INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT046-3-M-AML

APPLIED MACHINE LEARNING

ASSIGNMENT REPORT

HAND OUT DATE: 17 JUNE 2021

HAND IN DATE: 6 SEP 2021

INSTRUCTIONS TO CANDIDATES:

1 **Submit your assignment at the administrative counter.**

2 **Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing).**

3 **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**

4 **Cases of plagiarism will be penalized.**

5 **The assignment should be bound in an appropriate style (comb bound or stapled).**

6 **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**

7 **You must obtain 50% overall to pass this module.**

Abstract

In this technology-driven era, ride-sharing services like Grab and Uber has been growing in popularity, mainly due to the convenience provided to the users. Users no longer have to wait for public transports like LRT or bus for a long time and can just request a ride from their mobile phones. However, despite the convenience, it has been a concern when the prices of the rides can surge high to the extent of causing anxiety and discomfort to the users. With the pricing uncertainty, users are unable to estimate the fare of the rides and can only guess the prices based on their own experiences. Machine learning has been helping humans to provide better solutions in other different domains. Thus, this report subsequently investigates the application and evaluation of machine learning algorithms applied on existing rides and weather data, and thus finds the best performing algorithm to be implemented in ride price prediction in the ride-sharing domain by using the details of the rides such as distance, time and weather conditions etc.

Table of Contents

Table of Figures

Chapter 1: Introduction

1.1 Introduction

In recent years, ride-sharing services such as Uber and Grab have grown tremendously in popularity, due to their convenience, as well as their flexible and reasonable pricing. Besides, they also appeal to drivers who want to drive their own cars more flexibly.

One of the core and unique characteristics in ride-sharing service is dynamic pricing, which is used to balance the supply (numbers of cars on the road) and demand (number of passengers' requests. In a busy area, a higher price can reduce demand and increase supply. Dynamic pricing consists of 2 parts, which is the fixed normal price (calculated based on the distance and time of the trip) and the price multiplier (calculated based on the supply and demand condition at the particular area).

The dynamic price model sometimes can give mental stress on passengers and makes them not happy with their rides. By using traditional taxi service, passengers can estimate their ride fare based on fixed pricing and personal experience. However, in ride-sharing service, after factoring in the price multiplier in the dynamic model, passengers have to guess the price multiplier based on their estimation of the supply and demand nearby. Without relevant information, the estimate can be inaccurate and prevents the passenger from making informed decisions.

By making more information available to passengers, like explaining why the price is high or low and price forecast for next time slot in nearby places can help ease their anxiety in hesitation to make decisions.

1.2 Problem Statement

The uncertainty of the dynamic prices gives mental anxiety to the passengers because they are unable to know whether the dynamic price multiplier will become lower or not if they wait for some time, or whether they can get a lower price if try to walk for a few minutes to another location to get a ride.

1.3 Research Goal

The goal of this research is to develop machine learning approach to predict the dynamic prices of RoD services to help improve the user experience while using the services. The second goal is to find the most important factor that contribute the most in the dynamic price model and optimize the model with model evaluation method by using historical data.

1.4 Research Objectives

- To develop and compare multiple machine learning approach in ride-sharing price prediction
- To finetune and optimize the model for higher accuracy
- To investigate the correlation between factors of a ride and select the most important factor affecting the price surge

Chapter 2: Literature Review

2.1 Introduction

This literature review introduces investigation and discussion for related work on price surge modelling topic such as Uber and Lyft rides (Kaggle, 2019) and Dynamic Price Prediction of Ride-on-Demand (Guo et al., 2017) It consists of different kernel and technique comparison from different author and how well they perform the analysis and technique to increase the accuracy. Moreover, the conducted summary part could help to indicate the gap and list down variety of scope that can be used for implementation experiment.

2.2 Related Work

Citation	Dataset	Size	EDA	Pre-processing	Model	Fine Tuning	Score	Comments
Kunal, Arora & Kaur, Sharanjit & Sharma, Vinod. (2021)	Rapido Dataset	N/A	- Correlation matrix - Outlier analysis and removal - Plot missing data	- Calculate average of travel duration	- Linear Regression	- Scatter plot of true values vs predicted values - Train-test (80:20) split	ACC: R^2 = 93.40%	- Only numeric data - Use other machine learning algorithms
Guo et al., 2017	Shenzhou UCar	5.3 million x 6	- Plot price multiplier variation by hourly	- Characterize regularity of price multiplier and find maximum predictability	- Markov-chain - Neural network	- 75% training dataset for Markov-chain model - 500 times for each area for NN model	- Symmetric mean absolute percentage error, sMAPE = 0.0448 - NN predictor better than Markov-chain	- Add local events information to improve NN model - Use deep learning techniques
Umairkhan (kaggle.com), 2021	Uber & Lyft	693071 x 10	- Correlation matrix - Plot daily surge multiplier - Scatter plot	- Check missing data - Remove unwanted data columns - Rescale continuous feature data	N/A	N/A	N/A	- Only EDA done
Gabriel A. (kaggle.com), 2021	Uber & Lyft	693071 x 10	- One-hot encoding - Binary encoding for both Uber/Lyft	- Remove missing data - Combine average weather data	- Linear Regression	- Train-test (70:30) split	ACC: R^2 = 92.87%	
RaviMunde (kaggle.com), 2019	Uber & Lyft	693071 x 10	- N/A		- Random Forest	- Train-test (75:25) split - to balance imbalance data	ACC: 97.47%	
Hamed E. (kaggle.com), 2021	NYC Taxi Fare	1000000 x 8	- Plot data imbalance - Heat map	- Remove missing & duplicate data - Outlier removal	- Linear Regression - Decision Tree Regressor	- Train-test (80:20) split	RME = 4.533 (LR) RME = 5.268 (DT) RME = 3.638 (RF)	- Random Forest model perform the best

