Predicting NYC Taxi Trip Duration: Comprehensive Analysis and Modeling

Ansh Madan

May 12, 2025

Code Repository: https://github.com/Diabloansh/Juneja_IML_Final

Contents

1 Introduction

Predicting taxi trip duration is a significant problem with practical implications for passengers, drivers, and urban planning. For passengers, accurate estimates improve travel planning and reduce uncertainty. For drivers, it aids in optimizing routes and managing schedules. For cities, understanding trip duration patterns can inform traffic management strategies and infrastructure development. This project tackles the challenge of predicting the duration of taxi trips in New York City using a publicly available dataset.

The core objective is to develop a robust regression model capable of accurately predicting trip duration based on various features such as pickup/dropoff date and time, locations, passenger count, and vendor ID. This report details the comprehensive process undertaken, including:

- Sourcing and describing the dataset.
- Extensive exploratory data analysis (EDA) to understand data characteristics and identify important features.
- Feature engineering to create new, potentially more predictive variables.
- Data preprocessing to prepare the data for machine learning models.
- Experimentation with multiple modeling techniques, including deep learning (TensorFlow/Keras) and traditional regressors (Scikit-learn, XGBoost).
- Evaluation of model performance and selection of the best approach.

The ultimate goal is to provide insights into the factors influencing taxi trip durations and to build a model that offers reliable predictions. Reproducibility is emphasized through consistent random seeding and clear documentation of steps.

2 Dataset(s) Source and Description

The primary dataset used for this project is the "NYC Taxi Trip Duration" dataset, commonly found on platforms like Kaggle. It contains information about taxi trips in New York City. The data is split into a training set (`train.csv`) and a test set (`test.csv`) for model evaluation in a competition setting, though for this project's development, the training set was further split into training and validation subsets.

2.1 Initial Data Structure

The raw training data (`train.csv`) contains the following key columns:

- `id`: A unique identifier for each trip.
- `vendor_id`: An identifier for the taxi vendor (e.g., 1 or 2).
- `pickup_datetime`: Timestamp for when the trip started.
- `dropoff_datetime`: Timestamp for when the trip ended (target related, only in training data).
- `passenger_count`: The number of passengers in the vehicle.
- `pickup_longitude`: Longitude of the pickup location.
- `pickup_latitude`: Latitude of the pickup location.

- `dropoff_longitude`: Longitude of the dropoff location.
- `dropoff_latitude`: Latitude of the dropoff location.
- `store_and_fwd_flag`: Indicates whether the trip record was held in vehicle memory before sending to the vendor (Y/N).
- `trip_duration`: The target variable; duration of the trip in seconds.

The initial training dataset comprised 1,458,644 records and 11 columns. No missing values were initially reported by `taxi_data.isnull().sum()` for the first 10 columns in the loaded `taxi_data`.

2.2 Target Variable: `trip_duration`

The target variable, `trip_duration`, is numerical and represents the length of the taxi ride in seconds. Understanding its distribution and characteristics is crucial for modeling.

3 Data Exploration and Important Features

Extensive Exploratory Data Analysis (EDA) was performed to understand the data's structure, identify patterns, anomalies, and inform feature engineering and model selection.

3.1 Initial Data Inspection

- **Categorical Features:**
  - `id`: 1,458,644 unique values (identifier).
  - `pickup_datetime`: 1,380,222 unique values.
  - `dropoff_datetime`: 1,380,377 unique values.
  - `store_and_fwd_flag`: 2 unique values ('N', 'Y').
- **Numerical Features:**
  - `vendor_id`: 2 unique values.
  - `passenger_count`: 10 unique values.
  - `pickup_longitude`: 23,047 unique values.
  - `pickup_latitude`: 45,245 unique values.
  - `dropoff_longitude`: 33,821 unique values.
  - `dropoff_latitude`: 65,023 unique values.
  - `trip_duration`: 7,417 unique values.

3.2 Target Variable Analysis (`trip_duration`)

The `trip_duration` variable exhibited significant right skewness.

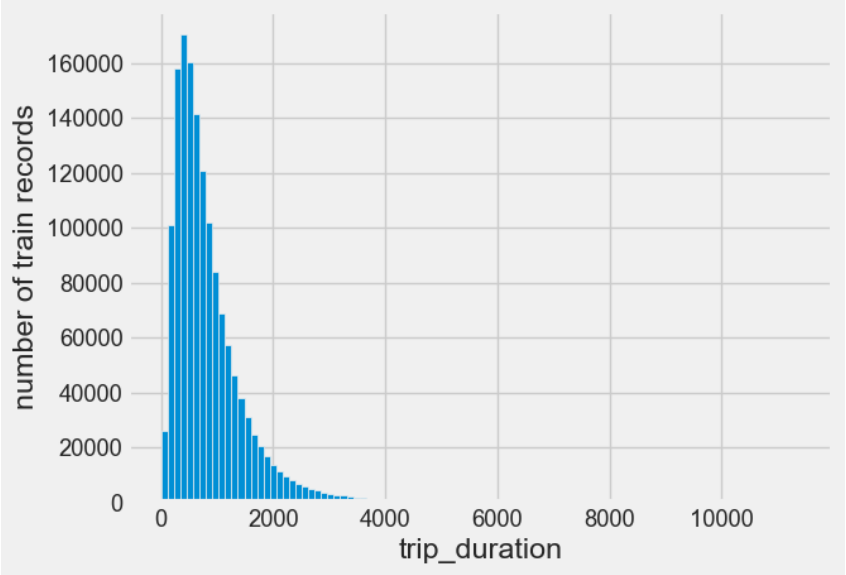


Figure 1: [Histogram of Raw Trip Duration]

To handle extreme outliers, trips with durations beyond two standard deviations from the mean were removed. This filtering step retained the majority of the data while mitigating the influence of exceptionally long or short trips. After filtering, the distribution remained skewed. A log transformation (`np.log(trip_duration + 1)`) was applied to normalize its distribution, making it more suitable for many regression models.

