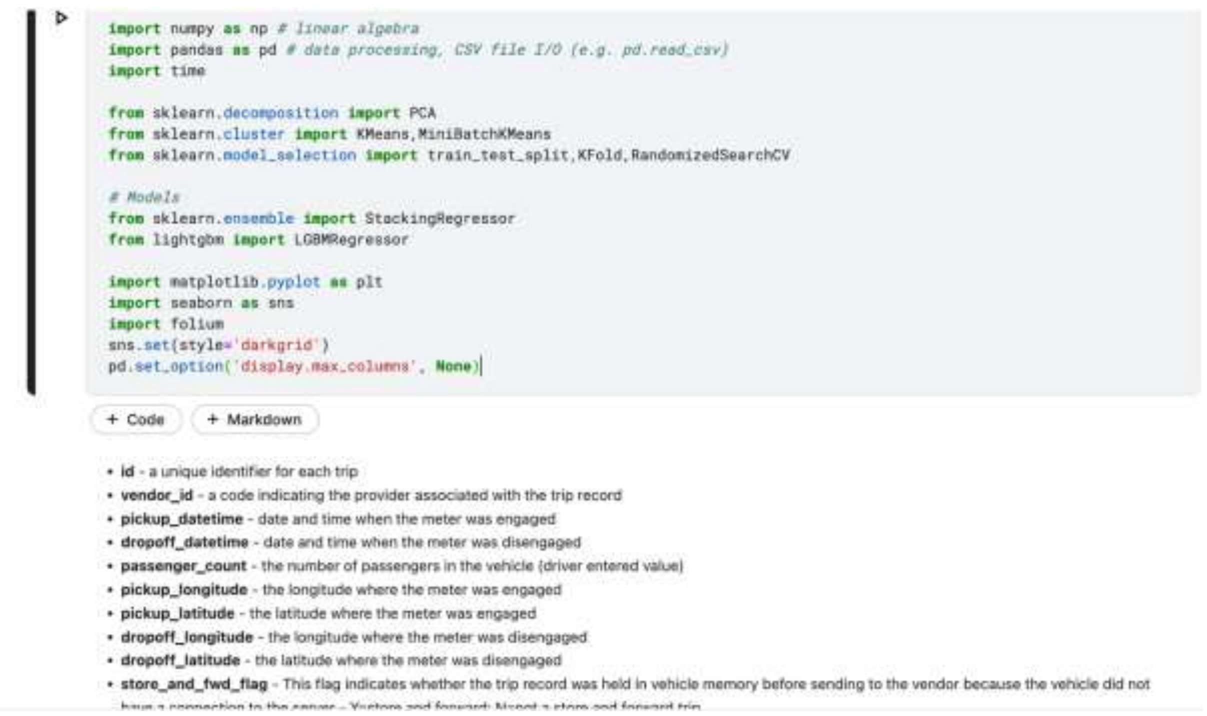
New York City Taxi Trip Duration Prediction.

1. Exploratory Data Analysis and Feature Engineering Report

Introduction
The New York City Taxi Trip Duration Prediction project involves predicting the duration of taxi trips in the bustling city of New York. In this report, we delve into the detailed exploratory data analysis (EDA) and feature engineering processes, shedding light on the steps taken to understand the dataset, pre-process the data, and derive meaningful features for building an effective predictive model.

Dataset Overview
The dataset comprises historical records of taxi trips, encompassing various attributes such as pickup and drop-off coordinates, trip duration, passenger count, and timestamps. Both training and testing datasets were provided, with the primary objective of training a model on the former and making predictions on the latter.

Loading and Initial Inspection
The initial phase involved loading the datasets into Pandas Data Frames and performing a preliminary exploration. This included examining the data types, checking for missing values, and gaining a high-level understanding of the dataset's structure.

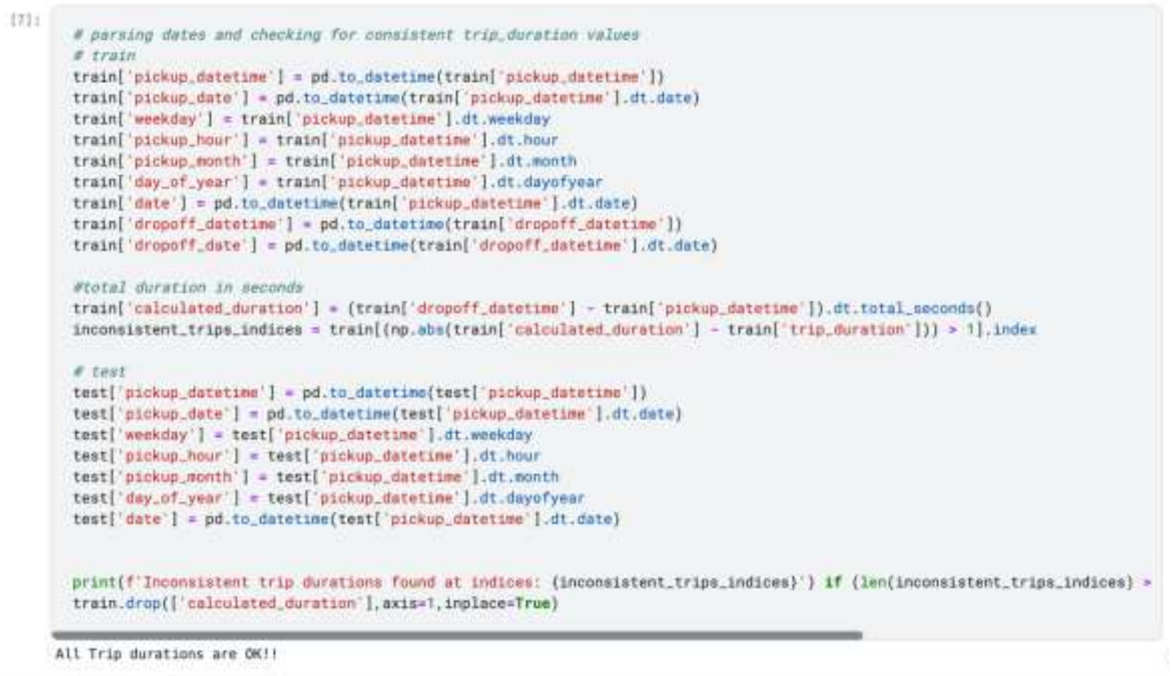


```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import time

from sklearn.decomposition import PCA
from sklearn.cluster import KMeans,MiniBatchKMeans
from sklearn.model_selection import train_test_split,KFold,RandomizedSearchCV

# Models
from sklearn.ensemble import StackingRegressor
from lightgbm import LGBMRegressor

import matplotlib.pyplot as plt
import seaborn as sns
import folium
sns.set(style='darkgrid')
pd.set_option('display.max_columns', None)
```

