BI Report: NYC Green Taxi

Team Members: Katrina Ying, Jin(Tina) Yu, Zhen(Zander) Gong

Introduction

This project focuses on analyzing New York City green taxi trip records (January 2020 to October 2024) to track revenue, uncover temporal and spatial trends, and inform operational decision-making. By combining data extraction, transformation, and loading (ETL) processes with a star schema design, it creates a centralized data warehouse for efficient analytical queries. Key points can be described as follows:

1. **Data-Driven Growth:** Leverage revenue insights to drive strategic decision-making and expansion opportunities.
2. **Operational Excellence:** Optimize resource allocation and pricing strategies to maximize efficiency and profitability.
3. **Customer-Centric Innovation:** Enhance service offerings and payment options based on customer behavior and preferences.

Data

Record Data

1. **Data type**: Parquet format

2. **Data Source**: Green Taxi – TLC Trip Record Data
    a. Link: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

3. **Data attributes**

*Note: Additional columns are highlighted (not part of the original dataset). The original dataset (downloaded from raw parquet files has [3876894 rows x 19 columns].

Column Name	Data Type
vendorid	int64
lpep_pickup_datetime	datetime64[ns]
lpep_dropoff_datetime	datetime64[ns]
trip_duration	timedelta64[ns]

day_of_week	object
hour_of_day	int32
store_and_fwd_flag	object
ratecodeid	float64
pulocationid	Int64
dolocationid	Int64
passenger_count	float64
trip_distance	float64
fare_amount	float64
extra	float64
mta_tax	float64
tip_amount	float64
tolls_amount	float64
improvement_surcharge	float64
total_amount	float64
payment_type	float64
trip_type	float64
congestion_surcharge	float64
fare_per_mile	float64
location_pair	object

4. **Important features**
    a. **Pick-up and drop-off dates/times**:
        i. Pick-up: The date and time when the meter was engaged.
        ii. Drop-off: The date and time when the meter was disengaged.
    b. **Pick-up and drop off locations**:
        i. Pick-up: TLC Taxi Zone in which the taximeter was engaged
        ii. Drop-off: TLC Taxi Zone in which the taximeter was disengaged
    c. **Trip distances**: The elapsed trip distance in miles reported by the taximeter.
    d. **Fare_amount**: The time-and-distance fare calculated by the meter.
    e. **Itemized fares**: The time-and-distance fare calculated by the meter.
    f. **Rate types**: The final rate code in effect at the end of the trip.

i. 1= Standard rate
ii. 2=JFK
iii. 3=Newark
iv. 4=Nassau or Westchester
v. 5=Negotiated fare
vi. 6=Group ride

g. **Payment types**: A numeric code signifying how the passenger paid for the trip.
i. 1= Credit card
ii. 2= Cash
iii. 3= No charge
iv. 4= Dispute
v. 5= Unknown
vi. 6= Voided trip

h. **Driver-reported passenger counts**: The number of passengers in the vehicle.
i. **Extra Charges**: Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.

Mapping Data

1. **Data type**: CSV format and shp format

2. **Data Source**: Green Taxi – TLC Trip Record Data
   a. Link: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

3. **Data attributes**

Column Name	Data Type
LocationID	int64
Borough	varchar(255)
Zone	varchar(255)
service_zone	varchar(255)

5. **Important features**
   a. **LocationID**: TLC Taxi Zone
   b. Borough: Indicates the top-level administrative district of New York City (e.g. Manhattan, Brooklyn, Queens, Bronx, or Staten Island) in which the location (identified by the LocationID) is found.

ETL/ELT

Scripts

Extraction & Basic Cleaning

```
import duckdb
import os
Import pandas as pd

# Select all the .parquet files under the current directory
current_dir = os.getcwd()
parquet_files_path = os.path.join(current_dir, '*.parquet')

# Bridge connection with duckdb
con = duckdb.connect()

# Merge all the .parquet into one named merged_data.parquet
query = f"""
   COPY (SELECT * FROM '{parquet_files_path}')
   TO 'merged_data.parquet' (FORMAT PARQUET)
"""
con.execute(query)

df = pd.read_parquet(‘merged_data.parquet’)
df.drop(columns=['ehail_fee'], inplace=True)
df.dropna(inplace=True)
df.to_parquet(‘clean_data.parquet’)
```

Summarize Central Tendencies (Mean, Median, Mode)

```
import pandas as pd
df = pd.read_parquet('clean_data.parquet', engine='pyarrow')
numerical_df = df.select_dtypes(include=['number'])

mean_values = numerical_df.mean()
median_values = numerical_df.median()
mode_values = numerical_df.mode().iloc[0]
summary_df = pd.DataFrame({
   'Mean': mean_values,
   'Median': median_values,
   'Mode': mode_values
})
```

```
print(summary_df)
```

Identify And Drop Outliers

```
import pandas as pd
df = pd.read_parquet('clean_data.parquet', engine='pyarrow')

def find_outliers_iqr(column):
    Q1 = column.quantile(0.25)  # First quartile (25th percentile)
    Q3 = column.quantile(0.75)  # Third quartile (75th percentile)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return (column < lower_bound) | (column > upper_bound)

numeric_cols = df.select_dtypes(include=['number']).columns
numeric_cols = [col for col in numeric_cols if col != 'DOLocationID']

# Identify outliers in numerical columns
outlier_summary = []
total_outliers = 0
outlier_mask = pd.Series(False, index=df.index)
for col in numeric_cols:
    outliers = find_outliers_iqr(df[col])
    count = outliers.sum()
    outlier_summary.append({"Column Name": col, "Outlier Count": count})
    total_outliers += count
    outlier_mask |= outliers

outlier_summary_df = pd.DataFrame(outlier_summary)
print("\nSummary of Outliers in Numerical Columns:\n")
print(outlier_summary_df.to_string(index=False))
print(f"\nTotal Number of Outliers: {total_outliers}")

# Drop rows with any outliers
no_outliers_df = df[~outlier_mask].reset_index(drop=True)
print(f"\nShape of the new dataset without outliers: {no_outliers_df.shape}")

from google.colab import files
no_outliers_df.to_csv('/content/no_outliers_data.csv', index=False)
files.download('/content/no_outliers_data.csv')
```

Transformations

```
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
file_path = '/content/drive/My Drive/no_outliers_data.parquet'

df = pd.read_parquet(file_path, engine='pyarrow')
dropoff_index = df.columns.get_loc('lpep_dropoff_datetime')
df.insert(
    dropoff_index + 1,
    'trip_duration',
    df['lpep_dropoff_datetime'] - df['lpep_pickup_datetime']
)

import numpy as np
df['fare_per_mile'] = np.where(
    (df['trip_distance'].isna()) | (df['trip_distance'] == 0),
    0,
    df['fare_amount'] / df['trip_distance']
)

df.insert(
    loc=df.columns.get_loc('trip_duration') + 1,
    column='day_of_week',
    value=df['lpep_pickup_datetime'].dt.day_name()
)

hour_series = df['lpep_pickup_datetime'].dt.hour

day_of_week_index = df.columns.get_loc('day_of_week')
df.insert(
    loc=day_of_week_index + 1,
    column='hour_of_day',
    value=hour_series
)

df['PULocationID'] = df['PULocationID'].astype(str)
df['DOLocationID'] = df['DOLocationID'].astype(str)
df['location_pair'] = df['PULocationID'] + '-' + df['DOLocationID']

df['DOLocationID'] = df['DOLocationID'].astype('Int64')
df['PULocationID'] = df['PULocationID'].astype('Int64')
output_path = '/content/drive/My Drive/data.parquet'
df.to_parquet(output_path)
```

Loading

DuckDB

```
import pandas as pd
import duckdb

parquet_file = "data.parquet"
con = duckdb.connect('taxi_analytics.duckdb')
df = duckdb.query(f"SELECT * FROM parquet_scan('{parquet_file}')").to_df()
con.execute("CREATE TABLE IF NOT EXISTS temp_taxi_data AS SELECT * FROM
read_parquet('data.parquet')")
schema_info = duckdb_con.execute("PRAGMA table_info('temp_taxi_data')").fetchall()
conn.close()
```

MySQL

```
import mysql.connector
import duckdb
import pandas as pd

def convert_type(duck_type):
    duck_type = duck_type.upper()
    if "BIGINT" in duck_type:
        return "BIGINT"
    elif "INT" in duck_type:
        return "INT"
    elif "DOUBLE" in duck_type or "FLOAT" in duck_type:
        return "DOUBLE"
    elif "VARCHAR" in duck_type or "CHAR" in duck_type:
        return "VARCHAR(255)"
    elif "BOOLEAN" in duck_type:
        return "TINYINT(1)"
    elif "DATE" in duck_type:
        return "DATE"
    elif "TIMESTAMP" in duck_type:
        return "DATETIME"
    else:
        return "TEXT"

columns_def = []
for cid, col_name, col_type, notnull, dflt_value, pk in schema_info:
    mysql_type = convert_type(col_type)
    col_def = f"`{col_name}` {mysql_type}"
    if notnull:
```

```
        col_def += " NOT NULL"
    if pk:
        col_def += " PRIMARY KEY"
    columns_def.append(col_def)

create_table_sql = f"CREATE TABLE temp_taxi_data ({', '.join(columns_def)});"
print("MySQL CREATE TABLE statement:")
print(create_table_sql)

duckdb_data = duckdb_con.execute("SELECT * FROM temp_taxi_data").fetchall()
print(f"Fetched {len(duckdb_data)} rows from DuckDB's 'temp_taxi_data' table.")

mysql_conn = mysql.connector.connect(option_files="/etc/my.cnf")
cursor = mysql_conn.cursor()

new_db = "taxi_data"
cursor.execute(f"CREATE DATABASE IF NOT EXISTS {new_db};")
print(f"Database '{new_db}' created or already exists.")
cursor.execute(f"USE {new_db};")
print(f"Using database '{new_db}'.")

cursor.execute("DROP TABLE IF EXISTS temp_taxi_data;")
cursor.execute(create_table_sql)
mysql_conn.commit()
print("Created table 'temp_taxi_data' in MySQL.")

num_cols = len(schema_info)
placeholders = ", ".join(["%s"] * num_cols)
insert_sql = f"INSERT INTO temp_taxi_data VALUES ({placeholders})"

batch_size = 10000
for i in range(0, len(duckdb_data), batch_size):
    batch = duckdb_data[i : i + batch_size]
    cursor.executemany(insert_sql, batch)
    mysql_conn.commit()
    print(f"Inserted rows {i} to {i + len(batch) - 1}")

print("Done!")
cursor.close()
mysql_conn.close()
duckdb_con.close()
```

