Technical Report: Final Project DS 5110: Introduction to Data Management and Processing

Team Members:
Rishi Kamtam and Sabari Mathavan
Khoury College of Computer Sciences
Data Science Program
`kamtam.r@northeastern.edu, ramasamybalasubram.s@northeastern.edu`

December 11, 2024

Contents

1 Introduction

Background: Living in a large city like Boston where common use of a car is not very popular, many of us rely on modes of public transportation like buses and trains. However, with the growing popularity of Ride share apps like Uber and Lyft, the market dynamic has greatly shifted with over 40 million ride share fares in Boston this past year. A large issue with ride shares however is pricing and variability across different apps like Uber and Lyft, and how prices are actually determined. Many times, we see drastically different prices across different locations like the North End, South End, and Back Bay which is the biggest motivation behind our project, to determine what really impacts the variability in ride share prices.

Objectives: For our project, we had 2 main objectives that would help us address both the instructions and requirements of the project as well as exploring the issues in price variability as mentioned in the background.

1. Develop a Prediction Model for Ride share Fares in Boston This involves utilizing Machine Learning Models, specifically supervised models that take in previous features that impact the target variable (price). This was a large objective for us because we believe that it would be helpful for students like us, who are on tight budgets, to have an application that can predict fare prices to see the most economical methods of transportation to wherever we are planning on traveling to.

2. Enhance Fare Estimation Accuracy This second objective involves doing exploratory data analysis to explore and identify common trends across our data set to give the users a comprehensive overview of our machine learning model. By looking into the correlation between features, differences in average and median pricing between both Uber and Lyft, it allows us and the user to better understand how the ML model actually predicts fare prices.

Goals:

- Deliver a solution that benefits the consumer
- Use Advanced ML Data Science Tools and Visualizations

Scope: The scope of our project first starts with utilizing a dataset from both Uber and Lyft that includes the source, destination, price, distance, car type, and time of hundreds of thousands of rides. Next, our project involves matching weather data to the rides based on the timestamps. This further involves cleaning both datasets then aggregating them together to match weather data with specific rides. The next step is visualizing trends and insights. This step is vital because it gives us and the user an understanding of general correlation and distribution metrics across out dataset, enhancing their understanding of the ML model that will be implemented. The final two steps relate to building multiple ML models and evaluating them through various common metrics and visualizations to see how the model actually performed in predicting ride share fare prices.

2 Literature Review

Existing work regarding predicting ride share fare prices includes utilizing Deep Learning, specifically Neural Networks, to predict prices. Important features for this research were distance traveled, time elapsed, and number of passengers. Based on the study's evaluation, they found that distance traveled, time elapsed, and number of passengers were the most important and had by far the largest impact compared to time of day, cab type, ect.. Feature Scaling and Selection were also large parts of existing work already done as they are important for distinguishing what metrics actually influence the pricing of ride shares.
A gap in this literature and how our project fits into the existing work is with the weather data and aggregating weather data to already existing ride share data. Weather has a large influence on demand of Rideshares and specific metrics like Rain and others could play a role in the variability of pricing. For example, someone is more likely to use ride share apps like Uber and Lyft when it is raining rather than walking or waiting outside for a train or bus. By matching rides to weather patterns, the ML model and exploratory analysis can provide insights into if Weather has a role in how Uber and Lyft determine prices.

3 Methodology

The workflow for the project can be presented as below.

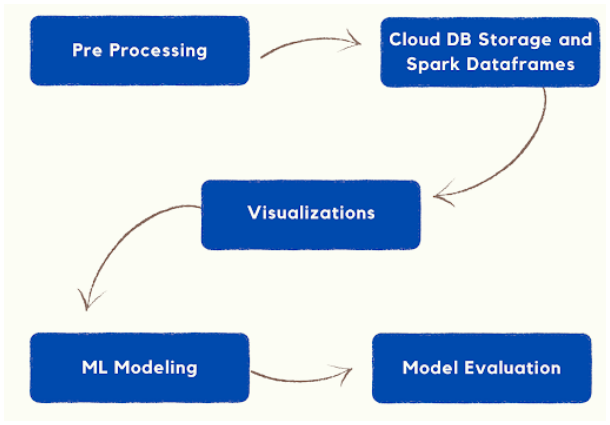


Figure 1: Working mechanism

3.1 Data Collection

The dataset utilized for this project is a static resource available on Kaggle (`https://www.kaggle.com/datasets/ravi72munde/uber-lyft-cab-prices`). It comprises two CSV files:

- **cab_rides.csv:** This file includes data on ride pricing, locations, and timestamps for both Uber and Lyft during the period from November 2018 to December 2018. The records were collected at intervals of 5 minutes, resulting in over 690,000 data points.

- **weather.csv:** This file contains weather metrics corresponding to the same timeframe as the `cab_rides.csv` file. The data were recorded at hourly intervals, amounting to more than 6,000 rows.

3.2 Data Preprocessing

The preprocessing phase ensured the dataset was clean and standardized for analysis. The steps included:

- **Rideshare Data (rides.csv):**
  - Renamed to "Rideshare Data" for clarity.
  - Removed null values to maintain data integrity.
  - Converted the `Time` column from Unix format to human-readable `Datetime`, truncating to the nearest 30 minutes.
- **Weather Data (weather.csv):**
  - Renamed to "Weather Data" for clarity.
  - Converted the `Time` column from Unix format to `Datetime`, truncated to the nearest 30 minutes.
  - Replaced null values in the `Rain` column with 0 to indicate no rain.
- Additional preprocessing tasks included removing duplicate entries, eliminating unnecessary columns, and splitting datasets as needed for analysis.

The datasets were transformed into Pandas DataFrames to facilitate the preprocessing tasks. The rides dataset was further divided into two subsets: *Uber Data* and *Lyft Data*, based on the ride-hailing service type. These subsets, along with the weather DataFrame, were uploaded to Google Cloud BigQuery using the BigQuery API.