- **id** - a unique identifier for each trip
- **vendor_id** - a code indicating the provider associated with the trip record
- **pickup_datetime** - date and time when the meter was engaged
- **dropoff_datetime** - date and time when the meter was disengaged
- **passenger_count** - the number of passengers in the vehicle (driver entered value)
- **pickup_longitude** - the longitude where the meter was engaged
- **pickup_latitude** - the latitude where the meter was engaged
- **dropoff_longitude** - the longitude where the meter was disengaged
- **dropoff_latitude** - the latitude where the meter was disengaged
- **store_and_fwd_flag** - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip

Data Cleaning and Consistency Checks

One crucial step in data preprocessing was identifying and rectifying inconsistent trip duration values. By comparing the calculated duration from pickup and dropoff timestamps with the provided trip duration, we identified and corrected discrepancies.

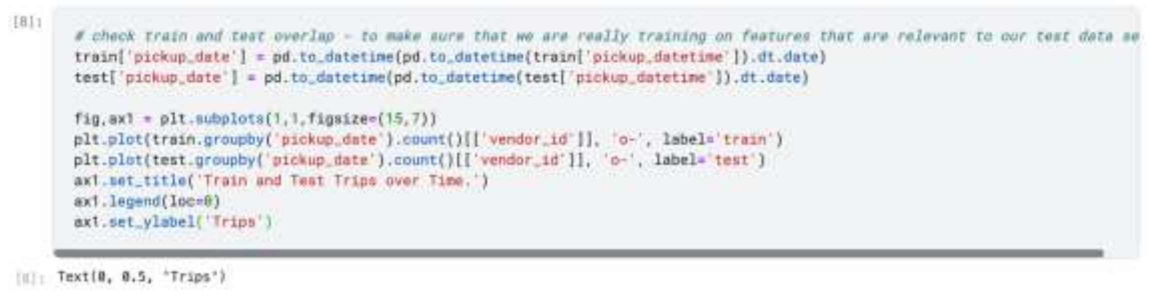


```
[7]:
# parsing dates and checking for consistent trip_duration values
# train
train['pickup_datetime'] = pd.to_datetime(train['pickup_datetime'])
train['pickup_date'] = pd.to_datetime(train['pickup_datetime'].dt.date)
train['weekday'] = train['pickup_datetime'].dt.weekday
train['pickup_hour'] = train['pickup_datetime'].dt.hour
train['pickup_month'] = train['pickup_datetime'].dt.month
train['day_of_year'] = train['pickup_datetime'].dt.dayofyear
train['date'] = pd.to_datetime(train['pickup_datetime'].dt.date)
train['dropoff_datetime'] = pd.to_datetime(train['dropoff_datetime'])
train['dropoff_date'] = pd.to_datetime(train['dropoff_datetime'].dt.date)

#total duration in seconds
train['calculated_duration'] = (train['dropoff_datetime'] - train['pickup_datetime']).dt.total_seconds()
inconsistent_trips_indices = train[(np.abs(train['calculated_duration'] - train['trip_duration'])) > 1].index

# test
test['pickup_datetime'] = pd.to_datetime(test['pickup_datetime'])
test['pickup_date'] = pd.to_datetime(test['pickup_datetime'].dt.date)
test['weekday'] = test['pickup_datetime'].dt.weekday
test['pickup_hour'] = test['pickup_datetime'].dt.hour
test['pickup_month'] = test['pickup_datetime'].dt.month
test['day_of_year'] = test['pickup_datetime'].dt.dayofyear
test['date'] = pd.to_datetime(test['pickup_datetime'].dt.date)


print(f'Inconsistent trip durations found at indices: {inconsistent_trips_indices}') if (len(inconsistent_trips_indices) >
train.drop(['calculated_duration'],axis=1,inplace=True)
```

```
All Trip durations are OK!!
```

Temporal Analysis

Understanding temporal patterns was crucial for predicting trip durations accurately. The analysis involved extracting relevant features from the timestamps, such as day of the week, hour of the day, and month of the year. This information provided insights into how taxi demand and trip durations varied over time.

