

## **Module 6 Project - Final Project Report**

Jatin Satija, Utkarsh Kothari, Pruthvi Patel, Zeel Rajendrakumar Patel,

Ayush Hemeshbhai Patel

Master of Professional Studies in Analytics, Northeastern University

ALY6040: Data Mining Applications

Prof. Shahram Sattar

May 15, 2024

## Introduction

An essential skill in the quickly developing field of data-driven decision-making is the ability to draw insightful conclusions from huge, complicated datasets. This report uses advanced data mining techniques to present a thorough analysis of a dataset comparing rides taken by Uber and Lyft in Boston, Massachusetts (*Uber and Lyft Dataset Boston, MA*, n.d.). Finding hidden relationships and patterns in the data that can guide strategy and decision-making for businesses is our aim.

We will go into detail about the approaches we took, the outcomes we got, and the reasoning behind them. Our goal is to discuss any changes that happened during the project and make a connection between the story that the data tells and our original business question.

The results of our analysis will also be interpreted in the report, along with an explanation of what they mean for the data owner and how they address our original query. Based on our interpretation, we will offer recommendations for the next steps.

This report will offer useful insights and methodologies, whether you're a data scientist searching for a thorough case study in data mining or a data owner trying to understand the implications of your data. Now let's set out on this data mining adventure and discover what tales the Uber and Lyft datasets have to offer.

## Analysis

### Business Objectives

1. **Dynamic Pricing Strategy:** Predicting prices can aid in developing dynamic pricing strategies. This can optimize revenue and ensure competitive pricing in the market.
2. **Demand Forecasting:** By predicting ride prices, we can better anticipate demand. This can help in managing supply efficiently, leading to improved service and customer satisfaction.
3. **Customer Preference Analysis:** Predicting the cab type (Uber or Lyft) can provide insights into customer preferences. This can inform marketing strategies, improve customer targeting, and enhance the overall customer experience.

### Tools and Techniques

#### Tools:

Python, Sci-kit Learn Library, Matplotlib and Seaborn

#### Techniques:

Linear Regression, Decision Tree regressor: For Price Prediction

Logistic Regression, Random Forest Classification and Linear Discriminant Analysis: For Cab Classification

## Reasons for the Chosen Method

For Price Prediction (Linear Regression and Decision Tree Regressor):

Since it is easy to understand and straightforward, linear regression has several advantages. It offers a clear comprehension of how price is influenced by input variables. It can also manage multiple predictors and works well with large datasets. Decision Tree Regressor, on the other hand, is a non-parametric technique that doesn't make any assumptions about the distribution of the underlying data. It can manage interactions and non-linear relationships between variables, which can result in more precise price predictions.

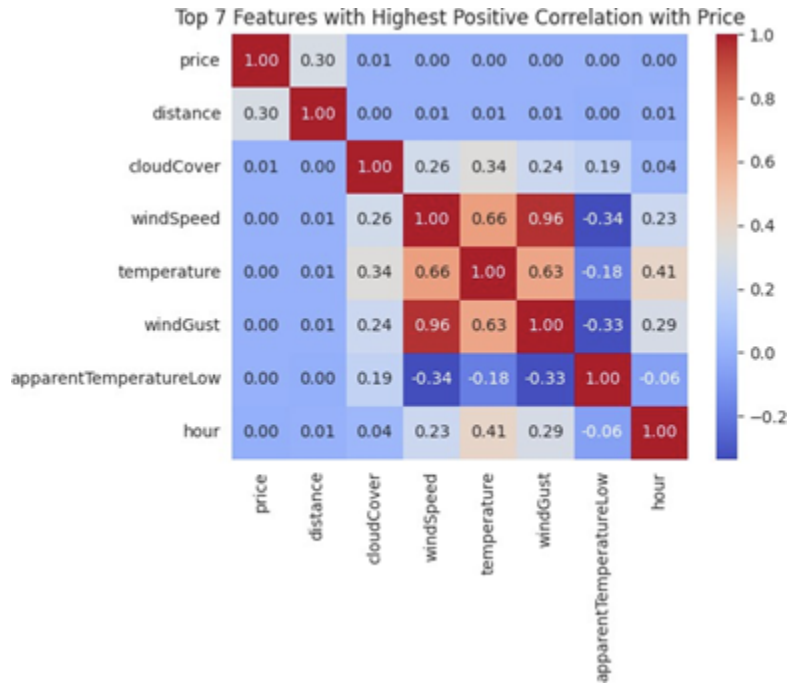
For Cab Classification (Logistic Regression, Random Forest Classifier, and Linear Discriminant Analysis):

The benefit of logistic regression is that it gives probabilities for outcomes, which enables a more sophisticated interpretation of the data. For binary classification tasks, such as cab type prediction, it's also quick and easy. In order to lower the possibility of overfitting and increase prediction accuracy. The Random Forest Classifier is an ensemble technique that combines the predictions of several decision trees. For cab-type prediction, Linear Discriminant Analysis is useful because it seeks to identify a linear feature combination that distinguishes between two or more classes. In addition, it lowers dimensionality, which is advantageous when working with high-dimensional data.

