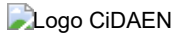


Módulo 7

Capstone 7. Parte 1: Ingesta de datos de demanda eléctrica en InfluxDB

Enrique González, Luis de la Ossa
Máster en Ciencia de Datos e Ingeniería de Datos en la Nube III
Universidad de Castilla-La Mancha

Marta Bellón Castro
Curso 2022-2023

Índice

In [1]:
```
import matplotlib.pyplot as plt

# Optimiza los gráficos para pantalla retina
%config InlineBackend.figure_format = 'retina'
# Por defecto usamos el backend inline
%matplotlib inline

# Oculta warnings
import warnings
warnings.simplefilter('ignore')

# La libreta ocupa así el 95% de la pantalla
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:95% !important; }</style>"))
```

1. Introducción

En este Capstone os pondréis en la piel de un *data scientist* que tiene como objetivo prototipar un sistema de visualización y pronóstico de la demanda de energía horaria española. Este prototipo incluirá las siguientes funcionalidades:

- **Extracción de datos de demanda eléctrica desde Red Eléctrica**. Se deberán extraer los datos de demanda peninsular del sistema de Red eléctrica que están disponibles de forma libre a través de su API de datos.
- **Ingesta de datos continua a InfluxDB**. Los datos extraídos y procesados se ingestarán a InfluxDB de forma continuada con el objetivo de poder tener la visualización de la demanda real así como de diversas métricas obtenidas a partir de esta serie.
- **Creación de un modelo de pronóstico de datos de demanda**. Crearéis un modelo de pronóstico sencillo para el siguiente día con los datos de demanda previamente ingestados en la base de datos.
- **Ingesta de pronóstico en la base de datos**. El pronóstico se ingestará también en la base de datos, para poder visualizar el resultado junto con el resto de información.

Este Capstone se ha dividido en dos partes para separar la parte de extracción e ingesta de datos de la parte de pronóstico. Se recomienda completar en gran medida esta primera parte ya que necesitaréis datos ya ingestados en la base de datos para poder hacer la segunda.

A continuación incluímos algunos import que necesitaréis para completar el resto del trabajo. Podéis incluir los que veáis necesarios.

In [2]:
```
import requests
import json
import pandas as pd
import datetime
import uuid
import pickle
import time

from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS
```

---

2. Extracción de datos de demanda de energía

Los datos de demanda de energía peninsular están disponibles a través de la página de Red Eléctrica Española, REDData (https://www.ree.es/es/apidatos). A través de esta página, Red Eléctrica ofrece de forma gratuita una gran cantidad de información no solo de la demanda si no también de otros temas relacionados con la generación, balance eléctrico, mercados, etc.

Para el caso de la demanda, Red Eléctrica ofrece una API tipo REST que contiene los datos actualizados de demanda (en MW) hasta el momento actual en intervalos de 10 minutos. Usaremos esta API para descargar todos los datos (tanto históricos como actuales) de la demanda.

En primer lugar, veamos qué tipos de datos incluye esta API. Para su uso, no es necesario ningún tipo de autenticación, si no que simplemente podremos realizar una petición de tipo GET al *endpoint* de la API, que nos devolverá los datos pedidos. Veamos un ejemplo.

In [3]:
```
import requests
import json

r = requests.get('https://apidatos.ree.es/en/datos/demanda/demanda-tiempo-real?start_date=2023-02-19T00:00&end_date=2023-02-19T23:59&time_trunc=hour')

r.json()
```

...

Como véis, el uso de la API es bastante sencillo. A continuación pasamos a describir los principales campos tanto de la petición como de la respuesta. La *url* para hacer peticiones sobre demanda de energía es la siguiente:

```
https://apidatos.ree.es/en/datos/demanda/demanda-tiempo-real
```

A esta URL es necesario pasarle una serie de parámetros para definir qué serie de datos queremos obtener. Los parámetros son los siguientes:

- `start_date` : La fecha y hora de inicio de los datos que queremos obtener. Tiene el formato `YYYY-MM-DDTHH:MM` .
- `end_date` : La fecha y hora de fin de los datos a obtener. Sigue el mismo formato que `start_date`
- `time_trunc` : El formato de agregación de los datos. Admite varios formatos, aunque nosotros elegiremos `hour` .

En cuanto a la respuesta de la API, podéis ver que es un json que nos proporciona a periodos de 10 minutos con (entre otra información):

- La demanda instantánea (en MW)
- La demanda pronosticada por REE (en MW)
- La demanda programada (en MW)

Podéis consultar más información sobre esta API en página de la API (https://www.ree.es/es/apidatos).

