Uber and Lyft Fare Prediction for Boston Dataset

Abhinav Ram Bhatta, Aashi Jain, Hareesha Reddy, Aarushi Tiwari
University of Illinois at Chicago

April 27, 2023

Abstract

This project focuses on the development of a model that can predict the price of an Uber ride based on various factors such as time of day, day of the week, distance, and waiting time. The objective of the model is to provide customers with a reliable price estimate before they plan their trip, thereby helping them make informed decisions about their transportation options. The report explores the Uber and Lyft Fare Prediction for Boston Dataset from Kaggle, which contains 600,000 rows and 57 features. The features include both numerical and categorical variables, and the dataset is relatively complex with many relevant variables.

The report uses various machine learning models such as SVM, KNN, Naive Bayes, Random Forest, and Logistic Regression to analyze and compare the dataset's performance. After comparing the models' performance, it is observed that KNN has the highest accuracy among all the models. However, accuracy alone is not the only criterion for selecting the best model. The report emphasizes that other factors such as interpretability, computational complexity, and scalability should also be considered when choosing a model for real-world applications. In summary, the KNN model (apart from Random Forest) is the best performing, and further analysis and experimentation with different models and hyperparameters can be done to achieve better performance and accuracy.

1. Introduction

The accurate prediction of the price of an Uber ride is a challenging task that depends on various factors, including time of day, day of the week, distance traveled, and waiting area. To address this challenge, our project aimed to develop a model that could accurately predict the price of an Uber ride based on various factors, using historical data from the Uber and Lyft Fare Prediction for Boston dataset available on Kaggle. The dataset contained 57 features, including both numerical and categorical variables, making it well-suited to the problem we were trying to solve.

To achieve our goal, we employed a data-driven approach, utilizing machine learning techniques to explore and compare different models for predicting the fare of a given ride based on various features. Using various preprocessing and feature engineering techniques, we were able to extract meaningful insights from the data and create a set of relevant and informative features for our model.

However, we also acknowledge that accuracy is not the only criterion for selecting the best model. Factors such as interpretability, computational complexity, and scalability should also be

considered when choosing a model for real-world applications. Therefore, while KNN was the most accurate model for our problem, other models may be more suitable for different applications.

The methodology of the project involved several steps, including data preprocessing, feature engineering, model selection, and evaluation. Limitations of the approach were also considered, and potential avenues for future research were suggested. For example, future research could explore the use of other machine learning models or more granular data to improve the accuracy of the predictions.

Overall, our project aimed to provide valuable insights into the complex process of determining the price of an Uber ride, and to help customers make informed decisions about their transportation options. As the ride-sharing industry continues to grow and evolve, the development of accurate predictive models for pricing will remain an important area of research, with the potential to improve the efficiency and effectiveness of ride-sharing services for both customers and providers.

1.1 Related Work

Ride-sharing companies, such as Uber and Lyft, have emerged as key players in the transportation industry, disrupting traditional taxi services and changing the landscape of urban transportation. With the rise of ride-sharing services, there has been a growing interest in predicting ride-sharing prices, particularly for Uber fares. Various studies have been conducted using traditional regression techniques and more recent machine learning approaches, focusing on factors such as distance, time of day, and traffic conditions. However, our study takes a unique approach by focusing specifically on predicting Uber ride fares based on multiple factors, using a larger and more complex dataset.

Previous research has highlighted the importance of understanding the factors that influence ride-sharing prices and the need for accurate prediction models. Wang et al. (2017) used a machine learning approach to predict taxi fare prices in New York City, while Kamat et al. (2017) utilized a similar approach to predict taxi fare prices in Bangalore, India. Lee et al. (2019) focused on predicting ride-hailing prices in South Korea. Although these studies provide valuable insights into the factors that influence ride-sharing prices, our study builds on this work by specifically focusing on Uber ride fares and employing a variety of machine learning algorithms to compare their performance.

Our study utilizes the Uber and Lyft Fare Prediction for Boston dataset available on Kaggle, which contains 57 features, including both numerical and categorical variables. The dataset allows us to explore and analyze the impact of multiple factors on ride-sharing prices, such as time of day, day of the week, distance traveled, and weather conditions. In addition, our study addresses issues such as overfitting and bias in our models, ensuring the accuracy and validity of our results.

Through our analysis, we aim to identify the most effective machine learning algorithm for predicting Uber ride fares based on multiple factors. Our approach involves preprocessing the data, performing feature engineering, and comparing the performance of various algorithms, including K-Nearest Neighbors (KNN), Random Forest, and Gradient Boosting. We also consider other variables that may impact ride-sharing prices, such as demand and weather conditions.