			- Plot daily & hourly trip count - Correlation matrix		- Random Forest Regressor			

2.3 Comparison of Previous Methods

Table 1 describes some related past work reports on study of price prediction models on ride-sharing datasets. Kunal (2021) did price prediction works of ride-on-demand services by using dataset from Rapido, an online bike taxi RoD service in India. They used the linear regression model as their machine learning model and achieved an efficiency of R^2=93.40%. (Kunal *et al.*, 2021)

On the other hand, Guo et al. (2017) did study on the dynamic pricing model based on Shenzhou Ucar datasets. They utilized Markov-chain and neural network as their approaches in building the machine learning algorithm. They concluded that neural network performed better than Markov chain model and is able to achieve an accuracy score with sMAPE (symmetric mean absolute percentage error) of 0.0448. (Guo *et al.*, 2017)

Besides, there are also a couple of past works done on the chosen dataset by the Kaggle community. Umairkhan (2021) did some EDA works like correlation matrix and scatter plot to explore the dataset. Gabriel (2021) did price prediction works by using the linear regression and able to achieve an accuracy of 92.87%. Hamed E. used New York City taxi fare dataset to try 3 different approaches, namely the linear regression, random forest and decision tree methods to compare their performances. It turns out that Random Forest performed best with mean square error of 3.638. This can serve as an indicator of which model can perform best with the chosen dataset for this study.

Most of the past studies uses train-test data split with ratio of 80:20 although Gabriel and Ravimunde uses ratio of 70:30 and 75:25 respectively. This could give some hints about the different ratio used in improving the accuracy of the model. However, the researchers do not use the same measure of accuracy to conclude their findings. Thus, this report will be using the same measure of accuracy to better compare the different machine learning approaches.

2.4 Summary

Model	Train Test Split	Data Standardization	Parameter Tuning	Accuracy
Linear Regression				
Kunal, 2021	Yes	Yes	No	R^2 = 93.40%
Gabriel, 2021	Yes	Yes	No	R^2 = 92.87%
Hamed, 2021	Yes	Yes	No	RME = 4.533
Random Forest				
Ravimunde, 2021	Yes	Yes	No	ACC: 97.47%
Hamed, 2021	Yes	Yes	No	RME = 3.638
Decision Tree				
Hamed, 2021	Yes	Yes	No	RME = 5.268

2.4.1 Gap Analysis

Most of the related works focus on predicting the ride-sharing price based on location, distance and time taken of the rides. However, they pay less attention of the weather factors that may also affect the price surge in ride-sharing services. Therefore, consideration of weather conditions can be taken into account when predicting the prices. In the proposed work, it is going to investigate the correlation between weather conditions and the price surge of ride-sharing services.

2.4.2 Scope Analysis

Based on the analysis above, there are few things that can help us to reduce the scope while achieving the objective:

- Perform data merging between weather dataset and rides fare dataset
- Perform feature selection function to find out the most important factor affecting the prices
- Perform parameter tuning for machine learning model used.

Chapter 3: Data Preparation

3.1 Problem Description

This assignment is to create a model for price prediction in ride-sharing so that users are able to gauge the estimated price before taking the rides. Important features affecting the price will be identified for the model building.

3.2 Dataset Used

The dataset chosen for this report is obtained from Kaggle (2019). It is a dataset consisting of details of Uber and Lyft rides. There is total of 2 dataset tables, which are cab rides and weather dataset. (Kaggle, 2019)

The cab rides dataset consists of information of rides, like destination, time and price. Every row in the datasets represent the details of every different rides. There are 10 columns in the cab rides dataset. The main objective is to predict the prices of the ride based on the details of the rides.

The weather dataset is made up of 8 columns. It has details of the weather conditions of different locations at different times. The relationship between the weather conditions and the prices of rides are to be investigated in this report.

Dataset URL: https://www.kaggle.com/ravi72munde/uber-lyft-cab-prices

3.3 Data Features Description

Cab_rides dataset

#	Attribute	Data Type	Description
1	distance	Numerical	distance travelled of the ride
2	cab_type	Categorical	Uber or Lyft ride
3	time_stamp	Date	epoch time when data was queried
4	destination	Categorical	the ending point of the ride
5	source	Categorical	beginning point of the ride
6	price	Numerical	estimation of price for the ride (USD)
7	surge_multiple	Numerical	the multiplier by which price was increased (default 1)
8	id	Character	unique identifier
9	product_id	Character	uber/lyft identifier for cab type
10	name	Categorical	Name of the cab type eg: Uber Pool, UberXL

Weather dataset

#	Attribute	Data Type	Description
1	temp	Numerical	Temperature (F)
2	location	Character	Location Name
3	clouds	Numerical	Clouds
4	pressure	Numerical	Pressure (mb)
5	rain	Numerical	Rain in inches for the last hour
6	time_stamp	Date	Epoch time when row data was collected
7	Humidity	Numerical	Humidity (%)
8	Wind	Numerical	Wind speed (mph)

3.4 Machine Learning Methods

There are many machine learning models available like linear regression, decision trees, support vector machines etc. However, models like decision trees and random forest are not suitable to use in this study because they are mainly used for classification problems and the target variable “price” is not a binary data. Therefore, the models to be used in this study are linear regression, ridge regression, lasso regression, elastic regression, and support vector machines.

