Project: New York City Taxi Trip Duration

Machine Learning

Professor:
Dirk VALKENBORG

Eleftherios Kokkinis

August 11, 2024

Contents

1 Data Description

I have two datasets at my disposal,n`train.csv` and `test.csv`.

The `train.csv` file contains 1,458,644 records of individual taxi trips, which is what I am going to use to train my models. The `test.csv` contains 625,134 records, and my objective is to predict each record's trip duration. Each record also includes the following features:

- **id**: A unique identifier for each trip.
- **vendor_id**: A code indicating the provider associated with the trip. There are two providers, represented by `1` and `2`.
- **pickup_datetime**: The timestamp when the passenger(s) were picked up.
- **dropoff_datetime**: The timestamp when the passenger(s) were dropped off.
- **passenger_count**: The number of passengers on the trip, ranging from 1 to 6.
- **pickup_longitude**: The longitude coordinate of the pickup location.
- **pickup_latitude**: The latitude coordinate of the pickup location.
- **dropoff_longitude**: The longitude coordinate of the dropoff location.
- **dropoff_latitude**: The latitude coordinate of the dropoff location.
- **store_and_fwd_flag**: A binary flag indicating whether the trip record was held in the taxi's memory before being sent to the vendor due to lack of connectivity (`Y` for yes, `N` for no).
- **trip_duration**: The target variable, representing the duration of the trip in seconds. This is the variable that I aim to predict with my model.

2 Short Exploratory Data Analysis

I'll start by getting a better idea of the data features that I am going to work with. Identifying their structure, distribution, classes and potential outliers.

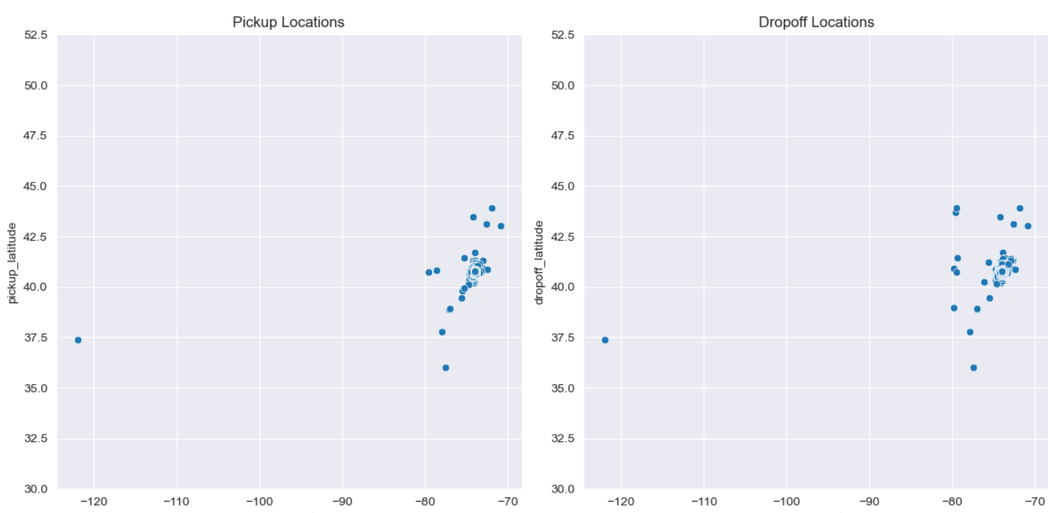

Figure 1: Distribution of Pickup and Dropoff Points

It appears that the majority of pickup and dropoff locations are limited within the coordinates of NYC and its surrounding area.

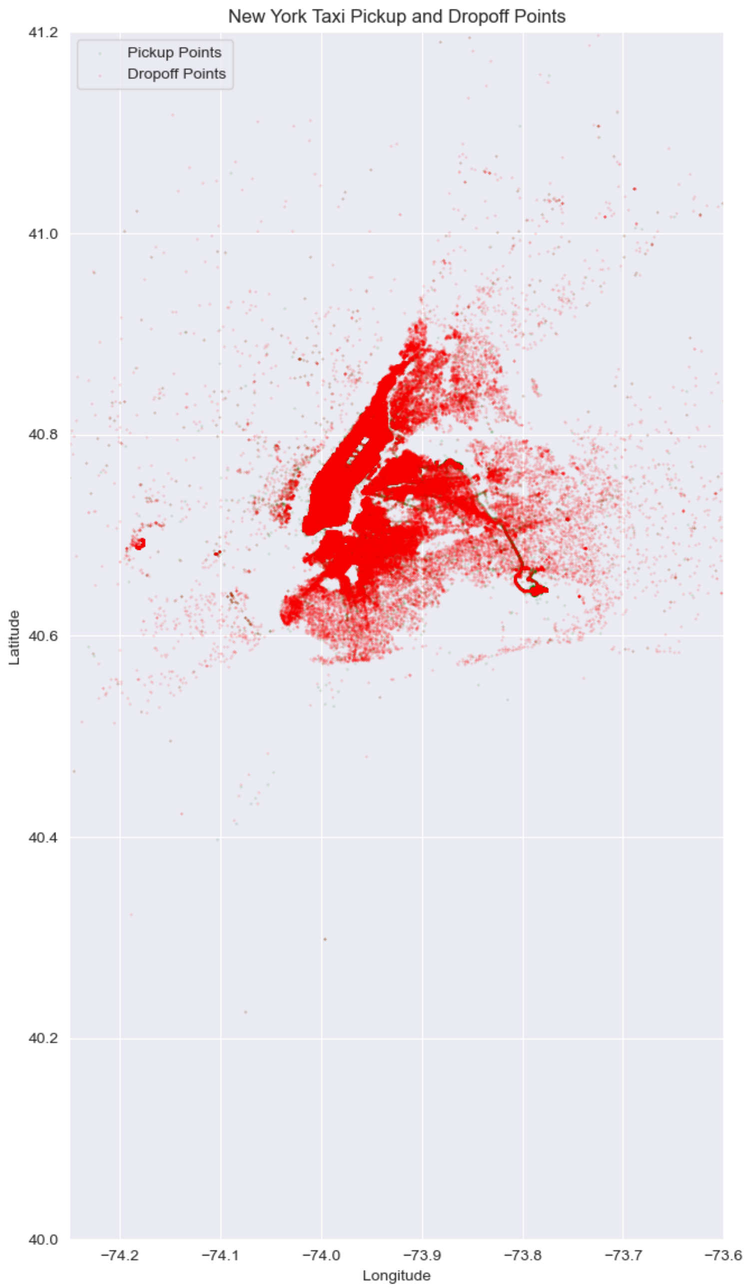

Figure 2: Visualizing Pickup and Dropoff points around NYC's coordinates

By limiting the coordinates range around NYC, I can clearly see the bulk of the pickup and dropoff points of the taxi trips whom density is able to give form to NYC's urban planning on the map.