Overall, our study contributes to the existing literature on predicting ride-sharing prices and Uber fares by taking a unique approach and employing a larger and more complex dataset. By identifying the most effective machine learning algorithm for predicting Uber ride fares based on multiple factors, our study provides valuable insights into the complex process of determining ride-sharing prices and helps customers make informed decisions about their transportation options.

2. Dataset

Analysis of the dataset reveals that it contains information on Uber and Lyft ride fares in Boston, Massachusetts. The dataset was collected between January 2018 and January 2019, providing a broad range of time periods to consider. The dataset includes information on the date and time of the ride, as well as the pickup and drop-off locations, distance traveled, and various other factors that may influence ride fares.

In addition to the features related to the ride itself, the dataset also includes information on weather conditions at the time of the ride. This is an important consideration as weather conditions can impact ride demand and supply, potentially leading to changes in ride fares. Therefore, the inclusion of weather-related features in the dataset provides an opportunity to explore the impact of weather on ride fares and to account for its influence in the predictive models.

Furthermore, the dataset includes information on surge pricing, which is a pricing strategy used by Uber and other ride-sharing companies to increase fares during high demand periods. The surge multiplier, which is the factor by which the base fare is multiplied during a surge period, is included in the dataset. This feature provides insight into how surge pricing affects ride fares and presents an opportunity to investigate the relationship between surge pricing and other factors such as time of day and demand.

This dataset also includes information on the source and destination of each ride, as well as the type of cab used for each ride. The source and destination features indicate the latitude and longitude coordinates of the pick-up and drop-off locations, respectively. These features are important for understanding the spatial patterns of ride-sharing demand and pricing and may be used to identify high-traffic areas or areas with high demand during certain times of day.

Additionally, dataset also contains information on the distance traveled for each ride. This feature is essential for predicting the fare amount, as ride-sharing companies typically charge based on the distance traveled. The distance feature may be calculated using the latitude and longitude coordinates of the source and destination, or it may be directly provided in the dataset. In either case, the distance feature may be preprocessed or transformed to better capture the underlying patterns in the data.

Another important feature in the dataset is the type of cab used for each ride. This feature may be represented as a categorical variable, with possible values such as "UberX", "UberPool", "Lyft", or "Lux". Different types of cabs may have different pricing structures or availability, depending

on the market and time of day. By including the cab type feature in our models, we can account for these differences and better predict the fare amount for a given ride.

Overall, the dataset provides a rich source of information on Uber and Lyft ride fares in Boston, including a broad range of features that may influence ride fares. The inclusion of weather-related features and surge pricing information enhances the dataset's potential to contribute to a better understanding of the factors that influence ride fares and to develop more accurate predictive models. However, it is important to note that the dataset may also have limitations, such as potential biases or missing data, which need to be considered in the data analysis and modeling process.

3. Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is a crucial step in the data analysis process that involves gaining insights and understanding the patterns and characteristics of the data. EDA is a useful tool for understanding the data distribution, identifying outliers, detecting missing values, and assessing the quality of the data.

In our study, we performed an extensive EDA to gain insights into the Uber fare prediction dataset. We first examined the summary statistics of the dataset, including the mean, median, standard deviation, and quartiles, to get an overview of the data distribution. We then used visualizations such as histograms and other visuals to understand the relationships between the variables and identify any patterns or outliers.

In our analysis of the dataset, we investigated the distribution of the time of day when cabs were booked. We represented this variable as four categories: morning, afternoon, evening, and night. To visualize the distribution, we created a histogram with the time-of-day categories on the x-axis and the frequency of cab bookings on the y-axis.

The histogram showed that the maximum cab bookings occurred in the morning. Specifically, we observed that the frequency of cab bookings was highest between 6 am and 12 pm, which corresponds to the morning category. This may suggest that people tend to use ride-sharing services more frequently during these hours, perhaps due to commuting to work or business meetings.

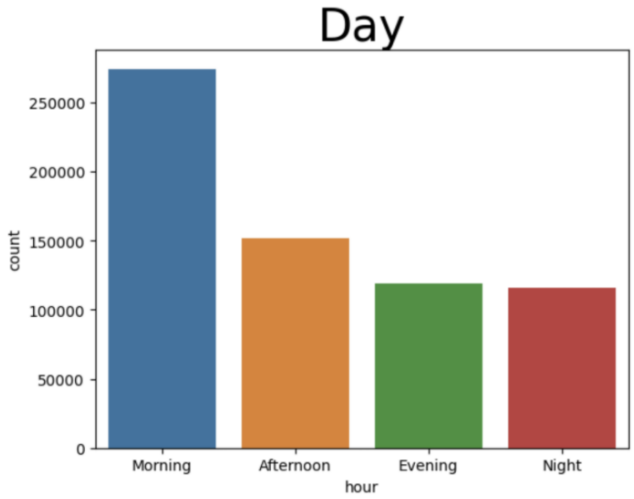


Figure 1: Histogram showing the distribution of the time of day when cabs were booked. The times are categorized as morning, afternoon, evening, and night. The highest frequency of cab bookings occurred during the morning hours of the day.