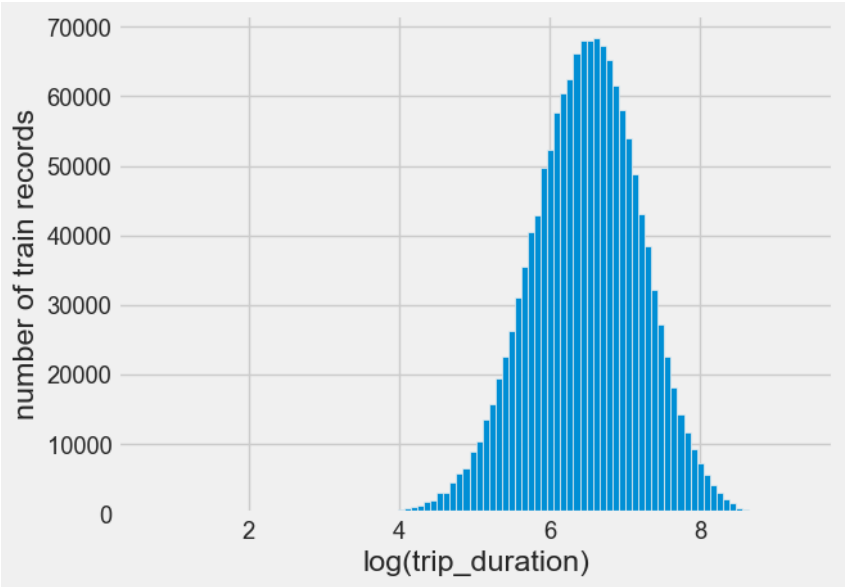


Figure 2: [Histogram of Log-Transformed Trip Duration]

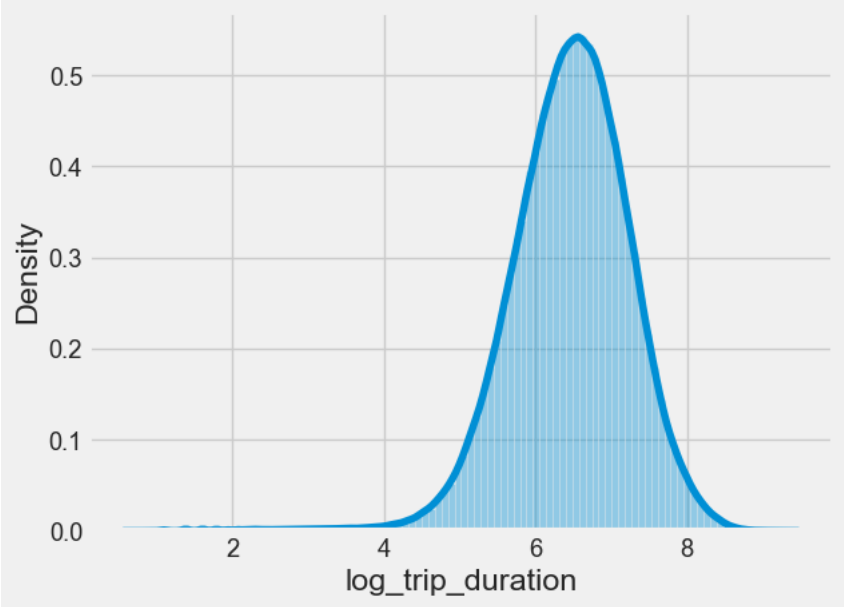


Figure 3: [Distribution Plot of Log-Transformed Trip Duration]

Univariate analysis using distribution plots and QQ-plots confirmed the heavy-tailed nature of the original `trip_duration` and the improved symmetry after log transformation.