Create Data Warehousing

```
import mysql.connector
```

```
import pandas as pd

try:
    conn = mysql.connector.connect(option_files="/etc/my.cnf",
                                   database="taxi_data")
    cursor = conn.cursor()

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS dim_date (
        date_id          BIGINT       NOT NULL,
        full_date        DATETIME     NOT NULL,
        year             INT          NOT NULL,
        month            INT          NOT NULL,
        day              INT          NOT NULL,
        day_of_week      VARCHAR(50)  NOT NULL,
        hour_of_day      INT          NOT NULL,
        am_pm            VARCHAR(2)   NOT NULL,
        is_peak_hour     BOOLEAN      NOT NULL,
        minute           INT          NOT NULL,
        second           INT          NOT NULL,
        PRIMARY KEY(date_id)
    );
    """)

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS dim_payment_type (
        payment_type_id          INT          NOT NULL,
        payment_type_description  VARCHAR(50)   NOT NULL,
        PRIMARY KEY(payment_type_id)
    );
    """)

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS dim_ratecode (
        ratecode_id             INT          NOT NULL,
        rate_type_description   VARCHAR(50)   NOT NULL,
        PRIMARY KEY(ratecode_id)
    );
    """)

    cursor.execute("""
    CREATE TABLE IF NOT EXISTS fact_trips (
        trip_id INT AUTO_INCREMENT,
        lpep_pickup_date_id    BIGINT      NOT NULL,  -- FK to dim_date
```

```
        lpep_dropoff_date_id   BIGINT      NOT NULL,  -- FK to dim_date
        pulocationid           INT,
        dolocationid           INT,
        payment_type_id        INT         NOT NULL,  -- FK to dim_payment_type
        ratecode_id            INT,                   -- FK to dim_ratecode (optional)
        fare_amount            DECIMAL(10,2),
        total_amount           DECIMAL(10,2),
        trip_distance          DECIMAL(10,2),
        trip_duration          INT,
        tip_amount             DECIMAL(10,2),
        extra                  DECIMAL(10,2),
        fare_per_mile          FLOAT,
        PRIMARY KEY (trip_id),
        FOREIGN KEY (lpep_pickup_date_id)  REFERENCES dim_date (date_id),
        FOREIGN KEY (lpep_dropoff_date_id) REFERENCES dim_date (date_id),
        FOREIGN KEY (payment_type_id)      REFERENCES dim_payment_type
(payment_type_id),
        FOREIGN KEY (ratecode_id)          REFERENCES dim_ratecode (ratecode_id)
    );
    """)

    try:
        cursor.execute("SET GLOBAL net_read_timeout=300;")
        cursor.execute("SET GLOBAL wait_timeout=300;")
    except:
        pass

    insert_dim_date = """
    INSERT INTO dim_date (
        date_id,
        full_date,
        year,
        month,
        day,
        day_of_week,
        hour_of_day,
        am_pm,
        is_peak_hour,
        minute,
        second
    )
    SELECT DISTINCT
        CAST(DATE_FORMAT(t_datetime, '%Y%m%d%H%i%s') AS UNSIGNED) AS date_id,
        t_datetime AS full_date,
```

```
        YEAR(t_datetime) AS year,
        MONTH(t_datetime) AS month,
        DAY(t_datetime) AS day,
        DAYNAME(t_datetime) AS day_of_week,
        HOUR(t_datetime) AS hour_of_day,
        CASE WHEN HOUR(t_datetime) < 12 THEN 'AM' ELSE 'PM' END AS AM_PM,
        CASE
            WHEN (HOUR(t_datetime) BETWEEN 6 AND 8)
                OR (HOUR(t_datetime) BETWEEN 15 AND 17) THEN 1
            ELSE 0
        END AS is_peak_hour,
        MINUTE(t_datetime),
        SECOND(t_datetime)
    FROM (
        SELECT lpep_pickup_datetime AS t_datetime FROM temp_taxi_data
        UNION
        SELECT lpep_dropoff_datetime FROM temp_taxi_data
    ) AS all_datetimes
    WHERE CAST(DATE_FORMAT(t_datetime, '%Y%m%d%H%i%s') AS UNSIGNED) NOT IN (
        SELECT date_id FROM dim_date
    );
    """
    cursor.execute(insert_dim_date)
    conn.commit()

    cursor.execute("CREATE INDEX idx_full_date ON dim_date (full_date);")
    cursor.execute("CREATE INDEX idx_pickup_datetime ON temp_taxi_data
(lpep_pickup_datetime);")
    cursor.execute("CREATE INDEX idx_dropoff_datetime ON temp_taxi_data
(lpep_dropoff_datetime);")

    insert_dim_payment_type = """
    INSERT INTO dim_payment_type (payment_type_id, payment_type_description)
    SELECT
        ROW_NUMBER() OVER() AS payment_type_id,
        CASE
            WHEN payment_type = 1 THEN 'Credit Card'
            WHEN payment_type = 2 THEN 'Cash'
            WHEN payment_type = 3 THEN 'No Charge'
            WHEN payment_type = 4 THEN 'Dispute'
            WHEN payment_type = 5 THEN 'Unknown'
            WHEN payment_type = 6 THEN 'Voided trip'
            ELSE 'Other'
        END AS payment_type_description
```

```
    FROM (SELECT DISTINCT payment_type FROM temp_taxi_data) AS
unique_payment_types;
    """
    cursor.execute(insert_dim_payment_type)
    conn.commit()

    insert_dim_ratecode = """
    INSERT INTO dim_ratecode (ratecode_id, rate_type_description)
    SELECT DISTINCT
        RatecodeID,
        CASE
            WHEN RatecodeID = 1 THEN 'Standard'
            WHEN RatecodeID = 2 THEN 'JFK'
            WHEN RatecodeID = 3 THEN 'Newark'
            WHEN RatecodeID = 4 THEN 'Nassau or Westchester'
            WHEN RatecodeID = 5 THEN 'Negotiated fare'
            WHEN RatecodeID = 6 THEN 'Group ride'
            ELSE 'Other'
        END AS rate_type_description
    FROM temp_taxi_data
    WHERE RatecodeID IS NOT NULL;
    """
    cursor.execute(insert_dim_ratecode)
    conn.commit()

    insert_fact_trips = """
    INSERT INTO fact_trips (
        lpep_pickup_date_id, lpep_dropoff_date_id, pulocationid, dolocationid,
        payment_type_id, ratecode_id, fare_amount, total_amount,
        trip_distance, trip_duration, tip_amount, extra, fare_per_mile
    )
    SELECT
        d1.date_id AS lpep_pickup_date_id,
        d2.date_id AS lpep_dropoff_date_id,
        t.PULocationID,
        t.DOLocationID,
        t.payment_type AS payment_type_id,
        t.RatecodeID AS ratecode_id,
        t.fare_amount,
        t.total_amount,
        t.trip_distance,
        (t.trip_duration / 1000000000) / 60 AS trip_duration,
        t.tip_amount,
        t.extra,
```

```
        CASE WHEN t.trip_distance > 0 THEN t.fare_amount / t.trip_distance ELSE NULL END
AS fare_per_mile
    FROM temp_taxi_data t
    JOIN dim_date d1 ON d1.full_date = t.lpep_pickup_datetime
    JOIN dim_date d2 ON d2.full_date = t.lpep_dropoff_datetime;
    """
    cursor.execute(insert_fact_trips)
    conn.commit()

    print("ETL Process completed successfully!")

except mysql.connector.Error as err:
    print(f"Error: {err}")
finally:
    if conn.is_connected():
        cursor.close()
        conn.close()
```

Extract

The extraction phase begins with identifying and accessing the data source. The dataset consists of Parquet files spanning from 2020 to 2025, sourced from the New York Green Taxi Trip Record Data website. These files contain crucial trip-related details, including pick-up and drop-off dates and times, trip distances, itemized fares, rate types, payment types, and more. Once identified, the Parquet files are ingested into Python using the api provided by duckdb. Through this process, the data is efficiently loaded and parsed to extract key attributes, ensuring a structured foundation for the subsequent transformation phase.

Transform

Once the raw data is extracted, the transformation phase occurs to ensure data accuracy, consistency, and usability. The process begins with data cleaning and standardization, where null values and duplicate records are removed to maintain data integrity. Next, normalization is applied to categorical fields such as rate types and payment types, reducing redundancy and improving data efficiency.

Additionally, we identified and removed outliers from the dataset using the Interquartile Range (IQR) rule. We found that there were 4,396,902 total outliers across all numerical columns (excluding the DOLocationID column, as it had no outliers). We decided to use the IQR rule because it effectively identifies extreme values that significantly deviate from the majority of

the data. This rule is beneficial because it identifies outliers by measuring the spread of the middle 50% of the data. It uses the difference between the 75th percentile (Q3) and the 25th percentile (Q1) to define the "interquartile range." Outliers are considered any data points that fall outside the range of Q1−1.5×IQRQ1 - 1.5 \times IQRQ1−1.5×IQR (below the lower bound) or Q3+1.5×IQRQ3 + 1.5 \times IQRQ3+1.5×IQR (above the upper bound). This rule helps to remove extreme values that don't fit within the expected distribution of data, ensuring more reliable and consistent analysis. As a result, removing outliers ensures that the dataset reflects more accurate trends while minimizing the risk of skewing the data analysis process.

Schema mapping follows, where the dataset is structured into the fact and dimension tables as defined below. Foreign key relationships are validated to establish accurate relationships between tables, ensuring referential integrity. These transformation steps refine the dataset, making it optimized for analytical querying and seamless integration into the target data warehouse.

Load

The final phase of the ETL process involves loading the transformed data into the target database, ensuring it aligns with the star schema. First, a database is created, followed by the construction of tables that correspond to the schema's fact and dimension structures. Transformed records are then inserted into the appropriate dimension tables before the fact table is populated, referencing the associated dimension records through foreign keys. To maintain data integrity, foreign key constraints are verified to ensure they correctly match existing dimension entries.

For ongoing data integration, an incremental update strategy is implemented to efficiently manage new data arrivals. This process begins with identifying newly ingested records, followed by applying the necessary transformations to standardize and validate them. Finally, the processed data is seamlessly integrated into the existing schema, ensuring the database remains current without redundant reloading.

Star Schema

Idea

This star schema is centered around the fact table (fact_trips), which stores the main data related to green taxi trips, such as fare_amount, total_amount, trip_distance, trip_duration,

and tip_amount. This fact table also contains foreign keys linking to several dimension tables, providing context to each trip. The fact_trips table allows for efficient analysis of trip data across different time periods, locations, payment types, and rate codes, enabling greater insight into green taxi trips. Calculated fields, such as fare_per_mile, offer additional insights into trip pricing efficiency.

