TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

NGÀNH KHOA HỌC DỮ LIỆU

PYTHON CHO KHOA HỌC DỮ LIỆU

ĐỒ ÁN CUỐI KÌ

TAXI TRIP RECORD ETL-PIPELINE

Giảng viên phụ trách

Thầy Hà Văn Thảo

Thành phố Hồ Chí Minh, tháng 1, năm 2024

MỤC LỤC

NỘI DUNG

1. GIỚI THIỆU

1.1 Thông tin nhóm

- Tên nhóm: Nhóm 7
- Thành viên:
  - 21280115 - Trần Đức Trung
  - 21280113 - Lê Quang Trung
  - 21280012 - Nguyễn Đông Hải
  - 21280018 - Trần Phi Hùng

1.2 Nguyên tắc hoạt động nhóm

- Ý tưởng được đóng góp từ tất cả thành viên của nhóm.
- Mọi người tôn trọng ý kiến của nhau.
- Giao và làm việc đúng thời hạn đưa ra. Nếu trễ thì sẽ chịu trách nhiệm về phần của mình.

1.3 Mục tiêu project

Với mục tiêu tìm hiểu và học tập về data engineering và big data, nhóm đã thực hiện project NYC Taxi Trip Record. Project tập trung vào quá trình ETL từ database (MySQL) đến datawarehouse (Psql), đồng thời tận dụng sức mạnh của Minio, Pyspark và Polars để thực hiện clean và transform. Cuối cùng sử dụng streamlit để visualize và analyze data Trong dự án này, nhóm mình sẽ minh họa rõ ràng và chi tiết các quy trình thực hiện ETL trên tập dữ liệu TLC Trip Record Data - một tập dữ liệu mở, phục vụ cho việc học tập và nghiên cứu.

1.4 Data sources

NYC Taxi Trip Record được chọn lọc từ tập dataset chính thức tại TLC Trip Record Data. Dữ liệu bao gồm:

- Yellow Taxi Trip Records: Đây là điều mọi người nghĩ đến khi nhắc về taxi ở New York. Chiếc taxi mang tính biểu tượng là tiêu chuẩn trong vận chuyển ô tô ở New York. Xe taxi màu vàng là phương tiện duy nhất được phép đón khách ở mọi nơi trong thành phố.

- Green Taxi Trip Records: Xe xanh có thể thả bạn đi bất cứ đâu nhưng chỉ được phép đón khách ở những khu vực nhất định.
- For Hire Vehicle Trip Records (FHV Trip Records): Là những taxi chủ yếu được sử dụng cho chuyến đi được sắp xếp trước.

Dữ liệu được sử dụng trong project:

- 1 tháng (January 2023) của TLC Trip Record Data

Link website: `https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`

Với data dictionary như sau:

`https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf`

`https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_green.pdf`

`https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_fhv.pdf`

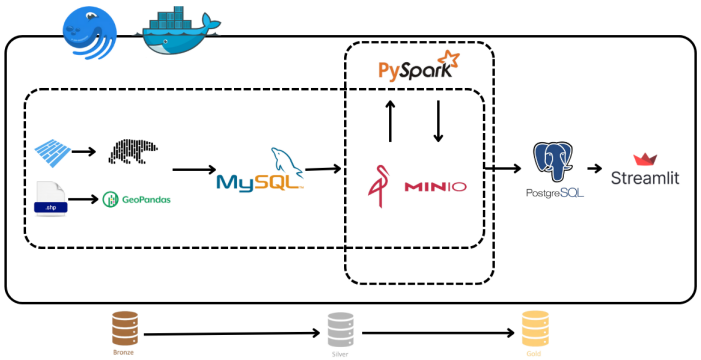
- Taxi Zone Shapefile: `https://d37ci6vzurychx.cloudfront.net/misc/taxi_zones.zip`

2. ARCHITECTURE

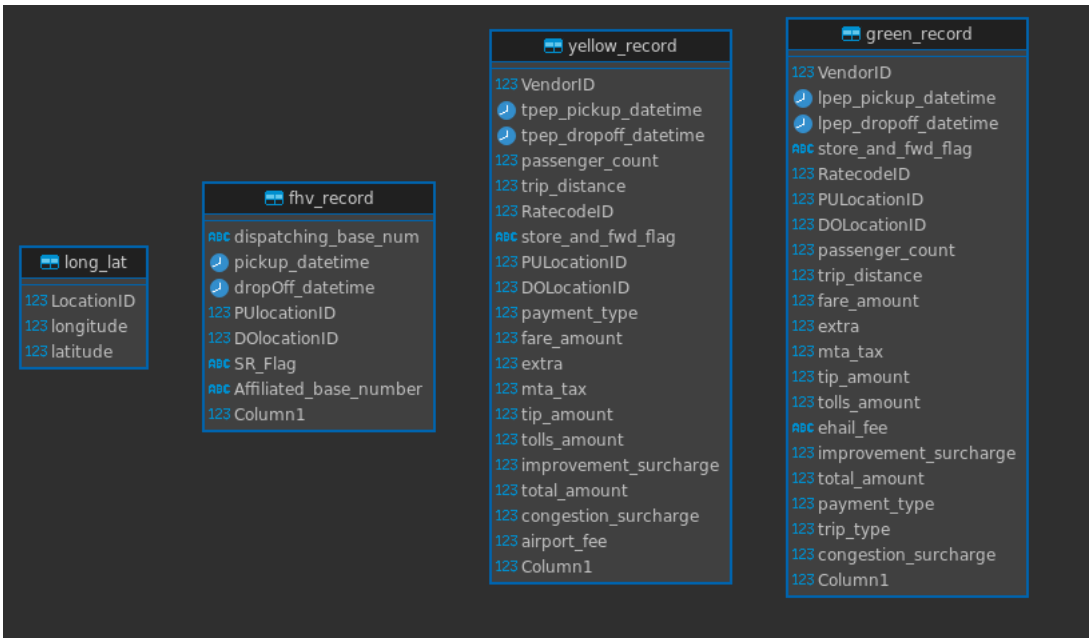
2.1 Dictionary tree

Chi tiết:

- **app**: The UI's application written with **streamlit**
- **dagster_home**: Dagit and dagster daemon's configurations
- **dockerimages**: self-built docker images, such as dagster (for dagit + daemon), spark master,...
- **etl_pipeline**: pipeline
- **load_dataset**: include files have .sql to create schema and load data to MySQL, Postgres
- **minio**: Docker container for MinIO
- **mysql**: Docker container for MySQL
- **postgresql**: Docker container for Psql
- **Test**: folder to test and EDA
- **.gitignore + .gitattributes**: Code versioning
- **docker-compose**: to compose docker containers
- **env**: Env variables.
- **makefile**: simplify terminal's commands
- **requirements.txt**: required library.



