Module 4 Homework

For this homework, you will need the following datasets:

- Green Taxi dataset (2019 and 2020)
- Yellow Taxi dataset (2019 and 2020)
- For Hire Vehicle dataset (2019)

web_to_gcs.py:

```
import os
import requests
from google.cloud import storage

"""
Pre-reqs:
1. `pip install pandas google-cloud-storage`
2. Set GOOGLE_APPLICATION_CREDENTIALS to your project/service-account key
3. Set GCP_GCS_BUCKET as your bucket or change default value of BUCKET
"""

# 初始化下载链接
init_url = 'https://github.com/DataTalksClub/nyc-tlc-data/releases/download/'
# 设置 GCS Bucket 名称
BUCKET = "database_sliu"

def upload_to_gcs(bucket_name, object_name, local_file):
    """
    上传文件到 GCS
    Ref: https://cloud.google.com/storage/docs/uploading-objects#storage-upload-object-python
    """
    CREDENTIALS_FILE = "gcs.json"
    client = storage.Client.from_service_account_json(CREDENTIALS_FILE)
    bucket = client.bucket(bucket_name)
    blob = bucket.blob(object_name)
    blob.upload_from_filename(local_file)
    print(f"Uploaded to GCS: gs://{bucket_name}/{object_name}")

def web_to_gcs(year, service):
    for i in range(12):
        month = f"{i+1:02d}"  # 格式化月份，确保两位数
        file_name = f"{service}_tripdata_{year}-{month}.csv.gz"
        request_url = f"{init_url}{service}/{file_name}"

        # 下载文件
        print(f"Downloading: {request_url}")
        r = requests.get(request_url)

        if r.status_code == 200:
            with open(file_name, 'wb') as f:
                f.write(r.content)
            print(f"Downloaded: {file_name}")
        else:
            print(f"Failed to download {file_name}, status code:
{r.status_code}")
            continue  # 如果下载失败，跳过当前月份

        # 直接上传 .csv.gz 到 GCS
        gcs_path = f"{service}/{file_name}"
        upload_to_gcs(BUCKET, gcs_path, file_name)

        # 删除本地文件
        os.remove(file_name)
        print(f"Deleted local file: {file_name}")

# 调用函数下载并上传到 GCS
# web_to_gcs('2019', 'green')
# web_to_gcs('2020', 'green')
web_to_gcs('2019', 'fhv')
```

load data sql in GCP:

```
-- LOAD DATA INTO `dbt-learn-452007.trips_data_all.fhv_tripdata`
-- FROM FILES(
--   format = 'CSV',
--   uris = ['gs://database_sliu/fhv/*.csv.gz']
-- );

select count(*) from `dbt-learn-452007.trips_data_all.fhv_tripdata`;
```

Before you start

1. Make sure you, **at least**, have them in GCS with a External Table **OR** a Native Table - use whichever method you prefer to accomplish that (Workflow Orchestration with pandas-gbq, dlt for gcs, dlt for BigQuery, gsutil, etc)
2. You should have exactly `7,778,101` records in your Green Taxi table
3. You should have exactly `109,047,518` records in your Yellow Taxi table
4. You should have exactly `43,244,696` records in your FHV table
5. Build the staging models for green/yellow as shown in here
6. Build the dimension/fact for taxi_trips joining with `dim_zones` as shown in here

Note: If you don't have access to GCP, you can spin up a local Postgres instance and ingest the datasets above

Question 1: Understanding dbt model resolution

- database.schema.table
- myproject.raw_nyc_tripdata.ext_green_taxi

Provided you've got the following sources.yaml

#如果为设置，则设置为默认

```
version: 2

sources:
  - name: raw_nyc_tripdata
    database: "{{ env_var('DBT_BIGQUERY_PROJECT', 'dtc_zoomcamp_2025') }}"
    schema:   "{{ env_var('DBT_BIGQUERY_SOURCE_DATASET', 'raw_nyc_tripdata') }}"
    tables:
      - name: ext_green_taxi
      - name: ext_yellow_taxi
```

with the following env variables setup where `dbt` runs:

```
export DBT_BIGQUERY_PROJECT=myproject
export DBT_BIGQUERY_DATASET=my_nyc_tripdata
```