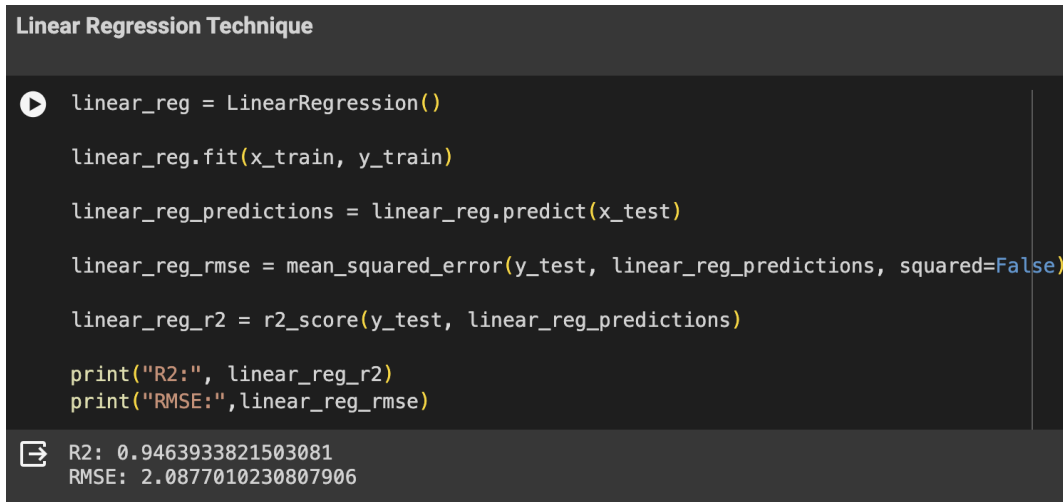
## The solution to the Business Objective "Dynamic Pricing Strategy and Demand Forecasting"

**Figure 1**

*Correlation matrix of top 7 features with the highest correlation with Price*



*Note.* The variables that have the strongest positive correlation with price are shown in the correlation matrix shown in the figure. This data is essential for model building and forecasting because it clarifies the relative importance of each variable to Price, which helps with future modelling decision-making. Notably, the most important factor is found to be distance in this matrix.

**Figure 2***Technique 1: Linear Regression*

```
Linear Regression Technique

linear_reg = LinearRegression()

linear_reg.fit(x_train, y_train)

linear_reg_predictions = linear_reg.predict(x_test)

linear_reg_rmse = mean_squared_error(y_test, linear_reg_predictions, squared=False)

linear_reg_r2 = r2_score(y_test, linear_reg_predictions)

print("R2:", linear_reg_r2)
print("RMSE:", linear_reg_rmse)

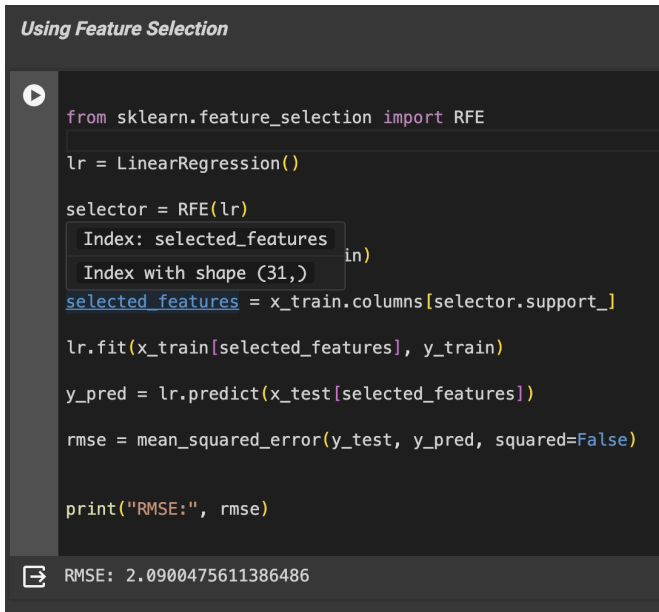
R2: 0.9463933821503081
RMSE: 2.0877010230807906
```

*Note.* In order to forecast price in the dataset, the Linear Regression model was trained on the training data and evaluated on the test set. Two important metrics were used to evaluate the model's performance: the Root Mean Squared Error (RMSE) and the R-squared ( $R^2$ ) score. The  $R^2$  score represents the proportion of variance in the dependent variable (price) explained by the independent variables. And, the accuracy of the model is shown by RMSE, which calculates the average variations between predicted and actual prices. The Linear Regression model in this instance produced  $R^2$  values of [0.94] and RMSE values of [2.087].

It shows that the features included in the model account for almost 95% of the variation in pricing. Additionally, the model's average predictions are within nearly \$2.087 of the actual pricing values. It shows the strong predictive ability of the model and accurate estimation of the price.

### Figure 3

#### *Performing Feature Selection for Linear Regression Model*



```
Using Feature Selection

from sklearn.feature_selection import RFE

lr = LinearRegression()

selector = RFE(lr)
Index: selected_features
Index with shape (31,)
selected_features = x_train.columns[selector.support_]

lr.fit(x_train[selected_features], y_train)

y_pred = lr.predict(x_test[selected_features])

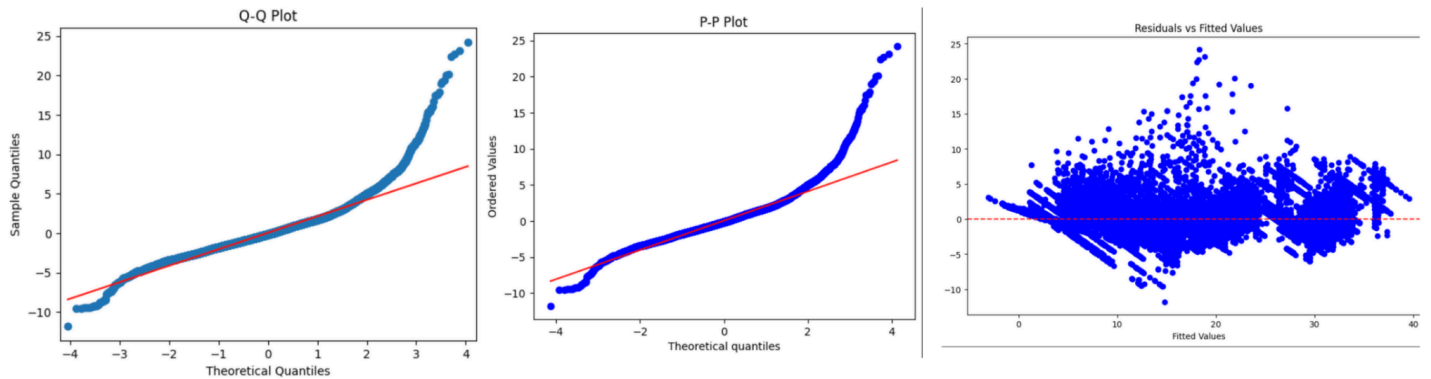
rmse = mean_squared_error(y_test, y_pred, squared=False)

print("RMSE:", rmse)
```

