Predictive Analytics for NYC Taxi Operations: Demand Forecasting and Traffic Congestion Insights

By: Ruhee Shrestha

MIS 502: Final Project Report

Instructor: Professor Jim Ryan

Date: 04/30/2025

Data Set Description

1. NYC Yellow Taxi Trip Data: Public dataset from NYC Taxi & Limousine Commission containing timestamped trip-level info, geospatial pickup/drop-off zones, fare breakdowns, and payment types.

Key Data Elements:

1. **Trip Metadata**:
    a. Pickup and drop-off timestamps
    b. Vendor ID
    c. Passenger count
    d. Trip distance
    e. Payment method
    f. Fare Amount
2. **Geospatial Information**:
    a. Pickup and drop-off coordinates (latitude and longitude)
    b. Based on Location ID map to NYC boroughs and taxi zones using the **Taxi Zone Lookup** dataset to join with Traffic Data.

2. NYC DOT Traffic Volume Counts Data: Collected via Automated Traffic Recorders across NYC roads and bridges, detailing hourly vehicle volume with GPS metadata. Used to contextualize taxi demand with real-world traffic conditions.

Key Data Elements:

- Traffic volume counts
- Date and time of traffic recording
- Geospatial location of the traffic recorder (latitude and longitude)

Scope:

- Historical data spans from 2006 to 2024
- For this project, only data from **2020 to 2024** is used to align with recent taxi activity trends

Key Objectives

The objective of this project is to integrate the **NYC Yellow Taxi Trip Data** with **NYC Traffic Volume Counts Data** to analyze the impact of traffic patterns on taxi operations, predict taxi demand, and identify high-congestion and high-risk zones across New York City. By combining mobility and traffic datasets, this project aims to generate actionable insights to improve transportation efficiency, demand forecasting, and urban mobility planning.

Here are the following areas of focuses from the dataset:

- **Taxi Demand Prediction**: Forecast taxi pickup demand across NYC neighborhoods by leveraging traffic volume trends, historical trip data, and temporal patterns. Time-series modeling and feature-based machine learning techniques will be employed to capture demand fluctuations and rush hour dynamics.
- **Traffic Congestion Prediction**: Analyze taxi pickup and drop-off activity as a proxy for urban congestion. Develop predictive models to identify traffic hotspots, delays, and heavy congestion periods based on taxi activity levels.
- **Visualization and Geospatial Analysis**: Generate spatial heatmaps of pickup and drop-off events to highlight high-demand zones across boroughs and neighborhoods.
- **Traffic Zone Analysis**: Study the relationship between traffic volume (measured by ATR sensors) and taxi trip frequency to uncover correlations between traffic congestion and taxi service patterns.
- **High-Risk Zone Identification**: Map and monitor zones with consistently high congestion or demand variability to support targeted urban mobility improvements.

Potential Outcomes of Analysis:

Analysis Area	Potential Business Outcomes
Taxi Demand Forecasting	- Anticipate surge demand hours and deploy more taxis. - Optimize driver shift schedules based on peak zones.
Traffic Congestion Prediction	- Proactively reroute drivers before predicted congestion. - Identify "surge precursors" from taxi pickups.
Zone Typology (Clustering)	- Tailor pricing, dispatch policies for airports vs downtown zones. - Identify zones where dynamic pricing is crucial.
Association Rules Discovery	- Predict weekend + evening surges reliably. - Use rules to trigger automatic surge alerts.
Correlation Insights	- Recognize when heavy trip demand correlates with road congestion. - Identify inefficiencies (e.g., high traffic, low trips).
Geospatial Visualization	- Deploy heatmaps to city officials showing demand hotspots.

	- Identify boroughs needing new taxi stands, better flow.
High-Risk Zone Identification	- Design interventions for zones with consistently unstable traffic/demand (infrastructure upgrades, policy tweaks).

Data Mining Objectives

For this analysis, only data from **2020 to 2024** will be used

1. **Predict pickup demand from traffic data:** This data set is used to link **traffic volume to taxi pickup/drop-off times** and locations to find **Taxi Demand**,
2. **Predict traffic congestion from taxi activity;** Taxi data can be used as a predictor to explain delays, and heavy traffic near pickup zone, and identify rush hour pattens and high traffic zones.
3. Explore bidirectional influence between taxi activity and road congestion

Both models will be analyzed separately, and findings will be compared to understand the bidirectional relationship between traffic flow and taxi demand.

Data Quality, Data Integrity, and Data Ethics Issues

Missing Data

- **Taxi Data:**
  - Missing pickup and drop-off coordinates, needs to look up Taxi Zones to retrieve coordinates.
- **Traffic Data:**
  - Missing data points for specific roadways or time frames.

Timestamp Mismatches

- Differences between trip timestamps, traffic recording times, and collision reporting times can lead to misalignment when merging datasets.

Geospatial Mismatches

- Taxi pickup and drop-off coordinates may not align perfectly with traffic zones due to GPS inaccuracies.

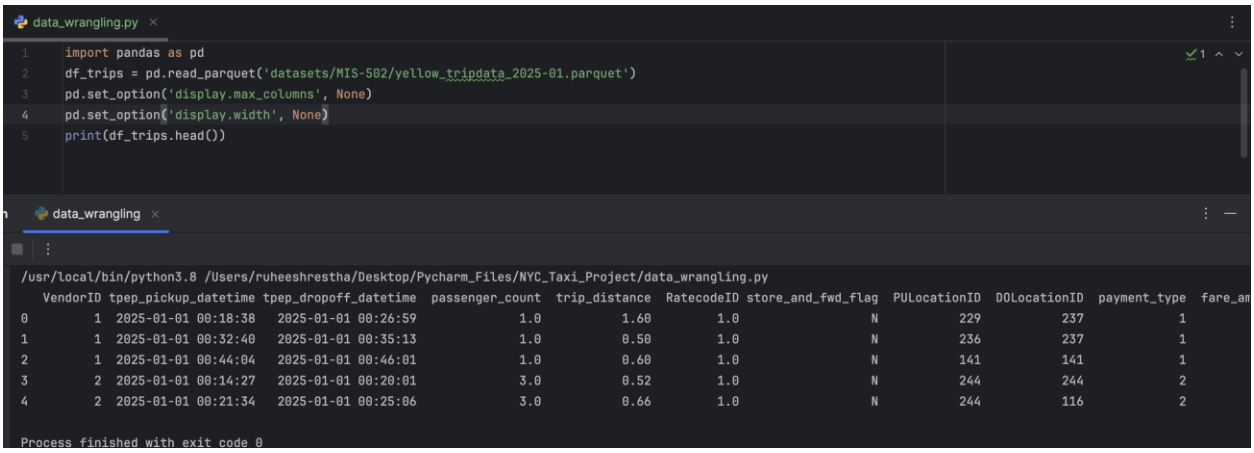
- Taxi zone IDs may not match geospatial boundaries precisely due to mapping differences.

Bias and Representation

- **Taxi Data:**
  - Fare data and passenger count could reflect socioeconomic biases (e.g., higher fares in wealthier areas).
  - Underrepresentation of trips from underserved neighborhoods may skew demand forecasts.

Data Preparation and Data Wrangling Steps

Loading NYC Taxi Trips Dataset



Data Profiling — NYC Taxi Trip Data

Retrieving Summary Statistics

```
print(df_trips.info())
```

```
data_wrangling
RangeIndex: 3475226 entries, 0 to 3475225
Data columns (total 19 columns):
 #   Column                Dtype
---  ------                -----
 0   VendorID              int32
 1   tpep_pickup_datetime  datetime64[us]
 2   tpep_dropoff_datetime datetime64[us]
 3   passenger_count       float64
 4   trip_distance         float64
 5   RatecodeID            float64
 6   store_and_fwd_flag    object
 7   PULocationID          int32
 8   DOLocationID          int32
 9   payment_type          int64
 10  fare_amount           float64
 11  extra                 float64
 12  mta_tax               float64
 13  tip_amount            float64
 14  tolls_amount          float64
 15  improvement_surcharge float64
 16  total_amount          float64
 17  congestion_surcharge  float64
 18  Airport_fee           float64
dtypes: datetime64[us](2), float64(12), int32(3), int64(1), object(1)
```

```
print(df_trips.describe())
print(df_trips.shape)
print(df_trips.columns)
```

```
data_wrangling
/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_Files/NYC_Taxi_Project/data_wrangling.py
          VendorID        tpep_pickup_datetime       tpep_dropoff_datetime  passenger_count  trip_distance    RatecodeID  PULocationID  DOLocationID  payment_type   fa
count  3.475226e+06                     3475226                     3475226     2.935077e+06   3.475226e+06  2.935077e+06  3.475226e+06  3.475226e+06  3.475226e+06  3.4
mean   1.785428e+00  2025-01-17 11:02:55.910964  2025-01-17 11:17:56.997901     1.297859e+00   5.855126e+00  2.482535e+00  1.651916e+02  1.641252e+02  1.036623e+00  1.7
min    1.000000e+00         2024-12-31 20:47:55         2024-12-18 07:52:40     0.000000e+00   0.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  0.000000e+00 -9.0
25%    2.000000e+00         2025-01-10 07:59:01  2025-01-10 08:15:29.500000     1.000000e+00   9.800000e-01  1.000000e+00  1.320000e+02  1.130000e+02  1.000000e+00  8.6
50%    2.000000e+00         2025-01-17 15:41:33         2025-01-17 15:59:34     1.000000e+00   1.670000e+00  1.000000e+00  1.620000e+02  1.620000e+02  1.000000e+00  1.2
75%    2.000000e+00         2025-01-24 19:34:06         2025-01-24 19:48:31     1.000000e+00   3.100000e+00  1.000000e+00  2.340000e+02  2.340000e+02  1.000000e+00  1.9
max    7.000000e+00         2025-02-01 00:00:44         2025-02-01 23:44:11     9.000000e+00   2.764236e+05  9.900000e+01  2.650000e+02  2.650000e+02  5.000000e+00  8.6
std    4.263282e-01                         NaN                         NaN     7.507503e-01   5.646016e+02  1.163277e+01  6.452948e+01  6.940169e+01  7.013334e-01  4.6
(3475226, 19)
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag',
       'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra',
       'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',
       'total_amount', 'congestion_surcharge', 'Airport_fee'],
      dtype='object')
```

Total Number of Rows = 3,475,226, Total Number of Columns = 19

Performing Data Cleaning

1. Dropping Duplicates

There are no duplicates in the NYC Taxi Dataset.

```
10
11  print(df_trips.duplicated().sum())
12
13  df_trips = df_trips.drop_duplicates()
14
15
```

```
Run   data_wrangling ×

/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_Files/NYC_Taxi_Project/data_wrangling.py
0
```

2. Dropping Missing Values

```
13
14  missing_summary = df_trips.isnull().sum()
15  print("Missing values per column:\n", missing_summary)
16
```

```
data_wrangling ×

/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_Files/NYC_Taxi_Project/data_wrangling.py
0
Missing values per column:
 VendorID                     0
tpep_pickup_datetime          0
tpep_dropoff_datetime         0
passenger_count          540149
trip_distance                 0
RatecodeID               540149
store_and_fwd_flag       540149
PULocationID                  0
DOLocationID                  0
payment_type                  0
fare_amount                   0
extra                         0
mta_tax                       0
tip_amount                    0
tolls_amount                  0
improvement_surcharge         0
total_amount                  0
congestion_surcharge     540149
Airport_fee              540149
dtype: int64
```

There are missing values for congestion_surcharge, airport_fee, and store_and_fwd_flag, which are non-essential columns, so I will dropping be those columns.

```
17    df_trips = df_trips.drop(columns=['congestion_surcharge', 'Airport_fee', 'store_and_fwd_flag'])
18    print(df_trips.columns)
19
20    # import matplotlib.pyplot as plt
21    #
22    # # List of key numerical features to check for outliers
```

data_wrangling

```
/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_Files/NYC_Taxi_Project/data_wrangling.py
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'RatecodeID', 'PULocationID',
       'DOLocationID', 'payment_type', 'fare_amount', 'extra', 'mta_tax',
       'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount'],
      dtype='object')
```

I will drop rows where passenger_count is 0, since a taxi ride should have at least one passenger.

```
19
20    df_trips = df_trips[df_trips['passenger_count'] != 0]
21    print(df_trips['passenger_count'].value_counts())
22
```

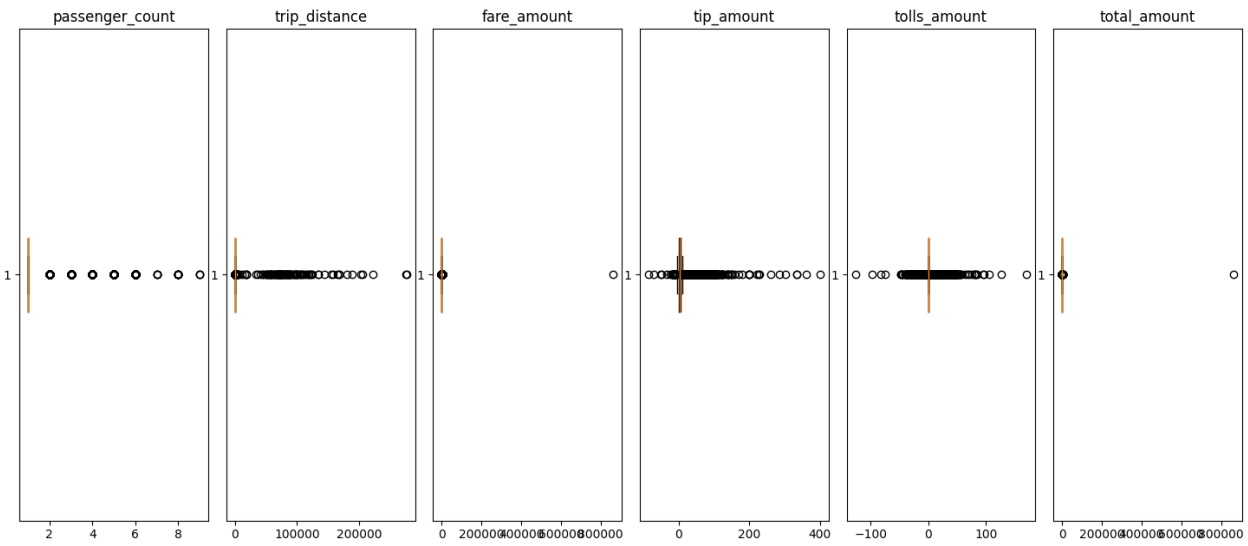
data_wrangling

```
/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_Files/NYC_Ta
passenger_count
1.0    2322434
2.0     407761
3.0      91409
4.0      59009
5.0      17786
6.0      12004
8.0         11
7.0          4
9.0          3
Name: count, dtype: int64
```

3. Detecting Outliers

Here, I am using a box plot to visualize the outliers in numerical values such as passenger count, trip distance, and fare amount, so that I can drop those values, or filter it out.

```python
import matplotlib.pyplot as plt
#
# List of key numerical features to check for outliers
numeric_features = [
    'passenger_count', 'trip_distance', 'fare_amount', 'tip_amount',
    'tolls_amount', 'total_amount', 'congestion_surcharge']

# Plot boxplots
fig, axes = plt.subplots(nrows=1, ncols=6, figsize=(24, 4))  # Adjust size as needed

for ax, feature in zip(axes, numeric_features):
    ax.boxplot(df_trips[feature].dropna(), vert=False)
    ax.set_title(feature)
    ax.set_xlabel('')

# Adjust layout
plt.tight_layout()
plt.show()
```



I did not remove any outliers here, because for Taxi demand analysis, long-distance trips, like to airports or suburbs, may show key pickup/drop-off zone, and there are fare surge zones, where high total_amount or fare_amount values may indicate areas with demand surges (e.g., during events or weather). Large passenger_count (5–6) may indicate carpooling or event transport demand. Long trips (20+ miles) can show travel corridors or underserved neighborhoods.

