

Uber data Engineering project



@DHRUV KUMAR
FOLLOW ME ON



Project Summary.

In this project, I successfully built a data engineering pipeline using **Pandas**, **Google Cloud**, **Mage**, **BigQuery** and **LookerStudio**. The key steps included:

- **Data Cleaning & Transformation:** Removed duplicates and created structured dimensions such as date, time, day, month, pick hours, etc.. using Pandas.
- **Data Modeling:** Designed a data modeling structure for dimensional analysis.
- **Cloud Integration:** Set up an instance on Google Cloud and used Mage to build an automated data pipeline.
- **Data Export & Processing:** Exported the transformed data to BigQuery using the configure credentials .
- **Data Merging & Visualization:** Merged multiple tables and imported the final dataset into **Looker Studio** for visualization and insights.

This project streamlined data processing, storage, and visualization, making it more efficient and scalable.

Transform Data using pandas

Uber data cleaning

```
import pandas as pd #Import pandas df = pd.read_csv('uber_data.csv') #read
csv file via pandas df.head() #fetch the first five rows df.info() #information
all the dataset
```

pickup and dropoff date are not correct data type so we can use the pandas datetime fuction.

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

```
df.info()
```

```
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_id','tpep_pickup_datetime','pick_hour','pick_d
ay','pick_month','pick_year','pick_weekday','tpep_dropoff_datetime',
```

```
'drop_hour','drop_day','drop_month','drop_year','drop_weekday']]

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

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

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['drop_location_id'] = dropoff_location_dim.index
dropoff_location_dim =
dropoff_location_dim[['drop_location_id','dropoff_longitude','dropoff_latitude']]
```

```
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

fact_table = df.merge(passenger_count_dim, on='passenger_count')\

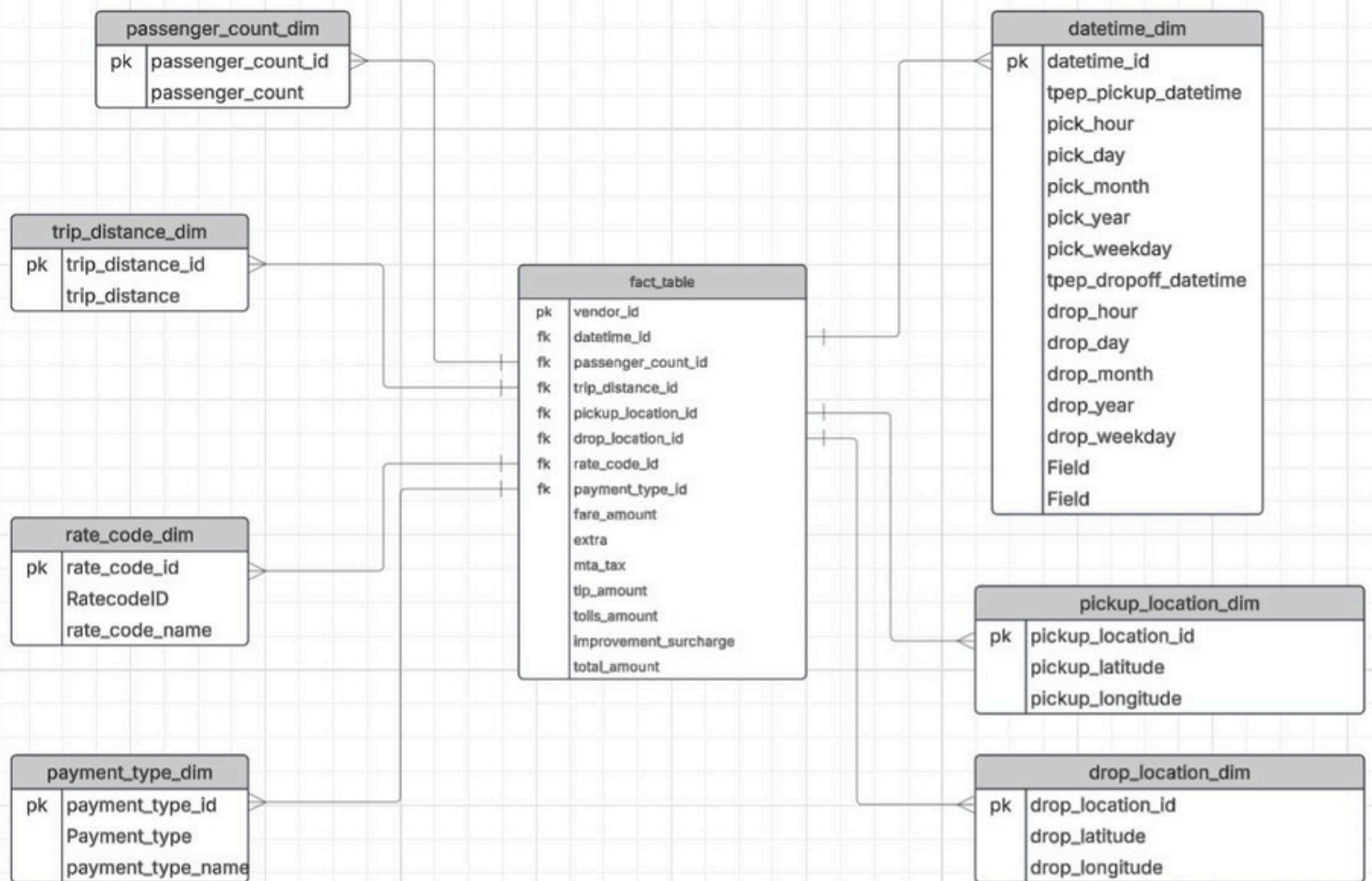
                .merge(trip_distance_dim, on='trip_distance')\
                .merge(rate_code_dim, on='RatecodeID')\
                .merge(pickup_location_dim,
on=['pickup_longitude', 'pickup_latitude'])\
                .merge(dropoff_location_dim,
on=['dropoff_longitude', 'dropoff_latitude'])\
                .merge(datetime_dim,
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_locatio
n_id', 'drop_location_id', 'payment_type_id',

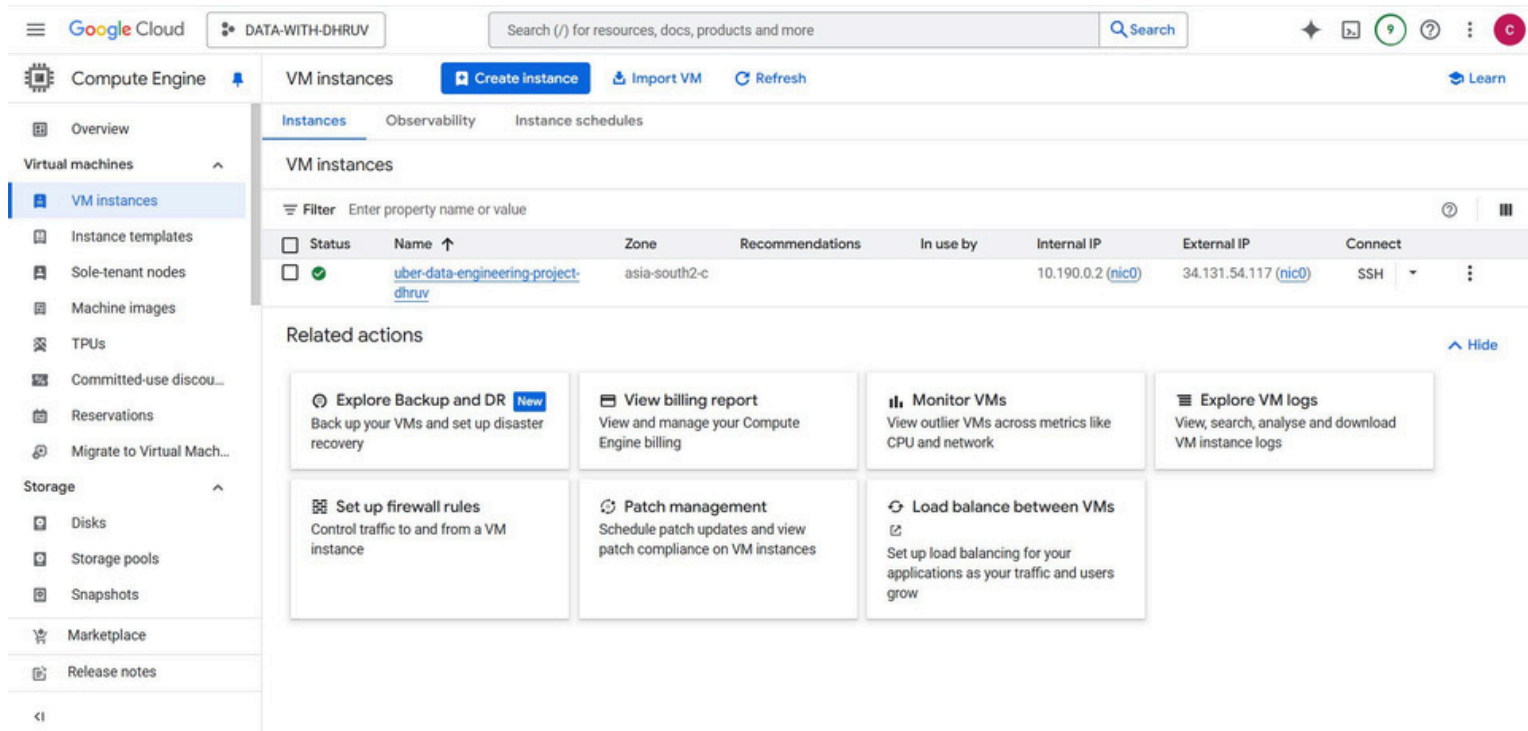
'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improveme
nt_surcharge', 'total_amount']]

fact_table
```