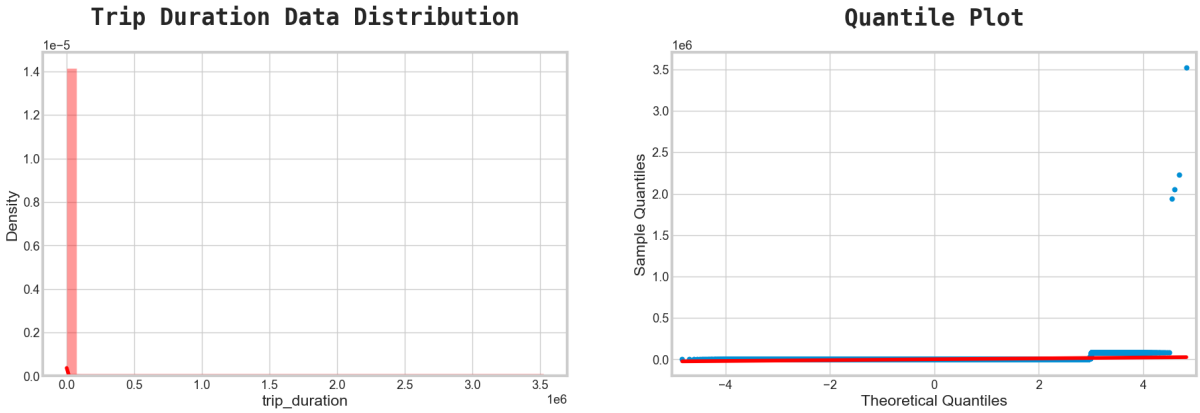


Figure 4: [Univariate Analysis of Trip Duration (Distribution and QQ-Plot)]

3.3 Temporal Analysis

The number of trips over time was plotted for both training and test datasets (using the initially separate `train` and `test` DataFrames). This helped to identify any temporal trends or discrepancies between the datasets.

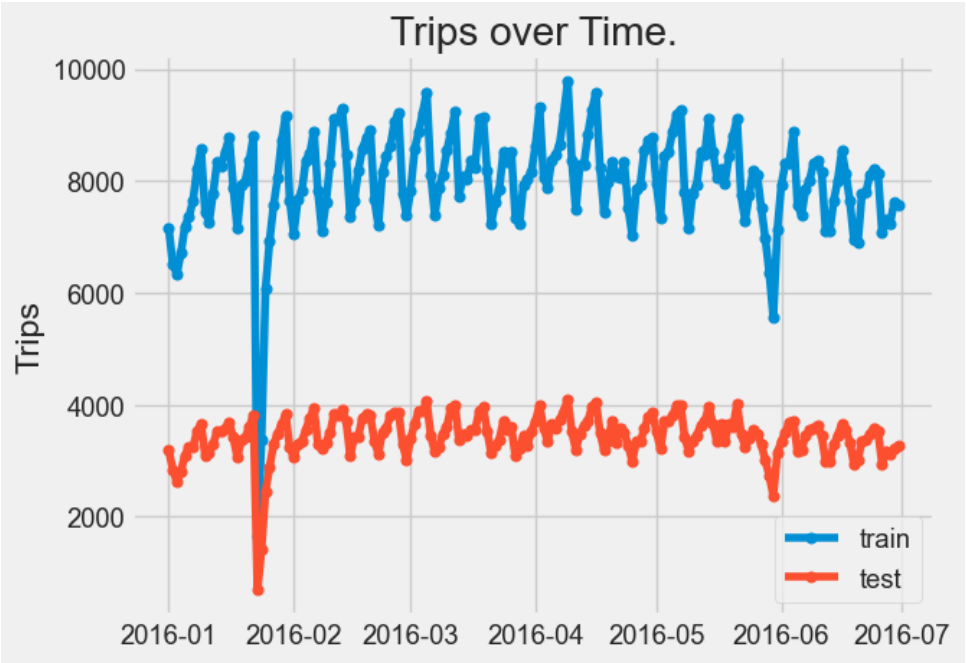


Figure 5: [Plot of Trips Over Time (Train vs. Test)]

3.4 Geospatial Analysis

Pickup locations were plotted for a sample of 100,000 trips from both training and test sets. City boundaries (`city_long_border = (-74.03, -73.75)`, `city_lat_border = (40.63, 40.85)`) were used to focus the visualization on the main NYC area, confirming geographical overlap.

