databricks 01 etl

(https://databricks.com)

```
%run ./includes/includes
```

```
Out[3]: DataFrame[]
```

VERY IMPORTANT TO UNDERSTAND THE USE OF THESE VARIABLES! Please ask if you are confused about their use.

Variable Name	Value	Description
NYC_WEATHER_FILE_PATH	dbfs:/FileStore/tables/raw/weather/	Historic NYC Weather for Model Building
BIKE_TRIP_DATA_PATH	dbfs:/FileStore/tables/raw/bike_trips/	Historic Bike Trip Data for Model Building (Stream this data source)
BRONZE_STATION_INFO_PATH	dbfs:/FileStore/tables/bronze_station_info.delta	Station Information (30 min refresh)
BRONZE_STATION_STATUS_PATH	dbfs:/FileStore/tables/bronze_station_status.delta	Station Status (30 min refresh)
BRONZE_NYC_WEATHER_PATH	dbfs:/FileStore/tables/bronze_nyc_weather.delta	NYC Weather (30 min refresh)
USER_NAME	jtschopp@u.rochester.edu	Email of the user executing this code/notebook
GROUP_NAME	G11	Group Assigment for this user
GROUP_STATION_ASSIGNMENT	Cleveland Pl & Spring St	Station Name to be modeled by this group
GROUP_DATA_PATH	dbfs:/FileStore/tables/G11/	Path to store all of your group data files (delta ect)
GROUP_MODEL_NAME	G11_model	Mlflow Model Name to be used to register your model
GROUP_DB_NAME	G11_db	Group Database to store any managed tables (pre-defined for you)

```
#Did not make use of widgets in this specific notebook
#start_date = str(dbutils.widgets.get('01.start_date'))
#end_date = str(dbutils.widgets.get('02.end_date'))
#hours_to_forecast = int(dbutils.widgets.get('03.hours_to_forecast'))
#promote_model = bool(True if str(dbutils.widgets.get('04.promote_model')).lower() == 'yes' else False)

#print(start_date,end_date,hours_to_forecast, promote_model)
#print("YOUR CODE HERE...")
```

Schema Definition and Stream Initialization for Historic Trip Data

```
#Initializing stream for historic trip data
from pyspark.sql.types import LongType, StringType, StructType, StructField, TimestampType, DoubleType
historic_trip_data_schema = StructType([
    StructField("ride_id", StringType(), False),
    StructField("rideable_type", StringType(), True),
    StructField("started_at", TimestampType(), True),
    StructField("ended_at", TimestampType(), True),
    StructField("start_station_name", StringType(), True),
    StructField("start_station_id", DoubleType(), True),
    StructField("end_station_name", StringType(), True),
    StructField("end_station_id", DoubleType(), True),
    StructField("start_lat", DoubleType(), True),
    StructField("start_lng", DoubleType(), True),
    StructField("end_lat", DoubleType(), True),
    StructField("end_lng", DoubleType(), True),
    StructField("member_casual", StringType(), True)
])
historic_trip_data_df = (spark.readStream
                         .option("header", True)
                         .schema(historic_trip_data_schema)
                         .csv(BIKE_TRIP_DATA_PATH))
#display(historic_trip_data_df)
```

```
#Filters the historic trip data to contain only our assigned station data
from pyspark.sql.functions import col
historic_trip_df = historic_trip_data_df.filter((col("start_station_name") == GROUP_STATION_ASSIGNMENT)
                                    | (col("end_station_name") == GROUP_STATION_ASSIGNMENT))
#display(historic_trip_df)
```

Writing the Historic Trip Data to our Group Data Path

```
#This command writes the stream for the historic trip data in order to read it in the EDA notebook

historic_trip_checkpoint_path = f"dbfs:/FileStore/tables/G11/bronze/historic_trip_data/.checkpoint"
historic_trip_output_path = f"dbfs:/FileStore/tables/G11/bronze/historic_trip_data/"
historic_trip_query = (historic_trip_df.writeStream
                       .outputMode("append")
                       .format("delta")
                       .queryName("historic_trip")
                       .option("checkpointLocation", historic_trip_checkpoint_path)
                       .trigger(availableNow=True)
                       .start(historic_trip_output_path))
```