What does this .sql model compile to?

```
select *
from {{ source('raw_nyc_tripdata', 'ext_green_taxi' ) }}
```

- `select * from dtc_zoomcamp_2025.raw_nyc_tripdata.ext_green_taxi`
- `select * from dtc_zoomcamp_2025.my_nyc_tripdata.ext_green_taxi`
- **`select * from myproject.raw_nyc_tripdata.ext_green_taxi`**
- `select * from myproject.my_nyc_tripdata.ext_green_taxi`
- `select * from dtc_zoomcamp_2025.raw_nyc_tripdata.green_taxi`

Question 2: dbt Variables & Dynamic Models

Say you have to modify the following dbt_model (`fct_recent_taxi_trips.sql`) to enable Analytics Engineers to dynamically control the date range.

- In development, you want to process only **the last 7 days of trips**
- In production, you need to process **the last 30 days** for analytics

```
select *
from {{ ref('fact_taxi_trips') }}
where pickup_datetime >= CURRENT_DATE - INTERVAL '30' DAY
```

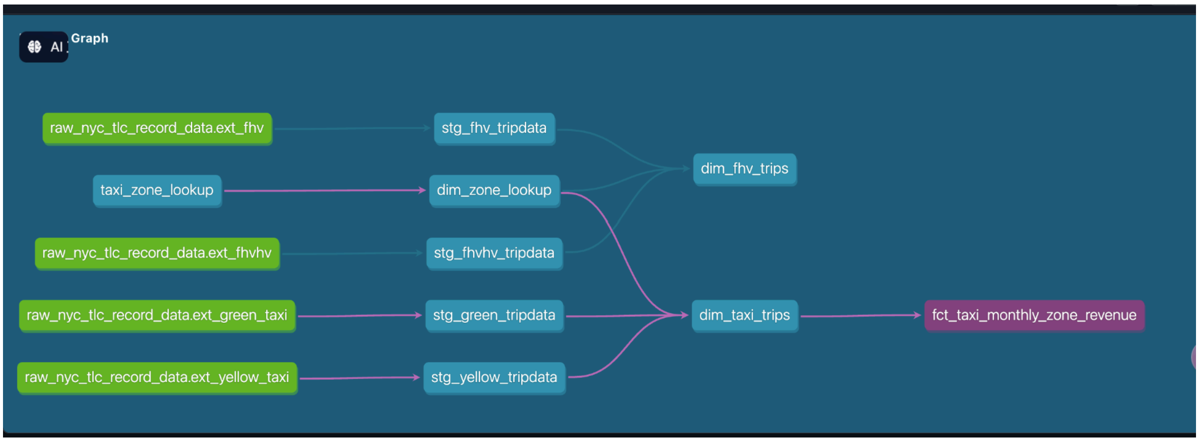
What would you change to accomplish that in a such way that command line arguments takes precedence over ENV_VARs, which takes precedence over DEFAULT value?

- Add `ORDER BY pickup_datetime DESC` and `LIMIT {{ var("days_back", 30) }}`
- Update the WHERE clause to `pickup_datetime >= CURRENT_DATE - INTERVAL '{{ var("days_back", 30) }}' DAY`

- Update the WHERE clause to `pickup_datetime >= CURRENT_DATE - INTERVAL '{{ env_var("DAYS_BACK", "30") }}' DAY`
- **Update the WHERE clause to** `pickup_datetime >= CURRENT_DATE - INTERVAL '{{ var("days_back", env_var("DAYS_BACK", "30")) }}' DAY`
- Update the WHERE clause to `pickup_datetime >= CURRENT_DATE - INTERVAL '{{ env_var("DAYS_BACK", var("days_back", "30")) }}' DAY`

Question 3: dbt Data Lineage and Execution

Considering the data lineage below **and** that taxi_zone_lookup is the **only** materialization build (from a .csv seed file):


