Mid-Program Project 1

Chicago Taxi Data Research

edureka!

Contents

1. Background

Marketix, a Market Research firm, has hired you as a big data engineer. You are asked to prepare the data for performing market research on the trends and patterns of Taxi services in Chicago. The report should equip to serve the following target audience:

- Executives and Leadership team from any of the existing taxi service provider or aggregator that operates already in the region.
- Executives and Leadership team from any of the taxi service providers operating in other regions, who are trying to enter the market for the first time.
- Any potential Entrepreneur or investor who is trying to invest in the business of taxi service or aggregator service.

The outcomes and findings must cater to the needs of diversified target audience, who have different goals in their mind.

While there are different ways to perform market research, the ideal one would be to follow a data driven approach. Typically, there are two types of data that can be used for this purpose, namely Primary and Secondary data.

The efforts involved in accumulating primary data is more, and therefore start your market research with secondary data. However, there are cases where secondary data might not be available. Fortunately, in our case, there is a huge dataset (~32 GB) available from Chicago Data Portal that carries information relevant to our market research problem.

2. Dataset Introduction

This dataset includes taxi trips from 2013 to 2019, reported to the City of Chicago in its role as a regulatory agency. To protect privacy but allow for aggregate analyses, the Taxi ID is consistent for any given taxi medallion number but does not show the number, Census Tracts are suppressed in some cases, and times are rounded to the nearest 15 minutes. Due to the data reporting process, not all trips are reported but the City believes that most are.

Columns present in the dataset are:

Column Name	Description	Data Type
Trip ID	A unique identifier for the trip.	Text
Taxi ID	A unique identifier for the taxi.	Text
Trip Start Timestamp	When the trip started, rounded to the nearest 15 minutes.	Floating Timestamp
Trip End Timestamp	When the trip ended, rounded to the nearest 15 minutes.	Floating Timestamp
Trip Seconds	Time of the trip in seconds.	Number
Trip Miles	Distance of the trip in miles.	Number
Pickup Census Tract	The Census Tract where the trip began. For privacy, this Census Tract is not shown for some trips. This column often will be blank for locations outside Chicago.	Text
Dropoff Census Tract	The Census Tract where the trip ended. For privacy, this Census Tract is not shown for some trips. This column often will be blank for locations outside Chicago.	Text
Pickup Community Area	The Community Area where the trip began. This column will be blank for locations outside Chicago.	Number
Dropoff Community Area	The Community Area where the trip ended. This column will be blank for locations outside Chicago.	Number
Fare	The fare for the trip.	Number
Tips	The tip for the trip. Cash tips generally will not be recorded.	Number
Tolls	The tolls for the trip.	Number
Extras	Extra charges for the trip.	Number
Trip Total	Total cost of the trip, the total of the previous columns.	Number
Payment Type	Type of payment for the trip.	Text
Company	The taxi companies.	Text
Pickup Centroid Latitude	The latitude of the centre of the pickup census tract or the community area if the census tract has been	Number

	hidden for privacy. This column often will be blank for locations outside Chicago.	
Pickup Centroid Longitude	The longitude of the centre of the pickup census tract or the community area if the census tract has been hidden for privacy. This column often will be blank for locations outside Chicago.	Number
Pickup Centroid Location	The location of the centre of the pickup census tract or the community area if the census tract has been hidden for privacy. This column often will be blank for locations outside Chicago.	Point
Dropoff Centroid Latitude	The latitude of the centre of the drop-off census tract or the community area if the census tract has been hidden for privacy. This column often will be blank for locations outside Chicago.	Number
Dropoff Centroid Longitude	The longitude of the centre of the drop-off census tract or the community area if the census tract has been hidden for privacy. This column often will be blank for locations outside Chicago.	Number
Dropoff Centroid Location	The location of the centre of the drop-off census tract or the community area if the census tract has been hidden for privacy. This column often will be blank for locations outside Chicago.	Point

Following are some of the key highlights about the dataset:

- It has data about around 71M trips for a duration of 7 years, starting from Jan 2013 to June 2019.
- These trips belong to ~8000 unique taxis identified by a unique identifier.
- There is a trip duration indicating trips ranging from few minutes all the way up to one full day.
- Trip distance ranges from few miles all the way up to ~3500 miles.
- The entire city is divided into 77 different communities, and for each pickup & drop, the community is mentioned in the dataset.
- The information about trip fare is available for each trip.
- Around 30% of the records has the information about the taxi company to which the trip belongs. This data will help you find the market share of these companies.
- The precise pickup location in the form of latitude and longitude is available.