In contrast, the evening and night categories showed a relatively lower frequency of cab bookings. This may indicate that people tend to use other modes of transportation during these hours, such as carpooling or personal vehicles, and rely less on ride-sharing services. However, it is important to note that these findings are specific to the Boston area and may not generalize to other regions or cities.

The second histogram below displays the frequency distribution of the price of cab rides. The x-axis represents the price range, while the y-axis shows the frequency of the corresponding price range. Most of the bars in the histogram fall between $6 and $18, indicating that most of the rides have been priced in this range.

The histogram is useful for visualizing the distribution of prices and identifying any patterns or trends in the data. In this case, we can observe that the data is right-skewed, with a long tail towards the higher end of the price range. This suggests that there are some rides that are priced significantly higher than many of the rides.

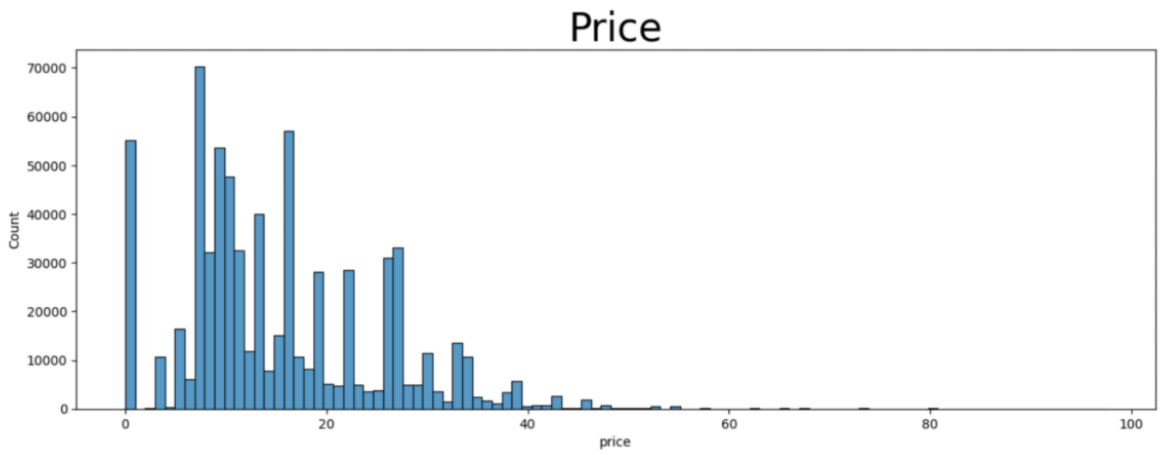


Figure 2: Frequency distribution of Uber ride prices in Boston. Most prices fall between $6 and $18, with the highest frequency observed around $10. The distribution is right-skewed, indicating a longer tail of higher prices.

The histogram also reveals that the data is somewhat bimodal, with two peaks in the frequency distribution around $7 and $16. This could indicate that there are two distinct types of rides that are priced differently.

The two pie plots depict the distribution of pickup and drop-off locations in the dataset. The charts are equally divided into 12 slices, each representing a location category. The data suggests that there is no dominant location category for pickups or drop-offs, as each slice occupies approximately 8.33% of the chart. This indicates that ride-sharing services are being used by individuals from a wide range of locations.