Data Modeling

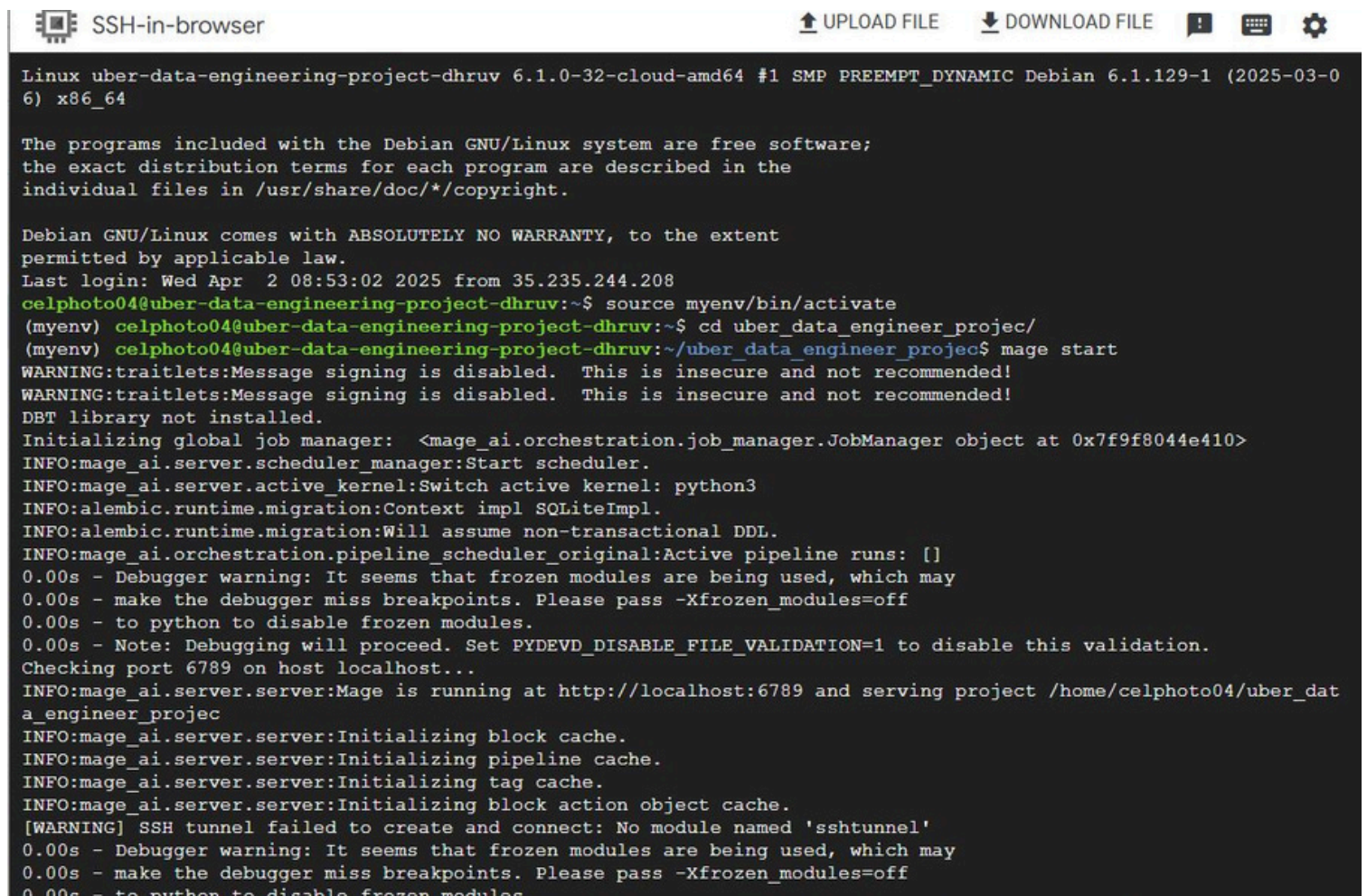


• Set up an instance



The screenshot shows the Google Cloud console interface for VM instances. The left sidebar contains navigation links for Compute Engine, Virtual machines, Instance templates, Sole-tenant nodes, Machine images, TPUs, Committed-use discounts, Reservations, and Migrate to Virtual Machine. The main content area displays a table of VM instances. The table has columns for Status, Name, Zone, Recommendations, In use by, Internal IP, External IP, and Connect. A single instance is listed with the name 'uber-data-engineering-project-dhruv' in the 'Name' column, 'asia-south2-c' in the 'Zone' column, and '10.190.0.2 (nic0)' in the 'Internal IP' column. Below the table, there are several 'Related actions' cards, including 'Explore Backup and DR', 'View billing report', 'Monitor VMs', 'Explore VM logs', 'Set up firewall rules', 'Patch management', and 'Load balance between VMs'.