Para este trabajo, solo nos interesan las dos primeras, ya que de esta forma podremos comparar el modelo que hemos obtenido con el modelo de pronóstico realizado por Red Eléctrica. Por tanto, lo primero será extraer estos datos y procesarlos correctamente para poder incluirlos en InfluxDB. Es relativamente sencillo modelar estos datos de serie temporal en la base de datos:

- Nuestra medida será la demanda: `demand` .
- Bajo esta, no incluiremos ninguna etiqueta adicional.
- Usaremos dos campos, uno de demanda real (`RealDemand`) y otro de pronosticada (`ForecastREE`).

Sin embargo, hasta llegar a este punto, tendremos que implementar algunas funciones que nos permitan tanto el procesado como la escritura de los datos en InfluxDB.

Tarea 1

Dados los resultados de la API (de la que podéis tomar como ejemplo, el resultado anterior), implementa la siguiente función que toma la respuesta de la API y la convierte en un DataFrame con las siguientes características:

- Debe ser un DataFrame con las siguientes columnas: `RealDemand` (con la demanda real de la API) y `ForecastREE` (con el Forecast de REE).
- El DataFrame tiene que tener un índice temporal. Os recomendamos usar timestamps en utc de cara a tener menos problemas (cambios de hora, etc.)
- La frecuencia de los datos debe ser horaria (deberéis agregar por la media)

De momento no os preocupéis por como obtener los datos simplemente queremos procesarlos.

Pista: Para este procesamiento os resultará útil las primeras prácticas sobre análisis de series temporales.

In [4]:
```python
# Crea un dataframe a partir de la respuesta json.

def process_response(json_response: dict) -> pd.DataFrame:

    # Obtengo la lista de valores
    demand_values = json_response['included'][0]['attributes']['values']

    # Convierto las fechas y extraigo los valores reales
    demand_dates = [pd.Timestamp(value['datetime']).tz_convert('UTC') for value in demand_values]
    demand_real = [value['value'] for value in demand_values]

    # Creo un df con los valores de la demanda y sus fechas correspondientes
    demand_df = pd.DataFrame({'RealDemand': demand_real}, index=demand_dates)

    # Extraigo la lista de valores del forecast de la API
    forecast_values = json_response['included'][2]['attributes']['values']

    # Convierto las fechas y extraigo los valores del forecast
    forecast_dates = [pd.Timestamp(value['datetime']) for value in forecast_values]
    forecast_values = [value['value'] for value in forecast_values]

    # Creo un df
    forecast_df = pd.DataFrame({'ForecastREE': forecast_values}, index=forecast_dates)

    # Uno el df de la demanda con el df del forecast
    final_df = pd.merge(demand_df, forecast_df, how='outer', left_index=True, right_index=True)

    # Promedio la frecuencia a horas
    final_df = final_df.resample('H').mean().rename_axis('datetime')
    final_df[['RealDemand', 'ForecastREE']] = final_df[['RealDemand', 'ForecastREE']].round(decimals=1)

    return final_df
```

In [5]:
```
result = process_response(r.json())
result
```

Out[5]:

datetime	RealDemand	ForecastREE
2023-02-18 23:00:00+00:00	23709.2	23799.8
2023-02-19 00:00:00+00:00	22150.0	22076.3
2023-02-19 01:00:00+00:00	20940.1	20941.2
2023-02-19 02:00:00+00:00	20202.0	20349.5
2023-02-19 03:00:00+00:00	19905.2	19983.6
2023-02-19 04:00:00+00:00	19929.2	19959.7
2023-02-19 05:00:00+00:00	20429.6	20499.9
2023-02-19 06:00:00+00:00	21052.8	21164.8
2023-02-19 07:00:00+00:00	21556.8	21523.7
2023-02-19 08:00:00+00:00	23180.3	23220.2
2023-02-19 09:00:00+00:00	24446.1	24526.8
2023-02-19 10:00:00+00:00	24540.7	24782.4
2023-02-19 11:00:00+00:00	24382.7	24553.8
2023-02-19 12:00:00+00:00	24376.3	24720.3
2023-02-19 13:00:00+00:00	24397.9	24813.8
2023-02-19 14:00:00+00:00	23391.8	23480.4
2023-02-19 15:00:00+00:00	23132.1	23017.6
2023-02-19 16:00:00+00:00	23338.2	23145.2
2023-02-19 17:00:00+00:00	24566.7	24787.1
2023-02-19 18:00:00+00:00	27442.2	27628.2
2023-02-19 19:00:00+00:00	28908.9	28982.0
2023-02-19 20:00:00+00:00	28946.8	29079.4
2023-02-19 21:00:00+00:00	27355.5	27365.2
2023-02-19 22:00:00+00:00	25043.3	24998.2

