

# Trabajo de fin de Máster: ETL y ejemplo de explotación de datos geográficos y climatológicos de España.

Andrea Delgado Galisteo



U N I V E R S I D A D  
COMPLUTENSE  
M A D R I D

## Índice

1.- Planteamiento del problema. ....	1
2.- Presentación de fuentes de datos y fase de investigación. ....	2
2.1.- Datos geográficos.....	2
Municipios, listado. ....	2
Códigos postales.....	3
Geometrías. ....	4
2.2.- Datos climatológicos. ....	4
3.- Arquitectura técnica.....	4
3.1.- ETL Python.....	5
Extracción. ....	5
Transformación. ....	8
Carga.....	11
3.2.- SQL ....	11
3.3.- Ejemplo de explotación, Tableau. ....	14
4.- Caso de negocio ....	17
5.- Futuras evoluciones y mejoras ....	18

## 1.- Planteamiento del problema.

Una de las primeras preguntas que surgen cuando nos planteamos este tipo de procesos y trabajos es *el porqué de la temática escogida*. Si nos encontrásemos desarrollando un proceso ETL en un entorno profesional, la respuesta sería fácil ya que no tendríamos más datos de los que ocuparnos que los que la empresa nos ofreciera como materia de estudio. En el marco académico, sin embargo, siento que a habido dos factores fundamentales que han hecho que este trabajo termine tratando datos geográficos y climatológicos.

El primero ha sido el criterio del desarrollador, en este caso, yo misma; ya que el no encontrarnos con temáticas acotadas quise encontrar algo que me motivara lo suficiente, pero que a la vez pudiera resolver algún problema real, algo que pudiera resultar útil.

El segundo factor ha sido la disponibilidad de datos abiertos y de calidad de ellos en internet. Es decir, ha habido que adaptarse a las circunstancias. A lo largo de este máster uno de los muchos conocimientos adquiridos ha sido el bucear en webs especializadas en este mundo del big data para obtener datos que explotar. Si bien encontrar datasets sobre los que aplicar diferentes modelos de predicción y estudiar sus resultados es relativamente sencillo, la tarea se complica cuando más de uno es necesario y además se les exige que puedan hablarse entre ellos de alguna forma.

Conociendo lo anteriormente expuesto me sentí cómoda escogiendo en primer momento los datos climatológicos centrados en el territorio español que ofrecen webs como la de la Agencia Estatal de Meteorología (AEMET de ahora en adelante).

Esto es así ya que los datos relativos al clima son sin duda unos de los más visitados incluso antes de que se recogieran en internet. El público general está muy familiarizado con ellos ya que, a día de hoy, por ejemplo, los asistentes digitales que nos acompañan es de las primeras cosas que nos informan en el día, de la predicción climatológica.

En el marco de la actualidad global, analizar las observaciones climatológicas para obtener tendencias y comprender hacia dónde nos dirigimos es fundamental. El calentamiento global es una preocupación compartida por todos los países ya que el nuevo escenario puede provocar que nuestros hábitos de consumo de fuentes de energía, agua potable, así como el uso de los medios y rutas de transporte que hemos establecido, se vean fuertemente afectados. Sin ir más lejos el análisis en la península es especialmente interesante dado que poseemos gran superficie de costa que puede verse afectada por una subida del nivel del mar.

Una vez me decidí por esta temática, y pensando en qué otro tipo de datos sería necesario y a la vez posible unificar, enseguida surgió la necesidad de homogeneizar el marco geográfico sobre el que tratar dichos datos climatológicos. De la propia experiencia que cualquiera podemos tener como usuario, sabemos lo frustrante que puede ser consultar tanto predicciones como pasadas observaciones en diferentes webs sobre un municipio, por ejemplo, y ver que los datos no coinciden. *Este es uno de los problemas fundamentales que propone resolver esta herramienta* (o versiones posteriores más desarrolladas de ella) ya que consigue unificar datos climatológicos de distintas fuentes de origen y ponerlas en común sobre un marco geográfico unificado.

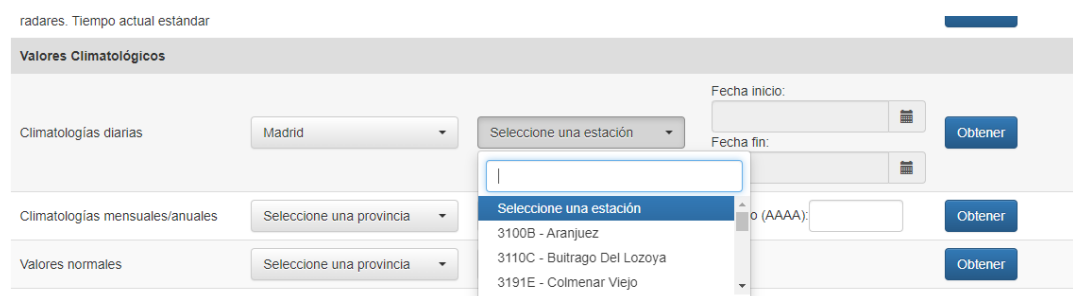
Una vez acotada la temática del trabajo, me dediqué pues a una fase de investigación en la que estudié todos los posibles orígenes de datos y su posibilidad de ser integrados.

## 2.- Presentación de fuentes de datos y fase de investigación.

El objetivo principal de este trabajo es entonces unificar diferentes tipos de datos y de diferentes fuentes en una sola base de datos que pueda ser empleada, por ejemplo, como fuente origen para trabajos de BI así como para conformar un dataset sobre el que poder aplicar modelos de predicción.

Antes de entrar en los detalles de la solución técnica es pues necesario comprender todas las fuentes de datos escogidas y el porqué de cada una de ellas.

La idea original de este trabajo era comenzar con los datos climatológicos y aunque posteriormente volveré a hablar de este tipo de datos, para que se comprendan las fuentes de datos escogidas a un nivel narrativo, es necesario mostrar lo primero con lo que me topé al explorar los primeros datasets ofrecidos por la AEMET Open Data.<sup>1</sup>



Esta web posee una interfaz muy amigable para el usuario con la que empecé a jugar. Lo primero que llama la atención es que para obtener los datasets de los valores diarios medidos en el pasado, los parámetros que tenemos que escoger son la provincia y una estación meteorológica, así como un rango de fechas. La primeras preguntas que surgen al ver esto son entonces ¿dónde está cada una de esas estaciones? ¿cuál es el listado de estaciones completo?

De cara a poder contestarlas, vi entonces clara la necesidad de tener de alguna forma un marco, sobre el que situarlas, es decir, un mapa.

### 2.1.- Datos geográficos.

#### Municipios, listado.

Lo primero a considerar es la división territorial mínima que tiene sentido considerar aquí. Debido a que la red de estaciones meteorológicas de la AEMET es amplia y pensando por ejemplo en soluciones de explotación del dato que promediaran y observarían tendencias en agrupaciones mayores de otros territorios, tenía mucho sentido considerar los municipios como la unidad básica de división territorial del mapa.

La propia web de la AEMET ofrece un dataset con el listado de municipios españoles acompañados de algunos datos geográficos (coordenadas del centro de cada municipio, elevación) y demográficos.<sup>2</sup> Unos de los datos que más en falta se echan en este dataset son las provincias y comunidades autónomas en las que se encuentran cada uno de ellos.