4. Performing Data Enrichment

I enriched the NYC Taxi data with designated Boroughs/Zones from the LocationID based on the Taxi Zone Lookup Data.

```python
taxi_zone_lookup  = pd.read_csv('datasets/MIS-502/taxi_zone_lookup.csv')
taxi_zone_lookup.head()

df_trips = df_trips.merge(
    taxi_zone_lookup[['LocationID', 'Borough', 'Zone']].rename(columns={
        'LocationID': 'PULocationID',
        'Borough': 'PUBorough',
        'Zone': 'PUZone'
    }),
    on='PULocationID',
    how='left'
)

df_trips = df_trips.merge(
    taxi_zone_lookup[['LocationID', 'Borough', 'Zone']].rename(columns={
        'LocationID': 'DOLocationID',
        'Borough': 'DOBorough',
        'Zone': 'DOZone'
    }),
    on='DOLocationID',
    how='left'
)
```

```
   LocationID        Borough                     Zone service_zone
0           1            EWR           Newark Airport          EWR
1           2         Queens              Jamaica Bay    Boro Zone
2           3          Bronx  Allerton/Pelham Gardens    Boro Zone
3           4      Manhattan            Alphabet City  Yellow Zone
4           5  Staten Island            Arden Heights    Boro Zone
```

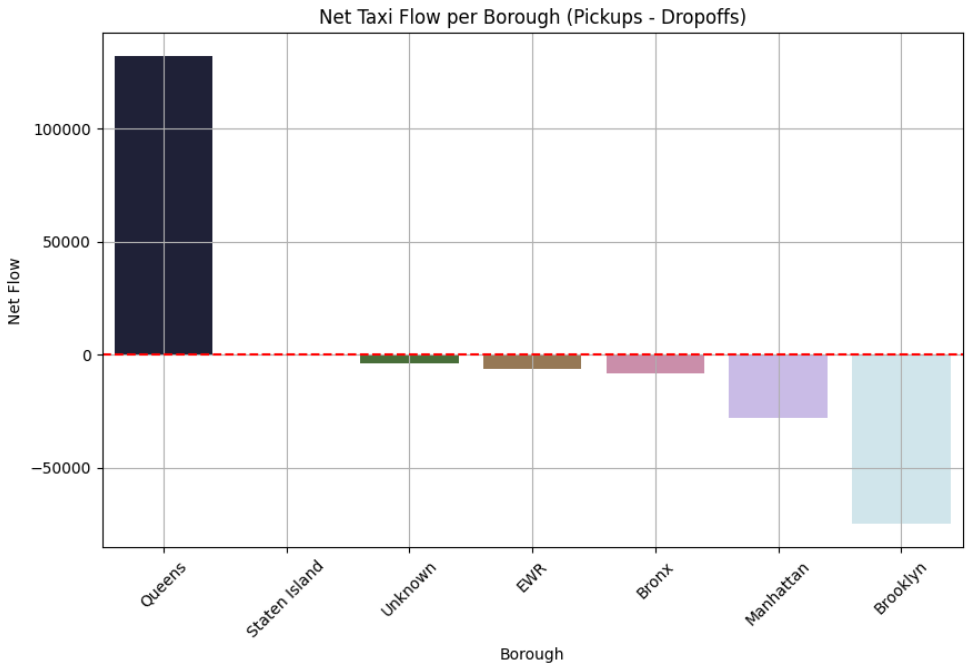
I enriched the NYC Taxi dataset by joining it with the Taxi Zone Lookup data, mapping each ride's pickup and drop-off LocationID to their corresponding boroughs and zones. The resulting columns were renamed as PUBorough, PUZone, DOBorough, and DOZone to represent the pickup and drop-off locations more descriptively.

Output:

```
/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_Files/NYC_Taxi_Project/data_wrangling.py
         PUBorough                      PUZone  DOBorough                  DOZone
0        Manhattan  Sutton Place/Turtle Bay North  Manhattan      Upper East Side South
1        Manhattan          Upper East Side North  Manhattan      Upper East Side South
2        Manhattan                Lenox Hill West  Manhattan            Lenox Hill West
3        Manhattan       Washington Heights South  Manhattan   Washington Heights South
4        Manhattan       Washington Heights South  Manhattan           Hamilton Heights
...            ...                            ...        ...                        ...
3450565  Manhattan                  East Village  Manhattan      Upper East Side South
3450566  Manhattan                Midtown Center  Manhattan           Hamilton Heights
3450567  Manhattan            Little Italy/NoLiTa  Manhattan  West Chelsea/Hudson Yards
3450568  Manhattan            Lincoln Square East  Manhattan                  Gramercy
3450569  Manhattan          Upper East Side South  Manhattan      Upper West Side North

[3450570 rows x 4 columns]
```

Performing EDA – Visualizing Taxi Drop Off Trends

```
pickup_counts = df_trips['PUBorough'].value_counts().sort_values(ascending=False)
print(pickup_counts)
dropoff_counts = df_trips['DOBorough'].value_counts(ascending=False)
print(dropoff_counts)
net_flow = (pickup_counts - dropoff_counts).sort_values(ascending=False)
print(net_flow)
avg_distance_pickup = df_trips.groupby('PUBorough')['trip_distance'].mean().sort_values(ascending=False)
print(avg_distance_pickup)

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
sns.barplot(x=net_flow.index, y=net_flow.values, palette='cubehelix')
plt.title('Net Taxi Flow per Borough (Pickups - Dropoffs)')
plt.ylabel('Net Flow')
plt.xlabel('Borough')
plt.xticks(rotation=45)
plt.axhline( y: 0, color='red', linestyle='--')
plt.grid(True)
```



This bar chart **"Net Taxi Flow per Borough (Pickups - Dropoffs)"** shows the **difference between the number of taxi pickups and drop-offs** across NYC boroughs. The **net flow** is calculated as:

Net Flow = Number of Pickups – Number of Drop-offs

Key Insights:

- **Queens is a major origin point** for taxi rides (likely due to airports).
- **Brooklyn and Manhattan act more as destinations**, especially for inbound traffic.
- These insights are valuable for **balancing taxi supply** and planning **return trip incentives or fleet placement**.

Loading Supplement Dataset - Traffic Volumes Dataset

```python
traffic_df = pd.read_csv('datasets/MIS-502/Automated_Traffic_Volume_Counts_20250421.csv')
print(traffic_df.head())

traffic_df = traffic_df[(traffic_df['Yr'] >= 2020) & (traffic_df['Yr'] <= 2024)]

traffic_df.reset_index(drop=True, inplace=True)

traffic_df['volume_count_datetime'] = pd.to_datetime(dict(
    year=traffic_df['Yr'],
    month=traffic_df['M'],
    day=traffic_df['D'],
    hour=traffic_df['HH'],
    minute=traffic_df['MM']
))
```

Run data_wrangling

```
/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_Files/NYC_Taxi_Project/data_wrangling.py
   RequestID      Boro    Yr   M   D  HH  MM  Vol  SegmentID                                              WktGeom          street                fromSt      toSt Direction
0      32970    Queens  2021   4  30   2   0    0     149701  POINT (997407.0998491726 208628.92612708386)  PULASKI BRIDGE  Newtown Creek Shoreline  Dead end        NB
1      32970    Queens  2021   4  30   2  15    1     149701  POINT (997407.0998491726 208628.92612708386)  PULASKI BRIDGE  Newtown Creek Shoreline  Dead end        NB
2      11342  Brooklyn  2012  12  18   8  15   33      20063                    POINT (985746.5 167127.4)           61 ST                   15 AV     16 AV        WB
3      32970    Queens  2021   4  30   2  30    0     149701  POINT (997407.0998491726 208628.92612708386)  PULASKI BRIDGE  Newtown Creek Shoreline  Dead end        NB
4      32970    Queens  2021   4  30   2  45    0     149701  POINT (997407.0998491726 208628.92612708386)  PULASKI BRIDGE  Newtown Creek Shoreline  Dead end        NB
    Boro  D  HH   vol_mean    vol_std  vol_max  vol_median
0  Bronx  1   0  20.232143  12.346499       54        18.0
1  Bronx  1   1  13.803571   7.770386       33        13.5
2  Bronx  1   2  10.178571   5.874775       24        10.0
3  Bronx  1   3   8.553571   5.232485       23         8.0
4  Bronx  1   4  10.892857   7.379772       33         8.5

Process finished with exit code 0
```

Joining NYC Taxi Datasets to Traffic Volumes Dataset by Boroughs

data_wrangling.py

```python
df_trips['pickup_datetime_rounded'] = pd.to_datetime(df_trips['tpep_pickup_datetime']).dt.floor('H')
df_trips['pickup_date'] = pd.to_datetime(df_trips['pickup_datetime_rounded']).dt.date
df_trips['pickup_hour'] = pd.to_datetime(df_trips['pickup_datetime_rounded']).dt.hour
df_trips['pickup_weekday'] = pd.to_datetime(df_trips['pickup_datetime_rounded']).dt.dayofweek
df_trips['pickup_month'] = pd.to_datetime(df_trips['pickup_datetime_rounded']).dt.month

df_trips_merged = df_trips.merge(
    traffic_features[['Boro','D', 'HH', 'vol_mean', 'vol_max', 'vol_median']],
    left_on=['PUBorough', 'pickup_weekday', 'pickup_hour'],
    right_on=['Boro', 'D', 'HH'],
    how='left'
)

print(df_trips_merged.head())
```

Run data_wrangling

```
urcharge  total_amount  PUBorough                      PUZone  DOBorough                  DOZone pickup_datetime_rounded pickup_date  pickup_hour  pickup_weekday
     1.0         18.00  Manhattan  Sutton Place/Turtle Bay North  Manhattan     Upper East Side South             2025-01-01  2025-01-01            0               2
     1.0         12.12  Manhattan          Upper East Side North  Manhattan     Upper East Side South             2025-01-01  2025-01-01            0               2
     1.0         12.10  Manhattan                  Lenox Hill West  Manhattan           Lenox Hill West             2025-01-01  2025-01-01            0               2
     1.0          9.70  Manhattan       Washington Heights South  Manhattan  Washington Heights South             2025-01-01  2025-01-01            0               2
     1.0          8.30  Manhattan       Washington Heights South  Manhattan          Hamilton Heights             2025-01-01  2025-01-01            0               2
```

Here I joined the NYC Taxi Data with the **aggregated traffic volume data** (e.g., vol_mean, vol_max) to each trip based on:

- PUBorough to Boro — Spatial dimension (pickup borough).
- pickup_weekday to D — Day of week (temporal dimension).

- pickup_hour to HH — Hour of day (temporal dimension).

I did a left join to ensure all taxi trips are preserved even if some don't have matching traffic volume data.

Feature Engineering for Predicting Taxi Demand

```python
# Aggregate
demand_df = (
    df_trips_merged.groupby(['PUBorough', 'PUZone', 'pickup_datetime_rounded'])
    .agg(
        pickup_count=('VendorID', 'count'),
        avg_trip_distance=('trip_distance', 'mean'),
        avg_passenger_count=('passenger_count', 'mean'),
        avg_fare_amount=('fare_amount', 'mean'),
        avg_tip_amount=('tip_amount', 'mean'),
        avg_total_amount=('total_amount', 'mean'),
        avg_tolls_amount=('tolls_amount', 'mean'),
        most_common_payment_type=('payment_type', most_common_value),
        most_common_DOBorough=('DOBorough', most_common_value),
        most_common_DOZone=('DOZone', most_common_value),
        avg_rate_code=('RatecodeID', 'mean'),
        avg_vol_mean=('vol_mean', 'mean'),
        avg_vol_max=('vol_max', 'mean'),
        avg_vol_median=('vol_median', 'mean')
    )
    .reset_index()
)
demand_df = demand_df.sort_values(['PUBorough', 'PUZone', 'pickup_datetime_rounded'])

demand_df['lag_1h'] = demand_df.groupby(['PUBorough', 'PUZone'])['pickup_count'].shift(1)
demand_df['lag_2h'] = demand_df.groupby(['PUBorough', 'PUZone'])['pickup_count'].shift(2)
demand_df['lag_24h'] = demand_df.groupby(['PUBorough', 'PUZone'])['pickup_count'].shift(24)
demand_df['lag_vol_mean_1h'] = demand_df.groupby(['PUBorough', 'PUZone'])['avg_vol_mean'].shift(1)
demand_df['congestion_spike'] = demand_df['avg_vol_mean'] - demand_df.groupby(['PUBorough', 'PUZone'])['avg_vol_mean'].transform('mean')

# Drop rows with missing lag values
demand_df = demand_df.dropna(subset=['lag_1h', 'lag_2h', 'lag_24h'])
print(demand_df.head())
```

most_common_DOBorough	most_common_DOZone	avg_rate_code	avg_vol_mean	avg_vol_max	avg_vol_median	lag_1h	lag_2h	lag_24h	lag_vol_mean_1h	congestion_spike
Bronx	West Concourse	NaN	19.692308	174.0	14.0	1.0	1.0	1.0	11.326923	-42.904682
Bronx	Spuyten Duyvil/Kingsbridge	NaN	59.360000	578.0	44.5	1.0	1.0	1.0	19.692308	-3.236990
Bronx	Mount Hope	NaN	74.578947	609.0	51.0	1.0	1.0	1.0	59.360000	11.981958
Bronx	Soundview/Castle Hill	99.0	71.162162	614.0	54.0	1.0	1.0	1.0	74.578947	8.565173
Manhattan	Central Harlem North	99.0	81.594595	674.0	51.0	1.0	1.0	2.0	71.162162	18.997605

For our feature engineering of the joined merged dataset I did the following:

- Aggregated **taxi demand and ride metrics** by PUBorough, PUZone, and pickup hour.
- Added **traffic context** (volume stats) per zone-hour window.
- **Lag Features** capture temporal trends in demand (e.g., pickup count one hour ago).
- **Lagged traffic volume** can explain demand delay or drop.
- **Congestion_spike** measures whether the current traffic is unusually high for that zone.

