales usando el test de Dickey-Fuller, usando la función que se nos dió en clase en uno de los notebooks de series temporales.

```python
def check_stationarity(series, name):
    result = adfuller(series)
    print(f'ADF Statistic for {name}:', result[0])
    print('p-value:', result[1])
    if result[1] <= 0.05:
        print("The series is likely stationary.")
    else:
        print("The series is likely non-stationary.")
```

```python
check_stationarity(df_mad['tmed'], 'tmed')
```

```
ADF Statistic for tmed: -3.0289243788909714
p-value: 0.0322801027729375 25
The series is likely stationary.
```

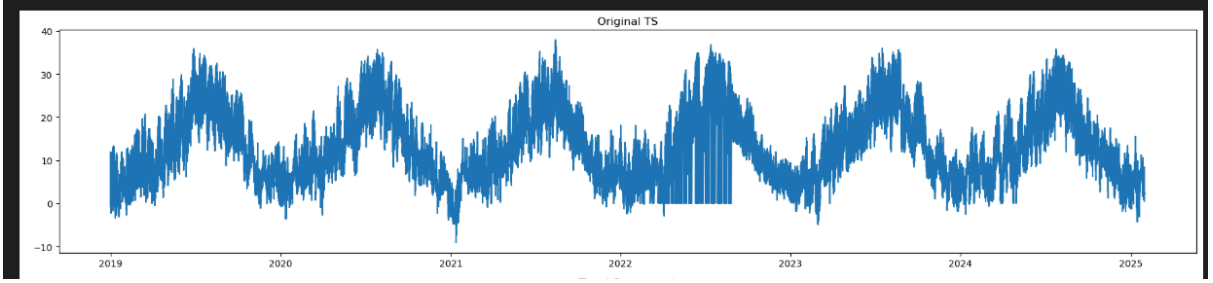
Por lo que devuelven los resultados de pasar las columnas a predecir por la función podemos corroborar que si son estacionarias.

b. Análisis de los datos horarios (Madrid_Data_EDA.ipynb)

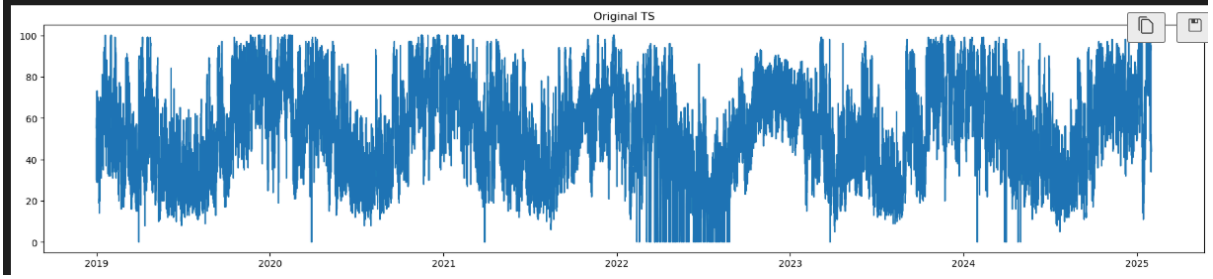
Para hacer el EDA de los datos horarios hemos decidido utilizar los datos homogeneizados y posteriormente los transformados ya que los datos raw tienen bastantes nulos y columnas inservibles.
Para el ejercicio de predicción nos pedían usar 2 variables, en nuestro caso hemos elegido las de Temperatura Media y Humedad Relativa, ya que son dos variables que teníamos en nuestras fuentes de datos.

Lo primero que hemos hecho sería la lectura de datos de csv a un dataframe de Pandas. Luego ordenamos los datos primero por fechas y después por horas.
Sacamos las gráficas de la serie de tiempo para ambas variables y podemos apreciar que tenemos varios huecos vacíos o con líneas que no deberían ocurrir.

```
show_graphics(df=df, col='temperatura_media', fecha='fecha')
```




```
show_graphics(df=df, col='humedad_media', fecha='fecha')
```

