Report: Optimising NYC Taxi Operations

Include your visualisations, analysis, results, insights, and outcomes. Explain your methodology and approach to the tasks. Add your conclusions to the sections.

## Data Preparation

* 1. Loading the dataset
  2. df = pd.read\_parquet('C:\\Required\\UPGRAD\\Assignment1\\Datasets and Dictionary\\trip\_records\\2023-1.parquet')
  3. df.info()
     1. **Sample the data and combine the files**
     2. # Take a small percentage of entries from each hour of every date.
     3. # Iterating through the monthly data:
     4. # read a month file -> day -> hour: append sampled data -> move to next hour -> move to next day after 24 hours -> move to next month file
     5. # Create a single dataframe for the year combining all the monthly data
     6. # Select the folder having data files
     7. import os
     8. # Select the folder having data files
     9. os.chdir('C:\\Required\\UPGRAD\\Assignment1\\Datasets and Dictionary\\trip\_records')
     10. # Create a list of all the twelve files to read
     11. file\_list = os.listdir()
     12. #print(file\_list)
     13. # initialise an empty dataframe
     14. df = pd.DataFrame()
     15. n=0
     16. # iterate through the list of files and sample one by one:
     17. for file\_name in file\_list:
     18. try:
     19. # file path for the current file
     20. file\_path = os.path.join(os.getcwd(), file\_name)
     21. print(file\_path)
     22. # Reading the current file
     23. df\_file = pd.read\_parquet(file\_path)
     24. #print(df\_file.head())
     25. df\_file['date'] = df\_file['tpep\_pickup\_datetime'].dt.date
     26. df\_file['hour'] = df\_file['tpep\_pickup\_datetime'].dt.hour
     27. #print(df\_file.head())
     28. unique\_dates = sorted(df\_file['date'].unique())
     29. print(unique\_dates)
     30. #print(df\_file.head())
     31. # We will store the sampled data for the current date in this df by appending the sampled data from each hour to this
     32. # After completing iteration through each date, we will append this data to the final dataframe.
     33. sampled\_data = pd.DataFrame()
     34. # Loop through dates and then loop through every hour of each date
     35. for date in unique\_dates:
     36. print(date)
     37. for hour in range(24):
     38. hour\_data = df\_file[(df\_file['date'] == date) & (df\_file['hour'] == hour)]
     39. #print("===========")
     40. #print(hour\_data)
     42. # Iterate through each hour of the selected date
     43. # Sample 5% of the hourly data randomly
     44. sample = hour\_data.sample(frac = 0.01, random\_state = 42)
     45. #print(sample)
     46. # add data of this hour to the dataframe
     47. sampled\_data = pd.concat([sampled\_data, sample]) # adding data for this hour to the DF
     49. # Concatenate the sampled data of all the dates to a single dataframe
     50. df = pd.concat([df, sampled\_data], ignore\_index=True)
     51. print(n+1)
     52. # Optional: Check the result
     53. #print(df.shape)
     54. #print(df['date'].value\_counts())# we initialised this empty DF earlier
     55. except Exception as e:
     56. print(f"Error reading file {file\_name}: {e}")

## Data Cleaning

### Fixing Columns

* + 1. **Fix the index**

# Fix the index and drop any columns that are not needed

df.reset\_index(drop = True, inplace=True)

df.drop('store\_and\_fwd\_flag',axis=1, inplace=True)

df.drop('congestion\_surcharge',axis=1, inplace=True)

df

* + 1. **Combine the two airport\_fee columns**

*df['combined\_airport\_fee'] = df[['airport\_fee', 'Airport\_fee']].sum(axis=1, skipna=True)*

*df.drop('airport\_fee',axis=1, inplace=True)*

*df.drop('Airport\_fee',axis=1, inplace=True)*

### Handling Missing Values

* + 1. **Find the proportion of missing values in each column**

missing\_proportion = df.isnull().mean()

missing\_proportion

* + 1. **Handling missing values in passenger\_count**

# Display the rows with null values

null\_rows = df[df.isnull().any(axis=1)]

print(null\_rows)

# Impute NaN values in 'passenger\_count'

median\_val = df['passenger\_count'].median()

df['passenger\_count'].fillna(median\_val, inplace=True)