The dimension tables in this star schema include dim_date, dim_payment_type, and dim_ratecode, each providing descriptive attributes to help contextualize the facts. The dim_date table breaks down time-related information, such as the year, month, day, and hour of the trip, while the dim_payment_type table categorizes the payment methods used for each trip, and dim_ratecode offers information about the fare structure. This structure facilitates clear and flexible querying, allowing us to generate graphs and perform trend analysis on key metrics such as trip volume, payment type, tip amounts, and more.

The star schema was built using Python and the ETL (Extract, Transform, Load) process. Initially, the raw green taxi data (in the form of parquet files) was extracted using Python libraries like Pandas. The extracted data spans a total of 5 years (2020-2024). During the transformation phase, the data was cleaned, formatted, and structured to fit the star schema design, with necessary calculations such as dropping duplicate and null values and adding the new columns as described above. The data was then loaded into the corresponding tables, with the fact table (fact_trips) populated with foreign key references to the dimension tables (dim_date, dim_payment_type, dim_ratecode). This process ensures data integrity and allows for efficient querying of the star schema, enabling seamless analysis of the green taxi trip data.

Star Schema Description

Fact Table: `fact_trips`

Column Name	**Description**
`trip_id`	Primary Key
`lpep_pickup_date_id`	FK to `dim_date`
`lpep_dropoff_date_id`	FK to `dim_date`
`pulocationid`	
`dolocationid`	
`payment_type_id`	FK to `dim_payment_type`

`ratecode_id`	FK to `dim_ratecode`
`fare_amount`	Fare price ($)
`total_amount`	Total fare paid ($)
`trip_distance`	Distance in miles
`trip_duration	Duration in minutes
`tip_amount`	
`extra`	
`fare_per_mile`	Calculated field

Dimension Tables

dim_date

Column Name	**Column Type**
date_id	BIGINT
full_date	TIMESTAMP
year	INTEGER
month	INTEGER
day	INTEGER
day_of_week	VARCHAR
hour_of_day	INTEGER
am_pm	VARCHAR
is_peak_hour	BOOLEAN
minute	INTEGER
second	INTEGER

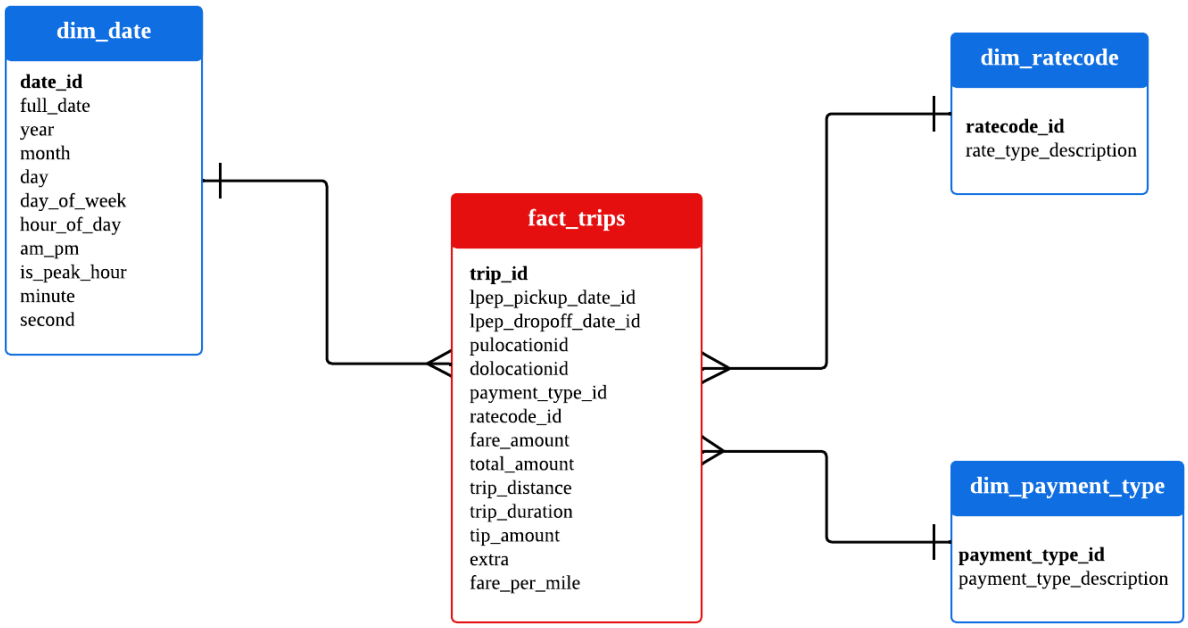
dim_payment_type

Column Name	**Column Type**
payment_type_id (PK)	INTEGER

payment_type_description	VARCHAR

dim_ratecode

Column Name	Column Type
ratecode_id (PK)	INTEGER
rate_type_description	VARCHAR



BI/OLAP Queries

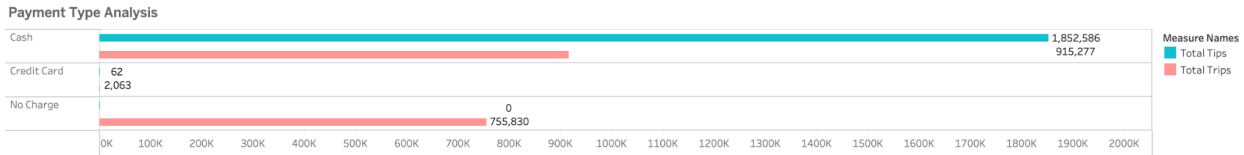
	Name	Description	OLAP Function	SQL Query
1	Payment Type Analysis	Tip amount distribution by payment type	Roll Up	SELECT d.payment_type_description, COUNT(t.trip_id) AS total_trips, AVG(t.tip_amount) AS avg_tip, SUM(t.tip_amount) AS total_tips FROM fact_trips t JOIN dim_payment_type d ON t.payment_type_id = d.payment_type_id GROUP BY d.payment_type_description

				ORDER BY total_tips DESC;
2	Trip Distance Bucketing	Analyze tip amount distribution and influencing factors like ride distance	Dice	SELECT CASE WHEN t.trip_distance < 2 THEN 'Short (0-2 miles)' WHEN t.trip_distance BETWEEN 2 AND 5 THEN 'Medium (2-5 miles)' WHEN t.trip_distance BETWEEN 5 AND 10 THEN 'Long (5-10 miles)' ELSE 'Very Long' END AS distance_category, d.payment_type_description, AVG(t.tip_amount) AS avg_tip, COUNT(t.trip_id) AS total_trips FROM fact_trips t JOIN dim_payment_type d ON t.payment_type_id = d.payment_type_id GROUP BY distance_category, d.payment_type_description ORDER BY avg_tip DESC;
3	Temporal & Location Trends	Identify passenger demand peaks by time and location (Displays the top three rush hour for different locations)	Roll Down	WITH RankedTrips AS (SELECT d.hour_of_day AS pickup_hour, f.pulocationid, COUNT(f.trip_id) AS total_trips, RANK() OVER (PARTITION BY f.pulocationid ORDER BY COUNT(f.trip_id) DESC) AS rank_num FROM fact_trips f JOIN dim_date d ON f.lpep_pickup_date_id = d.date_id GROUP BY d.hour_of_day, f.pulocationid HAVING COUNT(f.trip_id) > 50) SELECT pickup_hour, pulocationid, total_trips FROM RankedTrips WHERE rank_num <= 3 ORDER BY pulocationid, rank_num;
4				WITH PU_TripCounts AS (SELECT pulocationid, COUNT(trip_id) AS total_trips FROM fact_trips

	Revenue & Tip Percent-age by Pickup Location	Revenue & Tip Percentage for High-Trip Locations	Slice	GROUP BY pulocationid), AverageTrips AS (SELECT AVG(total_trips) AS avg_trips FROM PU_TripCounts) SELECT f.pulocationid, SUM(total_amount) AS total_revenue, AVG(f.tip_amount * 100.0 / f.total_amount) AS avg_tip_percentage FROM fact_trips f JOIN PU_TripCounts pu ON f.pulocationid = pu.pulocationid JOIN AverageTrips a ON pu.total_trips > a.avg_trips GROUP BY f.pulocationid ORDER BY avg_tip_percentage DESC;
5	Pivot Peak vs. Non-Peak Extra Charges	Pivot Extra Charges by Peak vs. Non-Peak Hours and Day of Week	Pivot	SELECT d.day_of_week, AVG(CASE WHEN d.is_peak_hour = 1 THEN f.extra END) AS avg_peak_extra, AVG(CASE WHEN d.is_peak_hour = 0 THEN f.extra END) AS avg_non_peak_extra FROM fact_trips f JOIN dim_date d ON f.lpep_pickup_date_id = d.date_id GROUP BY d.day_of_week;

Tableau Charts and Explanation

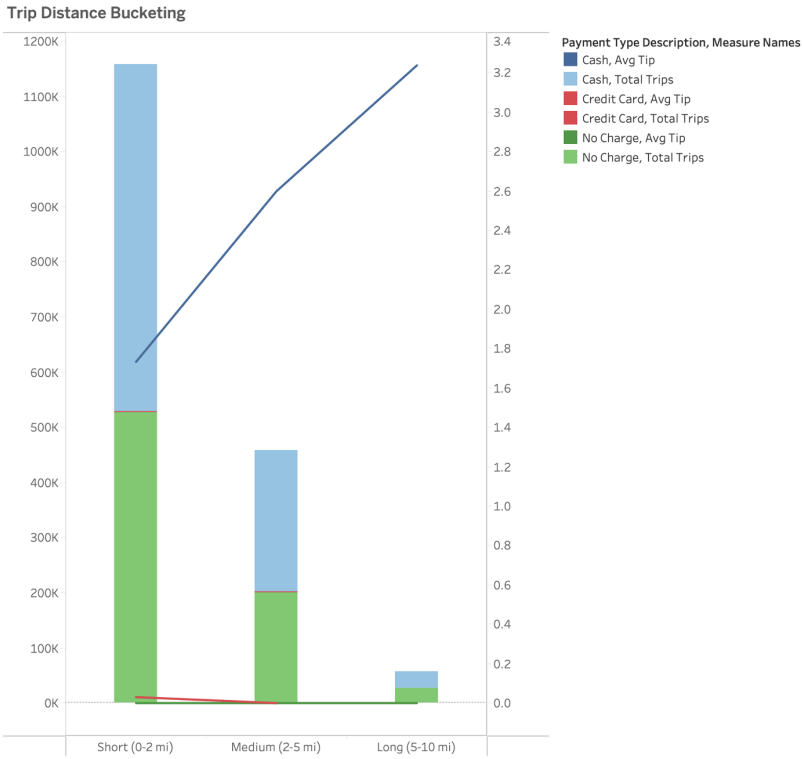
Query 1: Payment Type Analysis



Explanation: This bar graph shows the breakdown of tip amounts by payment type. Total trips is represented by the blue color and total tips amount is represented by the pink color. As can be seen in the graph, the most common tip payment type is cash. The second highest amount is No Charge, where there was no charge for the service. Customers tipped by credit card

