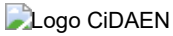

Módulo 7

Capstone 7. Parte 2: Pronóstico de demanda eléctrica

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

Marta Bellón Castro
Curso 2022-2023

Índice

In [1]:
```
import matplotlib.pyplot as plt
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error

# 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>"))
```

En la siguiente parte del capstone crearéis un modelo de pronóstico a partir de los datos ingestados en InfluxDB e añadiréis el pronóstico a un día vista de vuelta a InfluxDB de forma que en el sistema prototipado se pueda ver el pronóstico del día. Específicamente, para la creación del modelo utilizaréis los datos hasta el último día completo de datos y el pronóstico incluirá el dia siguiente (que podrá tener datos o no en nuestro sistema, pero que no los usaremos para entrenar).

---

1. Creación del modelo de pronóstico

Para la creación del modelo de pronóstico, necesitaremos descargarnos los datos de nuestra instancia para poder tener datos con los que entrenar.

Tarea 1

Implementa la siguiente función que devuelve un DataFrame con todos los datos disponibles de demanda eléctrica almacenados en tu instancia de InfluxDB. Solo nos interesa la serie de demanda real, por lo que tendréis que filtrar el Forecast de red eléctrica usando Flux.

Necesitaréis añadir la información de autenticación para vuestra instancia de InfluxDB a continuación.

Pista: La práctica de InfluxDB que hicimos en este módulo os resultará útil para poder formar la consulta.

In [2]:
```
token = "mcidaen_token"
org = "mcidaen"
bucket = "capstone7"

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

In [3]:
```
# Escribe el DataFrame en InfluxDB

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

    # Guardo el resultado de la consulta en df
    query_f = (
    f'from(bucket: "{bucket}")'
    f' |> range(start: 2022-01-01T00:00:00Z, stop: 2023-04-29T12:00:00Z)'
    f' |> filter(fn: (r) => r["_measurement"] == "demand")'
    f' |> filter(fn: (r) => r["_field"] == "RealDemand")'
    )
    query_client = client.query_api()
    df = query_client.query_data_frame(query_f, org=org)

    df_query = df.loc[:, ['_time', '_value']]
    df_query['datetime'] = pd.to_datetime(df_query['_time'], utc=True).dt.tz_localize(None)
    df_query['RealDemand'] = df_query['_value']
    df_query = df_query[['datetime', 'RealDemand']]
    df_query.index = df_query['datetime']

    return df_query['RealDemand'].sort_index()
```

In [4]:
```
df = load_from_influxdb(client, bucket, org)
df['2022'].sample(20, random_state=0)
```

Out[4]:
```
datetime
2022-02-09 04:00:00    23961.7
2022-01-24 13:00:00    35615.0
2022-02-28 20:00:00    31955.0
2022-02-17 17:00:00    30054.2
2022-01-07 04:00:00    21135.7
2022-01-02 13:00:00    24985.5
2022-03-26 21:00:00    26760.8
2022-02-06 14:00:00    25654.2
2022-03-12 14:00:00    26169.2
2022-01-06 15:00:00    23878.8
2022-03-21 07:00:00    29163.3
2022-03-20 17:00:00    24058.2
2022-02-10 02:00:00    23652.5
2022-01-31 11:00:00    32945.5
2022-03-04 01:00:00    23631.2
2022-03-17 20:00:00    32054.3
2022-03-09 03:00:00    22305.2
2022-02-03 19:00:00    34801.0
2022-02-16 22:00:00    27885.0
2022-01-17 07:00:00    33832.8
Name: RealDemand, dtype: float64
```

Resultado esperado:

datetime	RealDemand
2022-08-17 13:00:00	28558.1
2022-03-22 20:00:00	32101.3
2022-06-18 01:00:00	24592.2
2022-02-23 22:00:00	27416.5
2022-04-27 13:00:00	29158.5
2022-03-29 07:00:00	30680.5
2022-06-07 00:00:00	22885.6
2022-03-10 12:00:00	31206.7
2022-09-25 05:00:00	19363.2
2022-08-19 17:00:00	28068.4
2022-11-06 04:00:00	17734.4
2022-08-19 05:00:00	23990.7
2022-02-22 22:00:00	27137
2022-10-19 03:00:00	21814.9

datetime	RealDemand
2022-12-21 01:00:00	21955.2
2022-04-28 04:00:00	24237.8
2022-12-01 21:00:00	29663.8
2022-07-19 18:00:00	33729.7
2022-07-03 06:00:00	21313.8

Con estos datos, el objetivo es crear un modelo de forecasting que nos permita pronosticar a un día vista la demanda de energía horaria. Aunque podéis usar el modelo que estiméis (siempre que funcione con el resto de componentes) en este caso hemos optado por crear un modelo de Prophet.

Para facilitar su integración con el resto de código y poder ejecutar el modelo para pronosticar varios días, crearemos una función que entrene el modelo y realice un pronóstico para un conjunto de test. Para ello crearemos un modelo capaz de pronosticar, dado un histórico, la demanda de energía horaria del día siguiente al histórico.

Tarea 2

Implementa la siguiente función que, dada la serie temporal de demanda anterior, *training* y una lista de timestamps, *test*:

- Entrena un modelo de Prophet con los datos de *training*
- Pronostica las horas pasadas en *test*

Puedes asumir que el conjunto de datos tiene el formato de salida de la función de la tarea anterior.

Opcional: Como trabajo opcional, puedes probar varios modelos de los expuestos durante el módulo para ver si puedes obtener un mejor resultado. Esto se tendrá en cuenta a la hora de evaluación del capstone.

In [5]:
```python
# Entrena con training y pronostica los datos de test