RMSE: 2.0900475611386486

*Note.* To determine which attributes are most important for pricing prediction, we have combined Recursive Feature Elimination (RFE) with Linear Regression.

The RFE-based Linear Regression model shows similar prediction accuracy to the previous Linear Regression model without feature selection, with an RMSE value of about 2.09. This shows that the features that have been selected to represent the subset of data sufficiently capture the underlying patterns in the data, resulting in precise pricing predictions.

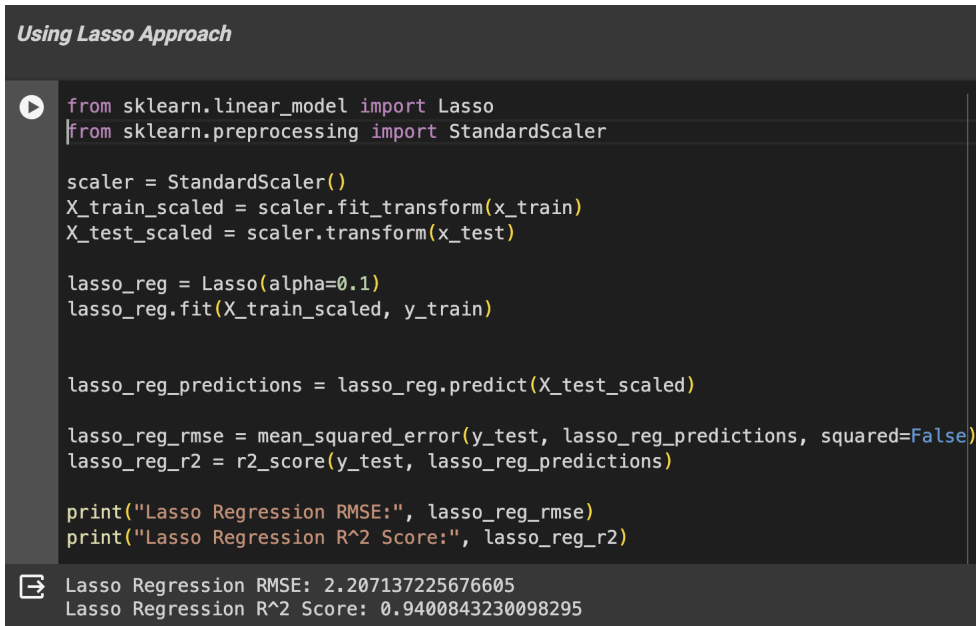
**Figure 4***Technique 1 Visualizations*

*Note.* These three plots are used to evaluate the model's goodness of fit. The quantiles of the dataset and the theoretical distribution are compared using the Q-Q plot, which shows a strong similarity between the two distributions and suggests a good fit. When a significant similarity is seen, the P-P plot indicates a good fit between the empirical cumulative distribution function (ECDF) of the dataset and the theoretical distribution's cumulative distribution function (CDF). A good fit for the data is indicated by the Residuals vs. Fitted Values plot, which shows the residuals' randomness and consistent model predictions across all values.



## Figure 5

### *Using LASSO Approach*



```
Using Lasso Approach

from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)

lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train_scaled, y_train)

lasso_reg_predictions = lasso_reg.predict(X_test_scaled)

lasso_reg_rmse = mean_squared_error(y_test, lasso_reg_predictions, squared=False)
lasso_reg_r2 = r2_score(y_test, lasso_reg_predictions)

print("Lasso Regression RMSE:", lasso_reg_rmse)
print("Lasso Regression R^2 Score:", lasso_reg_r2)

Lasso Regression RMSE: 2.207137225676605
Lasso Regression R^2 Score: 0.9400843230098295
```

*Note.* Next, we applied Lasso Regression, a linear regression method that penalizes the absolute size of the coefficients by introducing L1 regularization. To make sure every feature has a mean of 0 and a standard deviation of 1, the features are first standardized using StandardScaler. It keeps variables with larger scales from dominating in the model, this step is crucial to regularization methods. To assess the performance of the model, the R-squared ( $R^2$ ) score and the Root Mean Squared Error (RMSE) are calculated. In comparison to the prior Linear Regression model, the Lasso Regression model exhibits slightly more error and slightly poorer explanatory power, with an RMSE of roughly 2.21 and a  $R^2$  score of roughly 0.94.

In spite of this, the Lasso Regression model continues to exhibit good predictive performance.