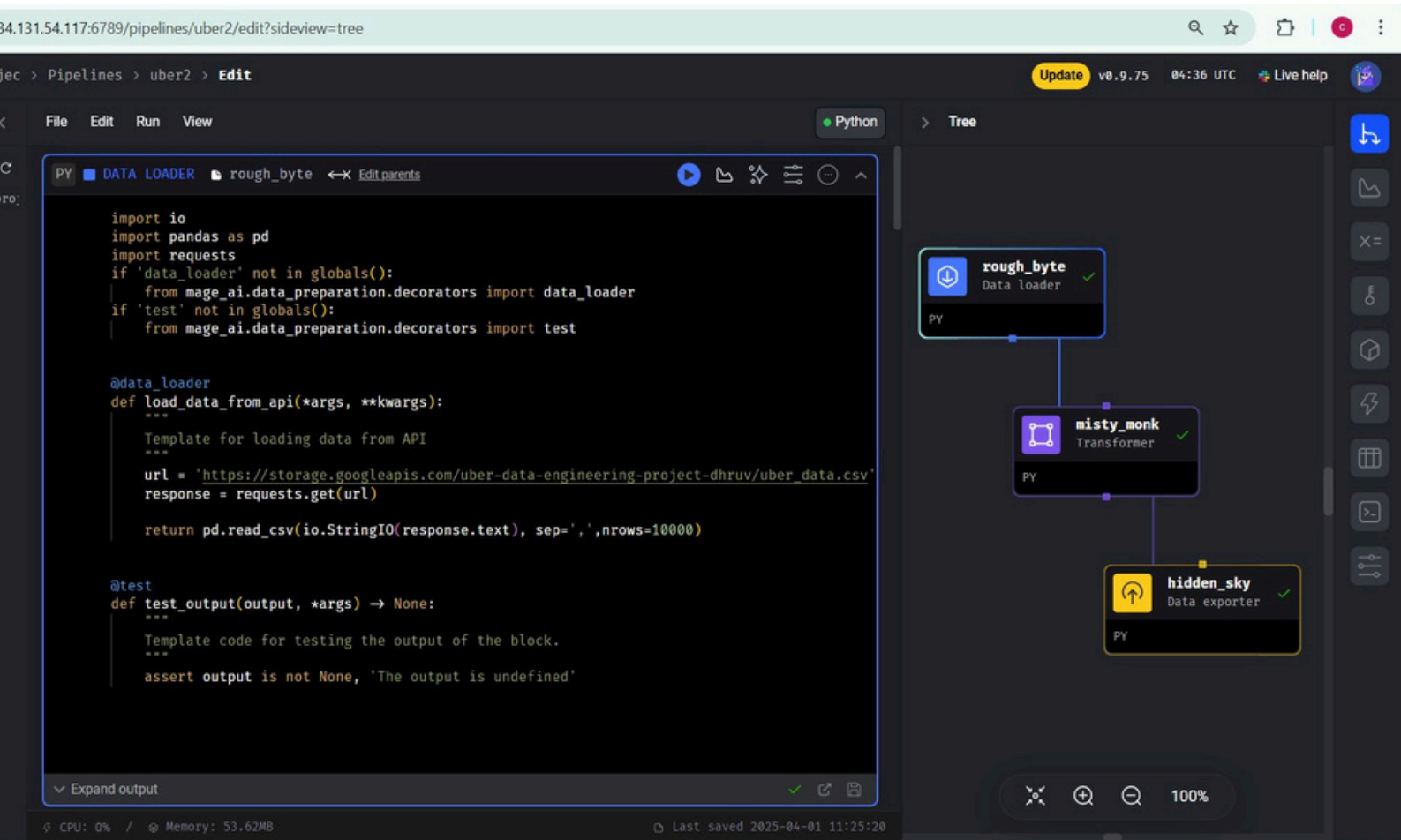
• VM Connect to Mage



The screenshot shows a terminal window titled 'SSH-in-browser' with a dark background and light-colored text. The terminal output shows the user 'celphoto04' logging into the VM 'uber-data-engineering-project-dhruv'. The user runs the command 'source myenv/bin/activate' and then 'mage start'. The terminal output shows the Mage AI server starting up, including messages about message signing, global job manager, scheduler, and pipeline runs. The terminal output also shows a warning about the SSH tunnel failing to create and connect, and a message about the debugger warning.

Mage to build an automated data pipeline.