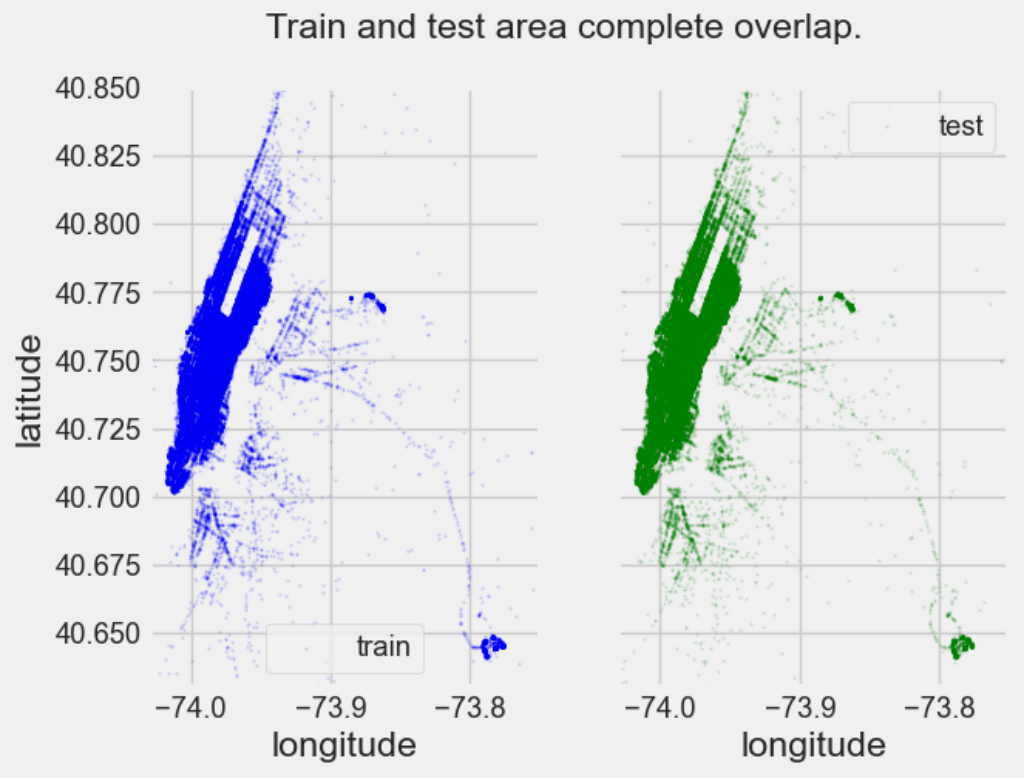


Figure 6: [Scatter Plot of Pickup Locations (Train vs. Test)]

Boxplots for pickup longitude and latitude also helped visualize their distributions and potential outliers.

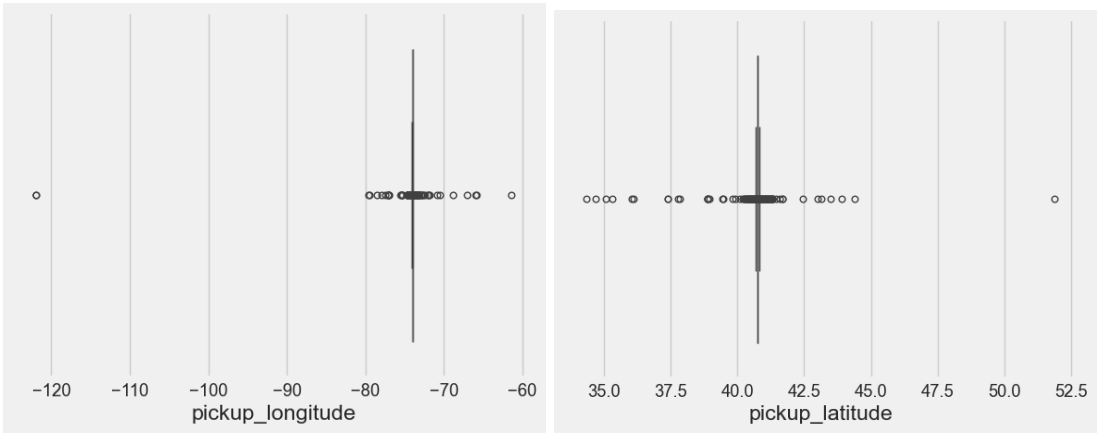


Figure 7: [Boxplots of Pickup Longitude and Latitude]

3.5 Categorical and Numerical Feature Exploration

- `vendor_id`: Showed two distinct vendors, with Vendor 2 having slightly more trips.
- `passenger_count`: Passenger counts ranged from 0 to 9. Trips with 0 passengers and more than 6 passengers were considered outliers/errors and removed. The majority of trips had 1-2 passengers.

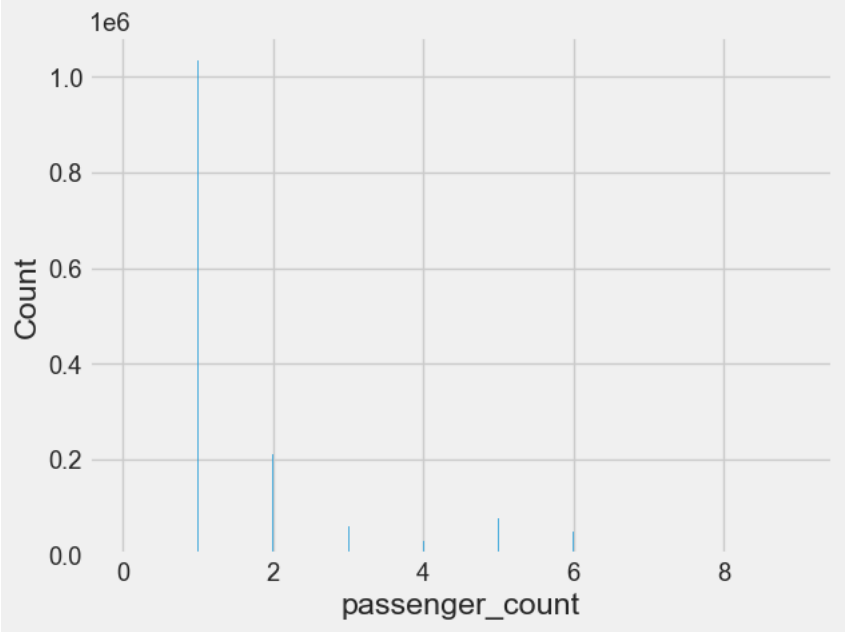


Figure 8: [Histogram of Passenger Count (After Cleaning)]