The schema for the ride-based data table is as follows:

Column Name	**Data Type**
distance	float
cab_type	string
time_stamp	datetime
destination	string
source	string
price	float
surge_multiplier	float
id	integer
name	string

Table 1: Schema for the ride-based data table

The schema for the weather data table is as follows:

Column Name	Data Type
temp	float
location	string
clouds	float
pressure	float
rain	float
time_stamp	datetime
humidity	float
wind	float

Table 2: Schema for the weather data table

Below is the entity-relationship diagram representing the tables:

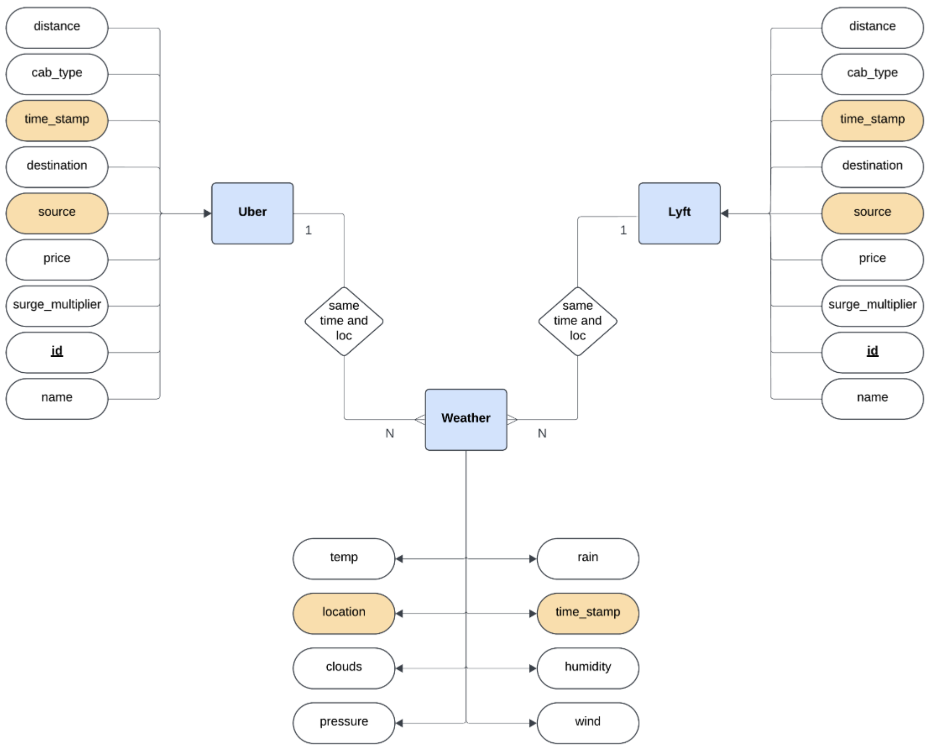


Figure 2: ER Diagram

The preprocessed datasets were stored in Google Cloud BigQuery using the BigQuery API with the following table names:

- Uber ride data: `uber_data`
- Lyft ride data: `lyft_data`
- Weather data: `weather_data`

3.3 Analysis Techniques

The data stored in Google Cloud's BigQuery was accessed using PySpark to enable efficient and faster computations. For example, retrieving data from the `uber_data` table

using PySpark took 12 seconds, while using the BigQuery API for the same data retrieval took over 4 minutes.
Several analyses and visualizations were performed on the datasets, including:

- Statistical comparisons for the ride-hailing services (mean, median, standard deviations, totals, range)
- Box plots (Trip frequency vs Fare)
- Bar charts (Trip frequency vs Fare)
- Scatter plot (Distance vs Fare)
- Heat map (Weather metrics vs Price)
- Line-Bar chart (Rain's influence on Price)
- Line chart (Hourly pricing)
- Line heat map (Hourly x Distance x Pricing)

The visualizations were presented on webpages using the Flask framework. Python Flask is a lightweight, yet powerful web framework suitable for developing web applications. In addition to the aforementioned static visualizations, a dynamic visualization dashboard was included. The dynamic dashboard allows user interaction, where the user inputs source and destination points. The dashboard then calculates and provides the average pricing for each ride-hailing service across the given locations at an hourly level. Upon entering the source and destination, these values are processed and passed as parameters to the Google BigQuery API. The corresponding data (hourly pricing for each cab service at the given locations) is then returned and displayed in the dashboard as a line graph and a table.

As for the machine learning portion of our project, the methodology we took first involved defining the features and target variable. Features are variables that impact the target variable and in the context of our project, features include weather data such as rain, clouds, ride distance, etc.. and the target variable is the price because it is what we are trying to predict. After defining these, we had to implement feature engineering for the categorical data. This is an important step because a supervised ML model cannot read text, only interpret numbers so we had to change our categorical data to numerical data. Upon doing this, we were ready to implement our first ML model (Random Forest) which combines independent decision trees. Then, we cross validated the model which splits our dataset and trains it on unseen data to avoid over fitting within our model. Over fitting in machine learning refers to "creating a model that matches (memorizes) the training set so closely that the model fails to make correct predictions on new data." Then we built our second ML model (XGBoost) which works with decision trees but the main difference being it develops one tree at a time, correcting faults caused by previous tree. Finally, we evaluated our model using metrics such as variations of Mean Absolute Error scores and R-Squared scores as well as visualizations regarding how well our model predicted fare prices.

4 Results

Data Analysis and Visualization Results: Through the analysis, the following key insights were obtained:

- **Uber's Pricing:** It is evident that Uber offers lower average pricing and has a lower standard deviation (SD) compared to Lyft. This indicates that Uber's fare prices are generally more consistent and predictable.
- **Weather Metrics Influence:** The analysis indicates that weather metrics do not significantly affect ride pricing. Despite initial expectations, factors such as temperature, rain, and humidity showed little correlation with fare fluctuations.
- **Influence of Hour and Location:** The time of day and specific location have a significant impact on pricing. Hourly variations and location-based trends were found to play a crucial role in determining fare prices for ride-hailing services.