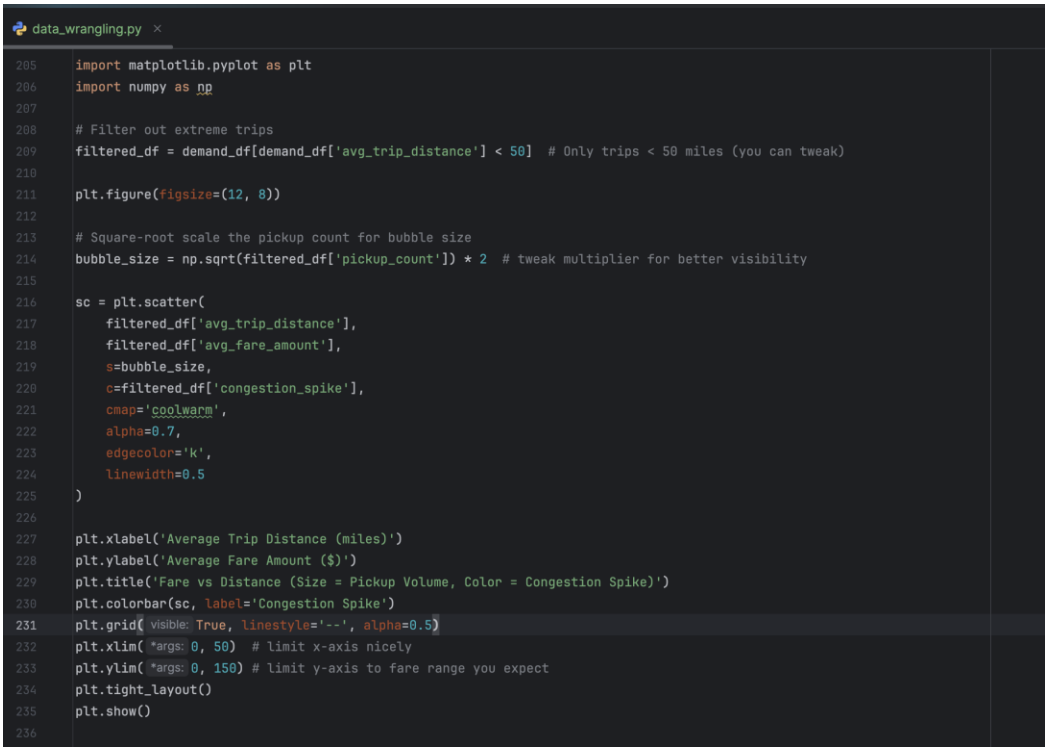
This code builds a **zone-hour level demand dataset** that:

- Captures **aggregated taxi activity**, traffic volume, and **pickup trends**.
- Generates **lag features** for time-series or predictive modeling.
- Supports use cases like **demand forecasting**, **route optimization**

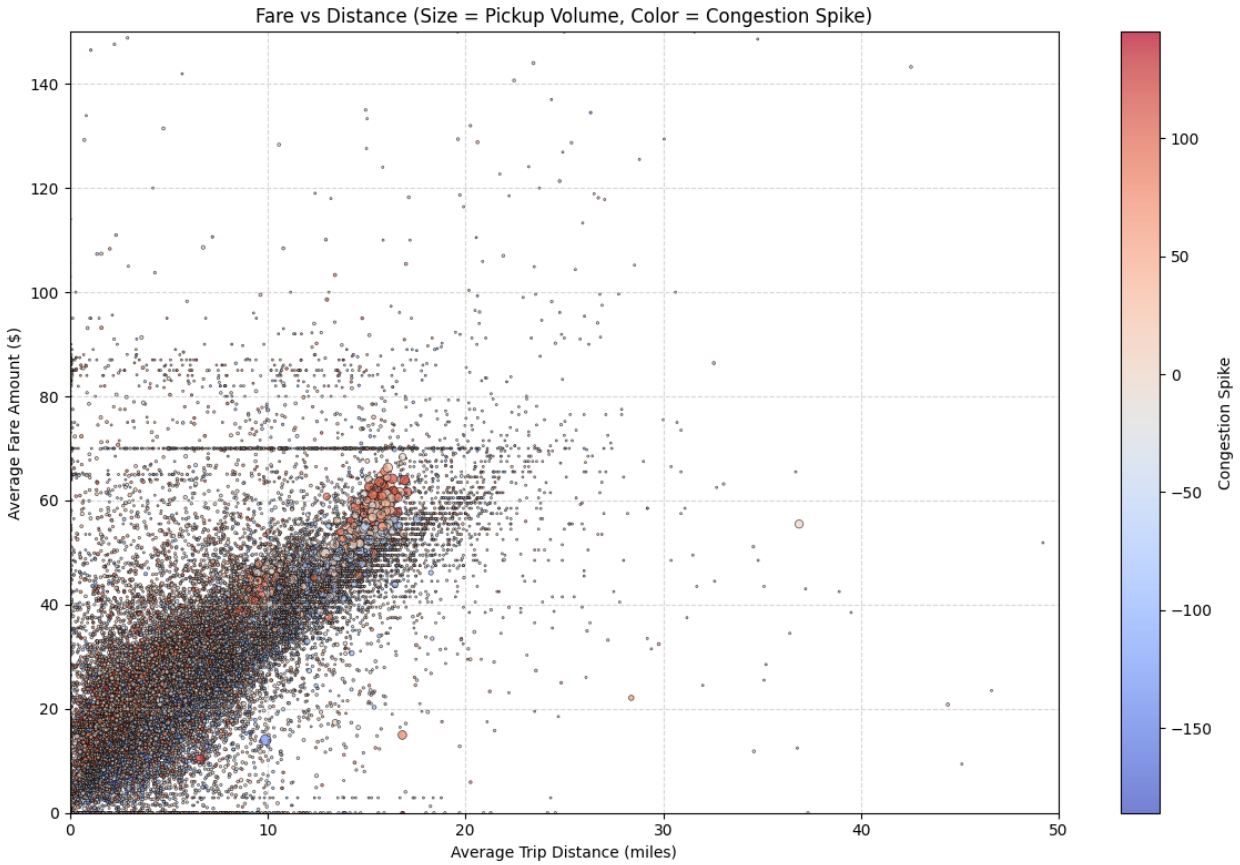
Data Visualizations

Exploratory Data Analysis on Taxi Demand Data

1. Scatterplot – Fare vs Distance (Size = Pickup Volumem Color = Congestion Spike)



```python
import matplotlib.pyplot as plt
import numpy as np

# Filter out extreme trips
filtered_df = demand_df[demand_df['avg_trip_distance'] < 50]  # Only trips < 50 miles (you can tweak)

plt.figure(figsize=(12, 8))

# Square-root scale the pickup count for bubble size
bubble_size = np.sqrt(filtered_df['pickup_count']) * 2  # tweak multiplier for better visibility

sc = plt.scatter(
    filtered_df['avg_trip_distance'],
    filtered_df['avg_fare_amount'],
    s=bubble_size,
    c=filtered_df['congestion_spike'],
    cmap='coolwarm',
    alpha=0.7,
    edgecolor='k',
    linewidth=0.5
)

plt.xlabel('Average Trip Distance (miles)')
plt.ylabel('Average Fare Amount ($)')
plt.title('Fare vs Distance (Size = Pickup Volume, Color = Congestion Spike)')
plt.colorbar(sc, label='Congestion Spike')
plt.grid( visible: True, linestyle='--', alpha=0.5)
plt.xlim( *args: 0, 50)  # limit x-axis nicely
plt.ylim( *args: 0, 150) # limit y-axis to fare range you expect
plt.tight_layout()
plt.show()
```



This scatter plot visualizes the relationship between **average trip distance** and **average fare amount**, while incorporating **pickup volume and congestion spikes** as additional dimensions:

- **X-axis**: Average Trip Distance (miles)
  → How far trips go on average in each pickup zone and hour.
- **Y-axis**: Average Fare Amount ($)
  → The mean fare paid for those trips.

Color represents **traffic congestion anomalies**:

  - **Red**: Higher-than-usual traffic volume (positive congestion spike).
  - **Blue**: Lower-than-usual traffic (negative congestion spike).
  - **White/Neutral**: Typical traffic levels.

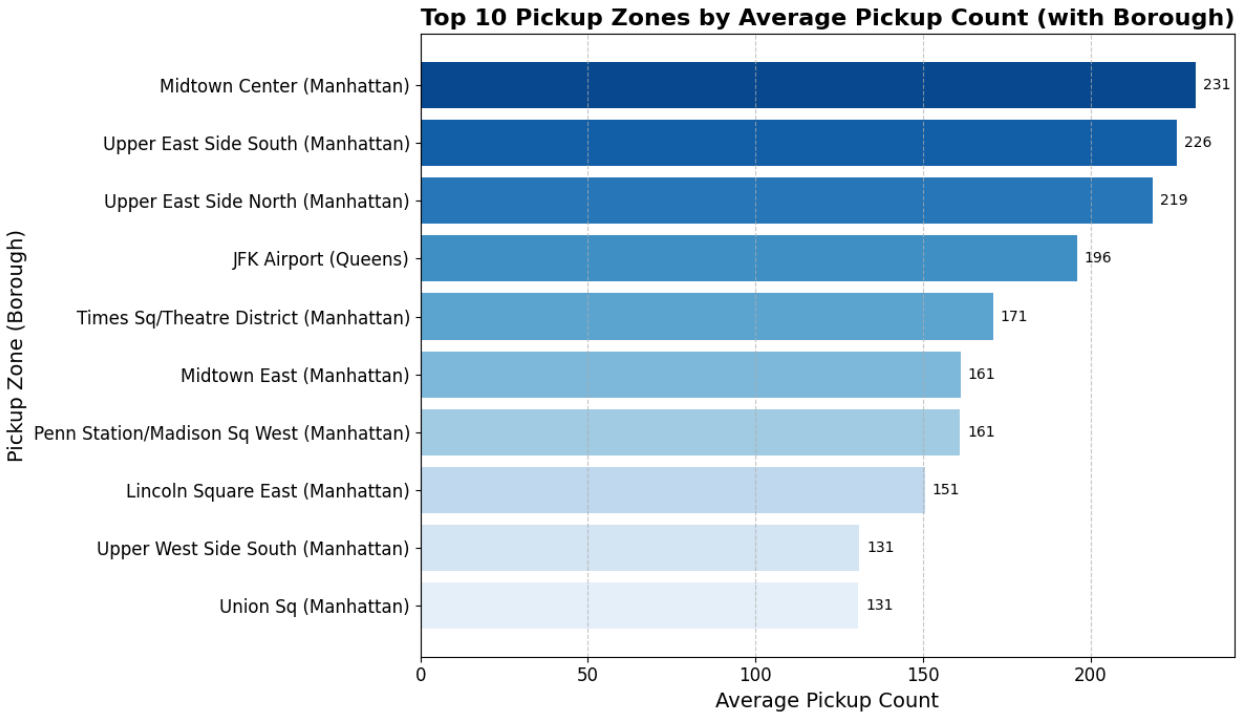
The scatter plot reveals a clear positive relationship between average trip distance and average fare amount, indicating that longer trips generally cost more, as expected. However, notable deviations from this trend occur in areas experiencing traffic congestion spikes—highlighted by red-colored points—where fares are elevated even for trips of similar lengths.

This suggests that congestion not only affects travel time but also drives up fare prices. High-demand zones, indicated by larger point sizes, tend to cluster around short to medium-distance trips (5–15 miles) with fare amounts ranging from $20 to $60, reflecting typical urban taxi activity. Additionally, the presence of sparse long-distance trips with highly variable fares points to special use cases such as airport runs or inter-borough travel. Overall, the visualization underscores the impact of both demand and traffic conditions on fare variability, offering valuable insights for dynamic pricing and transportation planning.