The dataset provides a lot of opportunity to come up with highly relevant insights for the research.

3. Problem Definition

The crucial step is to make the dataset ready to perform analyses, so as part of the Data Engineering team, you are responsible to perform,

- Data Cleaning, Loading & Data Pre-processing
- Data Preparation

in order to apply the following techniques:

- Forecasting: Possibly to find out the growth of market size at various granularity levels.
- Clustering: Applying at various dimensions like pickup, drop off communities, different time periods to find out interesting patterns.
- Classification: Labelling different time periods as peak hour, low activity, etc. and applying classification mechanisms on top of the same.
- Graph analysis: Creating origin-destination pairs based on the 77 communities and apply graph analysis such as connected components.

While above mentioned techniques can be applied on top of the data, each of them requires the data to be prepared in a specific format.

3.1 Cleaning, Loading and Pre-processing Data

NOTE: We have already pre-processed & loaded the data into Hive tables to provide you the final version of the dataset, which you will directly use for data preparation.

The Dataset that is available publicly is not in a proper format to be directly used for data preparation. So, we cleaned and transformed it in the pre-processing stage.

Below mentioned pre-processing steps are just for your reference to see what we did to pre-process the data. You don't have to perform any of the steps mentioned below. However, even after pre-processing the dataset using the steps mentioned below, some issues exist in the data which needs to be removed only while data preparation.

Kindly go through the below commands to understand the pre-processing steps. You must pick data from these tables to perform further steps in this project.

1. Creating taxi_details_str table & loading the whole row as a string

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_details_str (
taxi_trip_details_str String)
STORED AS TEXTFILE
tblproperties("skip.header.line.count"="1");
```

```
LOAD DATA INPATH '/bigdatapgp/common_folder/midproject/taxi_trip_dataset/taxi_trip.csv'
OVERWRITE INTO TABLE chicago_taxis.taxi_details_str;
```

2. Splitting one column in multiple columns and creating taxi_trip_details table,

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_trip_details
AS
select split(taxi_trip_details_str, ",")[0] as trip_id,
split(taxi_trip_details_str, ",")[1] as taxi_id,
split(taxi_trip_details_str, ",")[2] as trip_start_time,
split(taxi_trip_details_str, ",")[3] as trip_end_time,
cast(split(taxi_trip_details_str, ",")[4] as int) as trip_seconds,
cast(split(taxi_trip_details_str, ",")[5] as float) as trip_miles,
cast(split(taxi_trip_details_str, ",")[6] as bigint) as pickup_tract,
cast(split(taxi_trip_details_str, ",")[7] as bigint) as dropoff_tract,
cast(split(taxi_trip_details_str, ",")[8] as tinyint) as pickup_community,
cast(split(taxi_trip_details_str, ",")[9] as tinyint) as dropoff_community,
cast(split(taxi_trip_details_str, ",")[10] as float) as trip_fare,
cast(split(taxi_trip_details_str, ",")[11] as float) as tip_amt,
cast(split(taxi_trip_details_str, ",")[12] as float) as toll_amt,
cast(split(taxi_trip_details_str, ",")[13] as float) as extra_amt,
cast(split(taxi_trip_details_str, ",")[14] as float) as trip_total_amt,
split(taxi_trip_details_str, ",")[15] as payment_type,
split(taxi_trip_details_str, ",")[16] as company,
cast(split(taxi_trip_details_str, ",")[17] as double) as pickup_latitude,
cast(split(taxi_trip_details_str, ",")[18] as double) as pickup_longitude,
split(taxi_trip_details_str, ",")[19] as pickup_location,
cast(split(taxi_trip_details_str, ",")[20] as double) as dropoff_latitude,
cast(split(taxi_trip_details_str, ",")[21] as double) as dropoff_longitude,
split(taxi_trip_details_str, ",")[22] as dropoff_location,
```

```
split(taxi_trip_details_str, ",")[23] as community_areas
from
chicago_taxis.taxi_details_str;
```

```
select split(taxi_trip_details_str, ",")[0] as trip_id,
split(taxi_trip_details_str, ",")[1] as taxi_id,
split(taxi_trip_details_str, ",")[2] as trip_start_time
from
chicago_taxis.taxi_details_str limit 5;
```