Below are some visualization charts that demonstrate these findings:

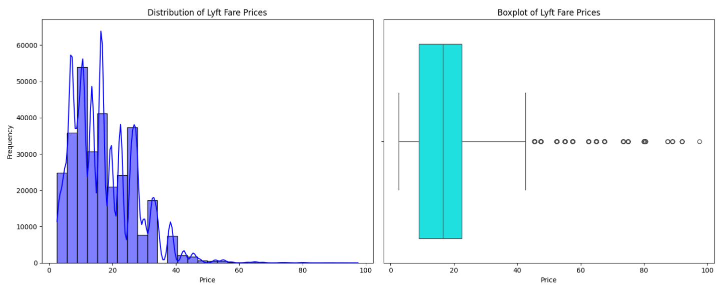


Figure 3: Lyft's Trip-Price Distribution

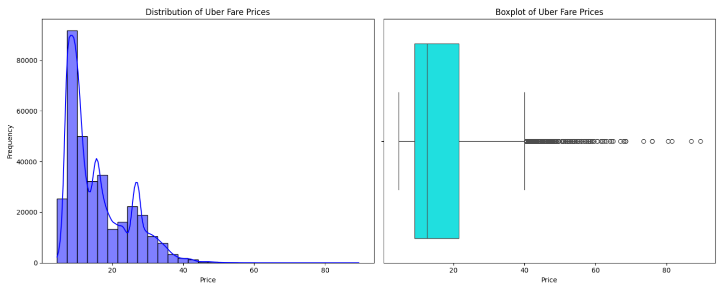


Figure 4: Uber's Trip-Price Distribution

The analysis using Python revealed that the number of outliers in the Lyft data set represented 1.33% of the total data for Lyft, while for Uber, the outliers accounted only 0.91%.

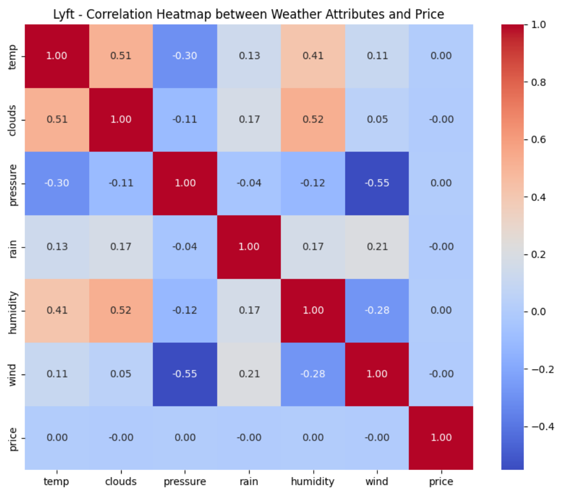


Figure 5: Lyft's Heatmap

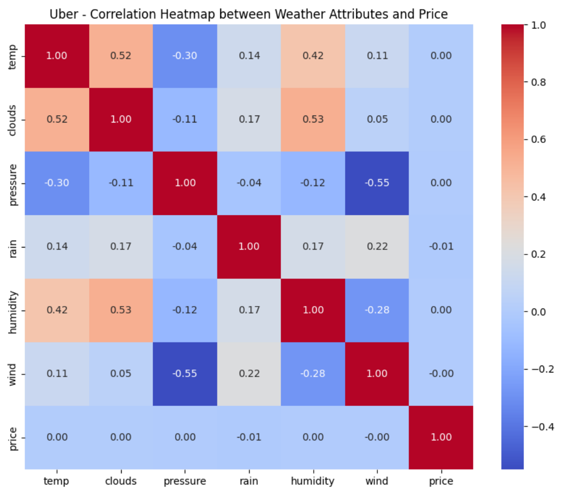


Figure 6: Uber's Heatmap

As we could see, the weather metrics have a very minimal effect on pricing based on the obtained data set.

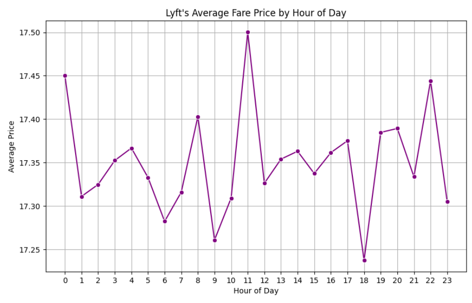


Figure 7: Lyft's Hourly Pricing

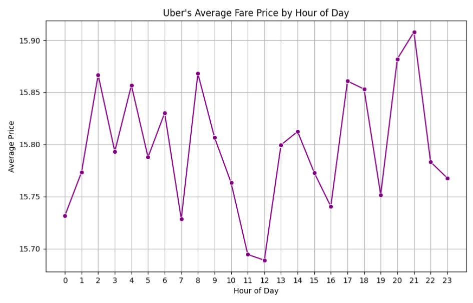


Figure 8: Uber's Hourly Pricing

From the above graph, we can find that the average pricing of Uber is lower compared to Lyft's pricing across all hours of the day.