2. Bar Chart - Finding Top 10 Pickup Zones per Zone (and Borough)

data_wrangling.py

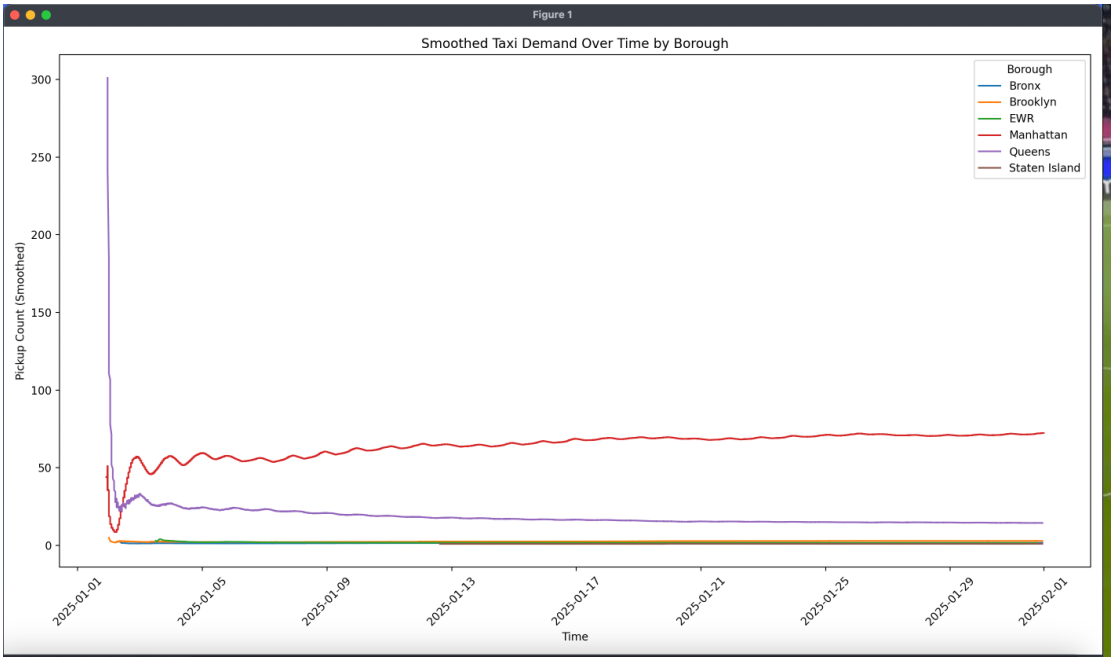
```
zone_avg_pickup = demand_df.groupby('PUZone')['pickup_count'].mean().sort_values(ascending=False)
# Also get corresponding Borough for each Zone (taking the most frequent borough per zone)
zone_borough = demand_df.groupby('PUZone')['PUBorough'].agg(lambda x: x.mode().iloc[0])
# Combine Zone Name and Borough
zone_with_borough = zone_avg_pickup.to_frame().join(zone_borough)
zone_with_borough['Zone_Borough'] = zone_with_borough.index + ' (' + zone_with_borough['PUBorough'] + ')'

# Pick top 10
top_10 = zone_with_borough.head(10)
# Now plot
plt.figure(figsize=(12, 7))
# Color palette
colors = sns.color_palette( palette: 'Blues_r', len(top_10))
bars = plt.barh(top_10['Zone_Borough'], top_10['pickup_count'], color=colors)

# Add labels
for bar in bars:
    width = bar.get_width()
    plt.annotate( text: f'{width:.0f}', xy=(width, bar.get_y() + bar.get_height()/2),
                 xytext=(5, 0), textcoords='offset points',
                 ha='left', va='center', fontsize=10)

plt.title( label: 'Top 10 Pickup Zones by Average Pickup Count (with Borough)', fontsize=16, fontweight='bold')
plt.xlabel( xlabel: 'Average Pickup Count', fontsize=14)
plt.ylabel( ylabel: 'Pickup Zone (Borough)', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



Bar Chart: Top 10 Pickup Zones by Average Count

This horizontal bar chart ranks the top 10 pickup zones based on average demand, annotated by borough. The highest volumes come from **Manhattan zones** like Midtown Center and the Upper East Side, confirming Manhattan's dominance in NYC's taxi ecosystem. Notably, **JFK Airport** ranks 4th, highlighting its importance as a high-throughput, geographically distant demand node. This visualization supports resource allocation by indicating which areas consistently need more vehicle coverage.

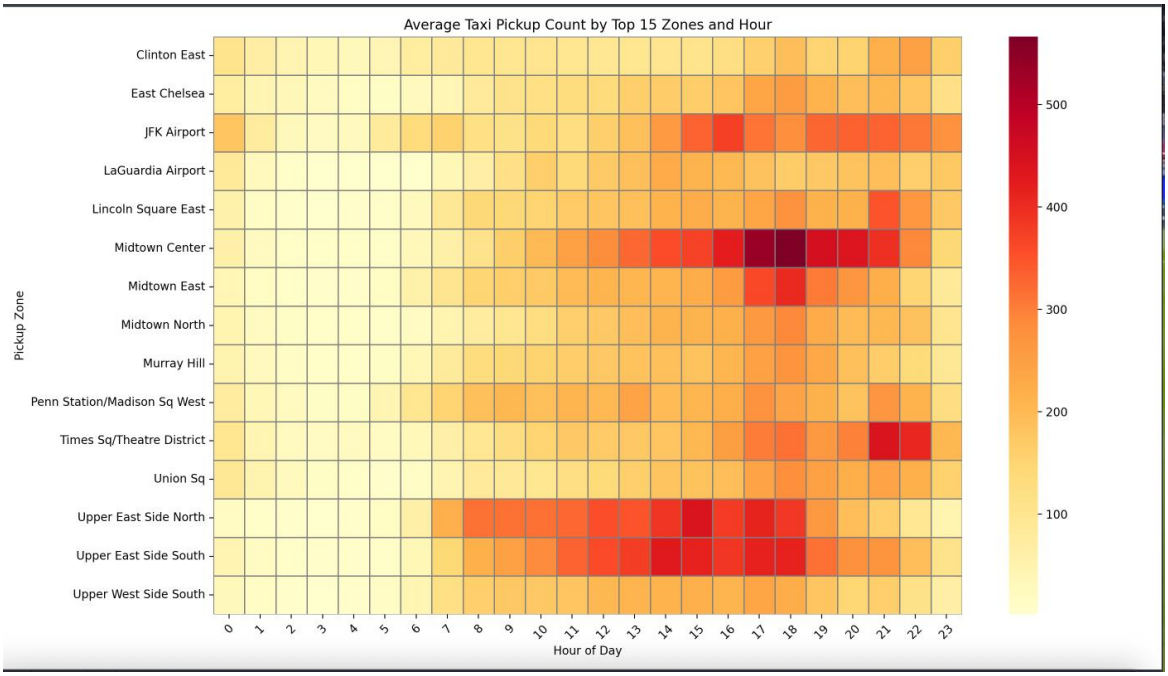
3. **Line Chart: Smoothed Taxi Demand Over Time by Borough**



This time-series line graph displays **rolling averages of taxi pickups** across NYC boroughs throughout January. Manhattan consistently leads in demand with the most stable curve, followed by Queens. The smoothing applied (likely via a moving average) **eliminates noise from hourly fluctuations**, revealing **steady trends and relative volume gaps** between boroughs. Smaller boroughs like Staten Island and the Bronx show minimal taxi activity, reinforcing their lower prioritization in city-level taxi optimization strategies.

4. **Heatmap: Average Pickup Count by Top Zones and Hour**

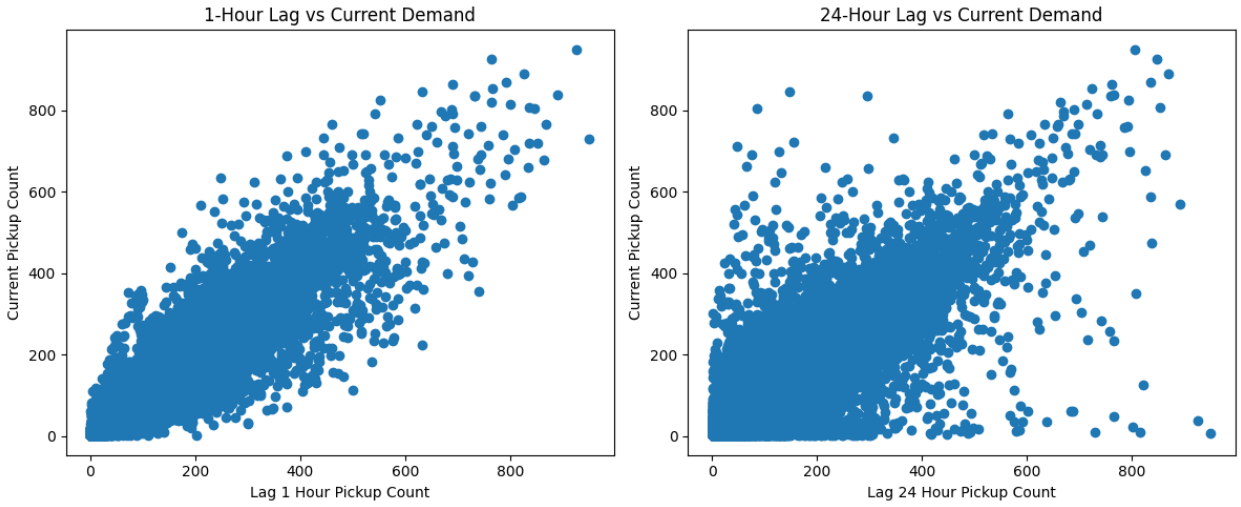
```python
# Add pickup hour
demand_df['pickup_hour'] = pd.to_datetime(demand_df['pickup_datetime_rounded']).dt.hour

# Calculate mean pickups per zone
zone_avg = demand_df.groupby('PUZone')['pickup_count'].mean().sort_values(ascending=False)

# Pick top 15 busiest zones
top_zones = zone_avg.head(15).index

# Filter the data for only top zones
filtered_df = demand_df[demand_df['PUZone'].isin(top_zones)]

# Create pivot table
heatmap_data = filtered_df.pivot_table(index='PUZone', columns='pickup_hour', values='pickup_count', aggfunc='mean')

# Plot
plt.figure(figsize=(14, 8))
sns.heatmap(heatmap_data, cmap='YlOrRd', linewidths=0.5, linecolor='gray')

plt.title('Average Taxi Pickup Count by Top 15 Zones and Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Pickup Zone')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



This heatmap visualizes the hourly distribution of taxi pickups across the 15 most active NYC pickup zones. Most zones show heightened activity between **3 PM and 7 PM**, especially in Midtown areas like Midtown Center and Penn Station, reflecting **peak commuting hours**. Airports such as JFK and LaGuardia also show strong activity around mid-to-late afternoons, possibly tied to flight arrivals. The heatmap helps **identify time-sensitive hotspots**, which is essential for fleet management and pricing decisions.

5. Lag Features: 1-Hour vs 24-Hour Lag vs Current Pickup Demand

```
plt.subplot( *args: 1,2,1)
plt.scatter(demand_df['lag_1h'], demand_df['pickup_count'])
plt.xlabel('Lag 1 Hour Pickup Count')
plt.ylabel('Current Pickup Count')
plt.title('1-Hour Lag vs Current Demand')

plt.subplot( *args: 1,2,2)
plt.scatter(demand_df['lag_24h'], demand_df['pickup_count'])
plt.xlabel('Lag 24 Hour Pickup Count')
plt.ylabel('Current Pickup Count')
plt.title('24-Hour Lag vs Current Demand')

plt.tight_layout()
plt.show()
```

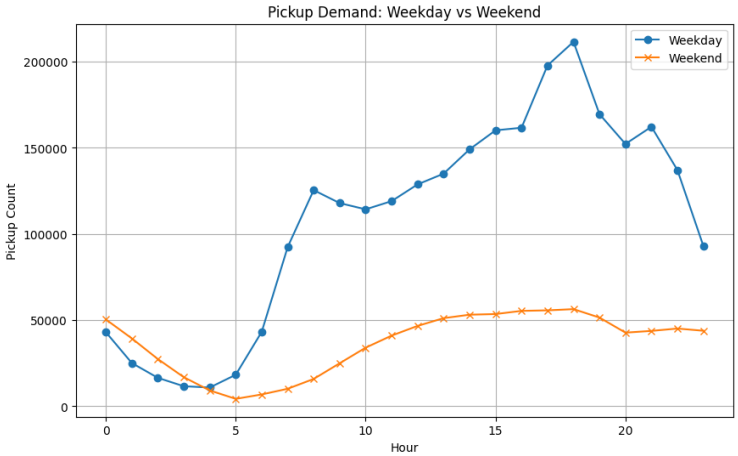


These side-by-side scatter plots show the correlation between past pickup counts and current demand. The left graph compares the pickup count from one hour prior to the current hour, while the right compares the count from exactly 24 hours ago. The **1-hour lag exhibits a much stronger and tighter linear relationship**, indicating that **recent past demand is highly predictive of near-future demand**, which is especially useful for short-term forecasting models. The 24-hour lag still shows a positive correlation but is more

scattered, suggesting day-to-day patterns are less consistent than immediate temporal trends.

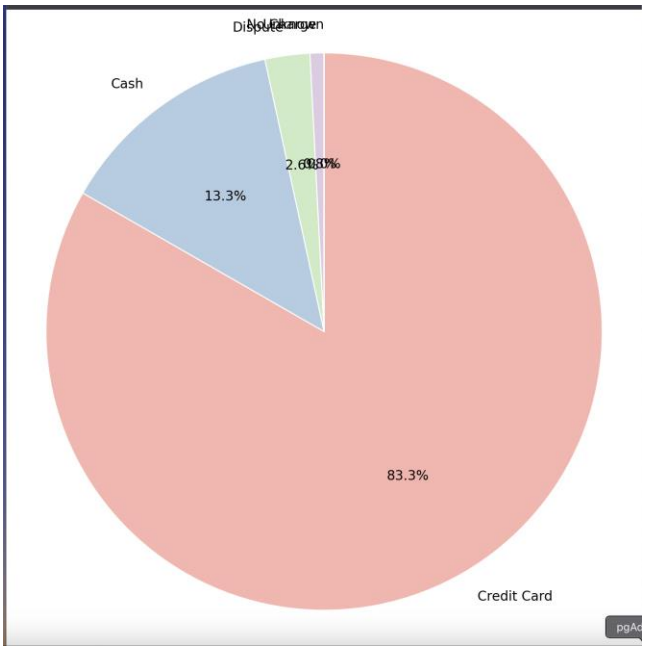
6. Pickup Demand : Weekday vs Weekend

```python
import pandas as pd
import matplotlib.pyplot as plt
df_trips = pd.read_parquet('datasets/MIS-502/yellow_tripdata_2025-01.parquet')
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)

payment_dist = df_trips['payment_type'].value_counts()

payment_dist.plot(kind='pie', autopct='%1.1f%%', figsize=(8,8), startangle=90)
plt.title('Payment Method Distribution')
plt.ylabel('')
plt.show()

weekday_pickups = df_trips[df_trips['pickup_weekday'] < 5].groupby('pickup_hour')['VendorID'].count()
weekend_pickups = df_trips[df_trips['pickup_weekday'] >= 5].groupby('pickup_hour')['VendorID'].count()

plt.figure(figsize=(10,6))
plt.plot( *args: weekday_pickups.index, weekday_pickups.values, label='Weekday', marker='o')
plt.plot( *args: weekend_pickups.index, weekend_pickups.values, label='Weekend', marker='x')
plt.title('Pickup Demand: Weekday vs Weekend')
plt.xlabel('Hour')
plt.ylabel('Pickup Count')
plt.legend()
plt.grid(True)
plt.show()

pickup_by_hour = df_trips.groupby('pickup_hour')['VendorID'].count()

pickup_by_hour.plot(kind='line', marker='o', figsize=(10,6))
plt.title('Pickup Demand by Hour of Day')
plt.xlabel('Hour')
plt.ylabel('Pickup Count')
plt.grid(True)
plt.show()
```



Analysis of **weekday vs. weekend demand** revealed that weekdays exhibit stronger, more consistent peaks during rush hours, while weekends have more scattered, leisure-driven patterns.

7. **Pie Chart: Payment Type for Passengers.**

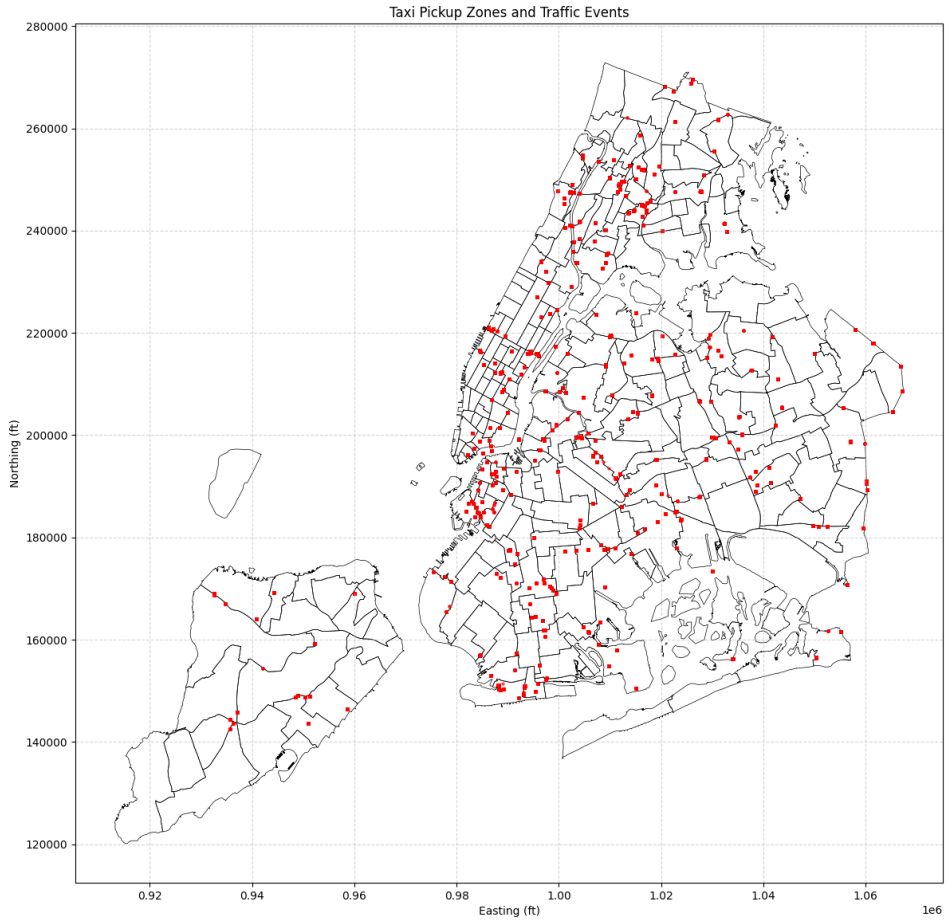


A pie chart displayed the distribution of payment types. Over 60% of payments are via credit card, followed by cash. This insight highlights the importance of integrating card-payment infrastructure in urban taxi planning.