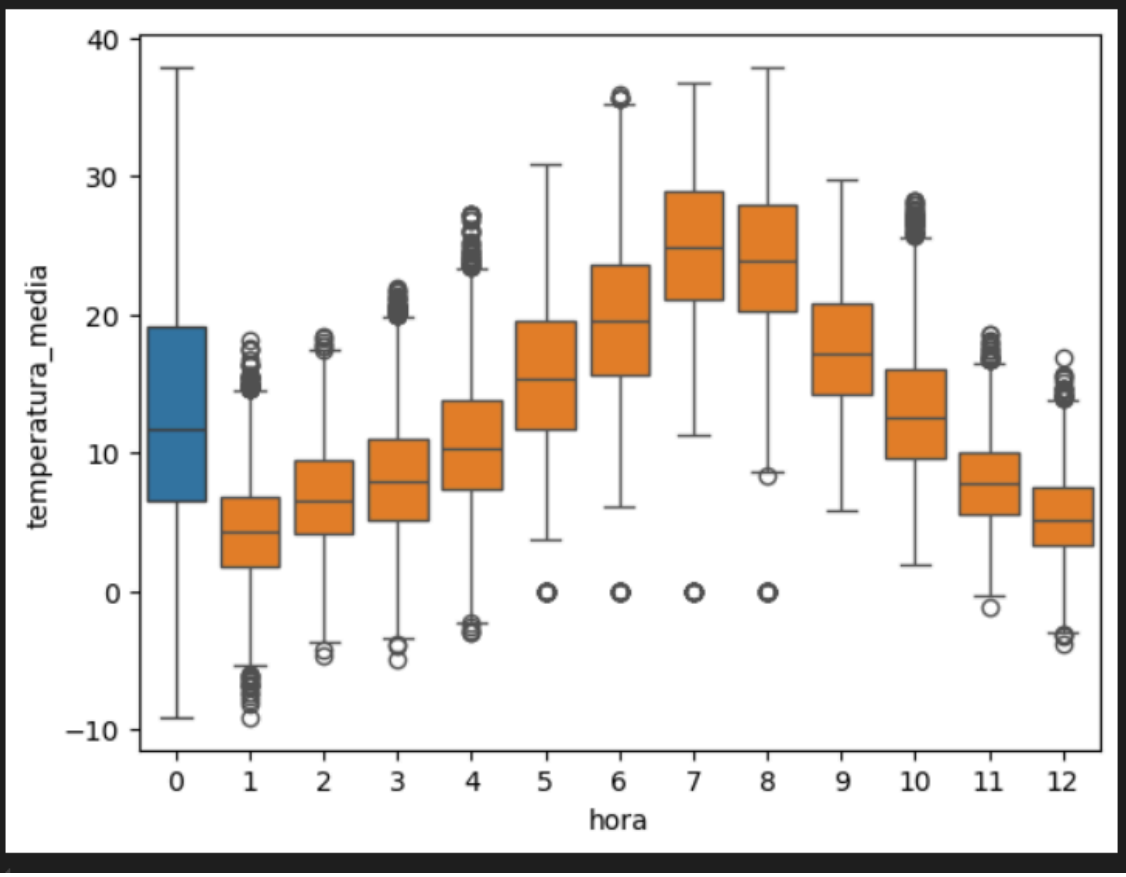


También descubrimos que durante el año 2022 la gráfica mostraba una línea recta en 0.000, indagando un poco en esto recogiendo sólo los datos del año 2022 vimos, haciendo un **value_counts()** que efectivamente teniamos 170 datos con el valor 0.000 de los 6700 datos que ocupan al año 2022. Esto no nos pareció un número significativo de datos ya que en total sólo habría unos 200 datos fallidos de los 53 mil que componen la serie temporal.

Luego sacamos una gráfica con boxplots de la temperatura media por meses que nos da la información de en qué meses la temperatura media suele ser más elevada.

```
# Por mes
sns.boxplot(x='mes', y='temperatura_media', data=df)
```

