Outline

1. Configuring Mage with Docker

We will use Docker and Docker-compose to run the Mage tool for orchestration. First, lets create our docker file by pulling from docker hub the latest version of mage image:

```
docker pull mageai/mageai:latest
```
and then create the docker file:

Dockerfile

```
FROM mageai/mageai:latest

  #  For a dynamic path based on environment variable inside .env
  ARG USER_CODE_PATH=/home/src/${PROJECT_NAME}

  # copy the requirements file to the container
  COPY ["requirements.txt", "${USER_CODE_PATH}"]

  # install the requirements
  RUN pip3 install -r ${USER_CODE_PATH}/requirements.txt
```
We use a specific path within the Docker container for our project, separate from our local project folder to leverage Docker's strengths in providing a consistent, isolated, and secure environment for development, testing, and deployment. We also use the `Pipfile` and `Pipfile.lock` files to install the project dependencies using `pipenv`. For the environment variable lets create a `.env` file with the following content:

.env

```
PROJECT_NAME=green_taxi
    POSTGRES_DBNAME=ny_taxi
    POSTGRES_SCHEMA=mage
    POSTGRES_USER=marcosbenicio
    POSTGRES_PASSWORD=0102
    POSTGRES_HOST=postgres
    POSTGRES_PORT=5432
```
For the `docker-compose.yml` file, we will use the following configuration:

docker-compose.yml

```
version: '3'
    services:

      magic:
        env_file: # to load environment variables from a file
          - .env
        build:
          context: .   # the context is the current directory in which the docker-compose.yml file is
located
          dockerfile: Dockerfile  # the Dockerfile to use for building the image
        command: mage start ${PROJECT_NAME}
        environment:
          USER_CODE_PATH: /home/src/${PROJECT_NAME}
          POSTGRES_DBNAME: ${POSTGRES_DBNAME}
          POSTGRES_SCHEMA: ${POSTGRES_SCHEMA}
          POSTGRES_USER: ${POSTGRES_USER}
          POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
          POSTGRES_HOST: ${POSTGRES_HOST}
          POSTGRES_PORT: ${POSTGRES_PORT}
        ports:
          - 6789:6789
        volumes:
          - .:/home/src/  # copy the current directory to the /home/src directory in the container
        restart: on-failure:5
      postgres: # must have same name as the environment variable POSTGRES_HOST
        image: postgres:14
        restart: on-failure
        container_name: ${PROJECT_NAME}-postgres
        env_file:
          - .env
        environment:
          POSTGRES_DB: ${POSTGRES_DBNAME}
          POSTGRES_USER: ${POSTGRES_USER}
          POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
        ports:
          - "${POSTGRES_PORT}:5432"
        volumes: # to persist data after container restarts
          - "./taxi-trip-postgres:/var/lib/postgresql/data:rw" # mount the local directory to the
container
```

The `docker-compose.yml` file specifies the instructions to build a custom Docker image for the `magic` service using a Dockerfile and then run a container based on that image with the `mage start` command. This command is intended to start the Mage UI and the Mage API within the `magic` container. Separately, the `docker-compose.yml` file also defines a `postgres` service, which runs a PostgreSQL container.

The volumes section is used to mount paths from the host machine into the container, allowing for data to be shared between host and the container or for data persistence across container restarts. For `.:/home/src/` we are mounting the current directory (represented by `.`) into the container inside `home/src/` structure.

After creating the `Dockerfile`, `.env`, and `docker-compose.yml` files, we can run the following command to build the custom Docker image and start the containers:

```
docker-compose build

    docker-compose up -d
```

To access Mage UI navigate to `http://localhost:6789` in a browser.

2. Extract, Transform, Load (ETL) Pipeline with Mage and Postgres

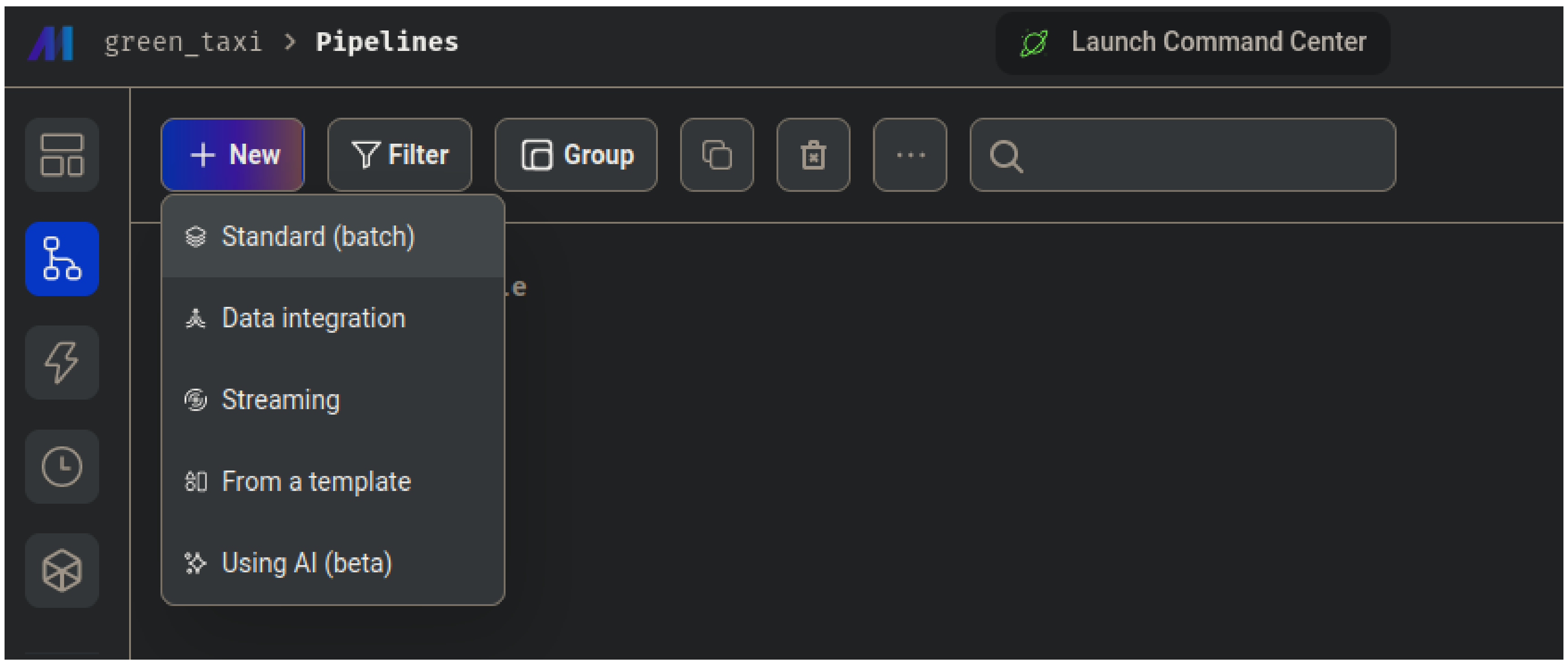
Mage has three main components, know as `blocks`, to create a pipeline: `Data Loader`, `transform`, and `Data Exporter`. The `Data Loader` component is responsible for reading data from a source, the `transform` component is responsible for transforming the data, and the `Data Exporter` component is responsible for writing the data to a destination.