Resultado esperado:

datetime	RealDemand	ForecastDemand
2023-02-18 23:00:00+00:00	23709.2	23799.8
2023-02-19 00:00:00+00:00	22150	22076.3
2023-02-19 01:00:00+00:00	20940.1	20941.2

datetime	RealDemand	ForecastDemand
2023-02-19 02:00:00+00:00	20202	20349.5
2023-02-19 03:00:00+00:00	19905.2	19983.6
2023-02-19 04:00:00+00:00	19929.2	19959.7
2023-02-19 05:00:00+00:00	20429.6	20499.9
2023-02-19 06:00:00+00:00	21052.8	21164.8
2023-02-19 07:00:00+00:00	21556.8	21523.7
2023-02-19 08:00:00+00:00	23180.3	23220.2
2023-02-19 09:00:00+00:00	24446.1	24526.8
2023-02-19 10:00:00+00:00	24540.7	24782.4
2023-02-19 11:00:00+00:00	24382.7	24553.8
2023-02-19 12:00:00+00:00	24376.3	24720.3
2023-02-19 13:00:00+00:00	24397.9	24813.8
2023-02-19 14:00:00+00:00	23391.8	23480.4
2023-02-19 15:00:00+00:00	23132.1	23017.6
2023-02-19 16:00:00+00:00	23338.2	23145.2
2023-02-19 17:00:00+00:00	24566.7	24787.1
2023-02-19 18:00:00+00:00	27442.2	27628.2
2023-02-19 19:00:00+00:00	28908.9	28982
2023-02-19 20:00:00+00:00	28946.8	29079.4
2023-02-19 21:00:00+00:00	27355.5	27365.2

Nuestro objetivo final con estos datos es poder ingestarlos de forma continuada en la base de datos de InfluxDB. Con el procesado de la tarea anterior obtenemos unos datos en un formato útil tanto para el análisis como para la ingesta de datos en InfluxDB, ya que, si recordáis de la práctica que hicimos con la base de datos, era posible ingestar datos directamente desde un DataFrame. El siguiente paso será, por tanto, facilitar la escritura en InfluxDB de los datos que hemos procesado en la tarea anterior a InfluxDB.

Tarea 2

Implementa la siguiente función que ingesta un dataframe como el que habéis procesado en el ejercicio anterior en InfluxDB. Para ello utilizad bucket llamado `capstone7`, que deberéis crear previamente.

Con ello ya podréis usar la información de acceso para escribir el DataFrame. Indicad como medida `demand`, la cual tendrá dos campos, la demanda real (`RealDemand`) y la demanda predicha por Red Eléctrica (`ForecastDemand`).

Una vez implementada la función os recomendamos ejecutad el código proporcionado para ingestar los datos y **incluíd una captura de los datos en formato gráfico como la que se da de ejemplo junto con el cuaderno**. Fijáos bien en el rango de tiempo en la plataforma de InfluxDB que coincida con la de los datos que estáis ingestando.

In [6]:
```
# Podéis incluir la información de vuestra instancia de InfluxDB a continuación
token = "mcidaen_token"
org = "mcidaen"
bucket = "capstone7"

client = InfluxDBClient(url="http://influxdb:8086", token=token, org=org)
```

In [7]:
```
# Creo un cliente de escritura síncrono
write_api = client.write_api(write_options=SYNCHRONOUS)
```

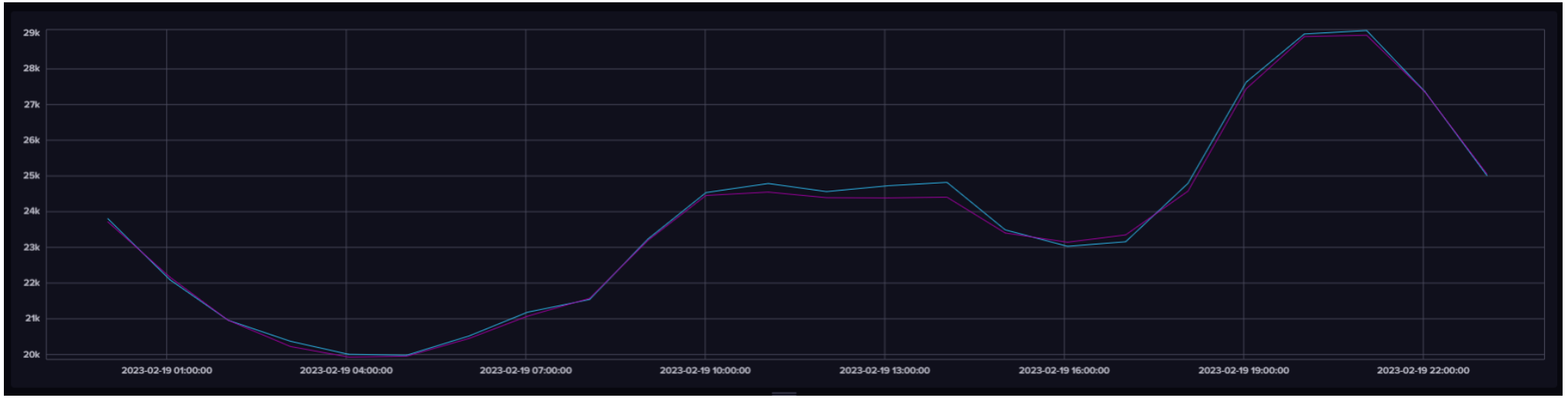
In [8]:
```
# Escribo el DataFrame en InfluxDB

def save_to_influxdb(df: pd.DataFrame, client: InfluxDBClient, bucket: str, org: str) -> pd.DataFrame:

    result = write_api.write(bucket, org, record=df, data_frame_measurement_name='demand')
    return result
```

