databricks Gold Table Ceation (current version)

(https://databricks.com)

```
# Initialize Database
databaseName = "group4_finalproject"
print("databaseName: " + databaseName)
spark.sql(f"use {databaseName}")
```

```
databaseName: group4_finalproject
```

```
DataFrame[]
```

Load silver tables into dataframes

```
train_silver_df = spark.table("train_silver")
client_silver_df = spark.table("client_silver")
electricity_prices_silver_df = spark.table("electricity_prices_silver")
gas_prices_silver_df = spark.table("gas_prices_silver")
historical_weather_silver_df = spark.table("historical_weather_silver")
forecast_weather_silver_df = spark.table("forecast_weather_silver")
```

Joins to create wide table for ML

Perform left join of train_silver_df with client_silver_df on "product_type", "county", "is_business", "data_block_id" columns

```
wide_silver_df = train_silver_df.join(client_silver_df, (train_silver_df.product_type == client_silver_df.client_product_t
client_silver_df.client_is_business) &(train_silver_df.data_block_id == client_silver_df.client_data_block_id), "left")
```

```
display(wide_silver_df.count())
```

Given that Client had no data for the first 2 data_block_ids, those rows have null values for client info and are dropped

```
display(wide_silver_df.filter("data_block_id = 0 OR data_block_id = 1"))
```

Table

	county	is_business	product_type	target	is_consumption	datetime	data_block_id
1	0	0	1	1.687	0	2021-09-02T00:00:00Z	1
2	0	0	1	109.366	1	2021-09-02T00:00:00Z	1
3	0	0	2	0	0	2021-09-02T00:00:00Z	1
4	0	0	2	21.008	1	2021-09-02T00:00:00Z	1
5	0	0	3	1.003	0	2021-09-02T00:00:00Z	1
6	0	0	3	735.696	1	2021-09-02T00:00:00Z	1
7	0	1	0	0	0	2021-09-02T00:00:00Z	1

5,760 rows

```
wide_silver_df = wide_silver_df.filter(train_silver_df["data_block_id"] !=
0).filter((train_silver_df["data_block_id"] != 1))
```

There are still some null values, owing to the fact that not all time periods in the original client table have data for all of the 68 distinct product_type, county, is_business combinations (see for example data_block_id 2 having 61). The 2784 nulls are dropped.

```
for column in wide_silver_df.columns:
  print(f"Column {column} has {wide_silver_df.filter(f'{column} is NULL').count()} nulls")
```

```
display(client_bronze_df.filter("data_block_id = 2"))
```

```
train_silver_df.select("prediction_unit_id").distinct().count()
```

```
wide_silver_df = wide_silver_df.dropna(how="any")
```

```
display(wide_silver_df.count())
```

Joins wide_silver_df with electricity_prices on wide_silver_df.datetime == electricity_prices_silver_df.electricity_available_datetime and provides a count to show the number rows is still the same and show columns

```
wide_silver_df = wide_silver_df.join(electricity_prices_silver_df, wide_silver_df.datetime ==
electricity_prices_silver_df.electricity_available_datetime, "left")
```

```
display(wide_silver_df.count())
```

```
1978664
```

```
wide_silver_df.columns
```

```
['county',
 'is_business',
 'product_type',
 'target',
 'is_consumption',
 'datetime',
 'data_block_id',
 'row_id',
 'prediction_unit_id',
 'client_product_type',
 'client_county',
 'eic_count',
 'installed_capacity',
 'client_is_business',
 'client_date',
 'client_data_block_id',
 'electricity_effective_datetime',
 'electricity_euros_per_mwh',
```

```
'electricity_origin_date',
'electricity_data_block_id',
```

Check that the 2 data_block_id columns are identical, drop the one that came from electricity_prices

```
wide_silver_df.filter(wide_silver_df["data_block_id"] != wide_silver_df["electricity_data_block_id"]).count()
```

```
0
```

```
from pyspark.sql.functions import col
```

```
wide_silver_df = wide_silver_df.drop(col("electricity_data_block_id"))
```

