The Maven Taxi Challenge by mavenanalytics.io

https://www.mavenanalytics.io/blog/maven-taxi-challenge?utm\_source=email&utm\_campaign=taxichallengelaunch\_email\_udemy

**Introduction**

The Maven Taxi Challenge is a real-world business case created by **ENRIQUE RUIZ and the team at Maven Analytics** to test analytics skills.

**About the dataset**

* This dataset contains **six tables** in CSV format, along with a **geospatial map** in TopoJSON and Shapefile formats
* The 4 **Taxi Trips** tables contain 28 million Green Taxi trips in New York City from 2017 to 2020. Each record represents one trip, with fields containing details about the pick-up/drop-off times and locations, distances, fares, passengers, and more
* The **454 Calendar** table contains a fiscal calendar (2017-2020) used by the Taxi & Limousine Commission, with fields containing the date and fiscal year, quarter, month, and week
* The **Taxi Zones** table contains information about 265 zone locations in New York City, including the location id, borough, and service zone
* The **Taxi Zones Map** files contain a map of New York City with divisions for the 265 locations that a person can be used to create custom map visuals in Power BI (TopoJSON) or Tableau (Shapefile)

**Data Dictionary**

|  |  |
| --- | --- |
| Field | Description |
| VendorID | A code indicating the LPEP provider that provided the record (1= Creative Mobile Technologies, LLC; 2= Verifone Inc.) |
| lpep\_pickup\_datetime | The date and time when the meter was engaged |
| lpep\_dropoff\_datetime | The date and time when the meter was disengaged |
| store\_and\_fwd\_flag | This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka ìstore and forward,î because the vehicle did not have a connection to the server (Y= store and forward trip; N= not a store and forward trip) |
| RatecodeID | The final rate code in effect at the end of the trip (1= Standard rate; 2= JFK; 3= Newark; 4= Nassau or Westchester; 5= Negotiated fare; 6= Group ride) |
| PULocationID | TLC Taxi Zone in which the taximeter was engaged |
| DOLocationID | TLC Taxi Zone in which the taximeter was disengaged |
| passenger\_count | The number of passengers in the vehicle (this is a driver entered value) |
| trip\_distance | The elapsed trip distance in miles reported by the taximeter |
| fare\_amount | The time-and-distance fare calculated by the meter |
| extra | Miscellaneous extras and surcharges (this only includes the $0.50 and $1 rush hour and overnight charges) |
| mta\_tax | $0.50 MTA tax that is automatically triggered based on the metered rate in use |
| tip\_amount | Tip amount (automatically populated for credit card tips - cash tips are not included) |
| tolls\_amount | Total amount of all tolls paid in trip |
| improvement\_surcharge | $0.30 improvement surcharge assessed on hailed trips at the flag drop |
| total\_amount | The total amount charged to passengers (does not include cash tips) |
| payment\_type | A numeric code signifying how the passenger paid for the trip (1= Credit card; 2= Cash; 3= No charge; 4= Dispute; 5= Unknown; 6= Voided trip) |
| trip\_type | A code indicating whether the trip was a street-hail or a dispatch that is automatically assigned based on the metered rate in use but can be altered by the driver (1= Street-hail; 2= Dispatch) |
| congestion\_surcharge | Congestion surcharge for trips that start, end or pass through the congestion zone in Manhattan, south of 96th street ($2.50 for non-shared trips in Yellow Taxis; $2.75 for non-shared trips in Green Taxis) |

**Message from the maven team:**

*Welcome to the team!*

*We’ve been collecting trip data for ~4 years now, but without a proper analyst, we haven’t been able to put it to good use. That's where you come in!*

*The raw data has some issues, so we'll need to make the following adjustments and assumptions to clean and prep the data:*

* *Let’s stick to trips that were NOT sent via “store and forward.”*
* *I’m only interested in street-hailed trips paid by card or cash, with a standard rate*
* *We can remove any trips with dates before 2017 or after 2020, along with any trips with pick-ups or drop-offs into unknown zones*
* *Let’s assume any trips with no recorded passengers had 1 passenger*
* *If a pick-up date/time is AFTER the drop-off date/time, let’s swap them*
* *We can remove trips lasting longer than a day, and any trips which show both a distance and fare amount of 0*
* *If you notice any records where the fare, taxes, and surcharges are ALL negative, please make them positive*
* *For any trips that have a fare amount but have a trip distance of 0, calculate the distance this way: (Fare amount - 2.5) / 2.5*
* *For any trips that have a trip distance but have a fare amount of 0, calculate the fare amount this way: 2.5 + (trip distance x 2.5)*