▸ historic_trip (id: 149e0854-266e-47f4-b15e-ccdd3c038e97) *Last updated: 2 days ago*

```
%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/bronze/historic_trip_data'
ZORDER BY started_at
/*Z ordering by a time stamp more specifically the start time seemed like the most sensible out of all the attributes in
this table*/
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/bronze/historic_trip_data	▸ {"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0, "t "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOptimiz {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputO 10018278}, "inputNumCubes": 0, "mergedFiles": {"num": 0, "size": 0}, "numOutputCubes": 0, "me "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": false "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplificati 1683406573256, "endTimeMs": 1683406578329, "totalClusterParallelism": 4, "totalScheduledTas "autoCompactParallelismStats": null}

1 row

Bronze Station Status read

```
#Reading bronze station status tables
STATION_ID = "66db2fd0-0aca-11e7-82f6-3863bb44ef7c"

bronze_station_status_df = (spark.read
                            .format("delta")
                            .load(BRONZE_STATION_STATUS_PATH))
#bronze_station_status_df.display()


#Filtering to have only our station
bronze_station_status_df = bronze_station_status_df.filter(col("station_id") == STATION_ID)
#bronze_station_status_df.display()
```

```
#Writing bronze station status
bronze_station_status_path = f"dbfs:/FileStore/tables/G11/bronze/station_status/"
bronze_station_status_query = (bronze_station_status_df.write
                               .format("delta")
                               .mode("overwrite")
                               .save(bronze_station_status_path))

#I know here it shows this 'light bulb' saying to optimize the delta table, but it referencing the delta table we are
reading in as it is specified in the path below
#so I assumed we should leave that untouched...


%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/bronze/station_status'
ZORDER BY last_reported
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/bronze/station_status	▸ {"numFilesAdded": 1, "numFilesRemoved": 51, "filesAdded": {"min": 23579, "max": 23579, "avg": 23 "totalSize": 23579}, "filesRemoved": {"min": 6974, "max": 17705, "avg": 7272.549019607844, "totalFile "partitionsOptimized": 0, "zOrderStats": {"strategyName": "minCubeSize(107374182400)", "inputCub "inputOtherFiles": {"num": 51, "size": 370900}, "inputNumCubes": 0, "mergedFiles": {"num": 51, "size" "numOutputCubes": 1, "mergedNumCubes": null}, "numBatches": 1, "totalConsideredFiles": 51, "tota "preserveInsertionOrder": false, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplification": 0, "startTimeMs": 1683406672656, "endTimeMs": 1 "totalClusterParallelism": 20, "totalScheduledTasks": 1, "autoCompactParallelismStats": null}

1 row

Reading Bronze Station Info Table

```
bronze_station_info_df = (spark.read
                          .format("delta")
                          .load(BRONZE_STATION_INFO_PATH))
#bronze_station_info_df.display()


bronze_station_info_df = bronze_station_info_df.filter("short_name == '5492.05'")
#bronze_station_info_df.display()
```

Writing Bronze Station Info Table

```
bronze_station_info_path = f"dbfs:/FileStore/tables/G11/bronze/station_info"
bronze_station_info_query = (bronze_station_info_df.write
                             .format("delta")
                             .mode("overwrite")
                             .save(bronze_station_info_path))


%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/bronze/station_info'
/* no need to zorder since there is only one row in this table */
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/bronze/station_info	▸ {"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0, "totalFil "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOptimized": 0 "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": true, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplification": 0, 1683406748239, "endTimeMs": 1683406752298, "totalClusterParallelism": 40, "totalScheduledTasks": 0 "autoCompactParallelismStats": null}

1 row

Reading Stream of Historic Weather Data and Schema Definition