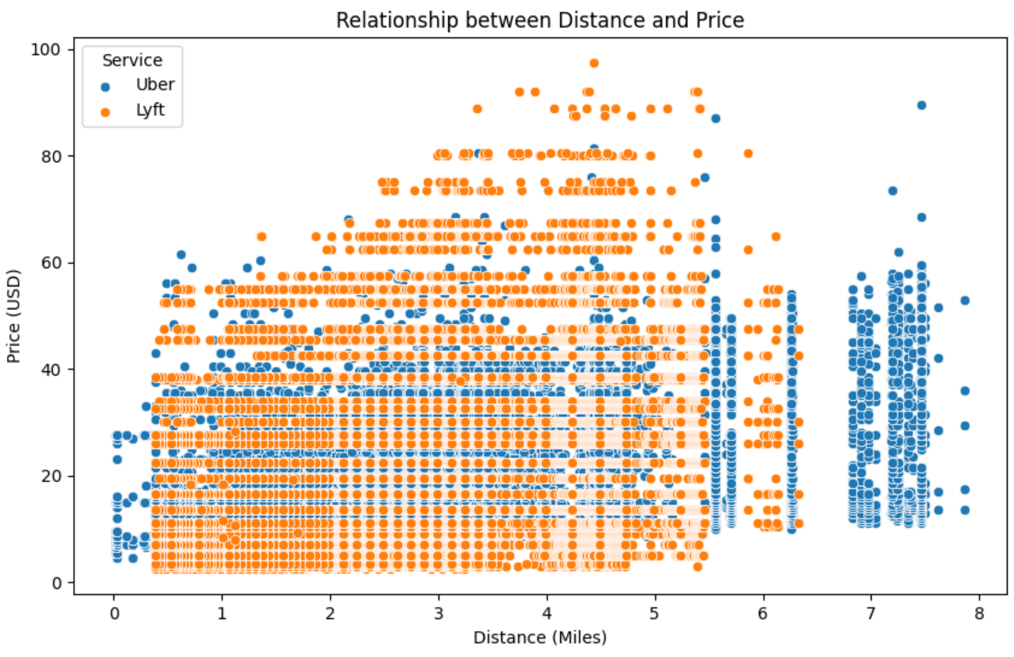


Figure 9: Scatter Plot for Price vs Distance

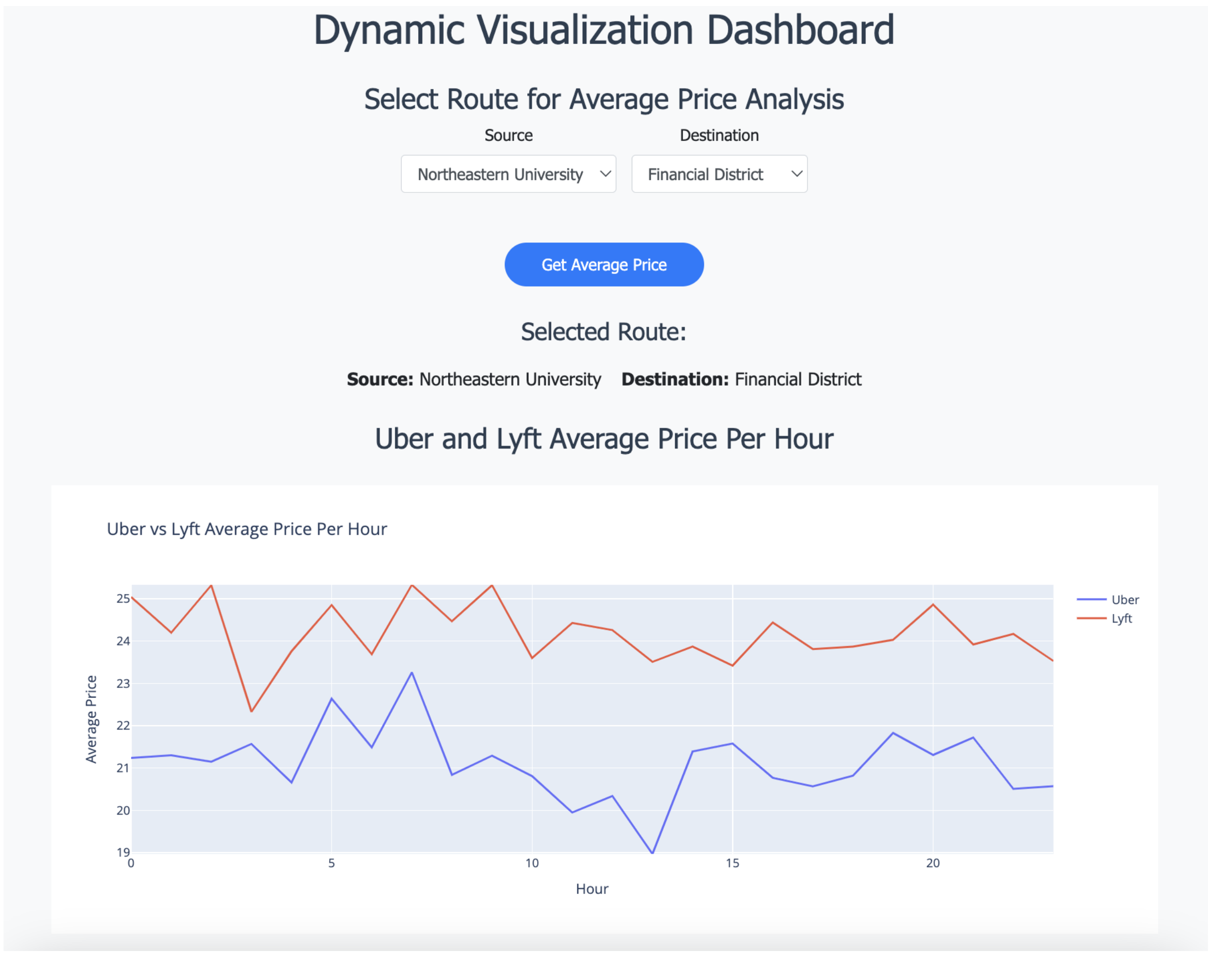

Dynamic Visualization Dashboard
Select Route for Average Price Analysis
Source
Northeastern University
Destination
Financial District
Get Average Price
Selected Route:
Source: Northeastern University Destination: Financial District
Uber and Lyft Average Price Per Hour
Uber vs Lyft Average Price Per Hour
Average Price
Hour
Uber
Lyft

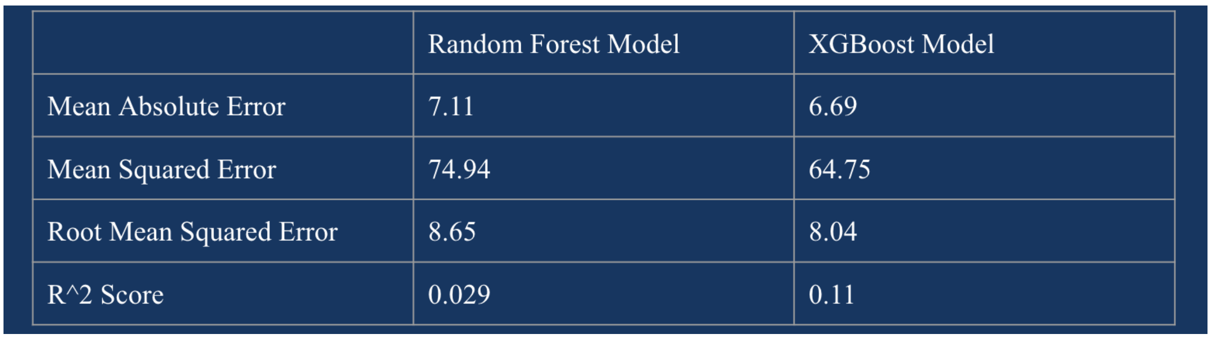
Figure 10: Dynamic Visualization

The image above displays a snippet of the dynamic visualization feature. Based on the user-provided source and destination inputs, the graph dynamically adjusts to show the hourly pricing for each ride-hailing service between the two locations.

