This project focuses on leveraging New York City taxi cab data to develop a practical business intelligence solution, inspired by modern technologies. By diving into this extensive dataset, our goal is to extract valuable insights that can optimize operations, enhance user experience, and fuel business growth. We'll be diving deep into the data, working on extraction, transformation, and analysis to cater to specific business needs.

**Final Goal of the Project**

* Deliver actionable intelligent insights derived from the analysis of NYC taxi data.
* Streamlining decision-making processes by addressing key business intelligence questions and visualizing trends.
* Boost operational efficiency within the taxi service ecosystem for both users and drivers.
* Ensuring a smoother experience for all stakeholders through the implementation of data-driven solutions.

**Software Required**

* Python
* Tableau
* Google Colab

**Target DW Database**

* Postgresql

**Data sources**

* Main Source:
  + New york Taxi Cab Data (Green)
    - <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>
* External Data:
  + Geographical data of NYC
    - Data explanation: Coordinates of each zone in NYC
    - <https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc>
  + Taxi Zone Lookup Table (Names of Each Zone)
    - Data explanation: Name of each zone and Borough
    - <https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc>

**ETL**

**Overview**

The Initial ETL (Extract, Transform, Load) program is designed to populate essential dimension tables and initialize the data warehouse with foundational data for analyzing taxi trip records. It performs data extraction, transformation, and loading tasks to ensure the availability of reference data and necessary structures in the data warehouse environment. **This program (ETL.py) will only be run one time, any future update will be addressed by the incremental update program (increment.py)**

**Purpose**

The primary purpose of this program is to lay the groundwork for subsequent analysis and reporting on taxi trip data by creating dimension tables that provide context and categorization for the trip records. It extracts location, vendor, rate code, payment type, and trip type information from external sources, as well as current data from original data source for datetime dimension and fact table, transforms them into a structured format, and loads it into corresponding tables within the data warehouse.

**Prerequisites**

Access to a PostgreSQL database where the data warehouse resides.

Python environment with necessary packages (pandas, psycopg2, sqlalchemy) installed.

### **Section 1: One-Time ETL for Dimension Tables**

Location Dimension

* Extract Taxi Zone Lookup Table, convert to dataframe format (<https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc>)
* rename columns in the from the above data frame to match the desired column names for the dimension table
* duplicate the pickup location ID column and rename it as dropoff\_location\_id for future use
* generates all possible combinations of pickup and dropoff location IDs using the **product function** from **itertools**
* construct the location\_df DataFrame by merging pickup and dropoff location information with their respective zones and boroughs
* creates a new column location\_id by concatenating pickup and dropoff location IDs
* uploads the location\_df DataFrame to the PostgreSQL database table named location\_dim