```
#Read in historic weather
from pyspark.sql.types import LongType, StringType, StructType, StructField, TimestampType, DoubleType, IntegerType
historic_weather_schema = StructType([
    StructField("dt", LongType(), True),
    StructField("temp", DoubleType(), True),
    StructField("feels_like", DoubleType(), True),
    StructField("pressure", IntegerType(), True),
    StructField("humidity", IntegerType(), True),
    StructField("dew_point", DoubleType(), True),
    StructField("uvi", DoubleType(), True),
    StructField("clouds", IntegerType(), True),
    StructField("visibility", IntegerType(), True),
    StructField("wind_speed", DoubleType(), True),
    StructField("wind_deg", IntegerType(), True),
    StructField("pop", DoubleType(), True),
    StructField("snow_1h", DoubleType(), True),
    StructField("id", IntegerType(), True),
    StructField("main", StringType(), True),
    StructField("description", StringType(), True),
    StructField("icon", StringType(), True),
    StructField("loc", StringType(), True),
    StructField("lat", DoubleType(), True),
    StructField("lon", DoubleType(), True),
    StructField("timezone", StringType(), True),
    StructField("timezone_offset", IntegerType(), True),
    StructField("rain_1h", DoubleType(), True)
])
historic_weather_df = (spark.readStream
                       .option("header", True)
                       .schema(historic_weather_schema)
                       .csv(NYC_WEATHER_FILE_PATH))
#historic_weather_df.display()
```

Writing Stream for Historic Weather Data

```
#This command writes the stream for the historic weather data in order to read it in the EDA notebook

historic_weather_data_path = f"dbfs:/FileStore/tables/G11/bronze/historic_weather_data/"
historic_weather_checkpoint_path = f"dbfs:/FileStore/tables/G11/bronze/historic_weather_data/.checkpoint"
historic_weather_query = (historic_weather_df.writeStream
                          .outputMode("append")
                          .format("delta")
                          .queryName("historic_weather")
                          .option("checkpointLocation", historic_weather_checkpoint_path)
                          .trigger(availableNow=True)
                          .start(historic_weather_data_path))
```

▸ historic_weather (id: 393b56ff-43a7-4df2-b2f2-79e5551729c8) *Last updated: 2 days ago*

```
%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/bronze/historic_weather_data'
ZORDER BY dt
/* z ordering by time stamp seemed the most sensible... again */
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/bronze/historic_weather_data	▸ {"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOp {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inp 288640}, "inputNumCubes": 0, "mergedFiles": {"num": 0, "size": 0}, "numOutputCubes": 0, "r "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplif 1683406761612, "endTimeMs": 1683406766071, "totalClusterParallelism": 40, "totalSchedul "autoCompactParallelismStats": null}

1 row

Reading Bronze Weather Table

```
#Read bronze weather table
bronze_nyc_weather_df = (spark.read
                         .format("delta")
                         .load(BRONZE_NYC_WEATHER_PATH))
#bronze_nyc_weather_df.display()


#Fixing some slight issues in the table so that the schema is nice and easy to deal with. In this case it was a column
that had an array so I exploded it and created some
#new columns to have each attribute separately
from pyspark.sql.functions import *
weather_exploded_df = (bronze_nyc_weather_df.withColumn("weather", explode(col("weather"))))
df = (weather_exploded_df.withColumn("description", col("weather.description"))
     .withColumn("icon", col("weather.icon"))
     .withColumn("id", col("weather.id"))
     .withColumn("main", col("weather.main")))
df = df.drop("weather")
#display(df)
```

