Project Summary

Maksuda Toma, Israt Zarin, Shadman Shakib

March 19, 2025

Context

Uber and Lyft's ride prices are not constant like public transport. They are greatly affected by the demand and supply of rides at a given time. So what exactly drives this demand? The first guess would be the time of the day; times around 9 am and 5 pm should see the highest surges on account of people commuting to work/home. Another guess would be the weather; rain/snow should cause more people to take rides.

Content

With no public data of rides/prices shared by any entity, we tried to collect real-time data using Uber&Lyft api queries and corresponding weather conditions. We chose a few hot locations in Boston from this map We built a custom application in Scala to query data at regular intervals and saved it to DynamoDB. The project can be found here on GitHub We queried cab ride estimates every 5 mins and weather data every 1 hr.

The data is approx. for a week of Nov '18 (I actually have included data collected while I was testing the 'querying' application so might have data spread out over more than a week. I didn't consider this as a time-series problem so did not worry about regular interval. The chosen interval was to query as much as data possible without unnecessary redundancy. So data can go from end week of Nov to few in Dec)

The Cab ride data covers various types of cabs for Uber & Lyft and their price for the given location. You can also find if there was any surge in the price during that time. Weather data contains weather attributes like temperature, rain, cloud, etc for all the locations taken into consideration.

Objective

Our aim was to try to analyze the prices of these ride-sharing apps and try to figure out what factors are driving the demand. Do Mondays have more demand than Sunday at 9 am? Do people avoid cabs on a sunny day? Was there a Red Sox match at Fenway that caused more people coming in? We have provided a small dataset as well as a mechanism to collect more data. We would love to see more conclusions drawn.

Problem Statement

The transformations in urban transportation through Uber and Lyft ridesharing have been significant, yet customers experience uncertainties in fare pricing because of changing factors. The proposed model development will create predictions for taxi fares by assessing relevant parameters, which include scheduling data alongside customer demand together with weather elements and transportation specifications. Price variations within the historical data become understandable through the model, which enables service providers and customers to make smart choices.

Operating with extensive data sets poses challenges because processing becomes complex and leads to suboptimal model accuracy because of data abnormalities together with database capacity limitations. The solution to these problems depends on selecting essential features alongside effective data management practices and using resistant machine learning approaches.

1. Data loading and Processing

1.1 Data Cleaning and Merging

In this part, we first converted the `time_stamp` column in both the cab and weather datasets into a readable datetime format using the `as.POSIXct()` function, which allows us to work with time-based features more effectively. Next, we created a new `merge_date` column by combining the source (for cab data) or location (for weather data) with the date and hour extracted from the timestamp. This step ensured that the datasets could be accurately aligned on a common key for merging.

We then aggregated the weather data by hour using the `group_by()` and `summarise()` functions, calculating the mean for key weather variables such as **temperature, clouds, pressure, rain, humidity, and wind**.

In cabe_ride dataset we have 207,921 records, 8 columns. price has 16,535 missing values (~8% of data). and Weather Data:6,276 records, 8 columns. rain has many missing values (~85% missing). To handle missing values, we filled any missing `rain` values with `0`, assuming that the absence of data implies no rain. He filled the median value for price where it's having NA.

Finally, we merged the cab and weather datasets using the `left_join()` function based on the `merge_date` key, and removed any rows with remaining missing values using `na.omit()` to ensure a clean, complete dataset for analysis.

```r
# Load libraries
library(dplyr)
library(lubridate)
library(readr)

# Load cab and weather data
cab_data <- read.csv("Reduced_data.csv")
weather_data <- read.csv("weather.csv")

# Convert timestamps to datetime
cab_data$time_stamp <- as.POSIXct(cab_data$time_stamp, origin = "1970-01-01")
weather_data$time_stamp <- as.POSIXct(weather_data$time_stamp, origin = "1970-01-01")

# Create a merge key (combine source/location with date and hour)
cab_data$merge_date <- paste(cab_data$source,
                             format(cab_data$time_stamp, "%Y-%m-%d"),
                             hour(cab_data$time_stamp),
                             sep = "-")

weather_data$merge_date <- paste(weather_data$location,
                                 format(weather_data$time_stamp, "%Y-%m-%d"),
                                 hour(weather_data$time_stamp),
                                 sep = "-")

# Aggregate weather data by hour
weather_grouped <- weather_data %>%
  group_by(merge_date) %>%
  summarise(across(c(temp, clouds, pressure, rain, humidity, wind), mean, na.rm = TRUE))

# Fill missing 'rain' with 0
weather_grouped$rain[is.na(weather_grouped$rain)] <- 0

# Merge cab data and weather data using the merge key
merged_data <- left_join(cab_data, weather_grouped, by = "merge_date")
```

```
# Handle missing price values with median
merged_data$price[is.na(merged_data$price)] <- median(merged_data$price, na.rm = TRUE)


# drop rows with missing values (if needed)
merged_data <- na.omit(merged_data)

# check result
head(merged_data)
```