* + 1. **Handle missing values in RatecodeID**

*df['RatecodeID'].fillna(0, inplace=True)*

* + 1. **Impute NaN in congestion\_surcharge**

**df['congestion\_surcharge'].fillna(0, inplace=True)**

### Handling Outliers and Standardising Values

* + 1. **Check outliers in payment type, trip distance and tip amount columns**

**Q1 = df['passenger\_count'].quantile(0.25)**

**Q3 = df['passenger\_count'].quantile(0.75)**

**IQR = Q3 - Q1**

**# outlier bounds**

**lower\_bound = Q1 - 1.5 \* IQR**

**upper\_bound = Q3 + 1.5 \* IQR**

**print(lower\_bound)**

**print(upper\_bound)**

**# outliers**

**pass\_cnt\_outliers = df[(df['passenger\_count'] < lower\_bound) | (df['passenger\_count'] > upper\_bound)]**

**print(pass\_cnt\_outliers[['fare\_amount', 'trip\_distance', 'passenger\_count', 'total\_amount','payment\_type']])**

## Exploratory Data Analysis

### General EDA: Finding Patterns and Trends

* + 1. **Classify variables into categorical and numerical**

1. VendorID:Categorical
2. tpep\_pickup\_datetime:Categorical
3. tpep\_dropoff\_datetime:Categorical
4. passenger\_count:Numerical
5. trip\_distance:Numerical
6. RatecodeID:Numerical
7. PULocationID:Numerical
8. DOLocationID:Numerical
9. payment\_type:Numerical
10. pickup\_hour:Numerical
11. trip\_duration:Numerical
12. fare\_amountNumerical
13. extraNumerical
14. mta\_taxNumerical
15. tip\_amountNumerical
16. tolls\_amountNumerical
17. improvement\_surchargeNumerical
18. total\_amountNumerical
19. congestion\_surchargeNumerical
20. airport\_feeNumerical

* + 1. **Analyse the distribution of taxi pickups by hours, days of the week, and months**

**A graph with blue dots

AI-generated content may be incorrect.**A graph with green bars

AI-generated content may be incorrect.

A graph with blue and white lines

AI-generated content may be incorrect.

* + 1. **Filter out the zero/negative values in fares, distance and tips**
  1. **A screenshot of a computer

     AI-generated content may be incorrect.**
     1. **Analyse the monthly revenue trends**

**A graph with numbers and letters

AI-generated content may be incorrect.**

* + 1. **Find the proportion of each quarter’s revenue in the yearly revenue**
  1. **A screenshot of a graph

     AI-generated content may be incorrect.**
     1. **Analyse and visualise the relationship between distance and fare amount**
     2. **Analyse the relationship between fare/tips and trips/passengers**

**A screen shot of a graph

AI-generated content may be incorrect.  
A screen shot of a graph

AI-generated content may be incorrect.**

* + 1. **Analyse the distribution of different payment types**
  1. **A screen shot of a graph

     AI-generated content may be incorrect.**
     1. **Load the taxi zones shapefile and display it  
        A screenshot of a computer

        AI-generated content may be incorrect.**
     2. **Merge the zone data with trips data**

**A screenshot of a computer

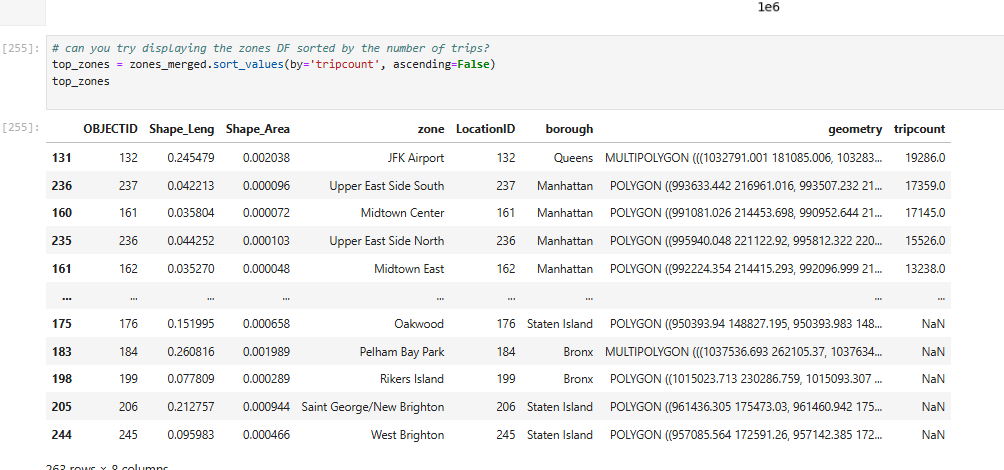
AI-generated content may be incorrect.**