Writing Bronze Weather Table

```
bronze_weather_path = f"dbfs:/FileStore/tables/G11/bronze/weather"
bronze_station_info_query = (df.write
                             .format("delta")
                             .mode("overwrite")
                             .save(bronze_weather_path))


%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/bronze/weather'
ZORDER BY dt
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/bronze/weather	▸ {"numFilesAdded": 1, "numFilesRemoved": 40, "filesAdded": {"min": 51626, "max": 51626, "avg": 51626, "t "totalSize": 51626}, "filesRemoved": {"min": 6603, "max": 9687, "avg": 7449.875, "totalFiles": 40, "totalSize": "partitionsOptimized": 0, "zOrderStats": {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles" "inputOtherFiles": {"num": 40, "size": 297995}, "inputNumCubes": 0, "mergedFiles": {"num": 40, "size": 2979 "numOutputCubes": 1, "mergedNumCubes": null}, "numBatches": 1, "totalConsideredFiles": 40, "totalFilesS "preserveInsertionOrder": false, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplification": 0, "startTimeMs": 1683406800455, "endTimeMs": 168340 "totalClusterParallelism": 40, "totalScheduledTasks": 1, "autoCompactParallelismStats": null}

1 row

Creating Silver Table for Historic Trip Data

```
#Silver table for historic trip data
historic_trip_silver = (spark.read
    .format("delta")
    .load(historic_trip_output_path))

historic_trip_silver = historic_trip_silver.select(
    'started_at',
    'ended_at',
    'start_station_name',
    'end_station_name'
)
historic_trip_silver = historic_trip_silver.withColumn("started_at", date_format(col("started_at"), "yyyy-MM-dd
HH:mm:ss"))
historic_trip_silver = historic_trip_silver.withColumn("ended_at", date_format(col("started_at"), "yyyy-MM-dd HH:mm:ss"))

silver_historic_trip_path = f"dbfs:/FileStore/tables/G11/silver/historic_trip_data/"
silver_historic_trip_query = (historic_trip_silver.write
    .format("delta")
    .mode("overwrite")
    .save(silver_historic_trip_path))


#historic_trip_silver.display()


%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/silver/historic_trip_data'
ZORDER BY started_at
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/silver/historic_trip_data	▸ {"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0, "tot "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOptimize {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputOth 4329710}, "inputNumCubes": 0, "mergedFiles": {"num": 0, "size": 0}, "numOutputCubes": 0, "merge "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": false, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplificatio 1683406816020, "endTimeMs": 1683406817280, "totalClusterParallelism": 40, "totalScheduledTask "autoCompactParallelismStats": null}

1 row

Creating Silver Table for Historic Weather Data

```
silver_historic_weather = (spark.read
    .format("delta")
    .load(historic_weather_data_path))

silver_historic_weather = (silver_historic_weather.withColumn("dt", date_format(from_unixtime(col("dt").cast("long")),
"yyyy-MM-dd HH:mm:ss"))
    .withColumn("temp", round((col("temp") - 273.15), 2))
    .withColumn("feels_like", round((col("feels_like") - 273.15), 2))
)

silver_historic_weather = silver_historic_weather.select(
    'dt',
    'temp',
    'feels_like',
    'snow_1h',
    'main',
    'rain_1h'
)

silver_historic_weather_path = f"dbfs:/FileStore/tables/G11/silver/historic_weather_data/"
silver_historic_weather_query = (silver_historic_weather.write
    .format("delta")
    .mode("overwrite")
    .save(silver_historic_weather_path))
```

```
%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/silver/historic_weather_data'
ZORDER BY dt
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/silver/historic_weather_data	▸ {"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0 "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOpti {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inpu 139916}, "inputNumCubes": 0, "mergedFiles": {"num": 0, "size": 0}, "numOutputCubes": 0, "me "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": f "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplific 1683406825942, "endTimeMs": 1683406827637, "totalClusterParallelism": 40, "totalSchedule "autoCompactParallelismStats": null}

1 row

```
#silver_historic_weather.display()
```

Creating Silver Table for Station Info

```
silver_station_info = (spark.read
    .format("delta")
    .load(bronze_station_info_path))

silver_station_info = silver_station_info.select(
    'capacity'
)

silver_station_info_path = f"dbfs:/FileStore/tables/G11/silver/station_info/"
silver_station_info_query = (silver_station_info.write
    .format("delta")
    .mode("overwrite")
    .save(silver_station_info_path))
```