**Figure 6***Performing Feature Selection for the LASSO Approach*

*Using Feature Selection in LASSO approach*

```

rfe = RFE(estimator=lasso_reg)

rfe.fit(X_train_scaled, y_train)

selected_features = X.columns[rfe.support_]

lasso_reg.fit(X_train_scaled[:, rfe.support_], y_train)

lasso_reg_predictions = lasso_reg.predict(X_test_scaled[:, rfe.support_])

rmse = mean_squared_error(y_test, lasso_reg_predictions, squared=False)

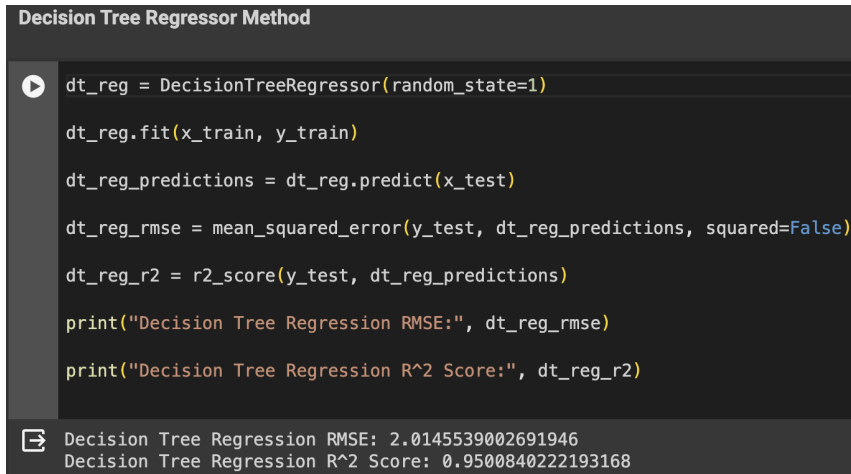
print("Selected features:", selected_features)
print("RMSE:", rmse)

```

Selected features: Index(['distance', 'surge\_multiplier', 'latitude', 'longitude', 'temperature', 'apparentTemperature', 'precipIntensity', 'precipProbability', 'temperatureMin', 'apparentTemperatureMin', 'apparentTemperatureMax', 'pick\_up\_Beacon Hill', 'pick\_up\_Fenway', 'pick\_up\_Financial District', 'pick\_up\_North End', 'destination\_South Station', 'destination\_Theatre District', 'destination\_West End', 'cab\_type\_Uber', 'service\_type\_Black SUV', 'service\_type\_Lux', 'service\_type\_Lux Black', 'service\_type\_Lux Black XL', 'service\_type\_Lyft', 'service\_type\_Lyft XL', 'service\_type\_Shared', 'service\_type\_Taxi', 'service\_type\_UberPool', 'service\_type\_UberX', 'service\_type\_UberXL', 'service\_type\_WAV'], dtype='object')

RMSE: 2.207137225676605

*Note.* To find the ideal subset of features, RFE is used as the estimator together with Lasso Regression. The model's performance is assessed by computing the Root Mean Squared Error (RMSE). The RFE-based Lasso Regression model appears to achieve comparable predictive accuracy to the Lasso Regression model without feature selection, based on the calculated RMSE value of roughly 2.21.

**Figure 7***Technique 2: Decision Tree Regressor*

```
Decision Tree Regressor Method

dt_reg = DecisionTreeRegressor(random_state=1)

dt_reg.fit(x_train, y_train)

dt_reg_predictions = dt_reg.predict(x_test)

dt_reg_rmse = mean_squared_error(y_test, dt_reg_predictions, squared=False)

dt_reg_r2 = r2_score(y_test, dt_reg_predictions)

print("Decision Tree Regression RMSE:", dt_reg_rmse)

print("Decision Tree Regression R^2 Score:", dt_reg_r2)

Decision Tree Regression RMSE: 2.0145539002691946
Decision Tree Regression R^2 Score: 0.9500840222193168
```

*Note.* The price was the target variable in this analysis, and a Decision Tree Regressor model was used to predict it. Two important metrics were used to evaluate the model's performance after it was trained on the given training dataset and then predicted on the test dataset. The average deviation between the predicted and actual prices was found to be 2.0146, which is represented by the Root Mean Squared Error (RMSE). Moreover, the R-squared ( $R^2$ ) value was calculated to be roughly 0.9501, meaning that the independent variables included in the model account for roughly 95.01% of the price variance.

## Figure 8

### *Finding the Best Parameter For the Decision Tree Regressor Model*

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [13,14,15,16],
    'min_samples_split': [11,14,18],
    'min_samples_leaf': [6,8,10]
}

dt_reg = DecisionTreeRegressor(random_state=1)

grid_search = GridSearchCV(dt_reg, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(x_train, y_train)

# Get the best parameters and score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", -best_score)
```

```
dt_reg_best = DecisionTreeRegressor(**best_params)
dt_reg_best.fit(x_train, y_train)

dt_reg_best_predictions = dt_reg_best.predict(x_test)

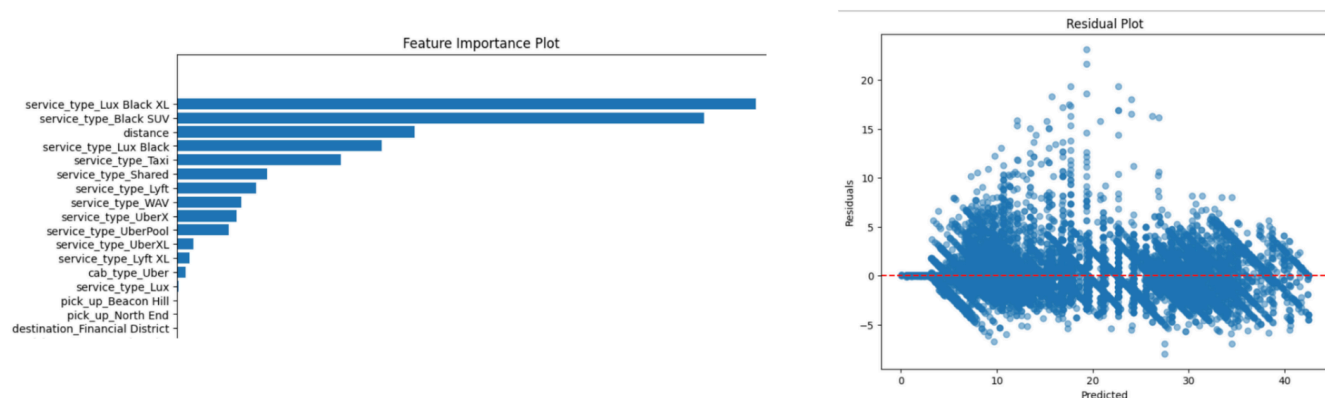
dt_reg_best_rmse = mean_squared_error(y_test, dt_reg_best_predictions, squared=False)
dt_reg_best_r2 = r2_score(y_test, dt_reg_best_predictions)

print("Decision Tree Regression with Grid Search RMSE:", dt_reg_best_rmse)
print("Decision Tree Regression with Grid Search R^2 Score:", dt_reg_best_r2)