import datetime
from prophet import Prophet

def train_forecast(training: pd.DataFrame, test: list) -> pd.DataFrame:
    # Creo el modelo Prophet
    model = Prophet()

    # Renombro las columnas para que se ajusten al formato requerido por Prophet
    df = training.to_frame().reset_index().rename(columns={'datetime': 'ds', 'RealDemand': 'y'})

    # Entreno el modelo con los datos de entrenamiento
    model.fit(df)

    # Creo un dataframe con las fechas de test
    future = pd.DataFrame({'ds': test})

    # Realizo la predicción para las fechas de test
    forecast = model.predict(future)

    # Devuelvo el dataframe de la predicción
    forecast=forecast[['ds', 'yhat']].rename(columns={'ds': 'datetime'}).round({'yhat': 1}).set_index('datetime')
    return forecast
```

In [6]:
```python
test = list(pd.date_range(start=datetime.datetime(2023, 2, 24), freq='H', periods=24))
```

In [7]:
```python
result = train_forecast(df[:datetime.datetime(2023, 2, 24)], test)
```

```
20:26:56 - cmdstanpy - INFO - Chain [1] start processing
20:26:59 - cmdstanpy - INFO - Chain [1] done processing
```

In [8]:
```
result
```

Out[8]:

datetime	yhat
2023-02-24 00:00:00	24193.1
2023-02-24 01:00:00	23214.7
2023-02-24 02:00:00	22464.5
2023-02-24 03:00:00	22177.6
2023-02-24 04:00:00	22899.7
2023-02-24 05:00:00	24845.2
2023-02-24 06:00:00	27478.9
2023-02-24 07:00:00	29790.5
2023-02-24 08:00:00	31035.9
2023-02-24 09:00:00	31245.5
2023-02-24 10:00:00	31029.5
2023-02-24 11:00:00	30905.8
2023-02-24 12:00:00	30842.0
2023-02-24 13:00:00	30459.6
2023-02-24 14:00:00	29654.4
2023-02-24 15:00:00	28936.7
2023-02-24 16:00:00	29067.6
2023-02-24 17:00:00	30291.2
2023-02-24 18:00:00	31918.1
2023-02-24 19:00:00	32728.0
2023-02-24 20:00:00	31885.2
2023-02-24 21:00:00	29564.5
2023-02-24 22:00:00	26742.5
2023-02-24 23:00:00	24383.9

Resultado esperado (aprox.):

datetime	yhat
2023-02-24 00:00:00	26204
2023-02-24 01:00:00	25438.5
2023-02-24 02:00:00	25002.7

datetime	yhat
2023-02-24 03:00:00	25110.6
2023-02-24 04:00:00	26045.4
2023-02-24 05:00:00	27773.1
2023-02-24 06:00:00	29804.7
2023-02-24 07:00:00	31483.9
2023-02-24 08:00:00	32447
2023-02-24 09:00:00	32819
2023-02-24 10:00:00	32970.4
2023-02-24 11:00:00	33098.5
2023-02-24 12:00:00	33064
2023-02-24 13:00:00	32644.8
2023-02-24 14:00:00	31925.6
2023-02-24 15:00:00	31379.3
2023-02-24 16:00:00	31499
2023-02-24 17:00:00	32298.6
2023-02-24 18:00:00	33173.7
2023-02-24 19:00:00	33299.9
2023-02-24 20:00:00	32246
2023-02-24 21:00:00	30273.3
2023-02-24 22:00:00	28081.7

Con esta función ya podemos evaluar nuestro modelo. En este caso, debemos recordar que debemos evaluar nuestro modelo de forma diaria, ya que nuestro objetivo es hacer pronósticos a un día vista. Para ello podemos valernos de la función anterior y recorrer diariamente los valores de un conjunto de test, añadiendo en el training los valores hasta la fecha. Solo usaremos los primeros días del mes de enero para probar el método. Veamos como hacerlo.

In [9]:
```
test_dates = list(pd.date_range(start=datetime.datetime(2023, 1, 1), end=datetime.datetime(2023, 1, 10), freq='D'))
```

In [10]:
```
maes = []
for test_date in test_dates:
    training = df[:test_date-datetime.timedelta(minutes=1)]
    test = df[test_date:test_date+datetime.timedelta(days=1, minutes=-1)]
    result = train_forecast(df, test.index)
    r = mean_absolute_error(test, result)
