ETL for CIS 9440 Data Warehousing and Analytics

- Project title: NYC Motor Vehicle Collision Transparency Data Warehouse Project
- Final Project Milestone 5 selected code chunks
- Group Number: 5
- Student(s): Gabriel Fernandez, Jason Jiang

ETL - Extract data

Show code

Show hidden output

Show code

Mount Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# import libraries
import pandas as pd
import numpy as np
from sodapy import Socrata
from google.cloud import bigquery
from google.oauth2 import service_account
pd.options.mode.chained_assignment = None  # default='warn'
import time

from tqdm.notebook import tqdm_notebook # to show progress bar
from IPython.display import Image # to attach images
```

Data sets

Dataset 1: Motor Vehicle Collisions - Crashes

he Motor Vehicle Collisions crash table contains details on the crash event. Each row represents a crash event. The Motor Vehicle Collisions data tables contain information from all police reported motor vehicle collisions in NYC.

https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95

Dataset 2: Motor Vehicle Collisions - Person

The Motor Vehicle Collisions person table contains details for people involved in the crash. Each row represents a person (driver, occupant, pedestrian, bicyclist,..) involved in a crash. The data in this table goes back to April 2016 when crash reporting switched to an electronic system.
https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Person/f55k-p6yu

---

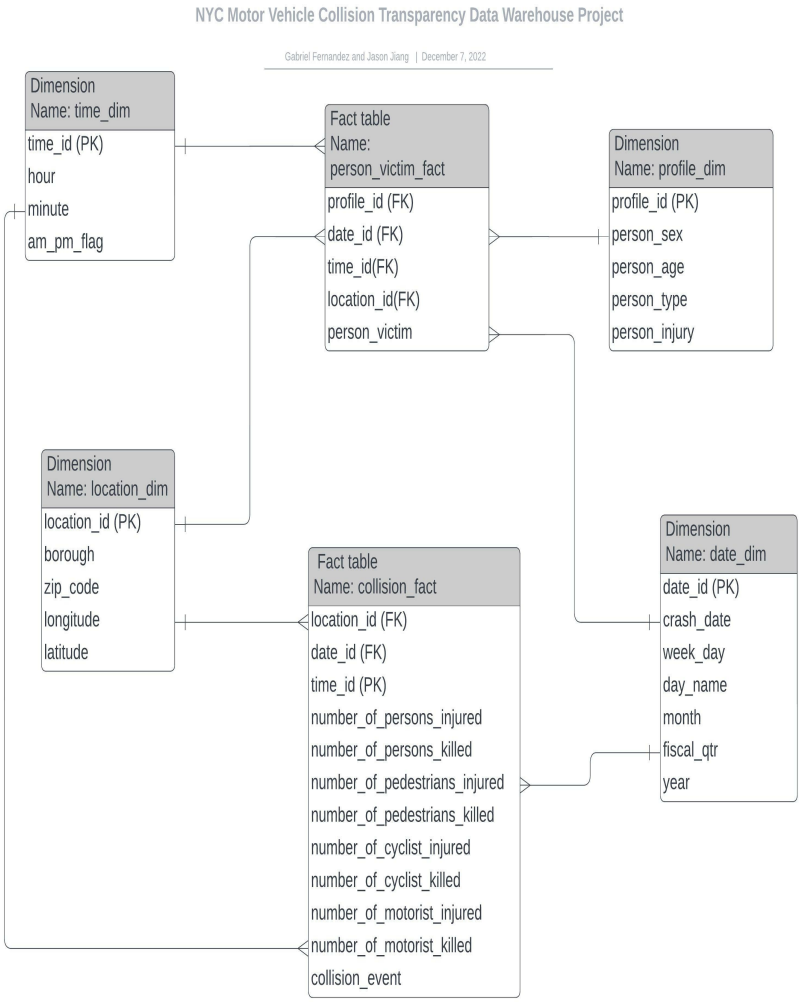
- Get your app-token from: https://data.cityofnewyork.us/profile/edit/developer_settings

Dimensional model

https://lucid.app/lucidchart/d42f0e3b-891b-49d3-9486-6ffabdc2f6d8/edit?page=0_0&invitationId=inv_b85589a9-5172-40f5-8ca8-450d9098461b#

We used Python.display to attach images to our Jupyter notebook

```
Image("/content/drive/MyDrive/project_DW/dimesional_model.jpeg",width=1000, height=800)
```



We used tqdm to add a progress bars to some of the ETL functions

- Colorful progress bar to track a loop in Python
- Nested progress bars for Nested loops in Python
- Working with Pandas (progress_apply)
- **Working with a while loop and unknown increments**
- Downloading and uploading files in Python with progress bars
- Multiprocessing and Threads (Linux and Windows)
- Machine Learning libraries (Keras and Tensorflow)

Other examples: https://medium.com/@harshit4084/track-your-loop-using-tqdm-7-ways-progress-bars-in-python-make-things-easier-fcbbb9233f24

Setting up ETL