- Created Data Loader File to Load Data



The screenshot shows the Mage IDE interface. On the left, a code editor displays a Python file named `DATA LOADER` with the following code:

```
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

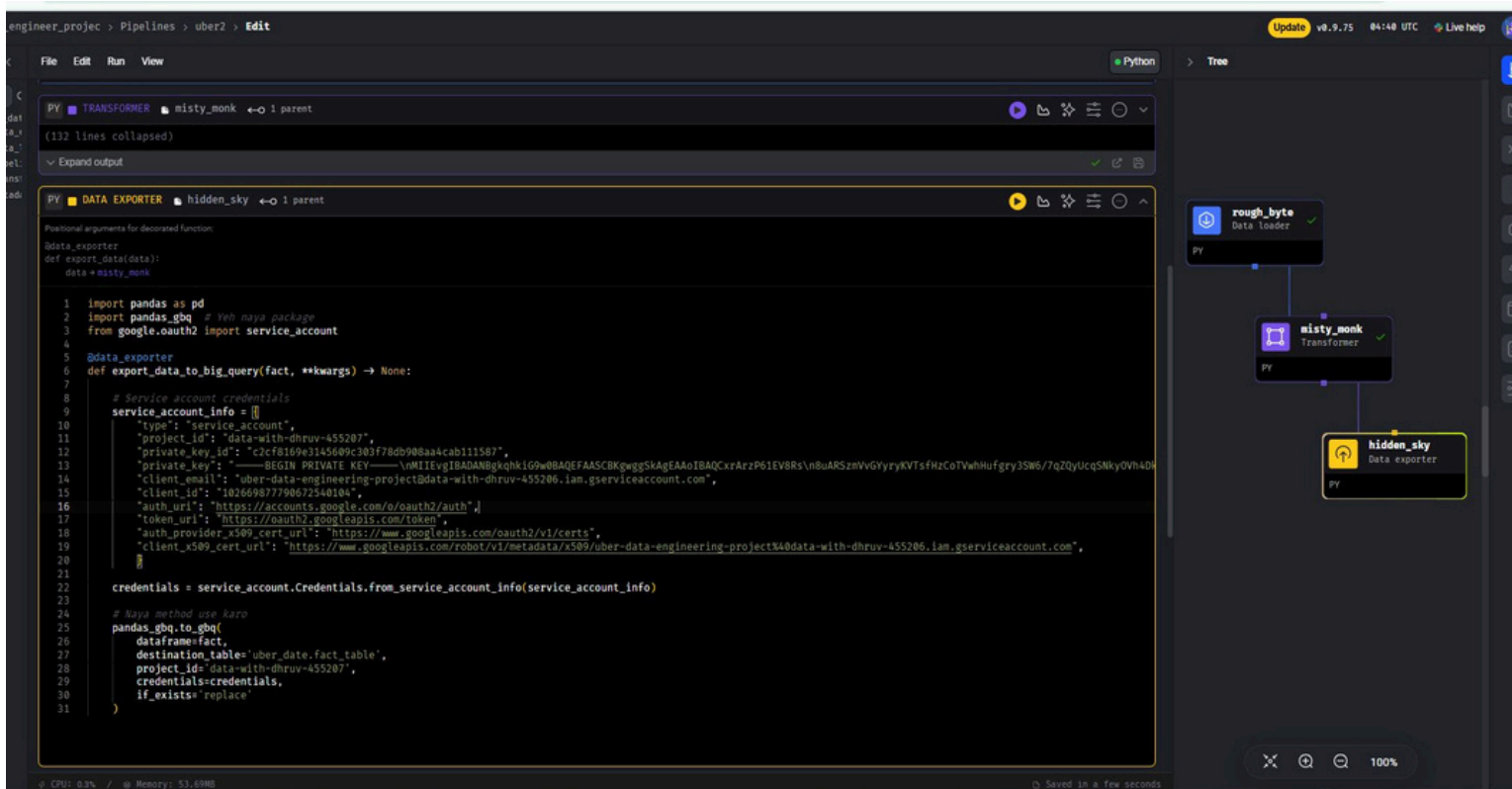
@data_loader
def load_data_from_api(*args, **kwargs):
    """
    Template for loading data from API
    """
    url = 'https://storage.googleapis.com/uber-data-engineering-project-dhruv/uber_data.csv'
    response = requests.get(url)

    return pd.read_csv(io.StringIO(response.text), sep=',', nrows=10000)

@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'
```

On the right, a pipeline diagram shows three blocks connected in sequence: `rough_byte` (Data loader), `misty_monk` (Transformer), and `hidden_sky` (Data exporter). The interface also shows a file explorer on the left and a status bar at the bottom indicating CPU usage, memory usage, and the last save time.

- Data Export to BigQuery using the configure credentials .