3.4.1 Linear Regression

Linear Regression is a machine learning model in which independent variables predict dependent variables. Regression analysis calculates the value of the dependent variable 'y' based on the range of independent variable values 'x'. (Maulud and Abdulazeez, 2020). Among the common linear regression models are simple linear regression and multivariate linear regression.

Simple linear regression is a model with only a single independent variable and the variable can be defined by the model, $y=\beta 0+\beta 1x+\varepsilon$. The influence of independent variables is distinguished from the interaction of dependent variables in simple regression.

Multivariate linear regression (MLR) is a statistical technique that uses a number of explanatory variables to predict the outcome of an answer variable. The goal of MLR is to model the linear relationship that will be analysed between the independent variables x and y. Stepwise Linear Regression (WLR) allowed variables to be added or removed as the model was being built. Forward Stepwise Linear Regression (FLR) began with a null model, then at each subsequent step, the predictor that produced the greatest increase in adjusted R^2 was added to the model. This procedure was repeated until the model adjusted R^2 ceased to maximise. Backward stepwise Linear Regression (BLR) began with all variables in the model and deletes the variable with the highest P-value one at a time. The procedure terminated when it produced a model with only significant predictors (significance level of 0.1) and the highest adjusted R^2. (Chen *et al.*, 2019)

3.4.2 Ridge Regression

By adding regularization to linear regression, the regularized regressions are then formed. They are ridge regression, lasso regression and elastic regression. When the predictors are highly correlated, regularisation algorithms are used to estimate reliable predictor coefficients. The regularized regression models are able to be built in Rstudio through the glmnet package. Glmnet is a package that uses penalised maximum likelihood to fit generalised linear and similar models. The lasso or elastic net penalty regularisation path is computed at a grid of values (on the log scale) for the regularisation parameter lambda.

Ridge regression retains all predictors in the final model by imposing different penalties. It does not remove irrelevant features but minimizes their impact. (Godwin, 2021)

3.4.3 Lasso Regression

Similar to ridge regression but slightly different, LASSO (Least Absolute Shrinkage and Selection Operator) regression includes a penalty for non-zero coefficients; however, unlike ridge regression, which penalises the sum of squared coefficients (the so-called L2 penalty), lasso penalises the sum of their absolute values (L1 penalty). As a result, for large values of lambda, many coefficients are precisely zeroed under lasso, which is never the case with ridge regression. (Godwin, 2021)

3.4.4 Elastic Regression

By varying the values of the hyperparameter alpha α, Elastic Net is a hybrid of ridge regression and LASSO to minimize the loss function. When α =1, elastic net is the same as lasso; as it decreases towards 0, it approaches ridge regression. The lambda is able to be tuned through the glmnet package using cross-validation for a fixed alpha, but it does not support alpha-tuning, but the caret package is able to do the job.

3.4.5 Support Vector Machine (SVM)

Support Vector Machines (SVMs) are a well-known machine learning (ML) technique for classification and other learning activities. SVM is a discriminative classifier that is formalised by an optimal hyperplane. It generates an optimal hyperplane result, which classifies new examples, and datasets that support the hyperplane are referred to as support vectors. In the two-dimensional (2D) region, this hyperplane is a line that divides into two segments, each of which lies on either side. For example, multiple line data classification was completed with

two distinct datasets (squares and dots) and was ready to propose an affirmative interpretation. However, choosing the best hyperplane is a difficult task because it must be noise-free and accurate in its generalisation of data sets. In short, SVM attempts to find an optimised hyperplane that provides a significant minimum distance to the trained data set. (Battineni, Chintalapudi and Amenta, 2019)

When using the Radial Basis Function (RBF) kernel to train an SVM, two parameters must be considered: C and gamma. The parameter C, the regularisation parameter (default is 1), is shared by all SVM kernels and trades off misclassification of training examples against decision surface simplicity. A low C can make the decision surface smooth with more regularization, but it may lead to model underfit with high bias. Gamma is the amount of influence that a single training example has. The greater the gamma, the closer (to the hyperplane) the other examples must be to be affected.

3.5 Performance Validation Method

There are two types of machine learning modelling, which are classification and regression. The task of approximating a mapping function (f) from discrete input variables (X) to discrete output variables is known as classification predictive modelling (y). The output variables are frequently referred to as labels or categories. For a given observation, the mapping function predicts the class or category. On the other hand, the task of approximating a mapping function (f) from input variables (X) to a continuous output variable is known as regression predictive modelling (y). A real-value, such as an integer or floating-point value, is a continuous output variable. These are frequently numbers, such as amounts and sizes.

It is extremely useful in machine learning to have a single number to judge a model's performance, whether during training, cross-validation, or monitoring after deployment. One of the most commonly used measures for this is Root Mean Square Error (RMSE). It is a proper scoring rule that is simple to understand and compatible with many common statistical assumptions. Residuals are a measure of how far away the data points are from the regression line; RMSE is a measure of how spread out these residuals are. In other words, it indicates how concentrated the data is near the line of best fit. To validate experimental results, root mean square error is commonly used in climatology, forecasting, and regression analysis. (Stephanie, 2016)

RMSE will be used as the comparison measure to compare the accuracy of the models built.