To create our connection with the Postgres database, we can create a profile by editing the `io_profile.yaml` file. The `io_profile.yaml` file is used to define the connection to databases. Inside the file we find a pre defined profile named default as example. Based on the examples for connection, we can create a new profile for our Postgres database call dev:

io_profile.yml

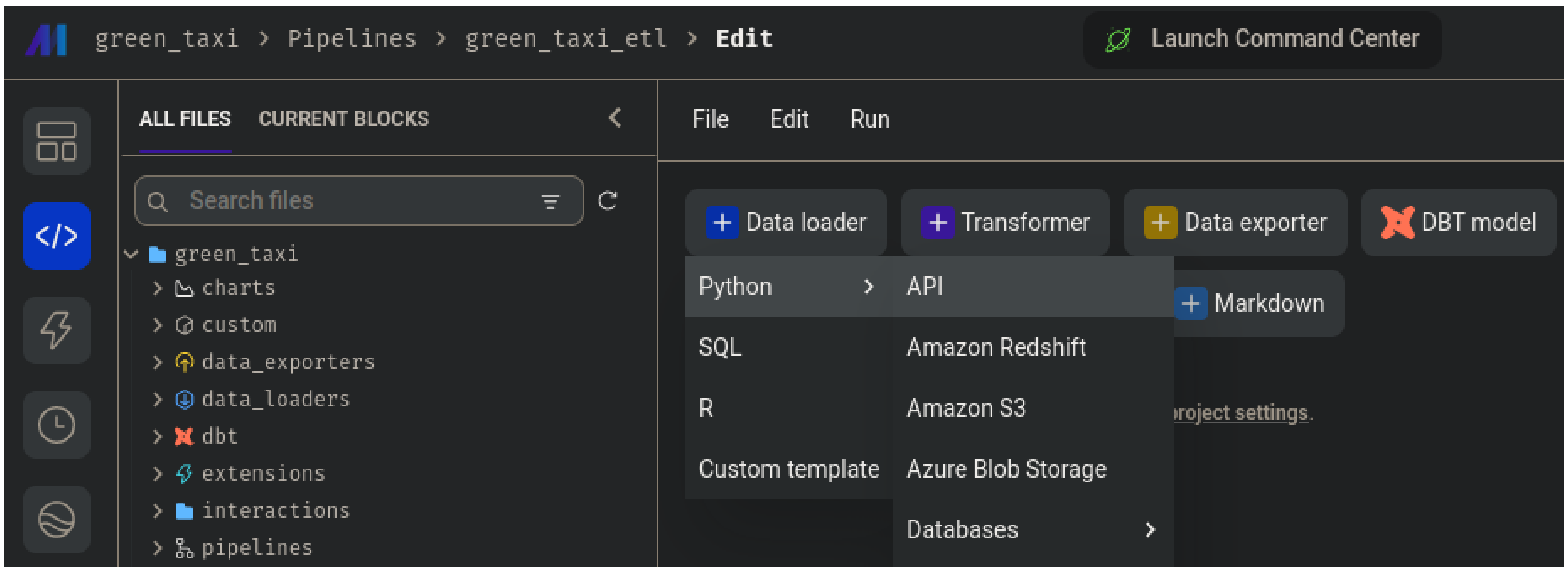
```
dev:
        POSTGRES_CONNECT_TIMEOUT: 10
        POSTGRES_DBNAME: "{{ env_var('POSTGRES_DBNAME') }}"
        POSTGRES_SCHEMA: "{{ env_var('POSTGRES_SCHEMA') }}"
        POSTGRES_USER: "{{ env_var('POSTGRES_USER') }}"
        POSTGRES_PASSWORD: "{{ env_var('POSTGRES_PASSWORD') }}"
        POSTGRES_HOST: "{{ env_var('POSTGRES_HOST') }}"
        POSTGRES_PORT: "{{ env_var('POSTGRES_PORT') }}"
```

the `env_var` function is used to access the environment variables defined in the `.env` file. The changes can be made either in vscode or the Mage GUI in our web browser. Now, let's create the pipeline, in the Mage GUI by clicking on the `+` button and selecting the Standard (batch) option as in the figure below:

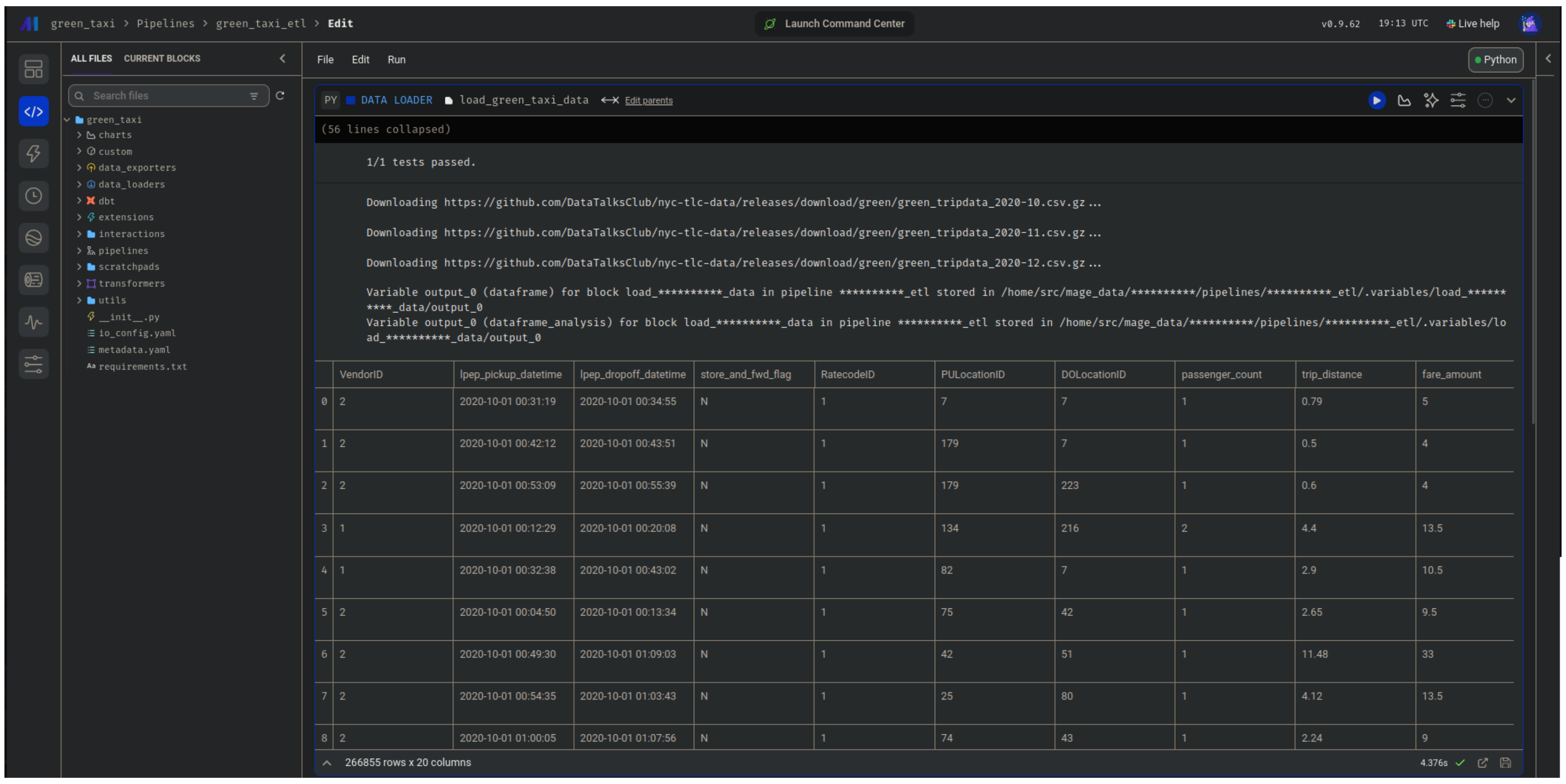


2.1 Data Loader

To change the pipline name we can click on the `Edit > Pipline settings` and change to `green_taxi_etl`. To create a data `Data Loader` block we can click on the `+` button and select the desired block with the python language and then click on the `API` option to create the block:



