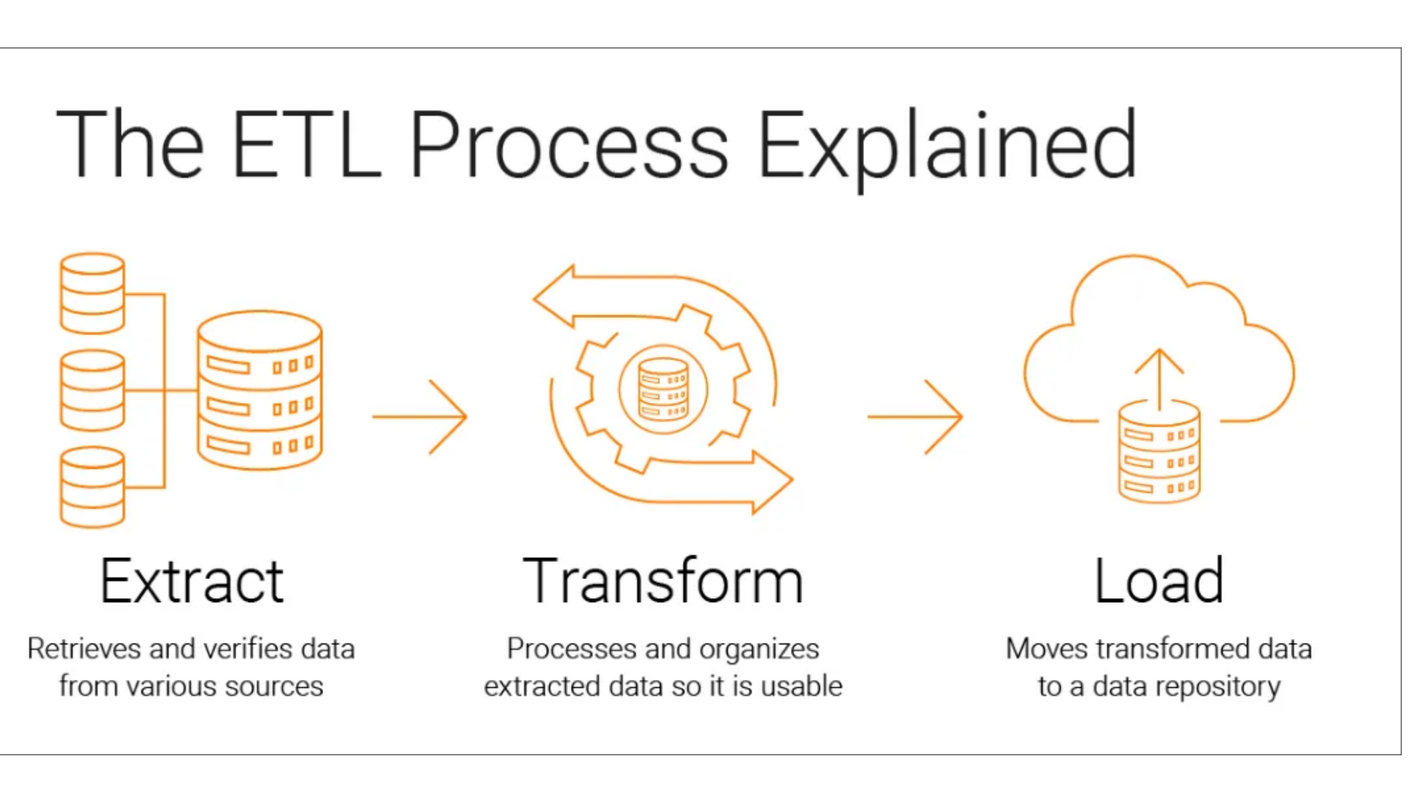
Data Engineering — Building an ETL Pipeline for NYC Taxi Trip Data

Sahil Sharma · Follow
4 min read · Apr 11, 2023



Every month, the New York City Taxi and Limousine Commission (TLC) publishes a dataset of taxi trips in New York City. Details such as pickup and dropoff times and locations, fare amount, and payment type are included in the dataset.