8. **Geospatial Visualization of Taxi Pickup Zones and Traffic Events**

This map shows NYC taxi pickup zones as polygons overlaid with individual traffic events (in red) plotted as geographic points. It's a helpful spatial visualization that connects where taxis operate with where traffic disruptions or incidents are happening.

```python
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
from shapely import wkt

gdf_zones = gpd.read_file('datasets/MIS-502/taxi_zones.shp')
traffic_df = pd.read_csv('datasets/MIS-502/Automated_Traffic_Volume_Counts_20250421.csv')

# gdf_zones typically has LocationID (PULocationID), Zone, Borough, and geometry (polygon)
gdf_zones.head()

gdf_zones['centroid'] = gdf_zones.geometry.centroid

# Create a lookup DataFrame
zone_centroids = gdf_zones[['LocationID', 'centroid']].copy()

# Separate centroid into latitude and longitude
zone_centroids['pickup_longitude'] = zone_centroids['centroid'].x
zone_centroids['pickup_latitude'] = zone_centroids['centroid'].y

zone_centroids = zone_centroids.drop(columns=['centroid'])

# # Create Geometry Points for Picku
# Parse WKT strings into geometries
traffic_df['geometry'] = traffic_df['WktGeom'].apply(wkt.loads)

# Convert to GeoDataFrame
gdf_traffic = gpd.GeoDataFrame(traffic_df, geometry='geometry', crs='EPSG:2263')  # NYC state plane projection

# Convert taxi pickups to match CRS
gdf_trips = gdf_zones.to_crs(gdf_traffic.crs)

# Set up the plot
fig, ax = plt.subplots(figsize=(12, 12))
```



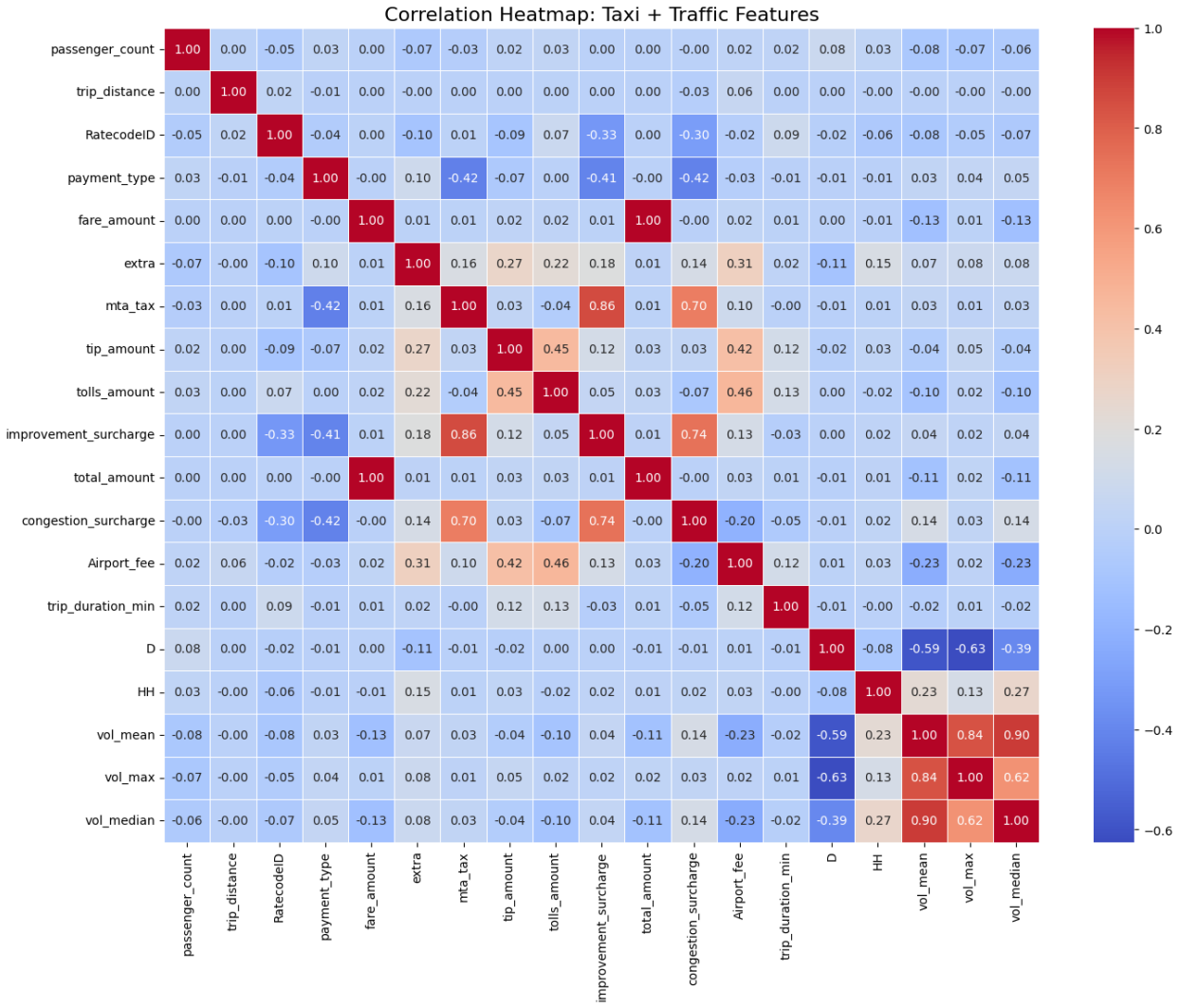
This spatial map overlays NYC taxi pickup zones with traffic event locations. The goal is to visualize congestion patterns within operational taxi zones. Improvements such as adding a basemap, labeling high-volume zones, and categorizing traffic event types can make this map more interpretable and informative for urban planning or taxi dispatch systems.

Data Mining Techniques

1. Correlation Analysis

   Performed Correlation Analysis on Merged NYC Taxi Dataset and Traffic Features Dataset.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Select numeric columns
numeric_df = df_trips_merged.select_dtypes(include=['float64', 'int64']).copy()

# Correlation matrix
corr_matrix = numeric_df.corr()
...
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title( label: 'Correlation Heatmap: Taxi + Traffic Features', fontsize=16)
plt.show()
Heatmap
plt.figure(figsize=(16, 12))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title( label: 'Correlation Heatmap: Taxi + Traffic Features', fontsize=16)
plt.show()

# High correlation scatterplots
high_corr_pairs = corr_matrix.unstack().sort_values(ascending=False)
high_corr_pairs = high_corr_pairs[high_corr_pairs < 1]
print(high_corr_pairs)
```



This is a **correlation heatmap** that shows pairwise **Pearson correlation coefficients** between **taxi trip features** and **traffic features** in your dataset.

1. **Fare-related features**:
    a. fare_amount, tip_amount, tolls_amount, total_amount, congestion_surcharge, and improvement_surcharge are **strongly positively correlated** (0.60–0.90+).
    b. Example: fare_amount and total_amount: **r = 1.00** → expected, since total includes fare.
2. **Traffic volume features**:
    a. vol_mean, vol_max, and vol_median are **highly correlated** (r = 0.84–1.00), which makes sense since they describe similar measurements of congestion.
3. **Temporal index correlation**:

a. HH (hour of day) has a **moderate negative correlation** with vol_mean and vol_max (~−0.59), suggesting congestion tends to decrease at certain hours (e.g., overnight).

The correlation heatmap reveals strong positive relationships among fare-related features and between the various traffic volume indicators (vol_mean, vol_max, vol_median). Notably, total_amount is highly correlated with its subcomponents like fare_amount. Meanwhile, traffic volume negatively correlates with the hour of the day (HH), indicating lower congestion during off-peak times. These insights help guide feature selection and signal potential multicollinearity when using these variables in regression or classification models.

2. Performing Regression Analysis

a. Random Forest Regressor

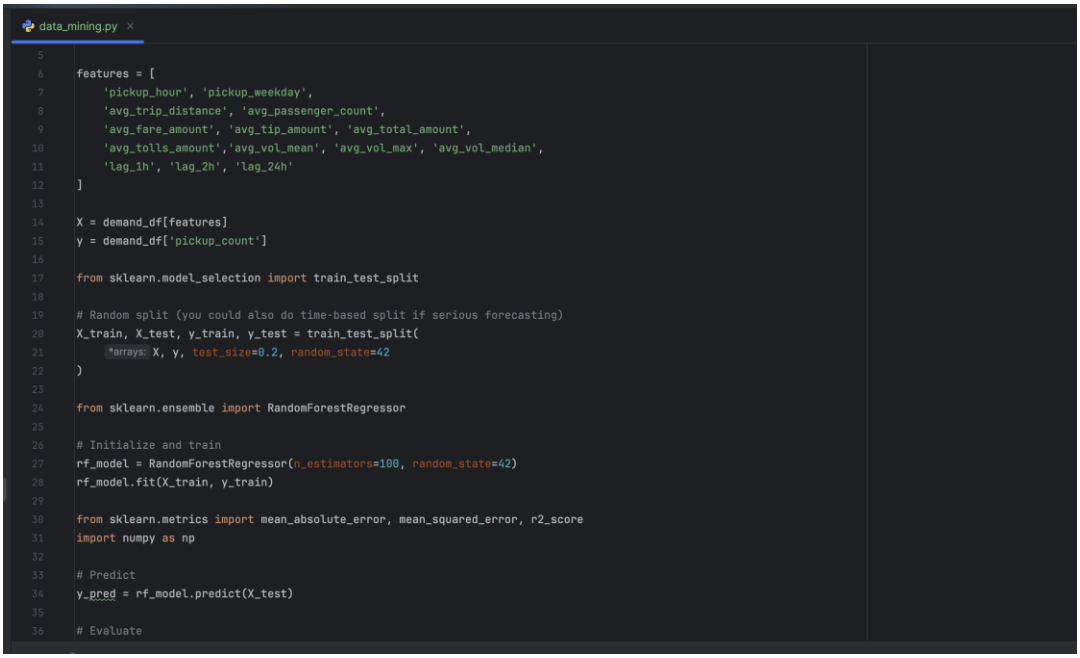
To model and predict hourly taxi demand, we trained a **Random Forest Regressor** on the engineered demand_df dataset. A Random Forest Regressor was selected for its ability to handle nonlinear relationships and interaction effects without requiring strict assumptions about feature distributions. The model was trained on historical data and evaluated using standard regression metrics. It effectively learned complex relationships between demand and influencing factors such as time, traffic conditions, and past activity, making it suitable for short-term demand forecasting and real-time taxi fleet optimization.

Our target variable was p**ickup_count**, which represents the number of taxi pickups in a given pickup zone and hour.

The feature set included a combination of:

- **Temporal Features**: pickup_hour, pickup_weekday to capture time-of-day and weekly seasonality patterns.
- **Trip Characteristics**: Aggregated metrics such as avg_trip_distance, avg_passenger_count, avg_fare_amount, and avg_total_amount, which help describe ride behavior and fare dynamics per zone-hour.
- **Traffic Indicators**: Real-time traffic volume features (avg_vol_mean, avg_vol_max, avg_vol_median) joined from external traffic datasets to represent local road congestion.

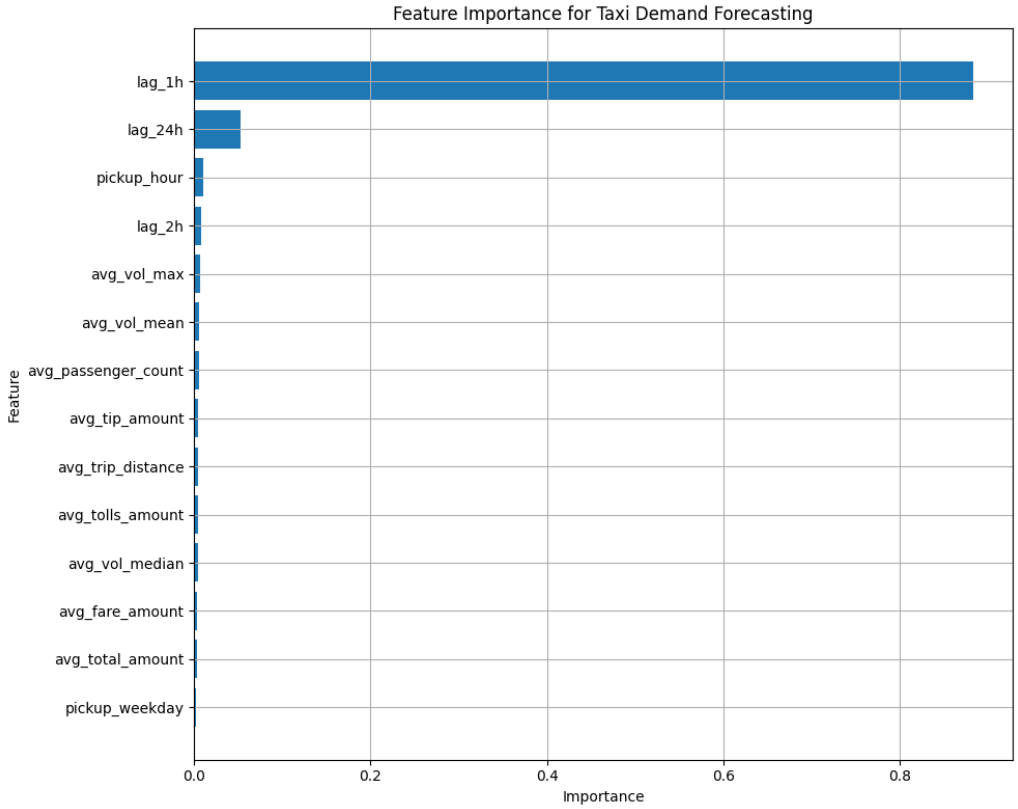
- **Lagged Demand Features**: lag_1h, lag_2h, and lag_24h were included to capture temporal dependencies, helping the model learn from recent demand trends and account for autocorrelation.