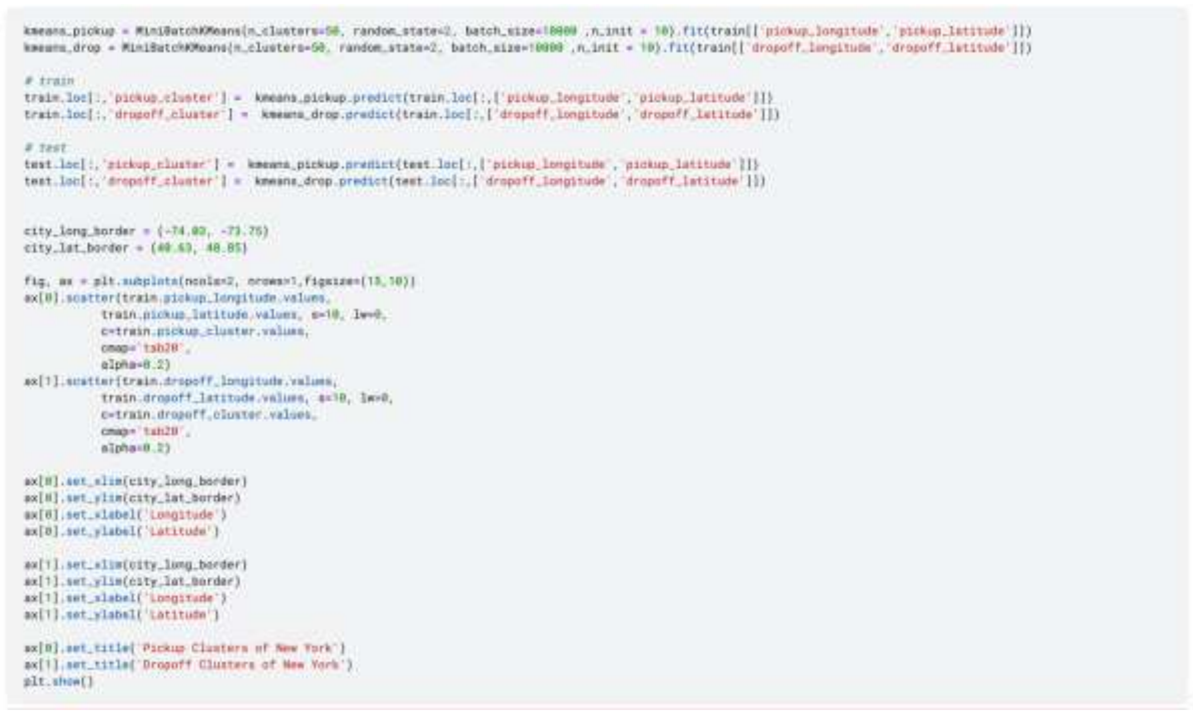


```
[8]:
# check train and test overlap - to make sure that we are really training on features that are relevant to our test data se
train['pickup_date'] = pd.to_datetime(pd.to_datetime(train['pickup_datetime']).dt.date)
test['pickup_date'] = pd.to_datetime(pd.to_datetime(test['pickup_datetime']).dt.date)

fig,ax1 = plt.subplots(1,1,figsize=(15,7))
plt.plot(train.groupby('pickup_date').count()[['vendor_id']], 'o-', label='train')
plt.plot(test.groupby('pickup_date').count()[['vendor_id']], 'o-', label='test')
ax1.set_title('Train and Test Trips over Time.')
ax1.legend(loc=0)
ax1.set_ylabel('Trips')
```