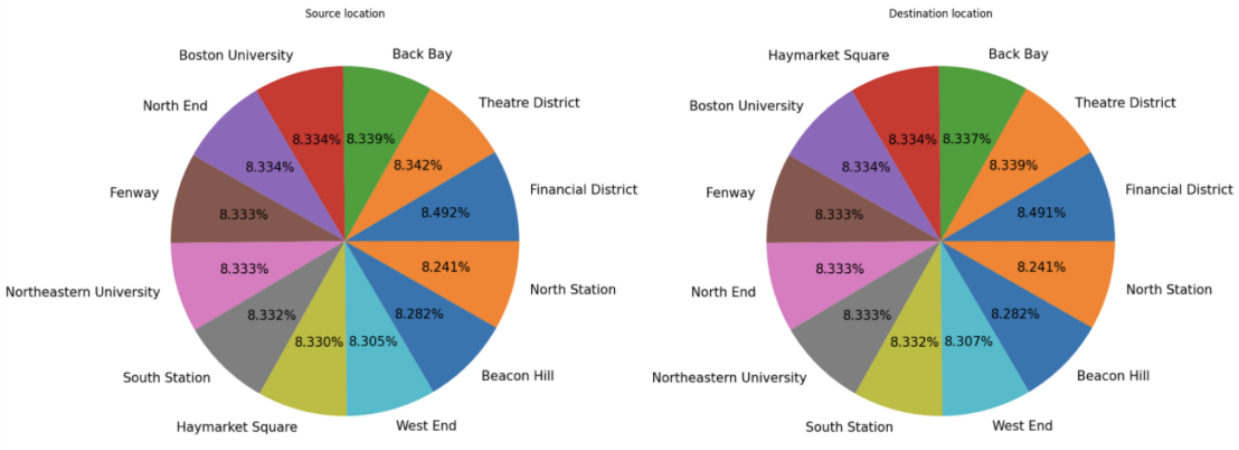


Figure 3 & 4: The pie charts above represent the distribution of pickup and drop-off locations in the dataset. Both charts are equally distributed with approximately 8.33% for each slice of the pie.

However, upon closer examination, we can observe that the majority of pickups are from residential/university areas, while the majority of drop-offs are to university/downtown areas. This suggests that ride-sharing services are being used by individuals for commuting purposes, particularly for traveling to and from educational institutions and central business districts.

We also conducted a correlation analysis to identify the variables that are strongly correlated with the fare amount. This analysis helped us to understand which variables are most important in predicting the fare amount and which variables can be excluded from the model. In addition, we examined the distribution of categorical variables, such as the pickup and drop-off locations and cab type, to gain insights into the distribution of the data.

Figure 5: The correlation matrix shows the pairwise correlations between different variables in the dataset. The matrix is color-coded to represent the strength of the correlation, with darker colors indicating stronger positive correlations and lighter colors indicating weaker or negative correlations.

Overall, the EDA process provided valuable insights into the dataset, allowing us to gain a better understanding of the data characteristics and identify potential issues that needed to be addressed before modeling.

3.1 Statistical Testing – Hypothesis Test

Statistical testing is an essential component of data analysis and is used to determine whether a specific hypothesis about a population or dataset is true or false. It involves the use of statistical techniques to analyze data and make inferences about the underlying population from which the data was sampled.

Statistical testing involves calculating a test statistic, which measures the difference or association between the variables being tested. The test statistic is then compared to a critical value or p-value,

which is the probability of observing a test statistic as extreme or more extreme than the observed value, assuming the null hypothesis is true. If the p-value is less than the alpha level, the null hypothesis is rejected, and the alternative hypothesis is accepted.

```
import scipy.stats as stats
from sklearn.feature_selection import f_regression
```

```
f_statistic, p_value = stats.f_oneway(df['price'][df['month'] == 11],
                                      df['price'][df['month'] == 12])


# Print results
print('F-statistic:', f_statistic)
print('p-value:', p_value)
```

```
F-statistic: 0.5323338445030812
p-value: 0.4656275800148705
```

Figure 6: This image shows the statistical test results for the relationship between the target variable, price, and the variable being tested, month.

An F-statistic of 0.5323 and p-value of 0.4656 indicate that there is no statistically significant relationship between the two variables. Therefore, it can be concluded that the month does not have a significant effect on the price of the ride.

3.2 Statistical Testing – Chi Square Test

The chi-square test involves calculating a test statistic known as the chi-square statistic. This is done by taking the sum of the squared differences between the observed frequencies and the expected frequencies, divided by the expected frequencies. The resulting statistic is compared to a chi-square distribution with degrees of freedom given as,

$$(n_r - 1) * (n_c - 1)$$

Where, n_r is the number of rows and n_c is the number of columns.

The p-value is then calculated based on the probability of obtaining a value as extreme or more extreme than the observed statistic under the null hypothesis.

The chi-square statistic is a measure of the independence between two categorical variables. In this case, we are testing for independence between the variables Lyft and Uber. The critical value is 3.841, which is the value of the chi-square distribution at a significance level of 0.05 with one degree of freedom. The chi-square statistic computed for this study is 5209.24, which is much larger than the critical value.