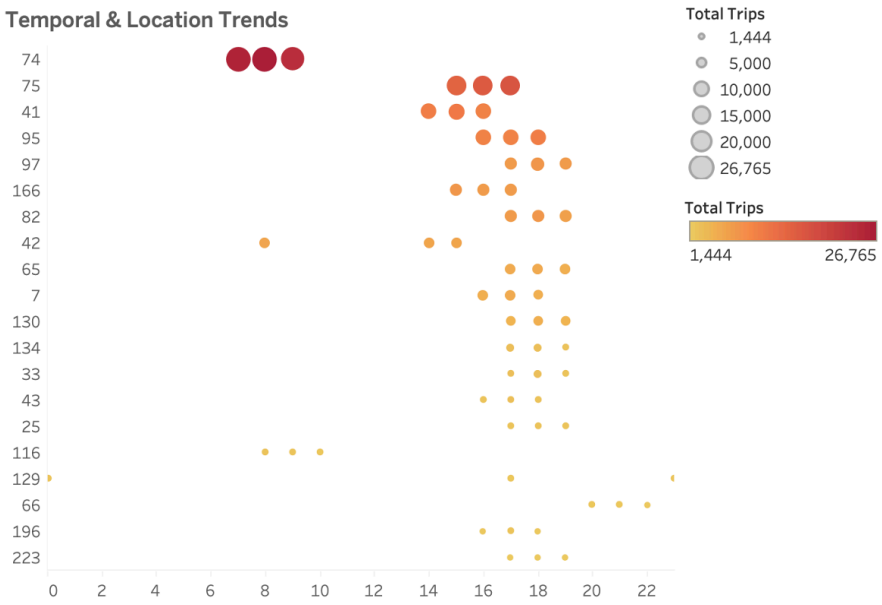
the least. Although digital payment methods are on the rise, this graph still indicates that customers prefer using cash to tip green taxi drivers in New York.

Query 2: Trip Distance Bucketing



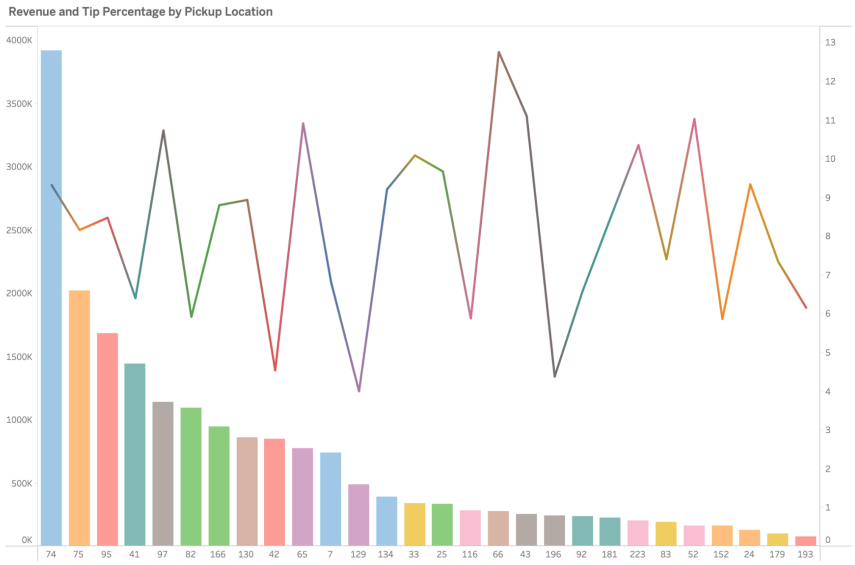
Explanation: This stacked bar graph with a dual axis analyzes tip amount distribution using influencing factors such as ride distance. Here, ride distance is split into two categories: Short (0-2 miles) and Medium (2-10 miles). The x-axis is distance categories while the y-axis has a dual axis of Total Trips (left axis) and Average Trips (right axis). The light green color represents “No Change” and the light blue color represents “Cash”. As can be seen in the graph, both No Change and Cash tip payment methods occur most often during short rides. Overall, however, cash is still the most common tip payment method no matter the ride distance.

Query 3: Temporal and Location Trends



Explanation: This bubble chart identifies passenger demand peaks by time and location. Time in hours (24 hours) is on the x-axis, and the pickup location ID is on the y-axis. The larger and darker the circle, the more trips occur at that location and hour. For example, the most trips in the dataset occurs at data points such as location 74 and the 6th hour. As the chart shows, the most trips occur from the 6th hour to the 20th hour of the day (6 am to 8 pm).

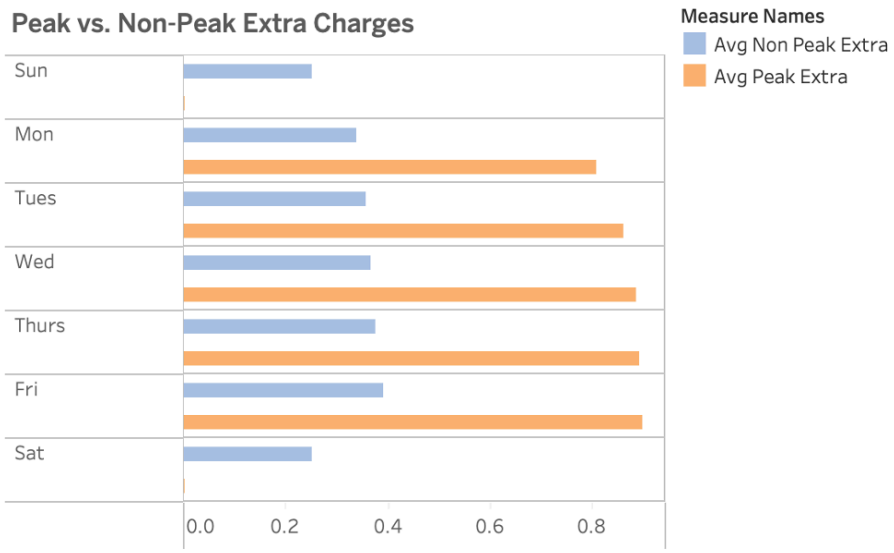
Query 4: Revenue and Tip Percentage by Pickup Location



Explanation: This dual-axis chart shows revenue and tip percentage for high-trip pickup locations. The total revenue amount is displayed on the left side of the y-axis and the average tip percentage is displayed on the right. The x-axis displays the pickup location ID. Total

revenue is represented by the bar chart and average tip percentage is represented by the line graph. In this case, the pickup location ID with the highest revenue is 74, and the highest average tip percentage occurs at location 66. The pickup location ID with the highest revenue AND average tip percentage is 97.

Query 5: Peak vs. Non-Peak Extra Charges

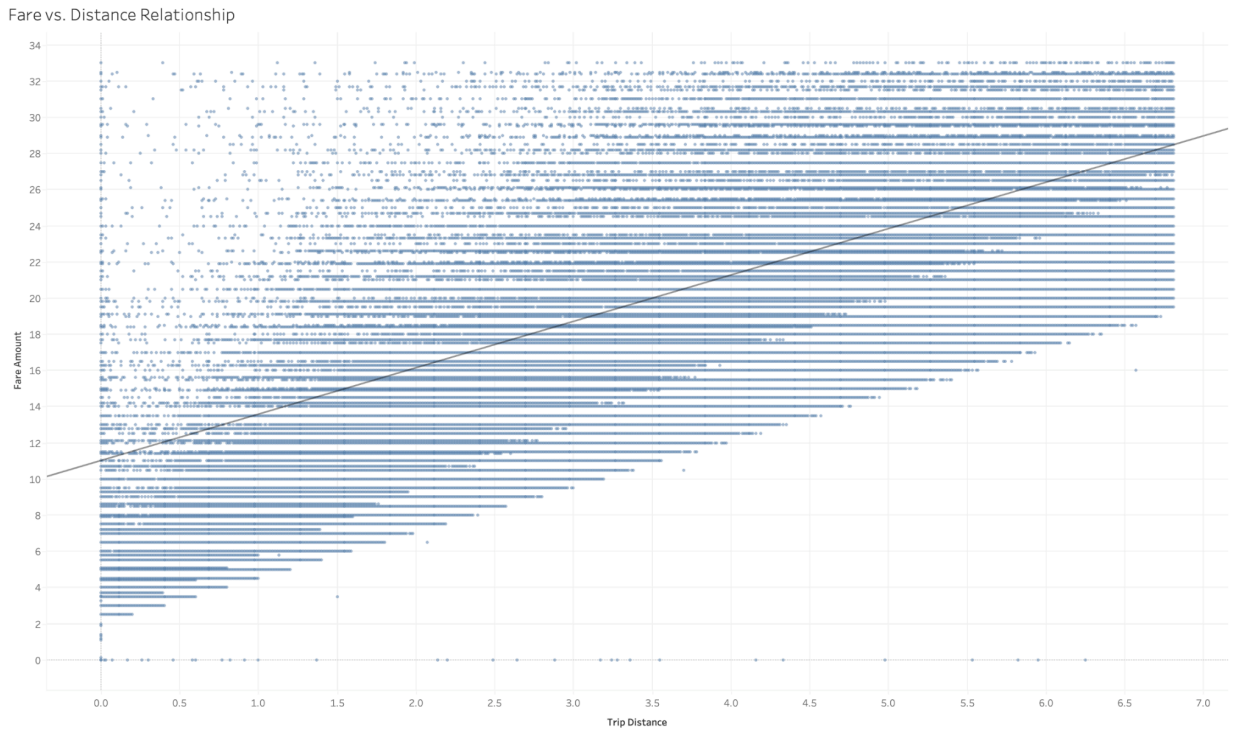


Explanation: This bar chart shows extra charges by peak vs. non-peak hours and day of week. Blue represents average non-peak extra charges and orange represents average peak extra charges. Days with the highest peak extra charges are: Wednesday, Thursday, and Friday. Days with the highest non-peak extra charges are also Wednesday, Thursday, and Friday. In comparison, Sunday and Saturday have the least amount of non-peak extra charges and Monday has the least amount of peak extra charges. Since extra charges in the raw data include the $0.50 and $1 rush hour and overnight charges, this indicates that most services on Wednesday, Thursday, and Friday occur at rush hour or overnight.