AI
Graph
raw_nyc_tlc_record_data.ext_fhv
stg_fhv_tripdata
taxi_zone_lookup
dim_zone_lookup
dim_fhv_trips
raw_nyc_tlc_record_data.ext_fhvhv
stg_fhvhv_tripdata
raw_nyc_tlc_record_data.ext_green_taxi
stg_green_tripdata
dim_taxi_trips
fct_taxi_monthly_zone_revenue
raw_nyc_tlc_record_data.ext_yellow_taxi
stg_yellow_tripdata

Select the option that does **NOT** apply for materializing `fct_taxi_monthly_zone_revenue`:

- `dbt run`
- `dbt run --select +models/core/dim_taxi_trips.sql+ --target prod`
- `dbt run --select +models/core/fct_taxi_monthly_zone_revenue.sql`
- `dbt run --select +models/core/`
- `dbt run --select models/staging/+`

Question 4: dbt Macros and Jinja

Consider you're dealing with sensitive data (e.g.: PII), that is **only available to your team and very selected few individuals**, in the `raw layer` of your DWH (e.g: a specific BigQuery dataset or PostgreSQL schema),

- Among other things, you decide to obfuscate/masquerade that data through your staging models, and make it available in a different schema (a `staging layer`) for other Data/Analytics Engineers to explore
- And **optionally**, yet another layer (`service layer`), where you'll build your dimension (`dim_`) and fact (`fct_`) tables (assuming the Star Schema dimensional modeling) for Dashboarding and for Tech Product Owners/Managers

You decide to make a macro to wrap a logic around it:

```
{% macro resolve_schema_for(model_type) -%}

    {%- set target_env_var = 'DBT_BIGQUERY_TARGET_DATASET'  -%}
    {%- set stging_env_var = 'DBT_BIGQUERY_STAGING_DATASET' -%}

    {%- if model_type == 'core' -%} {{- env_var(target_env_var) -}}
    {%- else -%}                    {{- env_var(stging_env_var,
env_var(target_env_var)) -}}
    {%- endif -%}

{%- endmacro %}
```

And use on your staging, dim_ and fact_ models as:

```
{{ config(
    schema=resolve_schema_for('core'),
) }}
```

That all being said, regarding macro above, **select all statements that are true to the models using it**:(I select the false one)

- Setting a value for `DBT_BIGQUERY_TARGET_DATASET` env var is mandatory, or it'll fail to compile
- **Setting a value for `DBT_BIGQUERY_STAGING_DATASET` env var is mandatory, or it'll fail to compile**
- When using `core`, it materializes in the dataset defined in `DBT_BIGQUERY_TARGET_DATASET`
- When using `stg`, it materializes in the dataset defined in `DBT_BIGQUERY_STAGING_DATASET`, or defaults to `DBT_BIGQUERY_TARGET_DATASET`
- When using `staging`, it materializes in the dataset defined in `DBT_BIGQUERY_STAGING_DATASET`, or defaults to `DBT_BIGQUERY_TARGET_DATASET`

Serious SQL

Alright, in module 1, you had a SQL refresher, so now let's build on top of that with some serious SQL.

These are not meant to be easy - but they'll boost your SQL and Analytics skills to the next level. So, without any further do, let's get started...

You might want to add some new dimensions `year` (e.g.: 2019, 2020), `quarter` (1, 2, 3, 4), `year_quarter` (e.g.: `2019/Q1`, `2019-Q2`), and `month` (e.g.: 1, 2, ..., 12), **extracted from pickup_datetime**, to your `fct_taxi_trips` OR `dim_taxi_trips.sql` models to facilitate filtering your queries

Question 5: Taxi Quarterly Revenue Growth

1. Create a new model `fct_taxi_trips_quarterly_revenue.sql`
2. Compute the Quarterly Revenues for each year for based on `total_amount`
3. Compute the Quarterly YoY (Year-over-Year) revenue growth