Para resolver esto, otra de las webs a las que acudí mucho en este trabajo es la web del Instituto Nacional de Estadística (INE). En ella, pude descargar otro dataset completo con todos los

<sup>1</sup> <https://opendata.aemet.es/centrodedescargas/productosAEMET?>

<sup>2</sup> Llamada vía API con key necesaria - <https://opendata.aemet.es/opendata/api/maestro/municipios/>

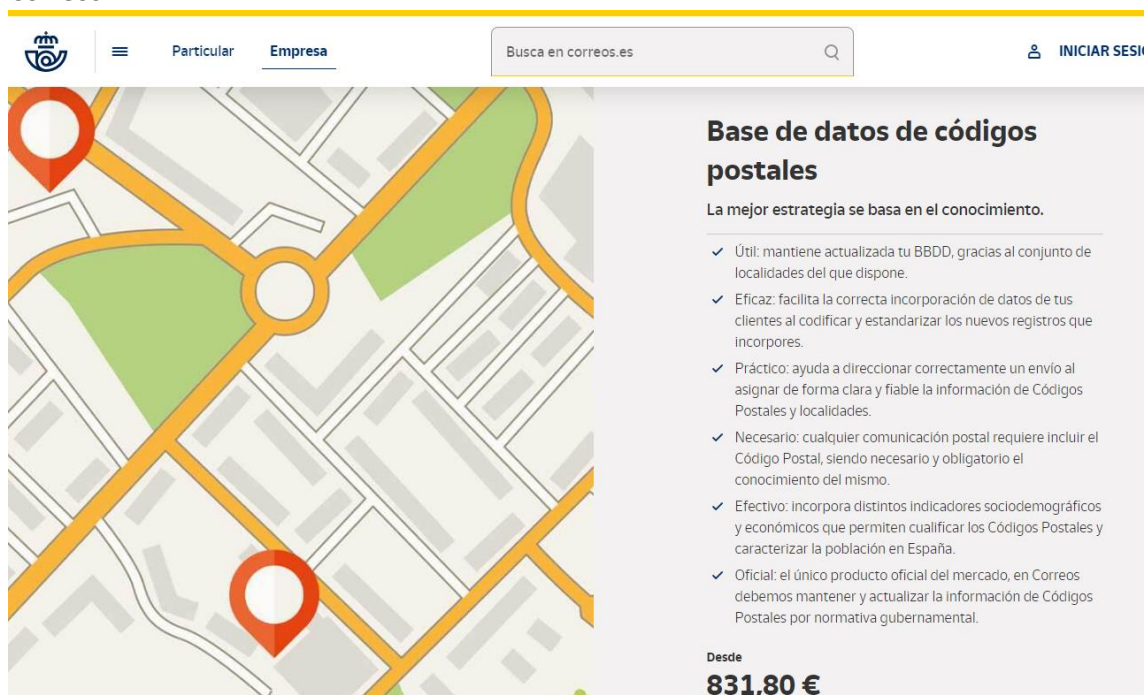
municipios de España acompañado de otro tipo de datos (códigos de identificación)<sup>3</sup> así como otra tabla con los nombres e identificadores de provincias y comunidades autónomas<sup>4</sup>.

Con las 3 colecciones de datos entre manos, tuve que buscar un denominador común que hiciera posible que pudieran unirse. Este no fue otro que lo que he llamado INE\_MUN\_ID. Se trata de un identificador de tipo texto y largo 5 único por cada municipio que pude encontrar en las 3 tablas. Dentro de cada set se obtenía de forma diferente (recortando dentro de strings más largos o concatenando varios campos). Gracias a este identificador, se termina con una tabla muy completa con información de los municipios, la llamada MUNICIPIOS en el código y posterior base de datos.

### Códigos postales.

Otro tipo de dato que normalmente manejamos cuando queremos consultar información relativa al clima es una dirección completa y, por tanto, de su código postal o código Zip. En España, un municipio puede poseer más de un código postal y a la vez, un mismo código postal puede ser empleado por más de un municipio. Ello hace que puedan considerarse nueva categoría de división territorial.

Cuando traté de descargar un dataset que contuviera los identificadores de cada municipio y sus códigos postales, me encontré con que esta información es privada y de pago y tan solo la ofrece Correos.<sup>5</sup>



Base de datos de códigos postales

La mejor estrategia se basa en el conocimiento.

- ✓ Útil: mantiene actualizada tu BBDD, gracias al conjunto de localidades del que dispone.
- ✓ Eficaz: facilita la correcta incorporación de datos de tus clientes al codificar y estandarizar los nuevos registros que incorporas.
- ✓ Práctico: ayuda a direccionar correctamente un envío al asignar de forma clara y fiable la información de Códigos Postales y localidades.
- ✓ Necesario: cualquier comunicación postal requiere incluir el Código Postal, siendo necesario y obligatorio el conocimiento del mismo.
- ✓ Efectivo: incorpora distintos indicadores sociodemográficos y económicos que permiten cualificar los Códigos Postales y caracterizar la población en España.
- ✓ Oficial: el único producto oficial del mercado, en Correos debemos mantener y actualizar la información de Códigos Postales por normativa gubernamental.

Desde  
**831,80 €**

Gracias a que era un tema muy compartido y tratado en foros y páginas internet, pude encontrar una solución alternativa para obtener esta información.<sup>6</sup> Para ello, volví a necesitar un dataset de la web del INE. Más concretamente el callejero electoral. Estos datasets suponen muchas

<sup>3</sup> Descarga directa de fichero <https://www.ine.es/daco/daco42/codmun/codmun20/20codmun.xlsx>

<sup>4</sup> [https://www.ine.es/daco/daco42/codmun/cod\\_ccaa\\_provincia.htm](https://www.ine.es/daco/daco42/codmun/cod_ccaa_provincia.htm)

<sup>5</sup> <https://www.correos.es/es/es/empresas/marketing/identifica-a-tus-clientes-potenciales/base-de-datos-de-codigos-postales>

<sup>6</sup> **Francisco Goerlich (Universidad de Valencia e Ivie) (2022)** - *Elaboración de un mapa de Códigos Postales de España con recursos libres* – “<https://www.uv.es/goerlich/Ivie/CodPost.html>”. Sitio Web

tablas de ancho fijo y poco interpretables ya que pertenecen a datasets de datos *ploteables*, geométricos (en concreto en su web lo llaman Cartografía digitalizada de secciones censales).<sup>7</sup>

En este fichero, se encuentra una tabla con, se supone, todas las direcciones postales recogidas en el censo español. En esta tabla, dentro de los muchos campos disponibles, recortando strings más largos podía obtenerse finalmente una tabla muy sencilla de dos columnas con el identificador de municipio y el código postal.

### Geometrías.

Después de todo esto aun así, solo tenía algunas tablas con información de municipios, provincias y comunidades autónomas pero ningún tipo de información que me permitiera situar una estación en un municipio. Con esto, busqué para terminar el marco geográfico unificado los recintos poligonales de cada uno de los municipios y provincias. Terminé visitando entonces el centro de descargas del Instituto geográfico Nacional (IGN). En su centro de descargas, pude encontrar conjuntos de datos de tipo *shape* justo con esta información. Gracias a estos, posteriormente en el código se pudo situar las diferentes estaciones en municipios del territorio.

### 2.2.- Datos climatológicos.

Tal y como se ha justificado al principio de esta sección, para que el trabajo y explotación de los datos importados pudieran ordenarse y tener sentido, necesitaríamos localizar las estaciones climatológicas en el mapa. Una vez conseguido el “mapa”, se descargaron entonces los inventarios de estaciones de las webs de las que se van a importar datos climatológicos. Estas webs son dos, en concreto AEMET<sup>8</sup> de la que ya he hablado y National Oceanic and Atmospheric Administration (NOAA), una web gubernamental oficial de EEUU.<sup>9</sup>

En ambos casos el listado completo de estaciones se descarga vía API y en los dos catálogos se presentan las coordenadas donde están cada una de las estaciones.

Por último y también vía API para los dos organismos, se descargan los conjuntos de datos de valores normales diarios entre las fechas 01/01/2010 y 31/12/2021 ambos inclusive. Las observaciones de ambos dataframes vienen referidas a una estación meteorológica en concreto. Los datos descargados de la web de la AEMET<sup>10</sup> eran mucho más completos e incluyen datos relativos a la temperatura, precipitación, presión y viento.

## 3.- Arquitectura técnica.

La idea entonces, después de conocer los datos que van a alimentar esta ETL, es llevar a cabo integraciones mediante diversas transformaciones necesarias; para así exportar los datasets unificados a una base de datos relacional SQL que pudiera ser explotada por cualquier empresa.

Para ello, las primeras tareas de descarga de los datos en bruto (extracción); las transformaciones necesarias sobre ellos, tales como, uniones entre los dataframes, operaciones sobre los strings, unificación de formatos entre datasets, agrupaciones y uso de otras funciones más elaboradas (es decir, la fase de transformación) y su extracción en tablas que pudieran alimentar una base de datos (carga); se ha realizado enteramente empleando el lenguaje Python, en un notebook de Google Colab.