3.6 Dataset Analysis & Preparation

3.6.1 Data Collection & Loading

The dataset is obtained from Kaggle (2019), it consists of cab_rides.csv and weather.csv. To proceed with the data analysis, the datasets are loaded into R studio. The structure of the dataset is printed to understand the raw data as shown in Figure 1: Data Structure of cab_rides & Figure 2: Data Structure of weather. There are 10 variables and 693071 observations in cab_rides.csv while weather.csv have 8 variables and 6276 observations.

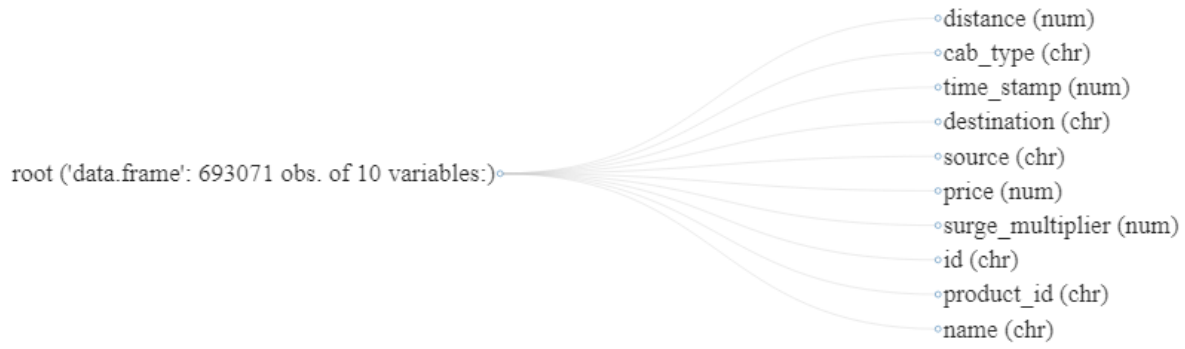


Figure 1: Data Structure of cab_rides

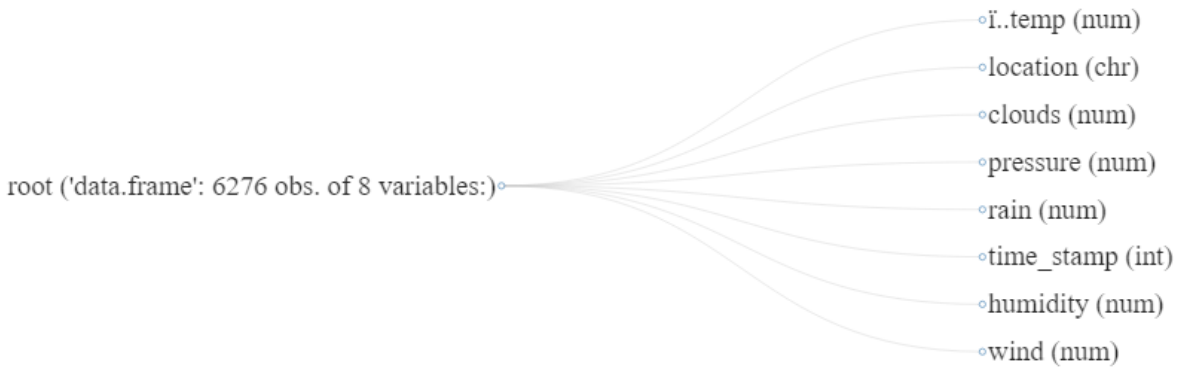


Figure 2: Data Structure of weather

3.6.2 Data Exploration

Before proceeding with any processing after loading the data into R studio, the data must be analysed and prepared to make it more suitable for the machine learning process. The data structure is examined during this stage, and any missing data is identified and treated. Furthermore, data must be explored, and data visualisations must be used to provide insights into the data, such as understanding data distribution and identifying important features and patterns.

```
> summary (rides)
    distance       cab_type          time_stamp        destination          source
 Min.   :0.020   Length:693071      Min.   :1.543e+12   Length:693071      Length:693071
 1st Qu.:1.280   Class :character   1st Qu.:1.543e+12   Class :character   Class :character
 Median :2.160   Mode  :character   Median :1.544e+12   Mode  :character   Mode  :character
 Mean   :2.189                      Mean   :1.544e+12
 3rd Qu.:2.920                      3rd Qu.:1.545e+12
 Max.   :7.860                      Max.   :1.545e+12

     price       surge_multiplier      id             product_id           name
 Min.   : 2.50   Min.   :1.000    Length:693071      Length:693071      Length:693071
 1st Qu.: 9.00   1st Qu.:1.000    Class :character   Class :character   Class :character
 Median :13.50   Median :1.000    Mode  :character   Mode  :character   Mode  :character
 Mean   :16.55   Mean   :1.014
 3rd Qu.:22.50   3rd Qu.:1.000
 Max.   :97.50   Max.   :3.000
 NA's   :55095
> str(rides) # structure
'data.frame':   693071 obs. of  10 variables:
 $ distance        : num  0.44 0.44 0.44 0.44 0.44 0.44 1.08 1.08 1.08 1.08 ...
 $ cab_type        : chr  "Lyft" "Lyft" "Lyft" "Lyft" ...
 $ time_stamp      : num  1.54e+12 1.54e+12 1.54e+12 1.54e+12 1.54e+12 ...
 $ destination     : chr  "North Station" "North Station" "North Station" "North Station" ...
 $ source          : chr  "Haymarket Square" "Haymarket Square" "Haymarket Square" "Haymarket Square" ...
 $ price           : num  5 11 7 26 9 16.5 10.5 16.5 3 27.5 ...
 $ surge_multiplier: num  1 1 1 1 1 1 1 1 1 1 ...
 $ id              : chr  "424553bb-7174-41ea-aeb4-fe06d4f4b9d7" "4bd23055-6827-41c6-b23b-3c491f24e74d" "981a
3613-77af-4620-a42a-0c0866077d1e" "c2d88af2-d278-4bfd-a8d0-29ca77cc5512" ...
 $ product_id      : chr  "lyft_line" "lyft_premier" "lyft" "lyft_luxsuv" ...
 $ name            : chr  "Shared" "Lux" "Lyft" "Lux Black XL" ...
>
```