- e.g.: In 2020/Q1, Green Taxi had -12.34% revenue growth compared to 2019/Q1

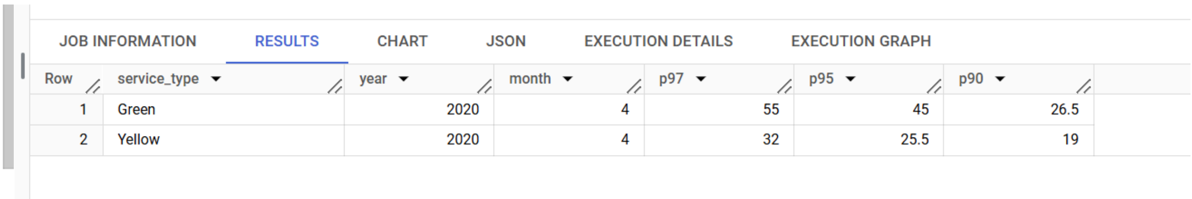
- e.g.: In 2020/Q4, Yellow Taxi had +34.56% revenue growth compared to 2019/Q4

Considering the YoY(Year over Year) Growth in 2020, which were the yearly quarters with the best (or less worse) and worst results for green, and yellow

- green: {best: 2020/Q2, worst: 2020/Q1}, yellow: {best: 2020/Q2, worst: 2020/Q1}
- green: {best: 2020/Q2, worst: 2020/Q1}, yellow: {best: 2020/Q3, worst: 2020/Q4}
- green: {best: 2020/Q1, worst: 2020/Q2}, yellow: {best: 2020/Q2, worst: 2020/Q1}
- **green: {best: 2020/Q1, worst: 2020/Q2}, yellow: {best: 2020/Q1, worst: 2020/Q2}**
- green: {best: 2020/Q1, worst: 2020/Q2}, yellow: {best: 2020/Q3, worst: 2020/Q4}

```
-- select service_type, year_quarter, sum(total_amount)
-- from `dbt_sliu.fact_trips`
-- where extract(year from pickup_datetime) in (2019, 2020)
-- group by service_type, year_quarter;
```

Question 6: P97/P95/P90 Taxi Monthly Fare



Row	service_type	year	month	p97	p95	p90
1	Green	2020	4	55	45	26.5
2	Yellow	2020	4	32	25.5	19

```
with filtered_trips as (
    select
        service_type,
        extract(year from pickup_datetime) as year,
        extract(month from pickup_datetime) as month,
        fare_amount
    from `dbt_sliu.fact_trips`
    where fare_amount > 0
        and trip_distance > 0
        and payment_type_description in ('Cash', 'Credit card')
),
percentiles as (
    select
        service_type,
        year,
        month,
        approx_quantiles(fare_amount, 100)[SAFE_OFFSET(97)] as p97,
        approx_quantiles(fare_amount, 100)[SAFE_OFFSET(95)] as p95,
        approx_quantiles(fare_amount, 100)[SAFE_OFFSET(90)] as p90
    from filtered_trips
    group by service_type, year, month
)
select *
from percentiles
where year = 2020 and month = 4;
```

1. Create a new model `fct_taxi_trips_monthly_fare_p95.sql`
2. Filter out invalid entries (`fare_amount > 0`, `trip_distance > 0`, and `payment_type_description in ('Cash', 'Credit Card')`)
3. Compute the **continous percentile** of `fare_amount` partitioning by service_type, year and and month

Now, what are the values of `p97`, `p95`, `p90` for Green Taxi and Yellow Taxi, in April 2020?

- green: {p97: 55.0, p95: 45.0, p90: 26.5}, yellow: {p97: 52.0, p95: 37.0, p90: 25.5}
- **green: {p97: 55.0, p95: 45.0, p90: 26.5}, yellow: {p97: 31.5, p95: 25.5, p90: 19.0}**
- green: {p97: 40.0, p95: 33.0, p90: 24.5}, yellow: {p97: 52.0, p95: 37.0, p90: 25.5}
- green: {p97: 40.0, p95: 33.0, p90: 24.5}, yellow: {p97: 31.5, p95: 25.5, p90: 19.0}
- green: {p97: 55.0, p95: 45.0, p90: 26.5}, yellow: {p97: 52.0, p95: 25.5, p90: 19.0}

Question 7: Top #Nth longest P90 travel time Location for FHV

Prerequisites:

- Create a staging model for FHV Data (2019), and **DO NOT** add a deduplication step, just filter out the entries where `where dispatching_base_num is not null`
- Create a core model for FHV Data (`dim_fhv_trips.sql`) joining with `dim_zones`. Similar to what has been done here
- Add some new dimensions `year` (e.g.: 2019) and `month` (e.g.: 1, 2, ..., 12), based on `pickup_datetime`, to the core model to facilitate filtering for your queries

Now...