---

<sup>7</sup> [http://www.ine.es/prodyser/callejero/caj\\_esp/caj\\_esp\\_072022.zip](http://www.ine.es/prodyser/callejero/caj_esp/caj_esp_072022.zip)

<sup>8</sup> <https://opendata.aemet.es/centrodedescargas/productosAEMET?>

<sup>9</sup> Interfaz para el usuario para descarga datos NOAA - <https://www.ncei.noaa.gov/cdo-web/search>

En segundo lugar, se creó una base de datos en lenguaje mySQL de acuerdo a las tablas extraídas en el punto anterior, donde los datos de salida del código Python sirvieron para alimentar las tablas SQL.

Por último y como un ejemplo de explotación de dicha base de datos, he creado un dashboard en Tableau que permite una interpretación mucho más visual de los datos. Esta interpretación visual de los datos, al tratarse de valores climatológicos recogidos a lo largo de los años, cobra gran importancia pues como sabemos, es el ejemplo perfecto de series temporales que presentan estacionalidad.

Los tres módulos de la herramienta son importantes, presentan una función clara y diferenciada y cobran sentido cuando se ejecutan secuencialmente.

Es posible visitar el código de las 3 partes de este pipeline en el repositorio GitHub:

<https://github.com/andreadgalis/TFM>

Se describen algunos aspectos más técnicos de cada uno de ellos a continuación.

### 3.1.- ETL Python

De cara a poder seguir la evolución de esta primera parte del código, y pensando en una futura ejecución de este en un entorno profesional, toda la ejecución del código Python se puede seguir gracias al fichero de logging que se descarga al final del programa. En él se escriben tanto mensajes de información general para el seguimiento de la tarea, es decir, mensajes a nivel DEBUG, así como mensajes de tipo ERROR en caso de que alguna tarea no haya podido realizarse. También se escriben mensajes de tipo INFO que van imprimiendo al usuario ciertos datos clave para poder comprobar que el funcionamiento es correcto. Se extrae en formato .txt y antes de cada mensaje se informa de la fecha y horas de ejecución.

#### Extracción.

Como ya se ha contado en puntos anteriores de esta documentación, una de las funciones principales de esta parte del pipeline fueron las operaciones de extracción de las diferentes fuentes de datos para lo que fueron necesarios diferentes enfoques.

Estas diferentes técnicas de extracción venían condicionadas a la forma en la que la fuente original nos presenta los datos. Gracias a esto, he tenido la oportunidad de explorar diferentes tipos de llamadas a páginas web, así como métodos de extracción de ficheros en código Python.

Uno de los que más he repetido es la llamada vía API, donde el siguiente podría ser un ejemplo:

```
#noaa stationsin
noaa_headers = {"token": "njEAHYVYIkaPyuJQKEZYAuXRVOWCPodq"}
noaa_url_stations = 'https://www.ncei.noaa.gov/cdo-web/api/v2/stations?locationid=FIPS:SP&limit=1000'

try:
    noa_stations = requests.request("GET", noaa_url_stations, headers=noaa_headers, timeout = (5, 10))
    noa_stations = noa_stations.json()
    estaciones_NOAA = pd.json_normalize(noa_stations['results'])
    estaciones_NOAA = estaciones_NOAA[['id', 'latitude', 'longitude', 'name', 'elevation']]
    estaciones_NOAA['id'] = estaciones_NOAA['id'].str.replace('GHCND:', '')
    estaciones_NOAA
```

Las llamadas vía API han sido todo un descubrimiento ya que, dependiendo de la web y configuración de cada una, se permiten hacer consultas muy variadas gracias a que consideran una serie de parámetros configurables. En la mostrada arriba en el ejemplo, se consulta el listado completo de estaciones meteorológicas en España de las que NOAA tiene lecturas. Para estos

casos la librería *requests* ha jugado un papel fundamental. Por ejemplo, el parámetro empleado en la URL anterior “locationid=FIPS:SP” especifica que le pedimos el listado de todas las estaciones en el país SP. Además, en muchas de las consultas era necesario además una key que permite a los dueños de los dominios web controlar el flujo de peticiones que reciben.

Cada URL de las empleadas en cada llamada ha sido escrita ad hoc para cada uno de los datasets que he necesitado descargar. La información para poder comprender la sintaxis de cada una de ellas se recoge en manuales de uso de las webs.<sup>11, 12, 13</sup> Así, en el caso de las ETLs que descargaban las observaciones climatológicas, tuve que crear bucles que crearan urls diferentes en cada iteración. En el caso de la API de la AEMET, tan solo se podían consultar datos de intervalo máximo 31 días, así que el bucle itera por fechas. Por el contrario, en el caso de la API de NOAA, podíamos considerar el rango de fecha que creyéramos conveniente, pero los datos se descargaban estación a estación, ya que el territorio a considerar no era uno de los parámetros de esta API. El bucle en este segundo caso itera por las estaciones del inventario de NOAA.

Otro de los métodos de extracción de dataframes ha consistido en leer directamente de páginas webs que mostraban la información a modo tabla, lo cual lo permite el propio paquete pandas:

```
4 #URL, esta vez directa a la página web del INE
ine_ccaa_prov_url = 'https://www.ine.es/daco/daco42/codmun/cod_ccaa_provincia.htm'

with open('ETL_logging.txt', 'a') as f:
    ahora = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    message = ahora+ ' - DEBUG - Comienza lectura tabla web INE códigos autonómicos y provinciales.'
    f.write(message)
    f.write('\n')
    f.close()

try:
    table_MN = pd.read_html(ine_ccaa_prov_url, skiprows=[51], converters={'CODAUTO':str,'CPRO':str})
```

Otras de las fuentes empleadas se han conseguido gracias a que el hiperenlace de algunas páginas webs redirecciona directamente al fichero que quería conseguir, como por ejemplo un fichero Excel o un comprimido .zip. Así, con tener la dirección web, la extracción era inmediata:

```
#URL, esta vez es directa a un xlsx
ine_municipios_url = 'https://www.ine.es/daco/daco42/codmun/codmun20/20codmun.xlsx'

with open('ETL_logging.txt', 'a') as f:
    ahora = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    message = ahora+ ' - DEBUG - Comenzando descarga dataframe municipios INE doc.'
    f.write(message)
    f.write('\n')
    f.close()

try:
    ine_municipios_file = pd.ExcelFile(ine_municipios_url)

[10] try:
    #Descargamos la carpeta comprimida zip contenedora de todos los ficheros de tablas de ancho fijo
    wget -nc 'http://www.ine.es/prodyser/calajejo/caj_esp/caj_esp_072022.zip' #nc para que no vuelva a descargarla
    zip = zipfile.ZipFile('caj_esp_072022.zip')

    fichero_direcciones = io.StringIO(zip.read('TRAM.P01-52.D220630.G220704')).decode('ISO-8859-1'))
```

<sup>11</sup> <https://www.ncei.noaa.gov/support/access-search-service-api-user-documentation>

<sup>12</sup> <https://www.ncdc.noaa.gov/cdo-web/webservices/v2>

<sup>13</sup> <https://opendata.aemet.es/dist/index.html?>



El último método empleado ha sido uno de los que más tiempo me ha costado conseguir. Se trata de una llamada, empleando el paquete requests, de tipo POST. El código en sí en esta ocasión no era lo difícil de conseguir, ya que su configuración es completamente homóloga a la del método GET. Lo difícil ha sido conseguir la URL y los parámetros a introducir ya que esta vez no había habilitada al usuario una herramienta API. Todo lo he conseguido estudiando el código fuente de la página web. Me detengo por esto algo más en este método.

Descargar la información del IGN<sup>14</sup> para el usuario es algo muy sencillo ya que la página web se entiende con facilidad, tan solo cuando intentamos descargar algo nos salta un aviso legal.