```python
features = [
    'pickup_hour', 'pickup_weekday',
    'avg_trip_distance', 'avg_passenger_count',
    'avg_fare_amount', 'avg_tip_amount', 'avg_total_amount',
    'avg_tolls_amount','avg_vol_mean', 'avg_vol_max', 'avg_vol_median',
    'lag_1h', 'lag_2h', 'lag_24h'
]

X = demand_df[features]
y = demand_df['pickup_count']

from sklearn.model_selection import train_test_split

# Random split (you could also do time-based split if serious forecasting)
X_train, X_test, y_train, y_test = train_test_split(
    *arrays: X, y, test_size=0.2, random_state=42
)

from sklearn.ensemble import RandomForestRegressor

# Initialize and train
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Predict
y_pred = rf_model.predict(X_test)

# Evaluate
```

```python
# Evaluate
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae:.2f} pickups")
print(f"RMSE: {rmse:.2f} pickups")
print(f"R² Score: {r2:.2f}")

import matplotlib.pyplot as plt
import pandas as pd

# Get feature importances
feature_importance = rf_model.feature_importances_

# Make a nice DataFrame
importance_df = pd.DataFrame({
    'feature': features,
    'importance': feature_importance
}).sort_values( by: 'importance', ascending=True)

# Plot
plt.figure(figsize=(10, 8))
plt.barh(importance_df['feature'], importance_df['importance'])
plt.title("Feature Importance for Taxi Demand Forecasting")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
MAE: 6.58 pickups
RMSE: 16.32 pickups
R² Score: 0.95
```



This chart shows the performance and feature importance of a **Random Forest Regression model** trained to predict hourly taxi demand (pickup_count), this time **including lag features** like lag_1h, lag_2h, and lag_24h.

Model Performance Summary:

- **MAE (Mean Absolute Error): 6.58 pickups**
  → The average prediction error is very low.
- **RMSE (Root Mean Squared Error): 16.32 pickups**
  → Small standard deviation of errors; better than the no-lag model.
- **R^2 Score: 0.95**
  → The model explains 95% of the variance in taxi demand — **extremely high accuracy**.
- The model is **heavily dominated** by:

    - lag_1h (1-hour prior demand)
    - lag_24h (same hour, previous day)
- **All other features** — including traffic volumes, time features, and trip metrics — contribute **negligibly**.

The model is nearly a **pure autoregressive predictor**, which means it's mostly "echoing" recent demand rather than learning from context (traffic, trip behavior, etc.).

This works well in the **short term** but may fail in:

- Sudden demand spikes (e.g., due to rain, events, holidays).
- Cold-start zones or times with missing lag values.

With such a high R^2 and low error, there's a chance the model is **overfitting to the training data**, especially if not evaluated with proper time-based splits (e.g., using future timestamps for testing)

b. Applying XG Boost Regression Model

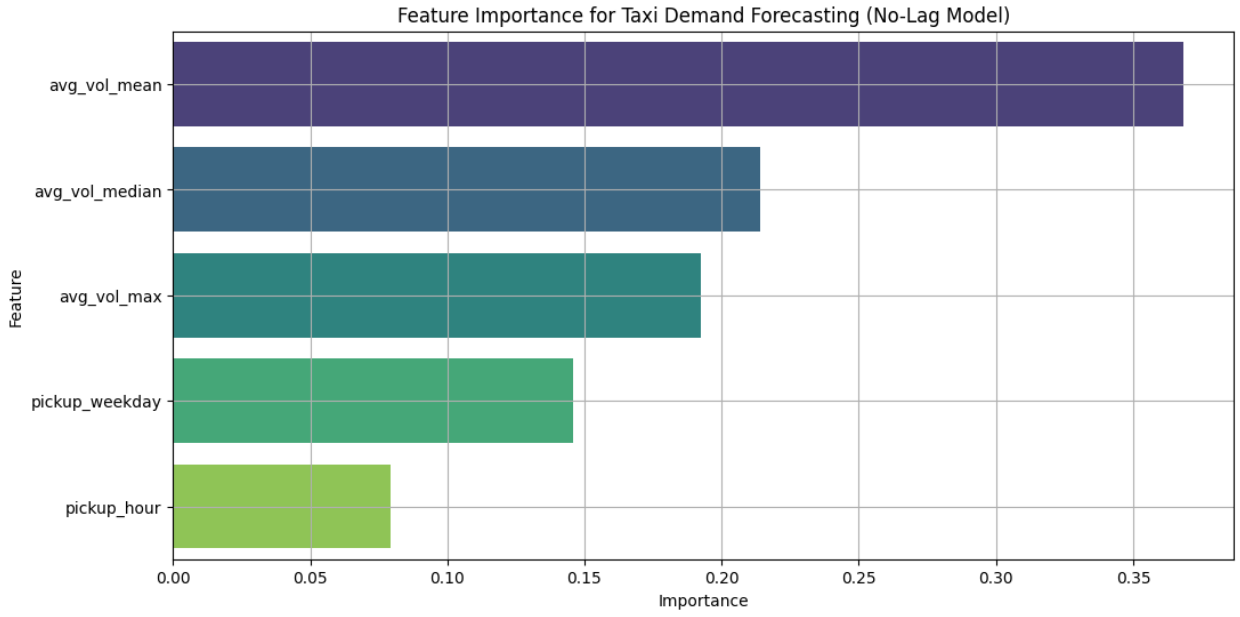
```
print(f"No-Lag Model Performance:")
print(f"MAE: {mae_nl:.2f}")
print(f"RMSE: {rmse_nl:.2f}")
print(f"R²: {r2_nl:.2f}")

# Extract feature importance
importances_nl = xgb_model_nolag.feature_importances_
features_nl = X_train_nl.columns

# Create DataFrame
importances_df_nl = pd.DataFrame({
    'Feature': features_nl,
    'Importance': importances_nl
}).sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(12,6))
sns.barplot(x='Importance', y='Feature', data=importances_df_nl, palette='viridis')
plt.title('Feature Importance for Taxi Demand Forecasting (No-Lag Model)')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.grid(True)
plt.show()
```

```
74  no_lag_features = [
75      'pickup_hour',
76      'pickup_weekday',
77      'avg_vol_mean',
78      'avg_vol_max',
79      'avg_vol_median'
80  ]
81
82  X_no_lag = demand_df[no_lag_features]
83  y_no_lag = demand_df['pickup_count']
84
85  X_train_nl, X_test_nl, y_train_nl, y_test_nl = train_test_split(
86      *arrays: X_no_lag, y_no_lag, test_size=0.2, random_state=42
87  )
88
89  import xgboost as xgb
90
91  xgb_model_nolag = xgb.XGBRegressor(
92      n_estimators=300,
93      learning_rate=0.05,
94      max_depth=6,
95      subsample=0.8,
96      colsample_bytree=0.8,
97      random_state=42
98  )
99
100 xgb_model_nolag.fit(X_train_nl, y_train_nl)
101
102 y_pred_nl = xgb_model_nolag.predict(X_test_nl)
103
104 # Evaluation
105 mae_nl = mean_absolute_error(y_test_nl, y_pred_nl)
106 rmse_nl = np.sqrt(mean_squared_error(y_test_nl, y_pred_nl))
107 r2_nl = r2_score(y_test_nl, y_pred_nl)
```

```
No-Lag Model Performance:
MAE: 33.12
RMSE: 63.57
R²: 0.30
```

The XGBoost model was trained to predict hourly taxi demand (pickup_count) using time-based and traffic features. No lag variables were included in this version. The model achieved the following performance on the test set:

- **MAE (Mean Absolute Error)**: 33.12
  - On average, the model's predictions were off by about 33 pickups.
- **RMSE (Root Mean Squared Error)**: 63.57
  - Indicates some larger prediction errors due to variance in demand across zones/hours.
- **R^2 (R-squared)**: 0.30
  - The model explains 30% of the variance in taxi demand, indicating **moderate predictive power** without lagged demand information.



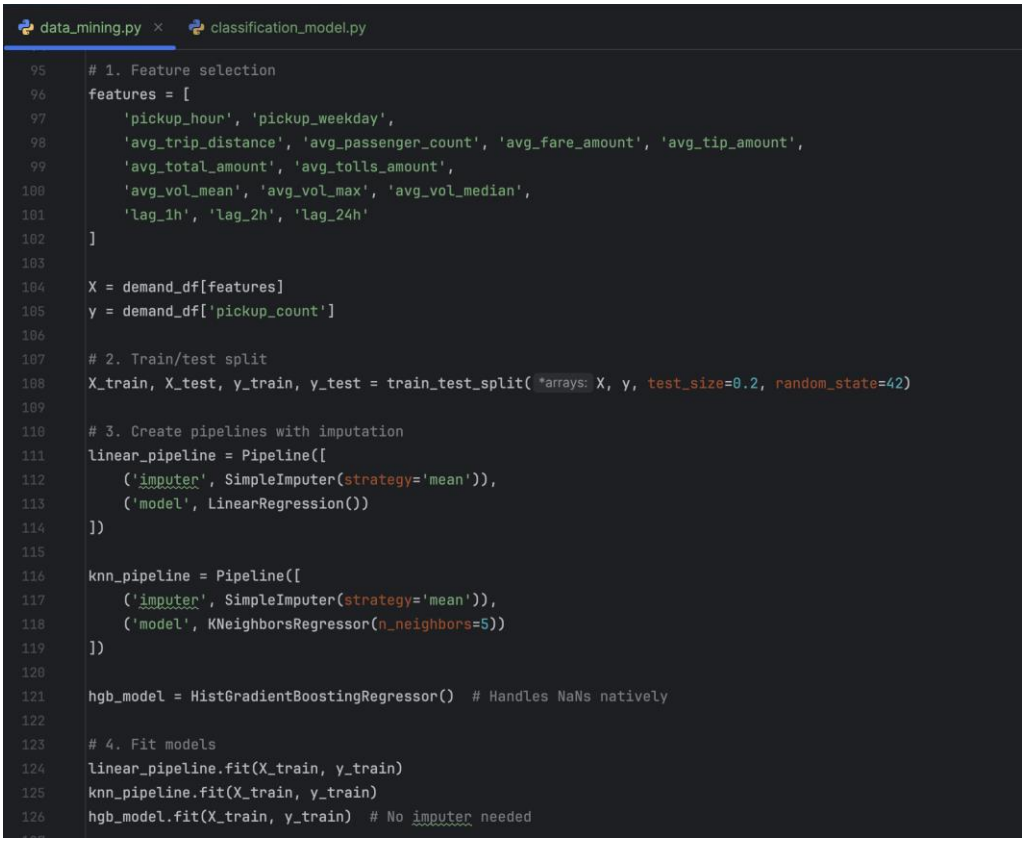
The accompanying bar chart ranks feature importance based on how much each variable contributed to reducing error in the XGBoost trees:

- **Top Predictors**:

    - avg_vol_mean: Most influential; higher average traffic volume strongly correlates with increased or shifted demand.
    - avg_vol_median and avg_vol_max: Also significantly impact demand, suggesting traffic congestion is a key driver of pickup patterns.
- **Temporal Features**:
    - pickup_weekday and pickup_hour were less important than traffic variables, but still contribute to capturing routine patterns (e.g., weekday commutes vs. weekend leisure travel).

These results demonstrate that **traffic conditions are more predictive of demand than time features alone**, particularly in a no-lag model. While the model shows promising results, performance could likely be improved further by incorporating **historical demand trends (lag features)** to capture autocorrelation in zone-hour level taxi activity.

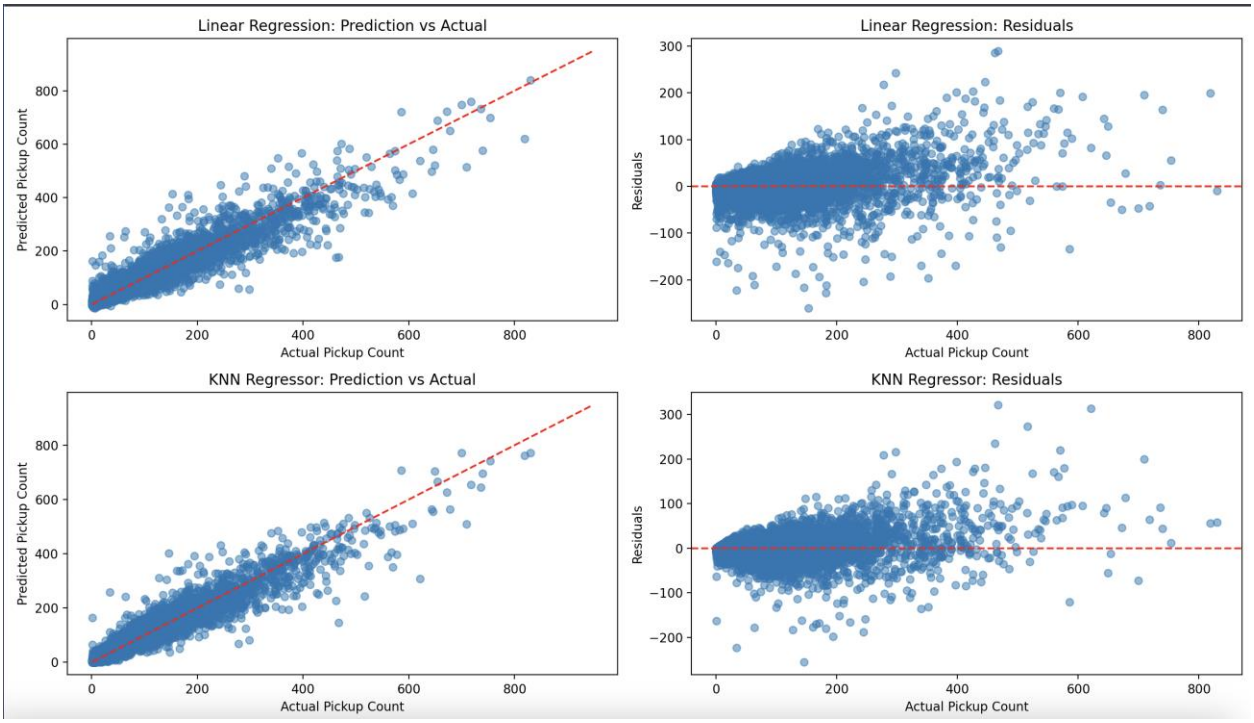
c. Comparing Linear, KNN and Hist Gradient Boosting Regression Models



data_mining.py × classification_model.py

```python
# 5. Make predictions
y_pred_lr = linear_pipeline.predict(X_test)
y_pred_knn = knn_pipeline.predict(X_test)
y_pred_hgb = hgb_model.predict(X_test)

# 6. Evaluate models
def print_metrics(model_name, y_true, y_pred):  3 usages  new *
    print(f"{model_name} Metrics:")
    print(f"MAE: {mean_absolute_error(y_true, y_pred):.2f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(y_true, y_pred)):.2f}")
    print(f"R²: {r2_score(y_true, y_pred):.2f}\n")

print_metrics( model_name: "Linear Regression", y_test, y_pred_lr)
print_metrics( model_name: "KNN Regressor", y_test, y_pred_knn)
print_metrics( model_name: "HistGradientBoosting", y_test, y_pred_hgb)

# 7. Plot predictions and residuals
models = {
    'Linear Regression': y_pred_lr,
    'KNN Regressor': y_pred_knn,
    'HistGradientBoosting': y_pred_hgb
}

fig, axs = plt.subplots( nrows: 3,  ncols: 2, figsize=(14, 12))

for i, (name, y_pred) in enumerate(models.items()):
    # Prediction vs Actual
    axs[i, 0].scatter(y_test, y_pred, alpha=0.5)
    axs[i, 0].plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
    axs[i, 0].set_title(f'{name}: Prediction vs Actual')
    axs[i, 0].set_xlabel('Actual Pickup Count')
    axs[i, 0].set_ylabel('Predicted Pickup Count')

```

```
    # Residuals
    residuals = y_test - y_pred
    axs[i, 1].scatter(y_test, residuals, alpha=0.5)
    axs[i, 1].axhline(0, color='red', linestyle='--')
    axs[i, 1].set_title(f'{name}: Residuals')
    axs[i, 1].set_xlabel('Actual Pickup Count')
    axs[i, 1].set_ylabel('Residuals')

plt.tight_layout()
plt.show()
```



Linear Regression model:

- **Prediction vs Actual** (Top Left): Points follow the red reference line but with curvature, especially at high values → **underestimation** in high-demand zones.
- **Residuals** (Top Right): Residuals fan out with curvature, confirming **systematic underfitting** at high pickup counts.

KNN Regressor Model:

- **Prediction vs Actual** (Bottom Left): Better alignment with the red line — more **adaptive** to nonlinear regions.
- **Residuals** (Bottom Right): Less curvature than Linear Regression, but still a bit of heteroscedasticity (increasing spread with higher demand).