The `Data Loader` block is responsible for reading data from a source. For this block, we pre-define the datatypes for a optimization in memory usage and download from this github repo for the green taxi data from 2019 - 2021. We select just the final quarter of 2020 (months `10`, `11`, `12`) using a for loop and then concatenate the dataframes into a single dataframe. The result of this data Loader block would be the following:



```
(56 lines collapsed)

    1/1 tests passed.

    Downloading https://github.com/DataTalksClub/nyc-tlc-data/releases/download/green/green_tripdata_2020-10.csv.gz ...

    Downloading https://github.com/DataTalksClub/nyc-tlc-data/releases/download/green/green_tripdata_2020-11.csv.gz ...

    Downloading https://github.com/DataTalksClub/nyc-tlc-data/releases/download/green/green_tripdata_2020-12.csv.gz ...

    Variable output_0 (dataframe) for block load_**********_data in pipeline **********_etl stored in /home/src/mage_data/**********/pipelines/**********_etl/.variables/load_******
    ****_data/output_0
    Variable output_0 (dataframe_analysis) for block load_**********_data in pipeline **********_etl stored in /home/src/mage_data/**********/pipelines/**********_etl/.variables/lo
    ad_**********_data/output_0
```

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	RatecodeID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount
0	2	2020-10-01 00:31:19	2020-10-01 00:34:55	N	1	7	7	1	0.79	5
1	2	2020-10-01 00:42:12	2020-10-01 00:43:51	N	1	179	7	1	0.5	4
2	2	2020-10-01 00:53:09	2020-10-01 00:55:39	N	1	179	223	1	0.6	4
3	1	2020-10-01 00:12:29	2020-10-01 00:20:08	N	1	134	216	2	4.4	13.5
4	1	2020-10-01 00:32:38	2020-10-01 00:43:02	N	1	82	7	1	2.9	10.5
5	2	2020-10-01 00:04:50	2020-10-01 00:13:34	N	1	75	42	1	2.65	9.5
6	2	2020-10-01 00:49:30	2020-10-01 01:09:03	N	1	42	51	1	11.48	33
7	2	2020-10-01 00:54:35	2020-10-01 01:03:43	N	1	25	80	1	4.12	13.5
8	2	2020-10-01 01:00:05	2020-10-01 01:07:56	N	1	74	43	1	2.24	9

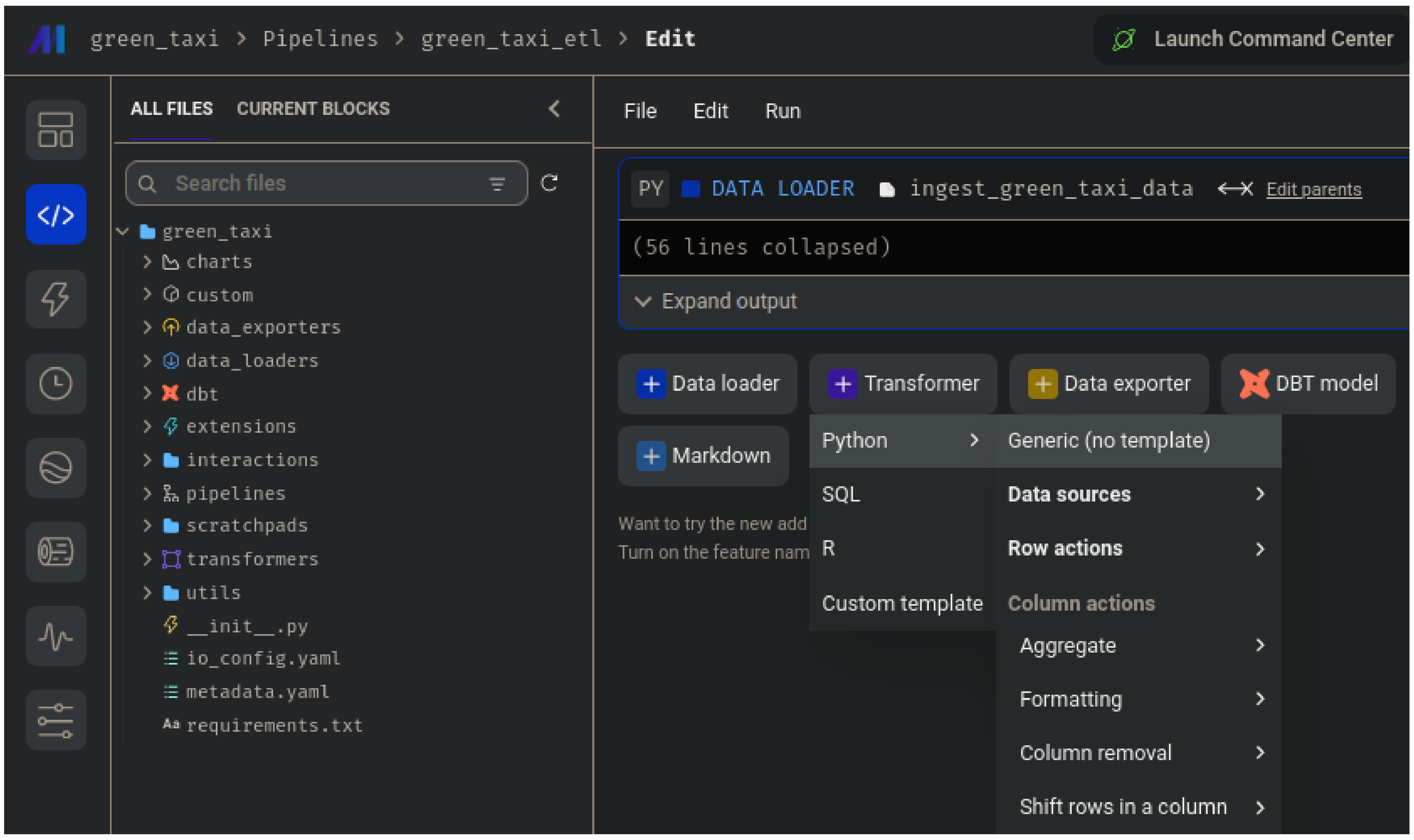
266855 rows x 20 columns   4.376s

The full code inside this block is:

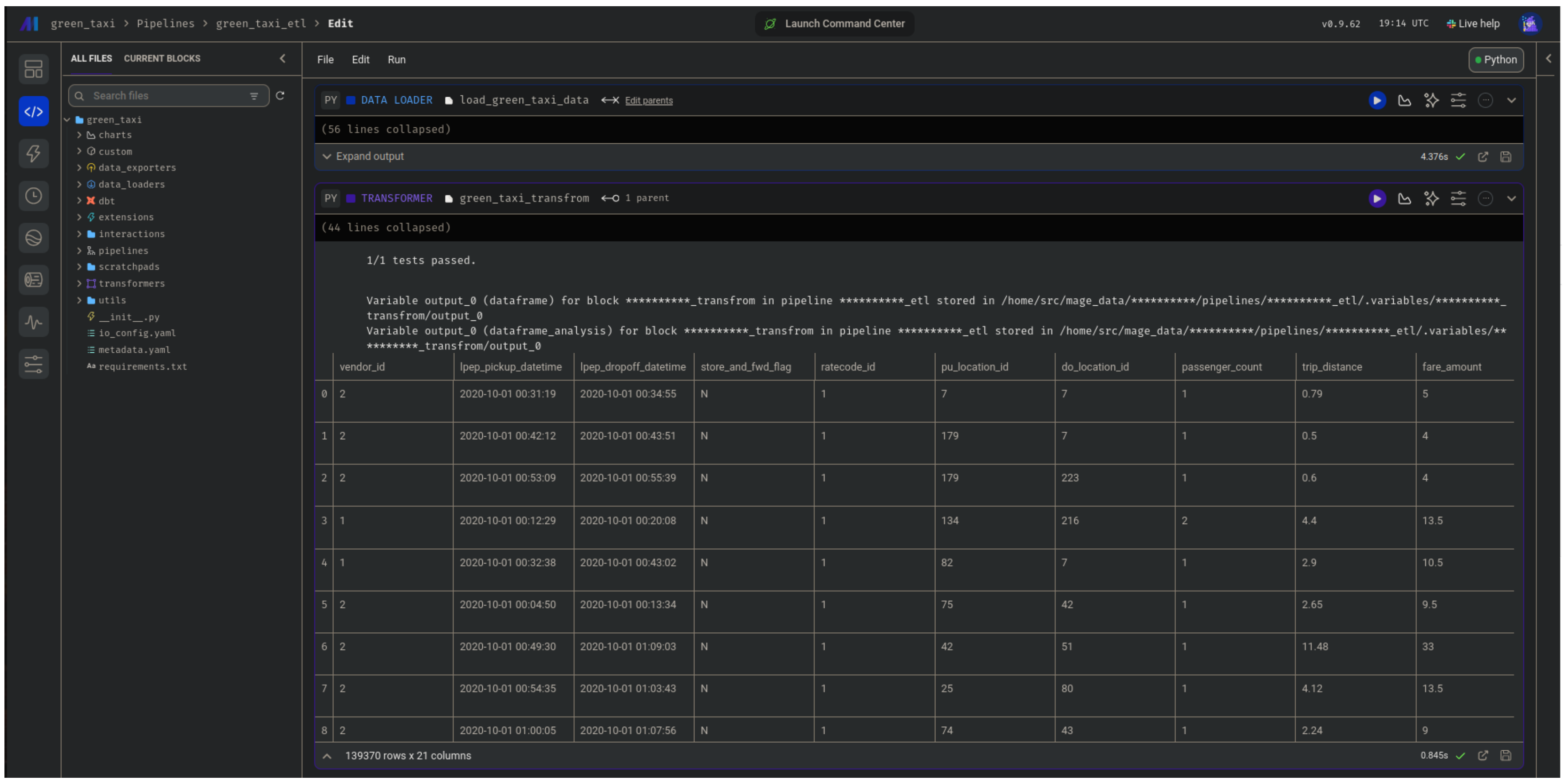
```python
import io
    import pandas as pd
    import requests
    if 'data_loader' not in globals():
        from mage_ai.data_preparation.decorators import data_loader
    if 'test' not in globals():
        from mage_ai.data_preparation.decorators import test


    @data_loader
    def load_data_from_api(*args, **kwargs):
        """
        Template for loading data from API
        """
        dates = ['2020-10', '2020-11', '2020-12']
        # Base URL pattern
        base_url = 'https://github.com/DataTalksClub/nyc-tlc-
data/releases/download/green/green_tripdata_{}.csv.gz'

        taxi_dtypes = {
            'VendorID': pd.Int64Dtype(),
            'passenger_count': pd.Int64Dtype(),
            'trip_distance': float,
            'RatecodeID': pd.Int64Dtype(),
            'store_and_fwd_flag': str,
            'PULocationID': pd.Int64Dtype(),
            'DOLocationID': pd.Int64Dtype(),
            'payment_type': pd.Int64Dtype(),
            'fare_amount': float,
            'extra': float,
            'mta_tax': float,
            'tip_amount': float,
            'tolls_amount': float,
            'improvement_surcharge': float,
            'total_amount': float,
            'congestion_surcharge': float
        }

        df_list = []
        for date in dates:
            # Construct the download URL for each date
            url = base_url.format(date)
            df = pd.read_csv(url, sep=',', compression='gzip', dtype=taxi_dtypes)
            print(f"Downloading {url}...")
            df_list.append(df)




        return pd.concat(df_list, ignore_index=True)

    @test
    def test_output(output, *args) -> None:
        """
        Template code for testing the output of the block.
        """
        assert output is not None, 'The output is undefined'
```

2.1 Data Transform

Now that we created our Data Loader block, we can create the `Data Transform` block. As we did before, we can click on the `+` button and select the desired block with the python language and then click on the `Generic (no template)` option to create the block:



The `Data Transform` block is responsible for transforming the data. For this block, we will remove rows where the passenger count is 0 or the trip distance is 0, create a new column 'lpep_pickup_date' by converting 'lpep_pickup_datetime' to a date, and rename columns in Camel Case to Snake Case. The result of this data transform block would be the following:



```
1/1 tests passed.

Variable output_0 (dataframe) for block **********_transfrom in pipeline **********_etl stored in /home/src/mage_data/**********/pipelines/**********_etl/.variables/**********_
transfrom/output_0
Variable output_0 (dataframe_analysis) for block **********_transfrom in pipeline **********_etl stored in /home/src/mage_data/**********/pipelines/**********_etl/.variables/**
********_transfrom/output_0
```

	vendor_id	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	ratecode_id	pu_location_id	do_location_id	passenger_count	trip_distance	fare_amount
0	2	2020-10-01 00:31:19	2020-10-01 00:34:55	N	1	7	7	1	0.79	5
1	2	2020-10-01 00:42:12	2020-10-01 00:43:51	N	1	179	7	1	0.5	4
2	2	2020-10-01 00:53:09	2020-10-01 00:55:39	N	1	179	223	1	0.6	4
3	1	2020-10-01 00:12:29	2020-10-01 00:20:08	N	1	134	216	2	4.4	13.5
4	1	2020-10-01 00:32:38	2020-10-01 00:43:02	N	1	82	7	1	2.9	10.5
5	2	2020-10-01 00:04:50	2020-10-01 00:13:34	N	1	75	42	1	2.65	9.5
6	2	2020-10-01 00:49:30	2020-10-01 01:09:03	N	1	42	51	1	11.48	33
7	2	2020-10-01 00:54:35	2020-10-01 01:03:43	N	1	25	80	1	4.12	13.5
8	2	2020-10-01 01:00:05	2020-10-01 01:07:56	N	1	74	43	1	2.24	9