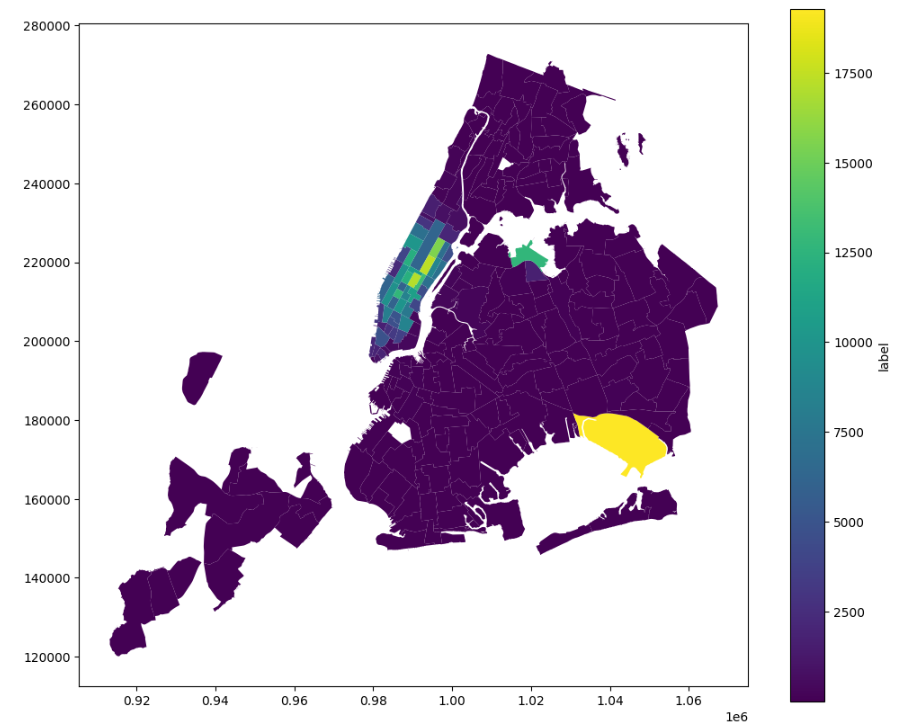
* + 1. **Find the number of trips for each zone/location ID**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* + 1. **Add the number of trips for each zone to the zones dataframe**

****

* + 1. **Plot a map of the zones showing number of trips**
    2. ****
    3. **Conclude with results**

### Detailed EDA: Insights and Strategies

* + 1. **Identify slow routes by comparing average speeds on different routes**

df\_new['speed'] = df\_new['trip\_distance'] / df\_new['tripduration']

route\_spd = df\_new.groupby(['hour', 'PULocationID', 'DOLocationID'])['speed'].mean().reset\_index()

route\_spd

**slowroutes = route\_spd.sort\_values(['hour', 'speed']).groupby('hour').first().reset\_index()**

**slowroutes**

* + 1. **Calculate the hourly number of trips and identify the busy hours**

**hour\_cnts = df\_new['hour'].value\_counts().sort\_index()**

**hour\_cnts.plot(kind='bar', color='blue')**

**A graph of a number of bars

AI-generated content may be incorrect.**

* + 1. **Scale up the number of trips from above to find the actual number of trips**