3. Numerical Mapping of taxi_id & trip_id to reduce the data volume

```
select count(distinct taxi_id) from taxi_trip_details;
```

There are only 8287 distinct values. But as we can see the uuid is a really large:

c1305c4490085b703eed20e95ab0c479c954ae3735a963578627d563fadbf2cc859e12ebcd12c4b8f34b7eb2d6c4782b17b56ba8ddfe896fff5763105e81e050

Keeping this column would result in significant performance implications.

Hence, we can either

- Drop this field
- Or move it to a separate table and create a numerical mapping of the same into the master table. This would reduce the data volume significantly as we are going to store only around 8K distinct values of taxi ids, which are repeating for over 90M trips.

Second option makes sense, as it would help us perform per taxi-wise analysis.

Similarly, we can drop the trip_id, which is a uuid and create a row_number which can be an integer in its place.

Following set of queries achieve the same.

3.a. Creating a separate table with distinct taxi_id values

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_id_mapping
AS
select distinct taxi_id from taxi_trip_details
```

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_id_mapping_with_id
AS
select row_number() over() as id, taxi_id from taxi_id_mapping
```

3.b. Joining taxi_id_mapping_with_id table to master table to replace the current taxi_id i.e uuid with a numerical id.

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_trip_details_taxi_id_removed
AS
SELECT
trip_id,
id as taxi_id_int,
trip_start_time,
trip_end_time,
trip_seconds,
trip_miles,
pickup_tract,
dropoff_tract,
pickup_community,
dropoff_community,
trip_fare,
tip_amt,
```

```
toll_amt,
extra_amt,
trip_total_amt,
payment_type,
company,
pickup_latitude,
pickup_longitude,
pickup_location,
dropoff_latitude,
dropoff_longitude,
dropoff_location,
community_areas
from
taxi_trip_details as a
join
taxi_id_mapping_with_id as b
on
a.taxi_id = b.taxi_id
```

3.c. Removing trip_id (uuid) and adding an int id instead

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_trip_details_taxi_trip_id_removed
STORED AS ORC
AS
SELECT
row_number() over() as trip_id_int,
taxi_id_int,
trip_start_time,
trip_end_time,
trip_seconds,
trip_miles,
```

pickup_tract,

dropoff_tract,

pickup_community,

dropoff_community,

trip_fare,

tip_amt,

toll_amt,

extra_amt,

trip_total_amt,

payment_type,

company,

pickup_latitude,

pickup_longitude,

pickup_location,

dropoff_latitude,

dropoff_longitude,

dropoff_location,

community_areas

from

chicago_taxis.taxi_trip_details_taxi_id_removed

4. Cleaning up the temp tables

drop table chicago_taxis.taxi_details_str

drop table chicago_taxis.taxi_trip_details

drop table chicago_taxis.taxi_trip_details_taxi_id_removed

5. Casting date fields

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_trip_details_taxi_trip_id_removed_ts

STORED AS ORC

AS

SELECT

taxi_id_int,

trip_start_time,

trip_end_time,

to_date(from_unixtime(unix_timestamp(split(trip_start_time, " ")[0], 'MM/dd/yyyy'), 'yyyy-MM-dd')) as
trip_start_date,

to_date(from_unixtime(unix_timestamp(split(trip_end_time, " ")[0], 'MM/dd/yyyy'), 'yyyy-MM-dd')) as
trip_end_date,

trip_seconds,

trip_miles,

pickup_tract,

dropoff_tract,

pickup_community,

dropoff_community,

trip_fare,

tip_amt,

toll_amt,

extra_amt,

trip_total_amt,

payment_type,

company,

pickup_latitude,

pickup_longitude,

pickup_location,

dropoff_latitude,

dropoff_longitude,

dropoff_location,

community_areas

from
```

chicago_taxis.taxi_trip_details_taxi_trip_id_removed

6. Adding two fields for the trip start & end day of the week

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_trip_details_processed_with_dayofweek
AS
SELECT
*,
from_unixtime(unix_timestamp(split(trip_start_time, " ")[0], 'MM/dd/yyyy'), 'u') as start_dayofweek,
from_unixtime(unix_timestamp(split(trip_end_time, " ")[0], 'MM/dd/yyyy'), 'u') as end_dayofweek
from
chicago_taxis.taxi_trip_details_taxi_trip_id_removed
```