*Once the data is cleaned up, I’m hoping you can build me a dashboard to help with weekly planning and logistics.****For any given fiscal week, I'd like to be able to use historical data to answer the following questions:***

* *What's the average number of trips we can expect this week?*
* *What's the average fare per trip we expect to collect?*
* *What's the average distance traveled per trip?*
* *How do we expect trip volume to change, relative to last week?*
* *Which days of the week and times of the day will be busiest?*
* *What will likely be the most popular pick-up and drop-off locations?*

*I realize this is a lot to ask for, but this type of analysis will have a huge impact on our business!*

*Thanks in advance,*

*Mario Maven (Lead Dispatcher, NYC Green Taxis)*

**Tools used for the project**

**Cleaning Tool:**

Postgres SQL server

pgAdmin 4

**Visualization Tool:**

Tableau public

**Uploading the 4 tables to Database:**

From about the dataset, we know that the dataset has “4 **Taxi Trips** tables containing a total of 28 million Green Taxi trips in New York City from 2017 to 2020”.

I created a database called Taxi to store the tables. Next, I created Identical tables for the 4 Taxi Trips with the corresponding schema and data type.

for 2017 and 2018 with the SQL command below:

CREATE TABLE public."TABLE NAME"

(

"VendorID" numeric,

lpep\_pickup\_datetime timestamp,

lpep\_dropoff\_datetime timestamp,

store\_and\_fwd\_flag text,

"RatecodeID" integer,

"PULocationID" integer,

"DOLocationID" integer,

passenger\_count integer,

trip\_distance double precision,

fare\_amount double precision,

extra double precision,

mta\_tax double precision,

tip\_amount double precision,

tolls\_amount numeric,

improvement\_surcharge numeric,

total\_amount numeric,

payment\_type integer,

trip\_type integer

);

Where TABLE NAME is 2017\_taxi\_trips for creating table for 2017 data and 2018\_taxi\_trips for creating a table for 2018 data

Since 2019 and 2020 tables have an additional column, slight modification needs to be made in the SQL code to add the extra column:

CREATE TABLE public."TABLE NAME"

(

"VendorID" numeric,

lpep\_pickup\_datetime timestamp,

lpep\_dropoff\_datetime timestamp,

store\_and\_fwd\_flag text,

"RatecodeID" integer,

"PULocationID" integer,

"DOLocationID" integer,

passenger\_count integer,

trip\_distance double precision,

fare\_amount double precision,

extra double precision,

mta\_tax double precision,

tip\_amount double precision,

tolls\_amount numeric,

improvement\_surcharge numeric,

total\_amount numeric,

payment\_type integer,

trip\_type integer,

congestion\_surcharge double precision

);

Where TABLE NAME is 2019\_taxi\_trips for the 2019 table and 2020\_taxi\_trips for the 2020 table.

Uploading the CSV file into the database to populate the corresponding tables

COPY ‘destination table name’

FROM 'the csv file path '

DELIMITER ','

CSV HEADER;

**Combining the tables**

After uploading the tables, cleaning them individually will be time consuming. To resolve this, the 4 tables need to be combined into one table called ‘total\_taxi\_trips’

The SQL UNION operator comes in handy for this, but for the union to work, the 4 tables need to meet the following requirement:

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

The tables failed the first requirement. To resolve this, 2017 and 2018 table need to be altered to add an additional column called ‘congestion\_surcharge’ with data type ‘double precision’

The SQL command to alter table is:

ALTER TABLE IF EXISTS public.TABLE NAME

ADD COLUMN congestion\_surcharge double precision;

Where TABLE NAME is 2017\_taxi\_trips for 2017 table and 2018\_taxi\_trips for 2018 table

After that, the union code should work just fine.