Small number of null values, from incomplete electricity price information (2 dates have only 23 predictions while the rest have 24). Nulls are dropped.

```
for column in wide_silver_df.columns:
  print(f"Column {column} has {wide_silver_df.filter(f'{column} is NULL').count()} nulls")
```

```
Column county has 0 nulls
Column is_business has 0 nulls
Column product_type has 0 nulls
Column target has 0 nulls
Column is_consumption has 0 nulls
Column datetime has 0 nulls
Column data_block_id has 0 nulls
Column row_id has 0 nulls
Column prediction_unit_id has 0 nulls
Column client_product_type has 0 nulls
Column client_county has 0 nulls
Column eic_count has 0 nulls
Column installed_capacity has 0 nulls
Column client_is_business has 0 nulls
Column client_date has 0 nulls
Column client_data_block_id has 0 nulls
Column electricity_effective_datetime has 262 nulls
Column electricity_euros_per_mwh has 262 nulls
Column electricity_origin_date has 262 nulls
Column electricity_available_datetime has 262 nulls
```

```
wide_silver_df = wide_silver_df.dropna(how="any")
```

```
wide_silver_df.count()
```

Join wide_silver with gas_prices on data_block_id, then drop the duplicative gas_data_block_id

```
wide_silver_df = wide_silver_df.join(gas_prices_silver_df, wide_silver_df.data_block_id ==
gas_prices_silver_df.gas_data_block_id, "left")
```

```
wide_silver_df = wide_silver_df.drop(col("gas_data_block_id"))
```

Show there are no nulls in the wide dataframe

```
if (wide_silver_df.dropna().count() == wide_silver_df.count()) == True:
  print("No nulls")
else:
  print("Nulls present, need to investigate")
```

```
No nulls
```

```
display(wide_silver_df.count())
```

```
wide_silver_df.columns
```

Average historical weather data for the same county and day/time to be able to join with wide table efficiently

```
historical_weather_silver_df.columns
```

```
['latitude',
 'longitude',
 'datetime',
 'temperature',
 'dewpoint',
 'rain',
 'snowfall',
 'surface_pressure',
 'cloudcover_total',
 'cloudcover_low',
 'cloudcover_mid',
 'cloudcover_high',
 'windspeed_10m',
 'winddirection_10m',
 'shortwave_radiation',
 'direct_solar_radiation',
 'diffuse_radiation',
 'data_block_id',
 'county_name',
 'county',
 'historical_weather_available_datetime']
```

```
historical_weather_silver_df.createOrReplaceTempView("temp_historical_weather")

query = """
SELECT datetime AS historical_weather_datetime, historical_weather_available_datetime,
       data_block_id AS historical_weather_data_block_id, county_name AS historical_weather_county_name, county AS
historical_weather_county, avg(temperature) AS historical_temperature,
       avg(dewpoint) AS historical_dewpoint, avg(rain) AS historical_rain, avg(snowfall) AS historical_snowfall,
avg(surface_pressure) AS historical_surface_pressure, avg(cloudcover_high) AS historical_cloudcover_high,
avg(cloudcover_mid) AS historical_cloudcover_mid, avg(cloudcover_low) AS historical_cloudcover_low,
avg(cloudcover_total) AS historical_cloudcover_total,
       avg(windspeed_10m) AS historical_windspeed_10m, avg(winddirection_10m) AS historical_winddirection_10m,
       avg(shortwave_radiation) AS historical_shortwave_radiation,
       avg(direct_solar_radiation) AS historical_direct_solar_radiation,
       avg(diffuse_radiation) AS historical_diffuse_radiation
  FROM temp_historical_weather
 GROUP BY historical_weather_data_block_id, historical_weather_datetime, historical_weather_available_datetime,
historical_weather_county, historical_weather_county_name"""

historical_weather_silver_df = spark.sql(query)
```

```
historical_weather_silver_df.display()
```

Table

	historical_weather_datetime	historical_weather_available_datetime	historical_weather_data_block_id	historical_weather
1	2022-07-13T02:00:00Z	2022-07-14T02:00:00Z	316	Ida-Virumaa
2	2022-07-13T07:00:00Z	2022-07-14T07:00:00Z	316	Võrumaa
3	2022-07-13T18:00:00Z	2022-07-15T18:00:00Z	317	Pärnumaa
4	2022-07-04T18:00:00Z	2022-07-06T18:00:00Z	308	Pärnumaa
5	2022-07-05T03:00:00Z	2022-07-06T03:00:00Z	308	Ida-Virumaa
6	2022-07-05T10:00:00Z	2022-07-06T10:00:00Z	308	Järvamaa
7	2022-07-11T08:00:00Z	2022-07-12T08:00:00Z	314	Jõgevamaa