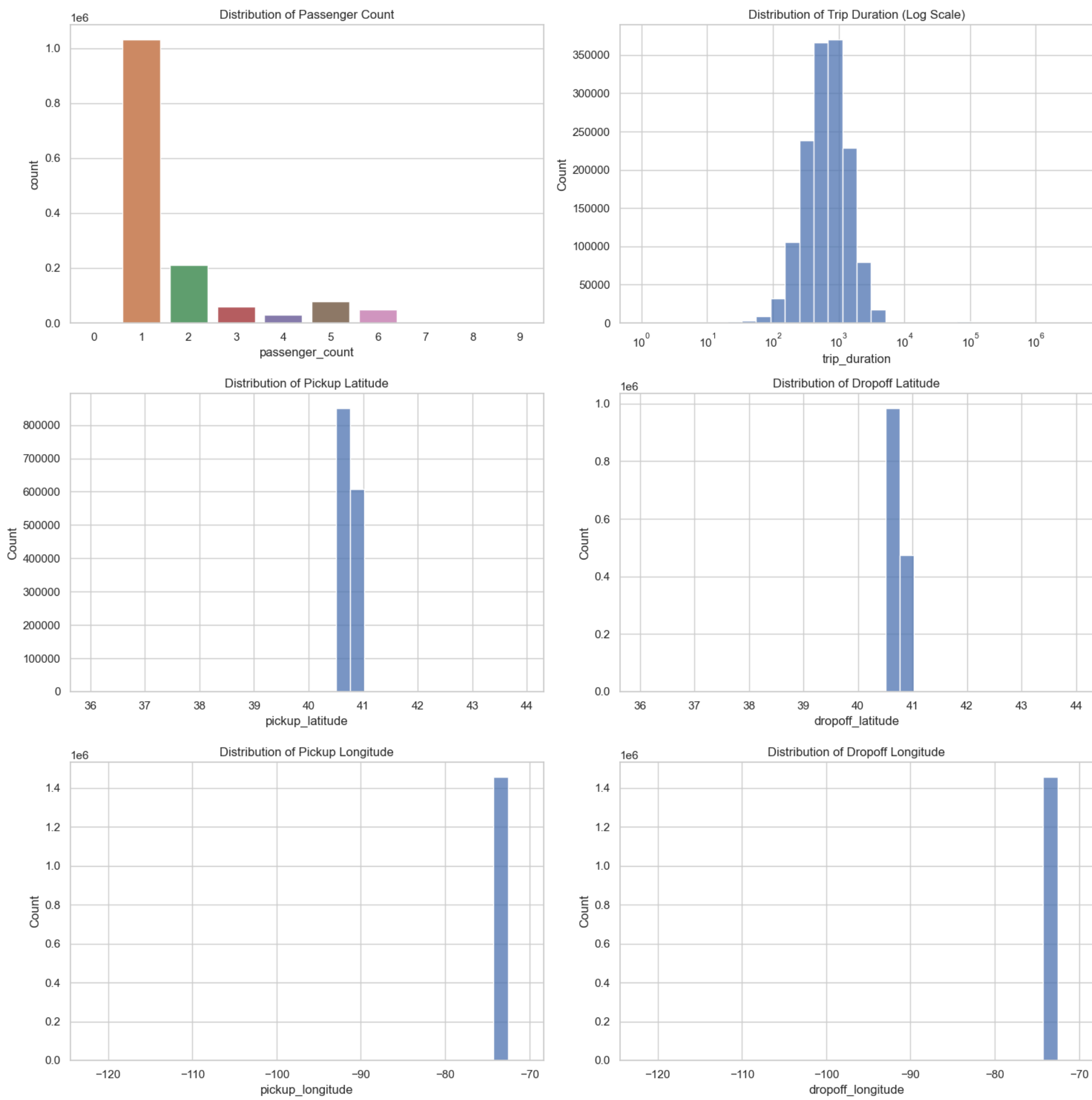

Figure 3: Visual Exploration of my features

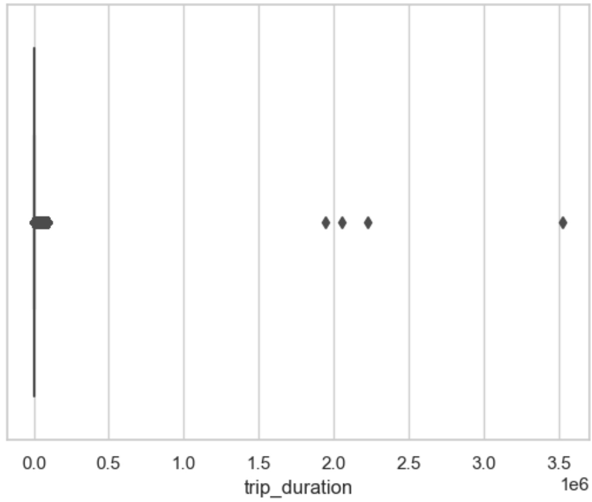

Figure 4: Trip Durations

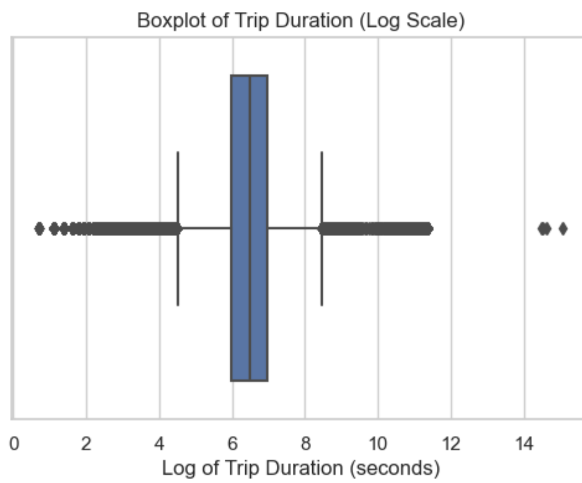

Figure 5: Log Trip Durations

Passenger count consists of 6 classes, most taxi trips had 1 passenger. A couple of extreme outliers also appear to exist in the trip durations. The counts of longitudes and latitutes tell the same story as before, with the overwhelming majority of them being in a restricted geographical location around NYC.

3 Feature Engineering and Data Enrichment

Considering the complexity of the question at hand, to predict taxi trip durations, it appears very unlikely that the currently available features can tell the whole story.

3.1 Temporal Features and Google's Distance Matrix API

First I am going to create temportal features by extracting the month, day, hour and minute from the datetime column.

Google offers a great variety of APIs when it comes to maps, routes, geolocations and geocoding, which I am going to make use of. It is a well known fact that taxi companies make use of GPS services, and Google's route optimizers and directions are probably the most widely used ones.

I will particularly use the Distance Matrix API to calculate the distances and travel times from pickup to dropoff points and consequently create 2 new features in my dataset.

Even if the GPS services used by NYC's taxis are not powered by Google, I expect that would still be good approximation of any other GPS service that the taxi drivers may have used.
The API takes into consideration NYC's street network as well as historical traffic conditions.

Let's plot the correlation to have a better look into the possible relations between my data's features as well the new ones.

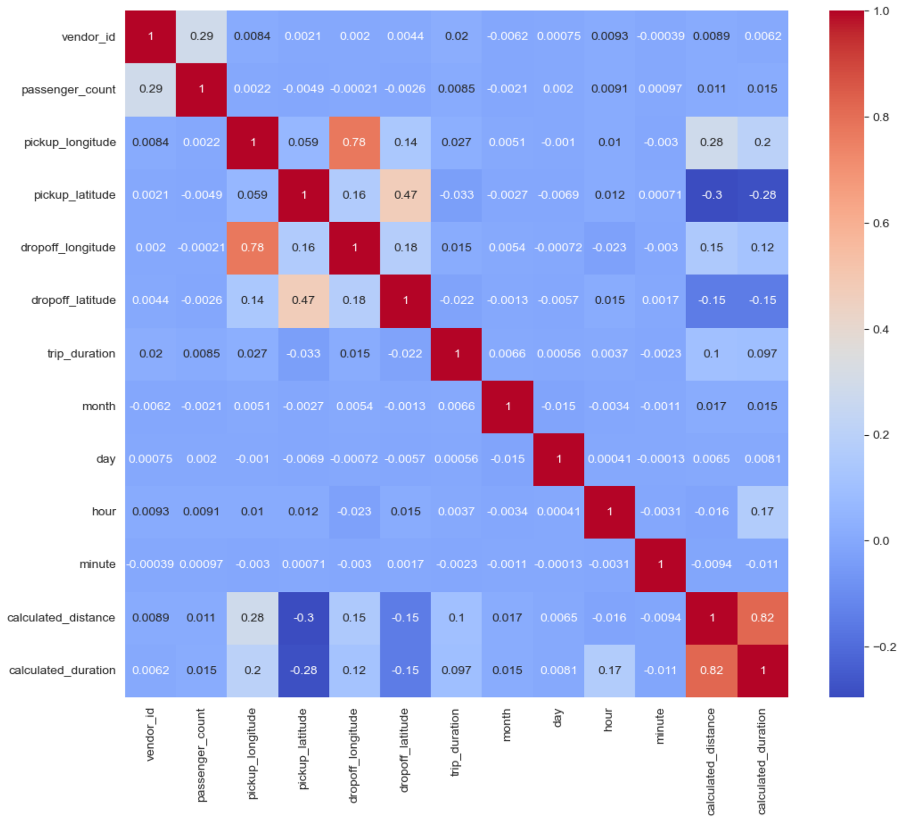

Figure 6: Correlation heatmap

The trip_duration shows relatively weak correlations with most other features, indicating that no single feature strongly predicts the trip duration on its own. However my 2 newly engineered features still show the highest correlation with my target variable (about 10 percent).
However, since Google's calculated_duration is derived from the distance, they also show a very high correlation between them and seem to convey about the same information. I wish however to keep the variable with the intention of engineering further features related to speed.

3.2 Weather Data - Holidays

I will enrich my data using publicly available weather data for the time period available in my dataset in NYC. That includes features like temperature, percipitation and snow indicators.

I am also making use of USFederalHolidayCalendar(), creating a new feature that indicates which days are public holidays. Intuition tells us that public holidays may have an effect on variables like potential road traffic and consequently taxi trip durations. I include weekends as holidays as well.

3.3 Google's Directions API

Next, I decided to calculate the possible taxi routes' polylines using the pickup and dropoff longitudes and latitudes as points of origins and destinations, and the timestamps as departure times. The departure times are important for the Direction's API to calculate the most optimal route based on averaged traffic conditions of the past (since it's not being used in real-time).

—c—c—

Index	Polyline
0	[(40.76781, -73.98228), (40.76776, -73.9822), ...]
1	[(40.73856, -73.98045), (40.73253, -73.9848), ...]
2	[(40.76415, -73.97887), (40.76304, -73.97622), ...]
3	[(40.71998, -74.01015), (40.71811, -74.01048), ...]
4	[(40.79319, -73.97301), (40.79036, -73.97506), ...]

The taxi routes polylines were then decoded into coordinates and used to create a new distance column extracted from the decoded polyline data, Since the polyline routes use Google's directions API, traffic data are incorporated to the suggested route.

Additionally I will create a route_points column and a new route_length, which can be an indicator of the complexity of the route. A higher number of points might suggest a more complex route with more turns or changes in direction, potentially affecting the trip duration.
The num points are basically the number of coordinates in each decoded polyline list (the number of tuples). While the route length was calculated using geopy's.distance, great_circle function.

3.4 Average Speed feature

I will proceed by creating 2 distinct average speed columns. Using the calculated_distance from the Distance Matrix API, and the route_length that is calculated from the polylines. Both of them will use the calculated_duration as time.

3.5 Discretized Directions

I will also create a feature about Direction by taking the differences between latitudes and longitudes. That is going to indicate whether the direction of the taxi is NW, NE, SW, SE, NS, SN, WE, EW or stationary.

3.6 Feature Aggregation

Beyond these changes, a whole lot of new features were manufactured by aggregating taxi trips based on a number of already existing features like average speed per time/cluster/direction etc, and aggregated counts of trips. The reasoning behind this decision was that potential underlying patterns (which are based on these features) may be unveiled and used during modelling.

4 Visualizing NYC's zones, boroughs and districts districts

Understanding the data's distribution on an actual map may be helpful if I am to use the city's planning and urban characteristics as features in my model.
For that purpose I made use of 2 new publicly available datasets about the zoning and districts of NYC.