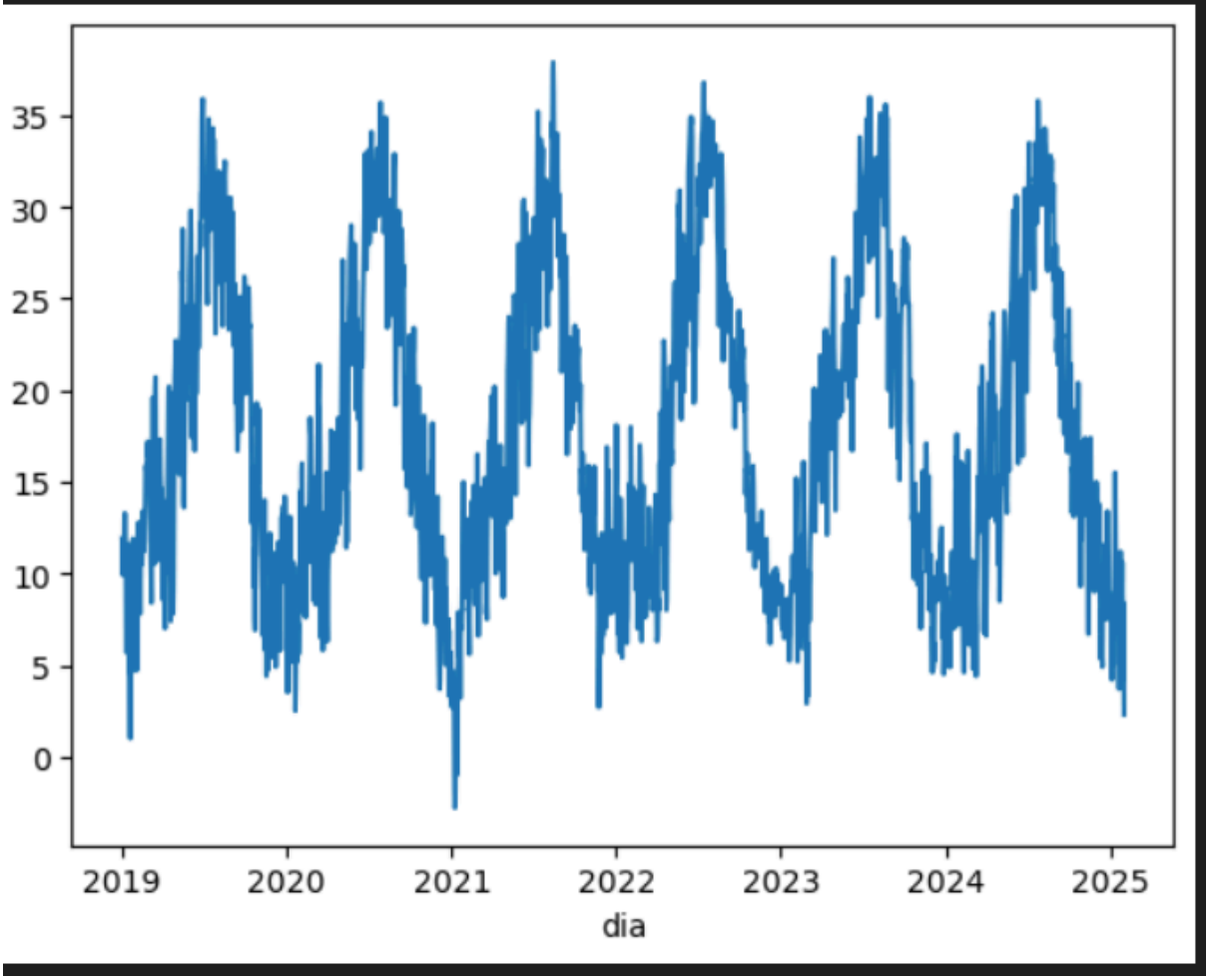
```
<Axes: xlabel='hora', ylabel='temperatura_media'>
```



Por último sacamos una gráfica que nos muestra los valores máximos agrupados por día de nuestra serie temporal.

```
df['dia'] = df['fecha'].dt.date
daily_max_temp = df.groupby('dia')['temperatura_media'].max()
daily_max_temp.plot()
```

```
xes: xlabel='dia'>
```



4. Modelos de ML utilizados (Explicación de los modelos probados para las predicciones y cuales se han escogido para las mismas)

a. ARIMA

El modelo ARIMA (AutoRegressive Integrated Moving Average) es una herramienta estadística ampliamente utilizada para el análisis y la predicción de series temporales. Este modelo es adecuado cuando los datos tienen una

estructura temporal y se desea predecir valores futuros basándose en patrones pasados. ARIMA combina tres componentes fundamentales: autorregresión (AR), diferencia (I) y media móvil (MA), lo que le permite capturar tendencias y ciclos en los datos.