8,878 rows | Truncated data

```
historical_weather_silver_df.count()
```

229125

```
if (historical_weather_silver_df.dropna().count() == historical_weather_silver_df.count()) == True:
  print("No nulls")
else:
  print("Nulls present, need to investigate")
```

No nulls

Join wide_silver_df with historical_weather_silver_df on data_block_id == historical_weather_block id and county == historical_weather_county and datetime == historical_weather_available_datetime

```
wide_silver_df = wide_silver_df.join(historical_weather_silver_df, (wide_silver_df.data_block_id ==
historical_weather_silver_df.historical_weather_data_block_id) & (wide_silver_df.county ==
historical_weather_silver_df.historical_weather_county) & (wide_silver_df.datetime ==
historical_weather_silver_df.historical_weather_available_datetime), "left")
```

```
wide_silver_df.count()
```

Check joined table for any nulls

```
if (wide_silver_df.dropna().count() == wide_silver_df.count()) == True:
  print("No nulls")
else:
  print("Nulls present, need to investigate")
```

```
No nulls
```

Drop forecasts which are less than 24 hours of origin_datetime and more than 47 hours

```
forecast_weather_silver_df = forecast_weather_silver_df.filter('hours_ahead > 23 and hours_ahead <48')
forecast_weather_silver_df.orderBy('origin_datetime', 'hours_ahead').display()
```

Table

	latitude	longitude	origin_datetime	hours_ahead	temperature	dewpoint	cl
1	57.6	21.7	2021-09-01T00:00:00Z	24	13.850854492187523	7.572045898437523	0
2	57.6	22.2	2021-09-01T00:00:00Z	24	12.025048828125023	5.299829101562523	0
3	57.6	22.7	2021-09-01T00:00:00Z	24	13.106103515625023	6.868920898437523	0
4	57.6	23.2	2021-09-01T00:00:00Z	24	13.237939453125023	7.634545898437523	0
5	57.6	23.7	2021-09-01T00:00:00Z	24	13.522729492187523	7.276147460937523	0
6	57.6	24.2	2021-09-01T00:00:00Z	24	13.632714843750023	6.841821289062523	0
7	57.6	24.7	2021-09-01T00:00:00Z	24	9.237084960937523	5.237817382812523	0

7,844 rows | Truncated data

```
forecast_weather_silver_df.count()
```

```
forecast_weather_silver_df[['hours_ahead']].distinct().orderBy('hours_ahead').display()
```

Average forecast weather data for the same county and day/time to be able to join with wide table efficiently

```
forecast_weather_silver_df.columns
```

```
['latitude',
 'longitude',
 'origin_datetime',
 'hours_ahead',
 'temperature',
 'dewpoint',
 'cloudcover_high',
 'cloudcover_low',
 'cloudcover_mid',
 'cloudcover_total',
 '10_metre_u_wind_component',
 '10_metre_v_wind_component',
 'data_block_id',
```

```
'forecast_datetime',
'direct_solar_radiation',
'surface_solar_radiation_downwards',
'snowfall',
'total_precipitation',
'county_name',
'county']
```

```
forecast_weather_silver_df.createOrReplaceTempView("temp_forecast_weather")
query = """
SELECT origin_datetime AS forecast_weather_origin_datetime, hours_ahead AS forecast_weather_hours_ahead,
       forecast_datetime AS forecast_weather_forecast_datetime,
       data_block_id AS forecast_weather_data_block_id, county_name AS forecast_weather_county_name, county AS
forecast_weather_county, avg(temperature) AS forecast_weather_temperature,
       avg(dewpoint) AS forecast_weather_dewpoint,  avg(snowfall) AS forecast_weather_snowfall, avg(cloudcover_high)
AS forecast_weather_cloudcover_high, avg(cloudcover_mid) AS forecast_weather_cloudcover_mid, avg(cloudcover_low) AS
forecast_weather_cloudcover_low, avg(cloudcover_total) AS forecast_weather_cloudcover_total,
       avg(direct_solar_radiation) AS forecast_weather_direct_solar_radiation,
       avg(surface_solar_radiation_downwards) AS forecast_weather_surface_solar_radiation_downwards
  FROM temp_forecast_weather
  GROUP BY forecast_weather_data_block_id, forecast_weather_origin_datetime, forecast_weather_county,
forecast_weather_county_name, forecast_weather_hours_ahead, forecast_weather_forecast_datetime"""

forecast_weather_silver_df = spark.sql(query)
```

```
forecast_weather_silver_df.count()
```

```
229320
```