Para poder reproducir la descarga manual del dataset con los polígonos de cada municipio y provincia, he tenido que estudiar la función php que lanzaba la web al pulsar en el botón de Continuar. Al estudiar la función se puede interpretar que llama a una dirección URL vía POST y que le introduce un parámetro denominado secuencial que identifica el dataset que vamos a descargar.

```
$(document).ready(function(){

    $('#bAceptLic').bind('click',function(event){
        fbq('trackCustom', 'descargaCdD', {
            secuencial: $('#secuencialDescDir').val()
        });

        $('#modalGral,#modalCata,#modalLicContent').slideUp();
        $('#textLicForm').prop('action','descargaDir');
        //control para que los html de actas y cuadernos de lineas limite salgan en ventana nueva
        if($('#showNewWindow').val()=="S"){
            $('#textLicForm').prop('target','_blank');
        }
        else{
            $('#textLicForm').prop('target','_self');
        }
        $('#textLicForm').submit();
    });

    $('#bCancelLic').bind('click',function(event){
        $('#txtModalConf').html(cancelarTramitacion($('#lang').val()));
        modalLic.dialog('open');
    });
});
```

<sup>14</sup> <https://centrodedescargas.cnig.es/CentroDescargas/index.jsp>

```
<form id="textLicForm" name="textLicForm" action="/CentroDescargas/loadLicDescDir.do" method="POST">
  <input id="showNewWindow" type="hidden" value="N">
  <input id="secuencialDescDir" type="hidden" name="secuencialDescDir" value="9000029">
```

Buscando el valor de este parámetro secuencial dentro del código de la web y comprendiendo que la dirección URL a la que llama es la genérica terminada en “/descargaDir”, pude configurar la petición POST. Esto genera que se descargue un documento de tipo .zip de igual manera que si hubiéramos pinchado en la página la opción de Continuar.

Además de este, en la ETL se extrae información de otro fichero .zip a los que puede accederse gracias a la librería zipfile.

```
[16] cnig_url = 'https://centrodedescargas.cnig.es/CentroDescargas/descargaDir'

      secuencial = {'secuencialDescDir': '9000029'}

      with open('ETL_logging.txt', 'a') as f:
          ahora = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
          message = ahora+ ' - DEBUG - Comienza descarga web IGN de los polígonos recintos de
          f.write(message)
          f.write('\n')
          f.close()

      try:
          ign_response = requests.post(cnig_url, data = secuencial, timeout = (10, 30))
```

El tipo de información descargada en este documento, es algo especial. Se trata de ficheros .shp y otros acompañantes. Son formatos geométricos especiales así que para poder interpretarlos correctamente ha sido necesario emplear la librería *geopandas*. Esta librería resulta ser algo conflictiva y es que es común que genere errores cuando se intenta instalar en Windows.

No solo por esta razón, pero por otras más de similares motivos (problemas de compatibilidad entre versiones, sistemas operativos diferentes, etc.), se ha decidido programar este código *Python* en un cuaderno de *Google Colab*. Además, una de las formas más cómodas y comunes que existen hoy en día para compartir este tipo de herramientas es a través de un repositorio de ficheros, como los famosos repositorios *GitHub*, que permiten funcionar como un controlador de versiones. Por todos estos motivos, ejecutar y disponer del código en una herramienta online unificada como lo es *Google Colab*, ha resultado ser muy buena solución.

### Transformación.

Una vez descargada y almacenada en dataframes toda la información necesaria, las transformaciones empleadas según el dataframe y el objetivo han sido muy variadas.

La motivación de llevar a cabo todas estas transformaciones no es otra que la de poder crear uniones entre las tablas descargadas, en su totalidad o sobre una selección de campos sobre ellas. El listado completo y los detalles de qué información contienen y cómo se relacionan entre sí, será explicado en el siguiente apartado de Extracción.

Algunas de estas transformaciones han sido muy básicas y se han repetido en la mayoría de tablas descargadas. Por ejemplo, operaciones en campos de cadena de texto (reemplazando caracteres, recortando, concatenando...), renombrar columnas, operaciones de selección de campos... Muchas de las fuentes de datos descargadas poseían más campos que los necesarios

para la integración y antes de unir varias fuentes origen resulta conveniente seleccionar las columnas pertinentes para el objetivo del trabajo.

Un ejemplo sería el trabajo llevado a cabo sobre el dataset del callejero electoral descargado del INE. Para obtener tanto el código ZIP como el identificador del INE para cada municipio era necesario recortar dos campos de texto de todo el dataframe que además originalmente presentaba muchas más columnas.

```
[11] subset = df[['ine_codigo_string', 'zip_string']]
subset['ine_codigo_string'] = subset['ine_codigo_string'].str[:5]
subset['zip_string'] = subset['zip_string'].str[:5]
ZIPs = subset.drop_duplicates()
ZIPs.rename(columns = {'zip_string': 'ZIP'}, inplace = True)
ZIPs.rename(columns = {'ine_codigo_string': 'INE_MUN_ID'}, inplace = True)
```

Otra de las transformaciones que merece la pena destacar es una operación de agrupación que realicé sobre el mismo dataframe. Como se dijo, un municipio puede emplear más de un código postal por lo que consideré podría ser útil agrupar todos los códigos postales de cada municipio en un array:

```
ZIPs_array = ZIPs.groupby('INE_MUN_ID').ZIP.apply(list).reset_index()
ZIPs_array.columns = ['INE_MUN_ID', 'ZIPs_array']
ZIPs_array.head()
```

	INE_MUN_ID	ZIPs_array
0	01001	[01240, 01193]
1	01002	[01470, 01450, 01468]
2	01003	[01169, 01160, 01165]
3	01004	[01474, 01478]
4	01006	[01220, 01211]

Por último a destacar en esta sección, cabe mencionar la forma en la que llevé a cabo la localización en municipios de las diferentes estaciones meteorológicas de ambos inventarios. Como los dos catálogos presentaban las coordenadas en las que se sitúa cada estación, aproveché el hecho de tener los polígonos recintos de cada municipio para asociar a cada estación un identificador de municipio.

Para esto, lo primero que necesité fue tener las coordenadas en formato decimal:

```
[ ] estaciones_AEMET['latitud_dec'] = estaciones_AEMET['latitud'].apply(lambda x: dms2dec(x))
estaciones_AEMET['longitud_dec'] = estaciones_AEMET['longitud'].apply(lambda x: dms2dec(x))
with open('ETL_logging.txt', 'a') as f:
    ahora = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    message = ahora + ' - INFO - Formato coordenadas geográficas entidad estaciones_AEMET corregido.'
    f.write(message)
    f.write('\n')
f.close()
```

Partiendo de aquí, hice uso de una simple función que extrae como verdadero o falso si un determinado punto se encuentra dentro de un polígono. Trasladándolo a este caso que nos ocupa, si una estación se encuentra dentro de un municipio. Gracias a esta función definí otras que permitieron asociar un ID de municipio a todas las estaciones.

Sobre esta función a la que llamé *localizar\_municipio*, desarrollé diferentes versiones, para hacerla más adecuada a cada uno de los catálogos. Por ejemplo, en el catálogo de estaciones de la AEMET se especifica en qué provincia se encuentra cada una por lo que el listado de municipios entre los que hay que buscar, se acota a los de dicha provincia y hace que la búsqueda sea mucho más eficiente. Esto no pudo ser posible con el catálogo de estaciones de NOAA por lo que para cada estación se recorren la totalidad de municipios preguntándonos si el punto se encuentra dentro del polígono.

```
def localizar_municipio (lat_estacion:float, long_estacion:float, cod_prov:str):
    id_mun_0 = '00000'
    coord_est = Point(long_estacion, lat_estacion)
    mun_list = (total_municipios.loc[total_municipios["CPRO"] == cod_prov, 'INE_MUN_ID'])
    for municipio in mun_list:
        poly = (total_municipios.loc[total_municipios["INE_MUN_ID"] == municipio, 'MUN_geometry'])
        poly = np.array(poly)
        poly = poly[0]
        if poly.contains(coord_est):
            id_mun_0 = municipio
        else:
            pass
    return id_mun_0
```

Al terminar de aplicar estas funciones a los dataframes completos (haciendo uso de funciones apply), aún quedaba una minoría de municipios que no habían sido encajados en ningún municipio. Al estudiar visualmente en Google Maps estos casos, pude ver que por ejemplo algunas estaciones caían en bases militares que se consideraban territorio neutro y no se encerraban por ningún polígono municipal. En la otra mitad de casos, las estaciones se encuentran tan pegadas a la costa (en faros normalmente) que al buscar las coordenadas en Google Maps, caían a pocos metros de la costa, en el agua; es decir, que no eran muy precisas.