Vendor Dimension/Ratecode Dimension/Payment Dimension/Trip Dimension

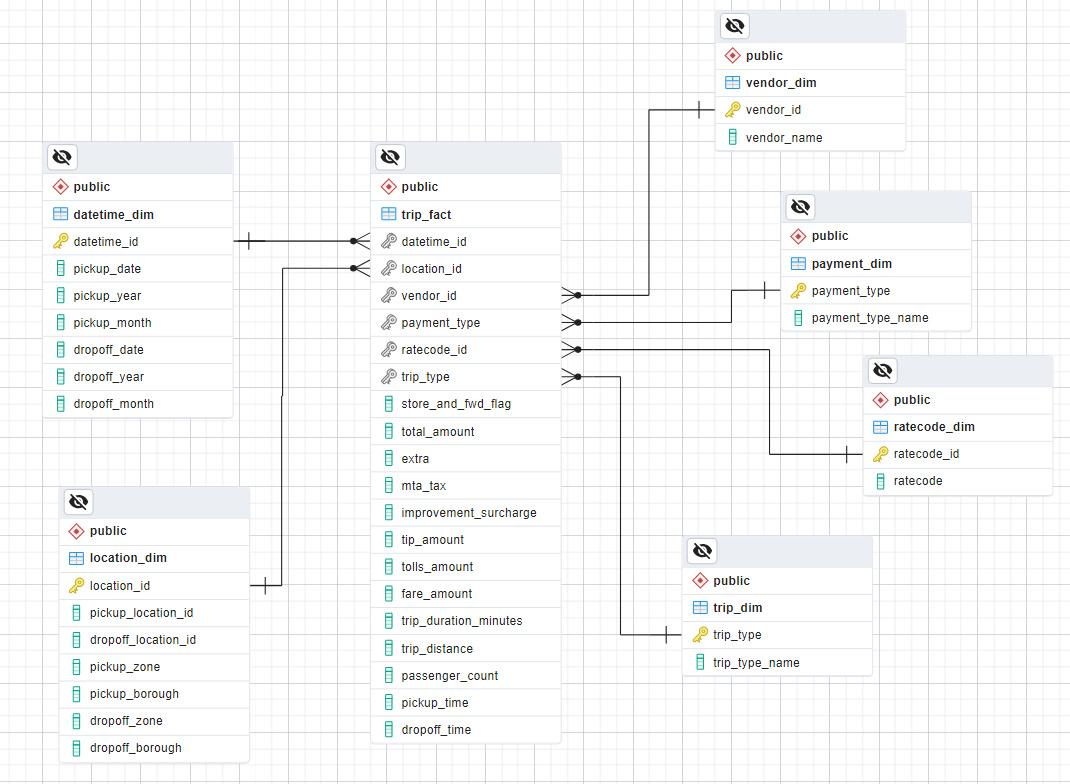
* Define the columns and their corresponding values for the dimension table
* Create a pandas DataFrame using the defined columns and values.
* The value '99' is chosen to represent cases where there is no valid value or where the value is missing in the dimension table.
* Use the to\_sql() method to upload the DataFrame to the corresponding table in the database.

### **Section 2: Loop ETL**

* + Extraction (Software: Python/Google Colab)
    - Extract data by month from data source main source
    - Data start date: 01/2015
    - Data end date:12/2023
    - Increment by one month until no more data is found
  + Transformations
    - Remove 50% of data from each month for faster computing without affecting data integrity
    - Drop error data
      * Negative fees in column “total\_amount”, “fare\_amount”, “extra”, “mta\_tax”, “improvement\_surchange”
      * Undefined code
      * Wrong dates
    - Rename columns for better readability:
      * “Ipep\_pickup\_datetime” to “pickup\_datetime”
      * “Ipep\_dropoff\_datetime” to “dropoff\_datetime”
      * “VendorID” to “vendor\_id”
      * “RatecodeID” to “ratecode\_id”
    - Datetime expansion
      * Transform “pickup\_datetime” to “pickup\_date”, “pickup\_month”, “pickup\_year”
      * Transform “dropoff\_datetime” to “dropoff\_date”, “dropoff\_month”, “dropoff\_year”
    - Calculate trip\_duration in minutes for easy of use
    - Construct fact table & datetime dimension table
      * Construct Keys:
        + datetime\_id: pickup\_date–dropoff\_date
        + location\_id: pickup\_location\_code–dropoff\_location\_code
  + Loading
    - Method 1: Google Colab(Recommend)
      * Connect to local drive before uploading to database
    - Method 2: Python
      * Warning: Loading time might be longer varied by GPU
  + Final result
    - After ETL process according to above process, fact table should have19 rows and around 33 millions rows
    - Datetime dimension table should have 7 columns and around 6500 rows
  + ***Full Detailed ETL script in Document (ETL.py)***