Missing Value imputation

```
# Load libraries
library(ggplot2)
library(dplyr)

#  Calculate missing percentage for cab data
cab_missing <- colSums(is.na(cab_data)) / nrow(cab_data) * 100

#  Plot for cab data
ggplot(data = data.frame(Variable = names(cab_missing), Percent = cab_missing),
       aes(x = reorder(Variable, -Percent), y = Percent)) +
  geom_bar(stat = "identity", fill = "skyblue", color = "black") +
  theme_minimal() +
  labs(title = "Percentage of Missing Values in Cab Data", x = "Variable", y = "Percentage
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

#  Calculate missing percentage for weather data
weather_missing <- colSums(is.na(weather_data)) / nrow(weather_data) * 100

#  Plot for weather data
ggplot(data = data.frame(Variable = names(weather_missing), Percent = weather_missing),
       aes(x = reorder(Variable, -Percent), y = Percent)) +
  geom_bar(stat = "identity", fill = "lightcoral", color = "black") +
  theme_minimal() +
  labs(title = "Percentage of Missing Values in Weather Data", x = "Variable", y = "Percen
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

1.2 EDA

Uber and Lyft prices by distance

Interpretation of the Average Price by Distance

The plot shows that both Uber and Lyft prices generally increase with distance, which aligns with the expected ride-sharing pricing model where longer rides cost more. However, Lyft prices exhibit greater variability at shorter distances (under 3 miles), suggesting that Lyft may be more sensitive to demand fluctuations and surge pricing. Uber prices, on the other hand, appear more stable and consistent for shorter rides. For longer distances (beyond 5 miles), Lyft prices tend to rise more steeply compared to Uber, indicating a different long-distance pricing strategy or more aggressive surge pricing. This suggests that Uber may be a more economical choice for shorter trips, while Lyft may become more expensive for longer rides due to increased price sensitivity to demand.

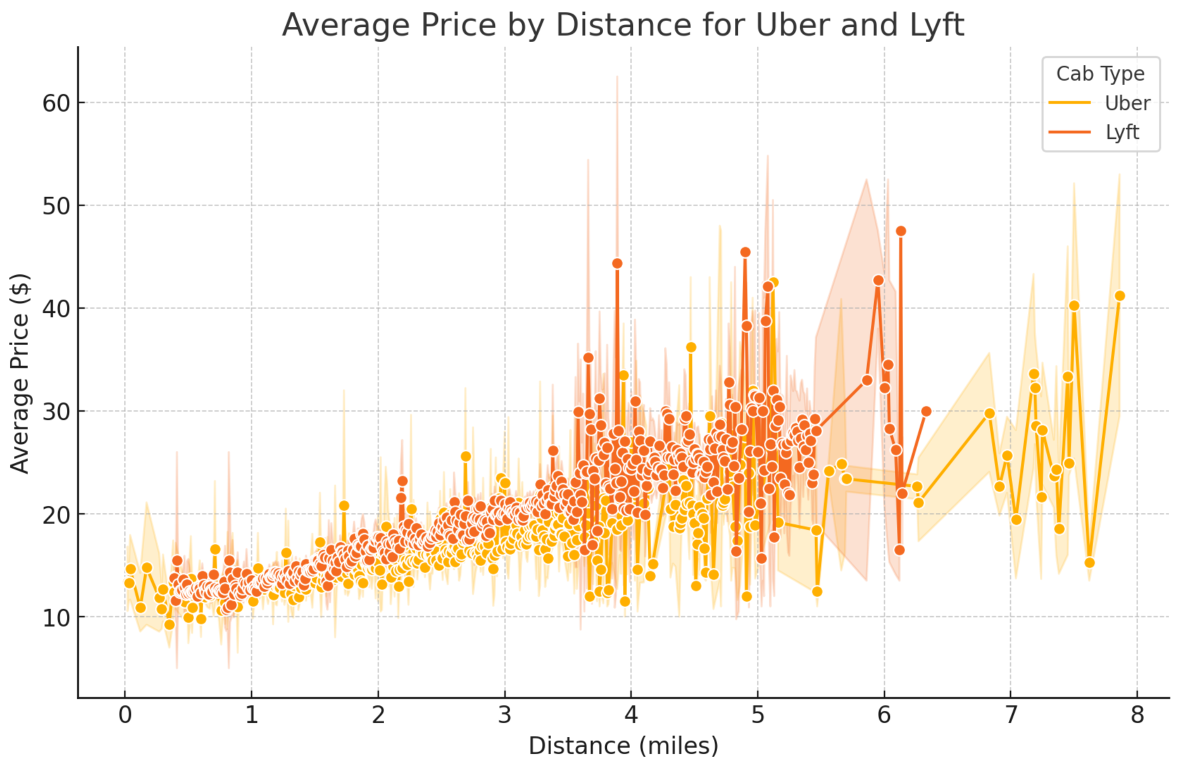

Figure 1: Fig-1

```
#| echo: false
# Load libraries
```

```r
library(ggplot2)