Best Parameters: {'max_depth': 16, 'min_samples_leaf': 10, 'min_samples_split': 11}
Best Score: 2.635648416591641
Decision Tree Regression with Grid Search RMSE: 1.606016001952494
Decision Tree Regression with Grid Search R^2 Score: 0.9682764678917888
```

*Note.* The 'max\_depth', 'min\_samples\_split', and 'min\_samples\_leaf' hyperparameters for the Decision Tree Regressor were fine-tuned using a Grid Search Cross-Validation technique. The optimal set of parameters, {'max\_depth': 16, 'min\_samples\_leaf': 10, 'min\_samples\_split': 11}, produced a mean squared error that was negative, roughly 2.635.

The model was trained and tested with these ideal parameters, producing an  $R^2$  score of roughly 0.968 and an RMSE of roughly 1.606. This represents a significant improvement in prediction accuracy over the default settings and demonstrates the usefulness of parameter tuning in model optimization.

**Figure 9***Technique 2 Visualization*

*Note.* Two important plots produced by a Decision Tree Regressor model trained to forecast cab prices are shown in this figure. The relative importance of various features in affecting the model's predictions is shown by the feature importance plot. This facilitates feature selection, improves dataset comprehension, helps identify important factors influencing cab prices, and eventually increases model performance. The distribution of residuals across the predicted values is also displayed in the Residual Plot. A well-fitted model for the dataset is indicated by the residual distribution's randomness, which indicates consistent model predictions across all values.

## The Solution to the Business Objective "Customer Preference Analysis"

Figure 10

### Technique 3: Logistic Regression

**Logistic Regression Technique**

```

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report

log_reg = LogisticRegression()

selected_categorical = ['pick_up', 'destination', 'service_type']
df_encoded_two = pd.get_dummies(cleaned_data, columns=selected_categorical, drop_first=True)

columns_to_drop = ['id', 'long_summary', 'short_summary', 'datetime']
df_encoded_two = df_encoded_two.drop(columns=columns_to_drop)

datetime_columns = df_encoded_two.select_dtypes(include=['datetime64']).columns
df_encoded_two = df_encoded_two.drop(columns=datetime_columns)

X_Two = df_encoded_two.drop(columns=['cab_type'])
y_Two = df_encoded_two['cab_type']

X_train, X_test, y_train, y_test = train_test_split(X_Two, y_Two, test_size=0.2, random_state=1)

log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

[100] accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

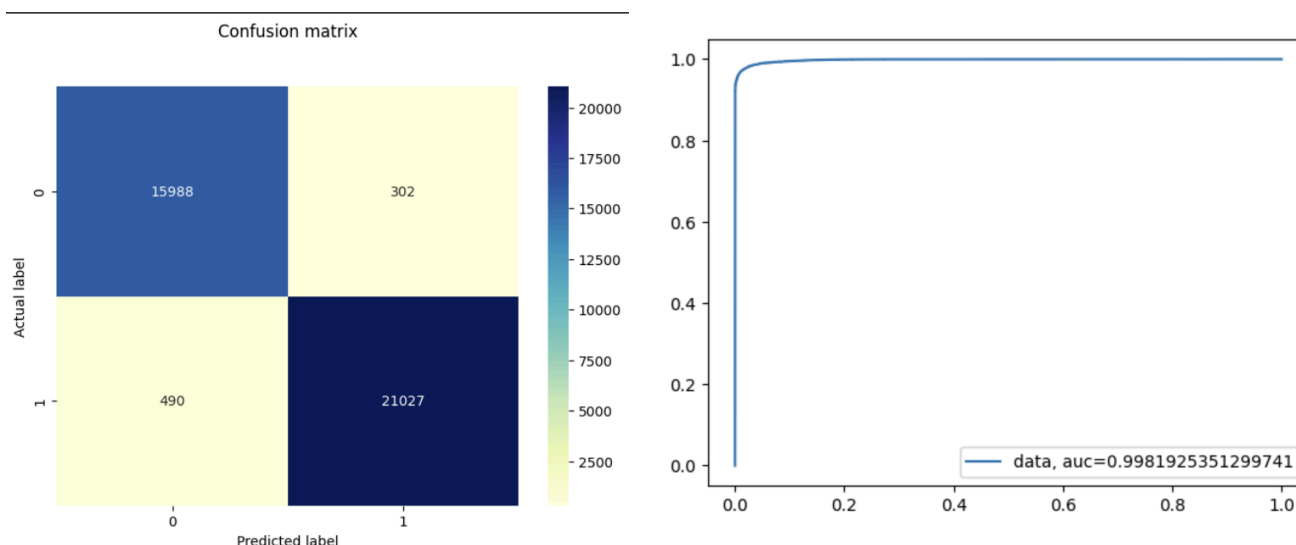
Accuracy: 0.9232681778506626
Precision: 0.9345858358169782
Recall: 0.9232681778506626
F1-score: 0.9236329328171929