El modelo ARIMA se representa mediante el trío de parámetros (p,d,q), donde:

- p es el orden del modelo autorregresivo (AR).
- d es el número de diferencias necesarias para hacer que la serie sea estacionaria.
- q es el orden del modelo de media móvil (MA).

Antes de poder implementar el modelo ARIMA tienes que hacer un analisis de los datos (EDA) y determinar que tus series temporales sean estacionarias.

Para determinar los parametros (p,d,q) hemos utilizado la funcion de auto_arima de la libreria pmdarima. La cual pasa por todas las variables posibles para dar los valores finales que se usarán en el modelo. Unas variables bien escogidas hacen que el modelo devuelva mejores predicciones.

Los datos obtenidos al pasarlos por un MAE nos dan un resultado peor que el que nos dan los otros 2 modelos a continuacion.

b. SARIMA

Muy parecido a ARIMA, solo que a este se le pueden añadir variables exagenarias que refuerzan al modelo y ayudan a obtener mejores predicciones.
Al igual que ARIMA para obtener los datos de (p,d,q) y (P,D,Q) hemos usado la funcion de auto_arima.
Lanzando el modelo con los datos obtenidos nos devolvió un resultado mucho más cercano que el de ARIMA.
El summary del modelo es el siguiente:

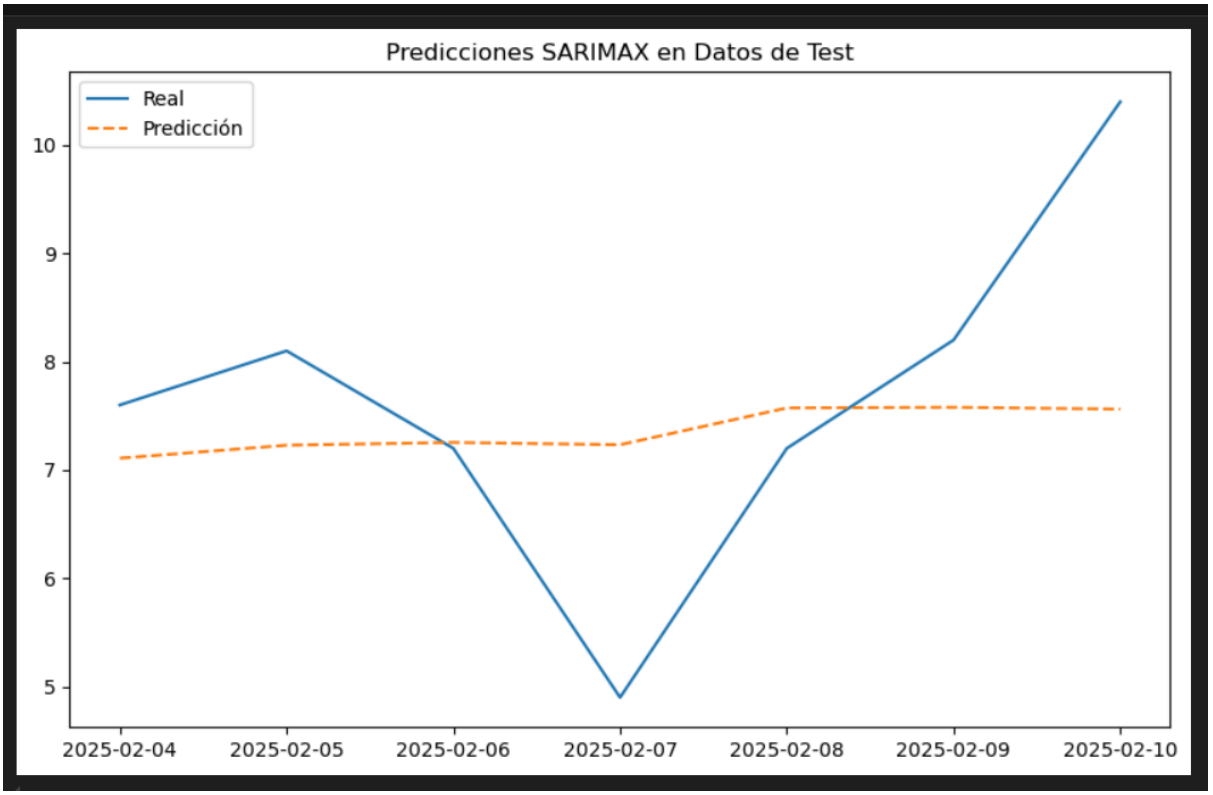
```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                   value   No. Observations:                 2226
Model:               SARIMAX(3, 0, 1)   Log Likelihood               -4491.644
Date:                Sun, 09 Mar 2025   AIC                          9001.288
Time:                        14:41:38   BIC                          9052.659
Sample:                             0   HQIC                         9020.048
                                - 2226
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sin_dia        0.0303      0.082      0.371      0.711      -0.130       0.190
cos_dia       -0.0210      0.071     -0.295      0.768      -0.161       0.119
sin_mes       -0.6552      0.599     -1.093      0.274      -1.830       0.520
cos_mes       -1.6989      0.562     -3.025      0.002      -2.800      -0.598
ar.L1          1.8475      0.029     64.634      0.000       1.792       1.904
ar.L2         -1.0473      0.039    -26.885      0.000      -1.124      -0.971
ar.L3          0.1994      0.022      9.261      0.000       0.157       0.242
ma.L1         -0.8546      0.023    -37.524      0.000      -0.899      -0.810
sigma2         3.3108      0.095     34.739      0.000       3.124       3.498
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):               103.98
Prob(Q):                              0.92   Prob(JB):                         0.00
Heteroskedasticity (H):               0.86   Skew:                            -0.44
...
===================================================================================
```