```
[ ]  ↳ 8 cells hidden
```

Extract data

```
[ ]  ↳ 4 cells hidden
```

Extract_socrata_data with progress bar

```
# Fetch all rows for data set1

#data_set1 = 'h9gi-nx95'

data1 = extract_socrata_data(chunk_size = 10000,
                             data_set = data_set1)
```

while loop: 1950000/? [04:43<00:00, 5182.70it/s]

```
Loop completed
Loop took 283.3 seconds
Transforming to pandas.DataFrame took 9.9 seconds
The shape of your dataframe is: (1949630, 29)
```

Show code

Show hidden output

```
# Fetch all rows for data set2
#data_set2 = 'f55k-p6yu'

data2 = extract_socrata_data(data_set = data_set2,
                             chunk_size = 50000)
```

while loop: 4900000/? [03:57<00:00, 17312.89it/s]

```
Loop completed
Loop took 240.5 seconds
Transforming to pandas.DataFrame took 18.0 seconds
The shape of your dataframe is: (4878868, 21)
```

Show code

Show hidden output

Show code

Show hidden output

Merge data

[ ] *3 cells hidden*

ETL - Transform data

Data profiling

1. Distinct values per column
2. Null values per column
3. Summary statistics per numeric column

Show code

Show hidden output

Show code

Show code

Easy to use

for column in data.columns:

for column in **tqdm_notebook**(data.columns):

```
# view your data profiling dataframe
#RUN DATA PROFILING FUNCTION HERE
data_profiling_df = create_data_profiling_df(data = data)
data_profiling_df
```

Show code

Data cleaning

[ ] ↳ *2 cells hidden*

Drop duplicates

We created functions for code chucks that we reuse often

> Indented block

```
def drop_dupli(data):

  #check number of rows
  print(f"number of rows before dropping duplicates: {len(data)}")
  #check for duplciates
  print(f"number of duplicate rows: {len(data[data.duplicated()])}")
  #drop duplicate rows based on entire row
  data = data.drop_duplicates(keep = 'first')
  print(f"number of rows after duplicates dropped: {len(data)}")

  return data
```

```
# drop duplicates
data_sin_du = drop_dupli(data)
```

Show code

Show code

Check for outliers

Show code

We used descriptive statistics to identify outliers

Show code

Double-click (or enter) to edit

```
data.shape
```

```
 # Filter out person_age < 0 and > 120
 data= data[(0 < data["person_age"]) & (data["person_age"] < 120)].copy()
```

```
data["person_age"].describe().T
```

```
data.shape
```

Create location dimension

[ ] ↳ *7 cells hidden*

Create date dimension

[ ] ↳ *9 cells hidden*

Create time dimension

[ ] ↳ *10 cells hidden*

Create profile dimension

[ ] ↳ *8 cells hidden*

Create collision fact table

[ ] ↳ *12 cells hidden*

Create person_victim fact table

[ ] ↳ *12 cells hidden*

ETL - Load data

Deliver Facts and Dimensions to Data Warehouse (BigQuery)

```
# create a function to load dataframes to BigQuery

def load_table_to_bigquery(df,
                           table_name,
                           dataset_id):

    dataset_id = dataset_id #change dataset id to match your project id

    dataset_ref = bigquery_client.dataset(dataset_id)
    job_config = bigquery.LoadJobConfig()
    job_config.autodetect = True
    job_config.write_disposition = "WRITE_TRUNCATE"

    upload_table_name = f"{dataset_id}.{table_name}"

    load_job = bigquery_client.load_table_from_dataframe(df,
                                            upload_table_name,
                                            job_config = job_config)
```

```
    print(f"completed job {load_job} for table {table_name}")
```

We used a loop to load all our tables to BigQuery

```
#Load each table to BigQuery: "collision_fact", "person_victim_fact", date_dim", "location_dim", "time_dim", and "person_dim."

#create an object with each table and their names
tables_objects = [[collision_fact, "collision_fact"],[person_victim_fact, "person_victim_fact"],
                  [date_dim,"date_dim" ],[location_dim,"location_dim"], [time_dim,"time_dim"],
                  [profile_dim,"profile_dim"]
                  ]
```

```
#use a loop to load all the tables with the function "load_table_to_bigquery"
for table in tables_objects:
    load_table_to_bigquery(df = table[0],
                       table_name = table[1],
                       dataset_id = dataset_id)
```