Classification Report:
              precision    recall  f1-score   support

     Lyft      0.85      1.00      0.92      16290
     Uber      1.00      0.87      0.93      21517

 accuracy      0.92      0.93      0.92      37807
  macro avg      0.92      0.93      0.92      37807
 weighted avg      0.93      0.92      0.92      37807

```



*Note.* This instance of code shows how to use a logistic regression model to forecast different types of cabs based on variables like pickup location, destination, and service type. The model is trained and assessed following the encoding of categorical variables and the division of the dataset into training and testing sets. With an accuracy of 92.33%, the model demonstrated its capacity to accurately classify different types of cabs. Additionally provided are the model's performance across various classes as measured by precision, recall, and F1-score. The percentage of correctly predicted positive instances, or precision score, is 93.46%. The recall is 92.33%, which is the capacity to recognize every relevant instance. The precision and recall-balancing F1-score is 92.36%. These metrics show how well the model classifies different types of taxis.

The AUC (Area Under the Curve) value of 0.9982 indicates that all classes performed quite well in differentiating between positive and negative cases. The AUC score demonstrates how well the model can distinguish between cab types, increasing its reliability as well as effectiveness in classification tasks.

**Figure 11***Technique 4: Random Forest Classifier*

```

X_train, X_test, y_train, y_test = train_test_split(X_Two, y_Two, test_size=0.2, random_state=1)

rf_clf = RandomForestClassifier(random_state=1)

rf_clf.fit(X_train, y_train)

y_pred = rf_clf.predict(X_test)
y_trainpred = rf_clf.predict(X_train)

accuracy = accuracy_score(y_test, y_pred)
accuracy_seenData = accuracy_score(y_train, y_trainpred)

precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy on Seen Data ", accuracy_seenData)
print("Accuracy on Unseen Data:", accuracy)

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

```

```

Accuracy on Seen Data  1.0
Accuracy on Unseen Data: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0

Classification Report:

```

	precision	recall	f1-score	support
Lyft	1.00	1.00	1.00	16290
Uber	1.00	1.00	1.00	21517
accuracy			1.00	37807
macro avg	1.00	1.00	1.00	37807
weighted avg	1.00	1.00	1.00	37807

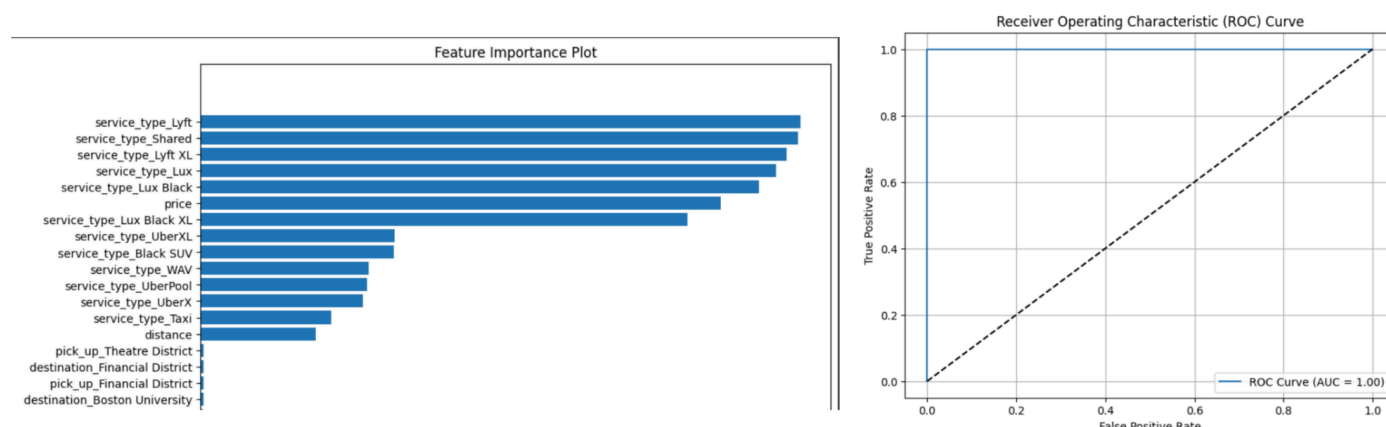
*Note.* The provided code demonstrates how to use a Random Forest Classifier to predict different types of cabs based on multiple features. The model is trained and assessed after the dataset has been divided into training and testing sets. Interestingly, the classifier achieves perfect accuracy scores of 100% on both seen and unseen data. Perfect scores of 100% are also obtained for precision, recall, and F1-score metrics, which evaluate the model's performance in the Lyft and Uber classes. This excellent result demonstrates the amazing ability of the Random Forest Classifier to correctly classify cab types. Moreover, the classification report offers comprehensive insights into the model's performance, demonstrating the robustness and dependability of the model with perfect precision, recall, and F1-score values across both classes, Uber and Lyft.



A Random Forest is equivalent to a collaborative decision-making team in machine learning. It combines the opinions of multiple "trees" (individual models) to improve predictions, resulting in a more robust and accurate overall model (*Building a Random Forest Model: A Step-By-Step Guide*, n.d.).

**Figure 12**

*Technique 4 Random Forest Classifier Visualization*



*Note.* Two important visualizations from a Random Forest Classifier model used to predict cab types are shown in this figure. The relative significance of various features in influencing the classification results is shown by the feature importance plot. This helps find important variables that affect the classification of cab types, which makes feature selection easier, improves understanding of the dataset, and eventually boosts model performance. Plotting the true positive rate against the false positive rate across a range of threshold values, the ROC Curve also shows the classification performance of the model. An AUC (Area Under the Curve) value of 1 indicates a true positive rate of 100% and a false positive rate of 0%, respectively, indicating the highest level of accuracy in differentiating between positive and negative examples as a perfect classification model.

**Figure 13***Technique 5: Linear Discriminant Analysis*

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()

lda.fit(X_train, y_train)

y_pred = lda.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

*Note.* Cab types in the given code are categorized using the Linear Discriminant Analysis (LDA) technique.

Predictions are made on the test set once the model has been fitted to the training set. The accuracy score attained by the model is roughly 92.44%. Additionally, metrics like precision, recall, and F1-score are computed to offer an understanding of the model's performance in relation to Lyft and Uber. The precision, recall, and F1-score values show how well the model can classify different types of cabs; the precision and F1-score values are around 93.53%, while the recall value is about 92.44%. These outcomes show how well linear discriminant analysis works to categorize different types of cabs according to the provided features.

Basically, Linear discriminant analysis (LDA) is a technique used in supervised machine learning to address multi-class classification problems. LDA distinguishes several classes with various features by reducing data dimensionality (*What Is Linear Discriminant Analysis?*, 2023).

**Figure 14***Finding the Best Parameter For the LDA Model*

