TITLE: UBER DATAANALYTICS

Introduction

- In the rapidly evolving landscape of the tech industry, companies like Uber have redefined traditional business models by leveraging advanced technologies. One pivotal aspect that has garnered increasing attention is customer satisfaction, as businesses recognize its fundamental role in ensuring long-term success. This recognition has led to a surge in interest and investment in data-driven approaches to understand and enhance customer experiences.

- The "Uber Data Analytics" project is a response to this paradigm shift, aiming to delve into the intricacies of customer satisfaction among Uber users. By harnessing the power of big data techniques, this project seeks to extract meaningful insights from the vast amounts of transactional data generated within the Uber platform. The integration of data science methodologies, machine learning techniques, and open-source tools is poised to offer a comprehensive understanding of customer behaviors, preferences, and pain points.

Objective

- The primary objective of the "Uber Data Analytics" project is to analyze customer satisfaction by employing a big data approach.
- By collecting and processing large volumes of data generated from Uber transactions, the project seeks to identify patterns, trends, and insights that can contribute to improving the overall user experience. The goal is to create a data-driven understanding of customer satisfaction factors and provide valuable insights for enhancing Uber's service.
- **Problem Statement**: Despite Uber's efforts to use data science methods, there may still be gaps in understanding customer satisfaction comprehensively. This project addresses the need for a more in-depth analysis of customer experiences by employing big data techniques. Challenges may include handling large datasets, ensuring data quality, and deriving meaningful insights from the complex, dynamic nature of Uber's transactional data.

Dataset

- Snippet of our dataset

VendorID	tpep_pickup_dateti	tpep_dropoff_date	passenger_c	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_ar	dropoff_longitude	dropoff_latitude	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount
1	01-03-2016 00:00	01-03-2016 00:07	1	2.5	-73.97674561	40.76515198	1	N	-74.00426483	40.74612808	1	9	0.5	0.5	2.05	0
1	01-03-2016 00:00	01-03-2016 00:11	1	2.9	-73.98348236	40.76792526	1	N	-74.0059433	40.73316574	1	11	0.5	0.5	3.05	0
2	01-03-2016 00:00	01-03-2016 00:31	2	19.98	-73.78202057	40.64480972	1	N	-73.97454071	40.67576981	1	54.5	0.5	0.5	8	0
2	01-03-2016 00:00	01-03-2016 00:00	3	10.78	-73.86341858	40.76981354	1	N	-73.96965027	40.75776672	1	31.5	0	0.5	3.78	5.54
2	01-03-2016 00:00	01-03-2016 00:00	5	30.43	-73.97174072	40.79218292	3	N	-74.1771698	40.6950531	1	98	0	0	0	15.5
2	01-03-2016 00:00	01-03-2016 00:00	5	5.92	-74.01719666	40.7053833	1	N	-73.97807312	40.7557869	1	23.5	1	0.5	5.06	0
2	01-03-2016 00:00	01-03-2016 00:00	6	5.72	-73.99458313	40.72784805	1	N	0	0	2	23	0.5	0.5	0	0
1	01-03-2016 00:00	01-03-2016 00:16	1	6.2	-73.78877258	40.64775848	1	N	-73.82920837	40.71234512	3	20.5	0.5	0.5	0	0
1	01-03-2016 00:00	01-03-2016 00:05	1	0.7	-73.95822144	40.76464081	1	N	-73.96789551	40.76290131	1	5.5	0.5	0.5	2	0
2	01-03-2016 00:00	01-03-2016 00:24	3	7.18	-73.98577881	40.74119186	1	N	-73.9463501	40.79787827	1	23.5	0.5	0.5	3.2	0
2	01-03-2016 00:00	01-03-2016 00:02	2	0.54	-73.98842621	40.76416016	1	N	-73.99239349	40.75822449	2	4	0.5	0.5	0	0
1	01-03-2016 00:00	01-03-2016 00:07	1	1.7	-73.96981812	40.79742813	1	N	-73.94377136	40.7961998	2	8	0.5	0.5	0	0
1	01-03-2016 00:00	01-03-2016 00:03	1	1.1	-73.95380402	40.7881279	1	N	-73.97154999	40.79523849	1	5.5	0.5	0.5	2.2	0
2	01-03-2016 00:00	01-03-2016 00:09	1	2.1	-73.97608948	40.75217056	1	N	-73.98744965	40.77078247	1	9	0.5	0.5	2.06	0
2	01-03-2016 00:00	01-03-2016 00:24	1	8.54	-74.00206757	40.71912003	1	N	-73.95211792	40.81124115	1	27	0.5	0.5	5.66	0

Each row in the dataset represents a specific Uber trip, and the columns provide detailed information about different aspects of each trip, such as time, location, distance, and fare details.