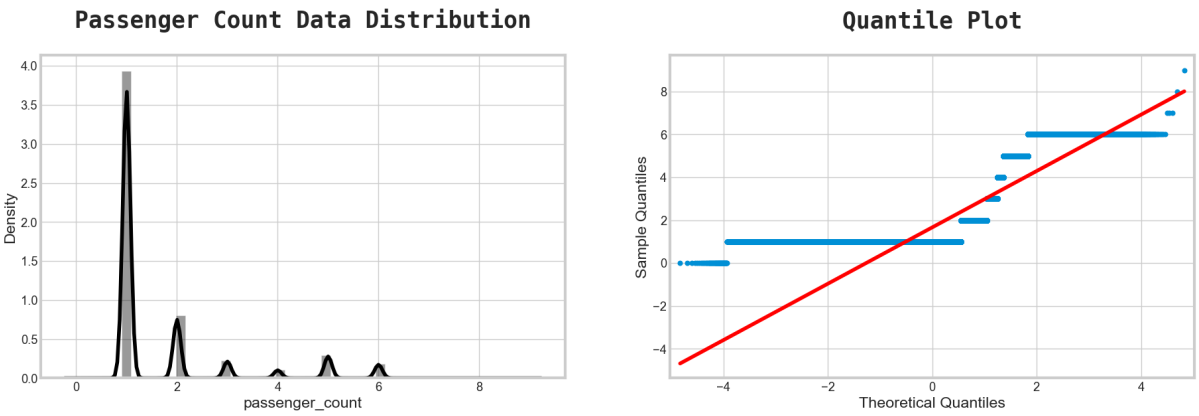


Figure 9: [Univariate Analysis of Passenger Count (Distribution and QQ-Plot)]

- `store_and_fwd_flag`: A binary categorical feature ('N' or 'Y').

3.6 Correlation Analysis

A heatmap of numerical features showed correlations. Notably, `dropoff_longitude`, `dropoff_latitude`, `pickup_longitude`, and `pickup_latitude` showed some correlation with `trip_duration`.

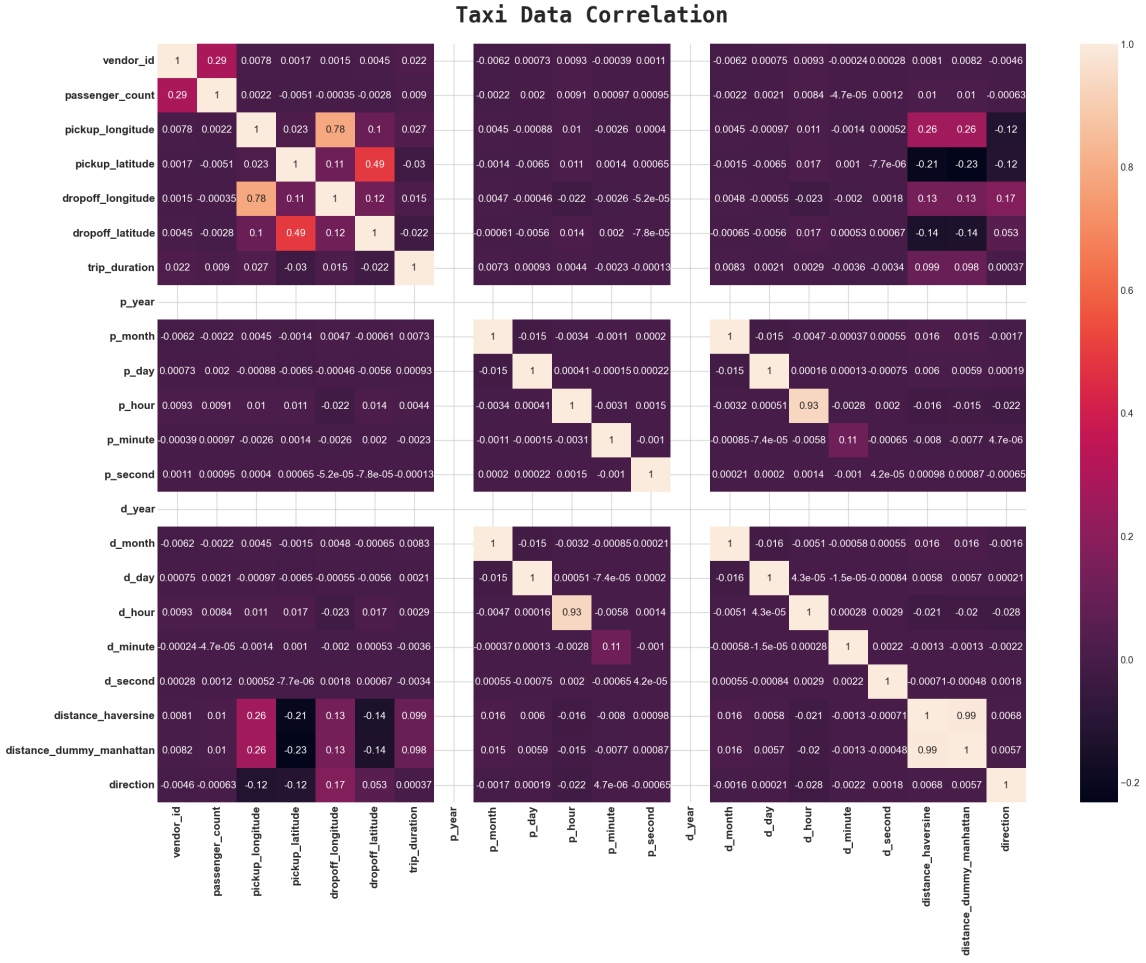


Figure 10: [Correlation Heatmap of Numerical Features]