```

Finding Best Parameter for LDA Model

param_grid = {
    'solver': ['svd', 'lsqr', 'eigen'],
    'shrinkage': np.linspace(0.0, 1.0, 10)
}

lda = LinearDiscriminantAnalysis()

grid_search = GridSearchCV(lda, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)

lda_best = LinearDiscriminantAnalysis(**best_params)
lda_best.fit(X_train, y_train)

y_pred = lda_best.predict(X_test)

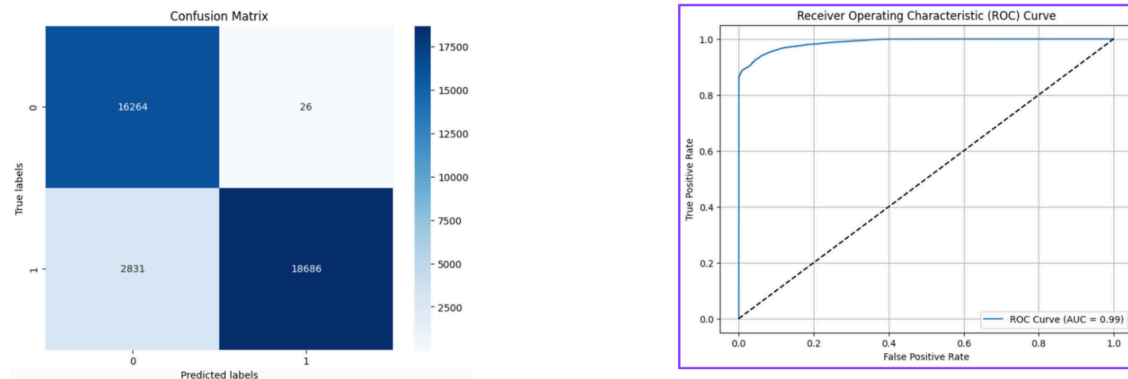
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