SQL command to combine all 4 taxi table data set into one:

CREATE TABLE total\_taxi\_trips AS (

SELECT \* FROM 2017\_taxi\_trips

UNION

SELECT \* FROM 2018\_taxi\_trips

UNION

SELECT \* FROM 2019\_taxi\_trips

UNION

SELECT \* FROM 2020\_taxi\_trips

UNION);

After successfully running the command, a total of 5 tables are in the database



The following SQL command was used to verify our dataset uploaded successfully and is indeed 28 million records

SELECT COUNT (\*) AS total\_number\_of\_records

FROM public.total\_taxi\_trips

The Result:



From the result, a total of 28 million records was successfully uploaded into the database

**--Cleaning The Dataset**

**-- creating a temp table from the total\_taxi\_trips table for cleaning**

**CREATE TABLE temp\_total\_taxi\_trips AS**

**(SELECT \***

**FROM public.total\_taxi\_trips)**

**-- removing trips that were sent via ‘store and forward’**

**DELETE FROM public.temp\_total\_taxi\_trips**

**WHERE store\_and\_fwd\_flag <> 'N'**

**-- removing trips that were not street hailed, paid by card or cash with a standard rate**

**CREATE TABLE public.temp1\_total\_taxi\_trips AS -- creates a new table from the temp 1 to store filtered table**

**(SELECT \* FROM public.temp\_total\_taxi\_trips**

**WHERE trip\_type = 1 -- selects only street-hailed trips**

**AND (payment\_type = 1 OR payment\_type = 2) -- only selects trips paid by card or cash**

**AND "RatecodeID" = 1) -- only selects trips with a standard rate**

**-- dropping temp table**

**DROP TABLE public.temp\_total\_taxi\_trips**

**-- deleting dates before 2017 or after 2020**

**DELETE FROM public.temp1\_total\_taxi\_trips**

**WHERE EXTRACT(YEAR FROM lpep\_pickup\_datetime) < 2017**

**OR EXTRACT(YEAR FROM lpep\_pickup\_datetime) > 2020**

**DELETE FROM public.temp1\_total\_taxi\_trips**

**WHERE EXTRACT(YEAR FROM lpep\_dropoff\_datetime) < 2017**

**OR EXTRACT(YEAR FROM lpep\_dropoff\_datetime) > 2020**

**-- removes trips with drop of or pickup in an unknown zone;**

**DELETE FROM public.temp1\_total\_taxi\_trips**

**WHERE "PULocationID" > 263**

**OR "DOLocationID" > 263**

**-- Changing passenger count from 0 to 1**

**UPDATE public.temp1\_total\_taxi\_trips**

**SET passenger\_count = 1**

**WHERE passenger\_count = 0;**

-- Verifying if change was implemented successfully

SELECT passenger\_count

FROM public.temp1\_total\_taxi\_trips

ORDER BY passenger\_count ASC

LIMIT 100;

-- Swapping date/time where pickup time is after drop off time

UPDATE public.temp1\_total\_taxi\_trips

SET lpep\_pickup\_datetime = lpep\_dropoff\_datetime,

lpep\_dropoff\_datetime = lpep\_pickup\_datetime

WHERE lpep\_pickup\_datetime > lpep\_dropoff\_datetime;

-- Verifying if change was implemented successfully it should read 0

SELECT count(\*)

FROM public.temp1\_total\_taxi\_trips

WHERE lpep\_pickup\_datetime > lpep\_dropoff\_datetime ;

-- removing trips lasting more than a day

DELETE FROM public.temp1\_total\_taxi\_trips

WHERE (lpep\_dropoff\_datetime - lpep\_pickup\_datetime) > '24:00:00';

--removing trips with both distance and fare amount of 0

DELETE FROM public.temp1\_total\_taxi\_trips

WHERE trip\_distance = 0 AND fare\_amount = 0;

-- converting to positive records where fare, taxes, and surcharges are all negative

UPDATE public.temp1\_total\_taxi\_trips

SET fare\_amount = abs(fare\_amount),

mta\_tax = abs(mta\_tax),

improvement\_surcharge = abs(improvement\_surcharge)

WHERE

fare\_amount < 0

AND mta\_tax < 0

AND improvement\_surcharge < 0;

-- Calculating distance for trips with fare amount but have a trip distance of 0

UPDATE public.temp1\_total\_taxi\_trips

SET trip\_distance = (fare\_amount - 2.5) / 2.5

WHERE fare\_amount > 0 AND trip\_distance = 0;

-- Verifying if change was implemented successfully

SELECT count(\*)

FROM public.temp1\_total\_taxi\_trips

WHERE fare\_amount > 0 AND trip\_distance = 0;