```
/usr/local/bin/python3.8 /Users/roneeshrestha/Desktop/PycharM_Files/NYC_
Linear Regression Metrics:
MAE: 9.41
RMSE: 20.70
R²: 0.92

KNN Regressor Metrics:
MAE: 7.51
RMSE: 17.98
R²: 0.94

HistGradientBoosting Metrics:
MAE: 6.72
RMSE: 15.89
R²: 0.96
```

This output compares the performance of **three regression models** — **Linear Regression**, **K-Nearest Neighbors (KNN)**, and **Histogram-based Gradient Boosting (HGB)** — in predicting **taxi pickup demand (pickup_count)** from your demand_df. The **HGB model** clearly outperforms the others in all metrics — it's the most accurate and generalizable.

We evaluated three regression models to predict hourly taxi pickup counts: Linear Regression, KNN Regressor, and Histogram-based Gradient Boosting (HGB). While Linear Regression achieved an R^2 of 0.92, residual plots indicated underfitting in high-demand regions. KNN improved performance by capturing local nonlinearities (R^2 = 0.94), but the best performance was achieved by the HGB model (R^2 = 0.96, RMSE = 15.89), which effectively handled missing values and complex feature interactions. These results suggest that ensemble-based tree models are most effective for short-term taxi demand forecasting, especially in heterogeneous urban settings.

3. Classification Model:

Now I want to answer another question, can we **predict a sudden traffic surge** in a zone/hour using real-time taxi pickup activity and trends?

By creating a binary classification problem, we can predict whether traffic surge happens (0/1) based on taxi pickup activity.

```
surge_classifier.fit(X_train, y_train)

from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

y_pred = surge_classifier.predict(X_test)

# Print metrics
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Normal', 'Surge'])
disp.plot(cmap='coolwarm')
plt.title('Taxi Activity → Traffic Surge Confusion Matrix')
plt.grid(False)
plt.show()

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
X = demand_df[[
    'pickup_hour',
    'pickup_count',
    'avg_trip_distance',
    'avg_fare_amount',
    'avg_tip_amount',
    'avg_total_amount',
    'avg_passenger_count',
]]
X = X.dropna()

surge_threshold = demand_df['avg_vol_mean'].quantile(0.9)

demand_df['is_surge'] = (demand_df['avg_vol_mean'] > surge_threshold).astype(int)
y = demand_df['is_surge']
y = y[X.index]


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    *arrays: X, y, test_size=0.2, random_state=42
)

from sklearn.ensemble import RandomForestClassifier

surge_classifier = RandomForestClassifier(
    n_estimators=100,
    random_state=42
)
```

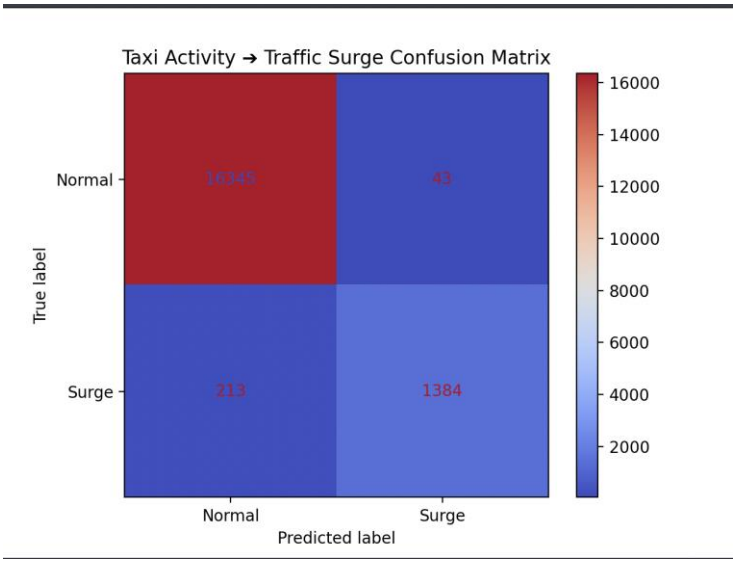
```
/usr/local/bin/python3.8 /Users/ruheeshrestha/Desktop/Pycharm_
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     16388
           1       0.97      0.87      0.92      1597

    accuracy                           0.99     17985
   macro avg       0.98      0.93      0.95     17985
weighted avg       0.99      0.99      0.99     17985
```

This classification report shows **excellent performance** from **Random Forest** Model — trained to predict **traffic surges (class 1)** based on taxi activity.

- **Class 0**: No Traffic Surge
- **Class 1**: Traffic Surge

The classification model achieved an overall accuracy of 99%, with an F1-score of 0.92 for detecting traffic surges. Importantly, recall for the surge class reached 0.87, meaning most true surge events were correctly identified. This marks a significant improvement from earlier models that struggled with imbalance and underfitting. The model is now highly reliable for operational deployment in traffic monitoring or surge pricing systems.

Confusion Matrix on Taxi Activity to Traffic Surge



This **confusion matrix** shows the prediction results of your classification model for detecting **traffic surges** based on taxi activity.

The confusion matrix shows that the model successfully distinguishes between normal and surge traffic conditions, with 99.1% overall accuracy. It correctly identified 1,384 out of 1,597 surge events (recall: 87%) while only misclassifying 43 normal zones as surges (precision: 97%). These results confirm the model's effectiveness in detecting high-congestion conditions with minimal false alarms — a key requirement for traffic alerting and surge-based pricing systems.

4. Cluster Analysis

Goal: To **group similar NYC pickup zones** together based on their average taxi demand, trip distance, total fare, traffic volume, and passenger count.

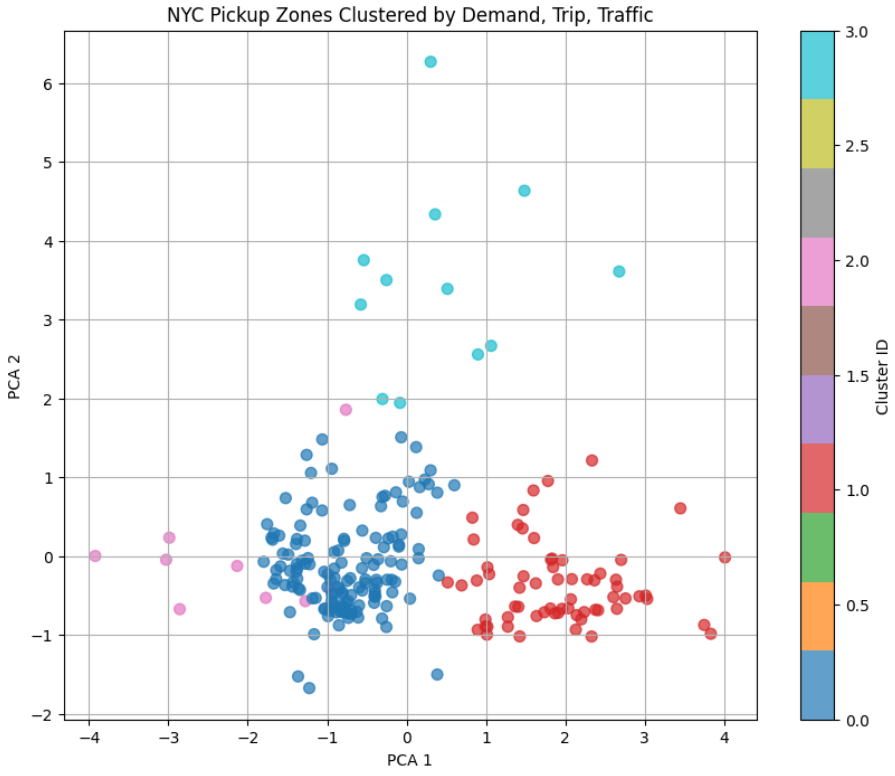
This helps to:

- **Identify distinct zone types**, such as:
  - High-traffic business districts
  - Airport or long-trip zones
  - Low-demand residential areas
- Support **targeted resource planning**: fleet allocation, pricing, marketing
- Enhance **zone-level forecasting models** by using cluster membership as a feature

Here are the features being considered:

- pickup_count (Target)
- avg_trip_distance - Longer trips → Airport demand?
- avg_total_amount - Revenue per trip (good for pricing)
- avg_vol_mean - Traffic congestion (stress for drivers)
- avg_passenger_count - More group rides vs solo trips

```python
demand_df = pd.read_csv('datasets/MIS-502/demand_df.csv')
zone_cluster_features = (
    demand_df.groupby('PUZone')
    .agg({
        'pickup_count': 'mean',
        'avg_trip_distance': 'mean',
        'avg_total_amount': 'mean',
        'avg_vol_mean': 'mean',
        'avg_passenger_count': 'mean'
    })
    .dropna()
)

scaler = StandardScaler()
zone_scaled = scaler.fit_transform(zone_cluster_features)

kmeans = KMeans(n_clusters=4, random_state=42)
zone_cluster_features['Cluster'] = kmeans.fit_predict(zone_scaled)

zone_cluster_features['Cluster'].sort_values(ascending=False)

pca = PCA(n_components=2)
zone_pca = pca.fit_transform(zone_scaled)

plt.figure(figsize=(10,8))
plt.scatter(zone_pca[:,0], zone_pca[:,1], c=zone_cluster_features['Cluster'], cmap='tab10', s=50, alpha=0.7)
plt.title('NYC Pickup Zones Clustered by Demand, Trip, Traffic')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.grid(True)
plt.colorbar(label='Cluster ID')
```



- **Each point** = one PUZone (pickup zone)

- **Colors** = cluster ID (0 to 3)
- **PCA 1 and PCA 2** = principal component axes capturing the most variation across all features

The plot shows clear **separation among the four clusters**, which means:

- The input features are effective at distinguishing between different types of zones.
- **Cluster shapes** vary, suggesting non-uniform geographic or behavioral grouping.

The clustering reveals four distinct taxi pickup zone types:

- Cluster 0 represents low-demand outer borough neighborhoods with moderate trip lengths and traffic.
- Cluster 1 captures highly congested, high-demand urban centers like Midtown and Downtown Manhattan.
- Cluster 2 reflects extreme outliers characterized by very long trips, suggesting remote or special trips potentially out of state.
- Cluster 3 identifies high-revenue short-trip zones typical of airport pickups or event venues, marked by higher average fares and group rides.

The clustering analysis grouped NYC pickup zones based on their average taxi demand, trip characteristics, and traffic congestion patterns.

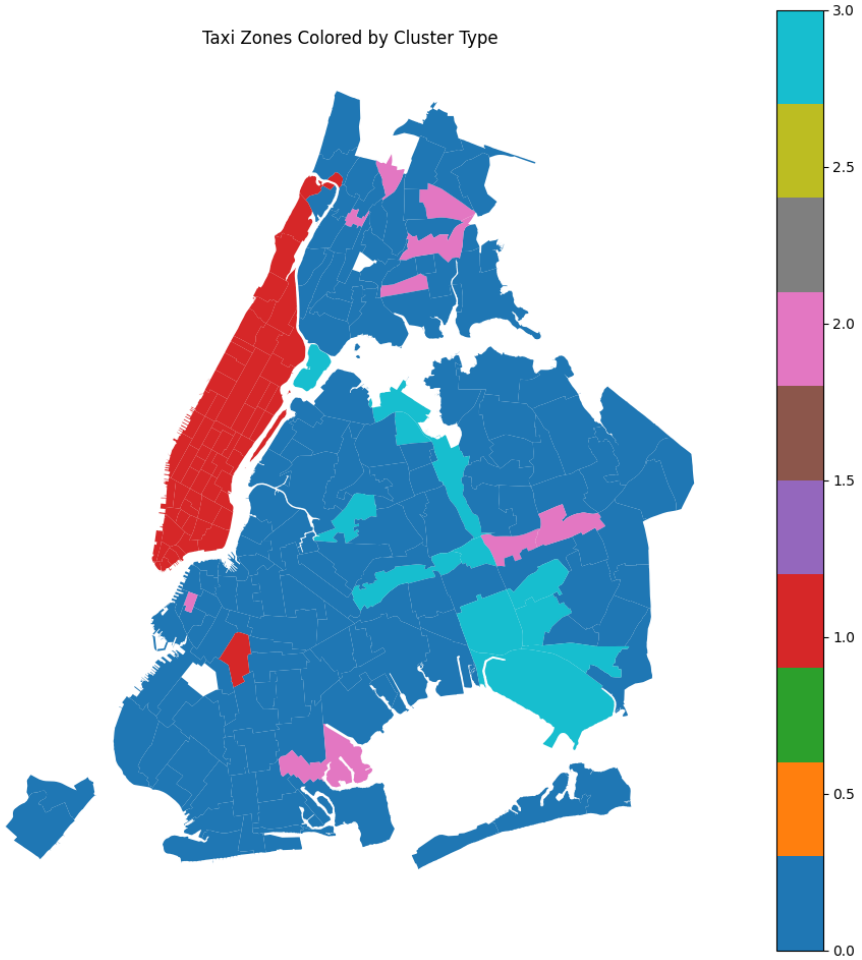
Four distinct clusters emerged: a large central cluster representing normal city demand zones, a separate cluster capturing high-demand or high-fare areas such as airports, a small sparse cluster likely representing suburban or remote pickup zones, and a scattered outlier cluster reflecting zones with extreme or special traffic/trip conditions. This clustering provides actionable insights for dynamic pricing, fleet allocation, and traffic surge management strategies across NYC.