In [9]:
```
save_to_influxdb(result, client, bucket, org)
```

Mi resultado es el siguiente:



Resultado esperado:

- Comprobad mediante el explorar que obtenéis una curva de demanda. Fijaos bien que estéis obteniendo el timestamp correcto en InfluxDB:

Con las dos funciones anteriores, ya tenemos la capacidad tanto de procesar los resultados de la API como de ingestar los datos en nuestra instancia de InfluxDB. Sin embargo, todavía no hemos implementado ninguna función para la petición de datos a la API de Red Eléctrica. Esto lo váis a hacer a continuación.

Tarea 3

Implementa la siguiente función que obtendrá los datos de la API para un periodo de tiempo determinado entre `begin` y `end`, ambos de tipo `datetime`. Devolverá el objeto `json` que parseamos con la función del primer ejercicio. No necesitáis parsear la respuesta más allá de convertirla en un objeto `json`.

Opcional: Como recomendación en este punto, y para evitar posibles fallos en las peticiones, podéis implementar una lógica de reintentos separados mediante un *sleep* y capturando, por ejemplo, request.RequestException o alguna de sus [subexcepciones (https://requests.readthedocs.io/en/master/_modules/requests/exceptions/)](https://requests.readthedocs.io/en/master/_modules/requests/exceptions/)

In [10]:
```python
# Obtén los datos de  demanda entre begin y end

def get_demand_data(begin: datetime.datetime, end: datetime.datetime) -> dict:

    url = 'https://apidatos.ree.es/en/datos/demanda/demanda-tiempo-real'
    params = {
        'start_date': begin.strftime('%Y-%m-%dT%H:%M'),
        'end_date': end.strftime('%Y-%m-%dT%H:%M'),
        'time_trunc': 'hour'
    }
    retries = 3 # Nº de intentos
    delay = 5  # Se reintenta después de 5 segundos

    for _ in range(retries):
        try:
            response = requests.get(url, params=params)
            response.raise_for_status()
            return response.json()
        except (requests.RequestException, ValueError) as e:
            print(f"Ocurrió un error: {e}")
            print(f"Reintentado en {delay} segundos...")
            time.sleep(delay)

    return {}  # No devuelve nada si hay algún fallo
```

In [11]:
```python
result = get_demand_data(
    datetime.datetime(2023, 2, 10),
    datetime.datetime(2023, 2, 10, 23, 59)
)
```

In [12]:
```
process_response(result)
```

Out[12]:

datetime	RealDemand	ForecastREE
2023-02-09 23:00:00+00:00	27342.2	27348.3
2023-02-10 00:00:00+00:00	25742.6	25497.5
2023-02-10 01:00:00+00:00	24570.7	24416.2
2023-02-10 02:00:00+00:00	24028.8	23943.8
2023-02-10 03:00:00+00:00	23989.6	23924.6
2023-02-10 04:00:00+00:00	24810.3	24531.0
2023-02-10 05:00:00+00:00	27752.0	27487.2
2023-02-10 06:00:00+00:00	32448.4	32457.0
2023-02-10 07:00:00+00:00	34902.8	35021.2
2023-02-10 08:00:00+00:00	35205.7	35168.2
2023-02-10 09:00:00+00:00	35077.1	34983.6
2023-02-10 10:00:00+00:00	34527.8	33946.7
2023-02-10 11:00:00+00:00	33772.0	33779.6
2023-02-10 12:00:00+00:00	33348.3	33825.4
2023-02-10 13:00:00+00:00	32682.5	32848.4
2023-02-10 14:00:00+00:00	31999.4	32146.8
2023-02-10 15:00:00+00:00	32090.7	31974.4
2023-02-10 16:00:00+00:00	32412.9	32210.0
2023-02-10 17:00:00+00:00	33319.7	33449.8
2023-02-10 18:00:00+00:00	35442.3	35570.4
2023-02-10 19:00:00+00:00	35978.5	36049.8
2023-02-10 20:00:00+00:00	35085.8	35067.9
2023-02-10 21:00:00+00:00	32079.8	32158.5
2023-02-10 22:00:00+00:00	29239.5	29208.1