```
completed job LoadJob<project=deft-stratum-361822, location=US, id=a5f9ed08-f7a7-4975-97f4-a2d116502398> for table collision_
completed job LoadJob<project=deft-stratum-361822, location=US, id=189581ed-a812-44dd-b940-d2c1df15fe3c> for table person_vic
completed job LoadJob<project=deft-stratum-361822, location=US, id=1ee4075a-6f9a-447b-be44-e76365f5c8c3> for table date_dim
completed job LoadJob<project=deft-stratum-361822, location=US, id=b11d5b45-e9cf-4082-8e1d-bf3db67b4012> for table location_d
completed job LoadJob<project=deft-stratum-361822, location=US, id=a869e488-dd61-4724-b78f-0c2b99208cf4> for table time_dim
completed job LoadJob<project=deft-stratum-361822, location=US, id=090c2a26-b208-4144-92b5-5326abc3368d> for table profile_di
```

```
notebook_end_time = time.time()

total_notebook_time = round(notebook_end_time - notebook_start_time, 1)
print(f"Running notebook took {total_notebook_time}seconds, {total_notebook_time/60} minutes")
```

```
Running notebook took 1243.9seconds, 20.73166666666667 minutes
```

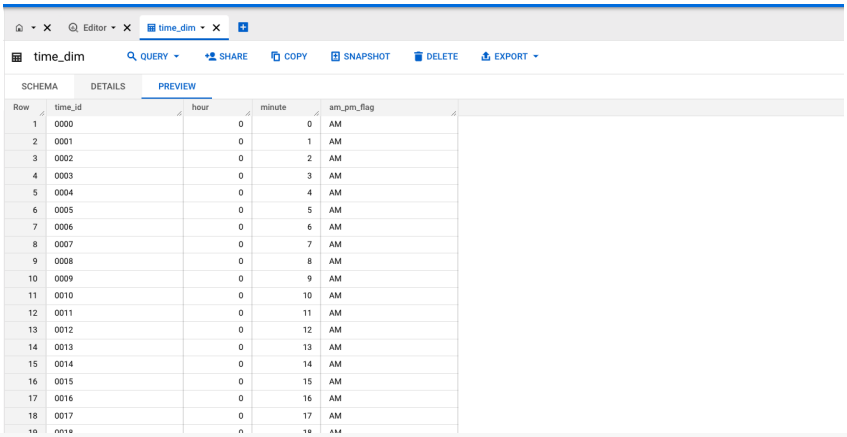
Screenshots of database tables

Date dimension

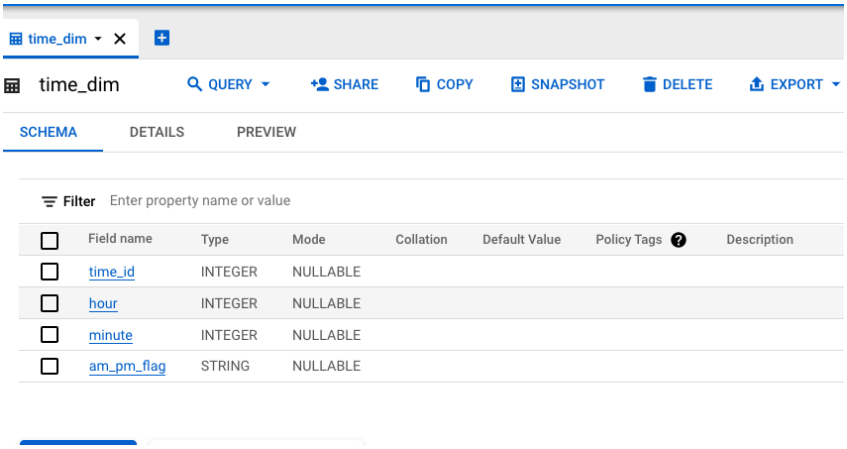
[ ] ↳ *2 cells hidden*

Time dimension