Tableau Dashboard for Green Taxi Analytics

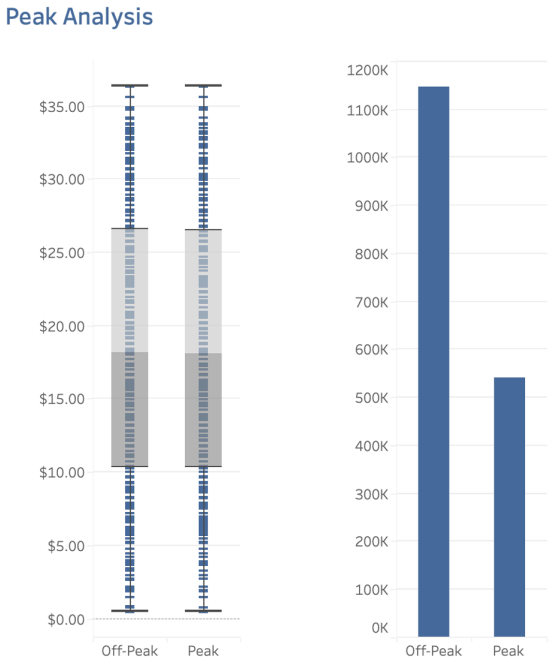
Explanation: This dashboard helps uncover patterns such as which day-of-week sees the most or least surcharges, which hours and zones have higher volumes or tips, which payment types dominate, and how trip distances correlate with revenue and tipping.

Additional Dashboard/Insights

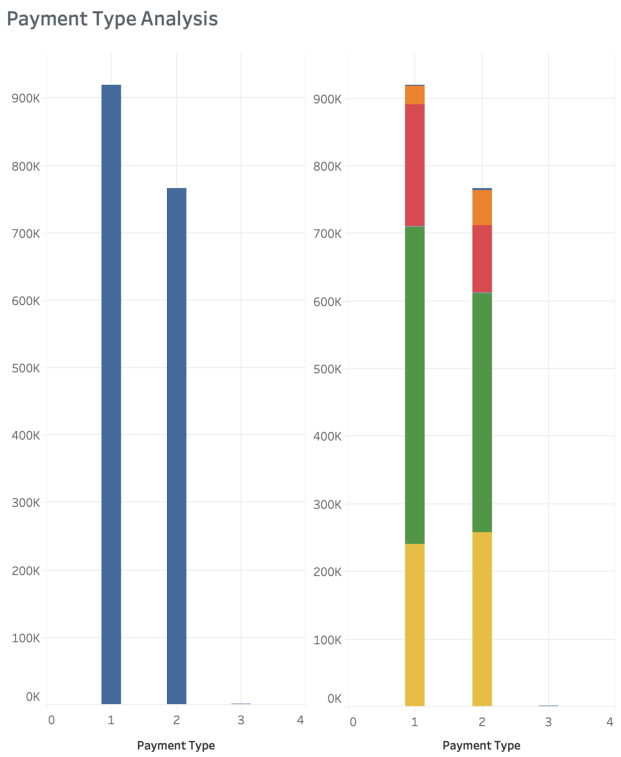


Explanation: This chart plots Fare Amount on the y-axis against Trip Distance on the x-axis, with each dot representing a single ride. The upward trend (as indicated by the

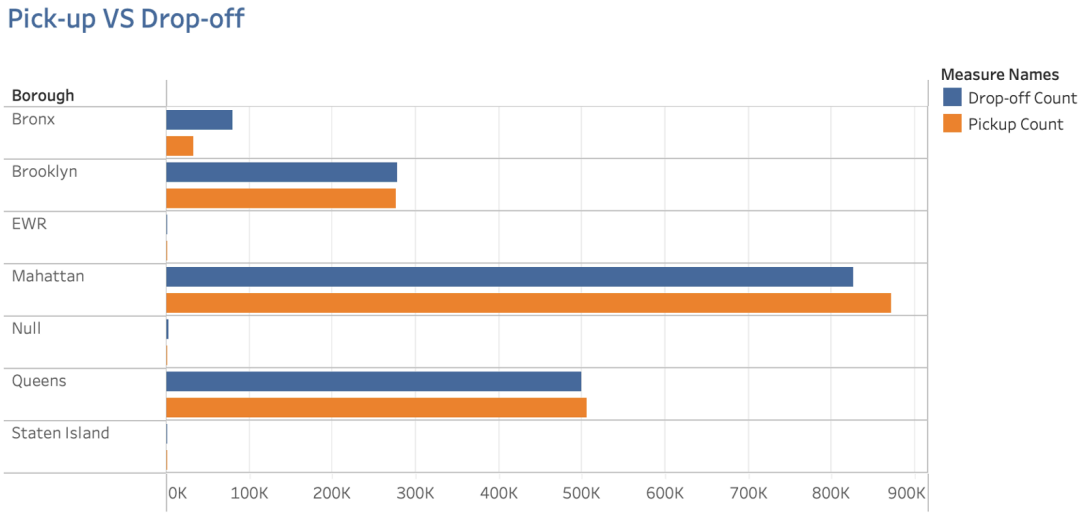
regression line) shows that longer trip distances generally correlate with higher fares—no surprise given metered rates typically rise with distance. However, it is obvious to see substantial scatter around the trend line, reflecting the fact that fare can be influenced by additional factors (e.g. time of day, surcharges, traffic delays, etc.) beyond simple distance alone. Green taxis follow a base fare plus metered fare structure, with a fixed amount charged at the start of each ride. After the base fare, the meter charges a per-mile fee when driving above a certain speed, and a per-minute fee when the speed is below that speed threshold (i.e., in traffic or when stopped). In addition, surcharges may apply during rush hour, late night hours, or trips to and from certain locations (e.g., airports).



Explanation: The box plot on the left shows the distribution of fare amounts for each box (one for off-peak and one for peak). It is obvious that the "peak" fare average (median and upper limit) appears to be slightly higher than the "off-peak" fare, reflecting the potential peak surcharge that increases the total fare. The bar chart on the right compares the total number of rides during off-peak and peak hours, with more rides during off-peak hours than during peak hours. The overall comparison of taxi activity during peak and off-peak hours in terms of fares and number of rides is presented horizontally.

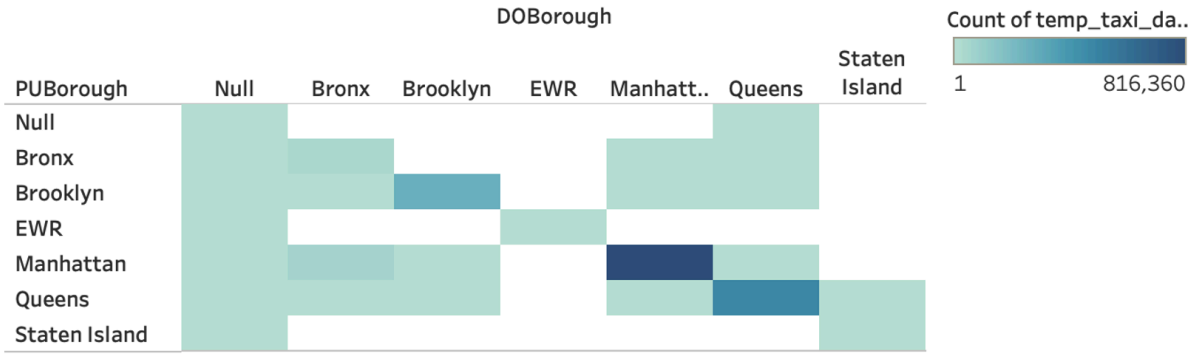


Explanation: Both views compare payment type, but with different levels of detail. The first single-color bar chart simply summarizes all rides by payment type. Credit card is the most common (~900,000), followed by cash (~700,000). The second stacked bar chart shows that credit card and cash account for the majority of rides, with Manhattan (green) accounting for the largest share, followed by Queens (yellow) and Brooklyn (red).

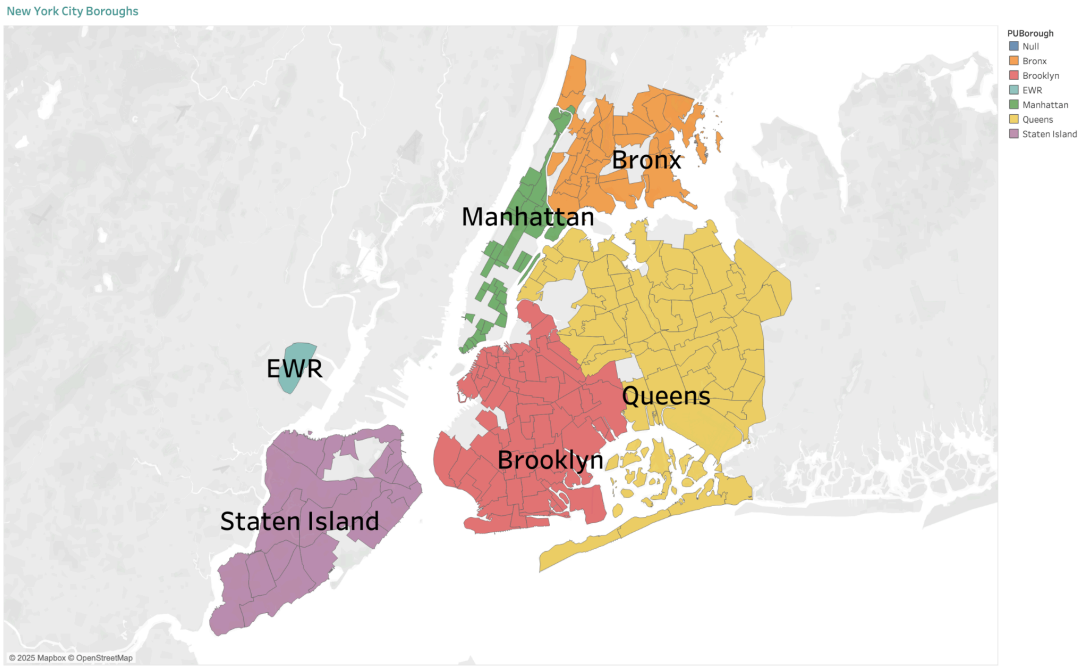


Explanation: This bar chart compares the number of taxi pickups (orange bars) and drop-offs (blue bars) across New York City boroughs. Manhattan has significantly higher pickups