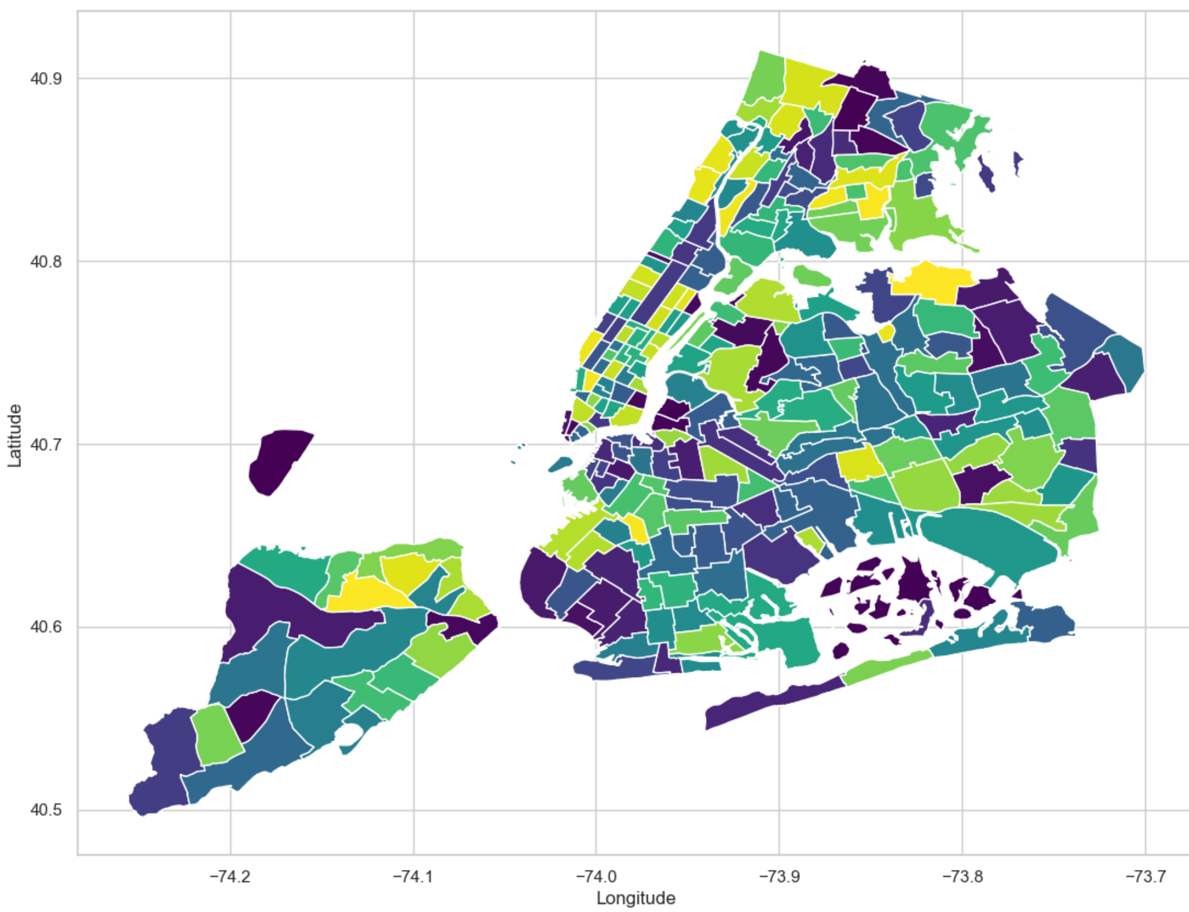

Figure 7: NYC's zones on Latitudes and Longitudes

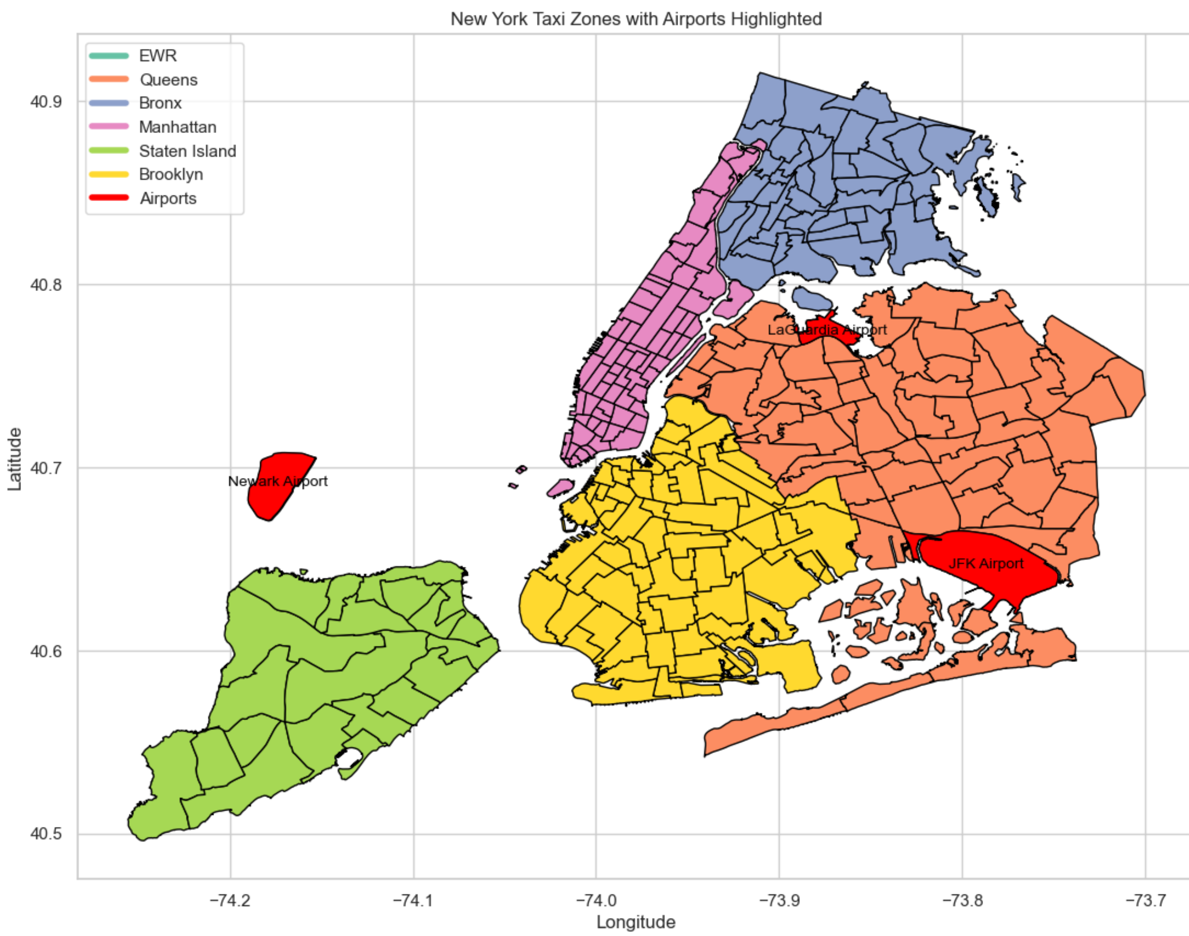

Figure 8: NYC's boroughs on Latitudes and Longitudes

First of all I can clearly see that the coordinates of the city range from about (-74.25,-73.65) in longitude and (40,41.2) in latitude. Then the city is divided into 5 boroughs which are further divided into smaller districts. Three airports also exist, which explain the clustered pickup/dropoff points in the map before.