**Star Schema**

* + Fact table
    - Foreign keys
      * datetime\_id: pickup\_date–dropoff\_date
      * location\_id: pickup\_code–dropoff\_code
      * vendor\_id: vendor code
      * payment\_id: payment code
      * ratecode\_id: rate code
      * trip\_type: trip code
    - Numerical values:
      * pickup\_time: hr-min-sec (extracted from “lpep\_pickup\_datetime“)
      * dropoff\_time: hr-min-sec (extracted from “lpep\_dropoff\_datetime”)
      * Other given quantitative measurements
  + Dimension tables
    - Datetime Dimension
      * PK: datetime\_id (pickup\_date–dropoff\_date)
    - Location Dimension
      * PK: location\_id (pickup\_loc–dropoff\_loc)
    - Vendor Dimension
      * PK: vendor code
    - Payment Dimension
      * PK: payment code
    - Ratecode Dimension
      * PK: rate code
    - Trip Dimension
      * PK: trip code
  + ***Full Detailed Star schema Creation in database is in Document (fact&dim\_creation.sql)***
  + Star Schema Design Graph



**Incremental update of DW**

**Purpose**

This script (increment.py) is designed to perform a monthly incremental update to a data warehouse containing taxi trip data. It fetches the latest trip data from a specified URL, performs data cleaning and transformation, and updates the Datetime dimension and fact table in the data warehouse. **This should be run once a month.**

**Steps**

1. Database Connection:

* The script establishes a connection to the PostgreSQL database using the provided connection string.

1. Fetch Latest Month:

* Queries the datetime\_dim table in the data warehouse to retrieve the latest year and month for which data has been loaded.
* Increments the month for the next data fetch. If the month exceeds 12, it increments the year and resets the month to 1.

1. Data Extraction:

* Constructs the URL to fetch the latest trip data based on the incremented year and month.
* Reads the Parquet file from the URL into a pandas DataFrame.

1. Data Cleaning and Transformation

(identical to the ‘Transformations’ part of the original ETL)

* + - Remove 50% of data from each month for faster computing without affecting data integrity
    - Drop error data
      * + Negative fees in column “total\_amount”, “fare\_amount”, “extra”, “mta\_tax”, “improvement\_surchange”
        + Undefined code
        + Wrong dates
    - Rename columns for better readability:
      * + “Ipep\_pickup\_datetime” to “pickup\_datetime”
        + “Ipep\_dropoff\_datetime” to “dropoff\_datetime”
        + “VendorID” to “vendor\_id”
        + “RatecodeID” to “ratecode\_id”
    - Datetime expansion
      * + Transform “pickup\_datetime” to “pickup\_date”, “pickup\_month”, “pickup\_year”
        + Transform “dropoff\_datetime” to “dropoff\_date”, “dropoff\_month”, “dropoff\_year”
    - Calculate trip\_duration in minutes for easy of use
    - Construct fact table & datetime dimension table

datetime\_id: pickup\_date–dropoff\_date

location\_id: pickup\_location\_code–dropoff\_location\_code

1. Dimension and Fact Table Construction

* Constructs the datetime\_dim DataFrame containing unique combinations of pickup and dropoff dates, months, and years.
* Constructs the trip\_fact DataFrame containing trip records along with associated datetime and location IDs.

1. Data Upload

* Appends the datetime\_dim DataFrame to the datetime\_dim table in the database.
* Appends the trip\_fact DataFrame to the trip\_fact table in the database.

1. Cleanup

* Closes the database connection.

**Output**

1. Prints a message indicating a successful upload.
2. Displays the uploaded trip\_fact and datetime\_dim DataFrames for verification.
   * ***Full Detailed incremental update script in Document (increment.py)***