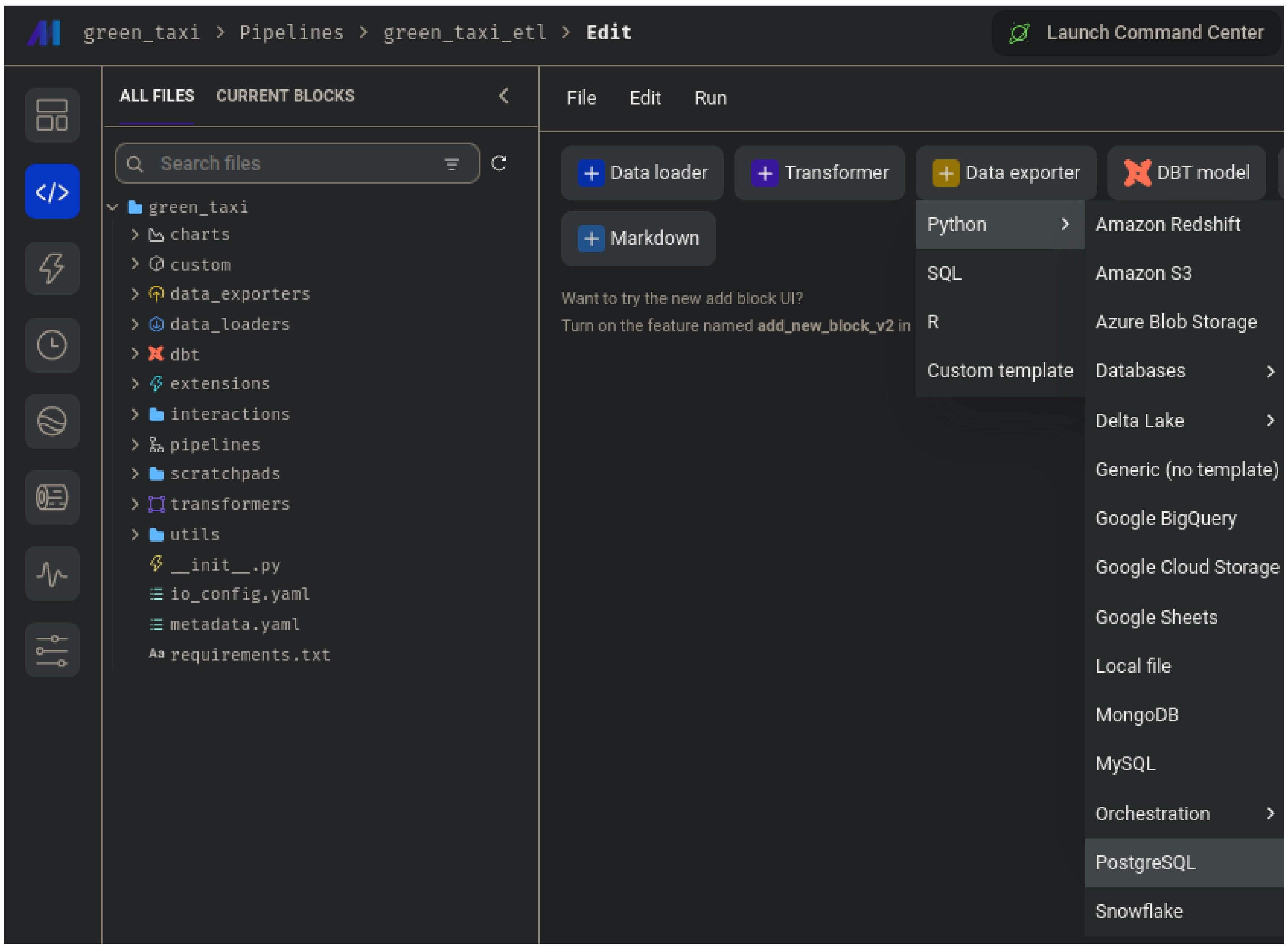
139370 rows x 21 columns

The full code inside this block is:

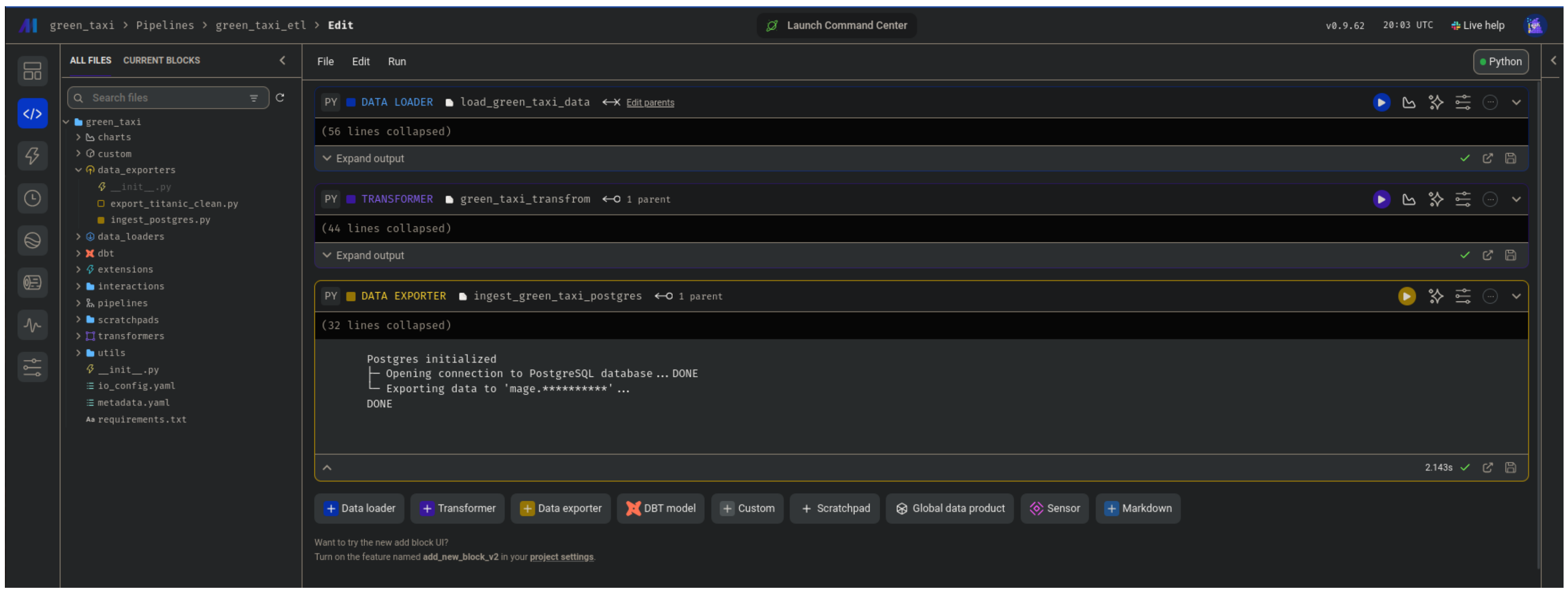
```python
import pandas as pd
    import re

    if 'transformer' not in globals():
        from mage_ai.data_preparation.decorators import transformer
    if 'test' not in globals():
        from mage_ai.data_preparation.decorators import test


    @transformer
    def transform(data, *args, **kwargs):

        # Check is all columns are snake case
        def is_snake_case(name):
            return re.match(r'^[a-z_][a-z0-9_]*$', name) is not None

        # convert all camel columns names to snake
        def camel_to_snake(name):
            return re.sub(r'(?<=[a-z0-9])([A-Z])|(?<=[A-Z])([A-Z])(?=[a-z])', r'_\g<0>', name).lower()

        # Remove rows where the passenger count is 0 or the trip distance is 0.
        condition = (data['passenger_count'] > 0) & (data['trip_distance'] > 0)
        data = data[condition]

        # Create a new column 'lpep_pickup_date' by converting 'lpep_pickup_datetime' to a date.
        data['lpep_pickup_date'] = pd.to_datetime(data['lpep_pickup_datetime']).dt.date.copy()

        # Rename columns in Camel Case to Snake Case
        data.columns = [camel_to_snake(column) for column in data.columns]

        # Add three assertion to check the transformations
        assert all(is_snake_case(column) for column in data.columns), "Not all column names are in
snake case."
        assert (data['passenger_count'] > 0).all(), "'passenger_count' contains non-positive values."
        assert (data['trip_distance'] > 0).all(), "'trip_distance' contains non-positive values."

        return data


    @test
    def test_output(output, *args) -> None:
        """
        Template code for testing the output of the block.
        """
        assert output is not None, 'The output is undefined'
```

2.3 Data Exporter

Finally, the `Data Exporter` block is responsible for writing the data to a destination after the process of transformation. For this block, we will write the data to a Postgres database that is running in another docker container.



The result of this data exporter block would be the following:



```
(56 lines collapsed)
```

```
(44 lines collapsed)
```

```
(32 lines collapsed)

      Postgres initialized
      ├─ Opening connection to PostgreSQL database ... DONE
      └─ Exporting data to 'mage.**********' ...
      DONE
```

Want to try the new add block UI?
Turn on the feature named **add_new_block_v2** in your project settings.