# Plot Uber and Lyft prices by distance
ggplot(merged_data, aes(x = distance, y = price, color = cab_type)) +
  geom_line(size = 1) +  # Increase line thickness
  ggtitle("The Average Price by Distance") +
  xlab("Distance (miles)") +
  ylab("Price ($)") +
  scale_color_manual(values = c("Uber" = "red", "Lyft" = "orange")) +
  theme_minimal() +
  theme(legend.title = element_text(size = 10), legend.text = element_text(size = 9))
```

Interpretation of the Surge Multiplier Effect on Price

The plot shows that as the surge multiplier increases, the median price increases for both Uber and Lyft, which aligns with the dynamic pricing model used by ride-sharing companies. However, Lyft exhibits greater sensitivity to surge pricing. At a surge multiplier of **1.0** (no surge), Lyft has higher median prices and greater variability compared to Uber. As the surge multiplier increases, Lyft prices become more widely spread, indicating greater price sensitivity. In contrast, Uber prices increase more gradually and remain more stable even at higher surge multipliers. At higher surge values (2.0+), Lyft prices rise more steeply and show greater variation, while Uber's prices remain more predictable. This suggests that Uber may offer more consistent pricing, while Lyft may apply more aggressive surge pricing strategies during high-demand periods.

```r
# Load libraries
library(ggplot2)

# Compare surge multiplier effect on price
ggplot(merged_data, aes(x = factor(surge_multiplier), y = price, fill = cab_type)) +
  geom_boxplot() +
  ggtitle("Effect of Surge Multiplier on Price for Uber and Lyft") +
  xlab("Surge Multiplier") +
  ylab("Price ($)") +
  theme_minimal() +
  scale_fill_manual(values = c("Uber" = "skyblue", "Lyft" = "pink"))
```

Don't use

```r
# Load libraries
library(ggplot2)

# Rain vs Price
ggplot(merged_data, aes(x = rain, y = price)) +
  geom_boxplot(fill = "gray") +
  ggtitle("Impact of Rain on Price") +
  xlab("Rain (inches)") +
  ylab("Price ($)") +
  theme_minimal()

# Temperature vs Price
ggplot(merged_data, aes(x = temp, y = price, color = cab_type)) +
  geom_point(alpha = 0.5) +
  ggtitle("Impact of Temperature on Price") +
  xlab("Temperature (°F)") +
  ylab("Price ($)") +
  theme_minimal()

# Cloud Cover vs Price
ggplot(merged_data, aes(x = clouds, y = price, color = cab_type)) +
  geom_point(alpha = 0.5) +
  ggtitle("Impact of Cloud Cover on Price") +
  xlab("Cloud Cover (%)") +
  ylab("Price ($)") +
  theme_minimal()
```

Interpretation of Average Prices by Vehicle Type for Uber and Lyft

The plot shows that for both Uber and Lyft, the average price increases with the size and luxury of the vehicle. For Uber, **UberPool** and **WAV** are the most affordable options, with average prices below **$10**. For Lyft, **Shared** rides are the cheapest, also averaging under **$10**. Standard ride options — **UberX** and **Lyft** — have similar average pricing, around **$10–15**, indicating consistent pricing for everyday rides.

For larger vehicles like **UberXL** and **Lyft XL**, the average price increases to around **$15–20**, reflecting the higher capacity and comfort. Luxury rides are the most expensive; for Uber, **Black** and **Black SUV** rides exceed **$20**, reaching up to **$30**. Lyft's luxury options, such as **Lux Black** and **Lux Black XL**, also have high average prices, with **Lux Black XL** averaging over **$30**.

Lyft shows greater variability in pricing for luxury rides, suggesting more aggressive dynamic pricing for high-end services. In contrast, Uber's luxury rides are more consistent but still expensive. For budget-conscious riders, UberPool and Lyft Shared are ideal options, while for premium comfort, Lyft and Uber's luxury options come at a higher cost.

```r
# Load libraries
library(ggplot2)
library(dplyr)
library(gridExtra)

# Define vehicle type order
uber_order <- c("UberPool", "WAV", "UberX", "UberXL", "Black", "Black SUV")
lyft_order <- c("Shared", "Lyft", "Lyft XL", "Lux", "Lux Black", "Lux Black XL")

#  Uber plot
uber_plot <- ggplot(merged_data %>% filter(cab_type == "Uber"),
                    aes(x = factor(name, levels = uber_order), y = price)) +
  geom_bar(stat = "summary", fun = "mean", fill = "steelblue") +
  ggtitle("The Uber Average Prices by Vehicle Type") +
  xlab("Vehicle Type") +
  ylab("Average Price ($)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1),
        plot.title = element_text(face = "bold"))

#  Lyft plot
lyft_plot <- ggplot(merged_data %>% filter(cab_type == "Lyft"),
                    aes(x = factor(name, levels = lyft_order), y = price)) +
  geom_bar(stat = "summary", fun = "mean", fill = "firebrick") +
  ggtitle("The Lyft Average Prices by Vehicle Type") +
  xlab("Vehicle Type") +
  ylab("Average Price ($)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1),
        plot.title = element_text(face = "bold"))