```
Image("/content/drive/MyDrive/project_DW/time_dim.png")
```




```
Image("/content/drive/MyDrive/project_DW/time_dim_types.png")
```



Location dimension

[ ] ↳ *2 cells hidden*

Profile dimension

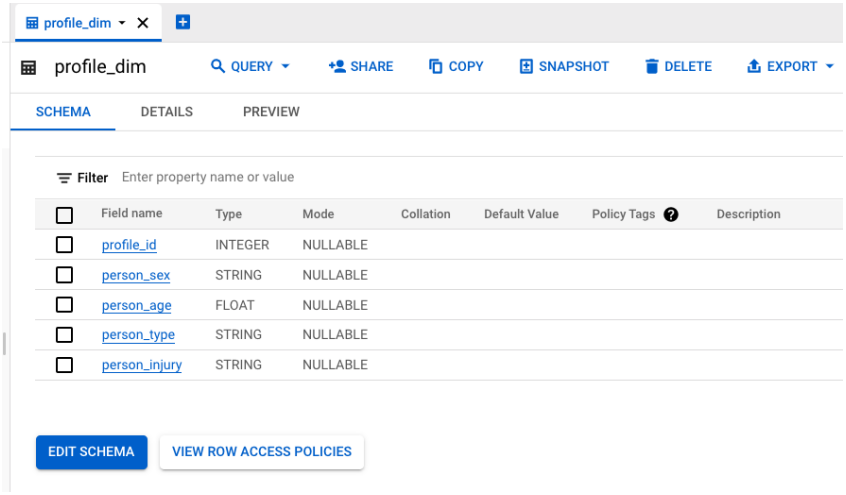
```
Image("/content/drive/MyDrive/project_DW/profile_dim.png")
```

profile_dim QUERY SHARE COPY SNAPSHOT DELETE EXPORT

SCHEMA DETAILS PREVIEW

Row	profile_id	person_sex	person_age	person_type	person_injury
1	200	F	1	Occupant	Unspecified
2	238	F	1	Occupant	Injured
3	846	F	1	Pedestrian	Injured
4	1077	F	1	Bicyclist	Injured
5	1167	F	1	Pedestrian	Unspecified
6	1632	F	1	Pedestrian	Killed
7	214	F	2	Occupant	Unspecified
8	466	F	2	Pedestrian	Injured
9	621	F	2	Occupant	Injured
10	869	F	2	Other Motorized	Injured
11	940	F	2	Pedestrian	Unspecified
12	1042	F	2	Bicyclist	Injured
13	1397	F	2	Pedestrian	Killed
14	1610	F	2	Bicyclist	Unspecified
15	85	F	3	Occupant	Unspecified
16	145	F	3	Occupant	Injured
17	448	F	3	Pedestrian	Injured
18	1172	F	3	Pedestrian	Unspecified
19	1734	F	3	Bicyclist	Injured
20	2051	F	3	Occupant	Killed

```
Image("/content/drive/MyDrive/project_DW/profile_dim_types.png")
```



Collision_fact table

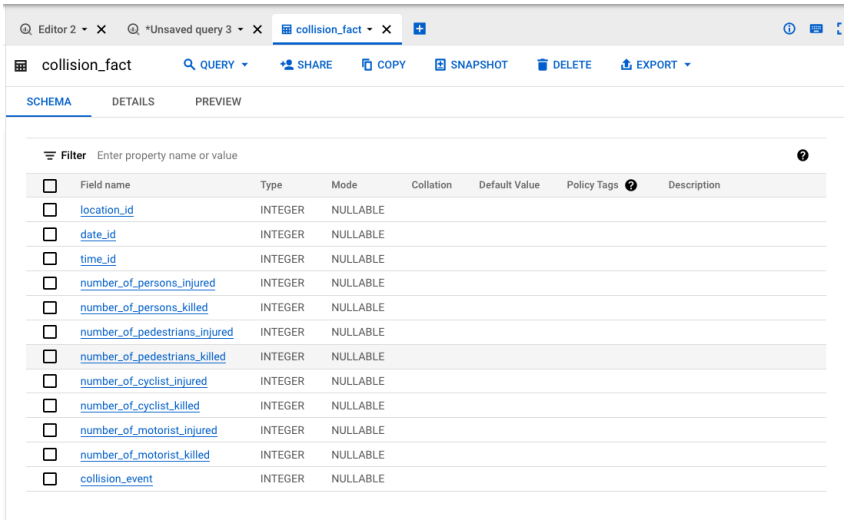
```
Image("/content/drive/MyDrive/project_DW/collision_fact.png")
```

collision_fact QUERY SHARE COPY SNAPSHOT DELETE EXPORT

SCHEMA DETAILS PREVIEW

Row	location_id	date_id	time_id	number_of_pers	number_of_pers	number_of_ped	number_of_ped	number_of_cycli	number_of_cycli
1	21	20220424	0	0	0	0	0	0	0
2	100	20220325	0	0	0	0	0	0	0
3	157	20210911	0	0	0	0	0	0	0
4	262	20210709	0	0	0	0	0	0	0
5	322	20210414	0	0	0	0	0	0	0
6	345	20210414	0	0	0	0	0	0	0
7	353	20210415	0	0	0	0	0	0	0
8	433	20210416	0	0	0	0	0	0	0
9	448	20210415	0	0	0	0	0	0	0
10	489	20210414	0	0	0	0	0	0	0
11	537	20210414	0	0	0	0	0	0	0
12	687	20210416	0	0	0	0	0	0	0

```
Image("/content/drive/MyDrive/project_DW/collision_fact_types.png")
```



Person_victim_fact table

[ ] *2 cells hidden*

References

- ETL Pipeline tutorial by Michael O'Donnell (CIS 9440 Data Warehousing and Analytics).
- Track your loop using tqdm: 7 ways progress bars in Python make things easier: https://medium.com/@harshit4084/track-your-loop-using-tqdm-7-ways-progress-bars-in-python-make-things-easier-fcbbb9233f24