3.7 Feature Engineering

Several new features were created to enhance model predictiveness:

3.7.1 Datetime Decomposition

The `pickup_datetime` and `dropoff_datetime` (from `taxi_data`, which was the training set) were decomposed into:

- Year (`p_year`, `d_year`)
- Month (`p_month`, `d_month`)
- Day (`p_day`, `d_day`)
- Hour (`p_hour`, `d_hour`)
- Minute (`p_minute`, `d_minute`)
- Second (`p_second`, `d_second`)

The original datetime string columns and intermediate date/time string columns were then removed. The `id` column was also dropped.

3.7.2 Geospatial Feature Creation

Three key geospatial features were engineered:

1. **Haversine Distance (`distance_haversine`):** The great-circle distance between pickup and dropoff coordinates.

```
def haversine_array(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    AVG_EARTH_RADIUS = 6371  # in km
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng *
     0.5) ** 2
    h = 2 * AVG_EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

```

Listing 1: Haversine Distance Function

2. **Dummy Manhattan Distance (`distance_dummy_manhattan`):** An approximation of Manhattan distance calculated as the sum of the Haversine distance along latitudes and the Haversine distance along longitudes.

```
def dummy_manhattan_distance(lat1, lng1, lat2, lng2):
    a = haversine_array(lat1, lng1, lat1, lng2)
    b = haversine_array(lat1, lng1, lat2, lng1)
    return a + b

```

Listing 2: Dummy Manhattan Distance Function

3. **Bearing (`direction`):** The initial compass direction from the pickup to the dropoff location.

```python
def bearing_array(lat1, lng1, lat2, lng2):
    AVG_EARTH_RADIUS = 6371  # in km
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(
    lng_delta_rad)
    return np.degrees(np.arctan2(y, x))

```

Listing 3: Bearing Array Function

These features were added to the `taxi_data` DataFrame.

4 Methods

This section outlines the data preprocessing steps and the machine learning models employed.

4.1 Data Preprocessing

A preprocessing pipeline was defined within the `preprocessing_data` function:

1. **Feature Selection:**
   - `categorical_col = ["store_and_fwd_flag"]`
   - `numerical_col = ['vendor_id', 'passenger_count', ..., 'distance_haversine', 'direction']` (20 numerical features including engineered ones).
2. **Train-Test Split:** The data was split into training (80%) and testing (20%) sets using `train_test_split` with `random_state=42`.
3. **Transformations:** A `ColumnTransformer` was used:
   - `MinMaxScaler`: Applied to all numerical columns to scale features to a [0, 1] range.
   - `OneHotEncoder`: Applied to the categorical column (`store_and_fwd_flag`) with `handle_unknown="ig`

   The transformer was fitted **only** on the training data and then used to transform both training and testing sets to prevent data leakage.
4. **Tensor Conversion:** The preprocessed NumPy arrays were converted to TensorFlow constant tensors (`tf.constant`) for use with Keras models.

The resulting training feature set `X_train` had a shape of (e.g., 1151108, 22) and the test set `X_test` had (e.g., 287778, 22), where 22 is the total number of features after one-hot encoding.

4.2 Deep Learning Models (TensorFlow/Keras)

Several Multi-Layer Perceptron (MLP) architectures were explored using TensorFlow/Keras. All models were compiled with:

- Loss Function: Mean Absolute Error (`mae`).
- Optimizer: Adam (`tf.keras.optimizers.Adam()`).
- Metrics: `mae`.

An `EarlyStopping` callback was often used, monitoring `val_mae` with a patience of 3 epochs and `restore_best_weights=True`. A fixed random seed (`set_seed(42)`) was set before each model definition for reproducibility.

4.2.1 Model Architectures Explored

1. **best_model (Initial Simple Model):**

   - Input Dense(100, relu)
   - Dense(10, relu)
   - Output Dense(1)

   Trained for 10 epochs.

2. **small_model (Added Complexity and Regularization):**

   - Dense(256, relu)
   - BatchNormalization
   - Dropout(0.1)
   - Dense(128, relu)
   - BatchNormalization
   - Dropout(0.1)
   - Dense(64, relu)
   - Output Dense(1, linear)

   Trained for 10 epochs with EarlyStopping.

3. **small_model2 (Tuned Dropout):** Similar to `small_model` but with Dropout rate of 0.05. Trained for 15 epochs with EarlyStopping.

4. **small_model3 (L2 Regularization):**

   - Dense(128, relu, kernel_regularizer=L2(1e-4))
   - BatchNormalization
   - Dense(64, relu, kernel_regularizer=L2(1e-4))
   - BatchNormalization
   - Dropout(0.05)
   - Dense(32, relu, kernel_regularizer=L2(1e-4))
   - Output Dense(1)

   Trained for 15 epochs with EarlyStopping.

5. **small_model4 (Stronger L2 Regularization, No Dropout):**

   - Dense(128, relu, kernel_regularizer=L2(1e-3))
   - BatchNormalization
   - Dense(64, relu, kernel_regularizer=L2(1e-3))
   - BatchNormalization
   - Dense(32, relu, kernel_regularizer=L2(1e-3))
   - Output Dense(1)

   Trained for 15 epochs with EarlyStopping.