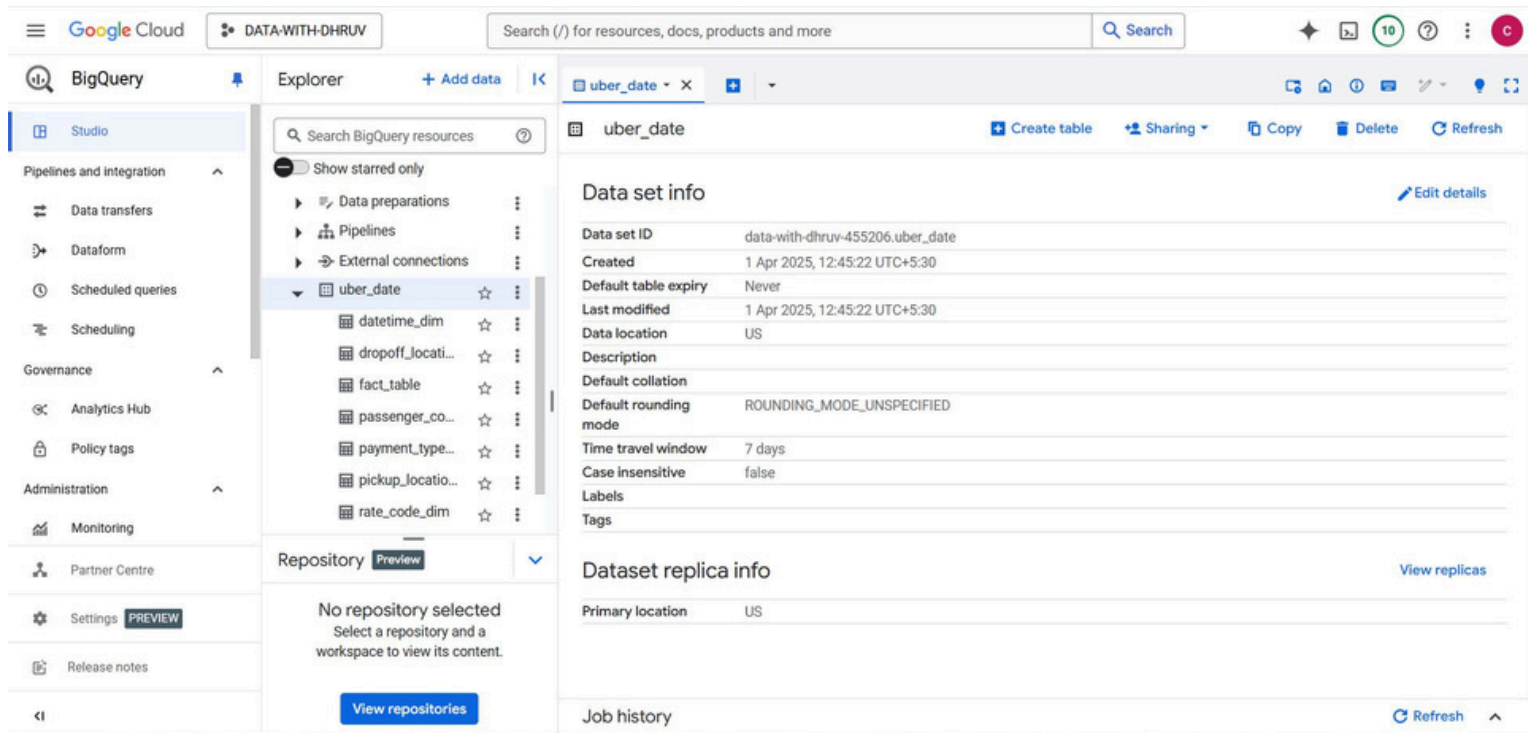
The screenshot shows the Mage IDE interface. On the left, a code editor displays a Python file named `DATA EXPORTER` with the following code:

```
Positional arguments for decorated function.
@data_exporter
def export_data(data):
    data = misty_monk

1 import pandas as pd
2 import pandas_gbq # Yeh naya package
3 from google.oauth2 import service_account
4
5 @data_exporter
6 def export_data_to_big_query(fact, **kwargs) → None:
7
8     # Service account credentials
9     service_account_info = {
10         'type': 'service_account',
11         'project_id': 'data-with-dhruv-455207',
12         'private_key_id': 'c2cf8169e3145609c303f78db908aa4cab111587',
13         'private_key': '-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANBgkqhkiG9w0BAQEFAAQCAQAwIhBACxArzP61EVBRS\nn8uARSznVvGYryXVTSfHzCoTVvHufgry3SW6/7qZQyUcQSNkyOVH4D\n14         'client_email': 'uber-data-engineering-project@data-with-dhruv-455206.iam.gserviceaccount.com',
15         'client_id': '10266987798072540104',
16         'auth_url': 'https://accounts.google.com/o/oauth2/auth',
17         'token_url': 'https://oauth2.googleapis.com/token',
18         'auth_provider_x509_cert_url': 'https://www.googleapis.com/oauth2/v1/certs',
19         'client_x509_cert_url': 'https://www.googleapis.com/robot/v1/metadata/x509/uber-data-engineering-project@data-with-dhruv-455206.iam.gserviceaccount.com',
20     }
21
22     credentials = service_account.Credentials.from_service_account_info(service_account_info)
23
24     # Naya method use karo
25     pandas_gbq.to_gbq(
26         dataframe=fact,
27         destination_table='uber_data.fact_table',
28         project_id='data-with-dhruv-455207',
29         credentials=credentials,
30         if_exists='replace'
31     )
```

On the right, a pipeline diagram shows three blocks connected in sequence: `rough_byte` (Data loader), `misty_monk` (Transformer), and `hidden_sky` (Data exporter). The interface also shows a file explorer on the left and a status bar at the bottom indicating CPU usage, memory usage, and the last save time.

• Data Tables



The screenshot shows the Google Cloud BigQuery Studio interface. The left sidebar contains navigation options like Studio, Pipelines and integration, Data transfers, Dataform, Scheduled queries, Scheduling, Governance, Analytics Hub, Policy tags, Administration, Monitoring, Partner Centre, Settings, and Release notes. The main area is divided into three sections: Explorer, Data set info, and Dataset replica info.

Explorer: Shows a tree view of BigQuery resources. The 'uber_date' dataset is selected, showing its contents: datetime_dim, dropoff_locati..., fact_table, passenger_co..., payment_type..., pickup_locatio..., and rate_code_dim.