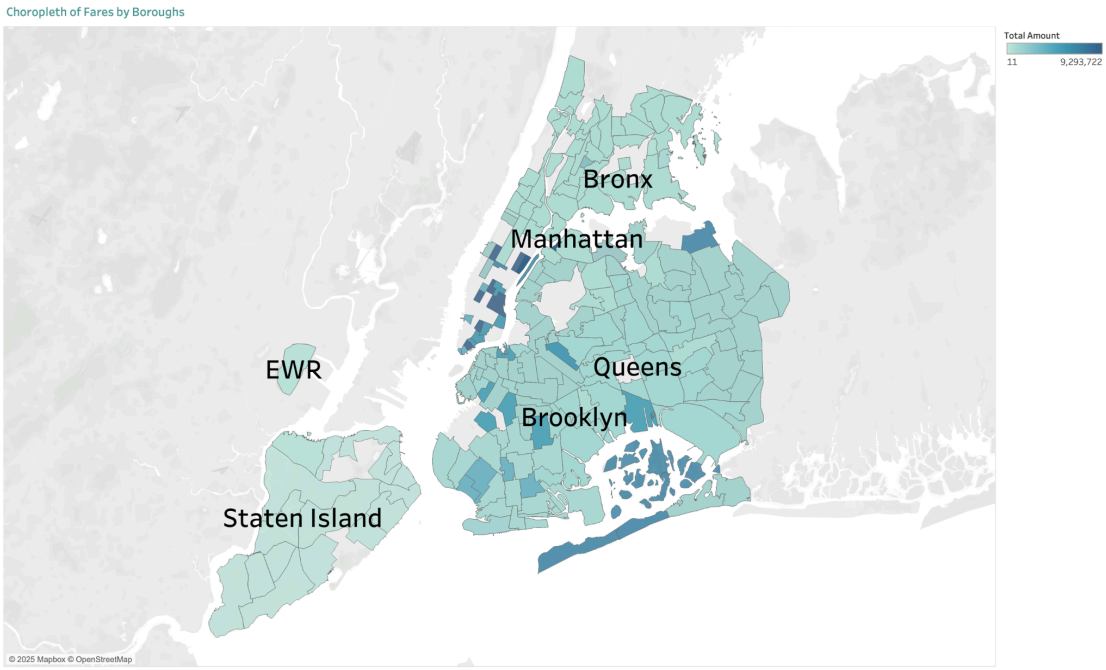
and drop-offs than the other boroughs, indicating that it is the busiest borough for taxi trips. Brooklyn and Queens also handle relatively more rides than the Bronx, Staten Island, and EWR.



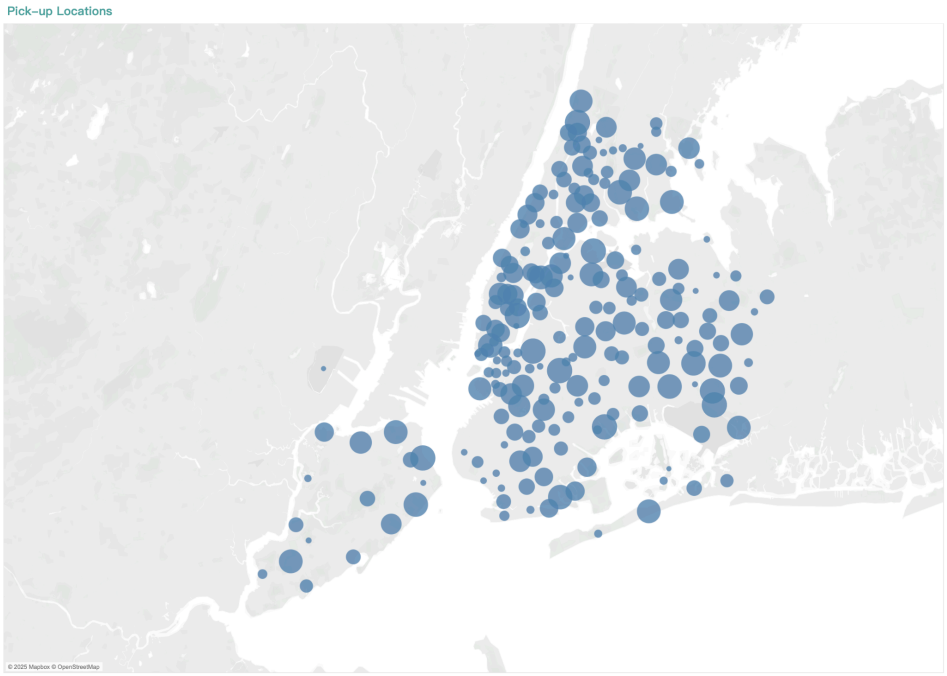
Explanation: This heatmap compares where a trip starts and where it ends. Darker cells indicate more rides, and lighter cells indicate fewer rides. Most rides are picked up and dropped off in the same borough, especially Manhattan → Manhattan, which appears as one of the darkest squares (highest count). Other notable flows can be seen, such as Brooklyn → Brooklyn or Manhattan → Queens, etc.



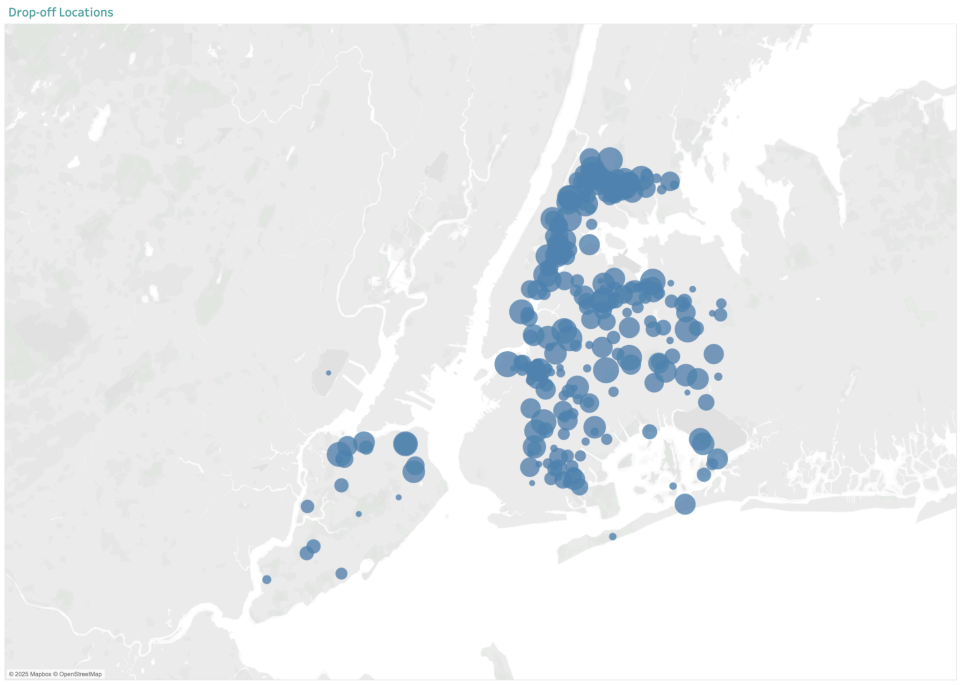
Explanation: This map provides geographic context for the other borough-based charts.



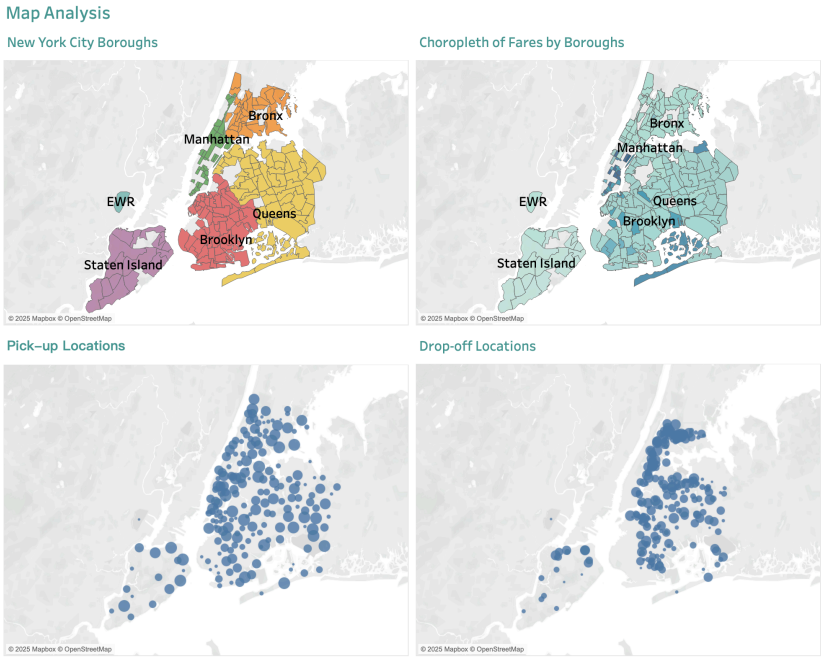
Explanation: This contour map shows New York City taxi zones, with colors determined by the total fares collected in each zone. Manhattan zones are generally darker in color, reflecting higher taxi usage (both in terms of rides and fare amounts). Brooklyn and Queens have medium colors, indicating lower (but still high) fare totals. Staten Island and some outlying boroughs have lighter colors, indicating lower total fares.



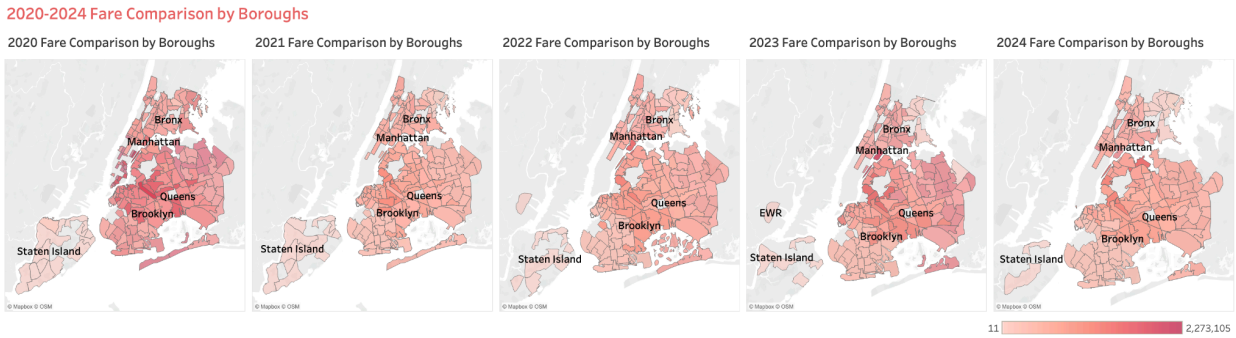
Explanation: This bubble chart shows taxi pickup locations in New York City, including the outer boroughs. The location of each circle reflects the pickup location, and the size of the circle indicates how many total rides originated there. Business trips, commuters, and tourists near the World Trade Center and Wall Street often result in many pickups there.