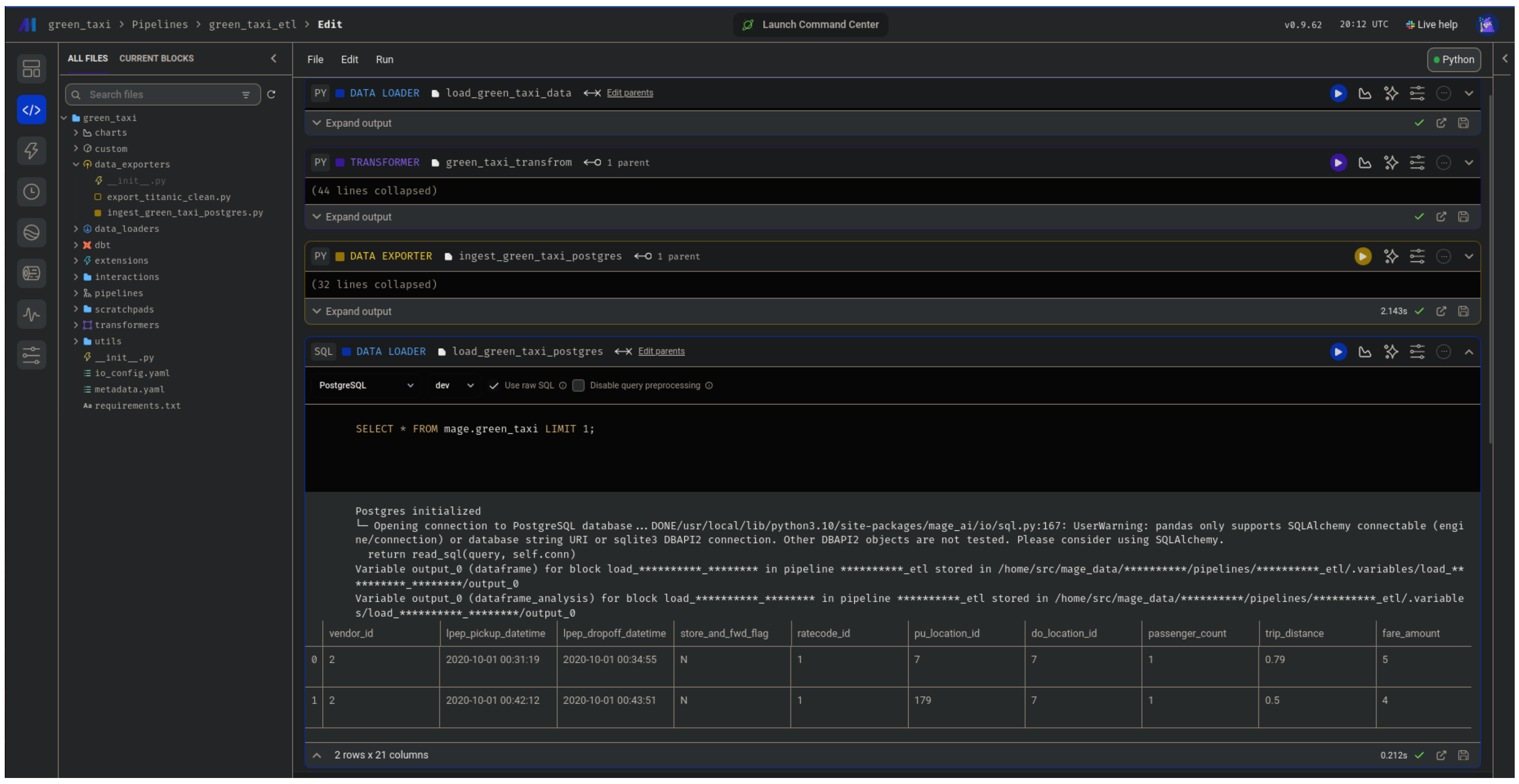
This should take the data from the `Data Transform` block, open a connection to Postgres container and export the data to our schema. The full code inside this block is:

```python
from mage_ai.settings.repo import get_repo_path
    from mage_ai.io.config import ConfigFileLoader
    from mage_ai.io.postgres import Postgres
    from pandas import DataFrame
    from os import path

    if 'data_exporter' not in globals():
        from mage_ai.data_preparation.decorators import data_exporter


    @data_exporter
    def export_data_to_postgres(df: DataFrame, **kwargs) -> None:
        """
        Template for exporting data to a PostgreSQL database.
        Specify your configuration settings in 'io_config.yaml'.

        Docs: https://docs.mage.ai/design/data-loading#postgresql
        """
        schema_name = 'mage'  # Specify the name of the schema to export data to
        table_name = 'green_taxi'  # Specify the name of the table to export data to
        config_path = path.join(get_repo_path(), 'io_config.yaml')
        config_profile = 'dev'

        with Postgres.with_config(ConfigFileLoader(config_path, config_profile)) as loader:
            loader.export(
                df,
                schema_name,
                table_name,
                index=False,  # Specifies whether to include index in exported table
                if_exists='replace',  # Specify resolution policy if table name already exists
            )
```

We can test the dataset exported to the postgres database by creating another data loader block for SQL, selecting the dev profile and typing the following query:

```sql
SELECT * FROM mage.green_taxi LIMIT 1;
```

The result should be the following:



```
(44 lines collapsed)
```

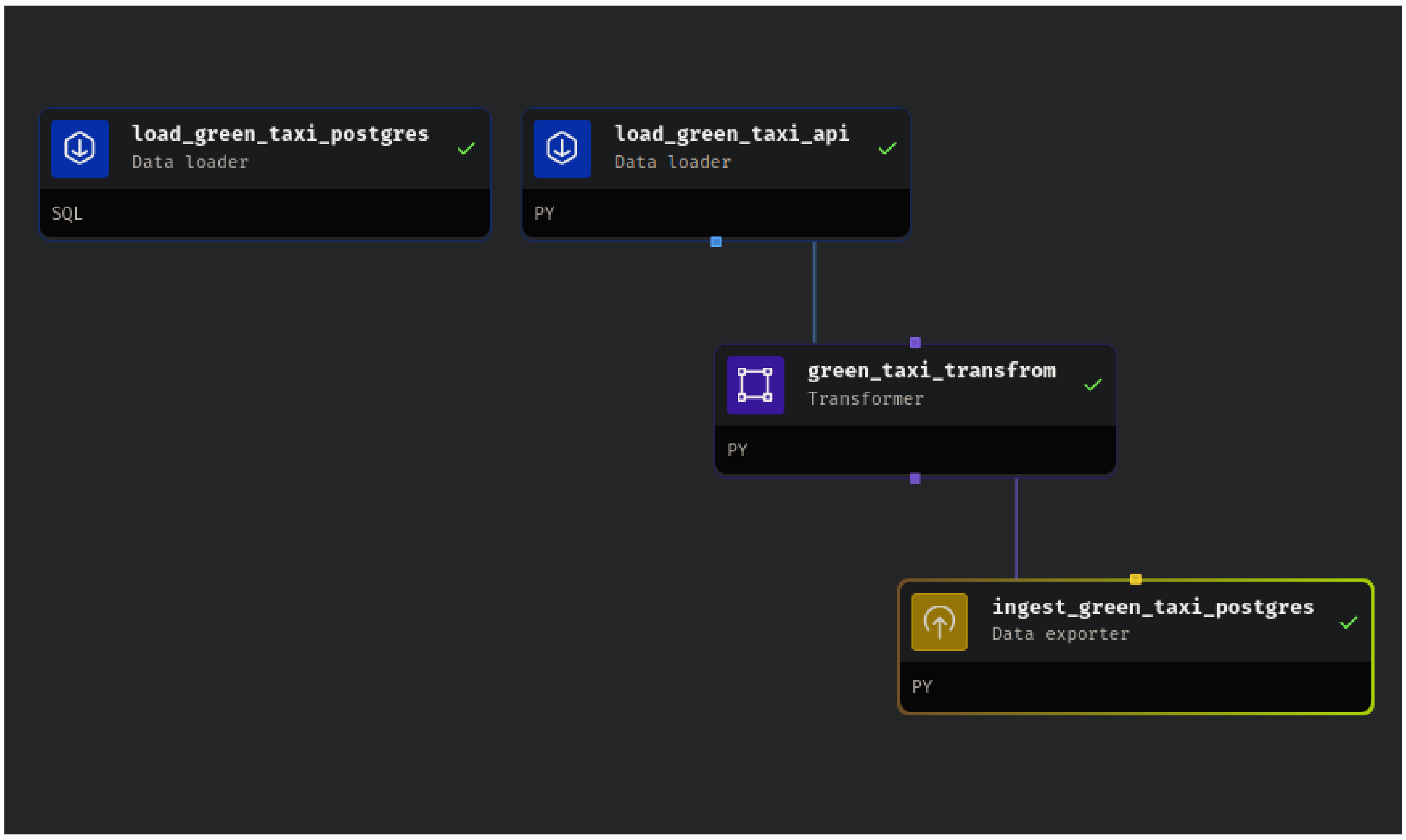
```
(32 lines collapsed)
```

```sql
SELECT * FROM mage.green_taxi LIMIT 1;
```

```
Postgres initialized
└─ Opening connection to PostgreSQL database ... DONE/usr/local/lib/python3.10/site-packages/mage_ai/io/sql.py:167: UserWarning: pandas only supports SQLAlchemy connectable (engi
ne/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  return read_sql(query, self.conn)
Variable output_0 (dataframe) for block load_**********_******** in pipeline **********_etl stored in /home/src/mage_data/**********/pipelines/**********_etl/.variables/load_**
********_********/output_0
Variable output_0 (dataframe_analysis) for block load_**********_******** in pipeline **********_etl stored in /home/src/mage_data/**********/pipelines/**********_etl/.variable
s/load_**********_********/output_0
```