Resultado esperado:

datetime	RealDemand	ForecastDemand
2023-02-09 23:00:00+00:00	27342.2	27348.3
2023-02-10 00:00:00+00:00	25742.6	25497.5
2023-02-10 01:00:00+00:00	24570.7	24416.2

datetime	RealDemand	ForecastDemand
2023-02-10 02:00:00+00:00	24028.8	23943.8
2023-02-10 03:00:00+00:00	23989.6	23924.6
2023-02-10 04:00:00+00:00	24810.3	24531
2023-02-10 05:00:00+00:00	27752	27487.2
2023-02-10 06:00:00+00:00	32448.4	32457
2023-02-10 07:00:00+00:00	34902.8	35021.2
2023-02-10 08:00:00+00:00	35205.7	35168.2
2023-02-10 09:00:00+00:00	35077.1	34983.6
2023-02-10 10:00:00+00:00	34527.8	33946.7
2023-02-10 11:00:00+00:00	33772	33779.6
2023-02-10 12:00:00+00:00	33348.3	33825.4
2023-02-10 13:00:00+00:00	32682.5	32848.4
2023-02-10 14:00:00+00:00	31999.4	32146.8
2023-02-10 15:00:00+00:00	32090.7	31974.4
2023-02-10 16:00:00+00:00	32412.9	32210
2023-02-10 17:00:00+00:00	33319.7	33449.8
2023-02-10 18:00:00+00:00	35442.3	35570.4
2023-02-10 19:00:00+00:00	35978.5	36049.8
2023-02-10 20:00:00+00:00	35085.8	35067.9
2023-02-10 21:00:00+00:00	32979.8	32158.5

Con estas tres funciones, ya tenemos lo necesario para la extracción de datos a partir de la API. En el siguiente apartado implementaréis la lógica completa de ingesta.

3. Ingesta y visualización de datos.

Con el objetivo de realizar un proceso de ingesta relativamente robusto y que nos permita ingestar de forma robusta incluso ante pequeños fallos, vamos a valernos de una lista de días ya procesados y de un proceso de extracción e ingesta que tenga en cuenta los valores ya procesados.

En realidad, para nuestro caso, ya que usamos InfluxDB, podemos sin problema reiniciar la ingesta tantas veces como queramos, ya que los datos antiguos se sobreescribirán (y el valor debería ser el mismo). Sin embargo, como pretendemos descargar una cantidad de datos considerable, no es recomendable hacer la ingesta completa en repetidas ocasiones. Para evitar volver a procesar días, mantendremos un conjunto con los dias ya ingestados y crearemos una lógica que tenga en cuenta este conjunto y que se limite a volver a ingestar datos desde un día hasta el momento actual.

Para realizar el proceso de ingesta se os proporciona la siguiente función implementada. Esta toma como parámetro el inicio desde el que queremos descargar datos `begin_ts` y un conjunto con los días que ya se han descargado previamente, `ingested`. El objetivo de la función es ingestar todos los días que no estén en InfluxDB desde la fecha inicial hasta el momento actual y retornar la lista de días (datetime) que se han ingestado. La función realiza descargas diarias (la API tiene un límite de puntos que se pueden descargar). Por supuesto, podéis modificar (y mejorar) esta función si lo creéis necesario.

In [13]:
```python
def ingest_data(begin_ts: datetime.datetime, ingested: set) -> list:
    """
    Obtiene los datos de demanda desde begin_ts
    """

    now = datetime.datetime.now()
    end_ts = datetime.datetime(now.year, now.month, now.day, now.hour, 0) - datetime.timedelta(seconds=1)
    begin = begin_ts
    end = min(begin + datetime.timedelta(days=1, minutes=-1), end_ts)
    l = []
    df_l = []
    try:
        while begin < end_ts:
            if begin not in ingested:
                print(f"Getting data for: {begin} ----> {end}")
                r = get_demand_data(begin, end)
                df = process_response(r)
                save_to_influxdb(df, client, bucket, org)
                l.append(begin)
            begin += datetime.timedelta(days=1)
            end = min(begin + datetime.timedelta(days=1, minutes=-1), end_ts)
    except Exception as e:
        print(f"An exception occurred: {e}")
        pass # Ignorar día y continuar con la siguiente iteración
    return l
```

In [14]:
```python
ingest_data(datetime.datetime(2023, 5, 1), set([]))
```

...