Machine Learning Results:

Considering the table of the results, the XGBoost Model had a better performance across all the metrics. Specifically, considering the lower Mean Absolute Error, we can conclude that the variability across pricing predictions was lower for the XGBoost model by about 0.4 dollars than the Random Forest Model. However, in a grander context, both models had a decent performance as there was still variability within the models' pricing predictions. Also in terms of the features, the R-Squared scores indicate that the features do not explain the pricing very well.

Note: Residual = Actual Y Value - Predicted Y Value Looking at the comparison of Residual Visualizations, it is evident that the XGBoost model performed better because there are less data points spread outside the red line (indicating perfect model performance). Also, it can be determined through the visualization that the model performed better for mid tier pricing (15-30) dollars as there is a higher concentration of data points in that range closer to the red line.



	Random Forest Model	XGBoost Model
Mean Absolute Error	7.11	6.69
Mean Squared Error	74.94	64.75
Root Mean Squared Error	8.65	8.04
R^2 Score	0.029	0.11

Figure 11: Table of ML Model MAE and R-Squared Result Metrics

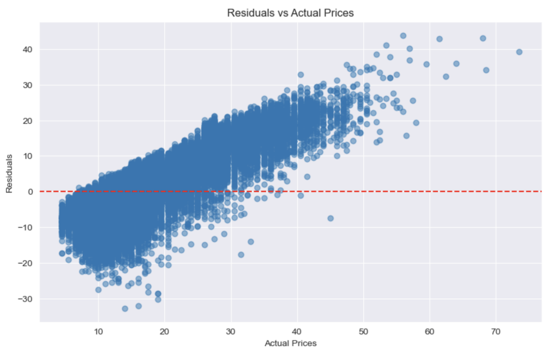


Figure 12: Residual Visualization for RF Model

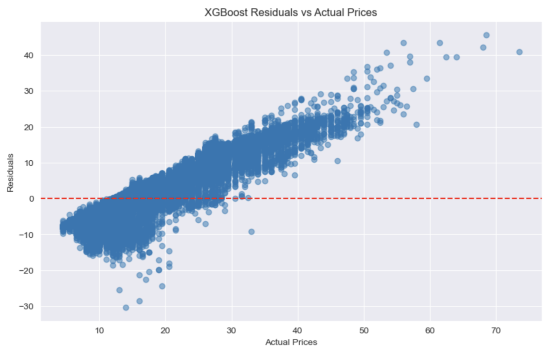


Figure 13: Residual Visualization for XGBoost Model

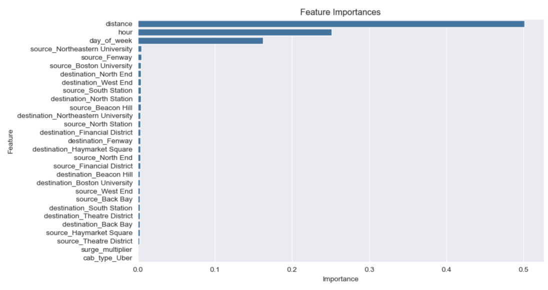


Figure 14: Feature Importance for RF Model

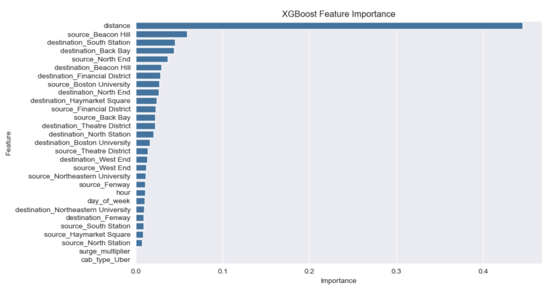


Figure 15: Feature Importance for XG-Boost Model

Upon building and analyzing the visualization of feature importance, one can conclude that distance is the most important in determining the price of a ride share for both models. Something that we found interesting was that the second most important feature was different for both models. For the RF model it was the hour (of the day) while for the XGBoost model it was the source (specifically Beacon Hill). The source of Beacon Hill being important for the XGBoost model we think can be explained by the high income of the Beacon Hill neighborhood and ride share prices most likely being high for that area. Overall, for both ML models, the results indicate that there was a decent performance especially for middle tier pricing and distance was the most important feature, impacting the models the most.

5 Discussion

The ride share prices predicted depend on different metrics, making it challenging to predict fares using machine learning models. Some factors include driver behavior, passenger preferences, location, time, weather conditions, etc.
Based on the visualization performed on the dataset accumulated, an inference can be made that there is roughly no correlation between price and specific weather metrics. This suggests that weather conditions (e.g., temperature, rain, wind, pressure, humidity, clouds) do not significantly influence fare. On the other hand, factors such as distance, location, and time were found to have a much larger impact on price fluctuations. For the better working of this project, the previously prescribed metrics should be prioritized for more accurate price predictions.

The project's results were restricted due to a few limitations, such as:

- **Inflation Impact:** The dataset is 6 years old (Nov 2018-Dec 2018) and does not account for inflation in rideshare prices over the years.

- **Changing Location Demographics:** Over the past 6 years, the demographic and economic makeup of the locations may have changed, causing a change in current pricing.
  The machine learning model developed for price prediction performs decently on the training data but struggles with unseen data. The model could not work properly when presented with geographical locations apart from the dataset, making the model look general and specific case-oriented to locations and time.

  Our findings align closely with those reported in the literature. While our Mean Absolute Error (MAE), a key metric for evaluating model performance, was 6.69, the literature reported a slightly better MAE of 5.96. This difference can be attributed to the use of Deep Learning Models in the literature, as opposed to our Supervised Machine Learning Model. Deep Learning Models are more effective at capturing and predicting non-linear relationships, suggesting that the features in our dataset may exhibit significant non-linearity with respect to the target variable (price).

  Notably, our analysis highlights the potential impact of weather on predictions. Although our results showed no correlation across weather metrics, this may indicate a complex, non-linear relationship that our model struggled to capture. Integrating weather data into a Deep Learning Model could provide valuable insights, as these models are better equipped to handle non-linear interactions.