```
%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/silver/station_info'
/* no need to z order since it is only one row and one column */
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/silver/station_info	{"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0, "totalFiles "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOptimized": 0, "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": true, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplification": 0, " 1683406832868, "endTimeMs": 1683406835863, "totalClusterParallelism": 40, "totalScheduledTasks": 0, "autoCompactParallelismStats": null}

1 row

```
#silver_station_info.display()
```

Creating Silver Table for Station Status

```
silver_station_status = (spark.read
    .format("delta")
    .load(bronze_station_status_path))

silver_station_status = silver_station_status.withColumn("last_reported",
date_format(from_unixtime(col("last_reported").cast("long")), "yyyy-MM-dd HH:mm:ss"))

silver_station_status = silver_station_status.select(
    'num_bikes_available',
    'num_bikes_disabled',
    'num_docks_available',
    'last_reported',
    'num_docks_disabled',
    'num_ebikes_available'
)

silver_station_status_path = f"dbfs:/FileStore/tables/G11/silver/station_status/"
silver_station_status_query = (silver_station_status.write
    .format("delta")
    .mode("overwrite")
    .save(silver_station_status_path))


%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/silver/station_status'
ZORDER BY last_reported
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/silver/station_status	{"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0, "totalFi "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOptimized": { {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputOtherFi 24535}, "inputNumCubes": 0, "mergedFiles": {"num": 0, "size": 0}, "numOutputCubes": 0, "mergedNum "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": false, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplification": 0 1683406840848, "endTimeMs": 1683406841557, "totalClusterParallelism": 24, "totalScheduledTasks": ("autoCompactParallelismStats": null}

1 row

```
#silver_station_status.display()
```

Creating Silver Table for Weather

```
silver_weather = (spark.read
    .format("delta")
    .load(bronze_weather_path))

silver_weather = (silver_weather.withColumn("dt", date_format(from_unixtime(col("dt").cast("long")), "yyyy-MM-dd
HH:mm:ss"))
    .withColumn("temp", round((col("temp") - 273.15), 2))
    .withColumn("feels_like", round((col("feels_like") - 273.15), 2))
    .withColumnRenamed("rain.1h", "rain_1h")
)

silver_weather = silver_weather.select(
    'dt',
    'temp',
    'feels_like',
    'main',
    'rain_1h'
)
silver_weather = silver_weather.fillna(value=0)

silver_weather_path = f"dbfs:/FileStore/tables/G11/silver/weather/"
silver_weather_query = (silver_weather.write
.format("delta")
.mode("overwrite")
.save(silver_weather_path))
```

```
%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/silver/weather'
ZORDER BY dt
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/silver/weather	▸ {"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOptimized": 0, "zOr {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputOtherFiles": {" 18165}, "inputNumCubes": 0, "mergedFiles": {"num": 0, "size": 0}, "numOutputCubes": 0, "mergedNumCubes "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": false, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplification": 0, "start 1683406846811, "endTimeMs": 1683406847860, "totalClusterParallelism": 24, "totalScheduledTasks": 0, "autoCompactParallelismStats": null}