	vendor_id	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	ratecode_id	pu_location_id	do_location_id	passenger_count	trip_distance	fare_amount
0	2	2020-10-01 00:31:19	2020-10-01 00:34:55	N	1	7	7	1	0.79	5
1	2	2020-10-01 00:42:12	2020-10-01 00:43:51	N	1	179	7	1	0.5	4

At the end, we would see the following pipeline graph showing each block and the connections between them:



This is a DAG (Directed Acyclic Graph), which is a tree describing our pipeline, its blocks and dependencies.

3. ETL Pipeline with Mage and Google Cloud

As we did previously in here, we can create a service account and generate a key to authenticate with Google Cloud. We then need to create a bucket to store data and access via Mage.

Mage Configuration File

With the service account created, download the key and save it in the Mage project directory. The `.json` will be automatically mounted into the container and we can use it to authenticate with Google Cloud. In the Mage GUI, go to files, open the `io_config.yaml` file and in the `default` profile search for the Google block:

```yaml
# Google
  GOOGLE_SERVICE_ACC_KEY:
    type: service_account
    project_id: project-id
    private_key_id: key-id
    private_key: "-----BEGIN PRIVATE KEY-----\nyour_private_key\n-----END_PRIVATE_KEY"
    client_email: your_service_account_email
    auth_uri: "https://accounts.google.com/o/oauth2/auth"
    token_uri: "https://accounts.google.com/o/oauth2/token"
    auth_provider_x509_cert_url: "https://www.googleapis.com/oauth2/v1/certs"
    client_x509_cert_url:
"https://www.googleapis.com/robot/v1/metadata/x509/your_service_account_email"
  GOOGLE_SERVICE_ACC_KEY_FILEPATH: "/path/to/your/service/account/key.json"
  GOOGLE_LOCATION: US # Optional
```

We can add the variables from `GOOGLE_SERVICE_ACC_KEY` or just specify the path to the `.json` file inside the container. Let's choose the second option and add the following configuration to the file and exclude the `GOOGLE_SERVICE_ACC_KEY`:

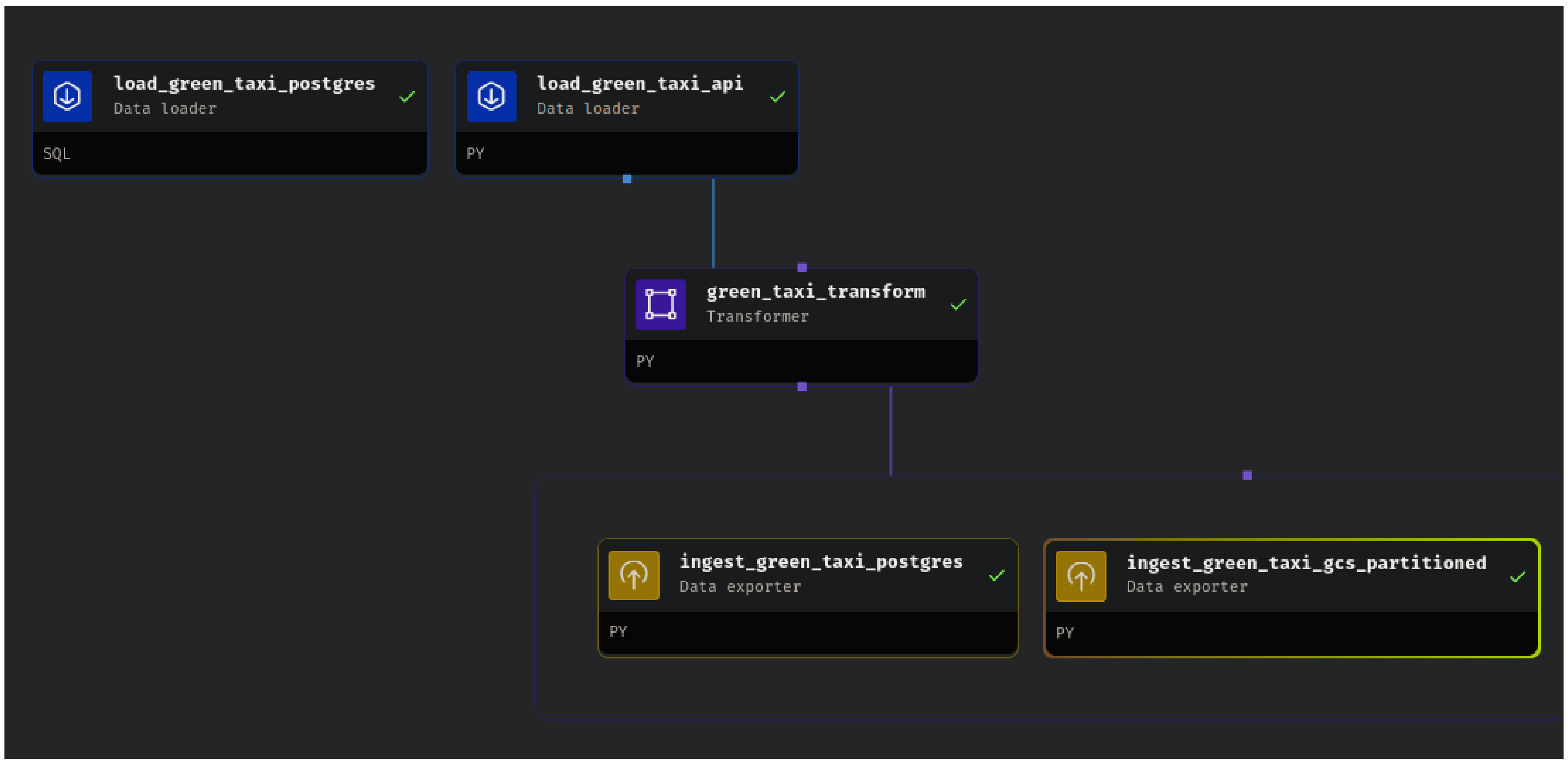
```
# Google
  GOOGLE_SERVICE_ACC_KEY_FILEPATH: "/home/src/mage-413610-2ec7db0afcf7.json"
  GOOGLE_LOCATION: US # Optional
```