uber data analytics

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM
Disk

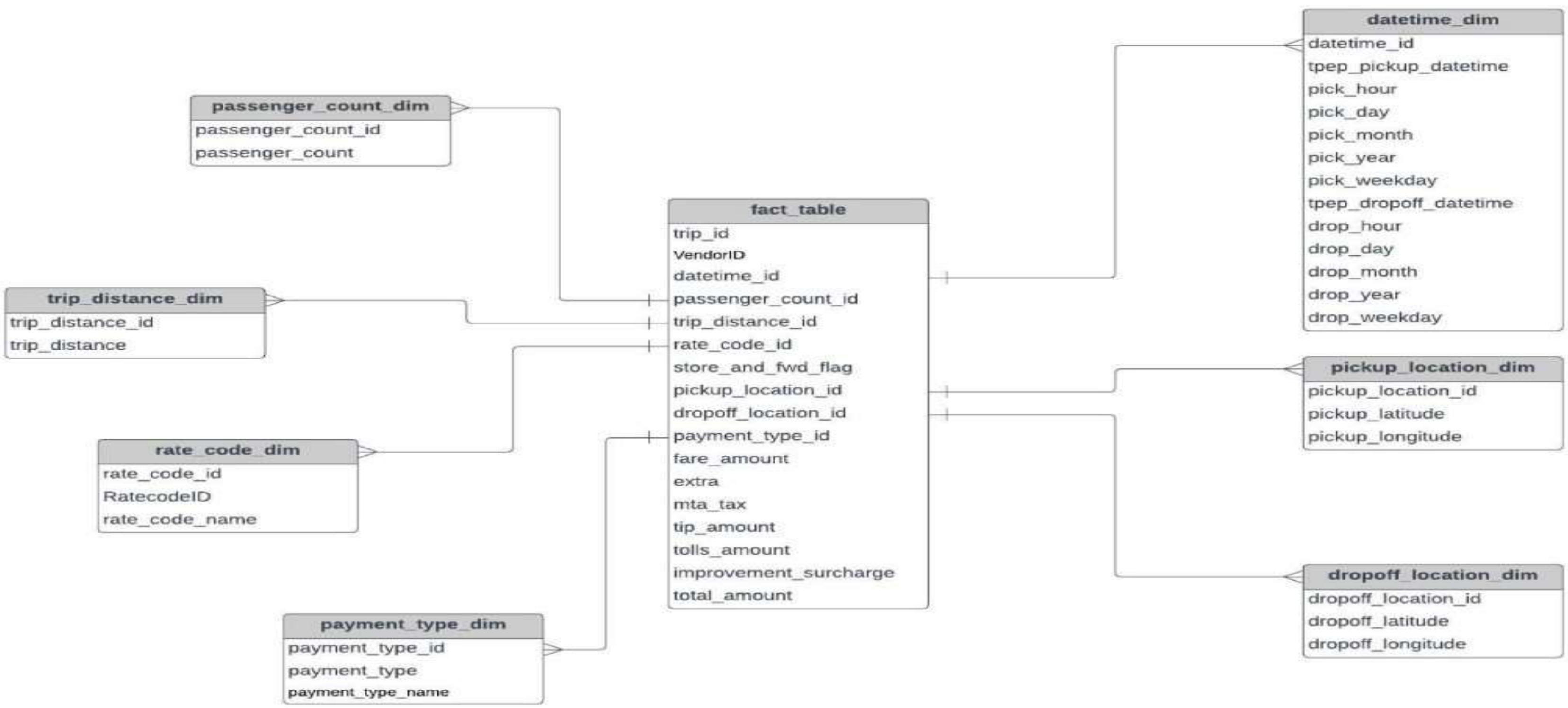
```
import pandas as pd

df = pd.read_csv("uber_data.csv")

df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	dropoff_longitude	dr
0	1	2016-03-01 00:00:00	2016-03-01 00:07:55	1	2.50	-73.976746	40.765152	1	N	-74.004265	
1	1	2016-03-01 00:00:00	2016-03-01 00:11:06	1	2.90	-73.983482	40.767925	1	N	-74.005943	
2	2	2016-03-01 00:00:00	2016-03-01 00:31:06	2	19.98	-73.782021	40.644810	1	N	-73.974541	
3	2	2016-03-01 00:00:00	2016-03-01 00:00:00	3	10.78	-73.863419	40.769814	1	N	-73.969650	
4	2	2016-03-01 00:00:00	2016-03-01 00:00:00	5	30.43	-73.971741	40.792183	3	N	-74.177170	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19858 entries, 0 to 19857
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               19858 non-null  int64
 1   tpep_pickup_datetime   19858 non-null  object
 2   tpep_dropoff_datetime  19858 non-null  object
 3   passenger_count        19858 non-null  int64
 4   trip_distance          19858 non-null  float64
 5   pickup_longitude       19858 non-null  float64
```

Data Model

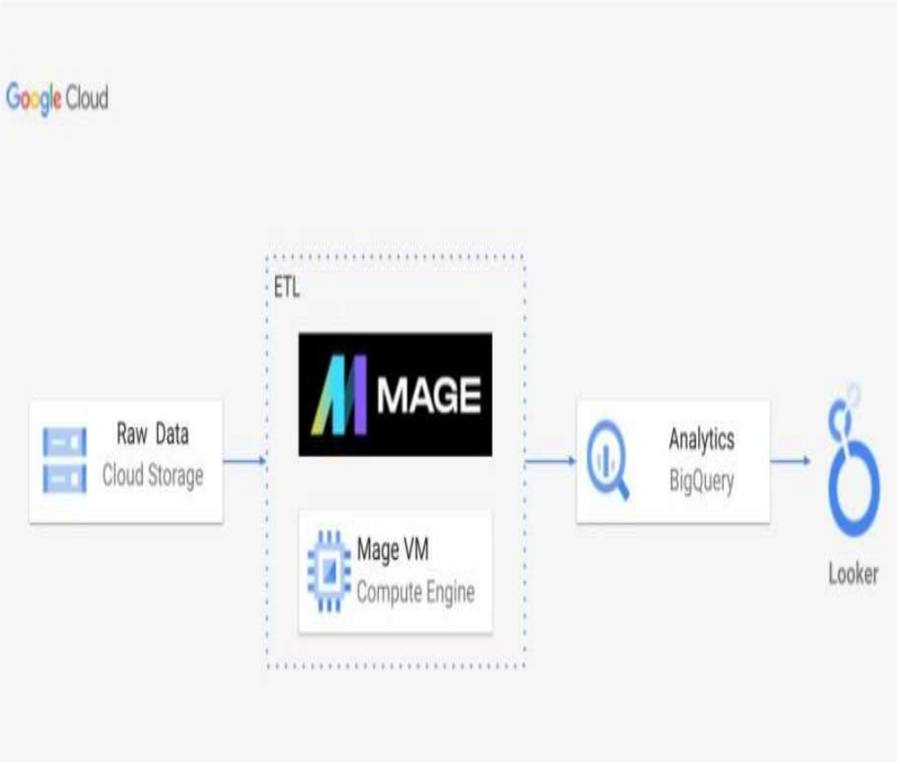


This is a class diagram of a passenger ticket system. It shows the relationships between the different classes in the Uber database.

- The class diagram is divided into three parts: dimensions, fact table, and payment type dimension.
- Dimensions are descriptive attributes of a fact table – datetime dim, passenger count dim, trip distance dim, rate code dim, pickup location dim and dropoff location dim.
- The fact table contains the numeric data for the Uber database – trip_id, VendorID, pickup_location_id, fare amount, extra, mta tax, tip amount, improvement surcharge etc.
- The payment type dimension contains information about the payment method that the passenger used to pay for the trip – payment_type_id, payment_type and payment type_name.
- This class diagram is used to understand the relationships between the different data elements in the Uber database.