    maes.append(r)
```

```
20:27:12 - cmdstanpy - INFO - Chain [1] start processing
20:27:17 - cmdstanpy - INFO - Chain [1] done processing
20:27:17 - cmdstanpy - INFO - Chain [1] start processing
20:27:22 - cmdstanpy - INFO - Chain [1] done processing
20:27:23 - cmdstanpy - INFO - Chain [1] start processing
20:27:27 - cmdstanpy - INFO - Chain [1] done processing
20:27:27 - cmdstanpy - INFO - Chain [1] start processing
20:27:32 - cmdstanpy - INFO - Chain [1] done processing
20:27:33 - cmdstanpy - INFO - Chain [1] start processing
20:27:38 - cmdstanpy - INFO - Chain [1] done processing
20:27:39 - cmdstanpy - INFO - Chain [1] start processing
20:27:43 - cmdstanpy - INFO - Chain [1] done processing
20:27:43 - cmdstanpy - INFO - Chain [1] start processing
20:27:48 - cmdstanpy - INFO - Chain [1] done processing
20:27:48 - cmdstanpy - INFO - Chain [1] start processing
20:27:54 - cmdstanpy - INFO - Chain [1] done processing
20:27:54 - cmdstanpy - INFO - Chain [1] start processing
20:27:59 - cmdstanpy - INFO - Chain [1] done processing
20:27:59 - cmdstanpy - INFO - Chain [1] start processing
20:28:05 - cmdstanpy - INFO - Chain [1] done processing
```

In [11]:
```
np.mean(maes)
```

Out[11]:
```
1617.3899999999996
```

Con este modelo ya podemos añadir la información de pronóstico como un nuevo campo en nuestra serie en InfluxDB. Para ello seguiremos la siguiente estrategia:

- Nos centraremos solo en el último més de datos disponible. Para el último día, que posiblemente no estará completo, deberemos añadir las horas que falten.
- Generaremos un pronóstico para cada día usando el modelo que creamos anteriormente.
- Subiremos este pronóstico directamente a InfluxDB

Implementaremos este proceso por partes a continuación.

Tarea 3

Implementa la siguiente función que, dada una fecha, devuelve una lista con los timestamps ordenados de sus horas

In [12]:
```python
def get_hours(date: datetime.datetime) -> list :
    start = datetime.datetime(date.year, date.month, date.day)
    end = start + pd.Timedelta(days=1)
    hours = pd.date_range(start=start, end=end, freq='H')
    return [hour for hour in hours if hour.date() == date.date()]
```

In [13]:
```python
get_hours(datetime.datetime(2023, 5, 1))
```

Out[13]:
```
[Timestamp('2023-05-01 00:00:00', freq='H'),
 Timestamp('2023-05-01 01:00:00', freq='H'),
 Timestamp('2023-05-01 02:00:00', freq='H'),
 Timestamp('2023-05-01 03:00:00', freq='H'),
 Timestamp('2023-05-01 04:00:00', freq='H'),
 Timestamp('2023-05-01 05:00:00', freq='H'),
 Timestamp('2023-05-01 06:00:00', freq='H'),
 Timestamp('2023-05-01 07:00:00', freq='H'),
 Timestamp('2023-05-01 08:00:00', freq='H'),
 Timestamp('2023-05-01 09:00:00', freq='H'),
 Timestamp('2023-05-01 10:00:00', freq='H'),
 Timestamp('2023-05-01 11:00:00', freq='H'),
 Timestamp('2023-05-01 12:00:00', freq='H'),
 Timestamp('2023-05-01 13:00:00', freq='H'),
 Timestamp('2023-05-01 14:00:00', freq='H'),
 Timestamp('2023-05-01 15:00:00', freq='H'),
 Timestamp('2023-05-01 16:00:00', freq='H'),
 Timestamp('2023-05-01 17:00:00', freq='H'),
 Timestamp('2023-05-01 18:00:00', freq='H'),
 Timestamp('2023-05-01 19:00:00', freq='H'),
 Timestamp('2023-05-01 20:00:00', freq='H'),
 Timestamp('2023-05-01 21:00:00', freq='H'),
 Timestamp('2023-05-01 22:00:00', freq='H'),
 Timestamp('2023-05-01 23:00:00', freq='H')]
```

Resutado esperado:

```
[Timestamp('2023-02-28 00:00:00', freq='H'),
 Timestamp('2023-02-28 01:00:00', freq='H'),
 Timestamp('2023-02-28 02:00:00', freq='H'),
 Timestamp('2023-02-28 03:00:00', freq='H'),
 Timestamp('2023-02-28 04:00:00', freq='H'),
 Timestamp('2023-02-28 05:00:00', freq='H'),
 Timestamp('2023-02-28 06:00:00', freq='H'),
 Timestamp('2023-02-28 07:00:00', freq='H'),
 Timestamp('2023-02-28 08:00:00', freq='H'),
 Timestamp('2023-02-28 09:00:00', freq='H'),
 Timestamp('2023-02-28 10:00:00', freq='H'),
 Timestamp('2023-02-28 11:00:00', freq='H'),
```

Para ingestar el pronóstico en InfluxDB, podemos usar la siguiente función, similar a la función que utilizásteis en la primera práctica y que básicamente toma el pronóstico del modelo de Prophet y lo ingesta usando la API de escritura de InfluxDB. El pronóstico se ingestará en la misma medida que la demanda, pero con el campo `CP7Forecast` .

In [14]:
```python
def save_to_influxdb(df: pd.DataFrame, client: InfluxDBClient, bucket: str, org: str) -> pd.DataFrame:
    """
    Escribe el pronóstico en InfluxDB
    """
    df = df.reset_index()
    df['time'] = df['datetime']
    df['CP7Forecast'] = df['yhat']
    to_write = df[['time', 'CP7Forecast']]
    to_write = to_write.set_index('time')[['CP7Forecast']]
    write_api = client.write_api(write_options=SYNCHRONOUS)
    write_api.write(bucket, org, record=to_write, data_frame_measurement_name='demand')
```

Con estas funciones ya tenemos todo lo necesario para poder escribir en InfluxDB los datos de pronóstico. Para ello, partiendo de un día inicial desde el que empezaremos a generar pronósticos, usaremos una estrategia similar al método de evaluación anterior, simplemente teniendo en cuenta que generaremos nosotros las horas en vez de usar las que aparecen en un conjunto de test. A continuación mostramos como ingestar los resultados a partir de un día determinado. Podéis ejecutar el código a continuación para añadir el pronóstico a vuestra instancia de InfluxDB.

In [15]:
```
begin = datetime.datetime(2023, 5, 1)
dates_to_ingest = pd.date_range(start=begin, end=datetime.datetime.now().date(), freq='D')
dates_to_ingest
```

Out[15]:
```
DatetimeIndex(['2023-05-01', '2023-05-02', '2023-05-03', '2023-05-04',
               '2023-05-05', '2023-05-06', '2023-05-07', '2023-05-08',
               '2023-05-09', '2023-05-10', '2023-05-11', '2023-05-12',
               '2023-05-13', '2023-05-14', '2023-05-15', '2023-05-16',
               '2023-05-17', '2023-05-18', '2023-05-19', '2023-05-20',
               '2023-05-21', '2023-05-22', '2023-05-23', '2023-05-24',
               '2023-05-25', '2023-05-26', '2023-05-27'],
              dtype='datetime64[ns]', freq='D')
```

In [16]:
```
for dt in dates_to_ingest:
    hours = get_hours(dt)
    print(f"Ingesting for date: {dt}")
    training = df[:dt-datetime.timedelta(minutes=1)]
    result = train_forecast(df, hours)
    save_to_influxdb(result, client, bucket, org)
```