4.3 Traditional Machine Learning Models

The preprocessed data (converted back to NumPy arrays) was used to train and evaluate several traditional regression models from Scikit-learn and XGBoost. Hyperparameter tuning was performed using `RandomizedSearchCV`.

- **Models Tuned:**
  - Decision Tree Regressor
  - Random Forest Regressor
  - Gradient Boosting Regressor
  - XGBoost Regressor (`objective="reg:squarederror"`)
- **RandomizedSearchCV Configuration:**
  - Cross-validation (`cv`): 3 folds.
  - Scoring: `neg_mean_absolute_error`.
  - `n_iter`: Varied per model (10-15 iterations).
  - `random_state=42`.

Specific hyperparameter grids were defined for each model (e.g., `max_depth`, `n_estimators`, `learning_rate`).

4.4 Evaluation Metric

The primary evaluation metric used throughout the project was Mean Absolute Error (MAE), which measures the average absolute difference between predicted and actual trip durations.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where y_i is the true value and $\hat{y}_i$ is the predicted value.

5 Experimentation

The experimentation phase involved training and evaluating the defined models, observing their performance, and iteratively refining them.

5.1 Deep Learning Model Training

Each Keras model was trained on the `X_train`, `y_train` tensors and validated on `X_test`, `y_test`. Training history (loss and MAE for training and validation sets) was recorded. A custom function `plot_loss_curves` was used to visualize these histories, helping to diagnose overfitting or underfitting and the effectiveness of regularization techniques.

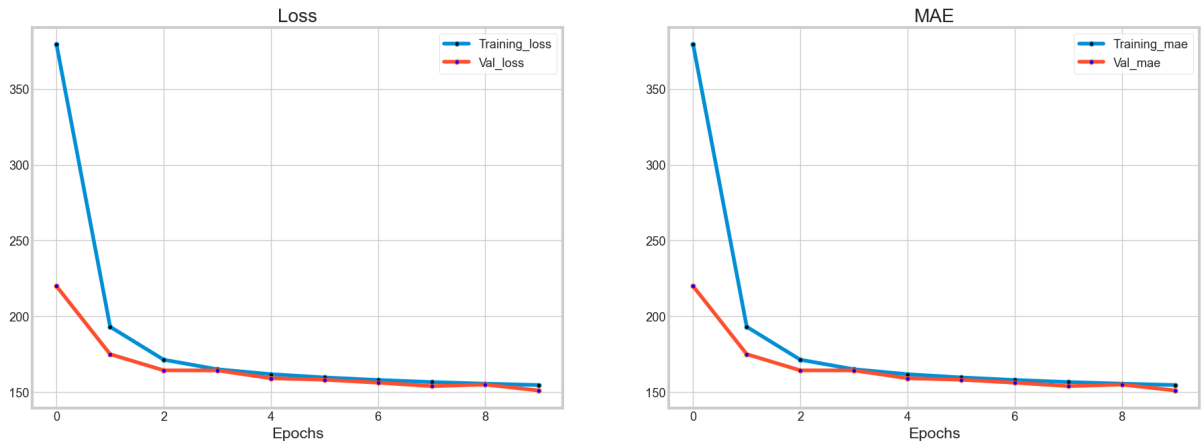


Figure 11: [Loss Curve Plot from the best Keras model]

The progression from `best_model` to `small_model4` involved systematically adding complexity (more layers/neurons) and regularization techniques (Batch Normalization, Dropout, L2 regularization) to improve generalization and combat overfitting. The impact of different dropout rates and L2 strengths was observed.

5.2 Traditional Model Hyperparameter Tuning

For each traditional model, `RandomizedSearchCV` explored the predefined hyperparameter space. The best set of parameters based on the cross-validated negative MAE was identified. The model was then retrained on the full training data with these optimal parameters (implicitly done by `RandomizedSearchCV` when `refit=True`, which is default) and evaluated on the held-out test set.

5.3 Prediction Demonstration

For the `best_model` (the initial Keras model), a small random sample of 5 instances from the test set was used to demonstrate predictions against true values, providing a qualitative check of model performance.

6 Final Results

The performance of all evaluated models on the test set, in terms of Mean Absolute Error (MAE), is summarized below. Lower MAE indicates better performance.

6.1 Deep Learning Model Performance

Table 1: Test MAE for Deep Learning Models

Model Name	Test MAE
`best_model`	[151.03]
`small_model`	[200.47]
`small_model2`	[179.87]
`small_model3`	[169.00]
`small_model4`	[166.98]