2.2 Database



Sau khi thực hiện load data vào database ở đây là MySQL ta được các table có như trên với:

- yellow_record, green_record, fhv_record: có data dictionary như đã mô tả
- long_lat: là 1 table chứa kinh độ, vĩ độ, tương ứng với các LocationID

Một vài xử lí nhỏ khi đọc file **taxizones.shp** để load các data về kinh độ, vĩ độ đưa vào trong database (MySQL). Sử dụng thư viện **GeoPandas** and **PyProj** và chuyển đổi từ **shapefile format** thành một **DataFrame**

```Python
shapefile_path = f"{adr_data}/{zone_data}/{zone_data}.shp"

gdf = gpd.read_file(shapefile_path)
# Define
source_crs = gdf.crs  # CRS of the shapefile
target_crs = 'EPSG:4326'  # WGS84 - lat/lon CRS

# Create a PyProj transformer
transformer = pyproj.Transformer.from_crs(source_crs, target_crs, always_xy=True)

gdf['longitude'] = gdf.geometry.centroid.x
gdf['latitude'] = gdf.geometry.centroid.y
gdf['longitude'], gdf['latitude'] = transformer.transform(gdf['longitude'], gdf['latitude'])

df = pl.DataFrame(gdf[['LocationID', 'longitude', 'latitude']])
```

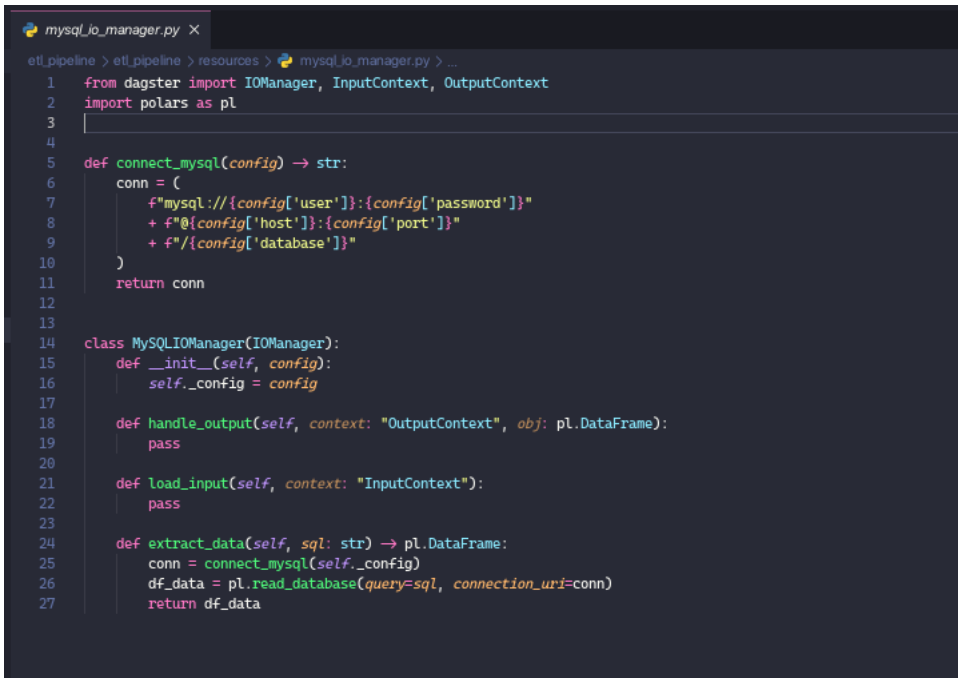
2.3 ETL PIPELINE

2.3.1 resources

Là folder chứa các input output manager

- **mysql_io_manager.py**

  Là 1 file python chứa **class MySQLIOManager**, với hàm **extract_data** làm nhiệm vụ kết nối vào MySQL đọc và trả ra 1 polars dataframe



- **minio_io_manager.py**

  Là 1 file python chứa **class MinIOIOManager**, với các hàm chính như:

  - **load_input** thực hiện lấy data từ MinIO đứa vào asset để xử lý

  - **handle_output** thực hiện chuyển data sau khi đã được xử lý trong asset và đứa lên MinIO

minio_io_manager.py

```python
from typing import Union
import polars as pl
from dagster import IOManager, OutputContext, InputContext
from minio import Minio
import os
import pyarrow.parquet as pq


def make_bucket(client: Minio, bucket_name):
    found = client.bucket_exists(bucket_name)
    if not found:
        client.make_bucket(bucket_name)
    else:
        print(f"Bucket {bucket_name} already exists.")

class MinIOIOManager(IOManager):
    def __init__(self, config):
        self._config = config
        self.minio_client = Minio(
            self._config["endpoint_url"],
            access_key=self._config["aws_access_key_id"],
            secret_key=self._config["aws_secret_access_key"],
            secure=False
        )

    def _get_path(self, context: Union[OutputContext, InputContext]):
        layer, schema, table = context.asset_key.path
        key = "/".join([layer, schema, table.replace(f"{layer}_", "")])
        tmp_file_path = "/tmp/file_{}.parquet".format(
            "_".join(context.asset_key.path),
        )
        if context.has_partition_key:
            partition_str = str(table) + "_" + context.asset_partition_key
            return os.path.join(key, f"{partition_str}.parquet"), tmp_file_path
        else:
            return f"{key}.parquet", tmp_file_path
```

minio_io_manager.py

```python
            return f"{key}.parquet", tmp_file_path


    def handle_output(self, context: OutputContext, obj: pl.DataFrame):
        key_name, tmp_file_path = self._get_path(context)
        table = obj.to_arrow()
        pq.write_table(table, tmp_file_path)

        try:
            bucket_name = self._config["bucket"]
            make_bucket(self.minio_client, bucket_name)
            self.minio_client.fput_object(
                bucket_name, key_name, tmp_file_path,
            )
            context.log.info(
                f"(MinIO handle_output) Number of rows and columns: {obj.shape}"
            )
            # Clean up tmp file
            os.remove(tmp_file_path)
        except Exception as e:
            raise e

    def load_input(self, context: InputContext) -> pl.DataFrame:
        bucket_name = self._config["bucket"]
        key_name, tmp_file_path = self._get_path(context)
        try:
            context.log.info(f"(MinIO load_input) from key_name is {key_name}")
            self.minio_client.fget_object(
                bucket_name, key_name, tmp_file_path,
            )
            df = pl.read_parquet(tmp_file_path)
            os.remove(tmp_file_path)
            return df
        except Exception as e:
            raise e
```

- **spark_io_manager.py**

  Là 1 file python chứa **class SparkIOManager** như sau:

spark_io_manager.py M

etl_pipeline > etl_pipeline > resources > spark_io_manager.py > get_spark_session

```python
from dagster import IOManager, OutputContext, InputContext
from pyspark.sql import SparkSession, DataFrame
from contextlib import contextmanager



@contextmanager
def get_spark_session(config, run_id="Spark IO Manager"):
    executor_memory = "1g" if run_id != "Spark IO Manager" else "1500m"
    try:
        spark = (
            SparkSession.builder.master("spark://spark-master:7077")
            # SparkSession.builder.master("local[*]")
            .appName(run_id)
            .config("spark.driver.memory", "1g")
            .config("spark.executor.memory", executor_memory)
            .config("spark.cores.max", "4")
            .config("spark.executor.cores", "2")
            .config(
                "spark.jars",
                "/usr/local/spark/jars/delta-core_2.12-2.2.0.jar,/usr/local/spark/jars/hadoop-aws-3.3.2.jar,/usr/local/spark/jars/delta-storage-2.2.0.jar,/usr/l
            )
            .config("spark.driver.extraClassPath", "/usr/local/spark/jars/postgresql-42.7.1.jar")
            .config("spark.executor.extraClassPath", "/usr/local/spark/jars/postgresql-42.7.1.jar")
            .config(
                "spark.sql.catalog.spark_catalog",
                "org.apache.spark.sql.delta.catalog.DeltaCatalog",
            )
            .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")
            .config("spark.hadoop.fs.s3a.endpoint", f"http://{config['endpoint_url']}")
            .config("spark.hadoop.fs.s3a.access.key", str(config["minio_access_key"]))
            .config("spark.hadoop.fs.s3a.secret.key", str(config["minio_secret_key"]))
            .config("spark.hadoop.fs.s3a.path.style.access", "true")
            .config("spark.hadoop.fs.connection.ssl.enabled", "false")
            .config(
                "spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem"
            )
            .config("spark.sql.execution.arrow.pyspark.enabled", "true")
            .config("spark.sql.execution.arrow.pyspark.fallback.enabled", "true")
            .getOrCreate()
        )
        yield spark
```