7. Adding a weekend field to store whether a day is weekday or weekend

```
CREATE TABLE IF NOT EXISTS chicago_taxis.taxi_trip_details_weekend_encoded
STORED AS ORC
AS
SELECT
*,
CASE
WHEN start_dayofweek in (6,7) THEN 1
WHEN start_dayofweek in (1,2,3,4,5) THEN 0
END AS weekend
from
chicago_taxis.taxi_trip_details_processed_with_dayofweek
```

3.2 Data Preparation

3.2.1 Data Summary

We have already pre-processed the data and made it available in ***chicago_taxis*** database in ***Hive***.

First step in data processing is to create a detailed summary of the data on various dimensions.

In order to create data summary, write hive queries with detailed explanation for the following questions:

1. What are the total number of trips per year? Present the findings in the below format.

Year	Total Number of Trips
2013	
2014	
2015	
2016	
2017	
2018	
2019	

2. Create the same summary for number of trips at monthly level. Present the findings in the below format.

Year/Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Grand Total
2013													
2014													
2015													
2016													
2017													
2018													
2019													
Grand Total													

3. Calculate the percentage of records that contains drop-off community value. Excluding all the NULL records, find out the top 10 communities, where people travel to, based on the drop-off community field and also find its percentage to the total number of trips. Present the findings in the below format.

drop_off community	Community_trips	Percentage

4. Create a table which contains the total number of trips for each drop-off community across each year. Using the above table, find the top 10 records based on number of trips with year and drop_off community. Remove the null record while creating the table to remove inconsistencies.

drop_off community	Year	Community_trips

5. Create a table which contains total number of trips for each drop-off communities across weekdays & weekends to check if there is any sort of pattern visible. After creating the table, find the top 10 drop off communities based on number of trips where people travel on weekdays. Find the same for the weekends. Also find the total number of trips taken on weekdays & weekends and their ratio.

Weekend Trips	
Community	**Total trips**

Weekday Trips	
Community	**Total trips**

Weekend vs Weekdays		
Weekend	**Total_trips**	**Ratio**

6. Find the distribution of total number of trips based on trip duration, like <1 hr, 1 to 2 hr, 2 to 3, … 22 to 23 hr. Note that this requires converting trip_seconds into trip_hours as pre-processing. Remove the trips that do not contain trip duration.

trip_hour	trip_count	ratio
0-1		
1-2		
2-3		
3-4		
4-5		
5-6		
6-7		
7-8 …		

7. Find the top 10 buckets of the number of trips distribution based on the distance covered. Also round off the trip miles to the nearest integer. Remove the trips that do not contain distance.

Roundoff Distance	Number of Trips

8. Find top 10 buckets of the number of trips distribution based on the trip fare. Also round off the trip fare to the nearest integer. Remove the trips that do not contain trip fare.

Roundoff Fare	Number of Trips

9. Compute the average trip fare per day. Also compute the average trip fare per trip. Compute the same based on weekdays and weekend days. Find out if there is any substantial difference observed.

10. Create a table to store the taxi wise total fare & total number of trips for each day. Find the following insights from the table:
    a. Find the top 10 taxis based on average trips per day.
    b. Find the top 10 taxis based on average fare per day.

taxi_id_int	avg_trip_count

taxi_id_int	avg_daily_total

Your task is to perform the above actions by cleaning, querying and processing the data present in the tables specified in the loading & pre-processing phase.

3.2.2 Data Preparation for Forecasting

Forecasting refers to predicting what will happen in the future by taking into consideration the events in the past and present. Basically, it is a technique that helps businesses cope with the impact of the future's uncertainty by using historical data and trends.

This technique can be applied at a daily, weekly and monthly levels depending on the amount of data available.

Your task is to create all the tables. Following is the format in which the data needs to be prepared for applying forecasting.