6 Conclusion

The results for the project were well executed, as mentioned in the scope of the project. Compiling them together:

- **Data Pre-processing:** Static data regarding rideshare pricing and weather conditions were accumulated from Kaggle. The data were preprocessed and cleaned to ensure that they could be stored and used for analysis.
- **Data Storage:** The data were stored in Google Cloud BigQuery as it involved cloud storage facilities that met the project's needs.
- **Data Analysis:** The data were later analyzed through PySpark to facilitate quicker and more reliable analysis on large datasets. The important findings from the analysis include:
  - It was found that weather metrics do not hold much significance in the pricing model. Despite initial expectations, factors such as temperature, rain, and humidity did not have a strong correlation with price fluctuations.
  - The scores of mean, median, and standard deviation favored Uber more than Lyft.
- **Data Visualization:** Both static and dynamic methods of data visualization were performed and showcased using a Flask webpage to ensure that no data leaks or unnecessary information were passed to the end user.
- **Predictive Modeling:** A predictive model was designed to predict the pricing of the ride-hailing services between two locations at a given time and weather conditions. The machine learning model performed well for mid-tier pricing but struggled with predicting low and high fare values accurately.

The final analysis suggests that Uber's pricing model tends to be more accurate and competitive compared to Lyft's pricing.

Building on the findings of this project, several areas for future improvement and expansion are proposed:

- **Incorporating Real-Time Data:** To enhance the accuracy and relevance of fare predictions, integrating live traffic, weather, and surge pricing data would be crucial. Real-time inputs would allow the model to adjust predictions dynamically, improving its utility in everyday rideshare operations.
- **Deep Learning Models:** The use of deep learning techniques could be explored to better capture non-linear relationships between pricing factors. This approach could improve model performance.
- **Geographical Expansion:** The model should be expanded to cover a wider geographical scope beyond Boston. Exploring areas such as suburban Boston and other major cities like New York and Chicago will provide a more comprehensive view of rideshare pricing trends across diverse regions.

Incorporating these changes would improve both the accuracy and scalability of the pricing model, making it applicable to a broader range of situations and locations.

7 References

References

1. Connecting Apache Spark to BigQuery. *Google Cloud Documentation*, Google, https://cloud.google.com/bigquery/docs/connect-to-spark. Accessed 3 Dec. 2024.
2. Random Forest Algorithm in Machine Learning. *GeeksforGeeks*, https://www.geeksforgeeks.c forest-algorithm-in-machine-learning/. Accessed 3 Dec. 2024.
3. Thomas, Moses Moncy. Predictive Analysis: Estimating UBER Fare Prices Using ML. *Medium*, 26 Apr. 2023, https://medium.com/@mosesmoncy1626/predictive-analysis-estimating-uber-fare-prices-using-ml-7bb8c54507e9.
4. XGBoost. *NVIDIA Glossary*, NVIDIA, https://www.nvidia.com/en-us/glossary/xgboost/. Accessed 3 Dec. 2024.

A Appendix A: Code

```python
# Converting Unix timestamp to Datetime, truncated to nearest hour
rides_df = rides_df.copy()
rides_df['time_stamp'] = pd.to_datetime(rides_df['time_stamp'],
    unit='ms').dt.floor('30min')
weather_df = weather_df.copy()
weather_df['time_stamp'] = pd.to_datetime(rides_df['time_stamp'],
    unit='ms').dt.round('30min')
```

Listing 1: Sample Data Preprocessing

```python
from google.cloud import bigquery
from google.api_core.exceptions import NotFound

# Initialize BigQuery client
client = bigquery.Client.from_service_account_json("secrets/
    serviceKey.json")

# Define the dataset reference
project_id = "xxx"
dataset_id = "yyy"
dataset_ref = f"{project_id}.{dataset_id}"

# List all tables in the dataset
tablesList = []
try:
    tables = client.list_tables(dataset_ref)
    print(f"Tables in dataset {dataset_id}:")
    for table in tables:
        tablesList.append(table.table_id)
        print(table.table_id)
except NotFound:
    print(f"Dataset {dataset_id} does not exist.")
```

Listing 2: Google Cloud BigQuery: Table creation

```python
def overWriteDataToTable(dataFrameName, table_ref):
    job_config = bigquery.LoadJobConfig(
        write_disposition="WRITE_TRUNCATE",  # Overwrite the data to the table
    )
    job = client.load_table_from_dataframe(dataFrameName, table_ref, job_config=job_config)

    # Wait for the job to complete
    job.result()

    print(f"Over written {len(dataFrameName)} rows to the table {table_id} in dataset {dataset_id}.")
```

Listing 3: Google Cloud BigQuery: Data Push

```python
from pyspark.sql import SparkSession
import os
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'secrets/serviceKey.json'


# JAR paths for BigQuery and GCS connectors
bigquery_connector_jar = "spark-bigquery-connector.jar"
gcs_connector_jar = "gcs-connector.jar"


# Create SparkSession with both connectors
spark = SparkSession.builder \
    .appName("PySpark with BigQuery and GCS") \
    .config("spark.jars", f"{bigquery_connector_jar},{gcs_connector_jar}") \
    .config("spark.sql.catalog.spark_bigquery", "com.google.cloud.spark.bigquery.BigQueryCatalog") \
    .config("spark.hadoop.google.cloud.auth.service.account.json.keyfile", "secrets/serviceKey.json") \
    .config("spark.bigquery.projectId", "idmpproject-441123") \
    .getOrCreate()

spark
```

Listing 4: Google Cloud BigQuery: Data Pull using PySpark

```python
# Visualization 1 Fare price distribution (histogram and bar plot) - Uber

from pyspark.sql.functions import col, lit
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the output file path
output_image_path = os.path.join(output_folder, "uber_fare_price_distribution.png")
```

```

# Convert the 'price' column to Pandas for visualization
price_data_pd = uber_df.select("price").filter(col("price").
    isNotNull()).toPandas()

# Create a figure with two subplots: one for the histogram and one
    for the boxplot
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Histogram Plot
sns.histplot(price_data_pd['price'], bins=30, kde=True, color='blue
    ', ax=axes[0])
axes[0].set_title('Distribution of Uber Fare Prices')
axes[0].set_xlabel('Price')
axes[0].set_ylabel('Frequency')

# Boxplot Plot
sns.boxplot(x=price_data_pd['price'], color='cyan', ax=axes[1])
axes[1].set_title('Boxplot of Uber Fare Prices')
axes[1].set_xlabel('Price')

# Display the plots
plt.tight_layout()

# Save the figure as a PNG image
plt.savefig(output_image_path)

# Inform the user
print(f"Visualization saved to '{output_image_path}'")

plt.show()
```