- Hình trên là về **hàm get_spark_session**, hàm sẽ thực hiện tạo 1 spark session và config các thông tin về core, memory, cũng như các extension của Spark mà sẽ được sử dụng (ví dụ như ext pgJDBC không có sẵn trong spark, để kết nối được với psql)

spark_io_manager.py M

etl_pipeline > etl_pipeline > resources > spark_io_manager.py > SparkIOManager > load_input

```python

class SparkIOManager(IOManager):
    def __init__(self, config):
        self._config = config

    def handle_output(self, context: OutputContext, obj: DataFrame):
        # Write output to s3a (MinIO)
        context.log.debug("(Spark handle_output) Writing output to MinIO ...")

        file_path = "s3a://lakehouse/" + "/".join(context.asset_key.path)
        if context.has_partition_key:
            file_path += f"/{context.partition_key}"
        context.log.debug(f"(Spark handle_output) File path: {file_path}")
        file_name = str(context.asset_key.path[-1])
        context.log.debug(f"(Spark handle_output) File name: {file_name}")

        try:
            obj.write.mode("overwrite").parquet(file_path)
            context.log.debug(f"Saved {file_name} to {file_path}")
        except Exception as e:
            raise Exception(f"(Spark handle_output) Error while writing output: {e}")

    def load_input(self, context: InputContext) -> DataFrame:
        context.log.debug(f"Loading input from {context.asset_key.path} ...")
        file_path = "s3a://lakehouse/" + "/".join(context.asset_key.path)
        check_partition = (context.metadata or {}).get("partition", True)
        if check_partition == True:
            if context.has_partition_key:
                file_path += f"/{context.partition_key}"
        full_load = (context.metadata or {}).get("full_load", False)

        try:
            with get_spark_session(self._config) as spark:
                df = None
                if full_load:
                    df = (
                        spark.read.format("parquet")
                        .options(header=True, inferSchema=False)
                        .load(file_path + "/*")
                    )
                else:
                    df = spark.read.parquet(file_path)

                return df
        except Exception as e:
            raise Exception(f"Error while loading input: {e}")
```

- Kế tới là các hàm **load_input** và **handle_output** để đứa data lên s3a (đã được setting là MinIO) hoặc lấy data về asset

- **psql_io_manager.py** - Cuối cùng là file python chứa **class PSQLIOManager**, chứa **hàm handle_output** sẽ sử dụng spark và ext pgJDBC của nó để kết nối spark session với psql và load data vào psql

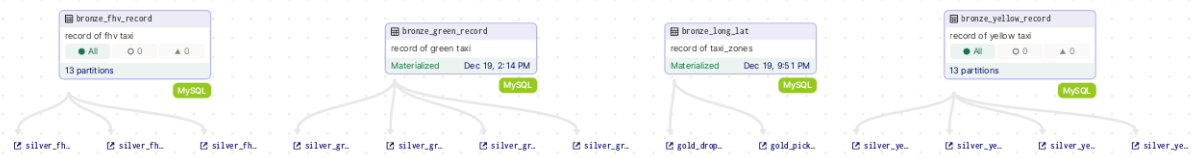
psql_io_manager.py

etl_pipeline > etl_pipeline > resources > psql_io_manager.py > ...

```python
from dagster import IOManager, InputContext, OutputContext
from pyspark.sql import DataFrame
import polars as pl


def connect_psql(config, table):
    conn = {
        "url": f"jdbc:postgresql://{config['host']}:{config['port']}/{config['database']}",
        "dbtable": table,
        "user": config["user"],
        "password": config["password"],
    }
    return conn

class PostgreSQLIOManager(IOManager):
    def __init__(self, config):
        self._config = config

    def handle_output(self, context: OutputContext, obj: DataFrame):
        table = context.asset_key.path[-1]
        schema_ = context.asset_key.path[-2]
        full_table = f"{schema_}.{table}"
        conn = connect_psql(self._config, full_table)
        context.log.info(f"Now use pgjdbc load to postgres")
        obj.write \
            .mode("overwrite") \
            .format("jdbc") \
            .option("url", conn['url'] ) \
            .option("dbtable", conn['dbtable']) \
            .option("user", conn['user']) \
            .option("password", conn['password']) \
            .save()

    def load_input(self, context: InputContext) -> pl.DataFrame:
        pass
```

2.3.2 assets

- **bronze_layer**



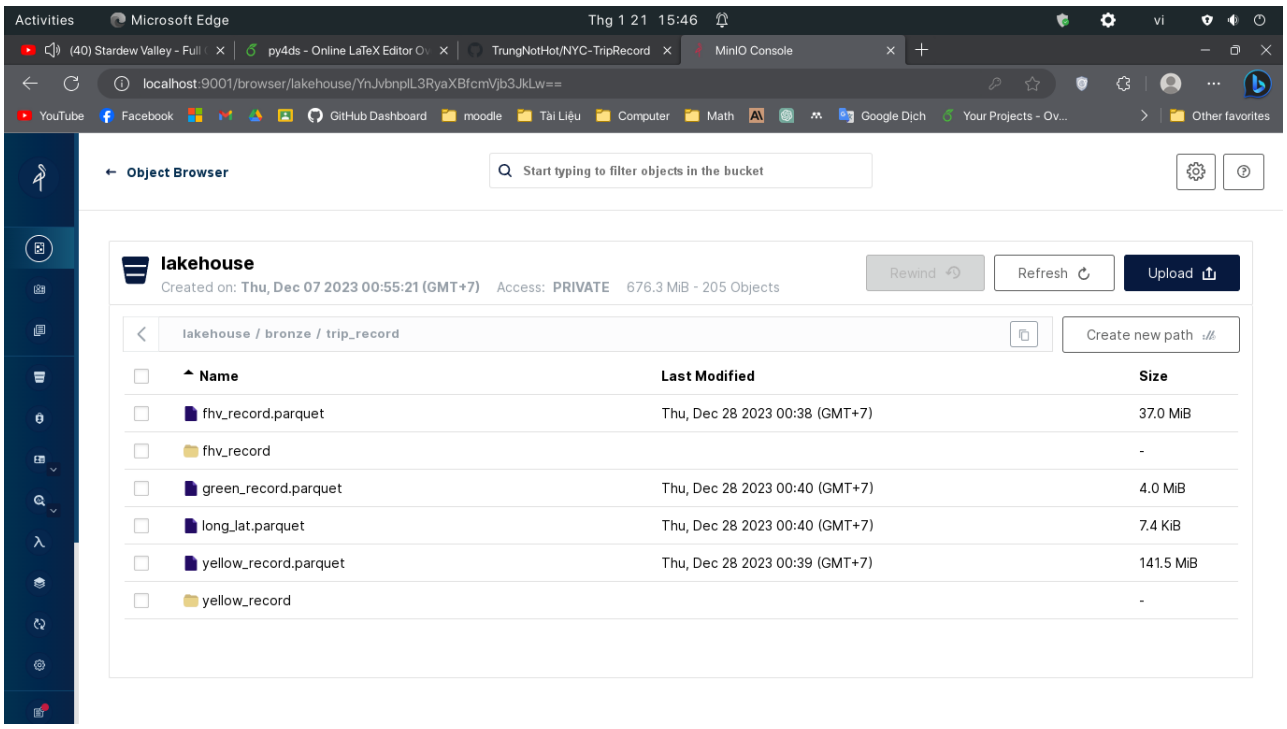
Tại các asset trên layer này, ta sẽ sử dụng mysql_io_manager làm **required resource keys** và minio_io_manager là **io manager key** để thực hiện đọc data trên database (MySQL) và đưa vào folder bronze_layer trên lakehouse (MinIO)