Con la función anterior ya podemos tener un proceso ejecutando de manera continua que se asegure de descargar los datos que nos falten en nuestro dataset. En este caso nosotros hemos optado por mantener el conjunto de datos ya procesados como un fichero (mediante `pickle`). En este conjunto incluiremos todos los datos procesados salvo el último día, que estará incompleto y que deberemos continuar descargando para seguir actualizando los datos de nuestra base de datos.

En primer lugar, necesitaréis inicializar el conjunto de días ingestados con la siguiente celda. Lo que hacemos es crear un conjunto vacío y guardarlo en el fichero `ingested.db`. Este fichero luego lo iremos actualizando a medida que procesemos nuevos días. Os recomendamos que comenteis la celda una vez inicializado para evitar borrar erróneamente los datos que habéis procesado hasta el momento.

In [15]:
```
import pickle
ingested_file = 'ingested.db'
ingested = set([])
with open(ingested_file, 'wb') as f:
    pickle.dump(ingested, f)
```

In [16]:
```
begin_ts = datetime.datetime(2023, 5, 1, 0, 0)
```

In [17]:
```
while True:
    with open(ingested_file, 'rb') as f:
        ingested = pickle.load(f)
    l = ingest_data(begin_ts, ingested)[:-1] # Remove last day to keep updating it
    ingested = ingested.union(set(l))
    with open(ingested_file, 'wb') as f:
        pickle.dump(ingested, f)
    print("Ingestion finished, sleeping for 10 seconds")
    time.sleep(1)
```

...

Este proceso tardará un tiempo (posiblemente, hasta 1 hora). Mientras podéis comprobar en vuestra instancia de InfluxDB que todos los datos están llegando correctamente. Asimismo, podéis ir desarrollando la siguiente tarea o incluso la siguiente parte del capstone.

Tarea 4

Con los datos de demanda eléctrica que ya habéis ingestado (o estáis en proceso de ingestión), crea un dashboard con las siguientes celdas:

- Una celda de tipo Graph con los datos graficados de RealDemand y ForecastDemand.
- Una celda tipo tabla con tres columnas: time, RealDemand y ForecastDemand. Para poder obtenerla necesitaréis usar Flux y el método pivot.
- Una celda de tipo Gauge con la diferencia entre RealDemand y ForecastDemand

Todas estas métricas se tienen que calcular en función del filtro de tiempo aplicado en el dashboard

Incluíd una captura del Dashboard (incluyendo el fichero de imagen con la entrega) realizado así como las consultas flux de los tres cuadros.

Mi resultado es el siguiente:

Capstone 7 - Marta Bellón Castro

ADD CELL | ADD NOTE | Show Variables | Enable Annotations | ··· | SET AUTO REFRESH | Local | 2023-02-19 13:00 - 2023-02-26...