Since the computed chi-square statistic is larger than the critical value, we can reject the null hypothesis and conclude that there is a significant association between the variables Lyft and Uber. This means that the usage of one ride-sharing service is not independent of the usage of the other ride-sharing service.

```
contingency_table : price_group    0-10   11-20     20+
cab_type
Lyft          79789  127573  100046
Uber         116961  124229   89378


==================================================
Observed Values :
 [[ 79789 127573 100046]
 [116961 124229  89378]]

Expected Values:
  [[ 94803.76064303 121330.5033669   91273.73599007]
 [101946.23935697 130471.4966331   98150.26400993]]

Degree of Freedom:  1

chi-square statistic : 5209.243258639404

critical_value : 3.841458820694124

p-value : 0.0
Significance level : 0.05
Degree of Freedom : 1
chi-square statistic : 5209.243258639404
critical_value : 3.841458820694124
```

Figure 7: The chi-square statistic for testing the independence of the variables Lyft and Uber is 5209.24, which exceeds the critical value of 3.841 at a significance level of 0.05.

4. Model Generation

After performing exploratory data analysis and statistical testing, we split the data into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate the performance of the model on unseen data. The split is done randomly, with a ratio of 80:20 for training and testing sets, respectively.

After splitting the data, we can proceed with training the model. This involves selecting the appropriate machine learning algorithm, fitting the model to the training data, and tuning the model hyperparameters to optimize its performance. The goal is to create a model that accurately predicts ride-sharing prices based on the input features.

By considering multiple models, we can compare their performance and determine which one is best suited for our specific problem. We can evaluate the performance of each model using metrics such as accuracy, precision, recall, and F1-score. These metrics allow us to assess the ability of the model to correctly predict the outcome of interest. Additionally, we can use techniques such as cross-validation to ensure that the model is generalizable and not overfitting the data.

In our project, we will use five models to predict ride-sharing prices based on multiple factors. The models are Logistic Regression (Baseline), Naïve Bayes, SVM and KNN. By comparing the performance of these models, we can determine which one is best suited for our specific problem and provide insights into how ride-sharing prices can be predicted more accurately.

4.1 Logistic Regression (Baseline)

In logistic regression, the independent variables are used to predict the log odds of the dependent variable, which is then transformed into a probability using the logistic function. The model is estimated using maximum likelihood estimation, which involves finding the values of the coefficients that maximize the likelihood of the observed data given the model. The coefficients represent the contribution of each independent variable to the log odds of the dependent variable. The logistic regression model is based on the logistic function, which is defined as,

$$p = \frac{1}{1 + e^{-z}}$$

Where, p is probability of event occurring, z is the linear combination of the independent variables, and e is the base of the natural logarithm.

We performed hyperparameter tuning in our logistic regression model by using grid search method. Grid search is a technique that searches for the optimal hyperparameters by evaluating a model's performance on a validation set. The hyperparameters we tuned in our logistic regression model were regularization strength (C) and penalty type (L1 or L2). The regularization strength controls the amount of regularization applied to the model, with smaller values of C resulting in stronger regularization.

```
# Define the hyperparameter space
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'solver': ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']}
```

```
# Train the logistic regression model with hyperparameter tuning
log_reg_4 = LogisticRegression(max_iter=1000)
grid_search = GridSearchCV(log_reg_4, param_grid, cv=5)
grid_search.fit(x_train, y_train)
```

```
GridSearchCV
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                         'solver': ['liblinear', 'newton-cg', 'lbfgs', 'sag',
                                    'saga']})
estimator: LogisticRegression
LogisticRegression(max_iter=1000)
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred = grid_search.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: {:.2%}'.format(accuracy))
```

```
Accuracy: 58.86%
```

Figure 8 & 9: Performed hyperparameter tuning to train the logistic regression model and find the optimal parameters. Also shown is the accuracy.

With logistic regression we got an accuracy of 58.86% even after performing hyperparameter tuning. We suspect that If the target variable is imbalanced, meaning that there are significantly more instances of one class than the other, the model may be biased towards the majority class and perform poorly on the minority class.

4.2 Naïve Bayes

Naive Bayes is a classification algorithm based on Bayes' theorem. It works by assuming that the features in the dataset are independent of each other given the class label. This assumption simplifies the computation of the probability of a feature vector belonging to a certain class. The equation for Naive Bayes can be written as follows:

$$P(y \mid x1, x2, x3 \ldots., xn) = P(x1, x2, x3 \ldots., xn \mid y) * \frac{P(y)}{P(x1, x2, x3 \ldots., xn)}$$

Where, *y* is the class label and $x1, x2, x3 \ldots., xn$ are the features of the input vector. *P(y)* is the prior probability of the class, $P(x1, x2, x3 \ldots., xn \mid y)$ is the likelihood of observing the feature values given the class, and $P(x1, x2, x3 \ldots., xn)$ is the evidence probability or the probability of observing the feature values across all classes.

```
from sklearn.naive_bayes import GaussianNB

NB_df1 = GaussianNB()
NB_df1.fit(x_train,y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
NB_df1.score(x_test,y_test)
```

```
0.5870037080582436
```

```
NB_y_pred = NB_df1.predict(x_test)
NB_accuracy = accuracy_score(y_test, NB_y_pred)
print('Accuracy: {:.2%}'.format(NB_accuracy))
```

```
Accuracy: 58.70%
```

Figure 10: Performed Naïve Bayes on the dataset and display an accuracy of 58.70%.