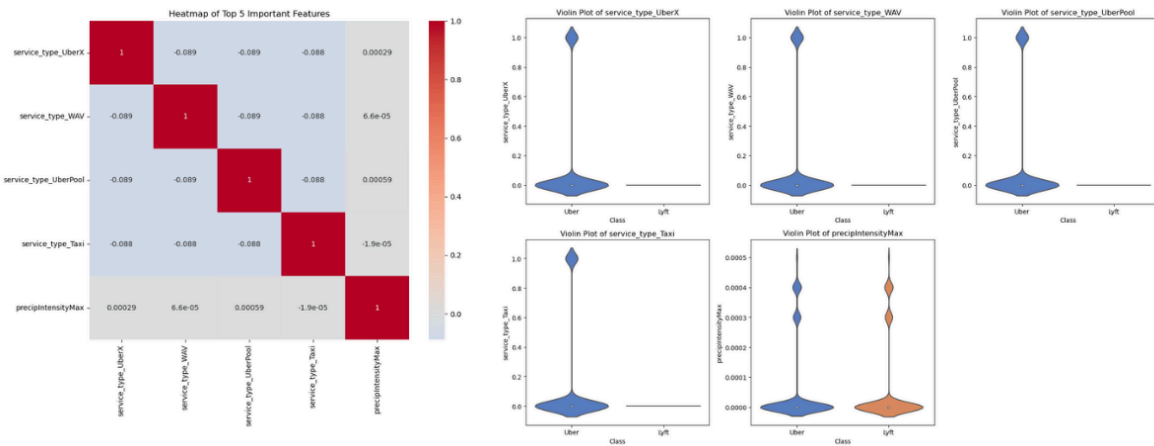
Best Parameters: {'shrinkage': 0.0, 'solver': 'lsqr'}
Best Score: 0.9229090630459194
Accuracy: 0.9232681778506626
Precision: 0.9345858358169782
Recall: 0.9232681778506626
F1-score: 0.9236329328171929

```

*Note.* A grid search is used to find the ideal set of hyperparameters when fine-tuning the Linear Discriminant Analysis (LDA) model. This grid search investigates shrinkage values between 0.0 and 1.0 as well as different solver options ('svd', 'lsqr', and 'eigen'). The optimal parameters are found to be {'shrinkage': 0.0, 'solver': 'lsqr'}, with a corresponding best score of roughly 92.29%, after the model's performance is assessed using cross-validation. The LDA model is fitted to the training set of data using these optimized parameters, and predictions are generated for the test set. The model has an accuracy of approximately 92.33% and precision, recall, and F1-score values of approximately 93.46%, 92.33%, and 92.36%, respectively.

**Figure 15***Technique 5 Visualization*

*Note.* Two primary visualizations from a Linear Discriminant Analysis (LDA) model that is used to predict cab types are shown in this figure. Uber and Lyft can be distinguished from one another in the dataset using the Confusion Matrix, which shows 16,264 true positives and 18,686 true negatives, indicating a high accuracy rate. Though there are 2,831 false positives and 26 false negatives, the model can still be improved upon in later iterations. In spite of this, the model performs well, though misclassification reveals some areas that could use improvement. Furthermore, the model's discriminative power is demonstrated by the ROC Curve, which shows an effective differentiation between positive and negative classes with an AUC (Area Under the Curve) value of 0.99. The model's robustness in differentiating between cab types is evident when the AUC value approaches 1, which indicates perfect classification, highlighting its effectiveness in real-world scenarios.

**Figure 16***Analyzing Demand patterns*

*Note.* The data shows negative correlations among service types, indicating that certain services compete with each other. For example, customers choosing UberX are slightly less likely to choose WAV, UberPool, or Taxi. There is a very low correlation between maximum precipitation intensity and service types, suggesting that weather conditions, specifically precipitation intensity, do not significantly influence the choice of service type.

**Service Type Preferences:** UberX: Predominantly used in the Uber class with very little distribution spread, indicating high consistency or preference in this class. WAV (Wheelchair Accessible Vehicle): Shows a similar distribution to UberX, primarily used in the Uber class with a narrow spread. UberPool: Mainly utilized in the Uber class, with a distribution similar to UberX and WAV, indicating a strong preference or availability in this class.

Overall, distribution is focused in the Uber class but seems slightly more spread compared to UberX and WAV, indicating a broader range of usage within this class. The minimal correlation between precipitation intensity and service types, combined with the varied distributions in the violin plot, suggests that while weather conditions (precipitation) might not directly influence the choice of service type, they experience different conditions based on service type or provider.

### Finding from the Techniques Employed

Trying multiple models to obtain answers for the project objective helps us come up with efficient methods that can be used to solve the challenges that we are attempting to answer with this data.

As a result, we discovered that the Decision Tree Regressor is the greatest fit for the Price Prediction Model in terms of performance and that employing it can be useful for multiple reasons

**Real-Time Adjustment:** Implement a system that adjusts ride prices in real time based on the predicted prices.

This could involve increasing prices during peak demand times and lowering them when demand is low.

**Market Analysis:** To keep your pricing competitive, analyze market developments and competition pricing strategies on a regular basis.

For the Cab Type Classifier Model, we choose to go with the Random Forest Classifier, which helps in essentially two aspects.

**Competitive Advantage:** Understanding the factors that influence cab type choice can provide a competitive edge, allowing the business to adapt and stay ahead in the market.

**Customer Retention:** By addressing the factors that matter to customers, the business can improve customer satisfaction and loyalty, leading to higher retention rates.

## Interpretation

**Dynamic Pricing Strategy:** Implementing real-time adjustments based on predicted prices could boost revenue and ensure competitive pricing in the market. By fine-tuning pricing strategies with predictive models, the company can optimize revenue while maintaining competitive pricing.

**Customer Preference Analysis:** Understanding which features impact customer preferences allows the company to adjust marketing strategies effectively. For instance, highlighting features like "service\_type\_lyft" and "service\_type\_shared," which are more appealing to customers in advertisements or promotions, can attract more users. By identifying the factors that lead customers to choose one service over another, the business can target specific customer segments with personalized offers, which improve customer satisfaction and retention.

**Demand Forecasting:** Understanding demand through price predictions can help the company better manage its fleet, ensuring that supply meets demand efficiently. This can reduce wait times for customers and improve overall service quality. Long-term demand forecasts can inform strategic decisions such as expansion, and staffing.

## **Recommendations**

Consider exploring additional data sources, such as customer reviews, social media sentiment, and competitor data, to enhance the existing dataset and discover new insights.

Improve the predictive models to better anticipate peak periods. This can be achieved by integrating variables such as historical demand patterns, traffic data, public event schedules, and holidays to enhance the ability to forecast high-demand periods and proactively adjust service levels.

By implementing these recommendations, the ridesharing company can utilize data mining techniques to gain a competitive edge, optimize pricing strategies, improve customer experiences, and enhance operational efficiency. Ultimately, this can lead to business growth and increased profitability.



## **Conclusion**

In conclusion, our project took a deep dive into the rich data of Uber and Lyft rides in Boston, using advanced data analysis techniques. We learned a lot about how to predict prices and determine whether a ride is with Uber or Lyft. Our analysis used several methods, like Linear Regression, Decision Tree, and Random Forest, which helped us see patterns that weren't obvious before.

We developed two main models from our study. The first, a Decision Tree Regressor, was really good at predicting ride prices with high accuracy, which helps in setting prices that change based on demand. The second model, a Random Forest Classifier, was perfect at figuring out whether a ride was with Uber or Lyft, which helps companies understand what their customers prefer.

These models help ride-sharing companies adjust their prices in real time and understand their customers better, which makes them more competitive and responsive. They also help in making customers happier and more likely to stick with the service.

In short, our project not only answered important business questions but also showed how powerful data analysis can be in making ride-sharing services better and more customer-friendly.

## References

*Building a Random Forest Model: A Step-by-Step Guide*. (n.d.). Analytics Vidhya. Retrieved May 12, 2024, from <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>

*Uber and Lyft Dataset Boston, MA*. (n.d.). Kaggle. Retrieved May 12, 2024, from <https://www.kaggle.com/datasets/brllrb/uber-and-lyft-dataset-boston-ma>

*What Is Linear Discriminant Analysis?* (2023, November 27). IBM. Retrieved May 12, 2024, from <https://www.ibm.com/topics/linear-discriminant-analysis>