Explanation: This bubble chart highlights where taxi trips end. A hotspot is near the Midtown/Times Square area. This is a major tourist and business center known for Broadway theaters and large attractions. Lower Manhattan/Financial District is also noteworthy, close to major transportation hubs such as Wall Street, the World Trade Center, and Fulton Street Station. The Upper East/Upper West Sides have high-density residential areas and popular museums and attractions (e.g., Central Park, Metropolitan Museum of Art). Outside of Manhattan, other notable areas with heavy traffic in general include: JFK Airport and LaGuardia Airport (also in Queens) in Queens. Downtown Brooklyn, close to major MetroTech and shopping centers.



Explanation: Together, the four maps provide a comprehensive view of the City's boroughs, overall fare levels, and pickup and drop-off locations, which can help quickly identify major transportation corridors and the busiest neighborhoods for boarding and alighting.



Explanation: These five side-by-side contour maps show total annual taxi fares from 2020 to 2024 for each borough of New York City, allowing for a comparative view over time. Across all years, the Manhattan area generally appears darkest, indicating that taxi usage and fares have been high compared to the other boroughs. Brooklyn and Queens also contain several moderately to heavily shaded areas, particularly in neighborhoods near downtown Brooklyn, Williamsburg, or the airport or Manhattan in Queens. Clear shifts in trips are seen in Manhattan, Staten Island, and Queens as a result of the decline in the pandemic in 2020/2021 (lighter shading overall) and the recovery in 2022/2023 (darker shading again). EWR (Newark Airport) data shows an increase in trips starting in 2023, perhaps due to an increase in air travelers, reflecting more travel or a returning tourism industry.

Business Recommendations

Based on these OLAP queries, we provide the following recommendations:

1. **Promote Payment Methods with Higher Tips**

Since credit cards yield the highest tips, consider implementing strategies to encourage their use. For instance, offer discounts, cashback incentives, or loyalty points for customers using credit cards, which could drive higher tip amounts and better customer retention.

Additionally, adopting flexible **digital tipping options**—such as adjusting preset tip suggestions and round-up features based on high-tip locations—can further increase tip amounts. Partnering with popular payment apps or ride-booking platforms to streamline transactions and provide seamless digital receipts could also enhance the passenger experience, leading to more frequent credit card usage and higher overall tips.

2. **Optimize Trip Length-Based Marketing**

In this case, the most common distance categories for our dataset are short (0-2 miles) and medium (2-10) miles. Based on this, the city can provide **targeted promotions for short trips** to boost frequency and loyalty, and **create incentives for medium trips** to maximize revenue. Tailoring discounts based on short vs. medium distances could drive customer behavior in a profitable direction.

For example, the city could introduce **a loyalty program** where passengers taking frequent short trips receive a discount after at least 10 rides, encouraging repeat usage. Additionally, for medium trips, the city could offer **time-based discounts** during off-peak hours to increase ridership when passenger demand is lower. This strategy helps balance taxi utilization throughout the day while maximizing revenue.

3. **Enhance Driver Deployment Based on Peak Hours and Location**

By leveraging location and time-based data, driver deployment can be optimized to match high-demand periods and key locations, ensuring efficient service coverage. **Historical trip data** can help identify peak hours (e.g., morning and evening rush hours) and high-performing areas (such as airports, business districts, nightlife hubs, and major transit stations), allowing for better workforce allocation and reduced passenger wait times.

To further enhance deployment, a **real-time demand monitoring system** can be implemented to provide drivers with **live updates on hotspot areas**, helping them position themselves strategically. By aligning driver supply with demand patterns, the city can maximize trip frequency, improve customer satisfaction, and boost overall taxi utilization.

4. **Improve Customer Experience in High-Tip and High-Volume Locations**

Locations with higher average tip percentages should be prioritized for customer experience enhancements. **Invest in training and rewarding top-performing drivers** in these areas to maintain high standards of service, encouraging more customers to tip generously and boosting overall customer satisfaction.

Likewise, the city can identify high-traffic locations with low average tip percentages. Implement driver incentive programs, training, or premium services (e.g., comfort features, cleanliness) to encourage higher tips. Additionally, focus on **service consistency** and customer satisfaction to improve overall ratings and tips.

Additionally, insights from the trip frequency data can be used to **target high-volume locations with marketing campaigns** that promote the benefits of using green taxi services. Highlight time-saving features, comfort, or competitive advantages, encouraging more customers to take trips from these top locations.

5. **Promote Off-Peak Travel with Incentives**

For non-peak hours or days, offer promotions or incentives to encourage more trips during slower periods. This will help balance demand and optimize the utilization of drivers throughout the day. For example, discounts on extra charges or loyalty rewards for off-peak travelers could help drive overall business.

One possible method would be to offer passengers a **small fare discount** or **bonus reward points for future rides** (for example, on weekdays between late morning and mid-afternoon or late at night). Additionally, partnerships with local businesses—such as offering discounts on rides to restaurants, theaters, or shopping districts during off-peak times—could further incentivize travel. By **making off-peak travel more appealing**, the city can increase overall ridership, stabilize revenue, and improve driver efficiency.

6. **Optimize Extra Charges for Fair Pricing**

Implement a strategic pricing structure that optimizes revenue while maintaining fairness for riders. One approach is to introduce **tiered surcharges** based on demand patterns, ensuring that peak-hour charges reflect increased taxi availability and service efficiency. Additionally, transparent communication about these charges can improve customer acceptance, making riders more willing to pay extra during high-demand periods.

To balance demand, the city could also explore **off-peak incentives** by temporarily reducing surcharges during slower hours, such as midday or late nights, to encourage more trips. Special **event-based surcharges** could be introduced for high-traffic periods, like concerts or major city events, ensuring that pricing aligns with real-world demand fluctuations. By refining the way extra charges are applied, the city can enhance taxi utilization, improve customer satisfaction, and create a more balanced and efficient pricing model.

7. **Implement Data-Driven Dynamic Pricing**

Utilize insights from customer behavior, trip distances, and peak times to implement dynamic pricing models. Offer pricing based on customer segments, time of day, and location, ensuring that the pricing is competitive yet maximizes revenue during high-demand periods.

For example, **location-based pricing** could be implemented, where fares in high-traffic areas (such as airports, business districts, and event venues) are adjusted to reflect real-time demand. A **customer-segmented pricing strategy** could also be explored, where discounts or loyalty rewards are offered to frequent riders, senior citizens, or students. By personalizing fare structures based on rider behavior, the city can encourage repeat usage and build long-term customer loyalty. By leveraging machine learning and predictive analytics, pricing models can become more responsive, ensuring that fares are optimized without discouraging riders.

Advanced Analytics

Predictive Analytics: Revenue Forecasting (XGBoost)

Script

```
import pandas as pd
import numpy as np
import xgboost as xgb
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.cluster import KMeans

df = pd.read_parquet("data.parquet")
df['day_of_week'] = pd.Categorical(df['day_of_week']).codes
```

```
df['payment_type'] = pd.Categorical(df['payment_type']).codes

features = ['trip_distance', 'hour_of_day', 'day_of_week', 'payment_type']
target = 'total_amount'

X = df[features]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
xgb_model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5)
xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)
xgb_model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5)
xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred)))
```

Result

```
Mean Absolute Error: 2.359051650850599
Mean Squared Error: 9.627680076308796
Root Mean Squared Error: 3.1028503148409845
```

Business Insights

- With a mean absolute error (MAE) of about 2.36, a mean squared error (MSE) of about 9.63, and an RMSE of about 3.10, the model performs well at predicting total fares. In practice, this means that on average, the predicted fares deviate from the actual fares by a little more than $2. While the model is not perfect, these errors indicate that the model can provide reasonably reliable estimates for planning and budgeting. Businesses can use these predictions to forecast revenue, schedule drivers more effectively, or plan marketing and promotional campaigns during slower seasons.

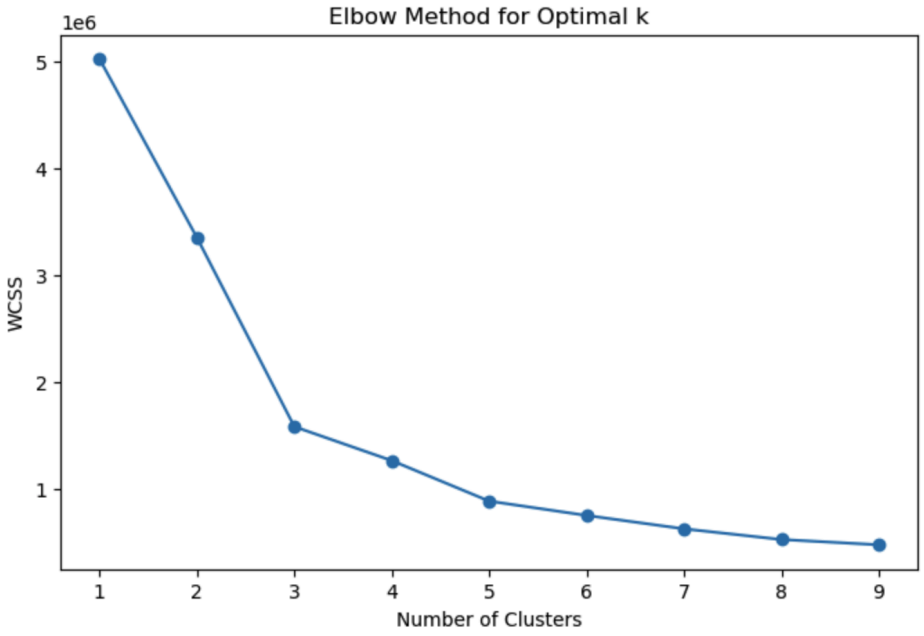
- If the predictions are still not accurate enough for critical decisions, further feature engineering or hyperparameter tuning may improve performance. For example, incorporating weather data or special events could improve predictions.