Figure 3: Structure and summary of the cab_rides dataset

Basic Statistics

```
#Basic Statistic
config <- configure_report(
  global_ggtheme = quote(theme_minimal(base_size = 14))
)

create_report(rides, output_file = "statistics by cab type", y="cab_type", config = config)
```

Figure 3 is the code used to generate the simple EDA with data explorer library.

3.6.3 Missing Data

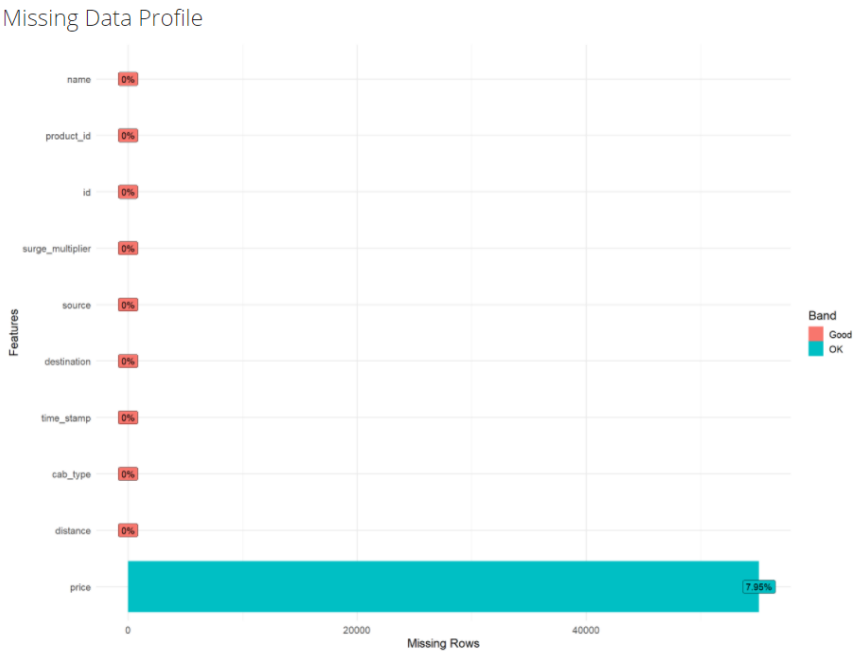


Figure 4: Missing Data Profile

As shown in the missing data profile in Figure 4, it is found that the 'price' column has 7.95% missing data (labelled as NAs) . To deal with the missing data, mean data imputation has been done to fill in the missing data. A total of 55095 missing data points is filled up using mean data imputation.

```
# To convert empty spaces to missing values
rides2 <- rides #keep rides as original, only do imputation to the copy
View (rides2)
rides2 <- mutate_all(rides2,na_if,"")
plot_missing(rides2)
colSums(sapply(rides2,is.na))

# Imputing missing values in continuous variables using mean
rides2$price = ifelse(is.na(rides2$price),
                      ave(rides2$price, FUN = function(x) mean(x, na.rm = TRUE)),
                      rides2$price)
colSums(sapply(rides2,is.na))
```

Figure 5: Code snippet for data imputation using mean

```
> colSums(sapply(rides2,is.na))
       distance         cab_type       time_stamp      destination           source
              0                0                0                0                0
          price surge_multiplier               id       product_id             name
          55095                0                0                0                0
> # Imputing missing values in continuous variables using mean
> rides2$price = ifelse(is.na(rides2$price),
+                       ave(rides2$price, FUN = function(x) mean(x, na.rm = TRUE)),
+                       rides2$price)
> colSums(sapply(rides2,is.na))
       distance         cab_type       time_stamp      destination           source
              0                0                0                0                0
          price surge_multiplier               id       product_id             name
              0                0                0                0                0
```

Figure 6: Before and After mean data imputation

```
# Visualizing missing values using VIM package
vim_plot <- aggr(rides, numbers=TRUE, prop = c(TRUE, FALSE))
```