The TLC makes the data available in parquet format, with one file per month.

You are tasked as a data engineer with creating an ETL pipeline to extract data from monthly parquet files, transform it to a consistent format, and load it into Snowflake for analysis and reporting.

Solution: To solve this issue, we can construct our ETL pipeline using Python and a number of third-party libraries.

Importing libraries

Installing the snowflake library.

```
pip install snowflake-connector-python
```

```
# Importing libraries
import pandas as pd
import os
import snowflake.connector
import pyarrow.parquet as pq
```

Extracting the Data

The data can be downloaded from — TLC Trip Record Data — TLC (nyc.gov)

We will take 3 months of data from Jan 2023 till Nov 2022 in parquet format and combine it into a single pandas dataframe.

```
# Parquet files
parquet_files = ['yellow_tripdata_2023-01.parquet',
                 'yellow_tripdata_2022-12.parquet',
                 'yellow_tripdata_2022-11.parquet']

# Extracting data into dataframes
data_frames = []

for file in parquet_files:
    df = pq.read_table(file).to_pandas()
    data_frames.append(df)

all_data = pd.concat(data_frames, ignore_index=True)
```

Transforming the Data

Now let's transform the data,

1. we will use dropna() to remove the rows with missing data.
2. Update the source columns with proper names.
3. Update the datetime fields with proper formats.
4. Create the required derived fields like, `trip_duration` and `trip_duration_minutes`.

```
# Transform the data
all_data = all_data.dropna()  # Remove rows with missing data
all_data = all_data.rename(columns={'tpep_pickup_datetime': 'pickup_datetime', '
all_data['pickup_datetime'] = pd.to_datetime(all_data['pickup_datetime'])
all_data['dropoff_datetime'] = pd.to_datetime(all_data['dropoff_datetime'])
all_data['trip_duration'] = all_data['dropoff_datetime'] - all_data['pickup_date
all_data['trip_duration_minutes'] = all_data['trip_duration'].dt.total_seconds()
```

Loading the Data

Now let's load the data in Snowflake, first we will create the connection with Snowflake after that we will create the ddl structure for the table and then load the data into the table `taxi_trips` in this case.

Link for Snowflake account_name criteria.

```
# Snowflake connection parameters
'''
Account names varies based on different region and cloud,
use the link above to choose the right one.
'''
snowflake_account = 'account_name'
snowflake_user = 'user_name'
snowflake_password = 'password'
snowflake_warehouse = 'warehouse_name'
snowflake_database = 'database_name'  # Specify the desired database name
snowflake_schema = 'schema_name'      # Specify the desired Schema name

# Create a Snowflake connection
conn = snowflake.connector.connect(
    user=snowflake_user,
    password=snowflake_password,
    account=snowflake_account,
    warehouse=snowflake_warehouse

# Create a cursor
cur = conn.cursor()

# Create the database if it doesn't exist
cur.execute(f"CREATE DATABASE IF NOT EXISTS {snowflake_database}")

# Switch to the specified database
cur.execute(f"USE DATABASE {snowflake_database}")

# Create schema if it doesn't exist
cur.execute(f"CREATE SCHEMA IF NOT EXISTS {snowflake_schema}")

# Use the specified schema
cur.execute(f"USE SCHEMA {snowflake_schema}")

# Create the table if it doesn't exist
cur.execute("""
    CREATE OR REPLACE TABLE IF NOT EXISTS taxi_trips (
        pickup_datetime TIMESTAMP,
        dropoff_datetime TIMESTAMP,
        passenger_count INTEGER,
        trip_distance FLOAT,
        rate_code_id INTEGER,
        store_and_fwd_flag STRING,
        payment_type INTEGER,
        fare_amount FLOAT,
        extra FLOAT,
        mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT,
        trip_duration INTERVAL MINUTE
    )
""")

# Load data from 'all_data' DataFrame into the Snowflake table 'taxi_trips'
cur.execute("TRUNCATE TABLE taxi_trips")  # Clear existing data if needed
cur.execute("BEGIN")  # Begin a transaction

# Use the 'copy_into' method from the DataFrame to load data into Snowflake
all_data.to_sql('taxi_trips', conn, if_exists='replace', index=False, schema=sno

# Commit the transaction
cur.execute("COMMIT")

# Close the cursor and connection
cur.close()
conn.close()
```

The `pyarrow.parquet` module is used in this implementation to read data from Parquet files into a Pandas DataFrame. In addition, we've changed the `CREATE TABLE` statement to match the schema of the data we're loading. Finally, we use the `to_sql()` function in Snowflake to write the transformed data to a table called `taxi_trips`.