Architecture

- The project is built on Google Cloud Platform (GCP), and it uses a variety of GCP services to collect, store, process, and analyze Uber data.
- The **Raw Data** for the project is collected from a variety of sources, including Uber's mobile app, driver app, and website.
- **ETL** is a process of extracting data from one or more sources, transforming it into a desired format, and loading it into a target system. Our project uses a variety of GCP services to implement its ETL pipeline, including Cloud Data Fusion, Cloud Dataproc, and Cloud Dataprep.



- An open-source website called **MAGE** is utilized as a data pipeline, aiding in the loading of our data and its processing. We will utilize Mage to construct our final dashboard by loading our data onto the massive query that is currently running in the data warehouse.
- **BigQuery** is a fully-managed, petabyte-scale analytics data warehouse that enables businesses to analyze all their data very quickly. It is used to store the transformed data from the ETL pipeline and to run analytics and machine learning jobs on the data.
- **Mage VM** Compute Engine is pre-configured with the tools and libraries that are needed to run machine learning jobs on MAGE. It is also used to run machine learning jobs on the transformed data in BigQuery.
- **Looker** is used to create dashboards and reports that can be used to visualize and analyze the data in BigQuery.

+ Code + Text

Reconnect

```
import pandas as pd

df = pd.read_csv("uber_data.csv")

df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	dropoff_longitude	dr
0	1	2016-03-01 00:00:00	2016-03-01 00:07:55	1	2.50	-73.976746	40.765152	1	N	-74.004265	
1	1	2016-03-01 00:00:00	2016-03-01 00:11:06	1	2.90	-73.983482	40.767925	1	N	-74.005943	
2	2	2016-03-01 00:00:00	2016-03-01 00:31:06	2	19.98	-73.782021	40.644810	1	N	-73.974541	
3	2	2016-03-01 00:00:00	2016-03-01 00:00:00	3	10.78	-73.863419	40.769814	1	N	-73.969650	
4	2	2016-03-01 00:00:00	2016-03-01 00:00:00	5	30.43	-73.971741	40.792183	3	N	-74.177170	

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19858 entries, 0 to 19857
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               19858 non-null  int64
 1   tpep_pickup_datetime   19858 non-null  object
 2   tpep_dropoff_datetime  19858 non-null  object
 3   passenger_count        19858 non-null  int64
 4   trip_distance          19858 non-null  float64
 5   pickup_longitude       19858 non-null  float64
```

+ Code + Text

RAM
Disk

```
[ ] df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
    df['tpep_dropoff_datetime'] = pd.to_datetime(df['tpep_dropoff_datetime'])
```

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               100000 non-null  int64
 1   tpep_pickup_datetime   100000 non-null  object
 2   tpep_dropoff_datetime  100000 non-null  object
 3   passenger_count        100000 non-null  int64
 4   trip_distance          100000 non-null  float64
 5   pickup_longitude       100000 non-null  float64
 6   pickup_latitude        100000 non-null  float64
 7   RatecodeID             100000 non-null  int64
 8   store_and_fwd_flag     100000 non-null  object
 9   dropoff_longitude      100000 non-null  float64
 10  dropoff_latitude       100000 non-null  float64
 11  payment_type           100000 non-null  int64
 12  fare_amount            100000 non-null  float64
 13  extra                  100000 non-null  float64
 14  mta_tax                100000 non-null  float64
 15  tip_amount             100000 non-null  float64
 16  tolls_amount           100000 non-null  float64
 17  improvement_surcharge  100000 non-null  float64
 18  total_amount           100000 non-null  float64
dtypes: float64(12), int64(4), object(3)
memory usage: 14.5+ MB
```

Dropping the duplicates and creating date_time_dim table

+ Code + Text

```
[ ] datetime_dim = df[['tpep_pickup_datetime','tpep_dropoff_datetime']].drop_duplicates().reset_index(drop=True)
    datetime_dim['pick_hour'] = datetime_dim['tpep_pickup_datetime'].dt.hour
    datetime_dim['pick_day'] = datetime_dim['tpep_pickup_datetime'].dt.day
    datetime_dim['pick_month'] = datetime_dim['tpep_pickup_datetime'].dt.month
    datetime_dim['pick_year'] = datetime_dim['tpep_pickup_datetime'].dt.year
    datetime_dim['pick_weekday'] = datetime_dim['tpep_pickup_datetime'].dt.weekday

    datetime_dim['drop_hour'] = datetime_dim['tpep_dropoff_datetime'].dt.hour
    datetime_dim['drop_day'] = datetime_dim['tpep_dropoff_datetime'].dt.day
    datetime_dim['drop_month'] = datetime_dim['tpep_dropoff_datetime'].dt.month
    datetime_dim['drop_year'] = datetime_dim['tpep_dropoff_datetime'].dt.year
    datetime_dim['drop_weekday'] = datetime_dim['tpep_dropoff_datetime'].dt.weekday
```

```
datetime_dim['datetime_id'] = datetime_dim.index
```

```
[ ] datetime_dim = datetime_dim[['datetime_id', 'tpep_pickup_datetime', 'pick_hour', 'pick_day', 'pick_month', 'pick_year', 'pick_weekday',
                                 'tpep_dropoff_datetime', 'drop_hour', 'drop_day', 'drop_month', 'drop_year', 'drop_weekday']]
    #
```

```
[ ] datetime_dim
```

	datetime_id	tpep_pickup_datetime	pick_hour	pick_day	pick_month	pick_year	pick_weekday	tpep_dropoff_datetime	drop_hour	drop_day	drop_month	drop_year	drop_weekday
0	0	2016-03-01 00:00:00	0	1	3	2016	1	2016-03-01 00:07:55	0	1	3	2016	1
1	1	2016-03-01 00:00:00	0	1	3	2016	1	2016-03-01 00:11:06	0	1	3	2016	1
2	2	2016-03-01 00:00:00	0	1	3	2016	1	2016-03-01 00:31:06	0	1	3	2016	1
3	3	2016-03-01 00:00:00	0	1	3	2016	1	2016-03-01 00:00:00	0	1	3	2016	1

Dropping passenger duplicates and creating passenger_count_dim and trip_distance_dim tables

+ Code + Text

RAM
Disk