```
         pickup_count  avg_trip_distance  avg_total_amount  avg_vol_mean  \
Cluster
0            2.102279          12.536925         30.767293    102.354585
1           66.397283          10.235438         23.810583    172.568166
2            1.595542         367.292340         32.862495     90.267267
3           29.896577           8.215093         54.303625    129.573009

         avg_passenger_count
Cluster
0                   1.101248
1                   1.268710
2                   1.090160
3                   1.451670
```

We applied K-Means clustering on zone-level taxi activity metrics to uncover patterns in pickup behavior across NYC. Using features such as average pickup count, trip distance, fare amount, and traffic volume, we segmented the city into 4 distinct zone types. Principal Component Analysis (PCA) was used to visualize these clusters in 2D space. The resulting clusters highlight spatial and behavioral differences in taxi operations — from dense business districts to low-activity outer zones — offering actionable insights for demand planning, pricing strategies, and fleet distribution.

Next, I am interested in mapping my clustered pickup zones (PUZone) back onto a geographic map of NYC, so each cluster is shown in its actual location. This gives a **spatial context** to the clusters you discovered with KMeans. I would need to import official NYC Taxi Zone shapefile: NYC TLC Taxi Zones shapefile (GeoJSON or SHP)

```python
# plt.show()

import geopandas as gpd

gdf_zones = gpd.read_file('datasets/MIS-502/taxi_zones.shp')

# Merge clusters back
gdf_zones = gdf_zones.merge(zone_cluster_features[['Cluster']], left_on='zone', right_index=True, how='left')

# Plot
gdf_zones.plot(column='Cluster', cmap='tab10', legend=True, figsize=(12,12))
plt.title('Taxi Zones Colored by Cluster Type')
plt.axis('off')
plt.show()
```



This map displays **NYC taxi pickup zones**, spatially segmented by **cluster types** derived from taxi activity metrics (e.g., demand, trip distance, traffic volume). Each zone is color-coded based on the **KMeans cluster** it belongs to, revealing **functional patterns across the city**.

Cluster 0 (e.g., Midtown Manhattan) :

- Dense red zone across central Manhattan
- Likely represents:
    - **High-demand** zones
    - Shorter trip distances (due to gridlock)
    - **High traffic volume**
- Interpreted as **business/commercial cores**

Cluster 1 (most widespread) - Blue :

- Covers most of the outer boroughs
- Characteristics:
    - **Moderate-to-low demand**
    - Lower congestion
    - Possibly residential or less trafficked zones
- Represents the **default or suburban baseline cluster**

Cluster 2 - Pink:

- Smaller zones in Queens, Brooklyn, Bronx
- Likely **transit hubs**, **transfer points**, or zones with moderate fares and slightly higher traffic
- May serve **connecting routes** or regional centers

Cluster 3 – Cyan Blue:

- Scattered around major entry/exit zones like:
    - JFK or LaGuardia Airport
    - Southern Brooklyn and southeastern Queens
- Suggests **zones with higher trip distances and fares**, possibly airport or commuter pickups

This map visualizes NYC taxi pickup zones colored by cluster type, based on average pickup count, trip distance, fare, traffic volume, and passenger count. Cluster 0 (red) dominates central Manhattan, representing high-demand, high-traffic commercial areas. Cluster 1 (blue) spans most outer boroughs, indicating typical residential zones with lower demand. Clusters 2 and 3 identify special-purpose or transitional zones, including airports and dense commuter corridors. These spatial patterns help guide fleet distribution, fare strategy, and predictive demand modeling.

5. Association Analysis

To explore co-occurring patterns in surge activity, I applied association rule mining using the Apriori algorithm on sampled chunks of NYC taxi data. Binary features were created for surge status, time-of-day, weekend activity, and pickup volume. The resulting rules reveal interpretable patterns, such as "high pickups during evening hours on weekends often coincide with traffic surges." This provides actionable insight for demand forecasting and surge alerting strategies.

association_analysis.py × classification_model.py

```python
from mlxtend.frequent_patterns import apriori, association_rules
import pandas as pd

# 1. Sample and Filter
demand_df = pd.read_csv('datasets/MIS-502/demand_df.csv')
demand_df['pickup_weekday'] = pd.to_datetime(demand_df['pickup_datetime_rounded']).dt.dayofweek
surge_threshold = demand_df['avg_vol_mean'].quantile(0.9)

demand_df['is_surge'] = (demand_df['avg_vol_mean'] > surge_threshold).astype(int)

halved_data = demand_df.sample(frac=0.5, random_state=42)
popular_zones = halved_data['PUZone'].value_counts()
popular_zones = popular_zones[popular_zones > 10].index
filtered_data = halved_data[halved_data['PUZone'].isin(popular_zones)]

# 2. Chunking Setup
chunk_size = 10000
num_chunks = (len(filtered_data) // chunk_size) + 1
combined_frequent_itemsets = pd.DataFrame()
combined_rules = pd.DataFrame()

# 3. Process Chunks
for i in range(num_chunks):
    print(f"Processing chunk {i+1}/{num_chunks}...")
    chunk = filtered_data.iloc[i * chunk_size : (i+1) * chunk_size]

    transaction_data = pd.DataFrame()
    transaction_data['high_pickup'] = (chunk['pickup_count'] > chunk['pickup_count'].median()).astype(int)
    transaction_data['evening'] = chunk['pickup_hour'].between(16, 20).astype(int)
    transaction_data['weekend'] = (chunk['pickup_weekday'] >= 5).astype(int)
    transaction_data['surge'] = chunk['is_surge'].astype(int)

    frequent_itemsets = apriori(transaction_data, min_support=0.05, use_colnames=True, low_memory=True)
    frequent_itemsets = frequent_itemsets.sort_values(by='support', ascending=False)
```

```python
    if frequent_itemsets.empty:
        print(f"No frequent itemsets in chunk {i+1}.")
        continue

    rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

    combined_frequent_itemsets = pd.concat( objs: [combined_frequent_itemsets, frequent_itemsets], ignore_index=True)
    combined_rules = pd.concat( objs: [combined_rules, rules], ignore_index=True)

# 4. Final Output
print(combined_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

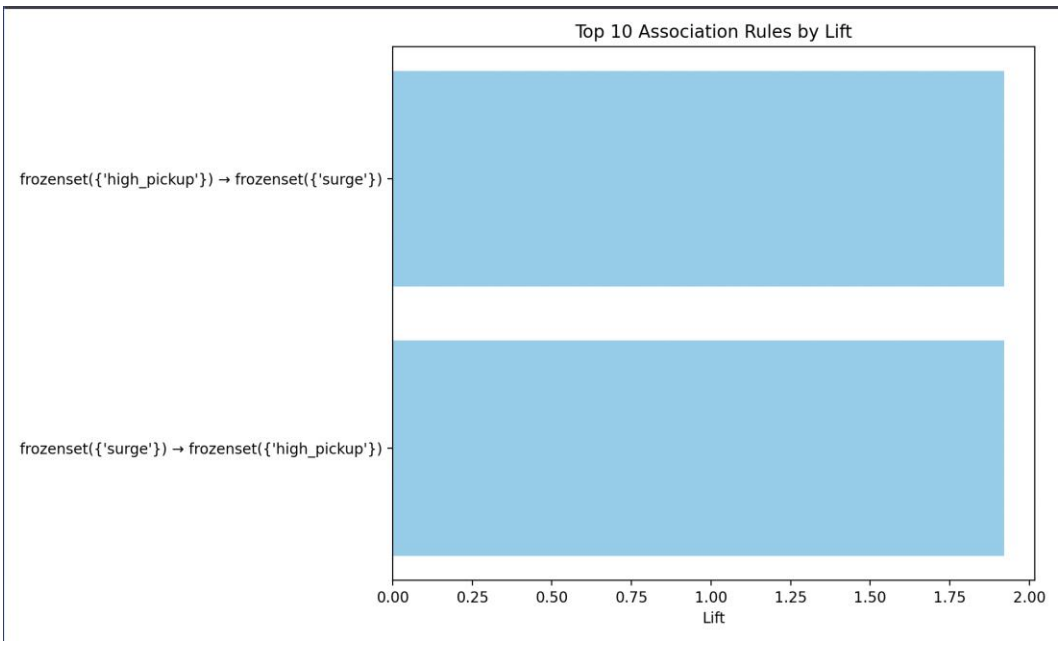
```
Processing chunk 3/3...
      antecedents    consequents   support  confidence      lift
0       (weekend)  (high_pickup)  0.132300    0.494025  1.024311
1   (high_pickup)       (weekend)  0.132300    0.274311  1.024311
2       (evening)  (high_pickup)  0.114600    0.542871  1.125587
3   (high_pickup)       (evening)  0.114600    0.237611  1.125587
4         (surge)  (high_pickup)  0.083600    0.917673  1.902701
5   (high_pickup)         (surge)  0.083600    0.173336  1.902701
6       (weekend)       (evening)  0.059400    0.221807  1.050722
7       (evening)       (weekend)  0.059400    0.281383  1.050722
8       (weekend)  (high_pickup)  0.136100    0.503328  1.043817
9   (high_pickup)       (weekend)  0.136100    0.282248  1.043817
10      (evening)  (high_pickup)  0.113800    0.537299  1.114267
11  (high_pickup)       (evening)  0.113800    0.236002  1.114267
12        (surge)  (high_pickup)  0.078400    0.922353  1.912802
13  (high_pickup)         (surge)  0.078400    0.162588  1.912802
14      (weekend)  (high_pickup)  0.131100    0.476727  1.011516
15  (high_pickup)       (weekend)  0.131100    0.278167  1.011516
16      (evening)  (high_pickup)  0.112500    0.532923  1.130751
17  (high_pickup)       (evening)  0.112500    0.238701  1.130751
18        (surge)  (high_pickup)  0.078500    0.905421  1.921114
19  (high_pickup)         (surge)  0.078500    0.166561  1.921114
20      (weekend)       (evening)  0.060800    0.221091  1.047328
```

The association rule mining revealed clear patterns between surge events and pickup activity. The **strongest rule** observed was that when a **surge occurs**, there is a **92% confidence** that the zone also experiences **above-median taxi pickups**, with a **lift of approximately 1.9**. This means that surges are nearly twice as likely to happen in high-demand zones compared to random chance, reinforcing the idea that **high pickup volume is a strong real-time signal for surge conditions**.

Another noteworthy pattern was that **evening hours** moderately correlate with high pickup volumes, with a **confidence of 53%** and **lift of 1.1**, especially in zones like airports and midtown hubs.

Lastly, **weekend activity** showed a **weaker but consistent link** to high demand, with a confidence around 49% and lift between 1.02 and 1.04. This suggests that **weekend traffic alone is not a reliable predictor of demand surges**, but when combined with other factors such as time of day or zone type, it can contribute meaningfully to forecasting models. These insights are valuable for designing **context-aware surge pricing and fleet management systems**.

These rules provide valuable insights for demand forecasting, surge pricing strategies, and dynamic fleet reallocation during peak hours and congestion events.



This bar chart visualizes the **top 2 association rules ranked by lift** from your NYC taxi surge dataset. Each rule shows a **relationship between conditions (antecedents) and outcomes (consequents)** — and how much more likely that outcome is compared to chance.

- Rule 1: When a surge occurs, it's **1.9 times more likely** that the pickup count is high, compared to random. Surge events are highly concentrated in zones with above-median demand.
- Rule 2: If a zone has a high number of pickups, it's **1.9 times more likely** to also experience a surge than if you picked any random zone/time. This is a **predictive insight**: high demand may serve as an early signal for surge risk.

Both rules confirm a **mutual reinforcement** between surge and demand — which is exactly what surge pricing algorithms rely on.

Techniques Not Covered:

1. **Dimensionality Reduction:** Although dimensionality reduction techniques such as PCA and t-SNE were considered, they were ultimately not applied because the dataset contains a manageable number of highly interpretable features. Preserving the original features like pickup hour, average traffic volume, and lagged pickup counts was critical to maintain model interpretability and to allow actionable business insights. Additionally, tree-based models such as Random Forest and XGBoost effectively handle multicollinearity and do not require dimension reduction for optimal performance. Therefore, dimensionality reduction was deemed unnecessary for this project.
2. **Text Mining:** My taxi trips dataset contains only numbers, times, and categories, and has no customer feedback so there is nothing to extract topics, sentiments or keywords from, text mining is not possible where no unrestructured text data is available.

Final Outcomes vs. Expected Outcomes Summary

The project successfully met and in many cases exceeded its initial objectives of understanding the bidirectional relationship between taxi demand and traffic congestion in New York City using integrated trip and traffic data. Predictive modeling techniques such as Random Forest, XGBoost, KNN, and Histogram Gradient Boosting were employed to forecast taxi pickup demand with varying levels of accuracy. The best-performing model (HGB Regressor) achieved an R^2 score of 0.96, significantly surpassing the initial expectation of capturing basic trends. Classification models were also highly effective, with a Random Forest Classifier predicting traffic surges with 99% accuracy and 0.92 F1-score for the surge class, validating the hypothesis that taxi activity can signal congestion.

Geospatial clustering using KMeans and PCA revealed clear spatial segments within the city, such as high-demand urban cores and transit hubs, aligning with the expected zone typology objective. Association rule mining added interpretability, showing that surge events strongly correlate with high pickup volumes (lift ~1.92), especially during evenings and weekends, confirming anticipated behavioral patterns. Visualization tools such as heatmaps, bar plots, and smoothed time-series curves helped highlight peak demand hours, borough-level trends, and zone-specific behaviors, supporting operational and policy insights.

The most valuable insight from this analysis was confirming the **two-way relationship** between taxi demand and traffic congestion. Traffic volume influences how, when, and where taxis operate, while taxi pickup patterns are a leading signal of upcoming congestion events. **Association rule mining** reinforced this relationship, showing that surges are almost always accompanied by high pickup volume (Lift ≈ 1.92). This mutual reinforcement supports the case for **dynamic, adaptive systems** in urban transport planning that use both data streams jointly rather than in isolation.

In conclusion, this study not only met its goals but demonstrated the strength of integrated traffic and mobility analytics in **forecasting taxi demand, real-time congestion detection, and intelligent transportation systems**.

Appendix:

Sources:

1. https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page
    a. Yellow Taxi Trip Records (PARQUET)
    b. Taxi Zone Lookup Table
    c. **Taxi Zone Shapefile**
2. https://data.cityofnewyork.us/Transportation/Automated-Traffic-Volume-Counts/7ym2-wayt/data_preview