Bao gồm các assets:

- bronze_yellow_record: là table yellow_record từ MySQL, bởi vì data quá lớn với số lượng dòng khoảng 3.5 triệu dòng, nên asset này sẽ được thực hiện partition ra theo tuần
- bronze_green_record: là table green_record từ MySQL, chỉ khoảng 65000 dòng
- bronze_fhv_record: là table fhv_record từ MySQL, khoảng 1000000 dòng, nên cũng sẽ parition tương tự bronze_yellow_record
- bronze_long_lat: chứa data về các kinh độ vĩ độ ở khu vực các taxi có thể đến

```python

@asset(
    name="bronze_yellow_record",
    description="record of yellow taxi",
    io_manager_key="minio_io_manager",
    required_resource_keys={"mysql_io_manager"},
    key_prefix=["bronze", "trip_record"],
    compute_kind="MySQL",
    group_name="bronze",
    partitions_def=WEEKLY,
)
def bronze_yellow_record(context) -> Output[pl.DataFrame]:
    query = "SELECT * FROM yellow_record"
    try:
        partition = context.asset_partition_key_for_output()
        partition_by = "tpep_pickup_datetime"
        query += f" WHERE DATE({partition_by}) ≥ '{partition}' AND DATE({partition_by}) < DATE_ADD('{partition}', INTERVAL 1 WEEK);"
        context.log.info(f"Partition by {partition_by}: {partition} to 1 week later")
    except Exception:
        context.log.info("No partition key found")
    df_data = context.resources.mysql_io_manager.extract_data(query)
    context.log.info(f"Table extracted with shape: {df_data.shape}")

    return Output(
        df_data,
        metadata={
            "table": "yellow_record",
            "row_count": df_data.shape[0],
            "column_count": df_data.shape[1],
            "columns": str(df_data.columns),
        },
    )
```

Picture 2.1 1 asset ví dụ trong bronze



Picture 2.2 folder bronze trong minio sau khi chạy xong

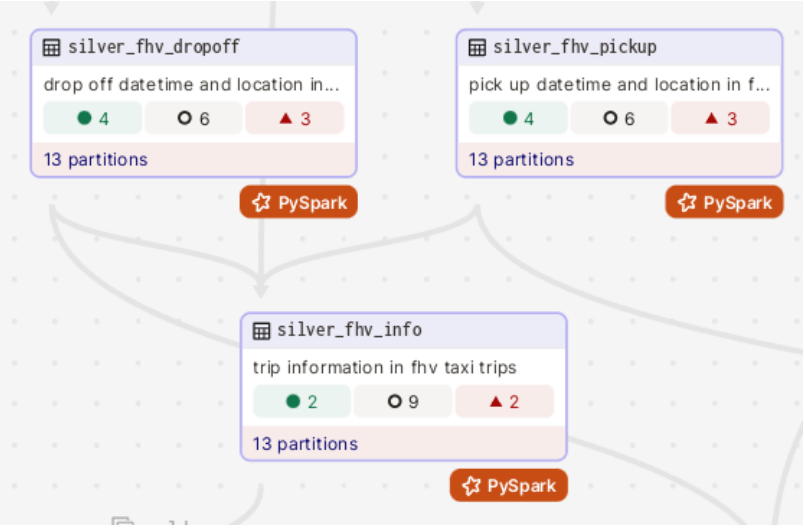
- **silver_layer**

  Silver FHV: dùng **spark_io_manager** làm **io manager key**, tiếp đó thực hiện data cleaning và transform từ upstream asset là bronze_fhv_record, tại các assets này thực hiện tách bronze_fhv_record ra thành các asset:

  – silver_fhv_dropoff: chứa các thông tin về thời gian và địa điểm khách xuống xe của các fhv taxi

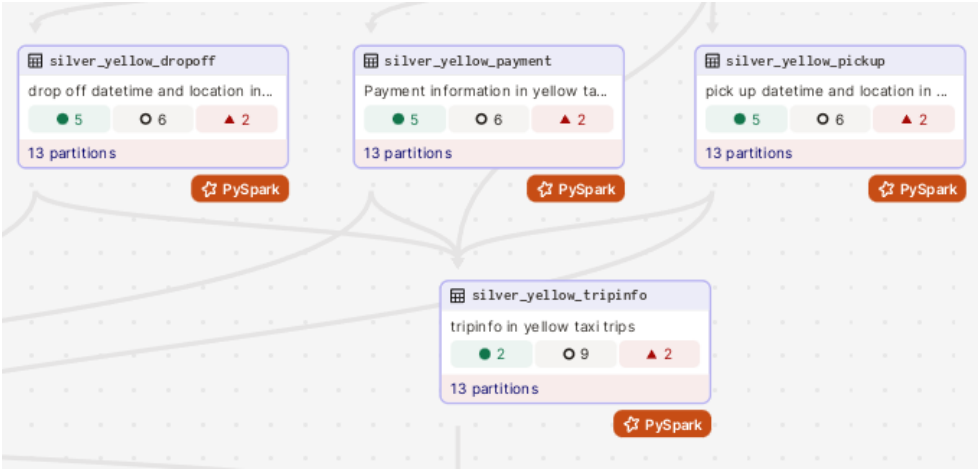
  – silver_fhv_pickup: tương tự là các thông tin về thời gian địa điểm đón khách

  – silver_fhv_info: là thông tin về chuyến đi như quảng đường, số hành khác, ...



Silver yellow: dùng **spark_io_manager** làm **io manager key**, data cleaning và transform từ upstream asset là bronze_yellow_record, tại các assets này thực hiện tách bronze_yellow_record ra thành các asset:

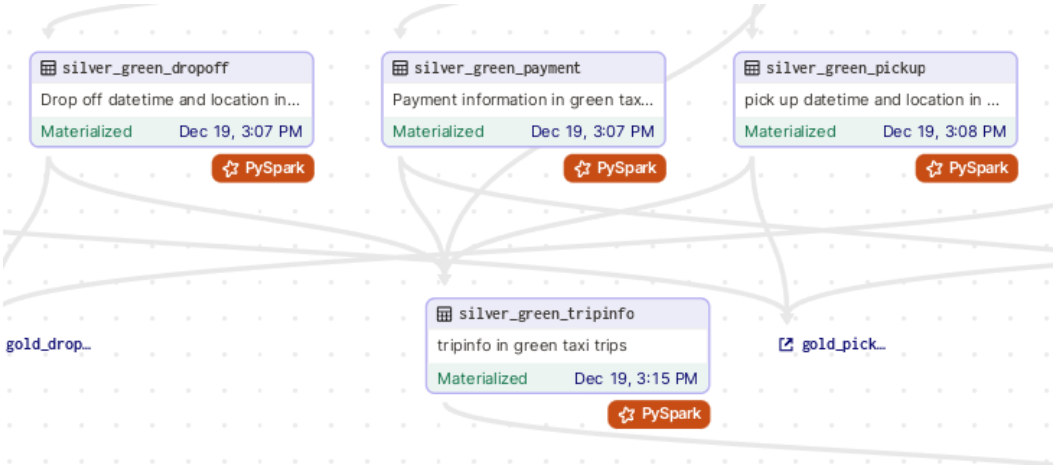
- silver_yellow_dropoff: chứa các thông tin về thời gian và địa điểm khách xuống xe của các taxi
- silver_yellow_pickup: tương tự là các thông tin về thời gian địa điểm đón khách
- silver_yellow_payment: Là thông tin về các chi phí như là phí xe, phụ phí, tips, ...
- silver_yellow_info: là thông tin về chuyến đi như quảng đường, số hành khách, ...