```
[ ] passenger_count_dim = df[['passenger_count']].drop_duplicates().reset_index(drop=True)
    passenger_count_dim['passenger_count_id'] = passenger_count_dim.index
    passenger_count_dim = passenger_count_dim[['passenger_count_id','passenger_count']]

    trip_distance_dim = df[['trip_distance']].drop_duplicates().reset_index(drop=True)
    trip_distance_dim['trip_distance_id'] = trip_distance_dim.index
    trip_distance_dim = trip_distance_dim[['trip_distance_id','trip_distance']]
```

```
[ ] passenger_count_dim.head()
```

	passenger_count_id	passenger_count
0	0	1
1	1	2
2	2	3
3	3	5
4	4	6

```
[ ] trip_distance_dim.head()
```

	trip_distance_id	trip_distance
0	0	2.50
1	1	2.90
2	2	19.98

Assigning rate as per the data dictionary, creating rate_code_dim table

+ Code + Text

```
[ ] rate_code_type = {
        1:"Standard rate",
        2:"JFK",
        3:"Newark",
        4:"Nassau or Westchester",
        5:"Negotiated fare",
        6:"Group ride"
    }

    rate_code_dim = df[['RatecodeID']].drop_duplicates().reset_index(drop=True)
    rate_code_dim['rate_code_id'] = rate_code_dim.index
    rate_code_dim['rate_code_name'] = rate_code_dim['RatecodeID'].map(rate_code_type)
    rate_code_dim = rate_code_dim[['rate_code_id','RatecodeID','rate_code_name']]
```

```
[ ] rate_code_dim.head()
```

	rate_code_id	RatecodeID	rate_code_name
0	0	1	Standard rate
1	1	3	Newark
2	2	2	JFK
3	3	5	Negotiated fare
4	4	4	Nassau or Westchester

Dropping the duplicates for the pickup & drop, assigning the index and reordering it.
Created payment_type_dim table

+ Code + Text

```
[ ] pickup_location_dim = df[['pickup_longitude', 'pickup_latitude']].drop_duplicates().reset_index(drop=True)
    pickup_location_dim['pickup_location_id'] = pickup_location_dim.index
    pickup_location_dim = pickup_location_dim[['pickup_location_id','pickup_latitude','pickup_longitude']]


    dropoff_location_dim = df[['dropoff_longitude', 'dropoff_latitude']].drop_duplicates().reset_index(drop=True)
    dropoff_location_dim['dropoff_location_id'] = dropoff_location_dim.index
    dropoff_location_dim = dropoff_location_dim[['dropoff_location_id','dropoff_latitude','dropoff_longitude']]
```

```
[ ] payment_type_name = {
        1:"Credit card",
        2:"Cash",
        3:"No charge",
        4:"Dispute",
        5:"Unknown",
        6:"Voided trip"
    }
    payment_type_dim = df[['payment_type']].drop_duplicates().reset_index(drop=True)
    payment_type_dim['payment_type_id'] = payment_type_dim.index
    payment_type_dim['payment_type_name'] = payment_type_dim['payment_type'].map(payment_type_name)
    payment_type_dim = payment_type_dim[['payment_type_id','payment_type','payment_type_name']]
```

```
[ ] payment_type_dim.head()
```

	payment_type_id	payment_type	payment_type_name
0	0	1	Credit card
1	1	2	Cash

Converting flat file into dimension model for the ‘fact table’

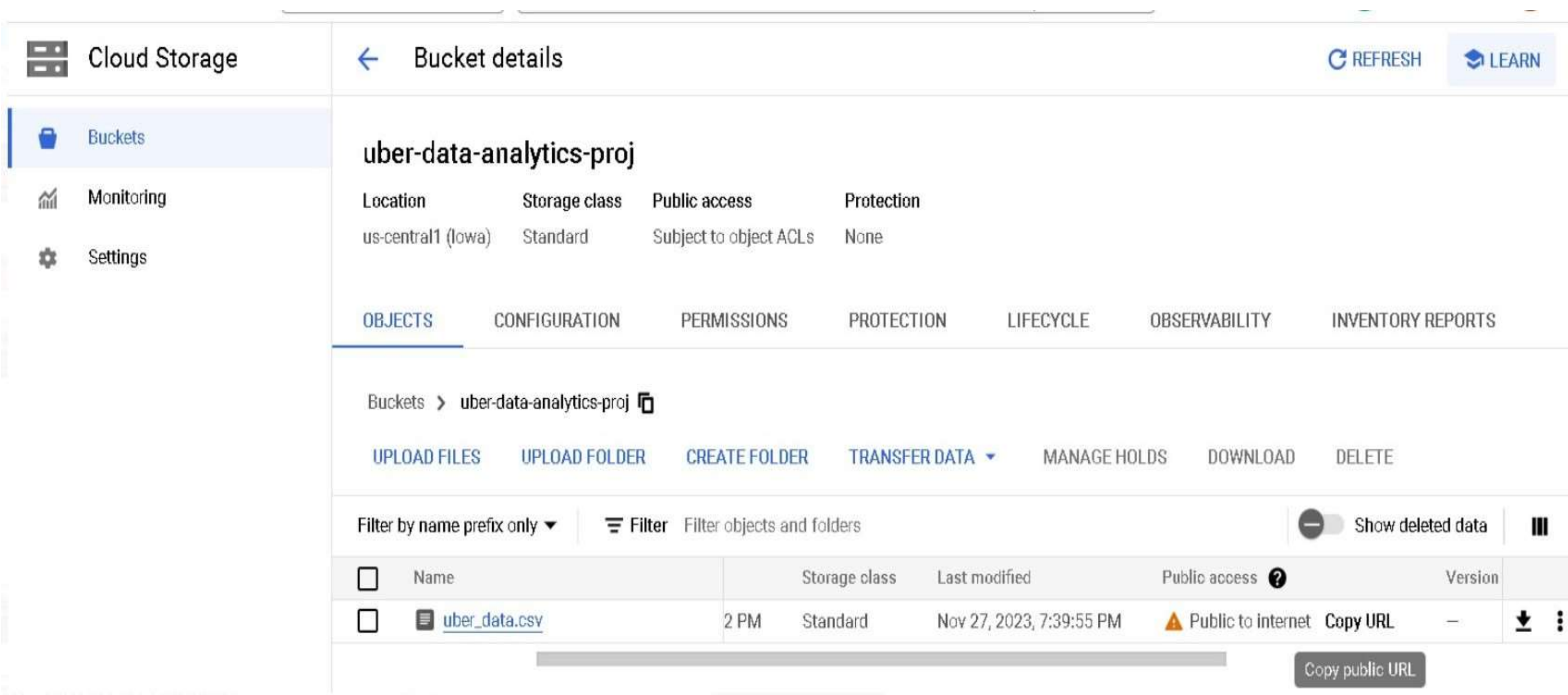
+ Code + Text