Listing 5: Sample Visualization using PySpark

```
import os
import pandas as pd
from flask import Flask, render_template, request
from google.cloud import bigquery
from datetime import datetime


# Dynamically determine the absolute path to the 'static' folder
BASE_DIR = os.path.abspath(os.path.dirname(__file__))
STATIC_FOLDER = os.path.join(BASE_DIR, 'static')
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'secrets/serviceKey.
    json'

client = bigquery.Client()
project_id = "xxx"
dataset_id = "yyy"


app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')
```

Listing 6: Flask initialization

```python
import plotly.express as px
import pandas as pd
import plotly.graph_objects as go

@app.route('/dynamic-visualizations', methods=['GET', 'POST'])
def dynamic_visualization():
    # Define project_id and dataset_id variables
    project_id = "idmpproject-441123"
    dataset_id = "uberFareEstimation"

    # Define the BigQuery queries to get unique sources and destinations from both tables
    source_query = f"""
    SELECT DISTINCT source FROM `{project_id}.{dataset_id}.uber_data`
    UNION DISTINCT
    SELECT DISTINCT source FROM `{project_id}.{dataset_id}.lyft_data`
    """

    destination_query = f"""
    SELECT DISTINCT destination FROM `{project_id}.{dataset_id}.uber_data`
    UNION DISTINCT
    SELECT DISTINCT destination FROM `{project_id}.{dataset_id}.lyft_data`
    """

    # Execute the queries
    sources = client.query(source_query).result()
    destinations = client.query(destination_query).result()

    source_list = [row.source for row in sources]
    destination_list = [row.destination for row in destinations]

    # Create a dictionary mapping sources to possible destinations
    source_to_dest = {}
    for source in source_list:
        query = f"""
        SELECT DISTINCT destination FROM `{project_id}.{dataset_id}.uber_data`
        WHERE source = '{source}'
        UNION DISTINCT
        SELECT DISTINCT destination FROM `{project_id}.{dataset_id}.lyft_data`
        WHERE source = '{source}'
        """
        dest_result = client.query(query).result()
        source_to_dest[source] = [row.destination for row in dest_result]

    # Initialize variables for source and destination
    source = None
    destination = None
    uber_avg_price_per_hour = None
    lyft_avg_price_per_hour = None
    df_html = None  # To store the HTML table

    if request.method == 'POST':
        source = request.form.get('source')
        destination = request.form.get('destination')

        # BigQuery query to get the data based on selected source and destination from both tables
        query = f"""
        WITH uber_data AS (
            SELECT
                EXTRACT(HOUR FROM time_stamp) AS hour,
                ROUND(AVG(price), 2) AS avg_price
            FROM `{project_id}.{dataset_id}.uber_data`
            WHERE source = '{source}' AND destination = '{destination}'
            GROUP BY hour
        ),
        lyft_data AS (
            SELECT
                EXTRACT(HOUR FROM time_stamp) AS hour,
                ROUND(AVG(price), 2) AS avg_price
            FROM `{project_id}.{dataset_id}.lyft_data`
            WHERE source = '{source}' AND destination = '{destination}'
            GROUP BY hour
        )
        SELECT
            COALESCE(u.hour, l.hour) AS hour,
            u.avg_price AS uber_avg_price,
            l.avg_price AS lyft_avg_price
        FROM uber_data u
        FULL OUTER JOIN lyft_data l ON u.hour = l.hour
        ORDER BY hour
        """

        # Execute the query and fetch the results
        result = client.query(query).result()

        # Convert the result to a DataFrame
        df = pd.DataFrame([dict(row) for row in result])

        if df.empty:
            uber_avg_price_per_hour = "No data available for the selected source and destination."
            lyft_avg_price_per_hour = "No data available for the selected source and destination."
        else:
            # Plot the line graph
            fig = go.Figure()

            fig.add_trace(go.Scatter(x=df['hour'], y=df['uber_avg_price'], mode='lines', name='Uber'))
            fig.add_trace(go.Scatter(x=df['hour'], y=df['lyft_avg_price'], mode='lines', name='Lyft'))

            # Set the range for the x-axis and y-axis
            fig.update_layout(
                title="Uber vs Lyft Average Price Per Hour",
                xaxis_title="Hour",
                yaxis_title="Average Price",
                xaxis=dict(
                    range=[df['hour'].min(), df['hour'].max()]  # Limit x-axis range to the hours in the data
                ),
                yaxis=dict(
                    range=[df[['uber_avg_price', 'lyft_avg_price']].min().min(), df[['uber_avg_price', 'lyft_avg_price']].max().max()]  # Adjust y-axis range dynamically
                )
            )

            # Render the plot in the HTML template
            graph_html = fig.to_html(full_html=False)

            # Convert the DataFrame to an HTML table
            df_html = df.to_html(classes='data', header=True, index=False)
            uber_avg_price_per_hour = graph_html  # Ensure this contains the full HTML of the graph

    return render_template('dynamic_visualizations.html',
                               sources=source_list,
                               destinations=destination_list,
                               source_to_dest=source_to_dest,
                               uber_avg_price_per_hour=uber_avg_price_per_hour,
                               df_html=df_html,
                               selected_source=source,
                               selected_destination=destination)
```

Listing 7: Flask dynamic visualization and data retrieval using BigQuery and SQL

```
# Define categorical and numerical features
categorical_features = ['cab_type', 'source', 'destination']
numerical_features = ['distance', 'surge_multiplier', 'hour', 'day_of_week']

# Set up the ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
        ('num', 'passthrough', numerical_features)
    ]
)

# Display the preprocessor to confirm setup
print(preprocessor)
```