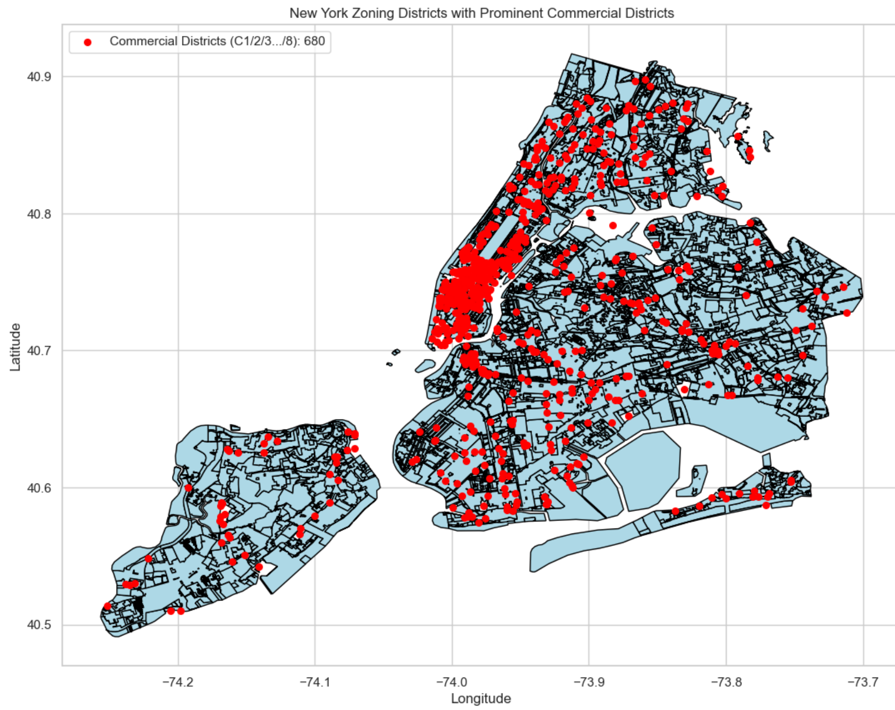

Figure 9: NYC's commercial districts

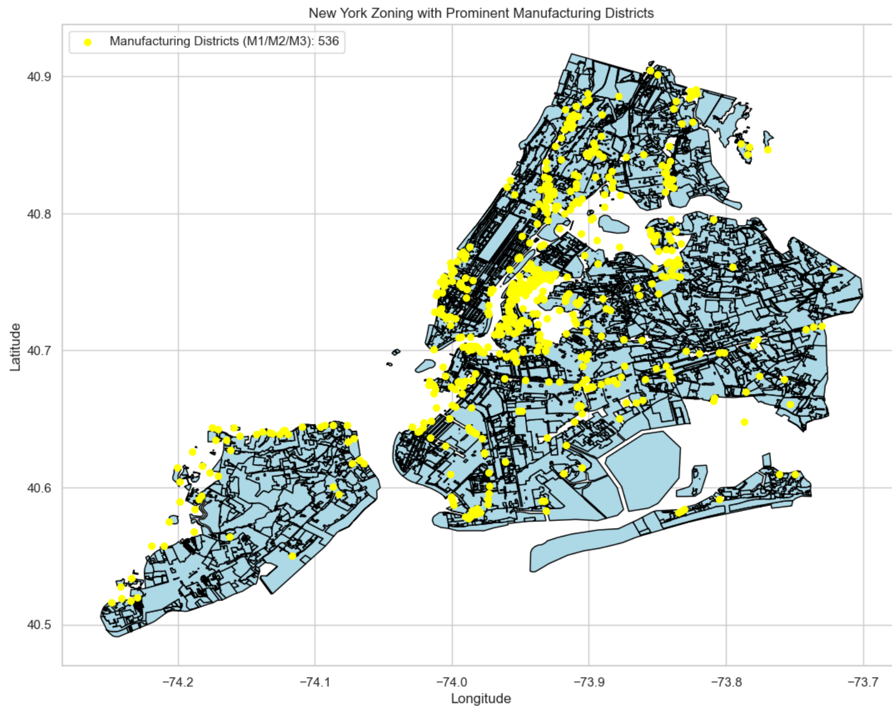

Figure 10: NYC's Manufacturing districts

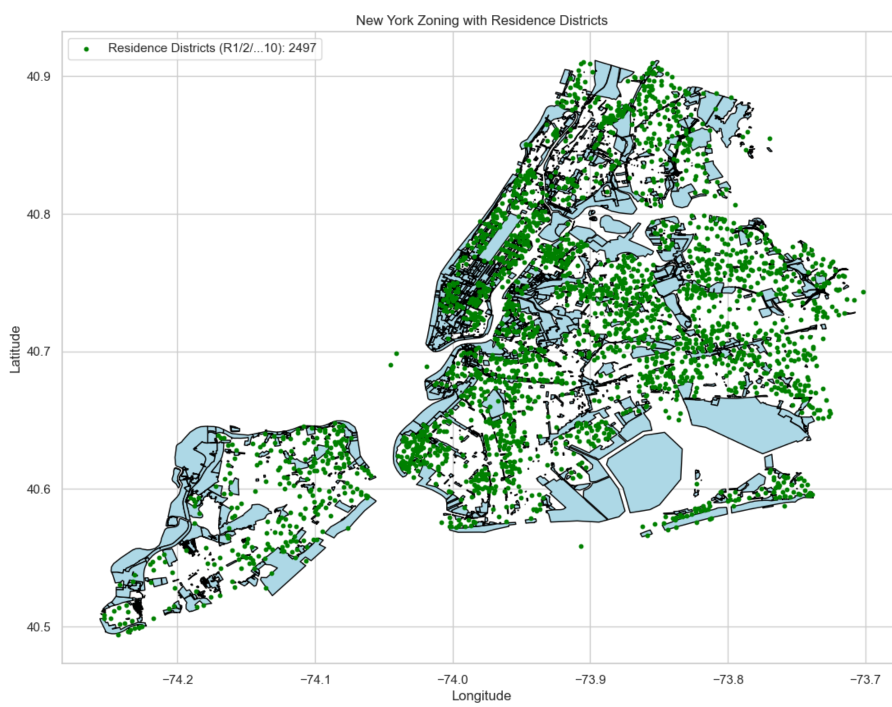

Figure 11: NYC's Residential districts

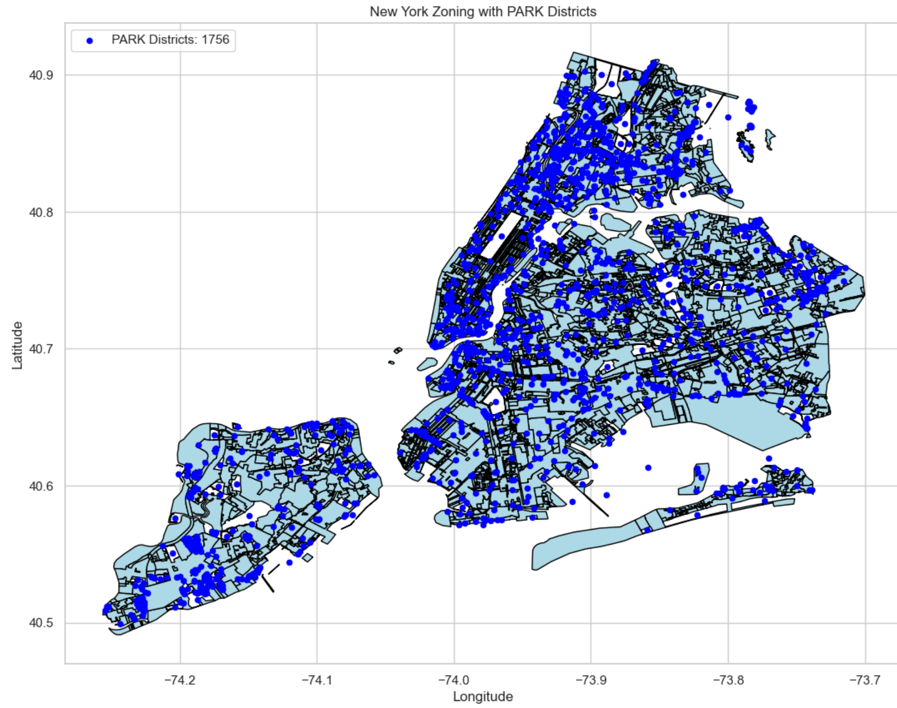

Figure 12: NYC's PARK districts

Manhattan and parts of Queens and Brooklyn appear to be the busiest areas. However we can also spot smaller clusters around the airport locations. My goal is now to combine my taxi data with the zones by mapping each pickup and dropoff point to their respective boroughs, zones and districts. I will do that by converting the geom column from WKT format to a geometry object and then the taxi data into a geodataframe. I will do a spatial join and then merge back to the original df. I will also print the number of trips without a borough/zone. I will now indicate the pick up and dropoff points that are in commercial districts. Commercial districts are the ones in the ZONEDIST column that are named C1/2/..8. I will filter out these districts and then do spatial matching with the pick up and drop off coordinates in the taxi dta. 1 = is in commercial district. 0 = it's not in commercial district

Decoded Polyline	**Pickup Zone**	**Pickup Borough**	**Dropoff Zone**
`[(40.76781, -73.98228), (40.76776, -73.9822), ...]`	Lincoln Square East	Manhattan	Upper East Side South
`[(40.73856, -73.98045), (40.73253, -73.9848), ...]`	Kips Bay	Manhattan	Greenwich Village South
`[(40.76415, -73.97887), (40.76304, -73.97622), ...]`	Midtown North	Manhattan	Seaport
`[(40.71998, -74.01015), (40.71811, -74.01048), ...]`	TriBeCa/Civic Center	Manhattan	Financial District South
`[(40.79319, -73.97301), (40.79036, -73.97506), ...]`	Upper West Side North	Manhattan	Upper West Side South

Table 1: Dataset Structure Showing Polylines, Pickup, and Dropoff Details

Things to note: I have 1102 trips where their pickup point was not within the city's zones. 3723 trips where their dropoff point was not within the city's zones and 846 trips where neither pickup or dropoff point was. I will now indicate the pick up and dropoff points that are in each district (commercial/residential/manufacturing). I will filter out these districts and then do spatial matching with the pick up and drop off coordinates in the taxi dta. 1 = is in commercial district. 0 = it's not in commercial district.

Therefore I have created as new features for each taxi trip, the pickup and dropoff borough regions, as well as whether it's a commercial, manufacturing etc district.

5 Iterative Modelling - Starting with a simple XGBoost model

I will apply iterative modelling with XGboost. I show in the beginning that the correlation heatmap showed no significant correlation of the trip_duration with the other variables which implied that no linear relationship between them exists, thus I am starting to try directly tree based algorithms like Boosting. Starting from a simple and moving towards a gradually more complex model. More aggregated features will be created or dropped based on the feature importances during iterations.

My first and simplest model included the following features:

Index	**Variable Name**
1	id
2	vendor_id
3	pickup_datetime
4	dropoff_datetime
5	passenger_count
6	pickup_longitude
7	pickup_latitude
8	dropoff_longitude
9	dropoff_latitude
10	trip_duration

Table 2: Variables in the Simple Form of the Dataset

5.0.1 Hyperparameter optimization

As with the every following iteration from now on, I am splitting my training dataset to training and validation, 80% of the data are retained for training and 20% for validation. I make use of optuna framework for hyperparameter optimization using the validation set.

For every model, I used optuna to optimize the following parameters:

Parameter	Description
`objective`	Specifies the learning task and theobjective, here my goal is to minimize the squared difference
`colsample_bytree`	Fraction of features used by the model at each tree node, in an effort to prevent overfitting
`tree_method`	The tree construction algorithm used in XGBoost
`reg_lambda`	L2 regularization term on weights, controls model complexity.
`learning_rate`	Controls how much to change the model in response to errors.
`max_depth`	Maximum depth of a tree, to control overfitting.
`subsample`	Fraction of samples used for training each tree, to avoid overfitting.
`alpha`	L1 regularization term on weights
`gamma`	Minimum loss reduction required
`n_estimators`	Number of boosting rounds or trees in the model.
`min_child_weight`	Minimum sum of instance weight (hessian) needed in a child node.
`eval_metric`	Metric used to evaluate model performance during training (e.g., RMSE).

Table 3: Parameters Used in the XGBoost Model's Objective Function

5.0.2 Results of the Simple Model

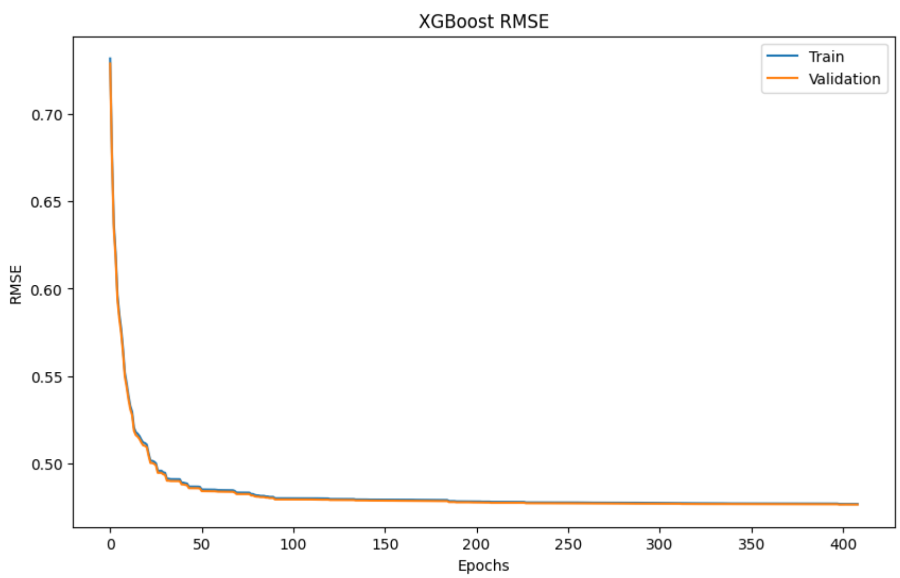

Figure 13: RMSE over boosting rounds

Testing and Validation scores follow each other very closely to reach a minimum of RMSE 0.48 which is an indication that no overfitting exists. However no significant improvement is observed after epoch 50 so my model is basically saturated and unable to capture other patterns. Epochs represent the boosting rounds my XGBoost model.

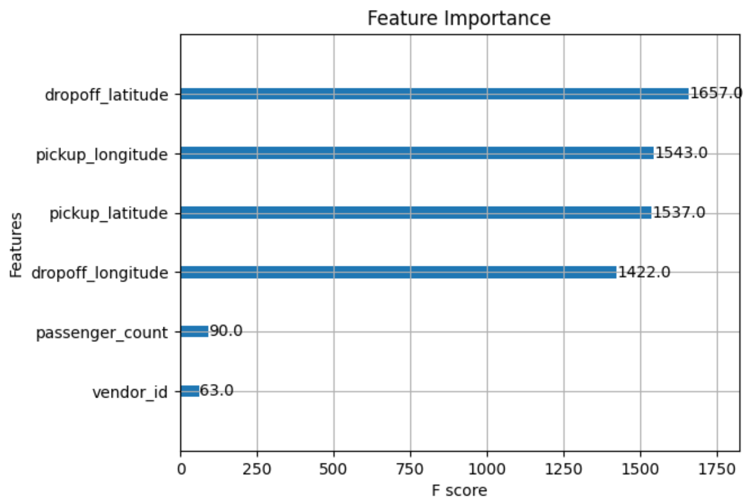

Figure 14: Feature Importance

The conclusion from that first model was that a training set that is data enriched and includes more features might be beneficial and help me achieve a better performance.

Further features were iteratively and gradually added to the model, based on the importance scores, included the features that were already engineered before.

6 PCA and Clustering

6.1 Clustering on Central Coordinate

Some of these new features were cluster-based. Trips for example were clustered based on their central coordinate. The central coordinate for each trip was calculated as the midpoint between the pickup and dropoff locations. By doing that I'm hoping to provide to the model a simplified representation of the trip's trajectory, group the trips based on geographical location and potentially revealing patterns that the previous model could not 'see'.

For all the data points that were not falling inside the defined bounds of the city (based on the maps before) were clustered in a single group together and separately from the others.

All the other data points were clustered using KMeans and the Elbow Method for picking the optimal number of clusters.