Creemos que los valores de AIC y BIC no son muy elevados comparados con otros modelos que hemos ido probando y el valor de Ljung-Box es basicamente 0 lo cual es muy buena noticia.
El mean_absolute_error de este modelo fue muy bueno tambien con solo 1.08 de error absoluto.

```python
mae_test = mean_absolute_error(test['value'], forecast)
print(f"MAE en datos de test: {mae_test}")
```

```
MAE en datos de test: 1.0831029252496926
```

Y la grafica con la prediccion obtenida no es la mejor pero al ser una linea que cruza por el medio de los datos acaba siendo bastante efectiva con las predicciones.



c. RandomForestRegression

Al ver que tuvimos problemas al intentar usar los modelos de ARIMA y SARIMA con parte de los datos y ya que tenemos los datos horarios en formato de seno y coseno decidimos darle una oportunidad al randomForestRegressor.
Para poder usarlo tuvimos que dividir los datos en train y test para poder entrenar al modelo.
Para obtener los mejores parametros del modelo lanzamos un grid_search más o menos básico pero que ayudó bastante a la eficiencia del modelo.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, 30],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=42),
                           param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

GridSearchCV

best_estimator_: RandomForestRegressor

RandomForestRegressor

```
print(f"Mejores parámetros: {grid_search.best_params_}")
```