Para poner solución a estos casos apliqué, solo sobre estos, un nuevo método de elección de municipio. Volviendo algo atrás, uno de los datos que contenía el dataframe de municipios de la AEMET eran las coordenadas geográficas del centro de cada pueblo. Haciendo uso de estos puntos, definí la función *localizar\_municipio\_3*, que a partir de unas coordenadas geométricas devuelve como salida el pueblo cuyo centro se halla a menor distancia.

```
def localizar_municipio_3 (lat_estacion:float, long_estacion:float, cod_prov:str):
    id_mun_0 = '00000'
    distancia_0 = 1000000
    coord_est = (lat_estacion, long_estacion)
    mun_list = (MUNICIPIOS.loc[MUNICIPIOS["CPRO"] == cod_prov, 'INE_MUN_ID'])
    for municipio in mun_list:
        lat_mun = float((MUNICIPIOS.loc[MUNICIPIOS["INE_MUN_ID"] == str(municipio), 'latitud_dec']).to_list()[0])
        long_mun = float((MUNICIPIOS.loc[MUNICIPIOS["INE_MUN_ID"] == str(municipio), 'longitud_dec']).to_list()[0])
        coord_mun = (lat_mun, long_mun)
        d = (geopy.distance.geodesic(coord_mun, coord_est).km)
        if d < distancia_0:
            distancia_0 = d
            id_mun_0 = municipio
        else:
            distancia_0 = distancia_0
            id_mun_0 = id_mun_0
    return id_mun_0
```

Puede que haya localizado alguna estación en un municipio erróneo (estar más cerca del centro de un pueblo no significa estar contenido en él) pero teniendo en cuenta el material del que dispongo, ha sido decidido así, como un criterio unificador. Me parecía más valioso conservar las medidas recogidas a lo largo de los años por estas estaciones que desecharlos.

#### Carga.

Tras considerar que los dataframes habían sido descargados y unificados de una forma correcta y contenían información suficiente, llegó el momento de la exportación.

Si nos hubiéramos encontrado montando esta herramienta en un entorno empresarial, donde tuviéramos una base de datos estable que un equipo de IT nos hubiera montado y mantuviera, sería muy fácil crear la base de datos directamente incluso desde el código Python.

Como esto no es así, he optado por configurar la base enteramente en lenguaje SQL; incluyendo la parte de alimentar las tablas. Para que esta carga pudiera realizarse de una forma cómoda, el último paso de la ETL en código Python consiste en una extracción en formato .csv de todos los dataframes que finalmente conformarán la base.

```
[ ] comunidades_autonomas.to_csv('comunidades_autonomas.csv', index=False, sep=';')
    provincias.to_csv('provincias.csv', index=False, sep=';')
    MUNICIPIOS.to_csv('MUNICIPIOS.csv', index=False, sep=';')
    ZIPs.to_csv('ZIPs.csv', index=False, sep=';')
    PRO_GEOMETRY.to_csv('PRO_GEOMETRY.csv', index=False, sep=';')
    MUN_GEOMETRY.to_csv('MUN_GEOMETRY.csv', index=False, sep=';')
    total_municipios.to_csv('total_municipios.csv', index=False, sep=';')
    estaciones_AEMET.to_csv('estaciones_AEMET.csv', index=False, sep=';')
    estaciones_NOAA.to_csv('estaciones_NOAA.csv', index=False, sep=';')
    AEMET_diarios.to_csv('AEMET_observaciones.csv', index=False, sep=';')
    NOAA_diarios.to_csv('NOAA_observaciones.csv', index=False, sep=';')
```

```
[ ] files.download('comunidades_autonomas.csv')
    files.download('provincias.csv')
    files.download('MUNICIPIOS.csv')
    files.download('ZIPs.csv')
    files.download('PRO_GEOMETRY.csv')
    files.download('MUN_GEOMETRY.csv')
    files.download('total_municipios.csv')
    files.download('estaciones_AEMET.csv')
    files.download('estaciones_NOAA.csv')
    files.download('AEMET_observaciones.csv')
    files.download('NOAA_observaciones.csv')
    files.download('ETL_logging.txt')
```

### 3.2.- SQL

La base de datos obtenida como resultado de los ficheros anteriores, es una base de datos sencilla pero funcional, teniendo en cuenta la gran variedad de tipos de datos que se almacenan en ella. Para que todo lo desarrollado anteriormente cobrara algo más de sentido, haré una breve descripción de cada una de las tablas que la conforman:

- comunidades\_autonomas (alimentada con comunidades\_autonomas.csv). Se trata de una tabla muy sencilla donde tan solo se almacena el código INE de cada comunidad autónoma y su nombre.

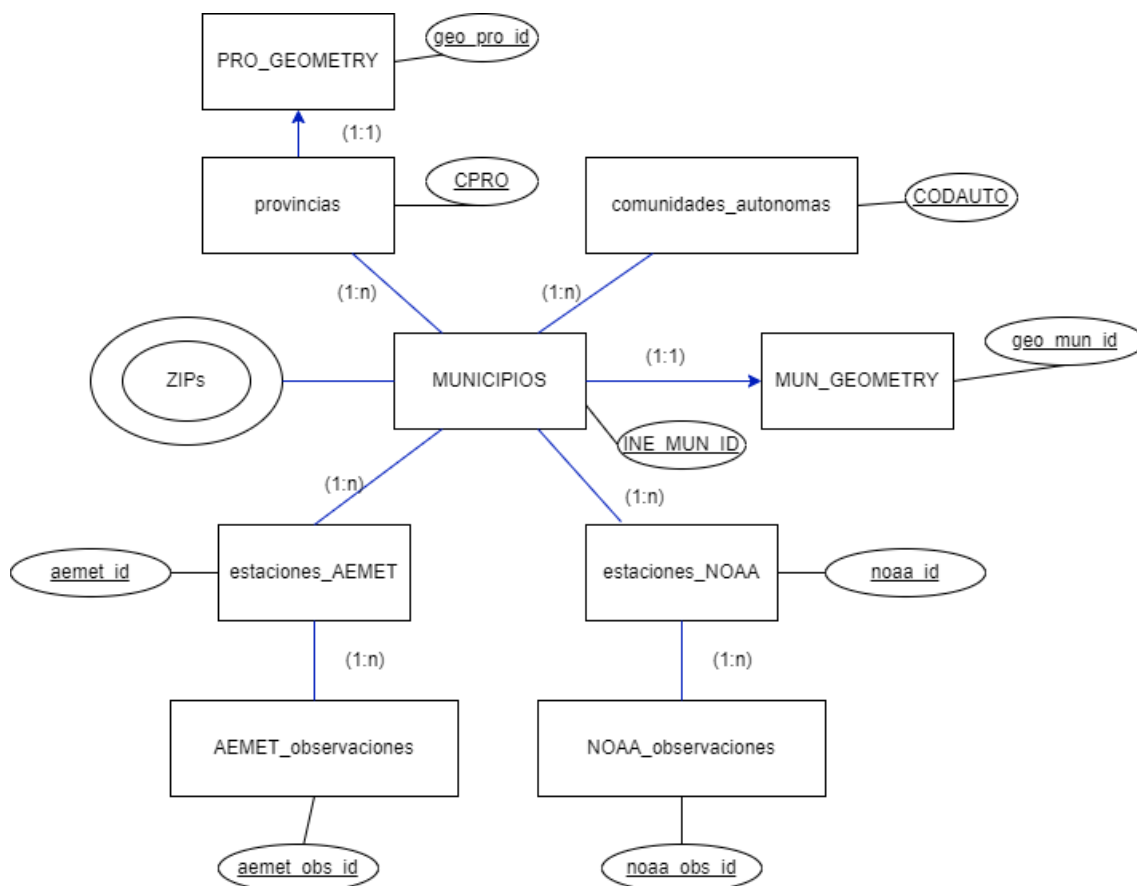
- provincias (alimentada con provincias.csv). Es una tabla muy parecida a la anterior ya que solo recoge el código INE de cada provincia y su nombre.
- MUNICIPIOS (alimentada con MUNICIPIOS.csv). Es una de las tablas centrales en esta base de datos, unifica los dataframes con información de todos los municipios del INE y todos los de la AEMET; así como los códigos de provincias y comunidades autónomas. Su primary key, INE\_MUN\_ID, es fundamental y es la que ha permitido la unificación de la mayoría de datos.
- ZIPs (alimentada con ZIPS.csv). Es una tabla conformada tan solo por dos columnas, el identificador de municipio y un código postal de este.
- PRO\_GEOMETRY (alimentada con PRO\_GEOMETRY.csv). Esta es una tabla que tan solo almacena el identificador INE de cada provincia y su polígono contorno en el mapa. Podría haberse unificado con la entidad provincias, pero he preferido almacenar los datos geográficos en tablas separadas.
- MUN\_GEOMETRY (alimentada con MUN\_GEOMETRY.csv). Homóloga a la anterior, pero con los identificadores y geometrías de municipios.
- estaciones\_AEMET (alimentada con estaciones\_AEMET.csv). Inventario de estaciones de la AEMET en el territorio español que contiene identificadores de municipio y estación, así como coordenadas geográficas.
- Estaciones\_NOAA (alimentada con estaciones\_NOAA.csv). Homóloga a la anterior con los datos disponibles en NOAA.
- AEMET\_observaciones (alimentada con AEMET\_observaciones.csv). Es la tabla con más información climatológica recogida. Guarda observaciones del pasado por fecha e identificador de estación, relativas a las temperaturas (en °C), precipitaciones (en mm), presión (en hPa) así como velocidad del viento (en Km/h).
- NOAA\_observaciones (alimentada con NOAA\_observaciones.csv). Homóloga anterior pero alimentada por las observaciones de las estaciones del NOAA, tan solo recoge información para las temperaturas y precipitación.