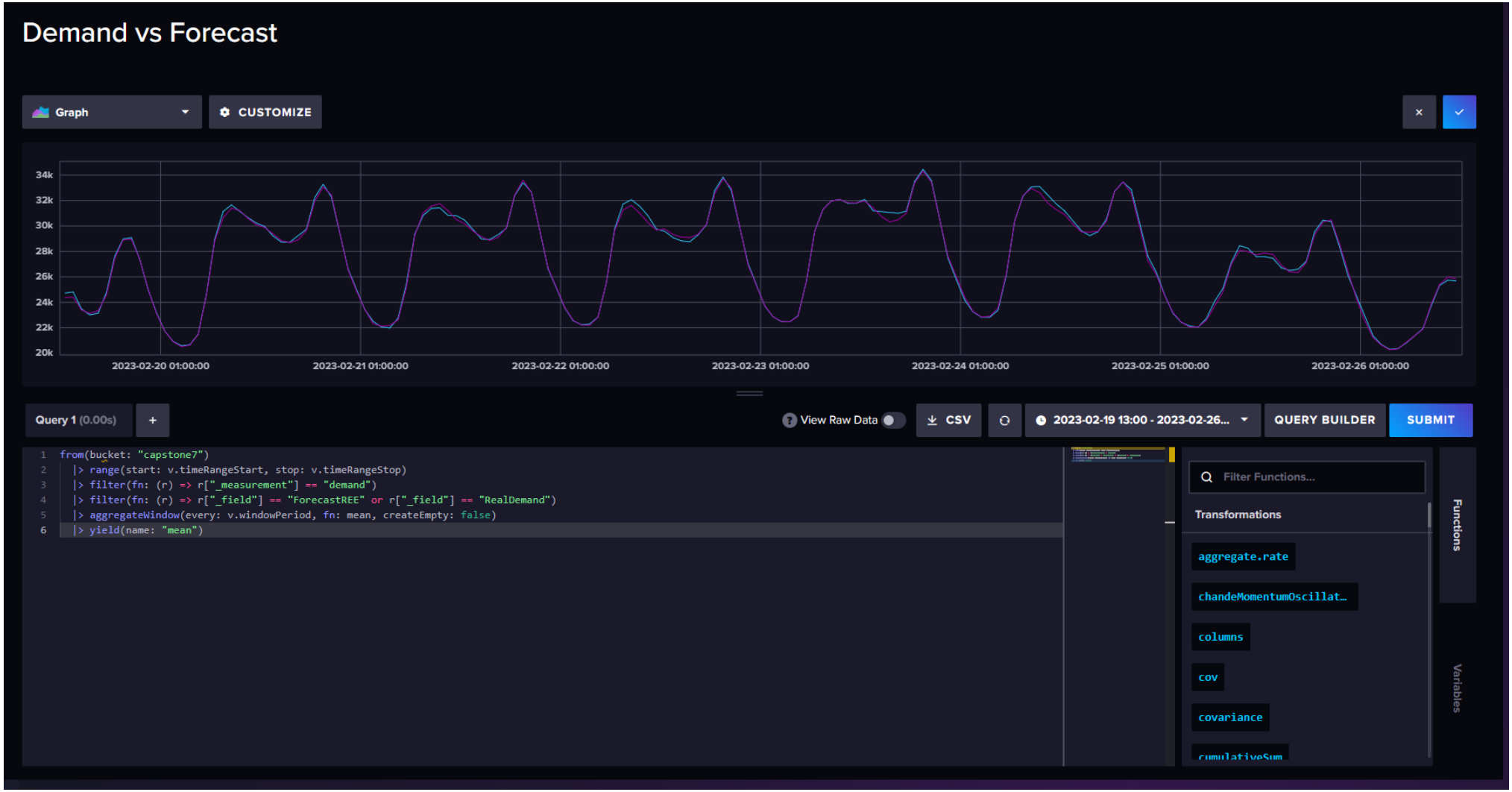
Demand vs Forecast

Demand table

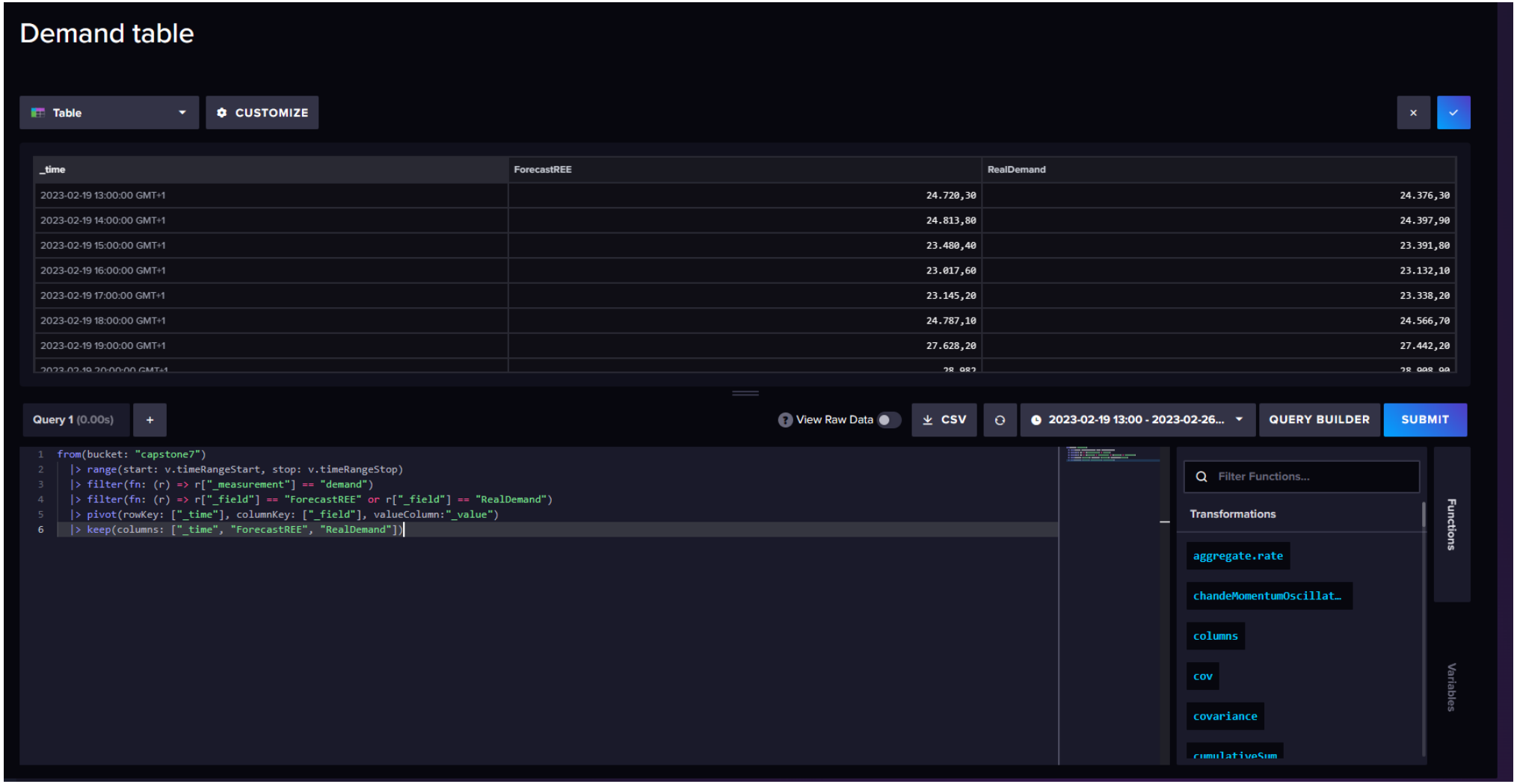
_time	ForecastREE	RealDemand
2023-02-19 13:00:00 GMT+1	24.720,30	24.376,30
2023-02-19 14:00:00 GMT+1	24.813,80	24.397,90
2023-02-19 15:00:00 GMT+1	23.480,40	23.391,80
2023-02-19 16:00:00 GMT+1	23.017,60	23.132,10
2023-02-19 17:00:00 GMT+1	23.145,20	23.338,20
2023-02-19 18:00:00 GMT+1	24.787,10	24.566,70
2023-02-19 19:00:00 GMT+1	27.628,20	27.442,20
2023-02-19 20:00:00 GMT+1	28.982	28.908,90
2023-02-19 21:00:00 GMT+1	29.079,40	28.946,80