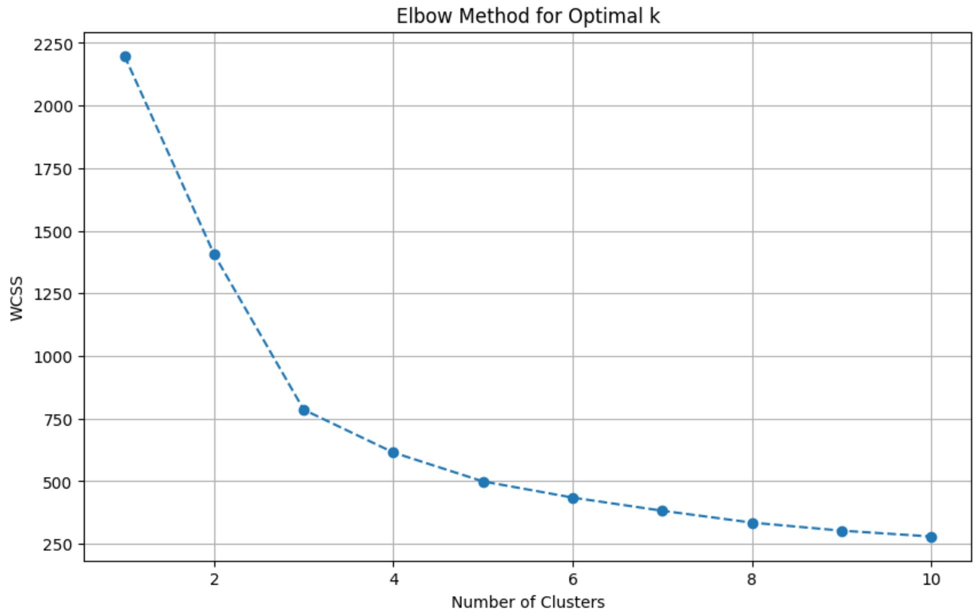

Figure 15: Elbow Method

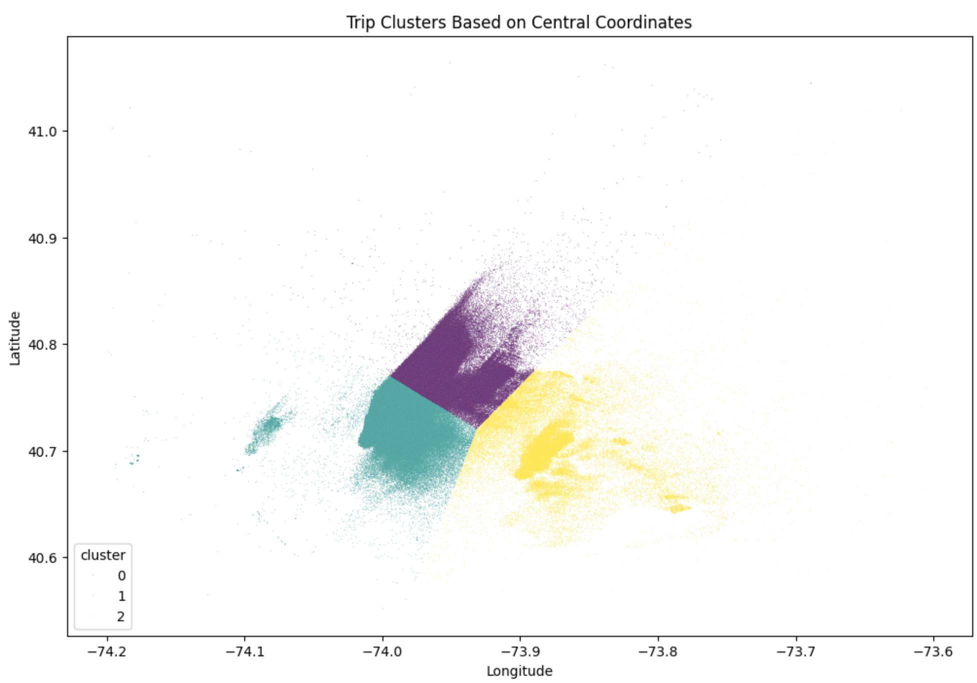

Figure 16: Central Coordinate clustering

6.2 Clustering on Consistently Important Features

Through several model iterations, specific features were observed as being consistently important in the model with F scores, meaning contributing a lot to the model's predictive power. These were then picked to cluster pickup and dropoff points.
The key idea here again is to try to enhance the predictive power of the model by providing it with additional structure and data insights and possibly capture more complex interactions.
These features included:

Variables
calculated_distance
calculated_duration
avg_speed_calculated_distance
pickup_longitude
pickup_latitude
dropoff_longitude
dropoff_latitude
trips_day_hour
num_points

Table 4: List of Variables in the Dataset

The elbow method was again used to pick the optimal number of clusters. (while points outside the boundaries were again clustered separately)

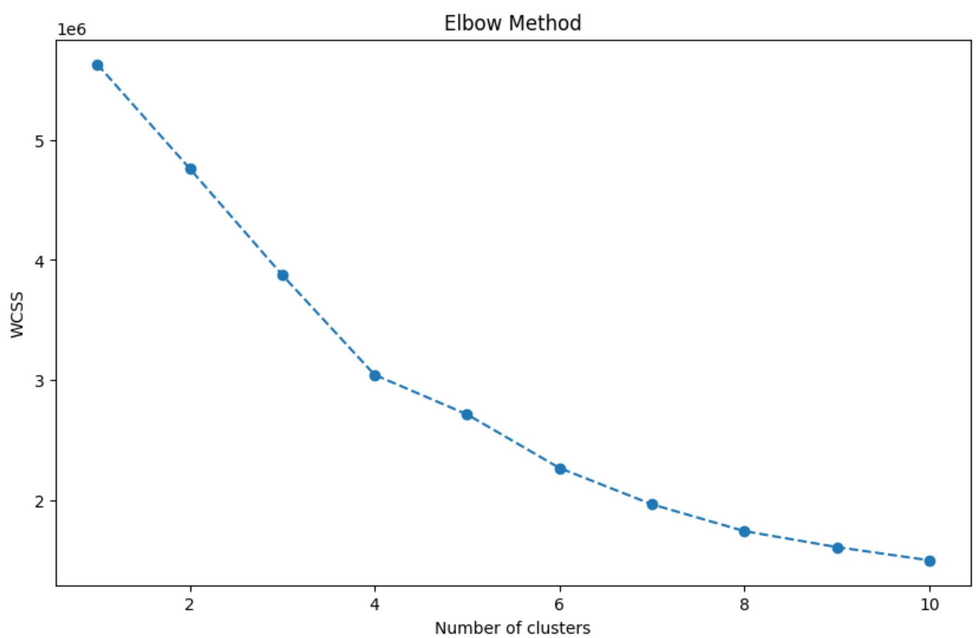

Figure 17: Elbow Method

6.3 PCA of Longitudes and Latitudes

Since that first heatmap I saw that pickup and dropoff longitudes and latitudes are positively correlated which gives a hint on the route patters, for example if pickup and dropoff points are mostly along the same latitude that could be valueable information for the model. By applying PCA on the coordinates, I'm making linear combinations of the originals.These components might represent more meaningful geographic features than the raw coordinates. For example, the first principal component might capture the general direction of travel eg north-south vs. east-west.

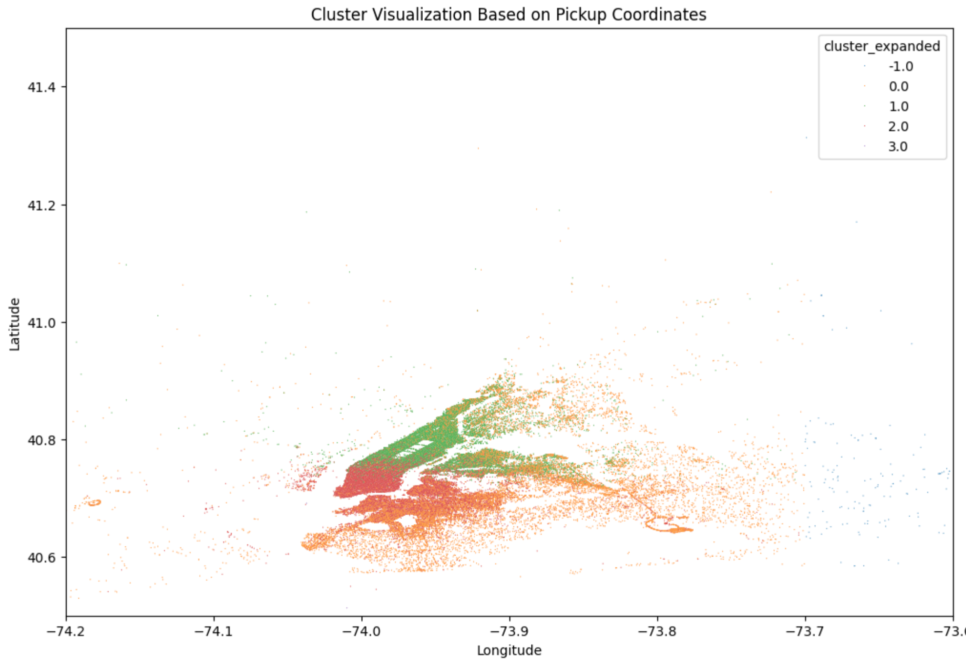

Figure 18: Dropoff Clusters

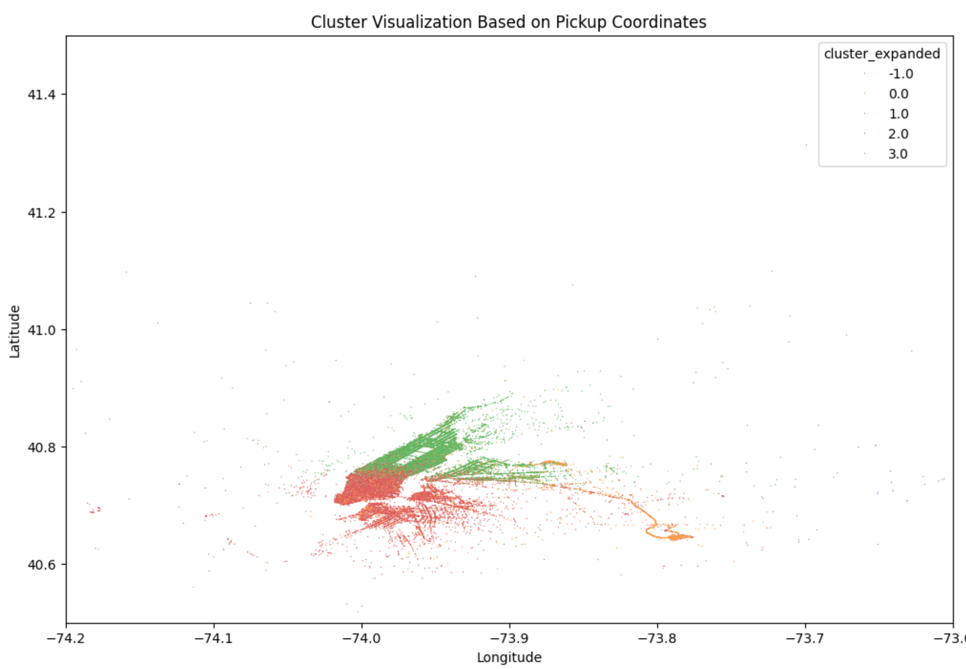

Figure 19: Pickup Clusters

Finally, based on the new clusters and pcs, new aggregated features were created eg about speed, calculated duration (from the polylines), num points etc.

7 Final modelling

With these new engineered features I proceeded fitting a final XGBoost model. The process was similar as before, using the same training and validation split and Optuna for the same hyperparameter optimization. The final training dataset included 82 features, listed below. The same pre-processing was performed to the testing dataset.