La base pues, se conforma tan solo de entidades y en su estructura puede incluso verse esta diferenciación de las tablas que tratan de datos geográficos y las que tratan de las observaciones meteorológicas.

Del código ETL se extrae además una tabla que no ha sido aquí incluida, la llamada "total\_municipios". Esta tabla fue pensada como una vista materializada, ya que unifica información de la tabla MUNICIPIOS, tabla ZIPs y tabla MUN\_GEOMETRY. Se pensó como vista materializada ya que Python es una herramienta mucho más potente que SQL y los datos geográficos que añadimos son realmente pesados. Además, no es una vista que necesite estar actualizándose constantemente. Dentro del código SQL se ha creado como tabla y ha sido alimentada por el fichero de igual nombre.

Además de la vista materializada ya comentada, vi interesante crear algunas vistas que pueden resultar como ejemplos de explotación de esta base de datos. El camino natural que toman estas bases de datos antes de ser explotadas por negocio, es el de pasar a conformar un datawarehouse, así que estas vistas pueden considerarse como sugerencias de explotación.

El diagrama entidad relación simplificado de esta base, quedaría entonces:



Como detalles técnicos, comentar que los datos geográficos los introduje como cadenas de texto largas (longtext) y posteriormente apliqué un update sobre las tablas originales que almacenara en un campo extra la misma información dentro de un campo de tipo geometry:

```
alter table PRO_GEOMETRY
add PRO_geometry_GEOM geometry;

update PRO_GEOMETRY
SET PRO_geometry_GEOM = st_geomfromtext(pro_geometry)
where geo_pro_id >0;
```

De cara a ir introduciendo el siguiente apartado, me centraré ahora en explicar la última de las tres vistas que se incluyen dentro del código .sql; la llamada *aemet\_vs\_noaa*.

```
create view aemet_vs_noaa (cpro, year_months, provincia,
noaa_mean_tmax, aemet_mean_tmax, noaa_mean_tmin, aemet_mean_tmin,
noaa_mean_prcp, aemet_mean_prcp, delta_tmax, delta_tmin, delta_prcp)
as
```

Lo que puede leerse arriba son todos los campos que conforman esta vista. Se trata del dataset que posteriormente exporté para poder analizar los datos de una forma mucho más visual en Tableau.



En esta tabla, se hace una comparativa entre las temperaturas máximas, temperaturas mínimas y precipitación promedio por provincia y mes durante todos los meses en los que existen observaciones. Los tres últimos campos resultan muy útiles porque representan de una forma directa la diferencia absoluta entre las medidas promedio de cada uno de los organismos de los que se han extraído datos, de ahí que se haya elegido llamarles deltas.

Para terminar esta sección y como comentario técnico, añadir que al no poder conectarme de forma remota a ninguna base de datos a la que el entorno académico tenga acceso, he decidido que la mejor solución era crear una en mi Host local y exportar el código en formato sql para que pueda ser empleado en otros servidores. La única modificación necesaria sería pues descargar todos los csv que alimentan las tablas y reemplazar las rutas de lectura de los mismos dentro del código.

Si en un futuro esta herramienta fuera puesta en marcha en un entorno profesional, la creación de estas tablas, así como alimentarlas, es algo que se podría haber hecho directamente también con Python, sin la necesidad de haber tenido que cambiar de herramienta.

### 3.3.- Ejemplo de explotación, Tableau.

Siempre que se quiere justificar la necesidad y funcionalidad de cualquier herramienta, el resultado suele ser más fructífero cuando las explicaciones y demostraciones se acompañan de material visual y atractivo. Por esto y por la satisfacción de ver que todas estas búsquedas en fuentes de datos remotas y muchas veces escondidas han servido para algo, decidí presentar un ejemplo de representación de los datos haciendo uso de la herramienta Tableau.

Como expliqué hace unos párrafos, para este propósito hice uso de una de las vistas que fueron generadas en la creación de la base de datos, en concreto, la que compara las observaciones recogidas por estaciones de NOAA y AEMET. Dentro de esta tabla, además, me centré solamente en las medidas promediadas de temperatura máxima.

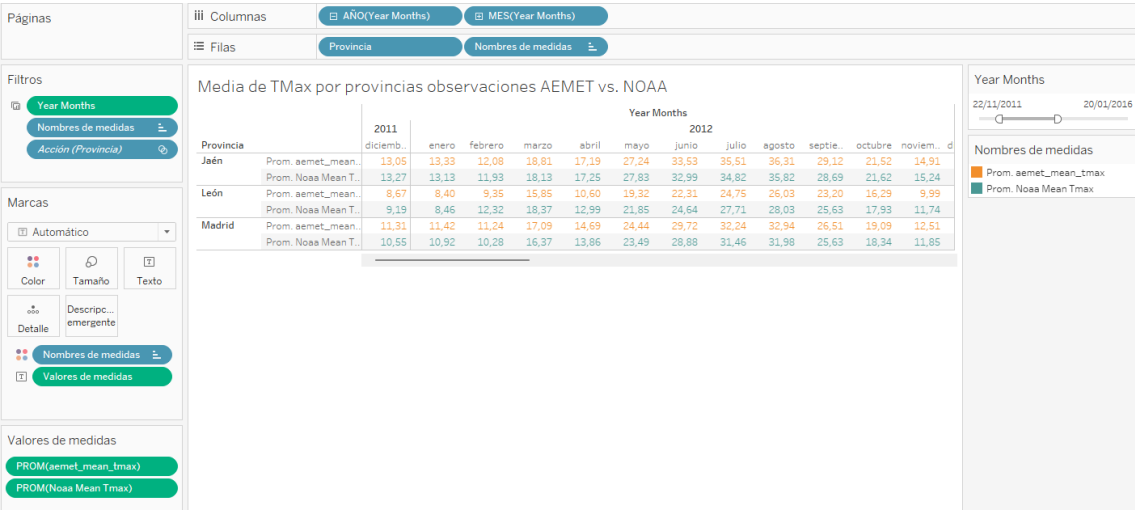
Además, hice un join con la entidad PRO\_GEOMETRY con la intención de confirmar si los datos geográficos descargados representaban correctamente el mapa con las provincias de España y poder así usarlo como un filtro de las demás representaciones al pinchar encima de una o más provincias. Así hice entonces la primera hoja de trabajo en Tableau.