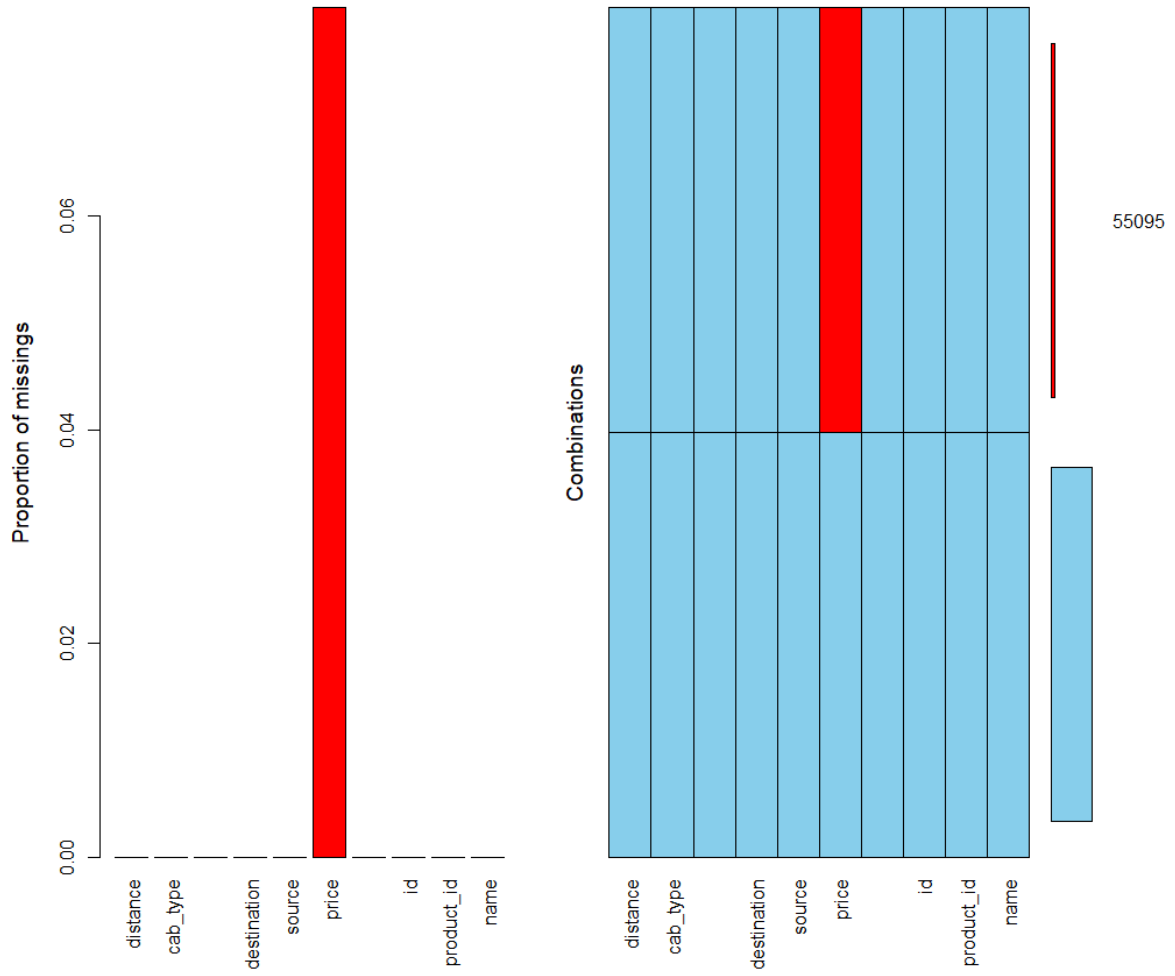


Figure 7: Visualizing the missing patterns

From the missing data profile, it is found that almost

3.6.4 Feature Selection

Since not all data within a data set is relevant to a problem, only the relevant ones are chosen. The dataset contained the terms 'id' and 'product id,' which were removed because they were irrelevant to the learning. This is because these id and product_id are only the unique identifier for the rides and also the cab types. There is already ‘name’ variable to describe the cab type. However, the ‘name’ variable are character variable, hence a new column called ‘nameno’ is added as label encoding of the name variable. The name variable is first factorized before converted to numeric data for the new variable.

As the problem in this study is the prediction of price, therefore it will be the target or dependent variable while the rest in the dataset is considered as input or independent variables.

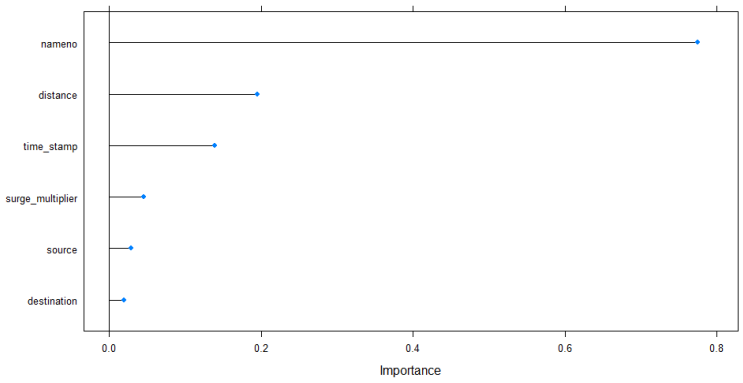
```
#column id & product_id, which is unique, irrelevant to the learning, therefore is removed from the table
cab_rides <- cab_rides[,c(-8,-9)]
#since name is the type of ride which will also affect the price, a new column is added by doing label encoding to the name
factors<-factor(rides$name)
factors
rides$nameno<-as.numeric(factors)
#since target variable is the price, it is put as first column
rides <- rides[, c(6, 1, 7, 9, 2, 3, 4, 5, 8)]
```

Figure 8: Code snippet for dropping the id & product_id column and adding the new variable