**# Fill in the value of your sampling fraction and use that to scale up the numbers**

**sample\_fraction = 0.01**

**hour\_counts = df['hour'].value\_counts().sort\_index()**

**scale\_hour\_counts = hour\_counts / sample\_fraction**

**busiest\_hour = scale\_hour\_counts.idxmax()**

**busiest\_count = int(scale\_hour\_counts.max())**

**print(scale\_hour\_counts)**

* + 1. **Compare hourly traffic on weekdays and weekends**

**df\_new['week'] = df\_new['day\_of\_week'].apply(lambda x: 'Weekend' if x >= 5 else 'Weekday')**

**traffic\_hr = df\_new.groupby(['hour', 'week']).size().reset\_index(name='trpcnt')**

**traffic\_hr.plot(kind='line', marker='o')**

* + 1. **Identify the top 10 zones with high hourly pickups and drops**

**pickup = df\_new.groupby('PULocationID').size().reset\_index(name='pickupcount')**

**pickup = pickup.merge(zones, left\_on='PULocationID', right\_on='LocationID', how='left')**

**pickup10 = pickup[['zone', 'borough', 'pickupcount']].sort\_values(by='pickupcount', ascending=False).head(10)**

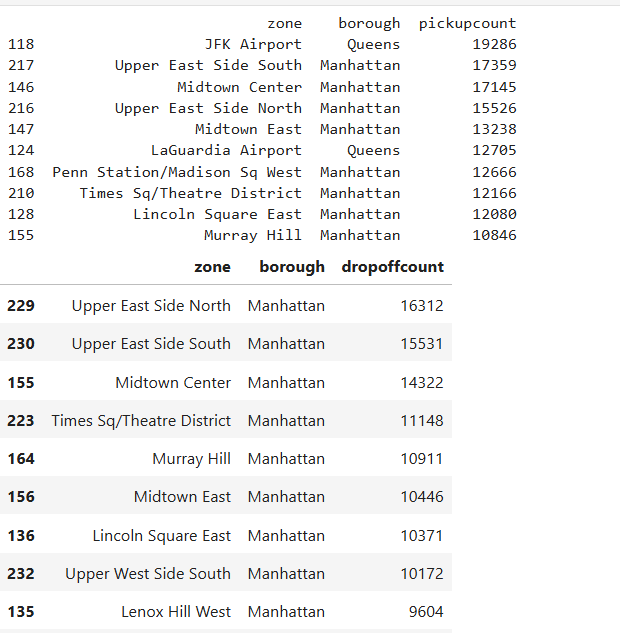
**print(pickup10)**

**# Top 10 dropoff zones**

**dropoff = df\_new.groupby('DOLocationID').size().reset\_index(name='dropoffcount')**

**dropoff = dropoff.merge(zones, left\_on='DOLocationID', right\_on='LocationID', how='left')**

**dropoff10 = dropoff[['zone', 'borough', 'dropoffcount']].sort\_values(by='dropoffcount', ascending=False).head(10)**

****

**dropoff10**

* + 1. **Find the ratio of pickups and dropoffs in each zone**
    2. **Identify the top zones with high traffic during night hours**
    3. **Find the revenue share for nighttime and daytime hours**
    4. **For the different passenger counts, find the average fare per mile per passenger**
    5. **Find the average fare per mile by hours of the day and by days of the week**
    6. **Analyse the average fare per mile for the different vendors**
    7. **Compare the fare rates of different vendors in a distance-tiered fashion**
    8. **Analyse the tip percentages**
    9. **Analyse the trends in passenger count**
    10. **Analyse the variation of passenger counts across zones**
    11. **Analyse the pickup/dropoff zones or times when extra charges are applied more frequently.**

## Conclusions

### Final Insights and Recommendations

* + 1. **Recommendations to optimize routing and dispatching based on demand patterns and operational inefficiencies.**

Increase fleet availability in core areas like Midtown, JFK, and East Village during peaks.

Use historical data to inform predictive relocation of taxis between under- and over-served areas.

Consider fare adjustments or incentives for shared rides on slow routes to reduce congestion and improve vehicle utilization.

* + 1. **Suggestions on strategically positioning cabs across different zones to make best use of insights uncovered by analysing trip trends across time, days and months.**

**Morning peak (7–10 AM): Strong demand in residential**

**Evening peak (4–8 PM): Demand in commercial area**

**Weekdays: Regular commuting patterns.**

**Weekends: Later demand start, higher activity near parks, shopping, nightlife, airports.**

* + 1. **Propose data-driven adjustments to the pricing strategy to maximize revenue while maintaining competitive rates with other vendors.**

**Introduce surge pricing during high-demand windows**

**Use historical hourly trends to set prices**

**Implement a minimum fare threshold during busy times**

**Use this to simulate revenue impact of 5–10% fare changes during specific periods or routes.**