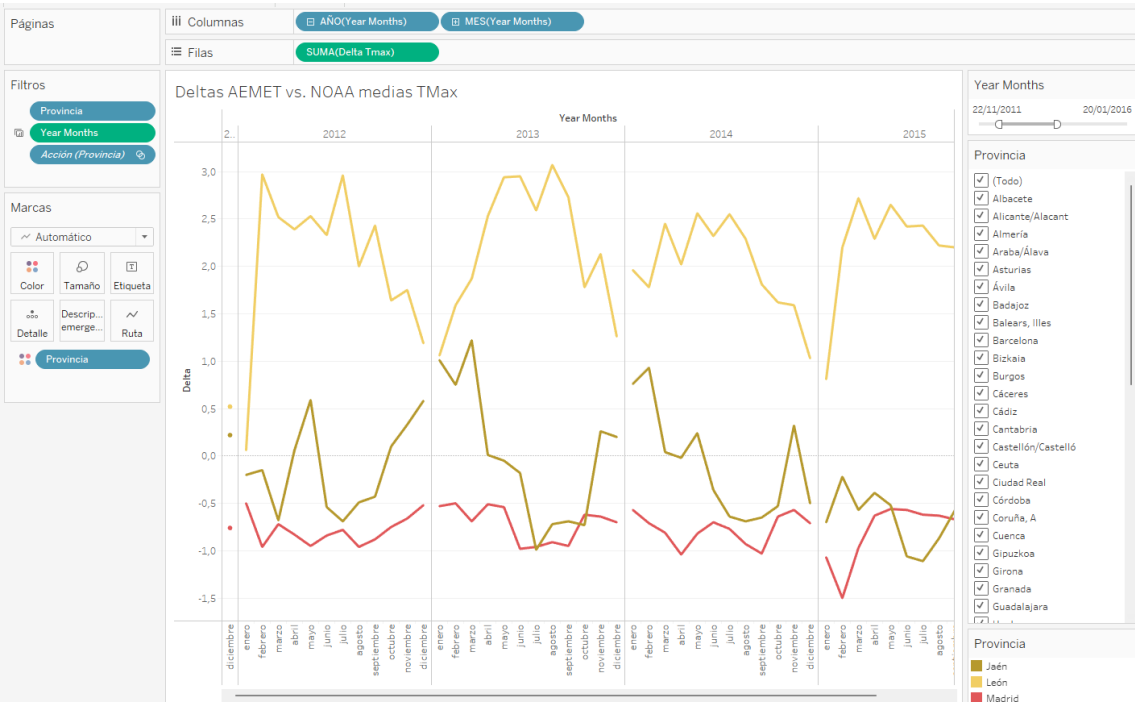


Como aclaración, a lo largo de todo el trabajo los datos de Canarias, Ceuta y Melilla también han sido considerados, pero en estas representaciones por comodidad quedan fuera de los recortes.

En segundo lugar, me pareció que podría aportar mucha información la representación de los valores directamente en una tabla. Así alguien podría consultar los datos concretos de diferentes meses de un solo vistazo sin tener que interpretar ninguna tabla ya que los datos de temperaturas un tipo de dato con los que todos estamos muy familiarizados.

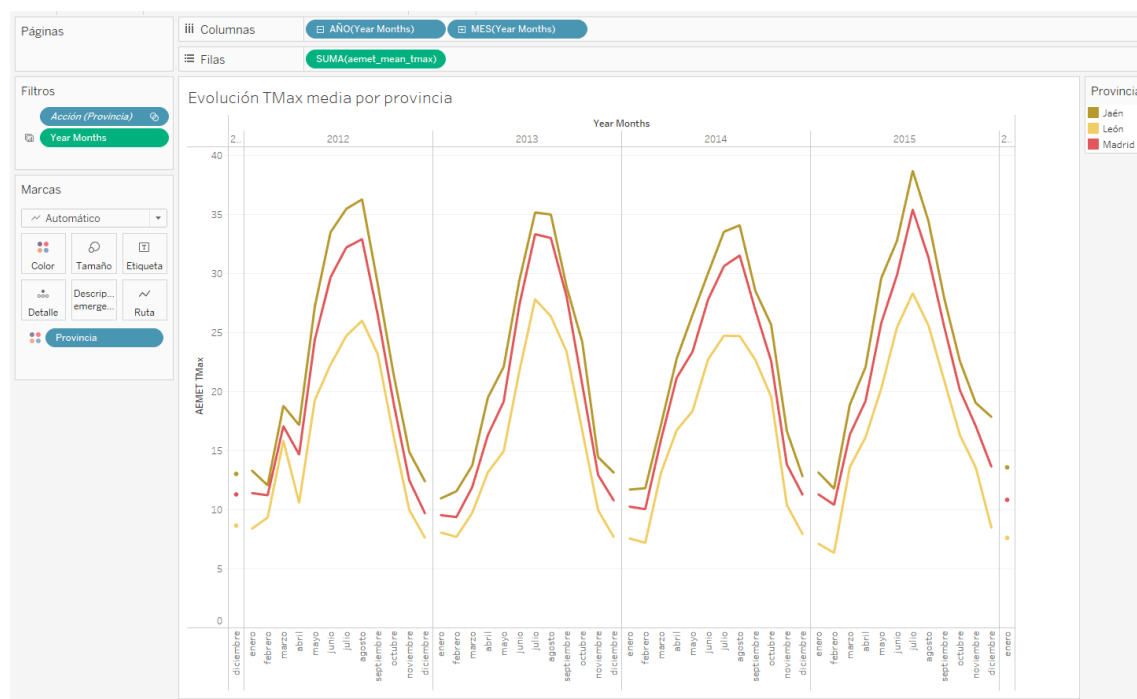


En la siguiente hoja de trabajo quise representar en una gráfica de línea los diferentes valores que toma la delta de la vista, que representa las diferencias entre los valores recogidos por ambas entidades. Esto podría ser interesante para detectar tendencias. Por ejemplo, podría detectarse que en las provincias del tercio norte de la península las medidas de las estaciones de NOAA siempre se encuentran entre 0 y 0,5 grados por debajo de las recogidas por las estaciones de AEMET. En un primer vistazo, sin embargo, el valor que toman estas deltas parece presentar un comportamiento errático y además depende fuertemente de la provincia escogida.



Por último, la cuarta hoja es un clásico. Se trata de la representación de la evolución de la temperatura máxima recogida solo por la AEMET en estos intervalos de tiempo considerados.

La intencionalidad de este gráfico final no es otra que el de ver que efectivamente son datos que presentan estacionalidad a medida que los años (o nunca mejor dicho, las estaciones) avanzan. Aunque justo en esta última hoja se representan datos solo de AEMET, la función de este gráfico es importante ya que permite dar contexto del rango de fechas seleccionado, haciéndonos a la idea de si hemos escogido uno demasiado ancho o pequeño; o también aportando información de años anteriores si por ejemplo nos encontramos mirando datos de unos meses específicos en la tabla que ofrece la segunda hoja.

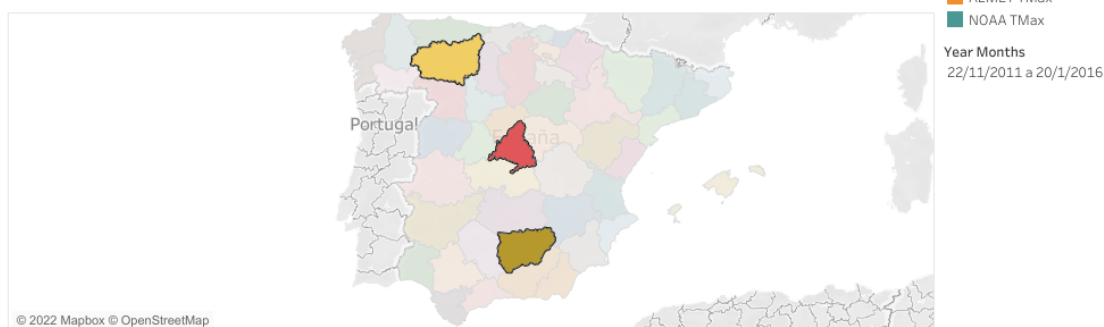


Unificando las cuatro hojas en un dashboard; haciendo que la primera de ellas funcione como filtro sobre todas las demás y añadiendo también un filtro para el rango de fechas seleccionado publiqué el dashboard resultante en *Tableau Cloud*:

[https://dub01.online.tableau.com/t/andreadgalis/views/AndreaDelgadoTFMBI/TFMBIvisualizacion?:origen=card\\_share\\_link&:embed=n](https://dub01.online.tableau.com/t/andreadgalis/views/AndreaDelgadoTFMBI/TFMBIvisualizacion?:origen=card_share_link&:embed=n)

La apariencia final de este capturada en una imagen queda como se muestra en la siguiente imagen.