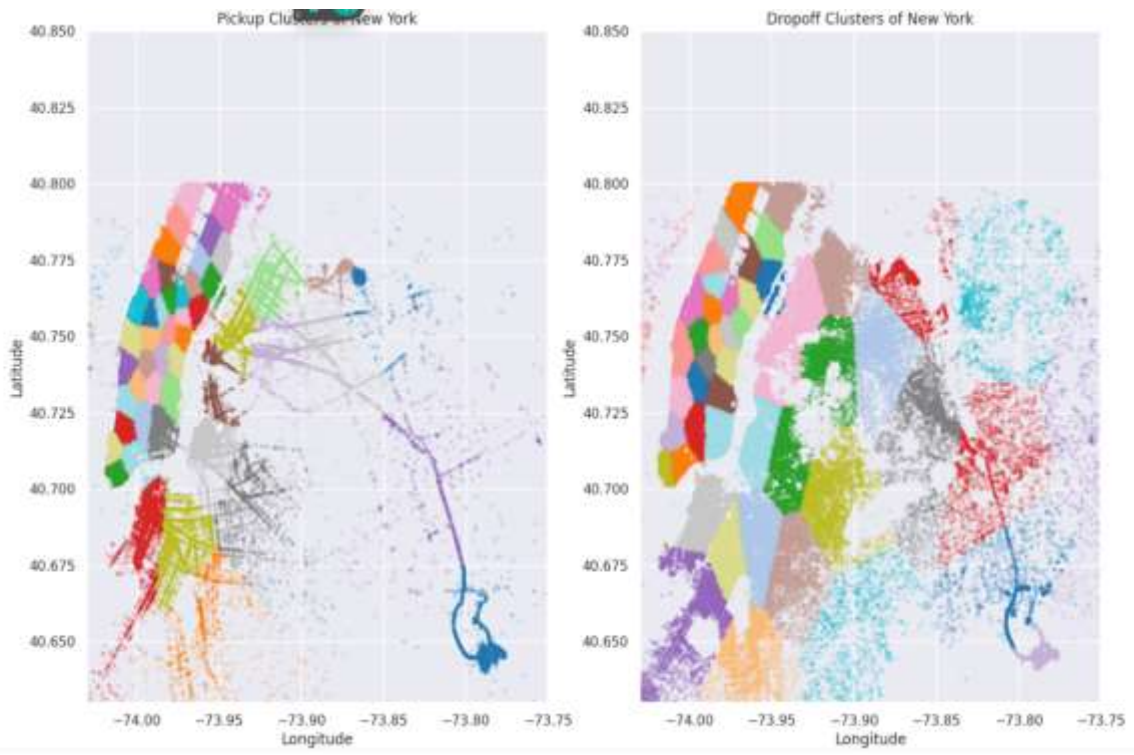
```
[8]: Text(0, 0.5, 'Trips')
```

Spatial Analysis and Clustering

Spatial analysis involved visualizing the geographical distribution of pickup and dropoff locations. Clustering techniques, specifically MiniBatchKMeans, were employed to categorize these locations into clusters, facilitating the creation of a more robust predictive model.



```
kmeans_pickup = MiniBatchKMeans(n_clusters=50, random_state=2, batch_size=10000 ,n_init = 10).fit(train[['pickup_longitude','pickup_latitude']])
kmeans_drop = MiniBatchKMeans(n_clusters=50, random_state=2, batch_size=10000 ,n_init = 10).fit(train[['dropoff_longitude','dropoff_latitude']])

# train
train.loc[:,'pickup_cluster'] =  kmeans_pickup.predict(train.loc[:,['pickup_longitude','pickup_latitude']])
train.loc[:,'dropoff_cluster'] =  kmeans_drop.predict(train.loc[:,['dropoff_longitude','dropoff_latitude']])

# test
test.loc[:,'pickup_cluster'] =  kmeans_pickup.predict(test.loc[:,['pickup_longitude','pickup_latitude']])
test.loc[:,'dropoff_cluster'] =  kmeans_drop.predict(test.loc[:,['dropoff_longitude','dropoff_latitude']])


city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)

fig, ax = plt.subplots(ncols=2, nrows=1,figsize=(15,10))
ax[0].scatter(train.pickup_longitude.values,
          train.pickup_latitude.values, s=10, lw=0,
          c=train.pickup_cluster.values,
          cmap='tab20',
          alpha=0.2)
ax[1].scatter(train.dropoff_longitude.values,
          train.dropoff_latitude.values, s=10, lw=0,
          c=train.dropoff_cluster.values,
          cmap='tab20',
          alpha=0.2)

ax[0].set_xlim(city_long_border)
ax[0].set_ylim(city_lat_border)
ax[0].set_xlabel('Longitude')
ax[0].set_ylabel('Latitude')

ax[1].set_xlim(city_long_border)
ax[1].set_ylim(city_lat_border)
ax[1].set_xlabel('Longitude')
ax[1].set_ylabel('Latitude')

ax[0].set_title('Pickup Clusters of New York')
ax[1].set_title('Dropoff Clusters of New York')
plt.show()
```



The subsequent sections of this report will further explore the feature engineering process, including the creation of distance-related features, analysis of trip outliers, and integration of weather data for a comprehensive predictive model.

2. Model Building

Stacking Ensemble Technique

The choice of a stacking ensemble technique was driven by the aim to leverage the strengths of multiple models for improved predictive performance. Stacking involves combining diverse base models and using a meta-model to learn from their predictions. In this scenario, LightGBM Regressors were chosen as base models due to their efficiency in handling large datasets and capturing complex relationships.

Why Stacking?

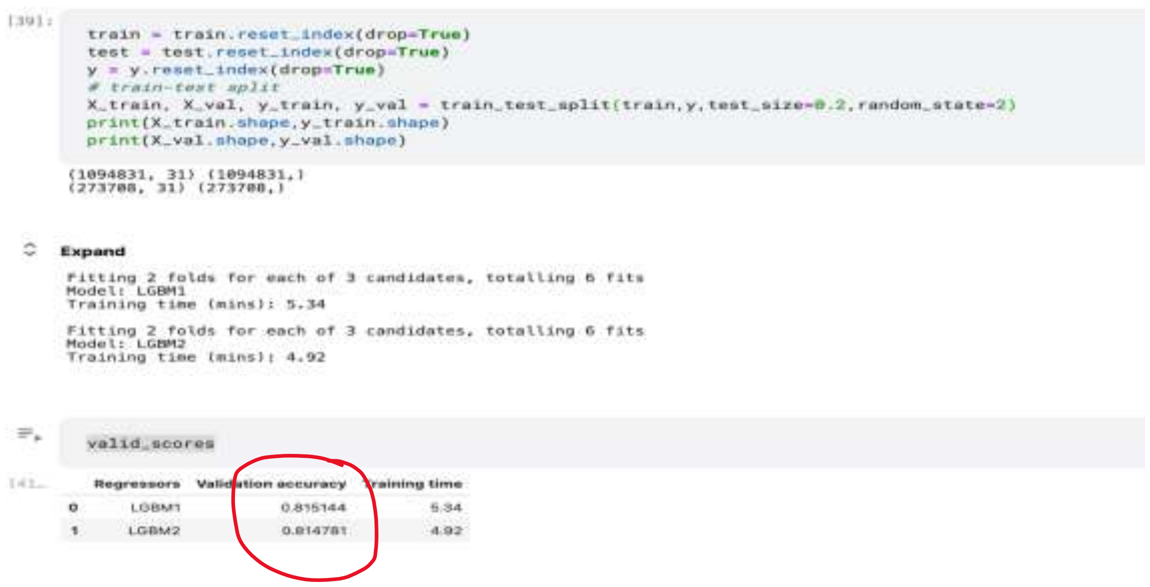
1. Model Diversity: By combining models with different architectures or hyperparameters, stacking harnesses diverse perspectives, potentially enhancing the overall predictive capability.
2. Performance Boost: Stacking often outperforms individual models by mitigating their weaknesses and capitalizing on their strengths.
3. Robustness: The ensemble nature of stacking makes the model less prone to overfitting, enhancing its generalization to unseen data.