Variables (1)	**Variables (2)**	**Variables (3)**
id	day_of_week	avg_duration_discretized_direction
vendor_id	date	avg_speed_discretized_direction
pickup_datetime	num_points	avg_distance_discretized_direction
passenger_count	month	avg_duration_precipitation
pickup_longitude	week	avg_speed_precipitation
pickup_latitude	day	avg_duration_temperature
dropoff_longitude	hour	avg_speed_temperature
dropoff_latitude	minute	count_trips_temperature
average temperature	hour_minute	count_trips_precipitation
precipitation	avg_speed_minute	count_trips_holidays
is_holiday	avg_speed_hour_minute	central_latitude
calculated_distance	total_trips_hour_minute	central_longitude
calculated_duration	avg_speed_calculated_distance_avg_hour	cluster
avg_speed_calculated_distance	avg_speed_calculated_distance_avg_day	avg_duration_centralized_clusters
id_count_hour	avg_speed_calculated_distance_avg_day_of_week	avg_speed_centralized_clusters
id_count_day	avg_speed_calculated_distance_avg_week	avg_distance_centralized_clusters
id_count_day_of_week	avg_speed_calculated_distance_avg_month	count_trips_centralized_clusters
id_count_week	avg_speed_calculated_distance_avg_holiday	avg_speed_day_hour
id_count_month	id_count_airports	avg_speed_by_calculated_duration
id_count_holiday	direction	avg_duration_day_hour
avg_duration_hour	discretized_direction	trips_day_hour
avg_duration_hour_minute	count_trips_discretized_direction	trip_count_week_hour
avg_speed_vendor	combined_pc1	avg_speed_week_hour
is_airport	combined_pc2	avg_duration_week_hour
cluster_expanded	combined_pc3	combined_pc4
avg_speed_expanded_clusters	avg_num_points_hour_minute	avg_speed_passenger_count
avg_duration_expanded_clusters	avg_speed_expanded_clusters	trip_duration
avg_distance_expanded_clusters	count_expanded_clusters	

Table 5: List of Variables in the Dataset

For the sake of time, only 10 trials were performed for hyperparameter optimization

7.1 Results

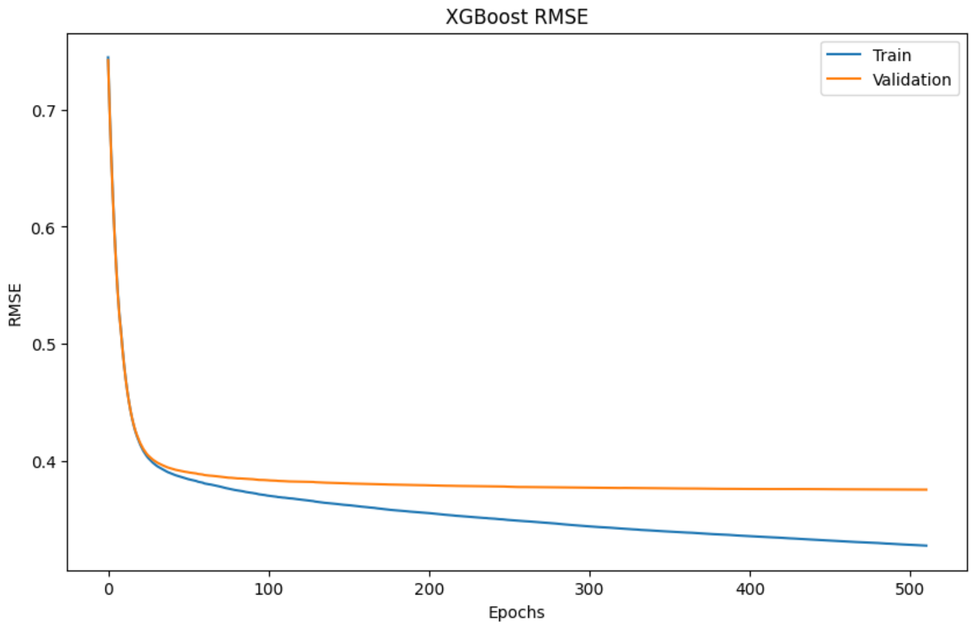

Figure 20: RMSE on final training

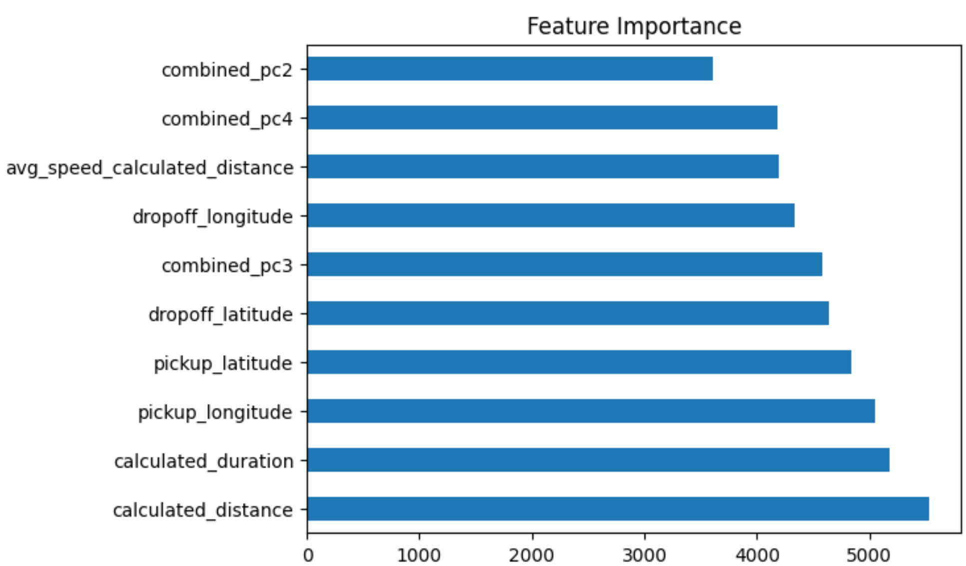

Figure 21: Most Important

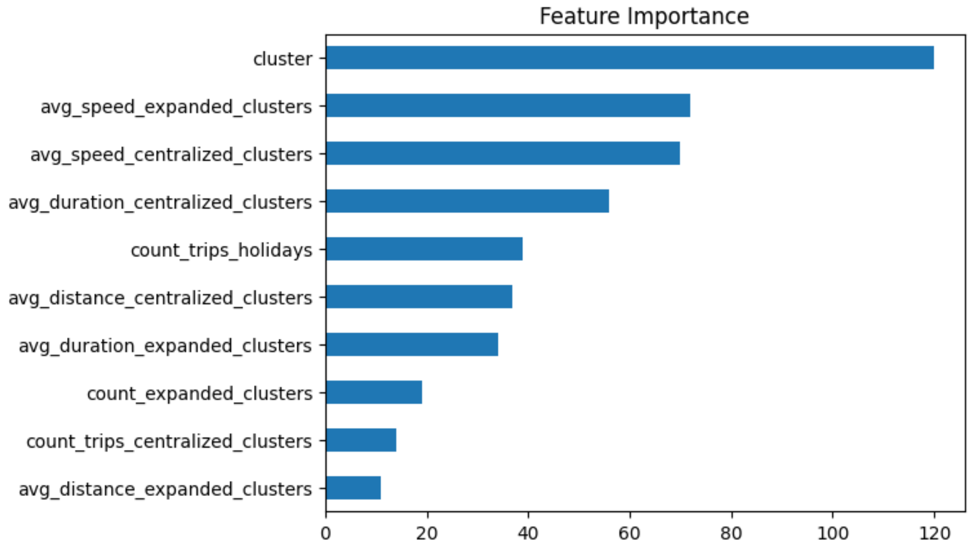

Figure 22: Least Important

I note that this time overfitting appears to exist which seem to begin around the 50th boosting round. The training's RMLSE score keep decreasing while validation is reaching a plateau. That's a clear indication that the model is no longer able to generalize its newly acquired knowledge to unseen data.

Metric	Value
Final Training RMSLE	0.3275234274552838
Final Validation RMSLE	0.3753682846040504

Table 6: Final Training and Validation RMSE

On the bright side, the new, enhanced training data were really used by my model to significantly decrease its validation score from 0.48 in the first model to 0.37, that alone is a noteworthy decrease. Although my model, far from perfect, with signs of overfitting, it should able to position itself competitively in the Kaggle ladder.

Unsurprisingly the most important features were the ones that I created using Google's geolocation API features. I managed through the given polylines to calculate probable routes (which were also based on historical data on traffic), their probable distance and durations. The raw longitudes and latitudes also played a great role, in addition to the principal components I created by applying PCA to them. Overall, certain decisions on data enhancement were the ones that played a significant role in creating a better model.

8 Further Enhancements for the future

Due to the restrictions of time, only few hyperparameter optimization trials were performed. Additionally Boosting with XGBoost was the only tried method due to its known effectiveness, speed and many parameters available for customization to achieve a better result.

Other algorithms like random forest, bagging, gradient boosting or support vector machines could be tested as well. Stacking several predictions together from different models might also give an edge and could be considered for the future.

9 Appendix

```python
import pandas as pd
import numpy as np
from pandas.tseries.holiday import USFederalHolidayCalendar
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import xgboost as xgb
from sklearn.metrics import mean_squared_error


import matplotlib.pyplot as plt
import seaborn as sns
import datashader as ds
import datashader.transfer_functions as tf
import holoviews as hv
from holoviews.element.tiles import StamenTerrain

import geopandas as gpd
from shapely.geometry import Point, LineString
from shapely import wkt
from geopy.distance import geodesic, great_circle
from geopandas.tools import sjoin

import optuna
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA


from concurrent.futures import ThreadPoolExecutor, as_completed

import warnings
import googlemaps
import ast
from ast import literal_eval
from tqdm import tqdm
from datetime import datetime

data = pd.read_csv('train.csv')

taxi_data = data

taxi_data['pickup_datetime'] = pd.to_datetime(taxi_data['pickup_datetime'])
taxi_data['month'] = taxi_data['pickup_datetime'].dt.month
# taxi_data['week'] = taxi_data['pickup_datetime'].dt.week
taxi_data['day'] = taxi_data['pickup_datetime'].dt.day
taxi_data['hour'] = taxi_data['pickup_datetime'].dt.hour
taxi_data['minute'] = taxi_data['pickup_datetime'].dt.minute
taxi_data['day_of_week'] = taxi_data['pickup_datetime'].dt.day_name()


taxi_data = taxi_data.sort_values('pickup_datetime')

def batch_data(data, batch_size):
    for i in range(0, len(data), batch_size):
        yield data.iloc[i:i + batch_size]

batch_size = 10

batches = batch_data(taxi_data, batch_size)
```

```
total_batches_count = 0

for _ in batches:
    total_batches_count += 1

print(f"Total number of batches: {total_batches_count}")

from collections import Counter
batch_sizes = Counter()
for batch in batches:
    batch_size = len(batch)
    batch_sizes[batch_size] += 1
for size, count in batch_sizes.items():
    print(f"Batch size {size}: {count} batches")

gmaps_client = googlemaps.Client(key=my_API_KEY)
taxi_data['calculated_distance'] = None
taxi_data['calculated_duration'] = None


def process_batch(batch, gmaps_client):
    origins = batch.apply(lambda row: (row['pickup_latitude'], row['pickup_longitude']), axis=1).tolist()
    destinations = batch.apply(lambda row: (row['dropoff_latitude'], row['dropoff_longitude']), axis=1).tolist()


    departure_times = batch['pickup_datetime'].apply(
        lambda x: datetime(2028, x.month, x.day, x.hour, x.minute, x.second).timestamp()).tolist()
    mean_departure_time = sum(departure_times) / len(departure_times)


    try:

        result = gmaps_client.distance_matrix(origins=origins,
                                              destinations=destinations,
                                              mode="driving",
                                              departure_time=mean_departure_time,
                                              traffic_model='best_guess')


        for i in range(len(batch)):
            distance = result['rows'][i]['elements'][i]['distance']['value']
            duration_in_traffic = result['rows'][i]['elements'][i].get('duration_in_traffic', {}).get('value')


            batch.at[batch.index[i], 'calculated_distance'] = distance
            batch.at[batch.index[i], 'calculated_duration'] = duration_in_traffic


    except Exception as e:
        print(f"Error processing batch: {e}")

    return batch

taxi_data = taxi_data.sort_values('pickup_datetime')