Silver green: dùng **spark_io_manager** làm **io manager key**, data cleaning và transform từ upstream asset là bronze_green_record, tại các assets này thực hiện tách bronze_green_record ra thành các asset:

- silver_green_dropoff: chứa các thông tin về thời gian và địa điểm khách xuống xe của các taxi
- silver_green_pickup: tương tự là các thông tin về thời gian địa điểm đón khách
- silver_green_payment: Là thông tin về các chi phí như là phí xe, phụ phí, tips, ...
- silver_green_info: là thông tin về chuyến đi như quảng đường, số hành khách, ...

Bên trong mỗi asset thực hiện cài đặt ID thích hợp, khử trừng lặp, xóa bớt null và các giá trị chưa thích hợp.



Và sau khi xử lý ta có được

Yellow

- silver_yellow_pickup: PickUpID, Pickup_datetime, PULocationID

- silver_yellow_dropoff: DropOffID, Dropoff_datetime, DOLocationID

- silver_yellow_payment: PaymentID, Fare_amount, MTA_tax, Improvement_surcharge, Payment_type ,RateCodeID, Extra, Tip_amount, Tolls_amount, Total_amount, Congestion_Surcharge, Airport_fee

- silver_yellow_tripinfo: VendorID, PickUpID, DropOffID, PaymentID, Passenger_count, Trip_distance, Store_and_fwd_flag

Green

- silver_green_pickup: PickUpID, Pickup_datetime, PULocation

- silver_green_dropoff: DropOffID, Dropoff_datetime, DOLocation

- silver_green_payment: PaymentID, Fare_amount, MTA_tax, Improvement_surchar
Payment_type, RateCodeID, Extra, Tip_amount, Tolls_amount, Total_amount

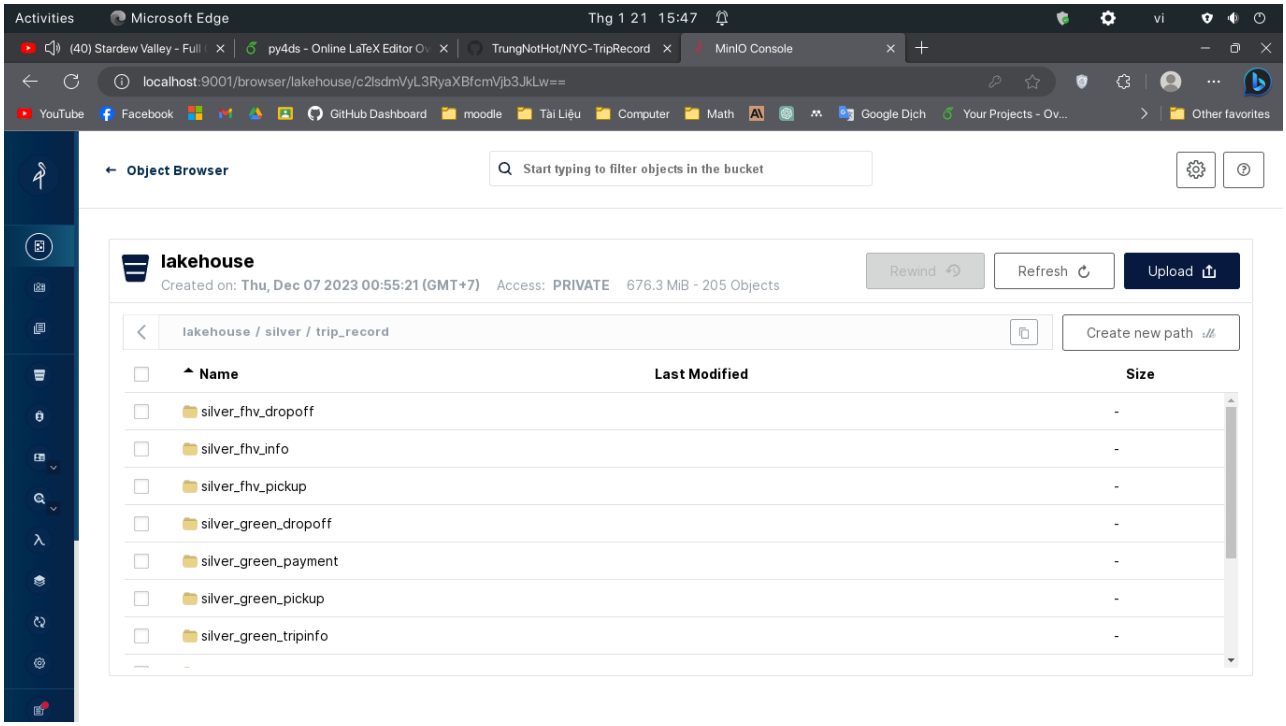
- silver_green_tripinfo: VendorID, PickUpID, DropOffID, PaymentID, Passenger_count,Trip_distance, Store_and_fwd_flag, Trip_type

Fhv

- silver_fhv_pickup: PickUpID, Pickup_datetime, PULocationID

- silver_fhv_dropoff: DropOffID, Dropoff_datetime, DOLocationID

- silver_fhvinfo: PickUpID, DropOffID, Dispatch_base_num, SR_Flag, Affiliated_base_number



Picture 2.3 folder silver trong minio sau khi chạy xong

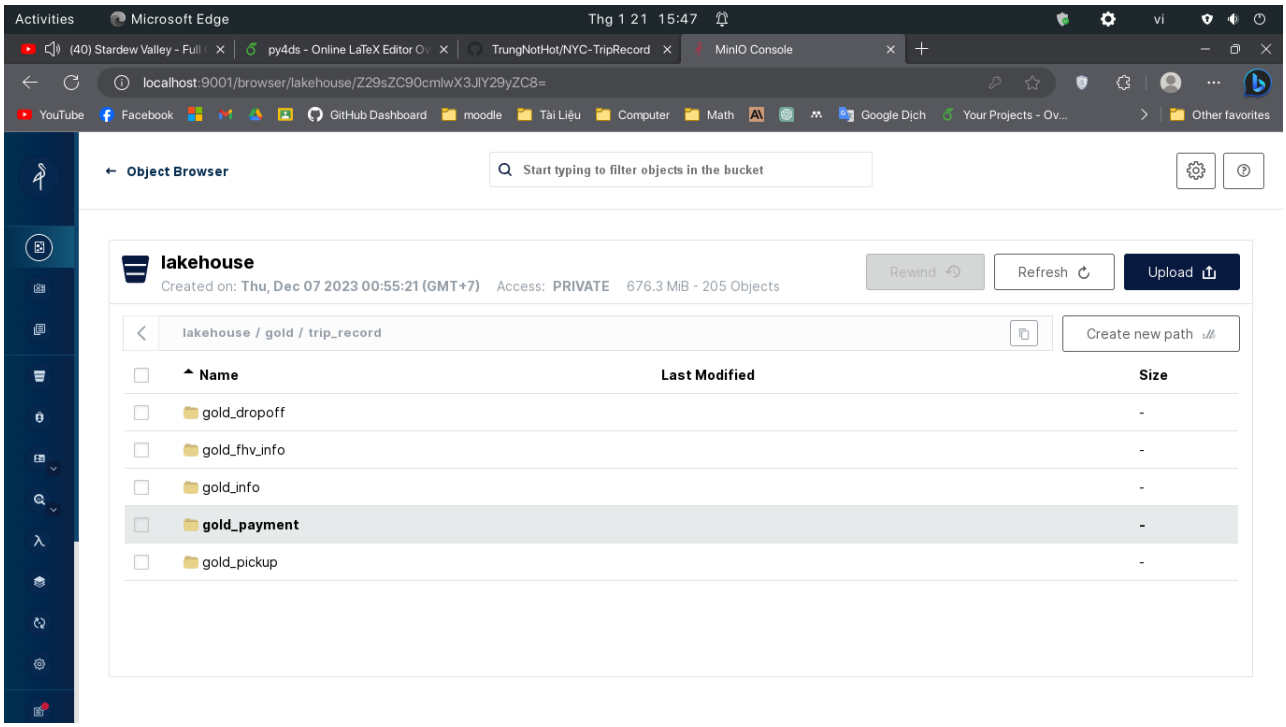
- **gold_layer**

Gold layer dùng **spark_io_manager** làm **io manager key**, tiếp đó thực hiện transform từ upstream asset