#  Arrange both plots side by side
gridExtra::grid.arrange(uber_plot, lyft_plot, ncol = 2)
```

Interpretation of Pickup Location vs Price

The plot shows that locations with higher demand, such as **Financial District**, **Fenway**, and **Boston University**, have higher median prices for both Uber and Lyft. This suggests that high-demand business and tourist areas trigger higher surge pricing. For most pickup locations, Lyft tends to have higher median prices than Uber, with the price difference being more noticeable at high-traffic locations like **Financial District** and **Fenway**.

Pricing is more stable at residential and low-traffic areas like **North End**, **North Station**, and **West End**, where price variation is smaller, indicating lower surge pricing or less demand. However, locations like **Beacon Hill**, **Boston University**, and **Fenway** show a wide interquartile range (IQR) and higher outliers, reflecting more aggressive surge pricing or demand spikes during peak hours.

The overall price variation is larger for Lyft, suggesting greater sensitivity to demand and surge pricing. Uber's pricing, on the other hand, remains more consistent across most locations. For more predictable pricing, Uber may be a better option in busy areas, while Lyft's aggressive surge pricing may result in higher fares during peak demand.

```r
# Load libraries
library(ggplot2)

# Create box plot of pickup location vs price with 10 intervals on y-axis
ggplot(merged_data, aes(x = source, y = price, fill = cab_type)) +
  geom_boxplot(outlier.shape = NA, position = position_dodge(width = 0.75)) +
  ggtitle("Distribution of Pickup Location vs Price") +
  xlab("Pickup Location") +
  ylab("Price ($)") +
  scale_fill_manual(values = c("Uber" = "steelblue", "Lyft" = "lightblue")) +
  scale_y_continuous(breaks = seq(0, 100, by = 10)) + # 10 intervals on y-axis
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        plot.title = element_text(face = "bold", size = 14),
        legend.title = element_text(size = 10),
        legend.text = element_text(size = 9))
```

2 Modeling

2.1 Splitting the Data

```
library(reticulate)
py_config()
```

Data Preprocessing Summary

The code splits the data into **80% training** and **20% testing** using the `createDataPartition()` function from the `caret` package, ensuring reproducibility with `set.seed(42)`. It separates predictors (`X_train`, `X_test`) and target variables (`y_train`, `y_test`) by excluding the `price` column from the predictors. Categorical variables are converted to factors using `lapply()` to enable proper encoding. The `model.matrix()` function then performs one-hot encoding, converting categorical features into binary numeric columns to make the data suitable for machine learning models. The processed data is now numeric and ready for model training.

```
# Load required libraries
library(dplyr)
library(caret)
library(lubridate)
library(randomForest)
library(glmnet)

#   Step 1: Convert timestamps to datetime

str(cab_data$time_stamp)

cab_data$time_stamp <- as.POSIXct(cab_data$time_stamp / 1000, origin = "1970-01-01")
weather_data$time_stamp <- as.POSIXct(weather_data$time_stamp, origin = "1970-01-01")

#   Step 2: Extract Time-Based Features
cab_data$day_of_week <- wday(cab_data$time_stamp, label = TRUE)  # Day of the week
cab_data$hour_of_day <- hour(cab_data$time_stamp)

weather_data$day_of_week <- wday(weather_data$time_stamp, label = TRUE)
weather_data$hour_of_day <- hour(weather_data$time_stamp)

#   Step 3: Handle Missing Values (Impute with Median)
cab_data$price[is.na(cab_data$price)] <- median(cab_data$price, na.rm = TRUE)
```

```
weather_data <- weather_data %>%
  mutate(across(where(is.numeric), ~ifelse(is.na(.), median(., na.rm = TRUE), .)))

#  Step 4: Merge Datasets
merged_data <- left_join(
  cab_data,
  weather_data,
  by = c("time_stamp", "day_of_week", "hour_of_day")
)

#  Step 5: Drop Unnecessary Columns
merged_data <- merged_data %>%
  select(-c(id, product_id, time_stamp, destination, source))

#  Step 6: One-Hot Encoding of Categorical Variables
cat_features <- c("cab_type", "name", "day_of_week")
num_features <- c("distance", "surge_multiplier", "temp", "clouds", "pressure", "rain", "h

# Convert to factors
merged_data[cat_features] <- lapply(merged_data[cat_features], as.factor)

# One-hot encode categorical features
dummy <- dummyVars(~ ., data = merged_data[cat_features])
encoded_data <- predict(dummy, newdata = merged_data)

# Combine encoded and numeric data
final_data <- cbind(encoded_data, merged_data[num_features], price = merged_data$price)

#  Step 7: Remove Outliers Using IQR
Q1 <- quantile(final_data$price, 0.25)
Q3 <- quantile(final_data$price, 0.75)
IQR <- Q3 - Q1
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR
final_data <- final_data %>%
  filter(price >= lower_bound & price <= upper_bound)

#  Step 8: Train-Test Split (80-20)
set.seed(42)
train_index <- createDataPartition(final_data$price, p = 0.8, list = FALSE)
train <- final_data[train_index, ]
```

```r
test <- final_data[-train_index, ]