Join wide_silver_df with forecast_weather_silver_df on data_block_id == forecast_weather_block id and county == forecast_weather_county and datetime == forecast_weather_forecast_datetime

```
wide_silver_df = wide_silver_df.join(forecast_weather_silver_df, (wide_silver_df.data_block_id ==
forecast_weather_silver_df.forecast_weather_data_block_id) & (wide_silver_df.county ==
forecast_weather_silver_df.forecast_weather_county) & (wide_silver_df.datetime ==
forecast_weather_silver_df.forecast_weather_forecast_datetime), "left")
```

Check for null values

```
if (wide_silver_df.dropna().count() == wide_silver_df.count()) == True:
  print("No nulls")
else:
  print("Nulls present, need to investigate")
```

```
No nulls
```

Check joined table count

```
wide_silver_df.count()
```

```
1978402
```

Drop unnecessary columns

```
wide_silver_df.columns
```

```
['county',
 'is_business',
 'product_type',
 'target',
 'is_consumption',
 'datetime',
 'data_block_id',
 'row_id',
 'prediction_unit_id',
 'client_product_type',
 'client_county',
 'eic_count',
 'installed_capacity',
 'client_is_business',
 'client_date',
 'client_data_block_id',
 'electricity_effective_datetime',
 'electricity_euros_per_mwh',
 'electricity_origin_date',
 'electricity_available_datetime',
 'gas_effective_date',
```

```
drop_columns = ['client_is_business', 'client_product_type', 'client_county','client_date',
 'client_data_block_id',
'electricity_effective_datetime','electricity_origin_date','electricity_available_datetime', 'gas_origin_date',
'gas_effective_date',
 'gas_available_date','historical_weather_data_block_id', 'historical_weather_county_name',
 'historical_weather_county', 'forecast_weather_data_block_id','historical_weather_available_datetime',
'historical_weather_datetime','forecast_weather_county_name', 'forecast_weather_county',
'forecast_weather_origin_datetime','forecast_weather_forecast_datetime',
 'forecast_weather_hours_ahead', 'electricity_available_datetime']
```

```
print(len(wide_silver_df.columns))
wide_silver_df = wide_silver_df.drop(*drop_columns)
print(len(wide_silver_df.columns))
```

```
58
37
```

Convert datetime to a double, as requested by data scientists

```
wide_silver_df.printSchema()
```

```
root
 |-- county: integer (nullable = true)
 |-- is_business: integer (nullable = true)
 |-- product_type: integer (nullable = true)
 |-- target: double (nullable = true)
 |-- is_consumption: integer (nullable = true)
 |-- datetime: timestamp (nullable = true)
 |-- data_block_id: integer (nullable = true)
 |-- row_id: integer (nullable = true)
 |-- prediction_unit_id: integer (nullable = true)
 |-- eic_count: integer (nullable = true)
 |-- installed_capacity: double (nullable = true)
```

```
 |-- electricity_euros_per_mwh: double (nullable = true)
 |-- gas_lowest_price_per_mwh: double (nullable = true)
 |-- gas_highest_price_per_mwh: double (nullable = true)
 |-- historical_temperature: double (nullable = true)
 |-- historical_dewpoint: double (nullable = true)
 |-- historical_rain: double (nullable = true)
 |-- historical_snowfall: double (nullable = true)
 |-- historical_surface_pressure: double (nullable = true)
```

```
from pyspark.sql.functions import unix_timestamp, cast
wide_silver_df = wide_silver_df.withColumn('datetime', unix_timestamp('datetime').cast('double'))
```

```
wide_silver_df.printSchema()
```

```
root
 |-- county: integer (nullable = true)
 |-- is_business: integer (nullable = true)
 |-- product_type: integer (nullable = true)
 |-- target: double (nullable = true)
 |-- is_consumption: integer (nullable = true)
 |-- datetime: double (nullable = true)
 |-- data_block_id: integer (nullable = true)
 |-- row_id: integer (nullable = true)
 |-- prediction_unit_id: integer (nullable = true)
 |-- eic_count: integer (nullable = true)
 |-- installed_capacity: double (nullable = true)
 |-- electricity_euros_per_mwh: double (nullable = true)
 |-- gas_lowest_price_per_mwh: double (nullable = true)
 |-- gas_highest_price_per_mwh: double (nullable = true)
 |-- historical_temperature: double (nullable = true)
 |-- historical_dewpoint: double (nullable = true)
 |-- historical_rain: double (nullable = true)
 |-- historical_snowfall: double (nullable = true)
 |-- historical_surface_pressure: double (nullable = true)
 |-- historical_cloudcover_high: double (nullable = true)
```