```

```
processed_batches = []
batch_count = 0


total_records = len(taxi_data)
batch_size = 10
total_batches = (total_records + batch_size - 1) // batch_size

for batch in batches:
    processed_batch = process_batch(batch, gmaps_client)
    processed_batches.append(processed_batch)

    batch_count += 1
    batches_left = total_batches - batch_count
    print(f"Processed batch number {batch_count}, {batches_left} batches left"
        )

processed_df = pd.concat(processed_batches)

taxi_data = taxi_data.merge(processed_df, left_index=True, right_index=True,
                            suffixes=('', '_updated'))


update_cols = ['calculated_distance', 'calculated_duration'
               ]
for col in update_cols:
    taxi_data[col] = taxi_data[col + '_updated']


for col in taxi_data.columns:
    if col.endswith('_updated'):
        taxi_data.drop(col, axis=1, inplace=True)

taxi_data.to_csv("updated_taxi_data.csv", index=False)


print(taxi_data.head())



calendar = USFederalHolidayCalendar()


start_date = merged_dataset['date'].min()
end_date = merged_dataset['date'].max()
holidays = calendar.holidays(start=start_date, end=end_date)

merged_dataset['date'] = pd.to_datetime(merged_dataset['date'])
merged_dataset['is_holiday'] = merged_dataset['date'].isin(holidays).astype(
    int)

weather = pd.read_csv('weather_data.csv')
weather


merged_dataset = taxi_data.merge(weather, on='date', how='left')

merged_dataset['is_holiday'] = merged_dataset.apply(lambda row: 1 if (row['
    is_holiday'] == 1 or row['day_of_week'] in ['Saturday', 'Sunday']) else 0,
     axis=1)