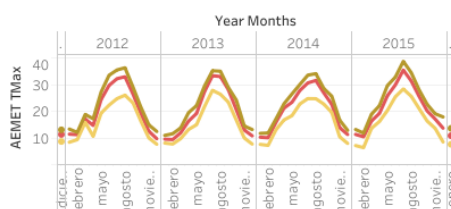
## Provincias España



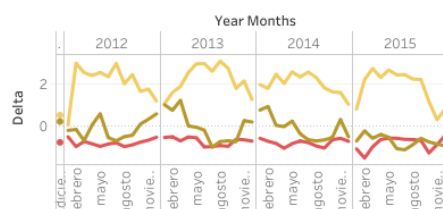
## Media de TMax por provincias observaciones AEMET vs. NOAA

Provincia		Year Months									
		2011									2012
		diciemb..	enero	febrero	marzo	abril	mayo	junio	julio	agosto	septie.
Jaén	AEMET TMax	13,05	13,33	12,08	18,81	17,19	27,24	33,53	35,51	36,31	29,12
	NOAA TMax	13,27	13,13	11,93	18,13	17,25	27,83	32,99	34,82	35,82	28,69
León	AEMET TMax	8,67	8,40	9,35	15,85	10,60	19,32	22,31	24,75	26,03	23,20
	NOAA TMax	9,19	8,46	12,32	18,37	12,99	21,85	24,64	27,71	28,03	25,63

## Evolución TMax media por provincia



## Deltas AEMET vs. NOAA medias TMax



## 4.- Caso de negocio

Lo primero a considerar cuando queremos presentar la herramienta como solución a alguien es ver quién puede necesitarla para así centrar el foco en uno o más sectores y ajustar pequeños detalles de la herramienta a cada uno de ellos para que pudiera ser más atractiva.

En el caso de la temática en torno a la cual gira este pipeline, ya se discutió en la introducción que podía ser aplicada a una gran variedad de clientes. Por ejemplo, empresas del sector transporte que quisieran planificar horarios y rutas de sus viajes, empresas energéticas que pudieran ajustar mejor las horas de producción de energías renovables, empresas constructoras o que quisieran estudiar los mejores revestimientos aislantes a emplear según el territorio, empresas de organización de eventos que quisieran minimizar la posibilidad de lluvia en sus fiestas.... Incluso podría ser empleada en el entorno académico de investigación donde físicos, geólogos o biólogos quisieran analizar la evolución del calentamiento global.

Una vez elegido el sector, los puntos fuertes de la herramienta que presentar a todos ellos son comunes. Estos son:

- Todas las fuentes de datos empleadas son de acceso público.
- Todos los datos pueden ser actualizados con facilidad y puede programarse para ser ejecutada con cierta periodicidad.
- Ha sido programada empleando variables ajustables (rangos de fechas de descarga de datos, rutas) que hacen posible su customización de una forma muy sencilla.
- Es una herramienta versátil a la que podrían agregarse observaciones meteorológicas de otras entidades. Puede crecer en módulos.

- Ayuda a reducir y homogeneizar las diferencias entre las observaciones y predicciones cuando estas provienen de diferentes proveedores de datos.
- Podría personalizarse la parte de creación de vistas en la base de datos a las que cada sector o compañía necesitase, es una herramienta adaptable a los contextos.
- La parte más técnica de esta se encuentra programada en *Python*, un lenguaje potente y conocido en casi todos los sectores. Además, el haberlo hecho en *Google Colab* hace que pueda ser ejecutado en la nube sin necesidad de consumir recursos de los servidores de los clientes
- Permite evolutivos de forma sencilla.

El primer elemento de la anterior lista es sin duda uno de los grandes puntos fuertes de esta herramienta, que pude ofrecerse como alternativa a otras fuentes de datos y programas de pago existente. Tan solo la tabla que almacena los códigos postales representa un gran valor ya que como se referenció, es una información que, sin bucear en exceso en *Google*, parece tan solo ofrecida por Correos a través de un desembolso de mínimo 831€.<sup>5</sup>

Otra necesidad que suple este pipeline, es que se centra en valores y medidas reales del pasado. Como es evidente, estos datos son los realmente valiosos y necesarios a la hora de estudiar tendencias (cambio climático) y poder aplicar algoritmos de predicción; y es que muchas de las fuentes de datos y búsquedas en páginas webs, se centran en mostrar tan solo las predicciones, no los valores de las observaciones reales recogidos por las estaciones.

Como no conozco herramientas similares, propondría esta directamente como alternativa a las descargas manuales de los datos en las páginas webs de NOAA y AEMET, lo cual no hay necesidad de desarrollar mucho porque el avance y la ganancia en comodidad es bastante evidente.

El ejemplo de uso de estos datos en BI desarrollado en el punto anterior de esta documentación, podría ser una estupenda carta de presentación de la herramienta.

## 5.- Futuras evoluciones y mejoras

Siempre que se cumple con los objetivos de un proyecto o desarrollo inicial, hay que dejar la puerta abierta a considerar posibles mejoras y evoluciones de los procesos. Sobre la versión “1.0” que podría llamarse a la que aquí presento, hay esencialmente 3 vías de mejora que se me han ocurrido podrían ponerse en práctica en entornos profesionales que fueran a mantener la herramienta:

- Integración total de las 3 partes del pipeline: Como ya he comentado en alguna parte de esta documentación, la solución podría funcionar de una forma 100% autónoma si por ejemplo tuviéramos una base de datos en algún CPD a la que un equipo de IT nos diera acceso y permisos. Así, el segundo módulo SQL podría haberse integrado también en Python y no hubiera sido necesario cambiar tan descaradamente de interfaz.
- Aumento de la red de estaciones: La segunda de las evoluciones que sería muy fácil llevar a cabo y que enriquecerían mucho esta herramienta es la de adquirir observaciones climatológicas de nuevas redes de estaciones. Hay empresas lo suficientemente grandes como para poseer su propia red de estaciones si por ejemplo en todas sus instalaciones industriales instalan una sencilla estación. En este escenario, entraría en juego una tercera fuente de observaciones que mejoraría aún más la parte de la homogeneización de las medidas recogidas y ayudaría a cubrir cada vez más territorio.

- Desarrollo del módulo predicción vs. Observación: Cuando investigué en la web de la AEMET Open Data, encontré más tipos de datos e información que descargar. Alguna era muy poco aprovechable en este trabajo, como por ejemplo algunos de los mapas o informes en PDF que ofrecen descargar. Uno de estos sets de datos con formato poco amigable, es precisamente el archivo de predicciones realizadas en el pasado. Aquí dejo un ejemplo de los datos que devuelve la API:

AGENCIA ESTATAL DE METEOROLOGÍA

PREDICCIÓN GENERAL PARA ESPAÑA

DÍA 09 DE AGOSTO DE 2022 A LAS 03:12 HORA OFICIAL

PREDICCIÓN VÁLIDA PARA EL JUEVES 11

A.- FENÓMENOS SIGNIFICATIVOS

Temperaturas elevadas en la mitad sur y en el nordeste peninsulares.

B.- PREDICCIÓN

Intervalos nubosos en la costa de Galicia y en el norte de las islas Canarias y, al principio, en el oeste de Andalucía, área del Estrecho y Melilla. Poco nuboso o despejado al comienzo del día en el resto del país, pero con nubosidad de evolución diurna en zonas de la mitad norte y del tercio oriental peninsular. Serán probables los chubascos y tormentas en el interior de Galicia, en la Cantábrica y en el Pirineo central y oriental, sin descartarlos en zonas del Cantábrico occidental, resto de Galicia y del Prepirineo aragonés y catalán.

Son probables las nieblas costeras en el litoral de Galicia y algo

Como se muestra, es una predicción presentada en texto, con descripciones generales de cómo se esperaba encontrar “el tiempo” en el territorio nacional. Una idea que aplicar aquí es hacer uso de las herramientas aprendidas en el módulo de *text mining* que permitieran “leer” los textos y con un analizador de sentimientos específico modificado hacia un analizador de buen/mal tiempo, poder comparar estas

predicciones con las observaciones recogidas en las estaciones. En estos textos de predicción es habitual además encontrar nombres de comunidades autónomas, provincias o municipios por lo que una vez más, contener toda esta información en la base de datos aquí desarrollada jugaría un papel fundamental a la hora de desarrollar este “lector” de predicciones.