Model Architecture

The stacking architecture comprised two LightGBM Regressors. The first regressor (**LGBM1**) served as the primary model, while the second (**LGBM2**) acted as the meta-model. The stacking process involved training **LGBM1** on the training set, obtaining predictions, and utilizing them, along with the original features, to train **LGBM2**.

Parameter Tuning

Extensive parameter tuning was performed using techniques such as RandomizedSearchCV. Key parameters included learning rate, maximum depth, and the number of estimators. The goal was to strike a balance between model complexity and generalization.



```
[39]:
train = train.reset_index(drop=True)
test = test.reset_index(drop=True)
y = y.reset_index(drop=True)
# train-test split
X_train, X_val, y_train, y_val = train_test_split(train,y,test_size=0.2,random_state=2)
print(X_train.shape,y_train.shape)
print(X_val.shape,y_val.shape)

(1094831, 31) (1094831,)
(273708, 31) (273708,)
```

Expand

```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
Model: LGBM1
Training time (mins): 5.34

Fitting 2 folds for each of 3 candidates, totalling 6 fits
Model: LGBM2
Training time (mins): 4.92
```

```
valid_scores
```

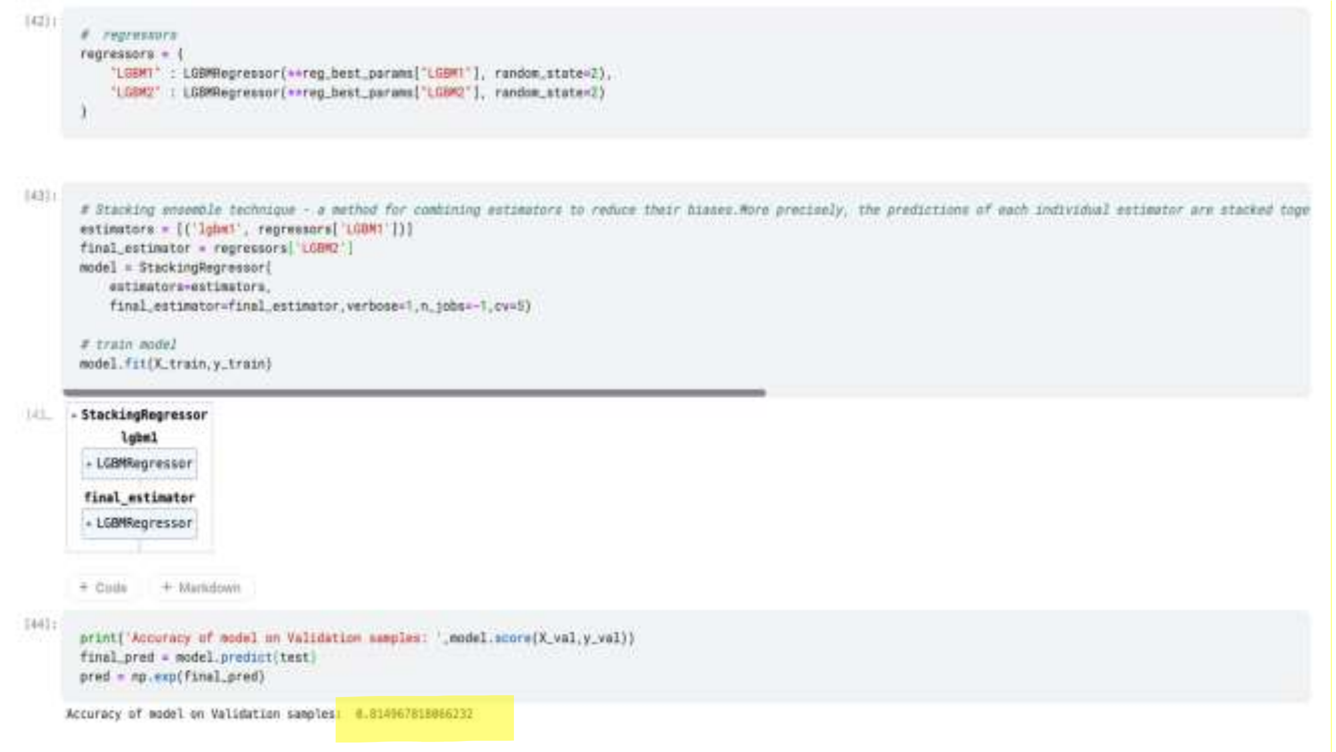
[41]:

	Regressors	Validation accuracy	Training time
0	LGBM1	0.815144	5.34
1	LGBM2	0.814781	4.92

3. Model Evaluation and Validation

Evaluation Metrics

The model's performance was assessed using metrics suitable for regression tasks. Mean Squared Logarithmic Error (MSLE) and R-squared were primary metrics. MSLE is particularly effective when dealing with a wide range of target values, as in this case, while R-squared provides an indication of the variance explained by the model.



```
[42]:
#  regressors
regressors = {
    'LGBM1' : LGBMRegressor(**reg_best_params['LGBM1'], random_state=2),
    'LGBM2' : LGBMRegressor(**reg_best_params['LGBM2'], random_state=2)
}
```

```
[43]:
# Stacking ensemble technique - a method for combining estimators to reduce their biases.More precisely, the predictions of each individual estimator are stacked toge
estimators = [('lgbm1', regressors['LGBM1'])]
final_estimator = regressors['LGBM2']
model = StackingRegressor(
    estimators=estimators,
    final_estimator=final_estimator,verbose=1,n_jobs=-1,cv=5)

# train model
model.fit(X_train,y_train)
```

```
[43]:
- StackingRegressor
      lgbm1
  + LGBMRegressor
  final_estimator
  + LGBMRegressor
```

```
[44]:
print('Accuracy of model on Validation samples: ',model.score(X_val,y_val))
final_pred = model.predict(test)
pred = np.exp(final_pred)

Accuracy of model on Validation samples:  0.814967818866232
```