After that, to identify the importance of features affecting the price, a feature ranking by importance is created by using the caret package. The varImp is used to estimate the variable importance, which is then printed and plotted. It shows that nameno, distance and time stamp are the top 3 most important attributes in the dataset and the locations (source & destination) are the least important.

```
### Feature Selection ###
# ensure results are repeatable
set.seed(7)
# load the library
library(mlbench)
library(caret)
# load the dataset
data(rides)
# prepare training scheme
control <- trainControl(method="repeatedcv", number=10, repeats=3)
# train the model
model <- train(price~ distance+surge_multiplier+nameno+time_stamp+destination+source, data=rides2, method="glmStepAIC", preProcess="scale", trControl=control)
# estimate variable importance
importance <- varImp(model, scale=FALSE)
# summarize importance
print(importance)
# plot importance
plot(importance)
```

Figure 9: Code snippet for feature importance identification



loess r-squared variable importance

	Overall
nameno	0.77523
distance	0.19457
time_stamp	0.13882
surge_multiplier	0.04593
source	0.02900
destination	0.01905

Figure 10: Feature importance ranking

Chapter 4: Model Implementation & Validation

In this chapter, the model implementation will be demonstrated. First, the model implementation experiment will conduct with the linear regression, followed by the regularized regression (ridge, lasso & elastic), and ending with the SVM model. Parameter tuning are also done to find the best parameters for the models.

The experiment plan and dataset creation details are conducted as below:

No	Experiment	Train-Test Split Ratio	Model
1	Linear Regression	Train test split (80:20)	Linear Regression
2	Ridge Regression	Train test split (80:20)	Ridge Regression
3	Lasso Regression	Train test split (80:20)	Lasso Regression
4	Elastic Regression	Train test split (80:20)	Elastic Regression
5	SVM 1	Train test split (80:20)	SVM RBF kernel
6	SVM 2	Train test split (70:30)	SVM RBF kernel
7	SVM 3	Train test split (60:40)	SVM RBF kernel

Table 1: Experiment Setup Plan

4.1 Dataset Splitting

As the dataset contained 693071 observations, it is a heavy load for Rstudio to process the data and learning. Therefore, the dataset is sample split to reduce the observations in the dataset for model learning.

```
#due to dataset too big, split function is used to reduce the dataset
library(caTools)
set.seed(123)
split11 <- sample.split(cab_rides, SplitRatio = 0.2)
cab_ride_split = subset(cab_rides, split11 == TRUE)
rides = subset(cab_ride_split, split11 == TRUE)
```

Figure 11: Code snippet of reducing dataset by using sample split

```
> dim(cab_rides)
[1] 693071      9
> dim(cab_ride_split)
[1] 77008     9
> dim(rides)
[1] 8557    9
```

Figure 12: Output for splitting the dataset

Before all the learning process, the data are further split into two different sets, which are the training set (used to train the model) and the test set (used to test the effectiveness of the previously trained model).

As shown in **Error! Reference source not found.**, the data was split in a 80:20 ratio for training and test sets respectively. Further experiments are done by using 70:30 and 60:40 split.

```
# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(rides2, SplitRatio = 0.8)
training_set = subset(rides2, split == TRUE)
test_set = subset(rides2, split == FALSE)
```

Figure 13: Code snippet for splitting the dataset into training and test sets

4.1 Linear Regression

```
# Fitting Multiple Linear Regression to the Training set
regressor1 = lm(formula = price ~ ., data = training_set)
summary(regressor1)

#install.packages("jtools")
library(jtools)
#library(Rcpp)
summ(regressor1, confint = TRUE)

# Predicting the Test set results
y_pred = predict(regressor1, newdata = test_set)
y_pred
table(y_pred, test_set$price) # Comparing the predicted and actual value
cbind(y_pred, test_set$price)
```