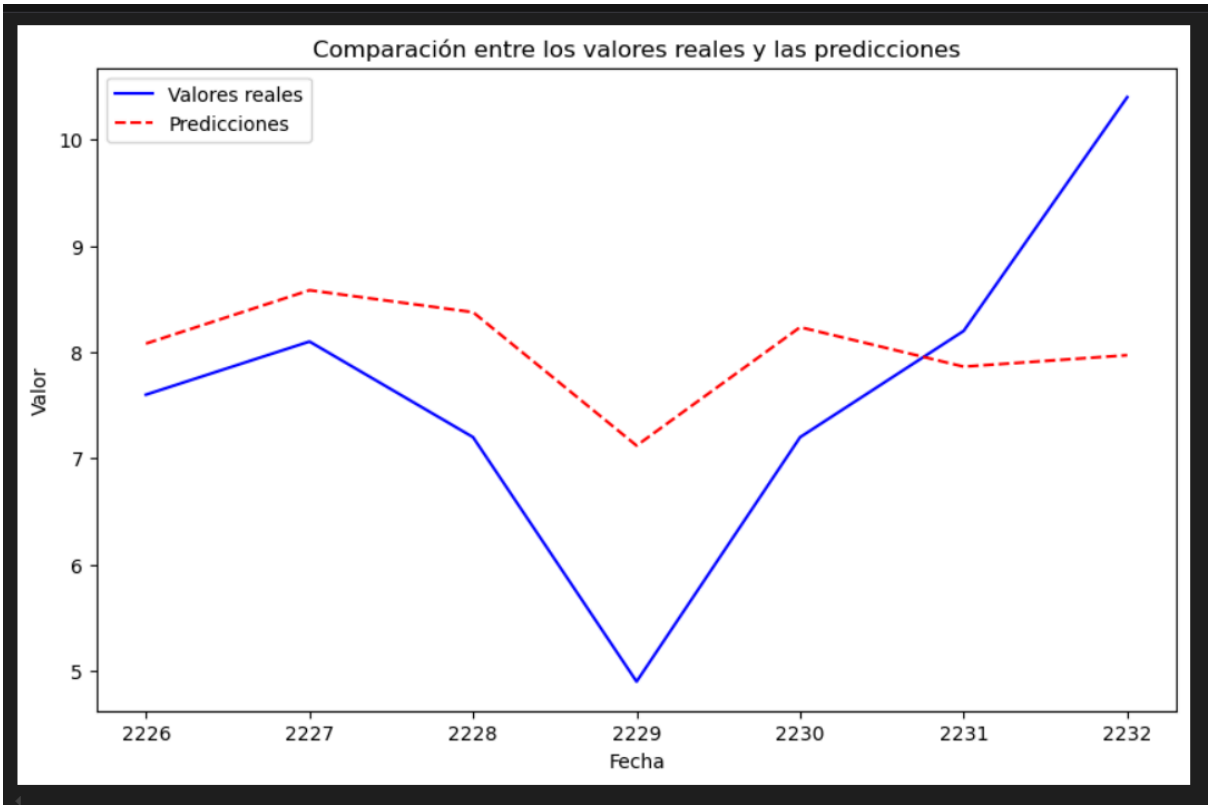
```
Mejores parámetros: {'max_depth': 20, 'min_samples_leaf': 4, 'n_estimators': 100}
```

El mean_absolute_error nos dio en el caso de estos datos bastante peor que usando SARIMAX, pero en otros conjuntos de datos si nos ofreció mucho mejor rendimiento.

```
mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')
```

```
Mean Absolute Error: 2.1411651617564673
```

La sorpresa llegó al ver la grafica de las predicciones, con mucha diferencia este modelo nos daba la mejor gráfica en cuanto a la forma que seguian los datos predichos comparados con los reales.



Estos han sido los 3 modelos que hemos usado a fondo, nos planteamos tambien usar otros aprendidos en clase como prophet o SKforecast, pero al final decidimos quedarnos con estos porque los conociamos bien y ofrecian buenos resultados.

5. Descripción ciclo de los datos

a. Como se han sacado los datos

Para este proyecto los datos se han sacado sin el uso del Web Scrapping ya que las fuentes que hemos escogido no han requerido de esta práctica.
Los datos sacados de la página de **Datos.Madrid** se obtuvieron simplemente descargando el archivo .csv que hay en la página.
Los datos de **AEMET** se obtuvieron desde la página web, sin la necesidad de montar un archivo con código para usar el API de AEMET. La página web solo permitia sacar los datos en intervalos de 6 meses con lo cual acabamos con unos 12 archivos .csv que tuvimos que unir en un archivo completo con los datos.
Por otro lado los datos de **Open-Meteo** si se descargaron usando la API que te daba la página web, en nuestro caso descargamos todas las posibles variables que nos ofrecía la API. Esto resultó en un sin fin de archivos .csv

que venian separados por años y meses, al igual que con los de AEMET los acabamos uniendo todos en un archivo .csv gigante.

b. Como se han homogeneizado y transformado

Para la homogeneización y la transformación de los datos hemos usado Spark con la libreria de PySpark. Spark maneja grandes cantidades de datos de una manera mucho más eficiente que Pandas lo cual se notaba bastante con los datos horarios ya que eran más de 50 mil datos en conjunto.