**Business Intelligence**

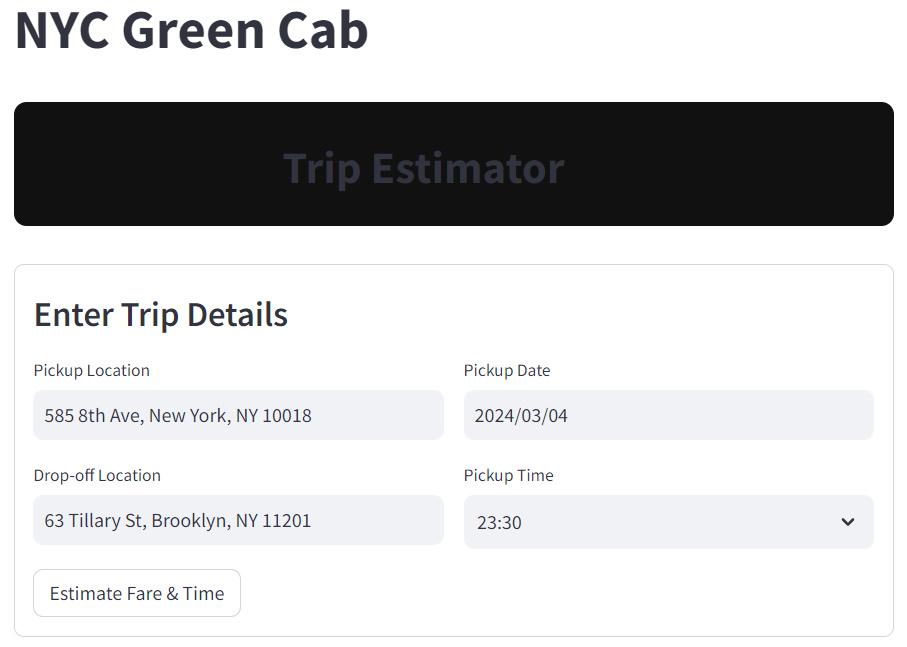
End users of DW

* + - Senior executives
    - Customers
    - Taxi Drivers

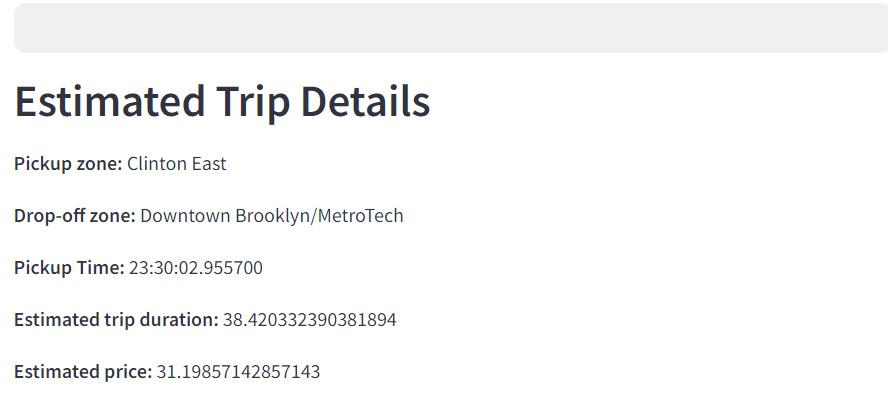
Business scenarios and associated SQL queries

1. **Estimate Trip Duration/Amount for Customers**
   * + - Provided a website for customers to get estimation of trip duration and trip fare based on the pick and drop off location they provide
       - Goal
         * Improving the taxi experience by doing so to increase customer satisfaction rate

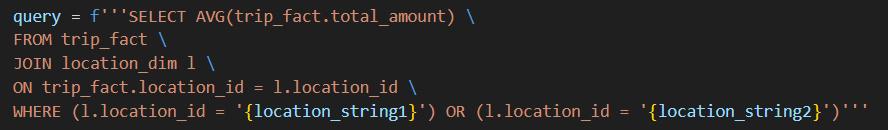
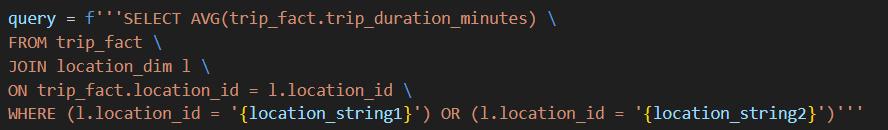
* Dependencies:
  + Streamlit
  + Pandas
  + SQLAlchemy
  + Requests
  + Shapely
  + DateTime
* Steps
  + Fetches taxi zone information from the NYC Taxi & Limousine Commission API
    - Function: process\_polygon\_string(data\_column)
  + Converts user-inputted addresses into latitude and longitude coordinates using the Radar.io geocoding API
    - Function: address\_coord(address)
  + Determine pickup and drop-off zones
    - Function: find\_zone(lon, lat)
  + Queries the PostgreSQL database to fetch relevant data for estimating trip duration and price
  + Calculates the average trip duration and price for the specified pickup and drop-off zones.
* ***Full program in Document (trip\_estimate.py)***
* Example Input:



* Example output:



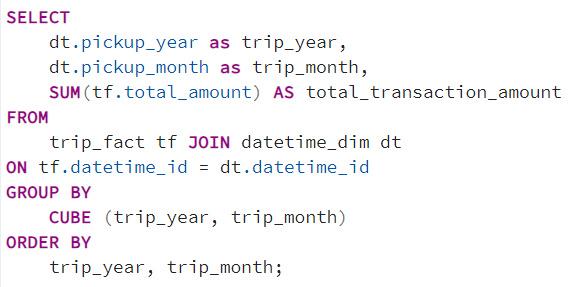
* Key Queries:



**Note:** This application will not work if no trip record of corresponding zones is found, in which case the output would be some error messages.

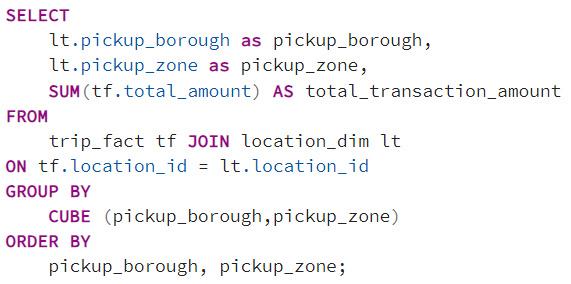
* **Examples address that could work, for the purpose of quick testing:**
  + 790 7th Ave, New York, NY 10019
  + 36-02 Ditmars Blvd, Queens, NY 11105

1. **Hourly/Seasonality/yearly trends Analysis**
   * + - Query code with OLAP roll-up
       - ***Full program in Document (OLAP\_query.sql)***



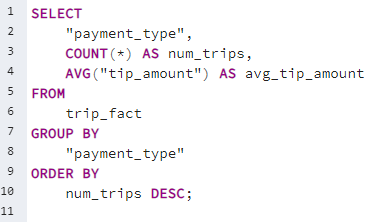
* + - * Findings
        + Cab Demand decreases over the years
        + Peak Month: May & December
        + Busiest hour: 3pm - 8pm Least Busiest hour: 4am - 6am
      * Goal
        + Provide senior management with this information, so that they can utilize it to decide when to decrease/ increase taxi fare to maximize overall company benefit

1. **Ride sharing Feature**
   * + - Explanation of ride sharing
         * Customers with less tight schedule can use the feature to call a cab with less money with a trade-off of a slightly longer wait time
       - Query code with OLAP roll-up
       - ***Full program in Document (OLAP\_query.sql)***



* + - * Purpose of query code
        + The query code is aimed to find the busiest pick up location, since this is a new feature and the company may want to test it in a few areas first. With the use of 1st BI development, we can also provide the customers with their estimated pickup time and drop off time.
      * Goal
        + To maximize the customer base of the company by providing an eco-friendly way to taxi rides.

1. **Payment analysis / Tips amount based on payment type**
   * + - Query code
       - ***Full program in Document (OLAP\_query.sql)***



* + - * ***Full Detailed SQL query is in Document***
      * Findings:
        + Most used payment type is type 1, credit card
        + Payment type 2, Cash is the 2nd popular payment type
        + Credit card has the highest average tip amount
      * Goal
        + Find the majority payment among customers and upgrade the payment system to adjust to that payment type to provide a better taxi ride experience for customers. Also can utilize the tip amount per payment type information to have cab drivers cooperate with the company on the upgrade request