```
Ingesting for date: 2023-05-01 00:00:00

20:28:06 - cmdstanpy - INFO - Chain [1] start processing
20:28:11 - cmdstanpy - INFO - Chain [1] done processing

Ingesting for date: 2023-05-02 00:00:00

20:28:12 - cmdstanpy - INFO - Chain [1] start processing
20:28:16 - cmdstanpy - INFO - Chain [1] done processing

Ingesting for date: 2023-05-03 00:00:00

20:28:17 - cmdstanpy - INFO - Chain [1] start processing
20:28:22 - cmdstanpy - INFO - Chain [1] done processing

Ingesting for date: 2023-05-04 00:00:00

20:28:23 - cmdstanpy - INFO - Chain [1] start processing
20:28:28 - cmdstanpy - INFO - Chain [1] done processing

Ingesting for date: 2023-05-05 00:00:00
```

Cuando acabe el proceso anterior ya deberíais tener los resultados en InfluxDB. Solo quedaría adaptar el dashboard anterior para visualizar también los datos de pronóstico.

Tarea 4

Añade a la celda tipo graph del dashboard que creaste en la primera parte del capstone la serie predicha por el modelo que hemos entrenado.

Para visualizar el pronóstico a futuro, necesitarás modificar el filtro de tiempo para incluir el día en curso.

Añade una captura de tu dashboard a continuación (incluyéndo el dashboard con la entrega).

Mi resultado es:

Resultado esperado (aprox):


This dashboard doesn't have any cells with defined variables. Learn How
Demand vs Forecast
34k
32k
30k
28k
26k
24k
22k
20k
2023-02-20 01:00:00
2023-02-21 01:00:00
2023-02-22 01:00:00
2023-02-23 01:00:00
2023-02-24 01:00:00
2023-02-25 01:00:00
2023-02-26 01:00:00
2023-02-27 01:00:00
Demand table
time
ForecastDemand
RealDemand
2023-02-26 04:14:52 GMT+1 20.308,58 20.277,58
2023-02-26 05:18:52 GMT+1 20.367,50 20.349,92
2023-02-20 03:10:53 GMT+1 20.590,58 20.517,75
2023-02-20 04:14:53 GMT+1 20.665,92 20.635,67
2023-02-26 03:10:52 GMT+1 20.690,33 20.648
2023-02-26 06:22:52 GMT+1 20.817,58 20.875,17
2023-02-20 02:06:53 GMT+1 20.928,17 20.911,50
2023-02-26 02:06:52 GMT+1 21.385,58 21.230
2023-02-26 07:26:52 GMT+1 21.408,17 21.397,42
2023-02-20 05:18:53 GMT+1 21.486,67 21.460,58
Forecast - Demand
-20,00
20,00
-60,00
60,00