```
fact_table = df.merge(passenger_count_dim, on='passenger_count') \
             .merge(trip_distance_dim, on='trip_distance') \
             .merge(rate_code_dim, on='RatecodeID') \
             .merge(pickup_location_dim, on=['pickup_longitude', 'pickup_latitude']) \
             .merge(dropoff_location_dim, on=['dropoff_longitude', 'dropoff_latitude'])\
             .merge(datetime_dim, on=['tpep_pickup_datetime','tpep_dropoff_datetime']) \
             .merge(payment_type_dim, on='payment_type') \
             [['VendorID', 'datetime_id', 'passenger_count_id',
               'trip_distance_id', 'rate_code_id', 'store_and_fwd_flag', 'pickup_location_id', 'dropoff_location_id',
               'payment_type_id', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
               'improvement_surcharge', 'total_amount']]
```

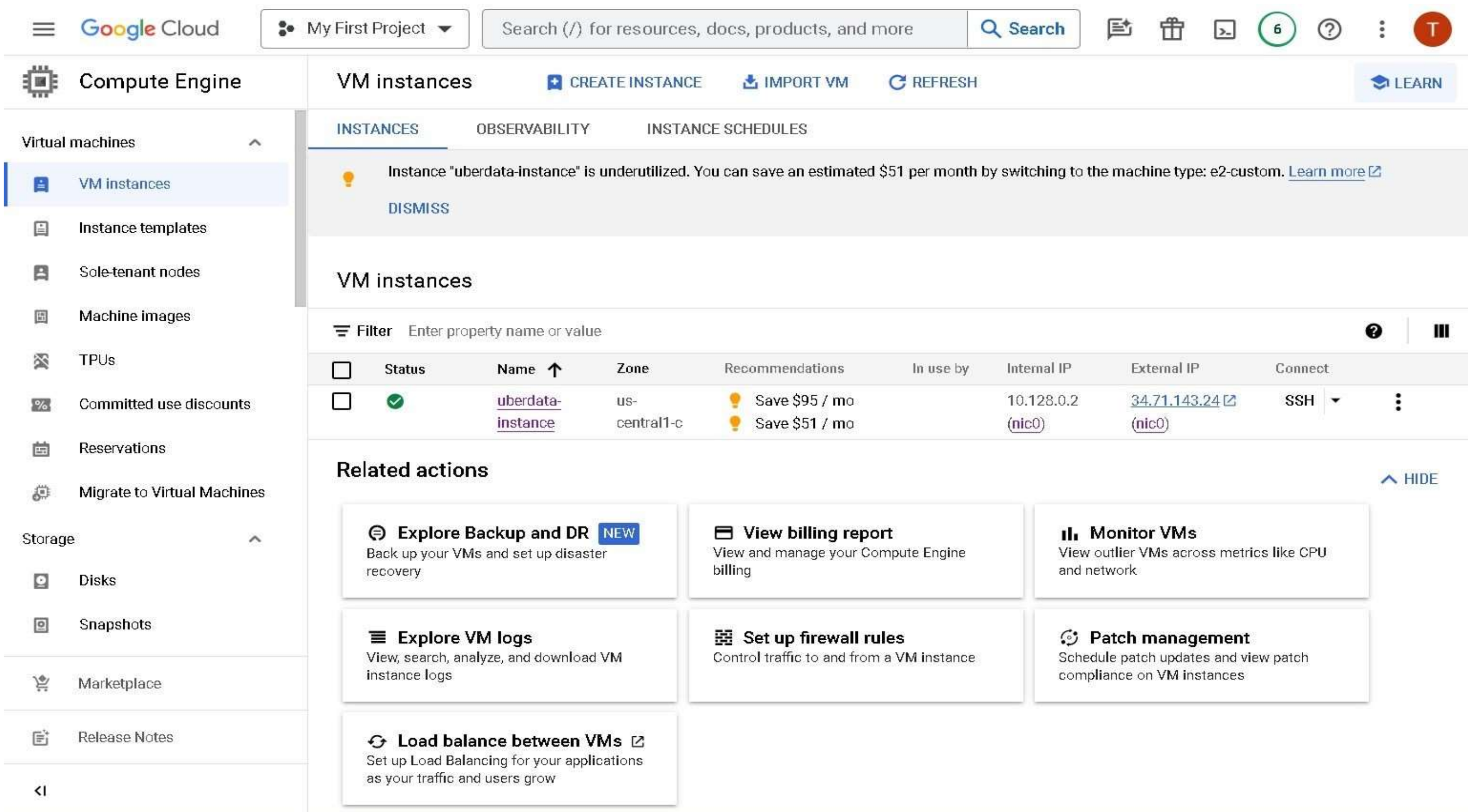
```
[ ] fact_table
```

	VendorID	datetime_id	passenger_count_id	trip_distance_id	rate_code_id	store_and_fwd_flag	pickup_location_id	dropoff_location_id	payment_type_id	fare_amount	extra
0	1	0	0	0	0	N	0	0	0	9.0	0.5
1	2	1491	0	0	0	N	1481	1484	0	10.5	0.0
2	2	2834	0	0	0	N	2816	2819	0	9.5	0.0
3	2	3488	0	0	0	N	3465	3470	0	13.5	0.0
4	2	3923	0	0	0	N	3899	3903	0	10.5	0.0
...	...	...	...	...	...	...	...	...	...	...	...
99995	1	65943	0	257	3	N	64896	65105	3	170.0	0.0
99996	1	81651	0	257	3	N	80276	80547	3	10.0	0.0
99997	2	87152	4	257	1	N	85670	85971	3	-20.0	-0.5

Uber_data.csv file has been uploaded on the cloud storage.



VM instance


Google Cloud
My First Project
Search (/) for resources, docs, products, and more
Search
Compute Engine
VM instances
CREATE INSTANCE
IMPORT VM
REFRESH
LEARN
Virtual machines
VM instances
Instance templates
Sole-tenant nodes
Machine images
TPUs
Committed use discounts
Reservations
Migrate to Virtual Machines
Storage
Disks
Snapshots
Marketplace
Release Notes
INSTANCES
OBSERVABILITY
INSTANCE SCHEDULES
Instance "uberdata-instance" is underutilized. You can save an estimated $51 per month by switching to the machine type: e2-custom. Learn more
DISMISS
VM instances
Filter
Enter property name or value
Status
Name
Zone
Recommendations
In use by
Internal IP
External IP
Connect
uberdata-instance
us-central1-c
Save $95 / mo
Save $51 / mo
10.128.0.2 (nic0)
34.71.143.24 (nic0)
SSH
Related actions
HIDE
Explore Backup and DR
NEW
Back up your VMs and set up disaster recovery
View billing report
View and manage your Compute Engine billing
Monitor VMs
View outlier VMs across metrics like CPU and network
Explore VM logs
View, search, analyze, and download VM instance logs
Set up firewall rules
Control traffic to and from a VM instance
Patch management
Schedule patch updates and view patch compliance on VM instances
Load balance between VMs
Set up Load Balancing for your applications as your traffic and users grow

Implementation on MAGE

PY TRANSFORMER uber_transformation ←o 1 parent

Positional arguments for decorated function:

```
@transformer
def transform(data):
    data → load_uber_data
```

```python
import pandas as pd
if 'transformer' not in globals():
    from mage_ai.data_preparation.decorators import transformer
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test


@transformer
def transform(df, *args, **kwargs):
    """
    Template code for a transformer block.

    Add more parameters to this function if this block has multiple parent blocks.
    There should be one parameter for each output variable from each parent block.

    Args:
        data: The output from the upstream parent block
        args: The output from any additional upstream blocks (if applicable)

    Returns:
        Anything (e.g. data frame, dictionary, array, int, str, etc.)
    """
    # Specify your transformation logic here
    df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
```

CPU: 0% / Memory: 46.98MB Last saved 2023-12-06T21:00:50+00:0

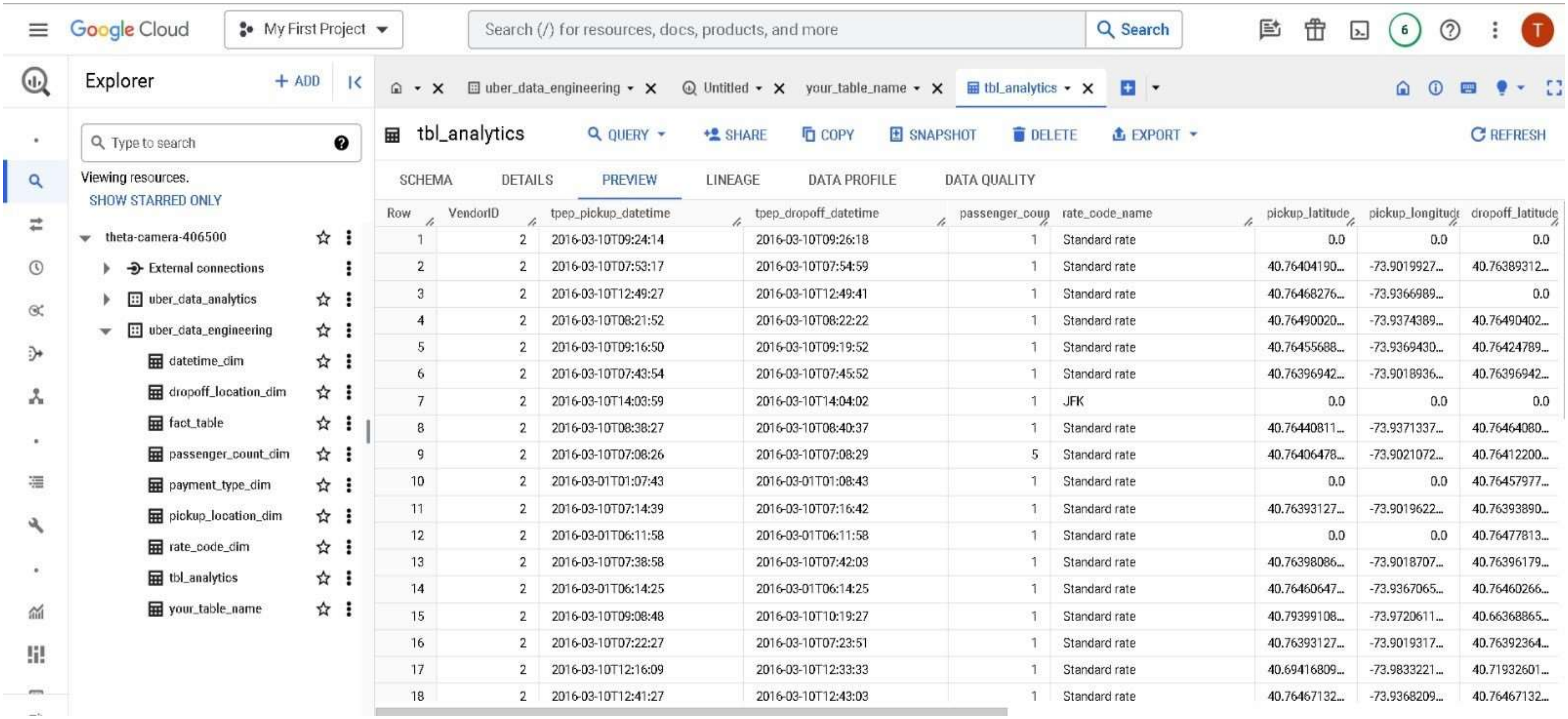
PY DATA LOADER load_uber_data Edit parents

```python
import io
import pandas as pd
import requests
if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test


@data_loader
def load_data_from_api(*args, **kwargs):
    """
    Template for loading data from API
    """
    url = 'https://storage.googleapis.com/uber-data-analytics-proj/uber_data.csv'
    response = requests.get(url)

    return pd.read_csv(io.StringIO(response.text), sep=',')