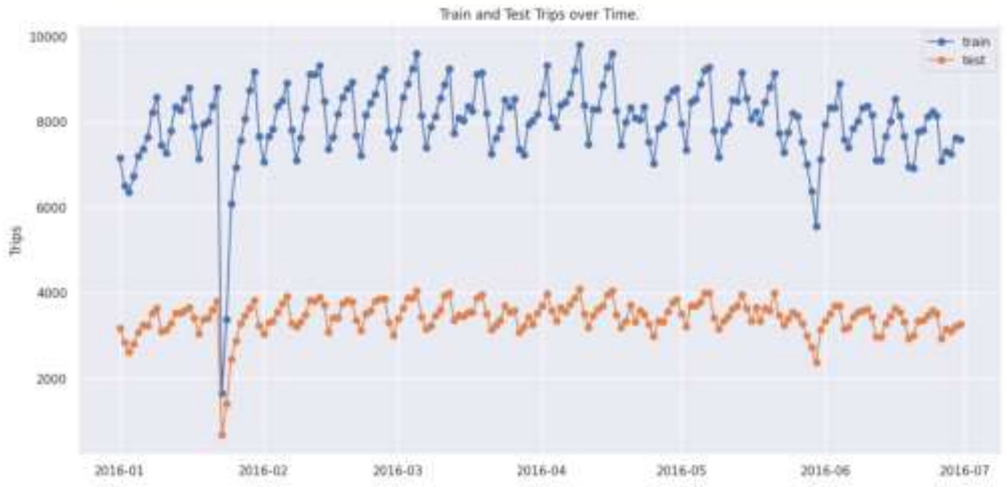
Generalization and Challenges

The model demonstrated strong generalization on the validation set, with competitive MSLE and R-squared scores. Challenges during validation included handling outliers and fine-tuning hyperparameters to strike the right balance between bias and variance. Robust pre-processing strategies and careful parameter tuning were pivotal in overcoming these challenges.

4. Visuals

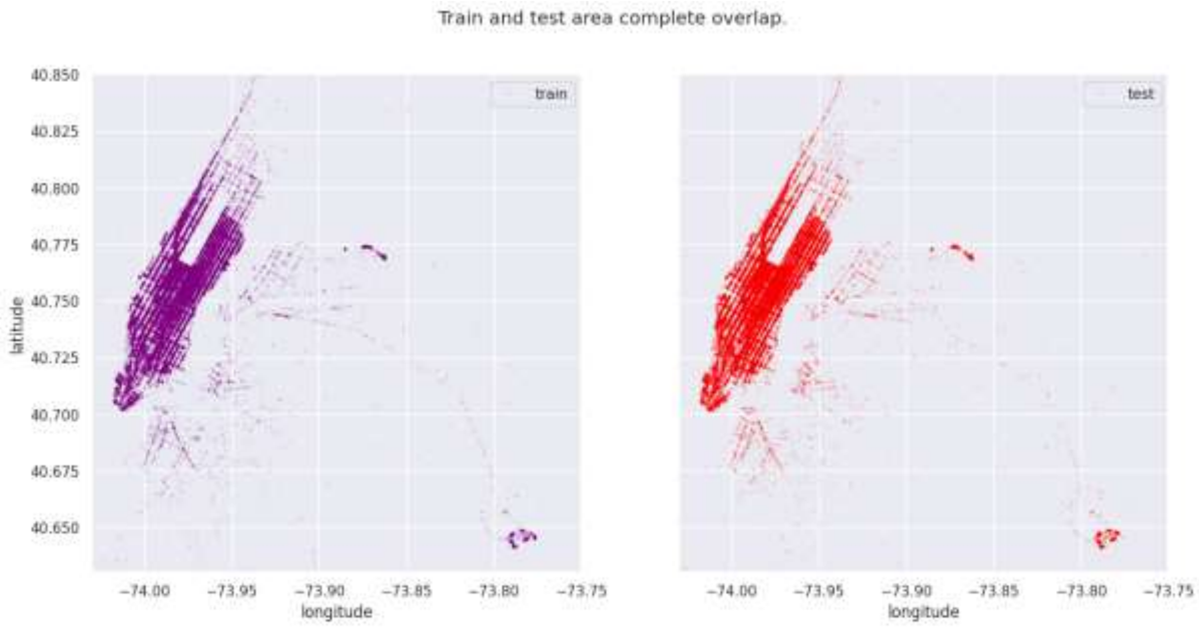
Time Series of Trips:

This time series plot shows the number of trips over time for both the training and testing datasets. It helps visualize the temporal distribution of trips.