3.1 Export data to Google Cloud Storage

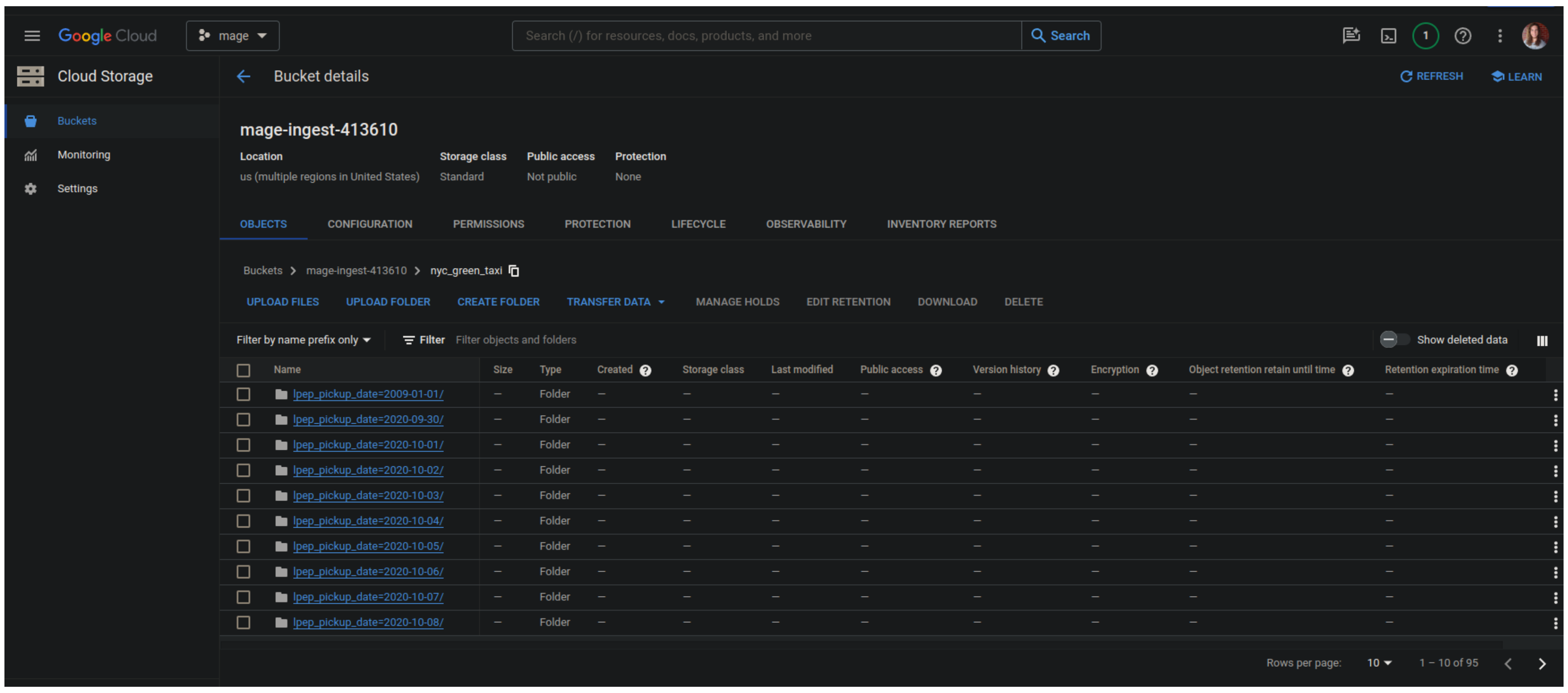
We can now use the created pipeline `green_taxi_etl` to add a data exporter block named `ingest_green_taxi_gcs` to export the data to Google Cloud Storage. In the green_taxi_etl pipeline we select `Python > Google Cloud Storage` when creating the data exporter. The code for the block is:

```python
from mage_ai.settings.repo import get_repo_path
  from mage_ai.io.config import ConfigFileLoader
  from mage_ai.io.google_cloud_storage import GoogleCloudStorage
  from pandas import DataFrame
  from os import path
  import pyarrow as pa
  import pyarrow.parquet as pq
  import os

  if 'data_exporter' not in globals():
      from mage_ai.data_preparation.decorators import data_exporter
  # Set the environment variable for Google Cloud authentication
  os.evitoment['GOOGLE_APPLICATION_CREDENTIALS'] = '/home/src/mage-413610-2ec7db0afcf7.json'

  # Define the bucket name, project ID, and table name for Google Cloud Storage
  bucket_name = 'mage-ingest-413610'
  project_id = '413610'
  table_name = 'nyc_green_taxi'

  # Construct the root path for the data to be stored in GCS
  # GCS follow file system structure gs://bucket_name/root_path
  root_path = f'{bucket_name}/{table_name}

  @data_exporter
  def export_data_to_google_cloud_storage(df: DataFrame, **kwargs) -> None:
      """
      Template for exporting data to a Google Cloud Storage bucket.
      Specify your configuration settings in 'io_config.yaml'.

      Docs: https://docs.mage.ai/design/data-loading#googlecloudstorage
      """
      # Convert the DataFrame to a PyArrow Table for efficient storage
      table = pa.Table.from_pandas(data)
      # Initialize Google Cloud Storage file syste
      gcs = pa.fs.GcsFileSystem()

      # Write the Table as a Parquet dataset to the specified path in GCS, partitioned by
'lpep_pickup_datetime'
      pq.write_to_dataset(
          table,
          root_path = root_path,
          partition_cols =  ['lpep_pickup_date'],
          filesystem = gcs
      )
```

We should have a tree for this pipeline as follows:



The data in this pipeline is being loaded from an API, transformed to handle some inconsistencies in the dataset and then exported to Google Cloud Storage and Postgres database. We then load the data from Postgres to check if the data was correctly ingested in the block `load_green_taxi_postgres`. To check the google cloud storage we should see the ingested data partitioned by date in the `google cloud storage > bucket` as follows:



mage-ingest-413610

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Standard	Not public	None

Buckets > mage-ingest-413610 > nyc_green_taxi

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Object retention retain until time	Retention expiration time
lpep_pickup_date=2009-01-01/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-09-30/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-01/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-02/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-03/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-04/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-05/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-06/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-07/	–	Folder	–	–	–	–	–	–	–	–
lpep_pickup_date=2020-10-08/	–	Folder	–	–	–	–	–	–	–	–

Rows per page: 10 1 – 10 of 95

The data has a total of 96 parquet files, one for each date. This is a good practice to optimize the query performance when we need to access the data and ingest a large dataset.

4. Homework

- Question 1. Data Loading

Once the dataset is loaded, what's the shape of the data?

- **266,855 rows x 20 columns**
- 544,898 rows x 18 columns
- 544,898 rows x 20 columns
- 133,744 rows x 20 columns

Question 2. Data Transformation

Upon filtering the dataset where the passenger count is greater than 0 *and* the trip distance is greater than zero, how many rows are left?

- 544,897 rows
- 266,855 rows
- **139,370 rows**
- 266,856 rows

- Question 3. Data Transformation

Which of the following creates a new column `lpep_pickup_date` by converting `lpep_pickup_datetime` to a date?

- `data = data['lpep_pickup_datetime'].date`
- `data('lpep_pickup_date') = data['lpep_pickup_datetime'].date`
- **`data['lpep_pickup_date'] = data['lpep_pickup_datetime'].dt.date`**
- `data['lpep_pickup_date'] = data['lpep_pickup_datetime'].dt().date()`

- Question 4. Data Transformation

What are the existing values of `VendorID` in the dataset?

- 1, 2, or 3
- **1 or 2**
- 1, 2, 3, 4
- 1

- Question 5. Data Transformation

How many columns need to be renamed to snake case?

- 3
- 6
- 2
- **4**

- Question 6. Data Exporting

Once exported, how many partitions (folders) are present in Google Cloud?

- **96**
- 56
- 67
- 108