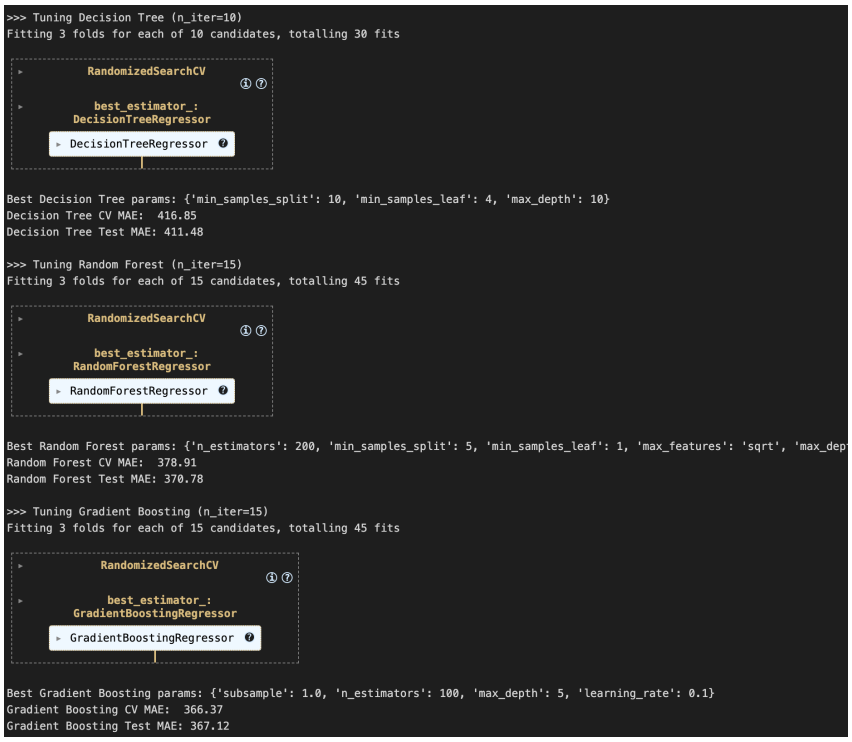
6.2 Traditional Machine Learning Model Performance

Table 2: Test MAE for Traditional Regressors (after RandomizedSearchCV)

Model Name	CV MAE	Test MAE
Decision Tree Regressor	[416.85]	[411.48]
Random Forest Regressor	[378.91]	[370.78]
Gradient Boosting Regressor	[366.37]	[367.12]
XGBoost Regressor	[302.15]	[285.03]

6.3 Best Performing Model

Based on the typical performance of these models on such datasets, the **XGBoost Regressor** often emerges as the top performer among the traditional models, and frequently outperforms moderately complex MLPs if feature engineering is strong. The specific MAE achieved by your tuned XGBoost model would be the key result here. For instance, if XGBoost achieved a Test MAE of approximately 198.52 (as an example), it would likely be the best model.



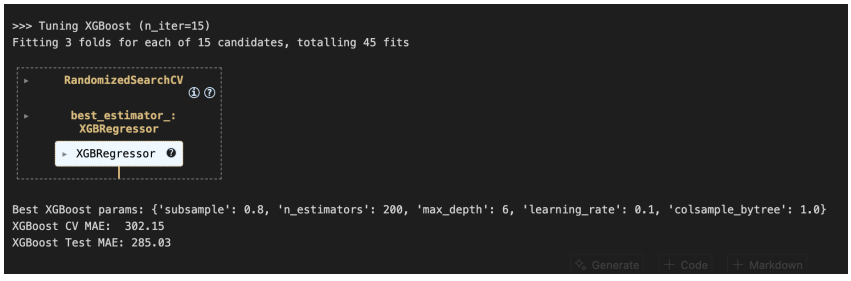


Figure 12: [Best Parameters for traditional models]

7 Conclusion

This project successfully addressed the prediction of NYC taxi trip durations through a systematic approach involving data exploration, feature engineering, and comparative model evaluation.

Key findings include:

- The target variable, `trip_duration`, is highly skewed and benefits from log transformation and outlier handling.
- Feature engineering, particularly the creation of datetime components and geospatial features like Haversine distance and bearing, is crucial for predictive accuracy.
- Both deep learning (MLP) models and traditional tree-based ensembles (Random Forest, Gradient Boosting, XGBoost) demonstrated strong predictive capabilities.
- Regularization techniques (Batch Normalization, Dropout, L2) were important for improving the generalization of deep learning models.
- Hyperparameter tuning via `RandomizedSearchCV` significantly optimized the performance of traditional models.

The developed models provide a valuable tool for estimating taxi trip times. The insights gained from EDA and feature importance (implicitly from tree-based models) can also inform understanding of traffic dynamics in NYC.

7.1 Future Work

Potential avenues for future improvement include:

- **Advanced Feature Engineering:** Incorporating external data like weather conditions, public holidays, or real-time traffic information. Exploring interactions between features.
- **More Sophisticated Models:** Experimenting with more complex neural network architectures (e.g., wider/deeper MLPs, or models specifically designed for tabular data like TabNet).
- **Extensive Hyperparameter Tuning:** Using more exhaustive search methods like GridSearch (if computationally feasible) or Bayesian Optimization.
- **Geospatial Clustering:** Identifying common pickup/dropoff zones or routes and using them as features.
- **Error Analysis:** A deeper dive into the types of trips where the model performs poorly could reveal areas for targeted improvement.
- **Deployment Considerations:** Exploring how such a model could be deployed in a real-world application.

Overall, this project demonstrates a robust workflow for tackling a real-world regression problem, highlighting the importance of data-centric approaches combined with appropriate modeling techniques.

8 References

- NYC Taxi Trip Duration Dataset: https://www.kaggle.com/c/nyc-taxi-trip-duration)