Listing 8: ML Modeling: Configures a ColumnTransformer to preprocess categorical features using OneHotEncoder

```
# Cross-validation and CV metrics for the RF model using MAE

# Sample 25% of the data for cross-validation
X_sample = X.sample(frac=0.25, random_state=42)
y_sample = y.loc[X_sample.index]

```

```python
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Define the scoring metric (negative MAE)
scorer = make_scorer(mean_absolute_error, greater_is_better=False)

# Perform cross-validation using the pipeline
cv_scores = cross_val_score(pipeline, X_sample, y_sample, scoring=
    scorer, cv=kf, n_jobs=-1)

# Convert negative MAE to positive MAE
cv_mae_scores = -cv_scores

# Print the results
print(f"Cross-Validation MAE Scores: {cv_mae_scores}")
print(f"Mean MAE: {np.mean(cv_mae_scores)}")
print(f"Standard Deviation of MAE: {np.std(cv_mae_scores)}")
```

Listing 9: Cross Validating RF Model and Getting Metrics for RF Model

```python
# Import the XGBoost library
from xgboost import XGBRegressor

# Define the XGBoost model
xgb_model = XGBRegressor(n_estimators=150, learning_rate=0.1,
    max_depth=6, random_state=42, enable_categorical=True)

# Create a pipeline with preprocessing and the XGBoost model
pipeline_xgb = Pipeline(steps=[
    ('preprocessor', preprocessor),  # Use the same preprocessor as
    Random Forest
    ('regressor', xgb_model)
])

# Train the pipeline on the training data
pipeline_xgb.fit(X_train, y_train)

print("XGBoost pipeline training complete.")

# Predict on the test set
y_pred_xgb = pipeline_xgb.predict(X_test)


# Evaluate performance
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
rmse_xgb = np.sqrt(mse_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

# Print the metrics
print(f"XGBoost Evaluation Metrics:")
print(f" - Mean Absolute Error (MAE): {mae_xgb}")
print(f" - Mean Squared Error (MSE): {mse_xgb}")
print(f" - Root Mean Squared Error (RMSE): {rmse_xgb}")
print(f" - R   Score: {r2_xgb}"
```

Listing 10: XGBoost Model Code

B Appendix B: Additional Figures

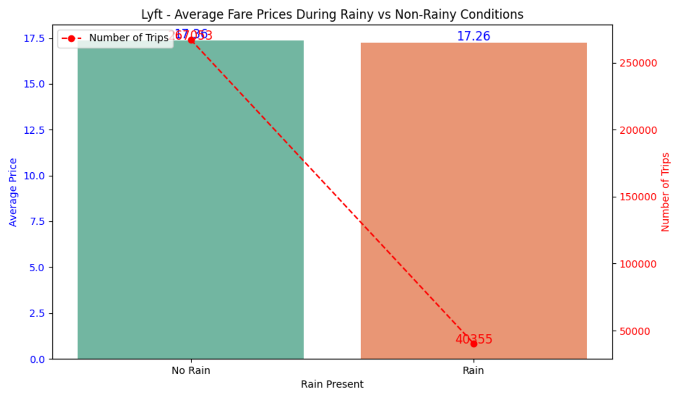


Figure 16: Line Graph on Lyft's pricing on rainy and non-rainy days

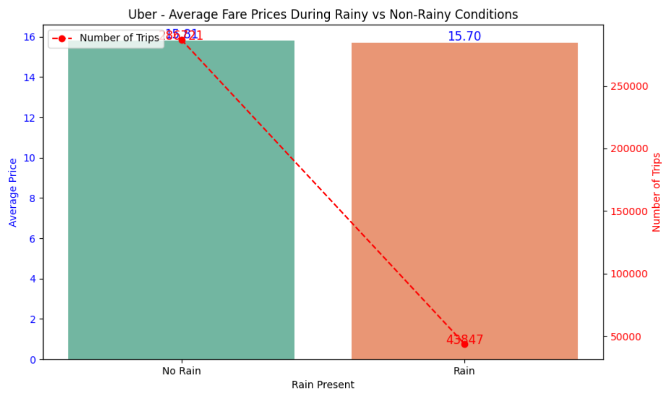


Figure 17: Line Graph on Uber's pricing on rainy and non-rainy days

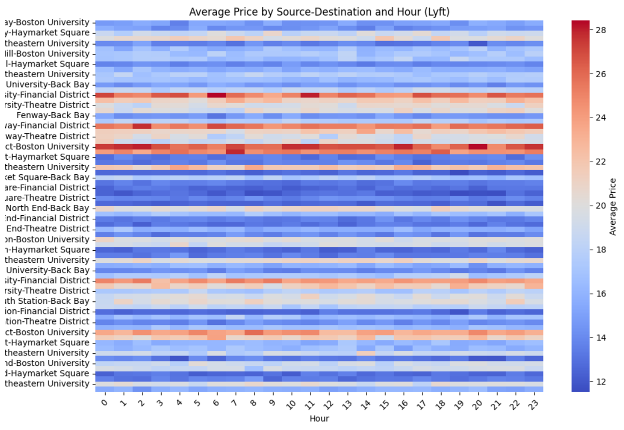


Figure 18: Lyft's Place x Hourly Pricing (Line heat map)

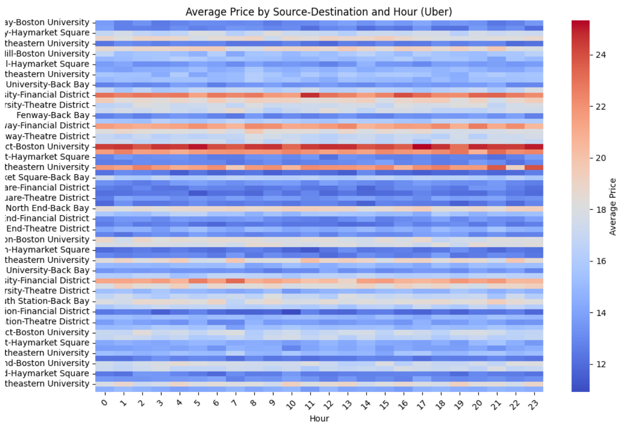


Figure 19: Uber's Place x Hourly Pricing (Line heat map)