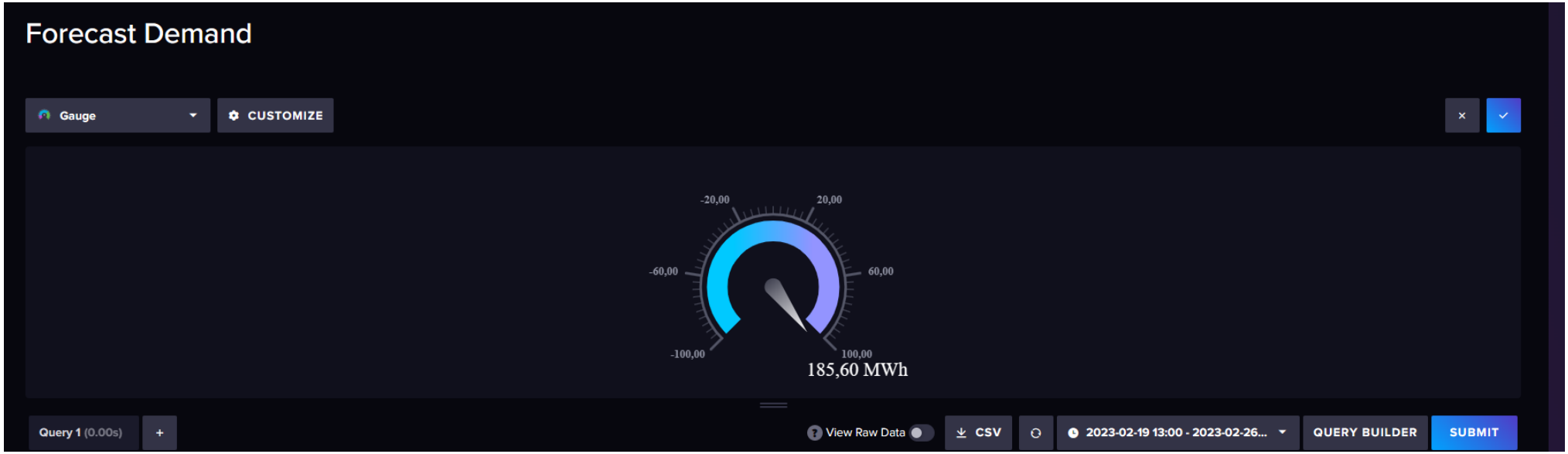
Forecast Demand

Demand vs Forecast



Demand table





Una vez completadas estas tareas, ya tenemos el prototipo de nuestro sistema de ingesta continua de datos de demanda eléctrica. Mientras dejéis en ejecución la celda anterior con la ingesta, los datos se seguiran actualizando en el sistema. Si tenéis que pararla, cuando volváis a ejecutarla, el proceso ingestará todos los días que faltan hasta el momento actual, por lo que podéis dejar el capstone y continuarlo en otro momento sin ningún problema.