1. **Daily Summary table**

Date	Day of week	Month	Year	Week end/ Week day	Total Trip count	Total Trip fare	Total trip miles	Total trip duration (min)	Avg Trip fare	Avg Trip miles	Avg trip duration (min)
2013-02-09	6	2	2013	0	80	5000	8500	125500	83	…	…
2013-03-29	5	3	2013	1	32	1500	5000	93500	50	…	…
2013-12-05	4	12	2013	1	50	2300	7000	100200	70	…	…

…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…

2. Weekly Summary table

Week_No	Date_From	Date_To	Month	Total Trip count	Total Trip fare	Total trip miles	Total trip duration (min)	Avg Trip fare	Avg Trip miles	Avg trip duration(min)
1	1/6/13	1/12/13	Jan	80	5000	8500	125500	83	…	…
2	1/13/13	1/19/13	Jan	32	1500	5000	93500	50	…	…
3	1/20/13	1/26/13	Jan	50	2300	7000	100200	70	…	…
…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…

3. Monthly Summary table

Month_no	Date_From	Date_To	Year	Total Trip count	Total Trip fare	Total trip miles	Total trip duration (min)	Avg Trip fare	Avg Trip miles	Avg trip duration(min)
1	1/1/13	1/31/13	2013	80	5000	8500	125500	83	…	…
2	2/1/13	2/28/13	2013	32	1500	5000	93500	50	…	…
3	3/1/13	3/31/13	2013	50	2300	7000	100200	70	…	…
…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…

Note:

Create the daily data by processing the provided tables and use them to create the other two tables so that you don't have to refer the provided tables again. It will help you optimize the processing time. Store the data in ORC format.

You have to use weighted average approach to calculate averages in the above tables. Careful consideration and processing are required in order to avoid computation errors in this process.

3.2.3 Data Preparation for Community Summary

There are couple of attributes called pickup_community and dropoff_community available in the dataset. They indicate the community area of the pickup and drop off location of the trip respectively.

In order to find insights related to community, you need to prepare the data in the below format for pickup & dropoff communities.

Pickup Community ID	Daily Trip Count	Daily Total Fare	Daily Total Distance	Daily Total Duration	Daily Average Amount	Daily Average Distance	Daily Average Duration

Dropoff Community ID	Daily Trip Count	Daily Total Fare	Daily Total Distance	Daily Total Duration	Daily Average Amount	Daily Average Distance	Daily Average Duration

Your task is to create the summary tables by querying and processing the raw data.

You can either use Hive or Spark for processing the raw data. But the final result should be available in the form of Hive tables.

3.2.4 Data Preparation for Origin to Destination Pair Summary

Another interesting analysis is something that can be performed at an Origin-Destination pair level. Ideally there are 2926 bidirectional and 5852 unidirectional connections/edges possible from the 77 communities. But not all the connections may have data. For example, between two given communities, there may not be any trips in the past.

The summary table has to be in the following format.

Community_orig	Community_dest	Trip_count	Total_trip_miles	Avg_trip_miles	Avg_trip_duration	Avg_trip_fare
1	…	…	…	…	…	…
2	…	…	…	…	…	…
3	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
…	…	…	…	…	…	…
77	…	…	…	…	…	…

Your task is to create the summary tables by processing the raw data.

You need to use Spark for processing the raw data. The final result should be available in the form of Hive tables, so that the rest of the team can query and process the data available in these tables.

More than any of the previous items, it makes real sense to use Spark to create this summary table. As coming out with these aggregates using Hive requires a self-join on the entire raw table. It will be a costly and time-consuming operation. (Implement this problem using Spark)

Note:

While performing this processing, find out the fill rate on the fields pickup community and drop-off community.

If large proportion of the trips have no values filled part of these fields, there will probably be lot of Origin Destination pairs not having data. Additionally, check if these fields can be filled in by using the pickup geo location or the pickup/drop-off census track fields. (**Consider this as an optional exercise)**

The data need to be present in following format.

Pickup Community	Dropoff Community	Trip Count	Total Amount	Total Miles	Total Mins	Avg Amount	Avg Miles	Avg Mins

3.2.5 Data Preparation for Company Summary

Only 30% of the trips have the taxi company detail filled in. Extract the available company name/detail and create a separate **company detail** table.