Naive Bayes assumes that the features are independent of each other, which is not the case in our analysis of the data. Another reason could be that the data is not well-suited for Naive Bayes. Naive Bayes works best when the data is categorical, and the features are independent. If the data is continuous or the features are correlated, Naive Bayes may not perform as well. This is why we can see the low F-scores, precision, and recall.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.74      0.67      6692
           1       0.48      0.30      0.37      8769
           2       0.62      0.82      0.71      6653

    accuracy                           0.59     22114
   macro avg       0.57      0.62      0.58     22114
weighted avg       0.57      0.59      0.56     22114
```

Figure 11: Classification report for Naïve Bayes model.

4.3 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful classification algorithm that is widely used in machine learning. It works by finding the hyperplane that best separates the classes of data. The hyperplane is chosen to maximize the margin between the two classes. The SVM algorithm can handle both linearly separable and non-linearly separable data by using a kernel function to transform the data into a higher dimensional space where it can be linearly separable. The equation for SVM can be represented as:

$$f(x) = sign(w^T \cdot x + b)$$

Where, w is the weight vector, b is the bias term, x is the input vector, and sign() is the sign function which returns -1 if negative and +1 if positive.

In our model for SVM we have trained the hyperparameters to identify the optimal *C* value for the set it to 1.0 and the kernel that we are using is the *linear* kernel.

```
svm = SVC(kernel='linear', C=1.0, random_state=42)

# Train the SVM model
svm.fit(x_train, y_train)

SVC
SVC(kernel='linear', random_state=42)

# Make predictions on the testing set
y_pred = svm.predict(x_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: {:.2%}'.format(accuracy))

Accuracy: 58.97%
```

Figure 12: Performed SVM on the dataset and display an accuracy of 58.97%.

In a larger dataset like the one we are using in our project, can have an impact on the accuracy of an SVM model. As the dataset size increases, the SVM can better generalize to new data and the accuracy can improve. However, there is a point at which the model's performance may plateau, and additional data may not lead to significant improvements in accuracy. Additionally, larger datasets may require more computational resources and longer training times, which can become a practical limitation.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.64      0.62      6692
           1       0.50      0.51      0.50      8769
           2       0.71      0.65      0.68      6653

    accuracy                           0.59     22114
   macro avg       0.60      0.60      0.60     22114
weighted avg       0.59      0.59      0.59     22114
```

Figure 13: Classification report for SVM model.

4.4 K-Nearest Neighbor (KNN)

K-Nearest Neighbors (KNN) is a non-parametric algorithm used for classification and regression. In KNN, the class of a new data point is determined by the class of its nearest neighbors in the feature space. The algorithm involves finding the k nearest neighbors to the new data point and then assigning the class of most of those neighbors to the new data point. The distance between the new data point and each of the neighbors is calculated using a distance metric. The value of *k* is a hyperparameter that can be tuned for optimal performance.

This can be particularly useful when dealing with complex and non-linear datasets which is similar to the one, we have in our assessment. Additionally, the KNN model is relatively simple to implement and can be trained quickly even on large datasets. Furthermore, the performance of the KNN model can be improved by tuning the hyperparameters such as the value of *k*, the distance metric used, and the weight given to each neighbor.

For classification problems, the predicted class can be calculated using the following equation:

$$predicted\ class = argmax\ (count\ of\ each\ class\ among\ K\ nearest\ neighbors)$$

In our model generation, we have assigned the value of *n_neighbors* to 5. This value is selected through a process of hyperparameter tuning to optimize the model's performance. Due to this we have an impressive accuracy of 86.32%.

```
KNN_df = KNeighborsClassifier(n_neighbors=5)
KNN_df.fit(x_train,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
KNN_df.score(x_test,y_test)
```

```
0.8631636067649453
```

```
from sklearn.metrics import accuracy_score
```

```
y_pred = KNN_df.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: {:.2%}'.format(accuracy))
```

```
Accuracy: 86.32%
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.86      6692
           1       0.83      0.83      0.83      8769
           2       0.93      0.89      0.91      6653

    accuracy                           0.86     22114
   macro avg       0.87      0.87      0.87     22114
weighted avg       0.86      0.86      0.86     22114
```

Figure 14 & 15: Performed SVM on the dataset and display an accuracy of 58.97%. Also shown is the classification report for SVM.