#  Step 9: Standardization
pre_process <- preProcess(train[, -ncol(train)], method = c("center", "scale"))
X_train <- predict(pre_process, train[, -ncol(train)])
X_test <- predict(pre_process, test[, -ncol(test)])
y_train <- train$price
y_test <- test$price

#  Step 10: Fill Remaining NA (Median)
X_train <- apply(X_train, 2, function(x) ifelse(is.na(x), median(x, na.rm = TRUE), x))
X_test <- apply(X_test, 2, function(x) ifelse(is.na(x), median(x, na.rm = TRUE), x))

#  Step 11: Train Initial Model (Random Forest)
set.seed(42)
rf_model <- randomForest(X_train, y_train)

#  Step 12: Evaluate Initial Model
y_pred <- predict(rf_model, X_test)

mae <- mean(abs(y_test - y_pred))
mse <- mean((y_test - y_pred)^2)
r2 <- 1 - sum((y_test - y_pred)^2) / sum((y_test - mean(y_test))^2)

cat("\nRandom Forest Initial Model Performance:\n")
cat("MAE:", round(mae, 2), "\n")
cat("MSE:", round(mse, 2), "\n")
cat("R2:", round(r2, 2), "\n")

#  Step 13: Present Results
results <- data.frame(
  Model = "Random Forest",
  MAE = mae,
  MSE = mse,
  R2 = r2
)

print(results)
```

```r
# Load required libraries
library(dplyr)
library(caret)
library(glmnet)




#   Step 1: Select Features (Added surge_multiplier)

data <- merged_data %>%
  mutate(date_time = as.POSIXct(time_stamp, origin = "1970-01-01")) %>%
  select(distance, cab_type, destination, source, price, name, date_time, surge_multiplier

#   Step 2: Create Time Period Variable
data$hour <- format(data$date_time, "%H")
data$time_period <- case_when(
  data$hour %in% c("06", "07", "08", "09") ~ "morning",
  data$hour %in% c("10", "11", "12", "13") ~ "noon",
  data$hour %in% c("14", "15", "16", "17") ~ "afternoon",
  data$hour %in% c("18", "19", "20", "21") ~ "evening",
  data$hour %in% c("22", "23", "00", "01", "02") ~ "night",
  data$hour %in% c("03", "04", "05") ~ "late_night"
)

# Drop unnecessary columns
data <- data %>% select(-c(date_time, hour))

#   Step 3: Train-Test Split (80-20 split)
set.seed(42)
train_index <- createDataPartition(data$price, p = 0.8, list = FALSE)
train <- data[train_index, ]
test <- data[-train_index, ]

#   Step 4: Preprocessing Pipeline (One-Hot Encoding + Scaling)
cat_features <- c("cab_type", "destination", "source", "name", "time_period")
num_features <- c("distance", "surge_multiplier", "temp", "clouds", "pressure", "rain", "h

preprocessor <- preProcess(train[, cat_features], method = c("center", "scale"))
train_preprocessed <- predict(preprocessor, train)
test_preprocessed <- predict(preprocessor, test)

#   Step 5: Combine Preprocessed Data
```

```
train_final <- cbind(train_preprocessed, train[, num_features], price = train$price)
test_final <- cbind(test_preprocessed, test[, num_features], price = test$price)



#  make column names unique
colnames(train_final) <- make.names(colnames(train_final), unique = TRUE)
colnames(test_final) <- make.names(colnames(test_final), unique = TRUE)



#   Step 6: Linear Regression (Base Model)
set.seed(42)
lin_reg <- train(
  price ~ .,
  data = train_final,
  method = "lm",
  trControl = trainControl(method = "cv", number = 5)
)

#   Step 7: Randomized Search for Linear Regression
rand_grid <- expand.grid(
  intercept = c(TRUE, FALSE),
  copy = c(TRUE, FALSE),
  positive = c(TRUE, FALSE)
)

set.seed(42)
lin_reg_random <- train(
  price ~ .,
  data = train_final,
  method = "lm",
  trControl = trainControl(method = "cv", number = 3, search = "random"),
  tuneLength = 4
)

#   Step 8: Performance Evaluation
train_pred <- predict(lin_reg, newdata = test_final)
rmse <- sqrt(mean((test_final$price - train_pred)^2))
r2 <- 1 - sum((test_final$price - train_pred)^2) / sum((test_final$price - mean(test_final
cat("\nLinear Regression Performance:\n")
cat("RMSE:", round(rmse, 2), "\n")
cat("R2:", round(r2, 2), "\n")

#  Step 9: Present Results
results <- data.frame(
  Model = c("Linear Regression"),
  RMSE = c(rmse),
  R2 = c(r2)
)

print(results)
```

```
# Load required libraries
library(dplyr)
library(caret)
library(glmnet)
library(randomForest)
library(xgboost)

#  Step 1: Select Features (Added surge_multiplier)
data <- merged_data %>%
  select(distance, cab_type, destination, source, price, name, surge_multiplier, temp, clo