Bao gồm các assets được gộp và transform từ các upstream như sau:

– gold_pickup = silver_yellow_pickup + silver_green_pickup + silver_fhv_pickup + bronze_long_lat

- gold_dropoff = silver_yellow_dropoff + silver_green_dropoff + silver_fhv_dropoff + bronze_long_lat
- gold_payment = silver_yellow_payment + silver_green_payment
- gold_tripinfo = silver_yellow_tripinfo + silver_green_tripinfo : vì cấu trúc table của silver_yellow_tripinfo, silver_green_tripinfo khác với silver_fhv_tripinfo nên nhóm chỉ thực hiện gom của yellow và green và giữ nguyên fhv_info
- gold_info = silver_fhv_tripinfo



Picture 2.4 folder gold trong minio sau khi chạy xong

- **warehouse_layer**

  Tại layer này sử dụng **psql_io_manager** làm **io manager key**, và thực hiện lấy data của gold layer đưa vào psql

```python
@asset(
    name = 'warehouse_pickup',
    description="Load gold_pickup data from spark to postgres",
    ins={
        "gold_pickup": AssetIn(
            key_prefix=["gold", "trip_record"],
        ),
    },
    metadata={
        "primary_keys": ["PickUpID"],
        "columns": ["PickUpID", "PULocationID", "pickup_datetime", "longitude", "latitude"],
    },
    io_manager_key="psql_io_manager",
    key_prefix=["warehouse"],
    compute_kind="Postgres",
    group_name="warehouse",
)
def warehouse_pickup(context, gold_pickup: DataFrame):

    context.log.info("Got spark DataFrame, loading to postgres")

    return Output(
        gold_pickup,
        metadata={
            "database": "trip_record",
            "schema": "warehouse",
            "table": "warehouse_pickup",
            "primary_keys": ["PickUpID"],
            "row_count": gold_pickup.count(),
        },
    )

```

Picture 2.5 1 asset ví dụ trong warehouse layer

Dữ liệu từ gold_layer sẽ được tải lên data warehouse (Postgresql) để lưu trữ và sử dụng sau khi đã được xử lý trong pipeline

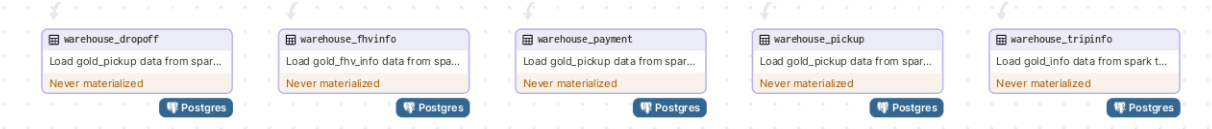
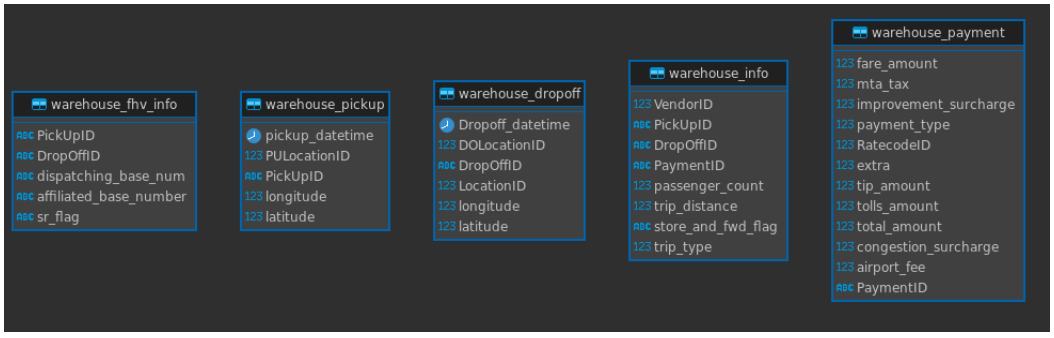
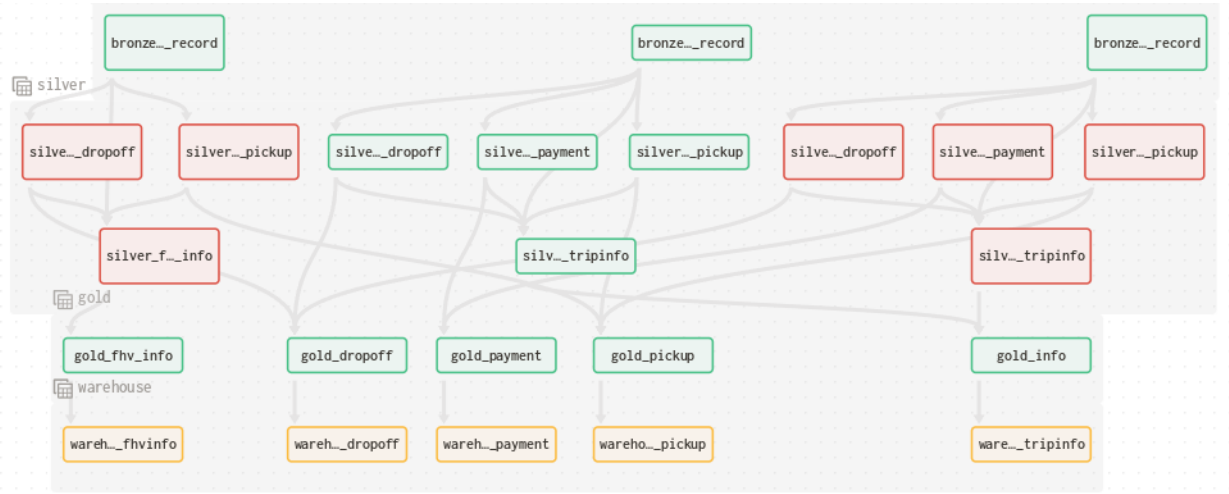