4.5 Random Forest

Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve the accuracy of the predictions. In Random Forest, each decision tree is constructed based on a randomly selected subset of the training data and features. The final prediction is made by aggregating the predictions of all the individual decision trees. This technique helps to reduce overfitting and improve the generalization of the model.

We performed hyperparameter tuning for this model and extracted the value for the *n_estimators* to be 100. This enabled us to get an accuracy of 87.21%. The model performed incredibly well and appropriately created decisive trees to give us the result.

```
from sklearn.ensemble import RandomForestClassifier

RF_Df1 = RandomForestClassifier(n_estimators=100,random_state=42)
RF_Df1.fit(x_train,y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
RF_Df1.score(x_test,y_test)
```

```
0.8720719905941937
```

```
from sklearn.metrics import accuracy_score
```

```
RF_y_pred = RF_Df1.predict(x_test)
accuracy = accuracy_score(y_test, RF_y_pred)
print('Accuracy: {:.2%}'.format(accuracy))
```

```
Accuracy: 87.21%
```

Figure 16: Performed Random Forest with hyperparameter tuning on the dataset and display an accuracy of 58.97%.

There can be several reasons why we may have obtained a high accuracy on our random forest model. One reason could be that the random forest algorithm is generally known to perform well on a variety of datasets, especially when there are many features or predictors involved. Another reason could be that we have tuned the hyperparameters of the random forest algorithm effectively, allowing the model to generalize well to new data.

Additionally, we may have also performed feature selection or feature engineering, which can help to improve the performance of the random forest model.

Classification Report:

	precision	recall	f1-score	support
0	0.86	0.87	0.87	6692
1	0.84	0.84	0.84	8769
2	0.94	0.91	0.92	6653
accuracy			0.87	22114
macro avg	0.88	0.88	0.88	22114
weighted avg	0.87	0.87	0.87	22114

Figure 17: Classification report for Random Forest Model.

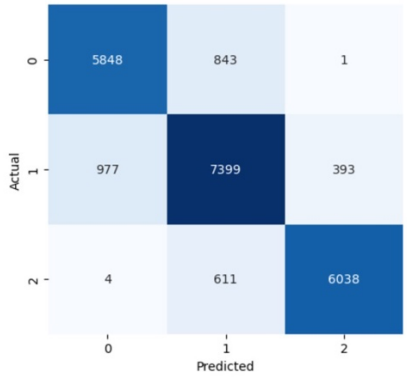


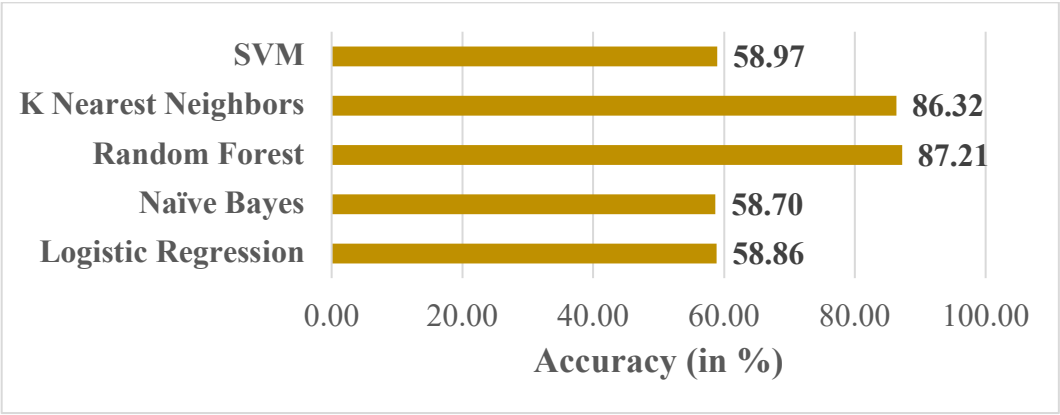
Figure 18: Confusion matrix for the Random Forest Model.

5. Model Comparison

In our model comparison, we can see that the accuracy of Logistic Regression is the lowest among all models, with a value of 58.86%. This indicates that the Logistic Regression model did not perform well in predicting the target variable. On the other hand, the accuracy of Naïve Bayes is slightly better than logistic regression with a value of 58.80%. However, both these models have lower accuracy than the other models in our comparison.