#  Step 2: Create Time Period Variable
data$hour <- format(data$time_stamp, "%H")
data$time_period <- case_when(
  data$hour %in% c("06", "07", "08", "09") ~ "morning",
  data$hour %in% c("10", "11", "12", "13") ~ "noon",
  data$hour %in% c("14", "15", "16", "17") ~ "afternoon",
  data$hour %in% c("18", "19", "20", "21") ~ "evening",
  data$hour %in% c("22", "23", "00", "01", "02") ~ "night",
  data$hour %in% c("03", "04", "05") ~ "late_night"
)

# Drop unnecessary columns
data <- data %>% select(-c(hour))

#  Step 3: Train-Test Split (80-20 split)
set.seed(42)
train_index <- createDataPartition(data$price, p = 0.8, list = FALSE)
```

```
train <- data[train_index, ]
test <- data[-train_index, ]

#  Step 4: Preprocessing Pipeline (One-Hot Encoding + Scaling)
cat_features <- c("cab_type", "destination", "source", "name", "time_period")
num_features <- c("distance", "surge_multiplier", "temp", "clouds", "pressure", "rain", "h

# Convert character variables to factors
train[cat_features] <- lapply(train[cat_features], as.factor)
test[cat_features] <- lapply(test[cat_features], as.factor)

# Remove categorical variables with only one level
cat_features <- cat_features[sapply(train[cat_features], function(x) length(unique(x)) > 1

# One-hot encode categorical variables
dummy <- dummyVars(~ ., data = train[, cat_features], fullRank = TRUE)
train_encoded <- predict(dummy, newdata = train)
test_encoded <- predict(dummy, newdata = test)



# Combine encoded + numeric features
train_final <- cbind(train_encoded, train[, num_features], price = train$price)
test_final <- cbind(test_encoded, test[, num_features], price = test$price)

#  Step 5: Remove Duplicates and Make Column Names Unique
train_final <- train_final[, !duplicated(names(train_final))]
test_final <- test_final[, !duplicated(names(test_final))]
colnames(train_final) <- make.names(colnames(train_final), unique = TRUE)
colnames(test_final) <- make.names(colnames(test_final), unique = TRUE)

#  Step 6: RANDOM FOREST MODEL
set.seed(42)
rf_model <- train(
  price ~ .,
  data = train_final,
  method = "rf",
  trControl = trainControl(method = "cv", number = 3), # 3-fold cross-validation
  tuneLength = 3 # Tune over 3 combinations of mtry
)
```

```
#  Step 7: ELASTIC NET MODEL
set.seed(42)
en_model <- train(
  price ~ .,
  data = train_final,
  method = "glmnet",
  trControl = trainControl(method = "cv", number = 3),
  tuneLength = 5 # Search over 5 combinations of alpha and lambda
)

#  Step 8: XGBOOST MODEL
set.seed(42)
xgb_grid <- expand.grid(
  nrounds = c(50, 100),       # Number of boosting rounds
  max_depth = c(3, 5),        # Maximum tree depth
  eta = c(0.01, 0.1),         # Learning rate
  gamma = 0,                  # Minimum loss reduction
  colsample_bytree = 0.7,     # Fraction of features to use in each tree
  min_child_weight = 1,       # Minimum sum of instance weight (hessian)
  subsample = 0.8             # Subsample ratio of the training instance
)

xgb_model <- train(
  price ~ .,
  data = train_final,
  method = "xgbTree",
  trControl = trainControl(method = "cv", number = 3),
  tuneGrid = xgb_grid
)

#  Step 9: Performance Evaluation
# Random Forest
rf_pred <- predict(rf_model, newdata = test_final)
rf_rmse <- sqrt(mean((test_final$price - rf_pred)^2))
rf_r2 <- 1 - sum((test_final$price - rf_pred)^2) / sum((test_final$price - mean(test_final

# Elastic Net
en_pred <- predict(en_model, newdata = test_final)
en_rmse <- sqrt(mean((test_final$price - en_pred)^2))
en_r2 <- 1 - sum((test_final$price - en_pred)^2) / sum((test_final$price - mean(test_final

# XGBoost
```

```
xgb_pred <- predict(xgb_model, newdata = test_final)
xgb_rmse <- sqrt(mean((test_final$price - xgb_pred)^2))
xgb_r2 <- 1 - sum((test_final$price - xgb_pred)^2) / sum((test_final$price - mean(test_fin

#  Step 10: Present Results
results <- data.frame(
  Model = c("Random Forest", "Elastic Net", "XGBoost"),
  RMSE = c(rf_rmse, en_rmse, xgb_rmse),
  R2 = c(rf_r2, en_r2, xgb_r2)
)

print(results)
```

```
# Load required libraries
library(caret)
library(randomForest)
library(glmnet)

# Split into training and test set (80% train, 20% test)
set.seed(42)
train_index <- createDataPartition(merged_data$price, p = 0.8, list = FALSE)
train <- merged_data[train_index, ]
test <- merged_data[-train_index, ]

# Define predictors and target
X_train <- train %>% select(-price)
y_train <- train$price
X_test <- test %>% select(-price)
y_test <- test$price

# Convert categorical variables to factors
X_train[] <- lapply(X_train, function(x) if(is.character(x)) as.factor(x) else x)
X_test[] <- lapply(X_test, function(x) if(is.character(x)) as.factor(x) else x)