Table có được trong Postgresql sau khi đi qua pipeline

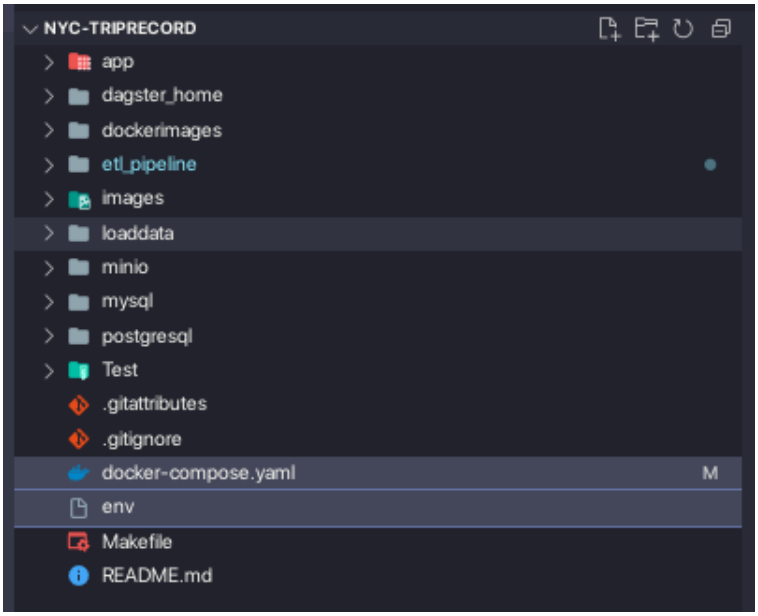


- **Data Lineage**

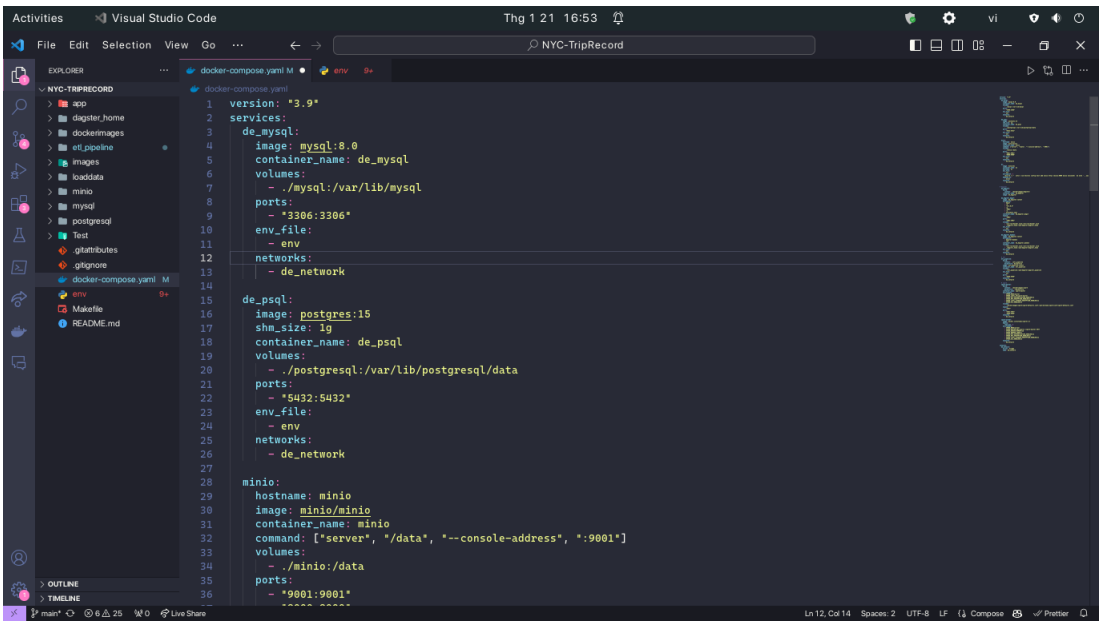


3. CÀI ĐẶT

3.1 Ngoài thư mục lớn NYC-TRIPRECORD

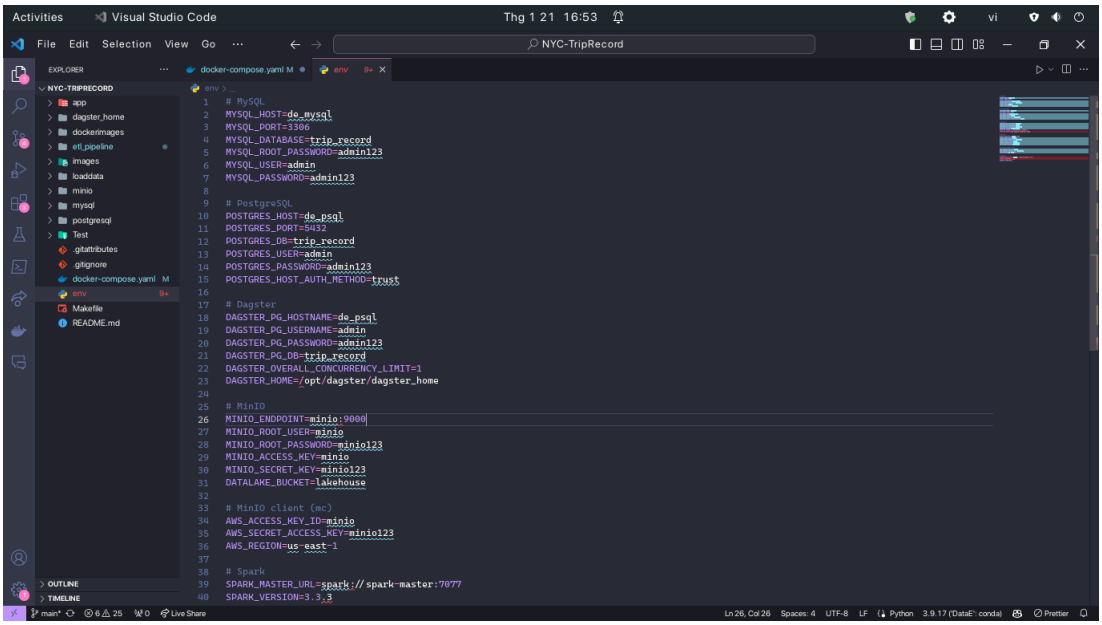


Tại 2 file env và docker-compose như hình trên



Picture 3.6 `https://github.com/TrungNotHot/NYC-TripRecord/blob/main/docker-compose.yaml`