Create a summary table at a company level with the following details:

Company	Daily Trip Count	Daily Total Fare	Daily Total Distance	Daily Total Duration	Daily Average Amount	Daily Average Distance	Daily Average Duration

Note:

For each company there has to be a record created for every year. This will help to identify the company level yearly summary.

3.2.6 Data Summary in RDBMS

Some of the data summary generated at company and community level can be stored in an RDBMS (like MySQL) for quick querying and inspection. It is also ideal to connect any reporting or BI tools to MySQL instead of trying to connect to Hive tables.

Following three are the potential candidates for storing in MySQL and serve.

Pickup Community ID	Daily Trip Count	Daily Total Fare	Daily Total Distance	Daily Total Duration	Daily Average Amount	Daily Average Distance	Daily Average Duration

Dropoff Community ID	Daily Trip Count	Daily Total Fare	Daily Total Distance	Daily Total Duration	Daily Average Amount	Daily Average Distance	Daily Average Duration

Company	Daily Trip Count	Daily Total Fare	Daily Total Distance	Daily Total Duration	Daily Average Amount	Daily Average Distance	Daily Average Duration

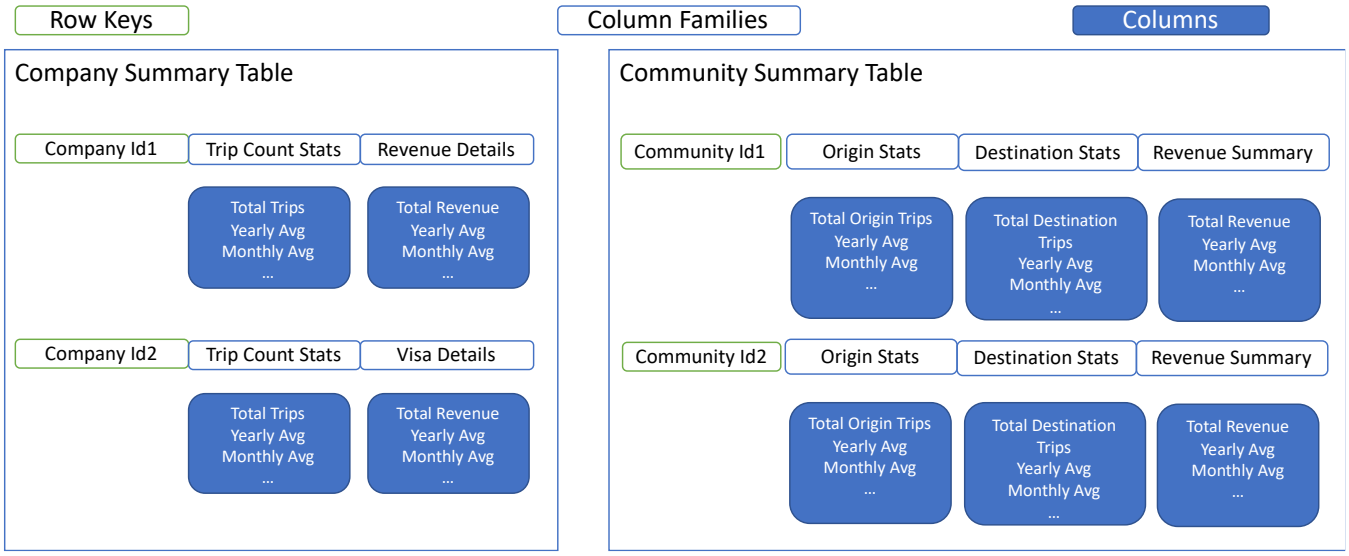
Data volume and structure of these tables make sense to store into MySQL.

3.2.7 Summary Data Mart

While the resulting summary can be stored into Hive in the form of various tables, it makes sense to store a detailed summary data mart in HBase or Cassandra. So that it is easier to,

- Search for entities (companies, communities, etc.) that match specific search criteria.
- To perform aggregations on selected set of dimensions, like revenue, trip count, etc.

Below image shows a possible data model to store in a columnar data store like HBase or Cassandra.



Create an HBase table in the above format. Write an HBase Java API to insert the data.

Data stored into these data marts can be queried for some complicated search use cases which will be hard to fit into MySQL. Any potential APIs to serve the required results with these search use cases can be developed on top of this data mart.

NOTE: For each step, you need to mention the detailed findings and present it in a document.