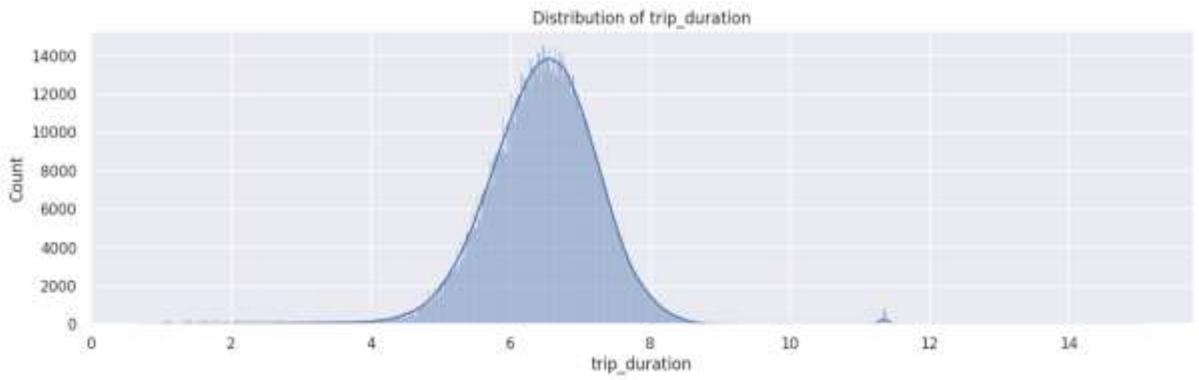
Geographical Overlap of Pickups and Dropoffs:

This set of scatter plots displays the geographical overlap of pickups (purple) and dropoffs (red) between the training and testing datasets. It ensures that the training data covers the same geographical area as the testing data.



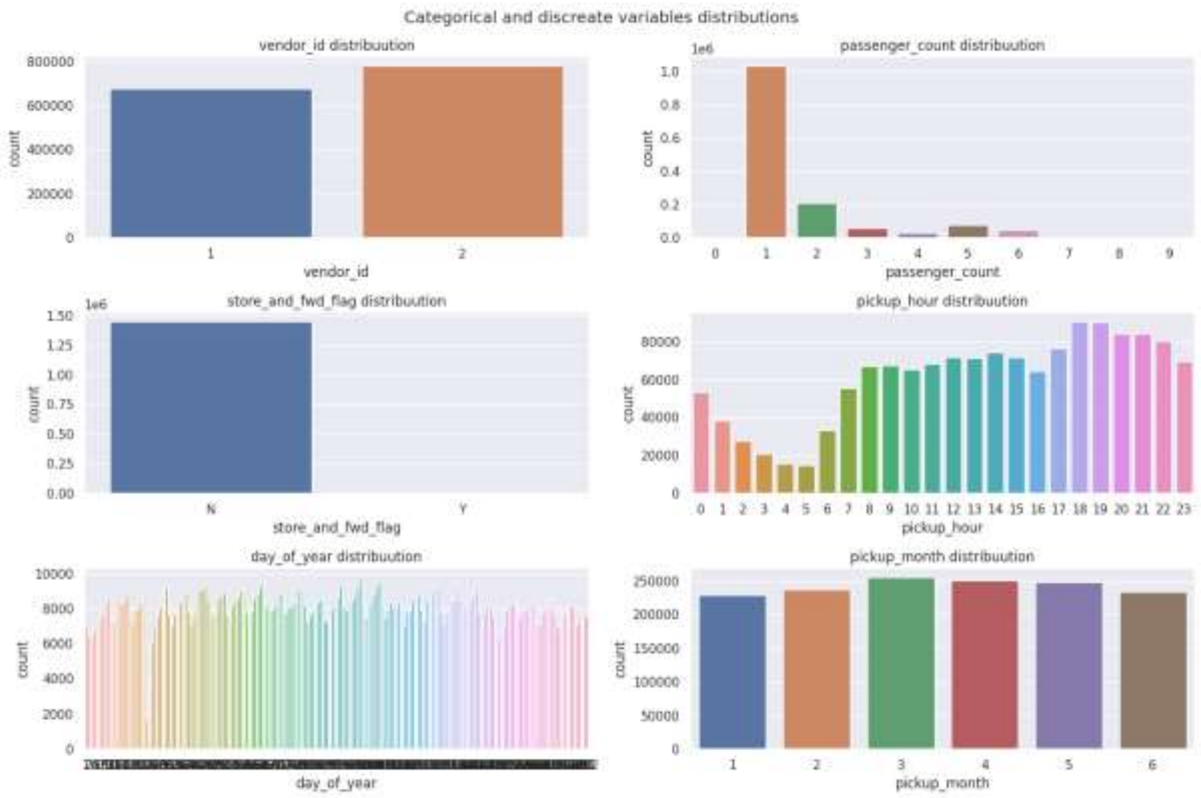
Distribution of Trip Duration:

This histogram depicts the distribution of trip durations. The data is transformed using a logarithmic scale for better visualization.