Para la homogeneización lo primero fue (gracias a lo obtenido en el EDA) cambiar algunas variables a float y las fechas a datetime para poder trabajar con ellas.
Los siguiente fue el cambio de nombre de las variables a un estandar para que todos los otros archivos fuesen iguales. Los nombres de las variables ahora iban a ser: temperatura_maxima, temperatura_media, temperatura_minima y humedad_media. Esto último se hizo con el metodo de dataframes de pySpark **.withColumnRenamed.**
Ahora que todas las columnas se llaman se llaman igual en las 3 fuentes de datos diarios, es mucho más facil trabajar con ellas. Para los datos de AEMET sacamos de la fecha dos nuevas columnas llamadas anio y mes que nos servirán más tarde para guardar los datos.
Con esas columnas nuevas ya tenemos todas las columnas que necesitamos para trabajar en el proyecto por ahora.
Lo siguiente es separar el cada dataframe con los datos en 4, uno por cada variable a predecir.

```
df_temp_media_bar = df_bar.select(F.col("fecha"), F.col("indicativo"), F.col("anio"), F.col("mes"), F.col("temperatura_media"))
df_humedad_bar = df_bar.select(F.col("fecha"), F.col("indicativo"), F.col("anio"), F.col("mes"), F.col("humedad_media"))
df_temp_maxima_bar = df_bar.select(F.col("fecha"), F.col("indicativo"), F.col("anio"), F.col("mes"), F.col("temperatura_maxima")
df_temp_minima_bar = df_bar.select(F.col("fecha"), F.col("indicativo"), F.col("anio"), F.col("mes"), F.col("temperatura_minima")
```

Por último convertimos la columna que contiene el valor a predecir de cada dataframe en 2 columnas diferentes, una que contiene un indicativo del tipo de dato (temperatura_media, humedad_media, etc) y otra con el valor del dato. Tambien eliminamos la columna inicial de las que se han sacado estas 2.

```
df_temp_media_bar = df_temp_media_bar.withColumn("tipo_dato", F.lit("temperatura_media")) \
                                     .withColumn("value", df_temp_media_bar["temperatura_media"]) \
                                     .drop("temperatura_media")
df_humedad_bar = df_humedad_bar.withColumn("tipo_dato", F.lit("humedad_media")) \
                                     .withColumn("value", df_humedad_bar["humedad_media"]) \
                                     .drop("humedad_media")
df_temp_maxima_bar = df_temp_maxima_bar.withColumn("tipo_dato", F.lit("temperatura_maxima")) \
                                     .withColumn("value", df_temp_maxima_bar["temperatura_maxima"]) \
                                     .drop("temperatura_maxima")
df_temp_minima_bar = df_temp_minima_bar.withColumn("tipo_dato", F.lit("temperatura_minima")) \
                                     .withColumn("value", df_temp_minima_bar["temperatura_minima"]) \
                                     .drop("temperatura_minima")
```

Ahora que tenemos todo preparado podemos volver a juntar todo en un dataframe gigante usando .unionByName. Esto es fácil de hacer ya que todos los dataframes contienen las mismas columnas y el tipo de dato.

Para la transformación de los datos nos hemos guiado en base a prueba y error de lanzar modelos, y gracias a la matriz de correlación que obtuvimos en el EDA.
Al final hemos decidido que lo único que necesitabamos era sacar el dato de dias a una columna aparte de la fecha y convertir los datos de dias y meses usando una función de seno y coseno que harán que los datos sean mucho más legibles para el modelo.

Por último solo queda guardar los datos.

c. Como se han guardado

Para guardar los datos hemos usado parquet ya que ofrece la capacidad de particionar los datos y de manera automatica crear carpetas con los nombres distintivos de las columnas.
La linea de código para hacer esto es la siguiente:

```
df_completo.write.format("parquet").partitionBy("indicativo","tipo_dato","anio","mes").mode("overwrite").save("../../../data/hom
print("finished")
```

Python

```
finished
```

.format te deja elegir en que formato se van a guardar los datos, en nuestro caso parquet.
.partitionBy te deja elegir las diferentes particiones que quieres que se hagan en el sistema de archivos. En nuestro caso queremos que se primero se haga una distinción por el indicativo, luego por el tipo de dato, y por ultimo año y mes.

.mode te deja elegir que modo de guardado quieres que ocurra, en nuestro caso overwrite hace que si ya existen datos en la ruta los sobreescribe con los nuevos.
.save sirve para poner la ruta en la que se guardarán los archivos al ejecutar la linea de código.