# One-hot encode categorical variables
dummy_train <- model.matrix(~ . - 1, data = X_train)
dummy_test <- model.matrix(~ . - 1, data = X_test)
```

New

```
# Load required libraries
library(caret)
library(randomForest)
library(glmnet)

# Split into training and test set (80% train, 20% test)
set.seed(42)
train_index <- createDataPartition(merged_data$price, p = 0.8, list = FALSE)
train <- merged_data[train_index, ]
test <- merged_data[-train_index, ]

# Define predictors and target
X_train <- train %>% select(-price)
y_train <- train$price
X_test <- test %>% select(-price)
y_test <- test$price

# Convert categorical variables to factors
X_train[] <- lapply(X_train, function(x) if(is.character(x)) as.factor(x) else x)
X_test[] <- lapply(X_test, function(x) if(is.character(x)) as.factor(x) else x)

# One-hot encode categorical variables
dummy_train <- model.matrix(~ . - 1, data = X_train)
dummy_test <- model.matrix(~ . - 1, data = X_test)

# Remove low-variance predictors to reduce dimensionality
nzv <- nearZeroVar(dummy_train)
if (length(nzv) > 0) {
  dummy_train <- dummy_train[, -nzv]
  dummy_test <- dummy_test[, -nzv]
}
```

Model

```
# Initialize result dataframe
results <- data.frame(Model = character(), RMSE = numeric(), R2 = numeric(), stringsAsFact

# 1. Ridge Regression (Linear Regression with Regularization)
set.seed(42)
lr_model <- cv.glmnet(as.matrix(dummy_train), y_train, alpha = 0) # Ridge (alpha = 0)
lr_pred <- predict(lr_model, newx = as.matrix(dummy_test), s = "lambda.min")
```

```
lr_rmse <- sqrt(mean((y_test - lr_pred) ^ 2))
lr_r2 <- 1 - (sum((y_test - lr_pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))
results <- rbind(results, data.frame(Model = "Ridge Regression", RMSE = lr_rmse, R2 = lr_r

# 2. Random Forest (Hyperparameter tuned)
set.seed(42)
rf_model <- randomForest(x = dummy_train,
                         y = y_train,
                         ntree = 50,
                         mtry = 3,
                         maxnodes = 30,
                         nodesize = 5)
rf_pred <- predict(rf_model, newdata = dummy_test)
rf_rmse <- sqrt(mean((y_test - rf_pred) ^ 2))
rf_r2 <- 1 - (sum((y_test - rf_pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))
results <- rbind(results, data.frame(Model = "Random Forest", RMSE = rf_rmse, R2 = rf_r2))

# 3. Elastic Net (Hyperparameter tuned)
set.seed(42)
en_model <- cv.glmnet(as.matrix(dummy_train),
                      y_train,
                      alpha = 0.5,
                      lambda = 10^seq(-3, 1, length = 100),
                      nfolds = 5)
en_pred <- predict(en_model, newx = as.matrix(dummy_test), s = "lambda.min")
en_rmse <- sqrt(mean((y_test - en_pred) ^ 2))
en_r2 <- 1 - (sum((y_test - en_pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))
results <- rbind(results, data.frame(Model = "Elastic Net", RMSE = en_rmse, R2 = en_r2))

# Display Results
print(results)
```

```
# Load required libraries
library(caret)
library(randomForest)
library(glmnet)

# Split into training and test set (80% train, 20% test)
set.seed(42)
train_index <- createDataPartition(merged_data$price, p = 0.8, list = FALSE)
train <- merged_data[train_index, ]
```

```r
test <- merged_data[-train_index, ]

# Define predictors and target
X_train <- train %>% select(-price)
y_train <- train$price
X_test <- test %>% select(-price)
y_test <- test$price

# Convert categorical variables to factors
X_train[] <- lapply(X_train, function(x) if(is.character(x)) as.factor(x) else x)
X_test[] <- lapply(X_test, function(x) if(is.character(x)) as.factor(x) else x)

# One-hot encode categorical variables
dummy_train <- model.matrix(~ . - 1, data = X_train)
dummy_test <- model.matrix(~ . - 1, data = X_test)

# Initialize result dataframe
results <- data.frame(Model = character(), RMSE = numeric(), R2 = numeric(), stringsAsFact

# Define models
models <- list(
  "Linear Regression" = lm(y_train ~ ., data = as.data.frame(dummy_train)),
  "Random Forest" = randomForest(x = dummy_train, y = y_train, ntree = 100, mtry = 5, impo
  "Elastic Net" = cv.glmnet(as.matrix(dummy_train), y_train, alpha = 0.5)
)

# Loop over models and evaluate performance
for (name in names(models)) {
  model <- models[[name]]

  if (name == "Elastic Net") {
    pred <- predict(model, newx = as.matrix(dummy_test), s = "lambda.min")
  } else {
    pred <- predict(model, newdata = as.data.frame(dummy_test))
  }

  # Compute RMSE and R-squared
  rmse <- sqrt(mean((y_test - pred) ^ 2))
  r2 <- 1 - (sum((y_test - pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))