Note that to use the PyArrow library, you will need to install the `pyarrow` package. You can install it via pip:

```
pip install pyarrow
```

Find the complete code below,

```
# Importing libraries
import pandas as pd
import os
import snowflake.connector
import pyarrow.parquet as pq

# Extract data from Parquet files
parquet_files = ['yellow_tripdata_2023-01.parquet',
                 'yellow_tripdata_2022-12.parquet',
                 'yellow_tripdata_2022-11.parquet']

data_frames = []

for file in parquet_files:
    df = pq.read_table(file).to_pandas()
    data_frames.append(df)

all_data = pd.concat(data_frames, ignore_index=True)

# Transform the data
all_data = all_data.dropna()  # Remove rows with missing data
all_data = all_data.rename(columns={'tpep_pickup_datetime': 'pickup_datetime', '
all_data['pickup_datetime'] = pd.to_datetime(all_data['pickup_datetime'])
all_data['dropoff_datetime'] = pd.to_datetime(all_data['dropoff_datetime'])
all_data['trip_duration'] = all_data['dropoff_datetime'] - all_data['pickup_date
all_data['trip_duration_minutes'] = all_data['trip_duration'].dt.total_seconds()

# Snowflake connection parameters
'''
Account names varies based on different region and cloud,
use the link above to choose the right one.
'''
snowflake_account = 'account_name'
snowflake_user = 'user_name'
snowflake_password = 'password'
snowflake_warehouse = 'warehouse_name'
snowflake_database = 'database_name'  # Specify the desired database name
snowflake_schema = 'schema_name'      # Specify the desired Schema name

# Create a Snowflake connection
conn = snowflake.connector.connect(
    user=snowflake_user,
    password=snowflake_password,
    account=snowflake_account,
    warehouse=snowflake_warehouse

# Create a cursor
cur = conn.cursor()

# Create the database if it doesn't exist
cur.execute(f"CREATE DATABASE IF NOT EXISTS {snowflake_database}")

# Switch to the specified database
cur.execute(f"USE DATABASE {snowflake_database}")

# Create schema if it doesn't exist
cur.execute(f"CREATE SCHEMA IF NOT EXISTS {snowflake_schema}")

# Use the specified schema
cur.execute(f"USE SCHEMA {snowflake_schema}")

# Create the table if it doesn't exist
cur.execute("""
    CREATE OR REPLACE TABLE IF NOT EXISTS taxi_trips (
        pickup_datetime TIMESTAMP,
        dropoff_datetime TIMESTAMP,
        passenger_count INTEGER,
        trip_distance FLOAT,
        rate_code_id INTEGER,
        store_and_fwd_flag STRING,
        payment_type INTEGER,
        fare_amount FLOAT,
        extra FLOAT,
        mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT,
        trip_duration INTERVAL MINUTE
    )
""")

# Load data from 'all_data' DataFrame into the Snowflake table 'taxi_trips'
cur.execute("TRUNCATE TABLE taxi_trips")  # Clear existing data if needed
cur.execute("BEGIN")  # Begin a transaction

# Use the 'copy_into' method from the DataFrame to load data into Snowflake
all_data.to_sql('taxi_trips', conn, if_exists='replace', index=False, schema=sno

# Commit the transaction
cur.execute("COMMIT")

# Close the cursor and connection
cur.close()
conn.close()
```

Wrapping Up

Please feel free to post in comments if you have some specific suggestions to be covered in next series or if you feel some of the information is inaccurate.

If you found this post useful, Follow me as I go on my content journey!

Data Engineering · Data Engineering 101 · Etl · Case Study · Data Engineering Project