Sử dụng docker để tạo các docker container sử dụng cho project

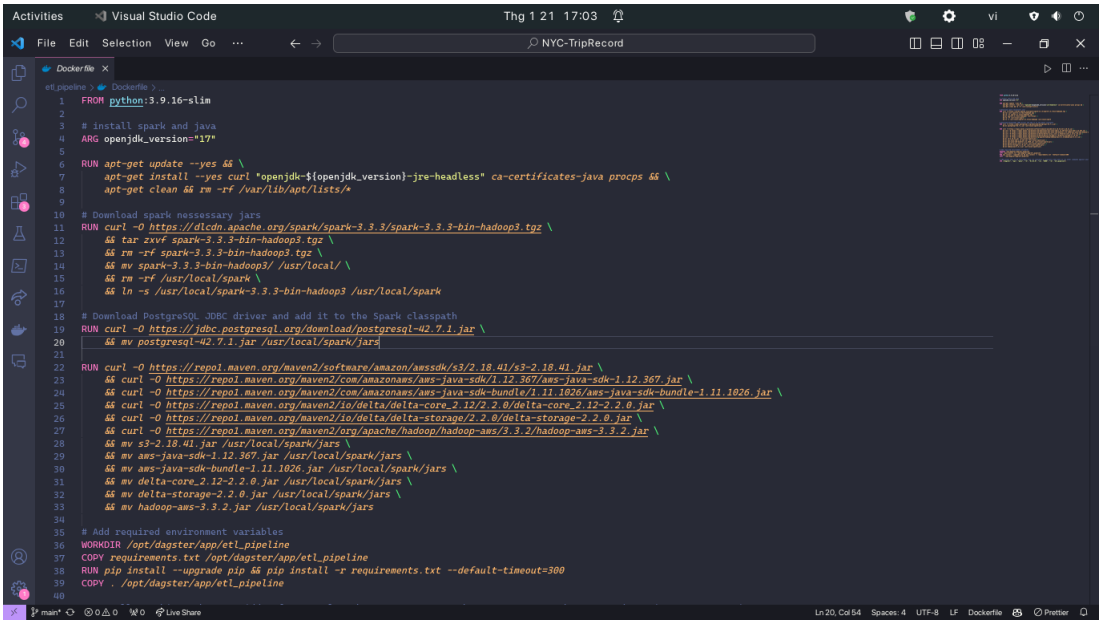


Picture 3.7 `https://github.com/TrungNotHot/NYC-TripRecord/blob/main/env`

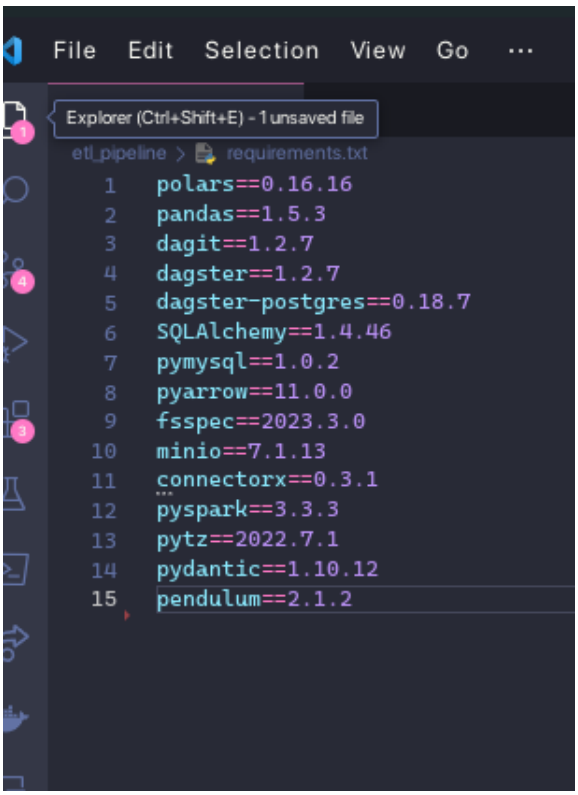
Tạo file env để chứa các biến môi trường như tên database, username , password ...

3.2 Trong thư mục etl_pipeline

Tạo file requirements.txt để cài các thư viện python cần thiết cho etl_pipeline, và 1 Dockerfile



Picture 3.8 `https://github.com/TrungNotHot/NYC-TripRecord/blob/main/etl_pipeline/Dockerfile`



Picture 3.9 Các thư viện python được cài trong etl_pipeline

4. DEMO KẾT QUẢ VÀ VISUALIZE

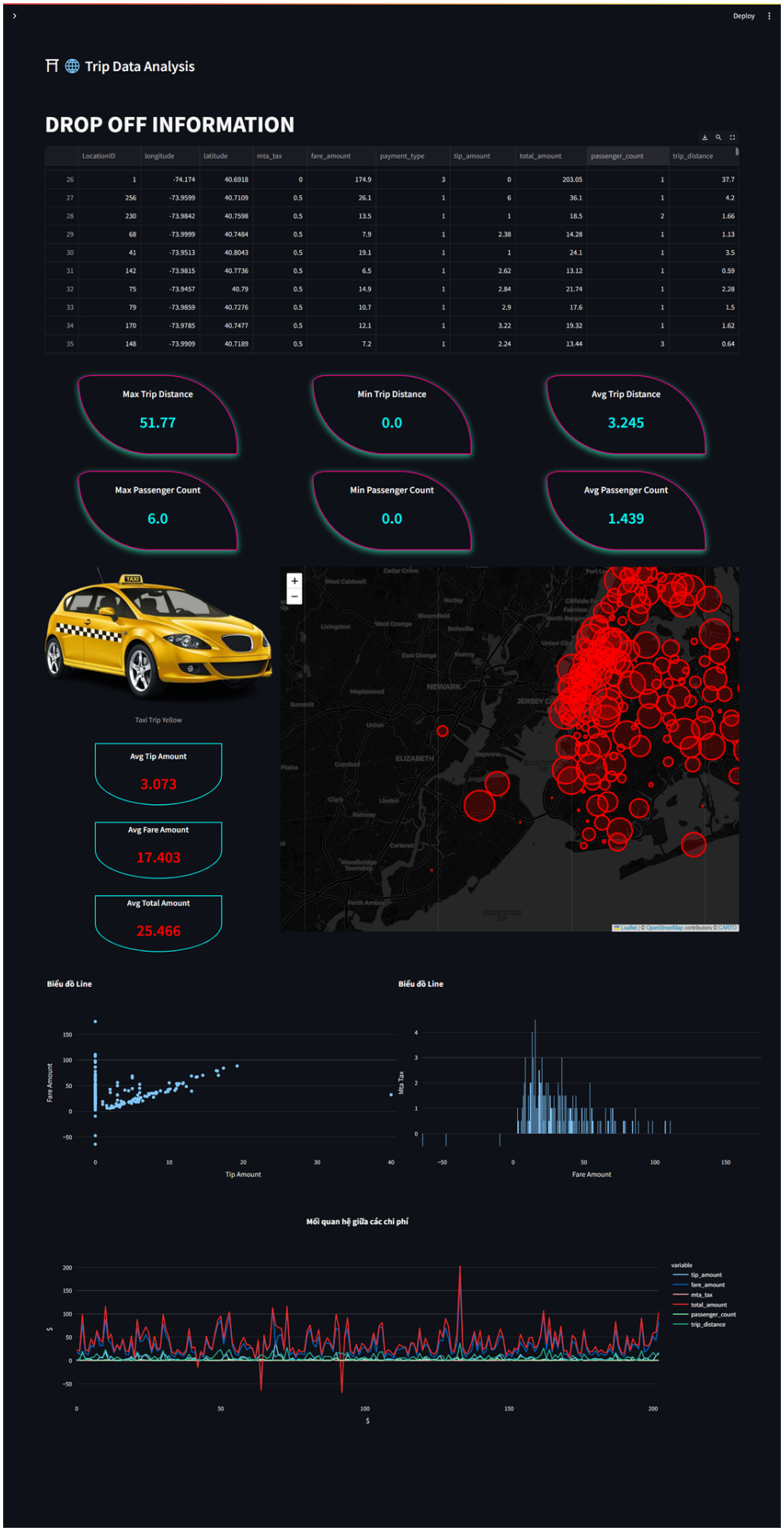
4.1 DEMO

Demo link: `https://studenthcmusedu-my.sharepoint.com/personal/21280012_student_hcmus_edu_vn/_layouts/15/stream.aspx?id=%2Fpersonal%2F21280012%5Fstudent%5Fhcmus%5Fedu%5Fvn%2FDocuments%2FDE%2Fdemo%2Emp4&referrer=StreamWeb2EWeb&referrerScenario=AddressBarCopied%2Eview`

4.2 VISUALIZE

Sử dụng **Streamlit** Library để visualize các dữ liệu

Dữ liệu được visualize được connect đến Postgres để lấy dữ liệu và chuyển đổi xử lí theo như mong muốn


Deploy
Trip Data Analysis
DROP OFF INFORMATION
Max Trip Distance
51.77
Min Trip Distance
0.0
Avg Trip Distance
3.245
Max Passenger Count
6.0
Min Passenger Count
0.0
Avg Passenger Count
1.439
Taxi Trip Yellow
Avg Tip Amount
3.073
Avg Fare Amount
17.403
Avg Total Amount
25.466
Biểu đồ Line
Fare Amount
Tip Amount
Biểu đồ Line
Mta Tax
Fare Amount
Mối quan hệ giữa các chi phí
variable
tip_amount
fare_amount
mta_tax
total_amount
passenger_count
trip_distance
$

TÀI LIỆU THAM KHẢO

[1] Tài liệu Python cho KHDL thầy Hà Văn Thảo

[2] PySpark Overview

[3] Dagster doc

[4] `https://github.com/vsonwork/yellow-taxi-trip-ETL`

[5] `https://github.com/lelouvincx/goodreads-elt-pipeline`

[6] Docker doc