@test
def test_output(output, *args) -> None:
    """
    Template code for testing the output of the block.
    """
    assert output is not None, 'The output is undefined'
```

PY DATA EXPORTER uber_bigquery_load 1 parent

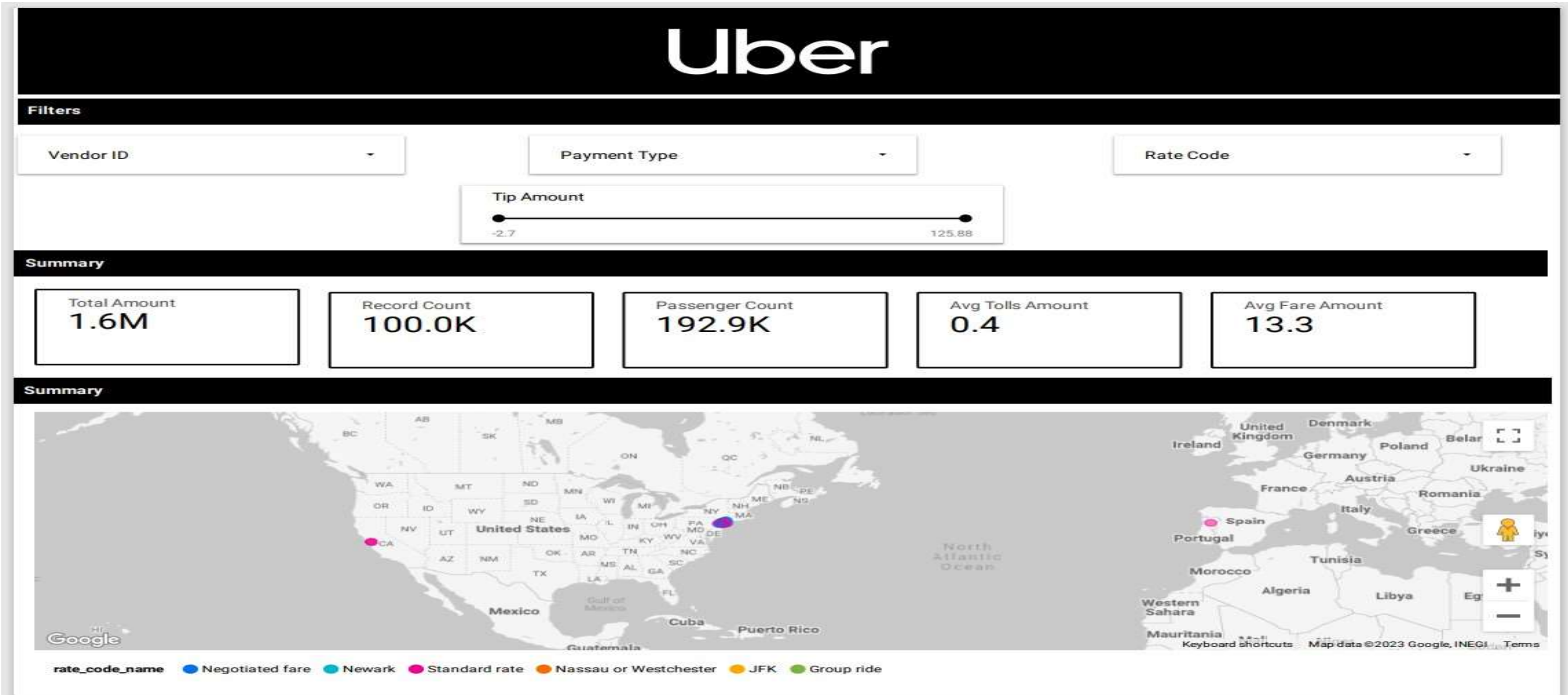
```python
from mage_ai.io.bigquery import BigQuery
from mage_ai.io.config import ConfigFileLoader
from pandas import DataFrame
from os import path

if 'data_exporter' not in globals():
    from mage_ai.data_preparation.decorators import data_exporter


@data_exporter
def export_data_to_big_query(df: DataFrame, **kwargs) -> None:
    """
    Template for exporting data to a BigQuery warehouse.
    Specify your configuration settings in 'io_config.yaml'.

    Docs: https://docs.mage.ai/design/data-loading#bigquery

    """

    config_path = path.join(get_repo_path(), 'io_config.yaml')
    config_profile = 'default'