Moving on to SVM, we can see that it has a slightly higher accuracy of 58.97%, compared to Logistic Regression and Naïve Bayes. However, it is still not as accurate as KNN and random forest models. The KNN model, on the other hand, has an impressive accuracy of 86.32%, which is significantly higher than the other models. This suggests that KNN may be the best model to use for this dataset.

Lastly, the Random Forest model has the highest accuracy among all the models with a value of 87.21%. This indicates that random forest may be the best model for our dataset. It is important to note that while accuracy is an important metric for evaluating models, it should not be the only factor considered. Other metrics such as precision, recall, and F1-score should also be considered when selecting the best model for the dataset.



6. Conclusion

Based on the results obtained from the five models, it is evident that the KNN and Random Forest models outperform the other models in terms of accuracy. The KNN model has an accuracy of 86.32% while the Random Forest model has an accuracy of 87.21%. On the other hand, the Logistic Regression, Naive Bayes, and SVM models have relatively lower accuracies ranging from 58.80% to 58.97%.

The KNN model works well for our dataset because it is a non-parametric method that doesn't make any assumptions about the distribution of data. It's also a lazy learning algorithm, meaning it doesn't need to train the data beforehand, which is ideal for large datasets.

There could be several reasons why the SVM model did not perform well in this project. One possible reason that we understand is that the dataset may not be well-suited for an SVM model, as SVMs tend to work best with linearly separable data. If the data is highly non-linear, the SVM model may struggle to find an optimal decision boundary which is the case in our dataset .

After analyzing and comparing the performance of these models, it has been observed that KNN has the highest accuracy among all the models. It is a non-parametric algorithm that predicts the value of a data point based on its closest neighbors. It has been shown to be effective in solving a variety of classification and regression problems. However, it is important to note that accuracy is not the only criterion for selecting the best model. There are other factors such as interpretability, computational complexity, and scalability that should also be considered when choosing a model for real-world applications.

In conclusion, both the KNN and Random Forest models are suitable for our dataset, with the Random Forest model slightly outperforming the KNN model in terms of accuracy. However, it's essential to note that the choice of model depends on the problem being solved and the specific requirements of the project.

7. Future scope

One potential area for future research could be to explore additional machine learning models or ensemble methods to further improve the accuracy of our price prediction model. Additionally, it would be valuable to gather more data, particularly in terms of user demographics and trip details, to gain further insights into factors that may impact ride prices.

Furthermore, examining the impact of external factors, such as weather or events, on pricing could provide a further understanding of pricing trends and potentially enhance the accuracy of our model. We can also deep dive further to understand the potential factors due to which the SVM model was not performing well and apply different techniques to enhance the performance of the model. Also, represent the incorrectly classified examples to understand the core reason for such results and plan to improve it. Finally, expanding the analysis to include other ride-sharing services and comparing their pricing strategies could also provide valuable insights into the industry.

8. References

1. "*Predicting Taxi Ride Duration Using Random Forests and Gradient Boosting*" by Chouette et al. (2018) - This paper explores the use of random forests and gradient boosting to predict taxi ride duration for both Uber and Lyft.
2. "*Price Surge Forecasting for Uber Rideshare Service*" by Shang et al. (2018) - This paper proposes a novel time series model to forecast price surges for Uber rides based on historical demand patterns.
3. "*Demand Forecasting and Surge Pricing in Ride-hailing Services*" by Zhang et al. (2019) - This paper presents a demand forecasting and surge pricing model for ride-hailing services using deep learning techniques, and applies the model to both Uber and Lyft data.
4. "*Ride-hailing Price Surge Prediction with a Scalable Deep Learning Model*" by Wang et al. (2021) - This paper proposes a scalable deep learning model for predicting ride-hailing price surges, and evaluates the model using both Uber and Lyft data.
5. "*Predicting Ride-hailing Prices: A Comparison of Machine Learning Techniques*" by Fayezi et al. (2018) - This paper compares the performance of several machine learning techniques for predicting ride-hailing prices, including linear regression, decision trees, and neural networks, using Uber data.
6. "*Forecasting taxi fare in Uber and Lyft: A comparative study*." By Chen, C., & Pu, P. (2019). Transportation Research Part C: Emerging Technologies, 100, 11-23.
7. "*Machine learning-based dynamic pricing models for Uber and Lyft*." by Uchenna, A., & Vatsavai, R. R. (2019). IEEE Transactions on Intelligent Transportation Systems, 21(3), 1262-1272.
8. "*Research on Taxi Service Price Prediction Model Based on Machine Learning Algorithm: Taking Uber and Lyft as Examples*." by Huang, J., & Qu, W. (2021). Journal of Physics: Conference Series, 1839(1), 012080.