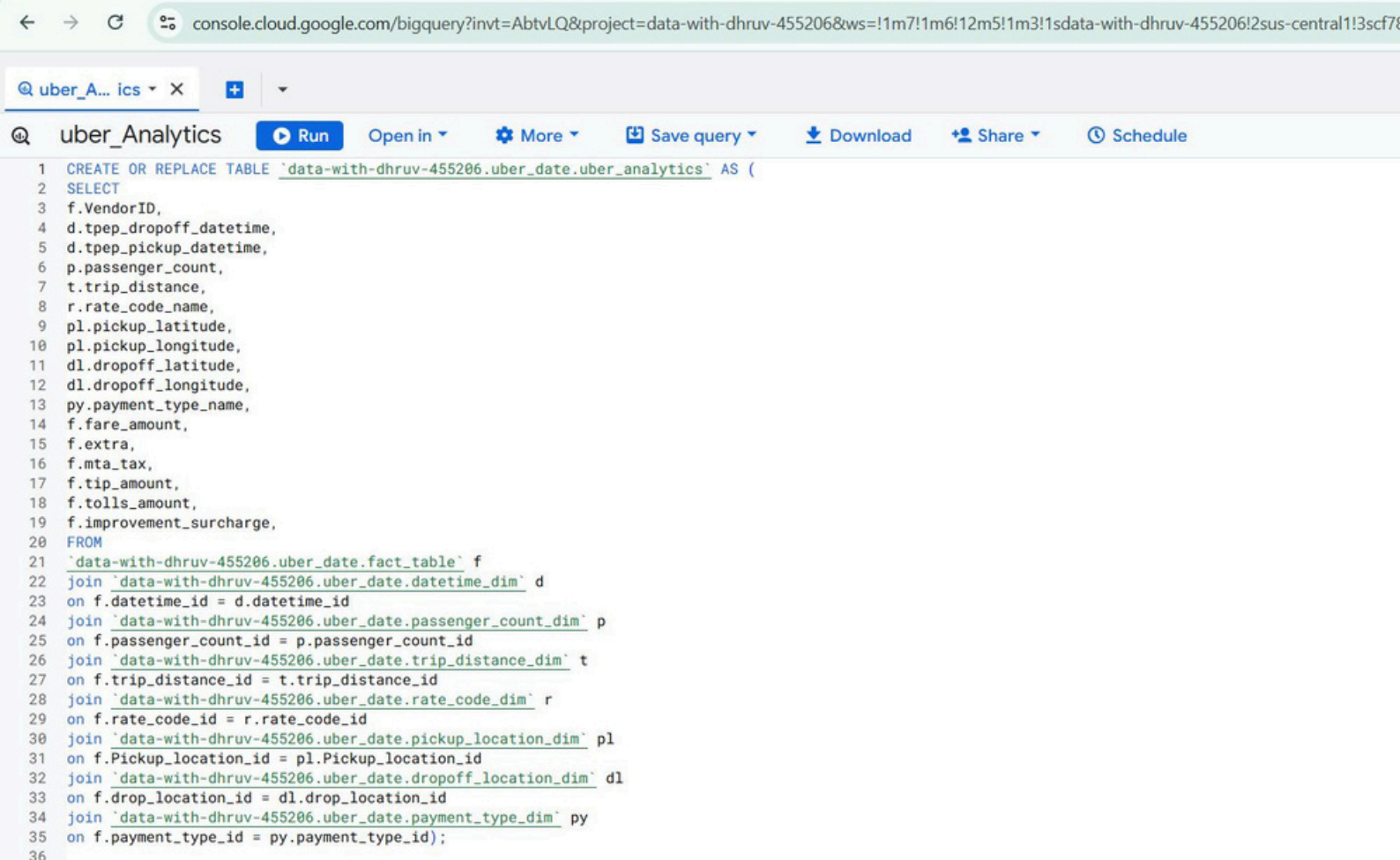
Data set info: Displays details for the 'uber_date' dataset.

Property	Value
Data set ID	data-with-dhruv-455206.uber_date
Created	1 Apr 2025, 12:45:22 UTC+5:30
Default table expiry	Never
Last modified	1 Apr 2025, 12:45:22 UTC+5:30
Data location	US
Description	
Default collation	
Default rounding mode	ROUNDING_MODE_UNSPECIFIED
Time travel window	7 days
Case insensitive	false
Labels	
Tags	

Dataset replica info: Shows the primary location as US.

Repository: A message states 'No repository selected. Select a repository and a workspace to view its content.' with a 'View repositories' button.