    for key, value in data.items():
        table_id = 'theta-camera-406500.uber_de {}'.format(key)
        BigQuery.with_config(ConfigFileLoader(config_path, config_profile)).export(
            DataFrame(value),
            table_id,
            if_exists='replace',  # Specify resolution policy if table name already exist
        )
```

BigQuery



Google Cloud
My First Project
Search (/) for resources, docs, products, and more
Search
Explorer
+ ADD
Type to search
Viewing resources.
SHOW STARRED ONLY
theta-camera-406500
External connections
uber_data_analytics
uber_data_engineering
datetime_dim
dropoff_location_dim
fact_table
passenger_count_dim
payment_type_dim
pickup_location_dim
rate_code_dim
tbl_analytics
your_table_name
uber_data_engineering
Untitled
your_table_name
tbl_analytics
tbl_analytics
QUERY
SHARE
COPY
SNAPSHOT
DELETE
EXPORT
REFRESH
SCHEMA
DETAILS
PREVIEW
LINEAGE
DATA PROFILE
DATA QUALITY
Row | VendorID | tpep_pickup_datetime | tpep_dropoff_datetime | passenger_coun | rate_code_name | pickup_latitude | pickup_longitude | dropoff_latitude
1 | 2 | 2016-03-10T09:24:14 | 2016-03-10T09:26:18 | 1 | Standard rate | 0.0 | 0.0 | 0.0
2 | 2 | 2016-03-10T07:53:17 | 2016-03-10T07:54:59 | 1 | Standard rate | 40.76404190... | -73.9019927... | 40.76389312...
3 | 2 | 2016-03-10T12:49:27 | 2016-03-10T12:49:41 | 1 | Standard rate | 40.76468276... | -73.9366989... | 0.0
4 | 2 | 2016-03-10T08:21:52 | 2016-03-10T08:22:22 | 1 | Standard rate | 40.76490020... | -73.9374389... | 40.76490402...
5 | 2 | 2016-03-10T09:16:50 | 2016-03-10T09:19:52 | 1 | Standard rate | 40.76455688... | -73.9369430... | 40.76424789...
6 | 2 | 2016-03-10T07:43:54 | 2016-03-10T07:45:52 | 1 | Standard rate | 40.76396942... | -73.9018936... | 40.76396942...
7 | 2 | 2016-03-10T14:03:59 | 2016-03-10T14:04:02 | 1 | JFK | 0.0 | 0.0 | 0.0
8 | 2 | 2016-03-10T08:38:27 | 2016-03-10T08:40:37 | 1 | Standard rate | 40.76440811... | -73.9371337... | 40.76464080...
9 | 2 | 2016-03-10T07:08:26 | 2016-03-10T07:08:29 | 5 | Standard rate | 40.76406478... | -73.9021072... | 40.76412200...
10 | 2 | 2016-03-01T01:07:43 | 2016-03-01T01:08:43 | 1 | Standard rate | 0.0 | 0.0 | 40.76457977...
11 | 2 | 2016-03-10T07:14:39 | 2016-03-10T07:16:42 | 1 | Standard rate | 40.76393127... | -73.9019622... | 40.76393890...
12 | 2 | 2016-03-01T06:11:58 | 2016-03-01T06:11:58 | 1 | Standard rate | 0.0 | 0.0 | 40.76477813...
13 | 2 | 2016-03-10T07:38:58 | 2016-03-10T07:42:03 | 1 | Standard rate | 40.76398086... | -73.9018707... | 40.76396179...
14 | 2 | 2016-03-01T06:14:25 | 2016-03-01T06:14:25 | 1 | Standard rate | 40.76460647... | -73.9367065... | 40.76460266...
15 | 2 | 2016-03-10T09:08:48 | 2016-03-10T10:19:27 | 1 | Standard rate | 40.79399108... | -73.9720611... | 40.66368865...
16 | 2 | 2016-03-10T07:22:27 | 2016-03-10T07:23:51 | 1 | Standard rate | 40.76393127... | -73.9019317... | 40.76392364...
17 | 2 | 2016-03-10T12:16:09 | 2016-03-10T12:33:33 | 1 | Standard rate | 40.69416809... | -73.9833221... | 40.71932601...
18 | 2 | 2016-03-10T12:41:27 | 2016-03-10T12:43:03 | 1 | Standard rate | 40.76467132... | -73.9368209... | 40.76467132...

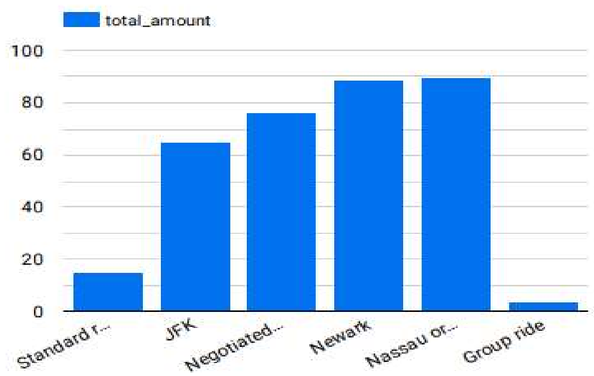
Uber Dashboard

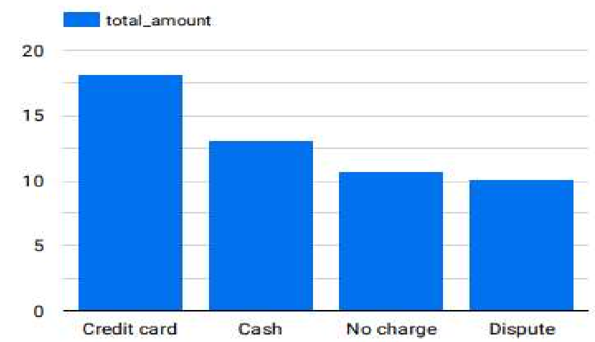


The dashboard can be used to track the performance of Uber in different regions and to identify trends in Uber usage. The dashboard is a valuable tool and it can be used to improve Uber's services and to make informed decisions about the future of the company.

Bar charts displaying different rate code and payment methods.

Charts





The chart shows that credit card is the most popular payment method for Uber rides, and Cash is the second most popular payment method.

Conclusion

This large-scale experiment demonstrated the concrete benefits that may be obtained from using big data techniques to comprehend client happiness. Significant insights were obtained from Uber's enormous amounts of unstructured transactional data by integrating scalable and adaptable data pipelines, storage, processing, modeling, analysis, and visualization components. The project's ultimate success was supported by several significant accomplishments, including transferring streams of unprocessed transportation data from dispersed systems into uniform storage, including trips, locations, timings, money, and metadata, Creating frameworks such as the star schema to connect atomic facts and dimensions, improving the accuracy of data, Developing an enterprise-class ETL method to convert massive amounts of detailed data into datasets suitable for analysis, Statistical modeling in conjunction with interactive slicing and dicing to facilitate multidimensional analysis, providing shared dashboards with important patterns and trends to inform strategic choices. Uber's customer-centricity was enhanced by these successes, which gave the initiative concrete insight into passenger preferences, habits, evolving demands, and pain areas throughout travel. It was possible to pinpoint exactly which key parts were causing the friction. Loyalty and promotional campaigns were found to be related. It was estimated how sensitive price thresholds were. In addition, the baseline dashboards, model designs, templated ETL logic, and architecture blueprint created throughout this project help expedite further analytics endeavours. Uber's culture may become more deeply ingrained with data-driven analysis if ownership is transferred to business teams.

References:

- Cloud Data Warehousing Solutions https://cloud.google.com/solutions/big-data/
- Schema Design for Analytical Workloads https://cloud.google.com/architecture/schema-design-for-analytical-workloads
- Streaming Analytics Pipeline Architecture https://cloud.google.com/solutions/big-data/streaming-analytics-pipeline-architecture
- Uber's Methodology for Experimentation at Scale https://eng.uber.com/xp/
- Recommending Optimal Partners for Uber Service Providers https://dl.acm.org/doi/abs/10.1145/3298689.3346969
- Uber Data Analysis & Visualization Case Study https://towardsdatascience.com/uber-data-analysis-visualization-db5759f51b5a
- Optimized Bucketing for Uber's Real-time Features Platform https://dl.acm.org/doi/10.14778/3436905.3436924
- Ride Sharing Innovation & Managing Emerging Technologies https://www.emerald.com/insight/content/doi/10.1108/JMTM-06-2018-0133/full/html