1. Create a new model `fct_fhv_monthly_zone_traveltime_p90.sql`
2. For each record in `dim_fhv_trips.sql`, compute the timestamp_diff in seconds between dropoff_datetime and pickup_datetime - we'll call it `trip_duration` for this exercise
3. Compute the **continous** `p90` of `trip_duration` partitioning by year, month, pickup_location_id, and dropoff_location_id

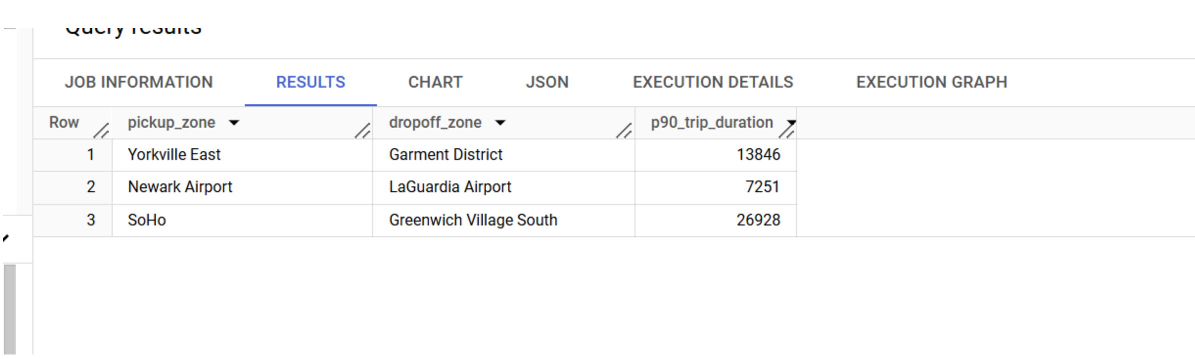
For the Trips that **respectively** started from `Newark Airport`, `SoHo`, and `Yorkville East`, in November 2019, what are **dropoff_zones** with the 2nd longest p90 trip_duration ?

- **LaGuardia Airport, Chinatown, Garment District**
- LaGuardia Airport, Park Slope, Clinton East
- LaGuardia Airport, Saint Albans, Howard Beach
- LaGuardia Airport, Rosedale, Bath Beach
- LaGuardia Airport, Yorkville East, Greenpoint

```
WITH trip_p90 AS (
    SELECT
        pickup_zone,
        dropoff_zone,
        APPROX_QUANTILES(trip_duration, 100)[OFFSET(90)] AS p90_trip_duration
    FROM `dbt_sliu.dim_fhv_trips`
    WHERE year = 2019
        AND month = 11
        AND pickup_zone IN ('Newark Airport', 'SoHo', 'Yorkville East')
    GROUP BY pickup_zone, dropoff_zone
),

ranked_trips AS (
    SELECT *,
        RANK() OVER (PARTITION BY pickup_zone ORDER BY p90_trip_duration DESC) AS
rnk
    FROM trip_p90
)

SELECT pickup_zone, dropoff_zone, p90_trip_duration
FROM ranked_trips
WHERE rnk = 2;
```



Row	pickup_zone	dropoff_zone	p90_trip_duration
1	Yorkville East	Garment District	13846
2	Newark Airport	LaGuardia Airport	7251
3	SoHo	Greenwich Village South	26928

Submitting the solutions

- Form for submitting: https://courses.datatalks.club/de-zoomcamp-2025/homework/hw4

Solution

- To be published after deadline