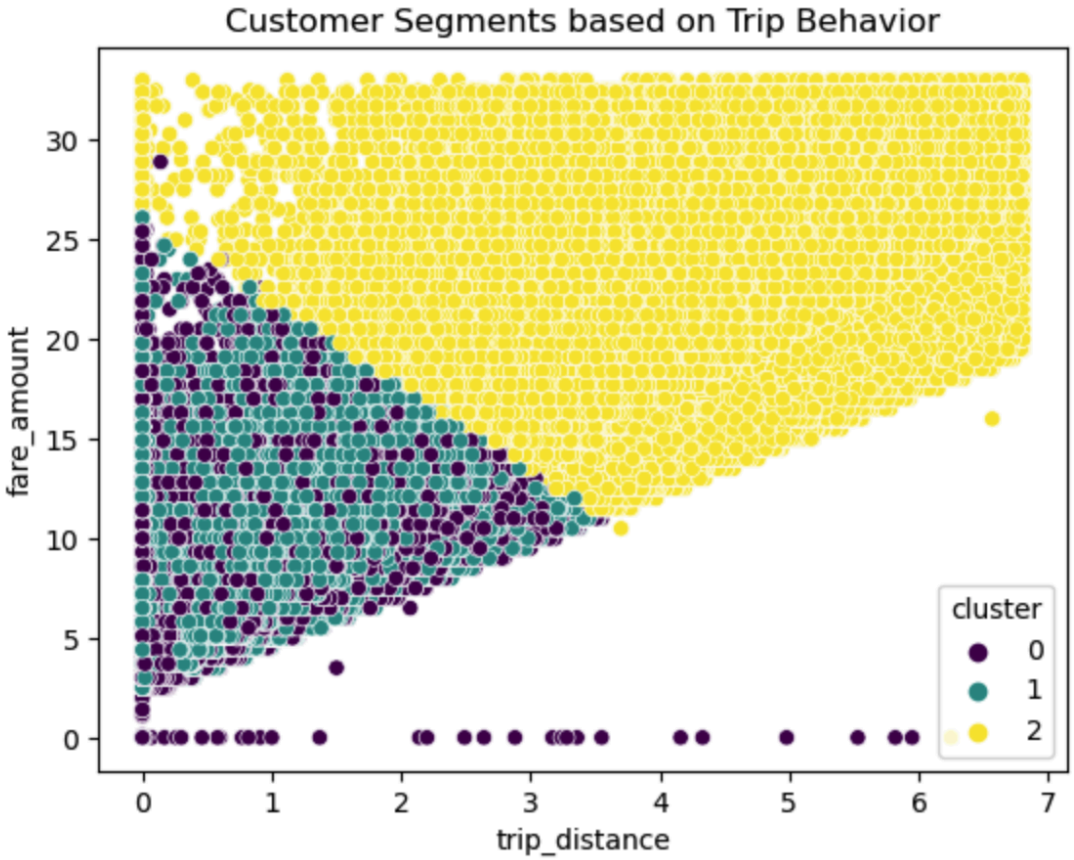
Customer Segmentation (K-Means Clustering)

Script

```
clustering_features = ['trip_distance', 'fare_amount', 'payment_type']

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[clustering_features])
wcss = []
for i in range(1, 10):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(8,5))
plt.plot(range(1, 10), wcss, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal k')
plt.show()
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
df['cluster'] = kmeans.fit_predict(df_scaled)

sns.scatterplot(x=df['trip_distance'], y=df['fare_amount'], hue=df['cluster'], palette='viridis')
plt.title('Customer Segments based on Trip Behavior')
plt.show()
```

Result





Business Insights

- The elbow method shows that three clusters are best suited to segment passengers based on features such as trip distance, fare amount, and payment type. The purple in the figure is primarily short, low-fare trips. The teal includes medium distances and fares. The yellow represents longer trips and higher fares. Identifying these segments can help tailor marketing campaigns and service offerings. For example, strategists can promote short-distance travel discounts or loyalty programs for passengers who frequently take short-distance flights, while offering premium services or packages to customers who travel longer distances.

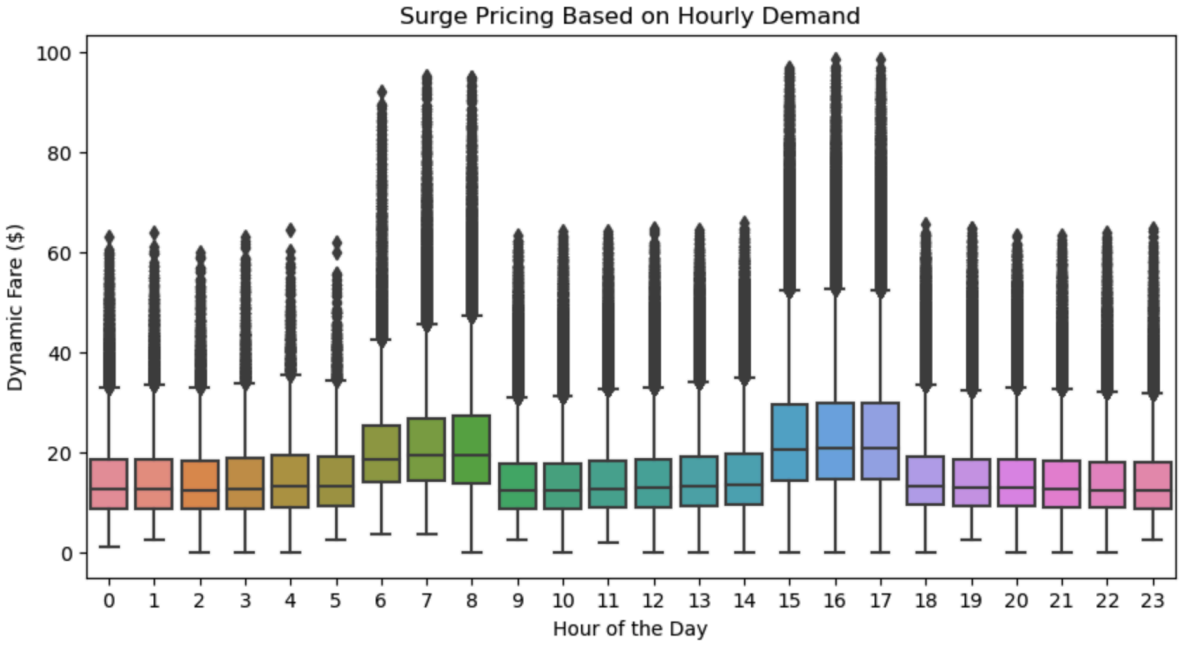
Dynamic Pricing Optimization

Script

```
def dynamic_pricing(hour, demand_factor, base_fare):
    peak_hours = [6, 7, 8, 15, 16, 17]
    if hour in peak_hours:
        surge_multiplier = 1.5
    else:
        surge_multiplier = 1.0

    adjusted_fare = base_fare * surge_multiplier * demand_factor
```

```
    return round(adjusted_fare, 2)
df['dynamic_fare'] = df.apply(lambda row: dynamic_pricing(row['hour_of_day'],
np.random.uniform(1.0, 2.0), row['fare_amount']), axis=1)

plt.figure(figsize=(10, 5))
sns.boxplot(x=df['hour_of_day'], y=df['dynamic_fare'])
plt.xlabel("Hour of the Day")
plt.ylabel("Dynamic Fare ($)")
plt.title("Surge Pricing Based on Hourly Demand")
plt.show()
```

Result



Business Insights

- The box plot shows that during traditional peak hours (e.g., 7-9 a.m., 4-6 p.m.), fares increase significantly, reflecting the surge multiplier. This aligns pricing with higher demand and can maximize revenue when driver supply is tight. Outside of peak hours, the surge multiplier is lower, indicating that the system is designed to encourage rides during slower hours. The random demand factor introduced in the code simulates the variability of demand. It shows how prices can fluctuate within the same hour. More sophisticated approaches (e.g., using real-time demand data) can further optimize fare adjustments.
- Dynamic pricing can help balance supply and demand, increase revenue during peak hours, and avoid missed opportunities when demand is high. However, large price increases can also lead to customer dissatisfaction, so monitoring customer feedback is critical.

Summary

Challenges

This project involved several challenges that required a combination of technical expertise and problem-solving skills. One of the main challenges was understanding the ETL (Extract, Transform, Load) process, which meant ensuring the data was accurately extracted from various sources, transformed into the desired format, and loaded into the database efficiently. This was particularly important to maintain data integrity and consistency throughout the workflow.

Another significant challenge was using Python and MySQL to build the database and star schema. This task required careful database design and table optimization, ensuring the schema was both scalable and efficient for complex queries. Finally, creating action-oriented visualizations with Tableau proved to be challenging, as the visualizations needed to be intuitive, interactive, and tailored to highlight key insights. This not only enabled better data analysis but also facilitated the clear presentation of key findings. Overcoming these challenges helped refine both the technical and analytical skills needed for data-driven projects.

Key Learnings

Throughout this project, several key learnings emerged that were crucial for the successful analysis and presentation of data. First, we learned how to extract key insights from the data by applying various analytical techniques to uncover meaningful patterns, trends, and correlations. This process involved identifying the most relevant metrics and utilizing aggregation, filtering, and statistical methods to highlight the most important findings.

Another important learning was how to use visualizations to effectively communicate these insights to the audience. By designing clear, intuitive, and interactive Tableau dashboards, we were able to translate complex data into easily digestible visual formats that enabled stakeholders to grasp the key takeaways quickly.

Finally, applying business intelligence concepts to our project helped us understand how to structure the data and present actionable insights that align with business goals. This experience reinforced the value of using BI principles to guide decision-making, ensuring that the data not only provided valuable insights but also supported informed, strategic decisions for improving New York's green taxi services.

Future Improvements

There are several improvements that could further enhance the project and provide deeper insights. One key improvement is the creation of additional dimension tables for analysis. By adding more dimension tables, such as those for special events, customer demographics, or geographic zones, we can better categorize and segment the data, enabling more detailed analysis and insights.

Another improvement is the incorporation of more data sources for analysis, such as weather data, special events, and demographic information. These additional datasets would provide valuable context, allowing us to analyze how external factors influence ride demand and pricing patterns.

Finally, a cross-city comparison of taxi data with similar datasets from other cities would allow us to identify regional trends, assess operational performance across different environments, and refine our strategies based on broader data. Incorporating these improvements would significantly expand the scope and depth of the analysis, providing more robust and actionable insights.