  # Append results
  results <- rbind(results, data.frame(Model = name, RMSE = rmse, R2 = r2))
}

# Display results
print(results)
```

2.2 Fitting Model

```
# Linear Regression
lr_model <- lm(y_train ~ ., data = as.data.frame(dummy_train))
lr_pred <- predict(lr_model, newdata = as.data.frame(dummy_test))

# Compute RMSE and R2
lr_rmse <- sqrt(mean((y_test - lr_pred) ^ 2))
lr_r2 <- 1 - (sum((y_test - lr_pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))

# Display Results
cat("Linear Regression Results:\n")
cat("RMSE:", lr_rmse, "\n")
cat("R2:", lr_r2, "\n")
```

```
# Model 1: Linear Regression
lr_model <- lm(y_train ~ ., data = as.data.frame(dummy_train))
lr_pred <- predict(lr_model, newdata = as.data.frame(dummy_test))
lr_rmse <- sqrt(mean((y_test - lr_pred)^2))

# Model 2: Random Forest
rf_model <- randomForest(x = dummy_train, y = y_train, ntree = 100, mtry = 5, importance =
rf_pred <- predict(rf_model, newdata = dummy_test)
rf_rmse <- sqrt(mean((y_test - rf_pred)^2))

# Model 3: Elastic Net
# Convert to matrix format for glmnet
X_train_mat <- as.matrix(dummy_train)
X_test_mat <- as.matrix(dummy_test)

# Fit Elastic Net using cross-validation
en_model <- cv.glmnet(X_train_mat, y_train, alpha = 0.5)  # alpha = 0.5 for Elastic Net
en_pred <- predict(en_model, newx = X_test_mat, s = "lambda.min")
```

```r
en_rmse <- sqrt(mean((y_test - en_pred)^2))

# Compare Model Performance
results <- data.frame(
  Model = c("Linear Regression", "Random Forest", "Elastic Net"),
  RMSE = c(lr_rmse, rf_rmse, en_rmse)
)

print(results)
```

Include PCA

```r
# Load required libraries
library(caret)
library(randomForest)
library(glmnet)

# Split into training and test set (80% train, 20% test)
set.seed(42)
train_index <- createDataPartition(merged_data$price, p = 0.8, list = FALSE)
train <- merged_data[train_index, ]
test <- merged_data[-train_index, ]

# Define predictors and target
X_train <- train %>% select(-price)
y_train <- train$price
X_test <- test %>% select(-price)
y_test <- test$price

# Convert categorical variables to factors
X_train[] <- lapply(X_train, function(x) if(is.character(x)) as.factor(x) else x)
X_test[] <- lapply(X_test, function(x) if(is.character(x)) as.factor(x) else x)

# One-hot encode categorical variables
dummy_train <- model.matrix(~ . - 1, data = X_train)
dummy_test <- model.matrix(~ . - 1, data = X_test)

#  Remove low-variance predictors to reduce dimensionality
nzv <- nearZeroVar(dummy_train)
if (length(nzv) > 0) {
  dummy_train <- dummy_train[, -nzv]
  dummy_test <- dummy_test[, -nzv]
}

#  Subsample the training data to speed up computation (5000 samples)
set.seed(42)
sample_index <- sample(1:nrow(dummy_train), 5000)
dummy_train <- dummy_train[sample_index, ]
y_train <- y_train[sample_index]

#  Initialize result dataframe
results <- data.frame(Model = character(), RMSE = numeric(), R2 = numeric(), stringsAsFact

#  Linear Regression
lr_model <- lm(y_train ~ ., data = as.data.frame(dummy_train))
lr_pred <- predict(lr_model, newdata = as.data.frame(dummy_test))
lr_rmse <- sqrt(mean((y_test - lr_pred) ^ 2))
lr_r2 <- 1 - (sum((y_test - lr_pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))
results <- rbind(results, data.frame(Model = "Linear Regression", RMSE = lr_rmse, R2 = lr_

#  Random Forest (Reduce ntree and mtry)
set.seed(42)
rf_model <- randomForest(x = dummy_train, y = y_train, ntree = 50, mtry = 3, importance =
rf_pred <- predict(rf_model, newdata = dummy_test)
rf_rmse <- sqrt(mean((y_test - rf_pred) ^ 2))
rf_r2 <- 1 - (sum((y_test - rf_pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))
results <- rbind(results, data.frame(Model = "Random Forest", RMSE = rf_rmse, R2 = rf_r2))

#  Elastic Net (Reduce number of folds)
set.seed(42)
en_model <- cv.glmnet(as.matrix(dummy_train), y_train, alpha = 0.5, nfolds = 5)
en_pred <- predict(en_model, newx = as.matrix(dummy_test), s = "lambda.min")
en_rmse <- sqrt(mean((y_test - en_pred) ^ 2))
en_r2 <- 1 - (sum((y_test - en_pred) ^ 2) / sum((y_test - mean(y_test)) ^ 2))
results <- rbind(results, data.frame(Model = "Elastic Net", RMSE = en_rmse, R2 = en_r2))

#  Display Results
print(results)
```