1 row

Creating a Silver Table for Historic Data to Get an Hourly Net Inventory Change

```
from pyspark.sql import Window
import pyspark.sql.functions as f
weather = (spark.read
    .format("delta")
    .load(silver_historic_weather_path))

trips = (spark.read
    .format("delta")
    .load(silver_historic_trip_path))


#round the start time to the nearest hour in order to join with weather df
mod_trips = trips.withColumn("unix", (round(unix_timestamp("started_at")/3600)*3600).cast("timestamp"))
trips = mod_trips.withColumn("rounded_started_at", date_format(from_unixtime(col("unix").cast("long")), "yyyy-MM-dd
HH:mm:ss"))



joined_df = trips.join(weather, trips.rounded_started_at == weather.dt, "inner")
joined_df = joined_df.drop("unix")

#round the end time to the nearest hour
joined_df = joined_df.withColumn("unix", (round(unix_timestamp("ended_at")/3600)*3600).cast("timestamp"))
joined_df = joined_df.withColumn("rounded_ended_at", date_format(from_unixtime(col("unix").cast("long")), "yyyy-MM-dd
HH:mm:ss"))
joined_df = joined_df.drop("unix")


#creating a data frame to count how many bikes are arriving to the station every hour
df1 = joined_df.filter(joined_df.end_station_name == GROUP_STATION_ASSIGNMENT)
df1 = df1.withColumnRenamed("rounded_ended_at", "end")
df1 = df1.groupBy("end").count()
df1 = df1.withColumnRenamed("count", "hour_increase")
df1 = df1.join(weather, weather.dt == df1.end, "outer")

#creating a dataframe to count how many bikes are leaving every hour
df2 = joined_df.filter(joined_df.start_station_name == GROUP_STATION_ASSIGNMENT)
df2 = df2.withColumnRenamed("rounded_started_at", "start")
df2 = df2.groupBy("start").count()
df2 = df2.withColumnRenamed("count", "hour_decrease")
df2 = df2.join(weather, weather.dt == df2.start, "outer")
df2 = df2.withColumnRenamed("dt", "date")
df2 = df2.drop("feels_like", "main", "snow_1h", "temp", "rain_1h")

#joining the dataframes
inventory = df1.join(df2, df1.dt == df2.date, "inner")

inventory = inventory.fillna(0)

inventory = inventory.withColumn("net_hour_change", (f.col("hour_increase") - f.col("hour_decrease")))

inventory = inventory.drop("start")
inventory = inventory.drop("end")



#older tests not sure if still needed
#inventory = invetory.join(weather, weather.dt == inventory)


#joined_df = joined_df.join(df1, joined_df.rounded_ended_at == df1.end, "inner")
#joined_df = joined_df.join(df2, joined_df.rounded_ended_at == df2.start, "inner")

#joined_df = joined_df.fillna(value=0)

#joined_df = joined_df.withColumn("net_hour_change", (f.col("hour_increase") - f.col("hour_decrease")))

#joined_df = joined_df.drop("start")
```

```
#joined_df = joined_df.drop("end")
```

```
#inventory.display()
```

```
#joined_df.display()
```

```
#old tests not sure if still needed, ignore for now
#joined_path = f"dbfs:/FileStore/tables/G11/silver/joined/"
#joined_query = (joined_df.write
#    .format("delta")
#    .mode("overwrite")
#    .save(joined_path))
```

Writing the new silver table with net inventory change

```
inventory = inventory.select(
    'dt',
    'temp',
    'feels_like',
    'snow_1h',
    'main',
    'rain_1h',
    'net_hour_change'
)

inventory_path = f"dbfs:/FileStore/tables/G11/silver/inventory/"
query = (inventory.write
    .format("delta")
    .mode("overwrite")
    .save(inventory_path))
```

```
%sql
OPTIMIZE 'dbfs:/FileStore/tables/G11/silver/inventory'
ZORDER BY dt
```

Table

	path	metrics
1	dbfs:/FileStore/tables/G11/silver/inventory	▸ {"numFilesAdded": 0, "numFilesRemoved": 0, "filesAdded": {"min": null, "max": null, "avg": 0, "totalFiles": ("filesRemoved": {"min": null, "max": null, "avg": 0, "totalFiles": 0, "totalSize": 0}, "partitionsOptimized": 0, "zO {"strategyName": "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputOtherFiles": 141696}, "inputNumCubes": 0, "mergedFiles": {"num": 0, "size": 0}, "numOutputCubes": 0, "mergedNumCu "numBatches": 0, "totalConsideredFiles": 1, "totalFilesSkipped": 1, "preserveInsertionOrder": false, "numFilesSkippedToReduceWriteAmplification": 0, "numBytesSkippedToReduceWriteAmplification": 0, "sta 1683406860297, "endTimeMs": 1683406861588, "totalClusterParallelism": 24, "totalScheduledTasks": 0, "autoCompactParallelismStats": null}

1 row

```
Notebook exited: {"exit_code": "OK"}
```