```

```python
final_data_weekend = merged_dataset[merged_dataset['day_of_week'].isin(['Saturday', 'Sunday'])]

final_data_weekend_holidays = final_data_weekend[final_data_weekend['is_holiday'] == 1]


number_of_rows = final_data_weekend_holidays.shape[0]

print(f"The number of rows where the day is Sunday or Saturday and are marked with 1 in the 'holidays' column is: {number_of_rows}")


# GOOGLE'S DIRECTIONS API


gmaps_client = googlemaps.Client(key=API_KEY)


taxi_data['route_polyline'] = None
taxi_data['pickup_datetime'] = pd.to_datetime(taxi_data['pickup_datetime'])


def process_row(row, gmaps_client):
    origin = (row['pickup_latitude'], row['pickup_longitude'])
    destination = (row['dropoff_latitude'], row['dropoff_longitude'])
    departure_time = datetime(2028, row['pickup_datetime'].month, row['pickup_datetime'].day,
                              row['pickup_datetime'].hour, row['pickup_datetime'].minute,
                              row['pickup_datetime'].second).timestamp()

    try:

        directions_result = gmaps_client.directions(origin, destination,
                                                     mode="driving",
                                                     departure_time=departure_time)


        if directions_result:
            polyline = directions_result[0]['overview_polyline']['points']
            return polyline
    except Exception as e:
        print(f"Error processing row: {e}")
        return None


def parallel_process(taxi_data, gmaps_client, max_workers=10):
    total_rows = len(taxi_data)
    processed_count = 0

    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        future_to_row = {executor.submit(process_row, row, gmaps_client): i for i, row in taxi_data.iterrows()}
        for future in as_completed(future_to_row):
            row_index = future_to_row[future]
            try:
                polyline = future.result()
                taxi_data.at[row_index, 'route_polyline'] = polyline
            except Exception as e:
                print(f"Error processing row {row_index}: {e}")
```

```python


            processed_count += 1
            print(f"Processed {processed_count}/{total_rows} rows...")

    return taxi_data

taxi_data = parallel_process(taxi_data, gmaps_client, max_workers=10)

taxi_data.to_csv("updated_taxi_data_routes.csv", index=False)
print(taxi_data.head())


# NUM POINTS AND ROUTE LENGTH CALCULATION


taxi_data['route_length'] = None
taxi_data['num_points'] = None

def extract_route_features(row_tuple):
    index, row = row_tuple
    if pd.notnull(row['decoded_polyline']):
        polyline = ast.literal_eval(row['decoded_polyline'])
        route_length = sum(great_circle(polyline[i], polyline[i+1]).meters for
             i in range(len(polyline)-1))
        num_points = len(polyline)

    return index, route_length, num_points

def process_data_in_parallel(data, func, workers):

    processed_count = 0
    with ThreadPoolExecutor(max_workers=workers) as executor:

        futures = [executor.submit(func, (index, row)) for index, row in data.
            iterrows()]

        for future in tqdm(as_completed(futures), total=len(futures), desc='
            Calculating routes'):
            index, route_length, num_points = future.result()
            data.at[index, 'route_length'] = route_length
            data.at[index, 'num_points'] = num_points
            processed_count += 1
            if processed_count % 1 == 0:
                print(f"Processed {processed_count} rows")
    return data


taxi_data = process_data_in_parallel(taxi_data, extract_route_features,
    workers=200)

taxi_data.to_csv('updated_taxi_data_routes_clustered_cleaned_polylined2',
    index=False)
print(taxi_data.head())

taxi_data['avg_speed_calculated_distance'] = np.where(
    taxi_data['calculated_duration'] > 0,
    taxi_data['calculated_distance'] / taxi_data['calculated_duration'],
    0
)

taxi_data['avg_speed_route_length'] = np.where(
```

```
    taxi_data['calculated_duration'] > 0,
    taxi_data['route_length'] / taxi_data['calculated_duration'],
    0
)

taxi_data[['avg_speed_calculated_distance', 'avg_speed_route_length']]


# DISTRICT VISUALIZATION


taxi_zones['the_geom'] = taxi_zones['the_geom'].apply(wkt.loads)
taxi_gdf = gpd.GeoDataFrame(taxi_zones, geometry='the_geom')

sns.set(style="whitegrid")

palette = sns.color_palette("viridis", as_cmap=True)

fig, ax = plt.subplots(1, 1, figsize=(15, 10))

taxi_gdf.plot(ax=ax, cmap=palette)

ax.set_title('New York Taxi Zones Enhanced with Seaborn')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.show()


airports = ['Airport' in name for name in taxi_gdf['zone']]

taxi_gdf['color'] = 'blue'
taxi_gdf.loc[airports, 'color'] = 'red'

fig, ax = plt.subplots(1, 1, figsize=(15, 10))

boroughs = taxi_gdf['borough'].unique()
palette = sns.color_palette("Set2", len(boroughs))

for borough, color in zip(boroughs, palette):
    taxi_gdf[taxi_gdf['borough'] == borough].plot(ax=ax, color=color,
        edgecolor='black')

taxi_gdf[taxi_gdf['color'] == 'red'].plot(ax=ax, color='red', edgecolor='black
    ')

for idx, row in airport_gdf.iterrows():
    centroid = row['the_geom'].centroid
    ax.annotate(row['zone'], (centroid.x, centroid.y), color='black', fontsize
        =10, ha='center', va='center')

ax.set_title('New York Taxi Zones with Airports Highlighted')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')

borough_patches = [plt.Line2D([0], [0], color=color, lw=4, label=borough) for
    borough, color in zip(boroughs, palette)]
airport_patch = plt.Line2D([0], [0], color='red', lw=4, label='Airports')
ax.legend(handles=borough_patches + [airport_patch], loc='upper left')
plt.show()
```

```python
ny_districts_data['the_geom'] = ny_districts_data['the_geom'].apply(wkt.loads)

gdf = gpd.GeoDataFrame(ny_districts_data, geometry='the_geom')


gdf['C1_C2'] = gdf['ZONEDIST'].apply(lambda x: x.startswith('C1') or x.startswith('C2') or x.startswith('C3')
                                               or x.startswith('C4') or x.startswith('C5') or x.startswith('C6')
                                               or x.startswith('C7') or x.startswith('C8'))

sns.set(style="whitegrid")
num_commercial_districts = gdf[gdf['C1_C2']].shape[0]

fig, ax = plt.subplots(1, 1, figsize=(15, 10))

gdf[gdf['C1_C2'] == False].plot(ax=ax, color='lightblue', edgecolor='black')
gdf[gdf['C1_C2']].centroid.plot(ax=ax, marker='o', color='red', markersize=30)


ax.set_title('New York Zoning Districts with Prominent Commercial Districts')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')

legend_label = f"Commercial Districts (C1/2/3.../8): {num_commercial_districts}"
ax.legend([legend_label], loc='upper left')
plt.show()

### (SAME GOES FOR OTHER DISTRICTS - CODE NOT ### INCLUDED)


##FURTHER FEATURE ENGINEERING OF GEO FEATURES

taxi_zone_df['geometry'] = taxi_zone_df['the_geom'].apply(wkt.loads)
geo_taxi_zone_df = gpd.GeoDataFrame(taxi_zone_df, geometry='geometry')

geo_taxi_zone_df.set_crs("EPSG:4326", inplace=True)

gdf_taxi_data = gpd.GeoDataFrame(
    taxi_data_df,
    geometry=gpd.points_from_xy(taxi_data_df.pickup_longitude, taxi_data_df.pickup_latitude),
    crs="EPSG:4326"
)

gdf_taxi_dropoff = gpd.GeoDataFrame(
    taxi_data_df,
    geometry=gpd.points_from_xy(taxi_data_df.dropoff_longitude, taxi_data_df.dropoff_latitude),
    crs="EPSG:4326"
)

taxi_data_with_pickup_zones = sjoin(gdf_taxi_data, geo_taxi_zone_df, how='left', predicate='within')
taxi_data_with_pickup_zones.rename(columns={'zone': 'pickup_zone', 'borough': 'pickup_borough'}, inplace=True)
taxi_data_with_dropoff_zones = sjoin(gdf_taxi_dropoff, geo_taxi_zone_df, how='left', predicate='within')
```

```
taxi_data_with_dropoff_zones.rename(columns={'zone': 'dropoff_zone', 'borough': 'dropoff_borough'}, inplace=True)

taxi_data_final = taxi_data_df.copy()
taxi_data_final = taxi_data_final.merge(taxi_data_with_pickup_zones[['pickup_zone', 'pickup_borough']], left_index=True, right_index=True, how='left')
taxi_data_final = taxi_data_final.merge(taxi_data_with_dropoff_zones[['dropoff_zone', 'dropoff_borough']], left_index=True, right_index=True, how='left')

no_pickup_zone_count = taxi_data_final['pickup_zone'].isnull().sum()
no_dropoff_zone_count = taxi_data_final['dropoff_zone'].isnull().sum()
both_zones_missing_count = taxi_data_final[(taxi_data_final['pickup_zone'].isnull()) & (taxi_data_final['dropoff_zone'].isnull())].shape[0]

taxi_data_final.head(), (no_pickup_zone_count, no_dropoff_zone_count, both_zones_missing_count)



commercial_districts = nyzd_df[nyzd_df['ZONEDIST'].str.startswith(('C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8'))]

commercial_districts['geometry'] = commercial_districts['the_geom'].apply(wkt.loads)
geo_commercial_districts = gpd.GeoDataFrame(commercial_districts, geometry='geometry')
geo_commercial_districts.set_crs("EPSG:4326", inplace=True)

pickup_in_commercial = sjoin(gdf_taxi_data, geo_commercial_districts, how='left', predicate='within')
taxi_data_df['pickup_commercial_district'] = pickup_in_commercial['ZONEDIST'].notnull().astype(int)

dropoff_in_commercial = sjoin(gdf_taxi_dropoff, geo_commercial_districts, how='left', predicate='within')
taxi_data_df['dropoff_commercial_district'] = dropoff_in_commercial['ZONEDIST'].notnull().astype(int)



residential_districts = nyzd_df[nyzd_df['ZONEDIST'].str.startswith(('R1', 'R2', 'R3','R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10'))]

residential_districts['geometry'] = residential_districts['the_geom'].apply(wkt.loads)
geo_residential_districts = gpd.GeoDataFrame(residential_districts, geometry='geometry')
geo_residential_districts.set_crs("EPSG:4326", inplace=True)

pickup_in_residential = sjoin(gdf_taxi_data, geo_residential_districts, how='left', predicate='within')
pickup_in_residential.reset_index(drop=True, inplace=True)
taxi_data_df['pickup_residential_district'] = pickup_in_residential['ZONEDIST'].notnull().astype(int)

dropoff_in_residential = sjoin(gdf_taxi_dropoff, geo_residential_districts, how='left', predicate='within')
dropoff_in_residential.reset_index(drop=True, inplace=True)
taxi_data_df['dropoff_residential_district'] = dropoff_in_residential['ZONEDIST'].notnull().astype(int)
```

```
taxi_zones_df['geometry'] = taxi_zones_df['the_geom'].apply(wkt.loads)
geo_taxi_zones_df = gpd.GeoDataFrame(taxi_zones_df, geometry='geometry')
geo_taxi_zones_df.set_crs("EPSG:4326", inplace=True)

airport_zones = geo_taxi_zones_df[geo_taxi_zones_df['zone'].str.contains("Airport", case=False, na=False)]

airport_zones_buffered = airport_zones.to_crs("EPSG:2263")

airport_zones_buffered['geometry'] = airport_zones_buffered['geometry'].buffer(50)

airport_zones_buffered = airport_zones_buffered.to_crs("EPSG:4326")

pickup_in_airport = sjoin(gdf_taxi_data, airport_zones_buffered, how='left', predicate='intersects')
taxi_data_df['pickup_airport'] = pickup_in_airport['zone'].notnull().astype(int)

dropoff_in_airport = sjoin(gdf_taxi_dropoff, airport_zones_buffered, how='left', predicate='intersects')
taxi_data_df['dropoff_airport'] = dropoff_in_airport['zone'].notnull().astype(int)

taxi_data_df.head()

#FIRST MODEL

data = simple_taxi_data.copy()


data = data.drop(['id', 'pickup_datetime', 'dropoff_datetime', 'store_and_fwd_flag'], axis=1)


label_encoders = {}
categorical_columns = ['vendor_id']

for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    data[column] = label_encoders[column].fit_transform(data[column])


features = [col for col in data.columns if col != 'trip_duration']
X = data[features]
y = np.log1p(data['trip_duration'])

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=0)

dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_val, label=y_val)

def objective(trial):
    param = {
        'objective': 'reg:squarederror',
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.1, 1),
        'tree_method': trial.suggest_categorical('tree_method', ['approx', 'hist']),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.001, 25, log=True),
        'learning_rate': trial.suggest_float('learning_rate', 0.001, 0.25),
```

```
        'max_depth': trial.suggest_int('max_depth', 1, 12),
        'subsample': trial.suggest_float('subsample', 0.05, 1),
        'alpha': trial.suggest_float('alpha', 1, 10),
        'gamma': trial.suggest_float('gamma', 0, 5),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 100),
        'eval_metric': 'rmse',
    }

    evals = [(dtrain, 'train'), (dval, 'eval')]
    evals_result = {}
    model = xgb.train(param, dtrain, num_boost_round=1000, evals=evals, evals_result=evals_result, early_stopping_rounds=5, verbose_eval=False)

    y_pred = model.predict(dval)
    rmse = np.sqrt(mean_squared_error(y_val, y_pred))

    return rmse

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=3)


best_params = study.best_params
best_params['eval_metric'] = 'rmse'

evals = [(dtrain, 'train'), (dval, 'eval')]
evals_result = {}
model = xgb.train(best_params, dtrain, num_boost_round=1000, evals=evals, evals_result=evals_result, early_stopping_rounds=10, verbose_eval=False)

epochs = len(evals_result['train']['rmse'])
x_axis = range(0, epochs)

plt.figure(figsize=(10, 6))
plt.plot(x_axis, evals_result['train']['rmse'], label='Train')
plt.plot(x_axis, evals_result['eval']['rmse'], label='Validation')
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.title('XGBoost RMSE')
plt.legend()
plt.show()

X_full = pd.concat([X_train, X_val], axis=0)
y_full = pd.concat([y_train, y_val], axis=0)

full_dtrain = xgb.DMatrix(X_full, label=y_full)

final_model = xgb.train(best_params, full_dtrain, num_boost_round=model.best_iteration)

xgb.plot_importance(final_model)
plt.title('Feature Importance')
plt.show()

##AGGREGATING FEATURES (I DONT SHOW ALL HERE)

taxi_data['avg_duration_precipitation'] = taxi_data.groupby('precipitation')['calculated_duration'].transform('mean')
taxi_data['avg_speed_precipitation'] = taxi_data.groupby('precipitation')['avg_speed_calculated_distance'].transform('mean')
```

```
taxi_data['avg_duration_temperature'] = taxi_data.groupby('average temperature
    ')['calculated_duration'].transform('mean')
taxi_data['avg_speed_temperature'] = taxi_data.groupby('average temperature')[
    'avg_speed_calculated_distance'].transform('mean')
taxi_data['avg_duration_snow'] = taxi_data.groupby('snow fall')['
    calculated_duration'].transform('mean')
taxi_data['avg_speed_snow'] = taxi_data.groupby('snow fall')['
    avg_speed_calculated_distance'].transform('mean')
taxi_data['count_trips_snow'] = taxi_data.groupby('snow fall')['id'].transform
    ('count')
taxi_data['count_trips_temperature'] = taxi_data.groupby('average temperature'
    )['id'].transform('count')
taxi_data['count_trips_precipitation'] = taxi_data.groupby('precipitation')['
    id'].transform('count')
taxi_data['avg_duration_holidays'] = taxi_data.groupby('is_holiday')['
    calculated_duration'].transform('mean')
taxi_data['count_trips_holidays'] = taxi_data.groupby('is_holiday')['id'].
    transform('count')

#clustering


taxi_data['central_latitude'] = (taxi_data['pickup_latitude'] + taxi_data['
    dropoff_latitude']) / 2
taxi_data['central_longitude'] = (taxi_data['pickup_longitude'] + taxi_data['
    dropoff_longitude']) / 2

X = taxi_data[['central_latitude', 'central_longitude']]

#elbow method


wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()


plt.figure(figsize=(12, 8))
sns.scatterplot(data=taxi_data, x='central_longitude', y='central_latitude',
    hue='cluster', palette='viridis', s = 0.2, legend='full')
plt.title('Trip Clusters Based on Central Coordinates')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()

#CLUSTERING ON TOP FEATURES



warnings.filterwarnings('ignore')

numerical_features = ['calculated_distance',
```

```
                            'calculated_duration',
                            'avg_speed_calculated_distance', 'pickup_longitude', '
                                pickup_latitude',
                            'dropoff_longitude','dropoff_latitude', 'trips_day_hour'
                                , 'num_points'


]
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features)

    ])


X = preprocessor.fit_transform(combined_data1[numerical_features])

#FINAL MODEL

data = train.copy()

data = data.drop(['id', 'pickup_datetime', 'date', 'direction'], axis=1)

label_encoders = {}
categorical_columns = ['vendor_id', 'day_of_week', 'discretized_direction']

for column in categorical_columns:
    label_encoders[column] = LabelEncoder()
    data[column] = label_encoders[column].fit_transform(data[column])

features = [col for col in data.columns if col != 'trip_duration']
X = data[features]
y = np.log1p(data['trip_duration'])

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
    random_state=0)

dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_val, label=y_val)

def objective(trial):
    param = {
        'objective': 'reg:squarederror',
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.1, 1),
        'tree_method': trial.suggest_categorical('tree_method', ['approx', '
            hist']),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.001, 25, log=True),
        'learning_rate': trial.suggest_float('learning_rate', 0.001, 0.25),
        'max_depth': trial.suggest_int('max_depth', 1, 12),
        'subsample': trial.suggest_float('subsample', 0.05, 1),
        'alpha': trial.suggest_float('alpha', 1, 10),
        'gamma': trial.suggest_float('gamma', 0, 5),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 100),
        'eval_metric': 'rmse',
    }

    evals = [(dtrain, 'train'), (dval, 'eval')]
    evals_result = {}
    model = xgb.train(param, dtrain, num_boost_round=1000, evals=evals,
        evals_result=evals_result, early_stopping_rounds=5, verbose_eval=False
        )

```

```
    y_pred = np.expm1(model.predict(dval))
    rmsle = np.sqrt(mean_squared_log_error(np.expm1(y_val), y_pred))

    return rmsle

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=5)

best_params = study.best_params
best_params['eval_metric'] = 'rmse'

evals = [(dtrain, 'train'), (dval, 'eval')]
evals_result = {}
model = xgb.train(best_params, dtrain, num_boost_round=1000, evals=evals,
    evals_result=evals_result, early_stopping_rounds=10, verbose_eval=False)

epochs = len(evals_result['train']['rmse'])
x_axis = range(0, epochs)

plt.figure(figsize=(10, 6))
plt.plot(x_axis, evals_result['train']['rmse'], label='Train')
plt.plot(x_axis, evals_result['eval']['rmse'], label='Validation')
plt.xlabel('Epochs')
plt.ylabel('RMLSE')
plt.title('XGBoost RMLSE')
plt.legend()
plt.show()

X_full = pd.concat([X_train, X_val], axis=0)
y_full = pd.concat([y_train, y_val], axis=0)

full_dtrain = xgb.DMatrix(X_full, label=y_full)

final_model = xgb.train(best_params, full_dtrain, num_boost_round=model.
    best_iteration)

xgb.plot_importance(final_model)
plt.title('Feature Importance')
plt.show()
```