Check if there are any null values

```
if (wide_silver_df.dropna().count() == wide_silver_df.count()) == True:
  print("No nulls")
else:
  print("Nulls present, need to investigate")
```

```
No nulls
```

Get final count of wide table

```
wide_silver_df.count()
```

```
1978402
```

Save wide_silver_df as enefit_gold_ML_table using upserts and merges

```
wide_silver_df.createOrReplaceTempView("temp_table")
merge_columns = ["datetime", "county", "product_type", "is_business", "is_consumption"]


non_merge_columns = [col for col in wide_silver_df.columns if col not in merge_columns]


update_conditions = " OR ".join([f"destination.{col} <> source.{col}" for col in non_merge_columns])
update_set = ", ".join([f"destination.{col} = source.{col}" for col in non_merge_columns])


on_clause = " AND ".join([f"destination.{col} = source.{col}" for col in merge_columns])


merge_sql = f"""
MERGE INTO enefit_gold_ML_table AS destination
     USING temp_table AS source
        ON {on_clause}

      WHEN MATCHED AND ({update_conditions}) THEN
           UPDATE SET {update_set}
      WHEN NOT MATCHED BY TARGET
           THEN INSERT *
      WHEN NOT MATCHED BY SOURCE THEN
           DELETE

"""


if not spark._jsparkSession.catalog().tableExists("enefit_gold_ML_table"):
   wide_silver_df.write.format("delta").option("mergeSchema",
"true").partitionBy("data_block_id").saveAsTable("enefit_gold_ML_table")
else:
   spark.sql(merge_sql)
```

Create Gold Table for BI

```
enefit_gold_ML_df = spark.readStream.format("delta").table("enefit_gold_ML_table")
```

First, create a year column from datetime column (for aggregation)

```
from pyspark.sql.functions import from_unixtime,year
enefit_gold_ML_df = enefit_gold_ML_df.withColumn('year', year(from_unixtime('datetime').cast('timestamp')))
enefit_gold_ML_df.display()
```

display_query_1 (id: 97eb4a68-0606-4c09-a8f8-144fd2735d3f) *Last updated: 262 days ago*

Table

	county	is_business	product_type	target	is_consumption	datetime	data_block_id	row_id
1	0	0	1	0	0	1648612800	210	644648
2	0	0	1	516.315	1	1648612800	210	644649
3	0	0	2	0	0	1648612800	210	644650
4	0	0	2	43.371	1	1648612800	210	644651

5	0	0	3	0.541	0	1648612800	210	644652
6	0	0	3	1688.682	1	1648612800	210	644653
7	0	1	0	0	0	1648612800	210	644654

1,000 rows | Truncated data

```
# create a view to do sql operations
enefit_gold_ML_df.createOrReplaceTempView("enefit_Gold_ML_streaming_view")
```