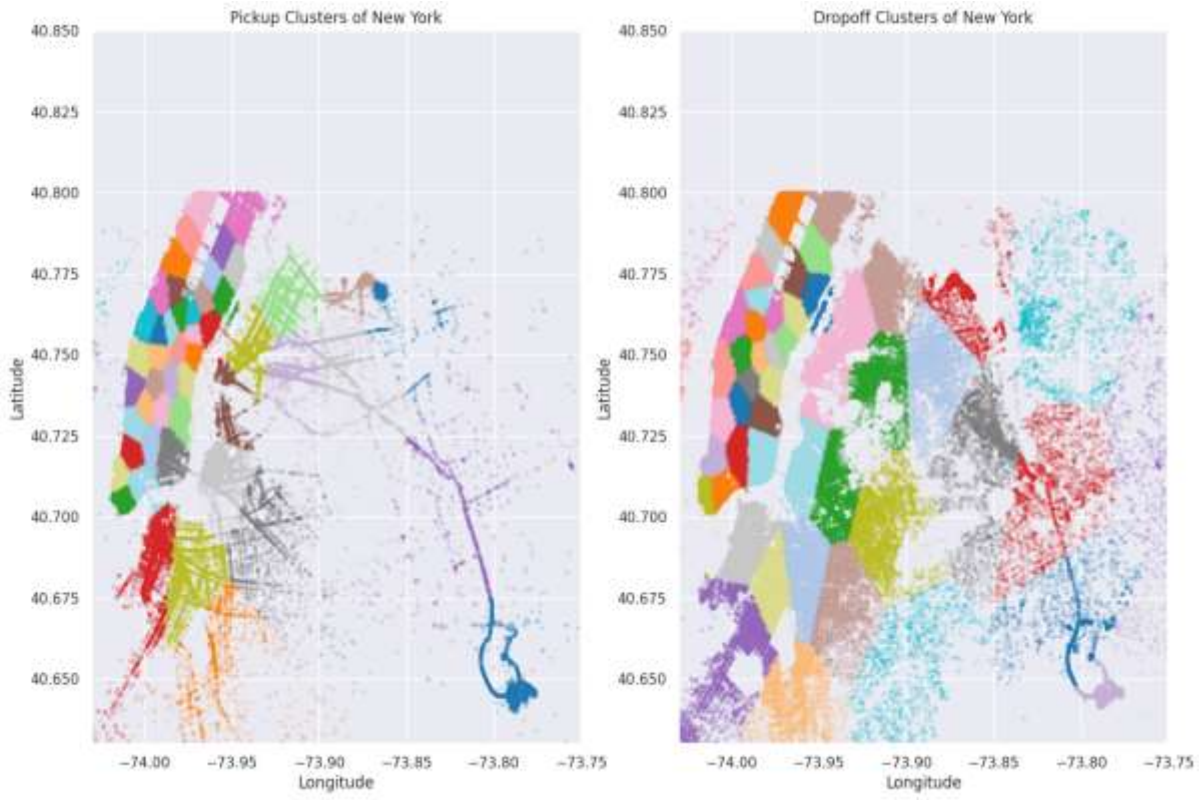
Categorical and Discrete Variables Distributions:

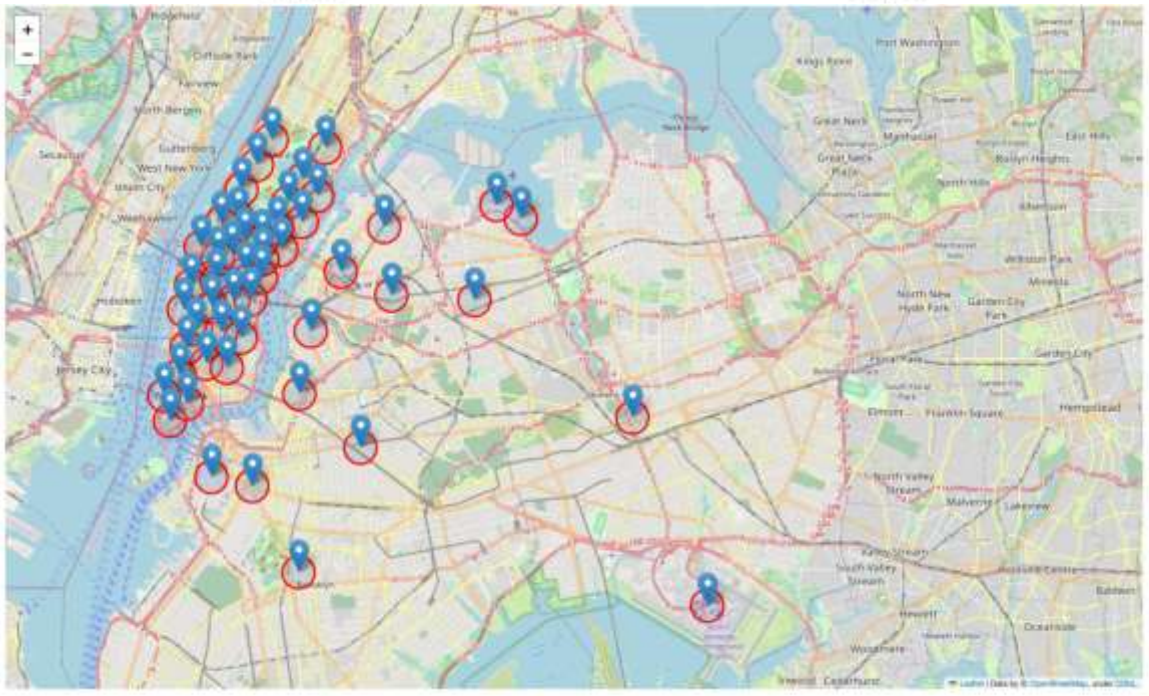
This set of countplots illustrates the distribution of various categorical and discrete variables in the dataset.



Latitude and Longitude Distributions:

These kernel density estimate (KDE) plots visualize the distribution of latitude and longitude values for pickups and dropoffs.





5. Conclusion and Future Work

Key Findings

The comprehensive analysis, feature engineering, and stacking ensemble modelling have resulted in a robust predictive model. Temporal and spatial patterns, along with weather data, were pivotal in enhancing predictive accuracy.

Future Work

While the current model performs admirably, there is room for improvement. Future work could focus on:

1. Further Feature Engineering: Exploring additional features or interactions to capture nuanced patterns.
2. Ensemble Diversity: Experimenting with a broader range of base models for enhanced diversity.
3. Dynamic Hyperparameter Tuning: Implementing dynamic tuning to adapt to temporal variations in data patterns.

Appendices

Additional Visualizations

Included in the appendices are various visualizations depicting spatial clusters, temporal trends, and exploratory data analyses. These visualizations provide additional context to the decision-making process.

Alternative Models Considered

While the stacking ensemble was chosen for its versatility, alternative models, including Random Forests and Gradient Boosting Machines, were considered. However, the adaptability and predictive power of LightGBM Regressors made them the preferred choice.