• Merged multiple tables



The screenshot shows the Google Cloud BigQuery console with a SQL query to merge multiple tables. The query is as follows:

```
1 CREATE OR REPLACE TABLE `data-with-dhruv-455206.uber_date.uber_analytics` AS (
2   SELECT
3     f.vendor_id,
4     d.tpep_dropoff_datetime,
5     d.tpep_pickup_datetime,
6     p.passenger_count,
7     t.trip_distance,
8     r.rate_code_name,
9     pl.pickup_latitude,
10    pl.pickup_longitude,
11    dl.dropoff_latitude,
12    dl.dropoff_longitude,
13    py.payment_type_name,
14    f.fare_amount,
15    f.extra,
16    f.mta_tax,
17    f.tip_amount,
18    f.tolls_amount,
19    f.improvement_surcharge,
20    FROM
21    `data-with-dhruv-455206.uber_date.fact_table` f
22    JOIN `data-with-dhruv-455206.uber_date.datetime_dim` d
23      ON f.datetime_id = d.datetime_id
24    JOIN `data-with-dhruv-455206.uber_date.passenger_count_dim` p
25      ON f.passenger_count_id = p.passenger_count_id
26    JOIN `data-with-dhruv-455206.uber_date.trip_distance_dim` t
27      ON f.trip_distance_id = t.trip_distance_id
28    JOIN `data-with-dhruv-455206.uber_date.rate_code_dim` r
29      ON f.rate_code_id = r.rate_code_id
30    JOIN `data-with-dhruv-455206.uber_date.pickup_location_dim` pl
31      ON f.Pickup_location_id = pl.Pickup_location_id
32    JOIN `data-with-dhruv-455206.uber_date.dropoff_location_dim` dl
33      ON f.drop_location_id = dl.drop_location_id
34    JOIN `data-with-dhruv-455206.uber_date.payment_type_dim` py
35      ON f.payment_type_id = py.payment_type_id);
36
```

Uber

Vendor ID

Payment Type

Rate Co...

Trip Distance
045.68

Clean filter

Download repor

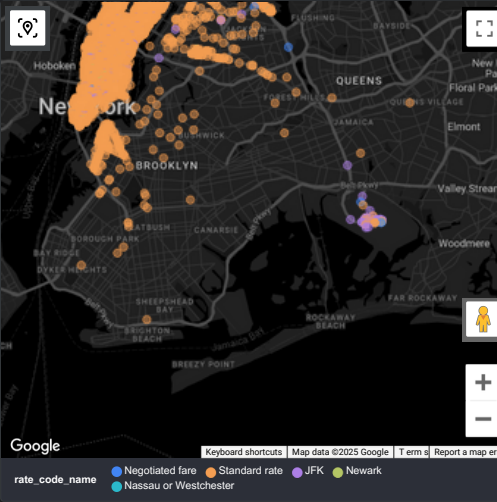
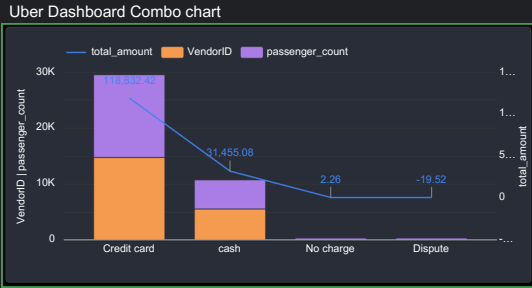
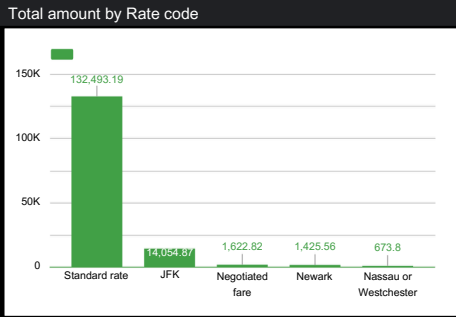
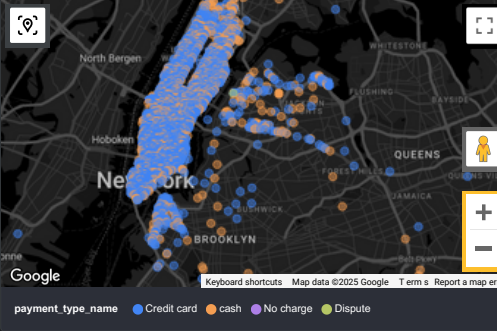
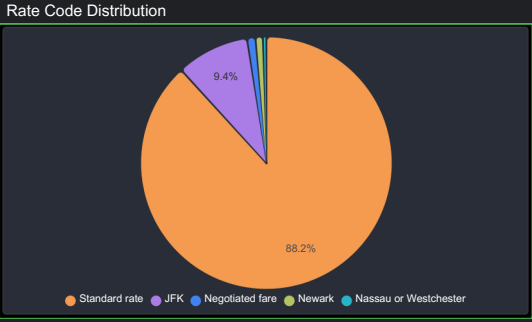
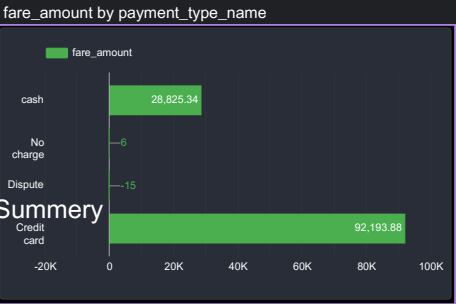
Summery

Card

Total Revenue	Total Tolls	Total Trip Amount	Avg. Fare Amount	Avg. Fare Amount	Avg. Trip Amount
150.3K	2.9K	18.3K	12.1	12.1	1.8

Summery

Chart



Uber Dashboard Table

	Rate Code	Fare amount	T. Amount
1.	Standard rate	106.8K	132.5K
2.	JFK	11.2K	14.1K
3.	Negotiated fare	1.3K	1.6K
4.	Newark	1K	1.4K

1 - 5 / 5<>