Perform aggregations on the gold ML dataframe, then save it to gold BI table

Below aggragation query aggregates the data by county, is_business, product_type, is_consumption, prediction_unit_id, year then takes sum of the target column and average of the rest of the columns

```
query = '''SELECT  county AS county_id, is_business, product_type, is_consumption, prediction_unit_id, year,
ROUND(AVG(eic_count),2) AS avg_eic_count, ROUND(SUM(target),2) AS sum_target, ROUND(AVG(installed_capacity),2) AS
avg_installed_capacity, ROUND(AVG(electricity_euros_per_mwh),2) AS avg_electricity_euros_per_mwh,
ROUND(AVG(gas_lowest_price_per_mwh),2) AS avg_gas_lowest_price_per_mwh, ROUND(AVG(gas_highest_price_per_mwh),2) AS
avg_gas_highest_price_per_mwh, ROUND(AVG(historical_temperature),2) AS avg_historical_temp, ROUND(AVG(
historical_dewpoint),2) AS avg_historical_dewpoint, ROUND(AVG(historical_rain),2) AS avg_historical_rain,
ROUND(AVG(historical_snowfall),2) AS avg_historical_snowball, ROUND(AVG(historical_surface_pressure),2) AS
avg_historical_surface_pressure, ROUND(AVG(
historical_cloudcover_high),2) AS historical_cloudcover_high, ROUND(AVG(
historical_cloudcover_mid),2) AS historical_cloudcover_mid,  ROUND(AVG(historical_cloudcover_low),2) AS
historical_cloudcover_low, ROUND(AVG(historical_cloudcover_total),2) AS avg_historical_cloudcover_total,
ROUND(AVG(historical_windspeed_10m),2) AS avg_historical_windspeed_10m, ROUND(AVG(historical_winddirection_10m),2)
AS avg_historical_windirection_10m, ROUND(AVG(historical_shortwave_radiation),2) AS
avg_historical_shortwave_radiation, ROUND(AVG(historical_direct_solar_radiation),2) AS
avg_historical_direct_solar_radiation, ROUND(AVG(historical_diffuse_radiation),2) AS
avg_historical_diffuse_radiation, ROUND(AVG(forecast_weather_temperature),2) AS avg_historical_weather_temperature,
ROUND(AVG(forecast_weather_dewpoint),2) AS avg_forecast_weather_dewpoint, ROUND(AVG(forecast_weather_snowfall),2) AS
avg_forecast_weather_snowfall, ROUND(AVG(forecast_weather_cloudcover_high),2) AS
avg_forecast_weather_cloudcover_high, ROUND(AVG(forecast_weather_cloudcover_mid),2) AS
avg_forecast_weather_cloudcover_mid, ROUND(AVG(forecast_weather_cloudcover_low),2) AS
avg_forecast_weather_cloudcover_low, ROUND(AVG(forecast_weather_cloudcover_total),2) AS
avg_forecast_weather_cloudcover_total,
ROUND(AVG(forecast_weather_direct_solar_radiation),2) AS avg_forecast_weather_direct_solar_radiation,
ROUND(AVG(forecast_weather_surface_solar_radiation_downwards),2) AS
avg_forecast_weather_surface_solar_radiation_downwards FROM enefit_gold_ML_streaming_view GROUP BY county,
is_business, product_type, is_consumption, prediction_unit_id, year'''
```

```
enefit_gold_BI_aggregated_df = spark.sql(query)
```

```
%sql
drop table if exists enefit_gold_BI_table
```

OK

```
# clean the checkpoint
dbutils.fs.rm('/tmp/checkpoint/enefit_gold_BI_table/', True)
```

```
False
```

```
checkpoint_loc="/tmp/checkpoint/enefit_gold_BI_table/"
gold_table_name = "enefit_gold_BI_table"

enefit_gold_BI_aggregated_df.writeStream\
  .format("delta")\
  .option("checkpointLocation", checkpoint_loc)\
  .option("mergeSchema", True)\
  .outputMode("complete")\
  .trigger(once=True)\
  .table(gold_table_name)
```

0adcc745-0583-479c-9a51-4b9d9aa3b64e *Last updated: 262 days ago*

```
<pyspark.sql.streaming.query.StreamingQuery at 0x7f04b47e9c30>
```

Table

	county_id	is_business	product_type	is_consumption	prediction_unit_id	year	avg_eic_count	sun
1	0	1	3	1	5	2021	268.91	210
2	15	1	3	0	60	2021	47.33	138
3	3	0	1	0	11	2021	17.73	167
4	10	1	1	1	40	2021	9	183
5	11	0	2	1	44	2021	8.4	298
6	2	0	3	0	9	2021	33.84	414
7	5	0	1	0	19	2021	25.98	256

400 rows