Figure 14: Code Snippet of Linear Regression Model

```
> regressor1 = lm(formula = price ~ ., data = training_set)
> summary(regressor1)

Call:
lm(formula = price ~ ., data = training_set)

Residuals:
     Min       1Q   Median       3Q      Max
-18.2384  -1.4731  -0.1967   1.2051  28.1273

Coefficients: (3 not defined because of singularities)
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                       2.935e+01  6.917e+01   0.424 0.671303
distance                          2.633e+00  3.527e-02  74.651  < 2e-16 ***
surge_multiplier                  1.924e+01  3.943e-01  48.809  < 2e-16 ***
nameno                           -8.557e-01  1.292e-02 -66.247  < 2e-16 ***
cab_typeUber                      8.255e+00  1.399e-01  59.010  < 2e-16 ***
time_stamp                       -2.689e-11  4.479e-11  -0.600 0.548343
destinationBeacon Hill           -1.098e-01  1.495e-01  -0.734 0.462695
destinationBoston University      5.923e-02  2.042e-01   0.290 0.771789
destinationFenway                -3.006e-01  2.059e-01  -1.460 0.144413
destinationFinancial District     4.698e-01  1.539e-01   3.053 0.002276 **
destinationHaymarket Square       4.391e-01  2.071e-01   2.120 0.034035 *
destinationNorth End             -1.081e-03  2.012e-01  -0.005 0.995713
destinationNorth Station          2.089e-01  1.501e-01   1.392 0.164055
destinationNortheastern University -3.923e-02  2.026e-01  -0.194 0.846433
destinationSouth Station         -3.024e-02  2.044e-01  -0.148 0.882375
destinationTheatre District       2.186e-01  1.481e-01   1.475 0.140139
destinationWest End               1.101e-01  1.514e-01   0.728 0.466911
sourceBeacon Hill                -5.106e-01  1.469e-01  -3.476 0.000512 ***
sourceBoston University          -3.930e-01  1.585e-01  -2.480 0.013163 *
sourceFenway                     -1.561e-01  1.554e-01  -1.005 0.315098
sourceFinancial District         -6.247e-02  1.526e-01  -0.409 0.682387
sourceHaymarket Square            8.092e-02  1.516e-01   0.534 0.593544
sourceNorth End                   4.267e-01  1.499e-01   2.847 0.004425 **
sourceNorth Station              -1.912e-01  1.512e-01  -1.264 0.206199
sourceNortheastern University    -4.943e-01  1.515e-01  -3.263 0.001107 **
sourceSouth Station                      NA         NA      NA       NA
sourceTheatre District            3.972e-01  1.512e-01   2.626 0.008647 **
sourceWest End                   -1.451e-01  1.503e-01  -0.965 0.334427
nameBlack SUV                     1.087e+01  1.494e-01  72.721  < 2e-16 ***
nameLux                           6.746e+00  1.738e-01  38.828  < 2e-16 ***
nameLux Black                     1.282e+01  1.699e-01  75.418  < 2e-16 ***
nameLux Black XL                  2.299e+01  1.678e-01 137.063  < 2e-16 ***
nameLyft                          1.105e+00  1.640e-01   6.742 1.69e-11 ***
nameLyft XL                       7.885e+00  1.616e-01  48.805  < 2e-16 ***
nameShared                               NA         NA      NA       NA
nameTaxi                          2.960e+00  1.378e-01  21.490  < 2e-16 ***
nameUberPool                     -3.759e+00  1.383e-01 -27.184  < 2e-16 ***
nameUberX                        -1.747e+00  1.424e-01 -12.273  < 2e-16 ***
nameUberXL                        5.075e+00  1.462e-01  34.723  < 2e-16 ***
nameWAV                                  NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.538 on 6811 degrees of freedom
Multiple R-squared:  0.9174,    Adjusted R-squared:  0.917
F-statistic:  2102 on 36 and 6811 DF,  p-value: < 2.2e-16
```

```
> summ(regressor1, confint = TRUE)
MODEL INFO:
Observations: 6848
Dependent Variable: price
Type: OLS linear regression

MODEL FIT:
F(36,6811) = 2101.80, p = 0.00
R² = 0.92
Adj. R² = 0.92
```

Figure 15: Output of Linear Regression Model

The standard linear regression model obtained an accuracy of $R^2 = 0.92$

A multiple linear regression is used to determine whether one continuous dependent variable can be predicted from a set of independent (or predictor) variables. Alternatively, how much variance in a continuous dependent variable is explained by a set of predictors. Certain regression selection approaches are useful in testing predictors, increasing analysis efficiency.

Backward elimination is the reverse process of forward selection. All of the independent variables are entered into the equation first, and if they do not contribute to the regression equation, they are deleted one at a time.

Since the feature importance was done in chapter 3.6.4, it is found that the top 3 independent variables are nameno, distance and time_stamp. Therefore, they are chosen as the variables for the backward elimination to build an optimal model.

```
> regressor2 = lm(formula = price ~ nameno + distance + surge_multiplier,
+                 data = training_set)
> summary(regressor2)

Call:
lm(formula = price ~ nameno + distance + surge_multiplier, data = training_set)

Residuals:
    Min      1Q  Median      3Q     Max
-22.172  -4.593  -0.660   4.444  40.813

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      0.18523    1.03049    0.18    0.857
nameno          -1.27719    0.02077  -61.50   <2e-16 ***
distance         2.62137    0.06944   37.75   <2e-16 ***
surge_multiplier 19.35977    0.98755   19.60   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.486 on 6844 degrees of freedom
Multiple R-squared:  0.4579,    Adjusted R-squared:  0.4576
F-statistic:  1927 on 3 and 6844 DF,  p-value: < 2.2e-16
```

```
> summ(regressor2, confint = TRUE, ci.width=.5)
MODEL INFO:
Observations: 6848
Dependent Variable: price
Type: OLS linear regression

MODEL FIT:
F(3,6844) = 1926.80, p = 0.00
R² = 0.46
Adj. R² = 0.46
```

Figure 16: Backward elimination (nameno+distance+time_stamp)

```
> regressor3 = lm(formula = price ~ nameno + distance,
+                 data = training_set)
> summary(regressor3)

Call:
lm(formula = price ~ nameno + distance, data = training_set)

Residuals:
    Min      1Q  Median      3Q     Max
-18.579  -4.719  -0.748   4.430  54.969

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 19.90509    0.22989   86.58   <2e-16 ***
nameno      -1.30750    0.02128  -61.43   <2e-16 ***
distance     2.65679    0.07133   37.25   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.665 on 6845 degrees of freedom
Multiple R-squared:  0.4274,    Adjusted R-squared:  0.4273
F-statistic:  2555 on 2 and 6845 DF,  p-value: < 2.2e-16
```

```
> summ(regressor3, confint = TRUE, ci.width=.5)
MODEL INFO:
Observations: 6848
Dependent Variable: price
Type: OLS linear regression

MODEL FIT:
F(2,6845) = 2554.97, p = 0.00
R² = 0.43
Adj. R² = 0.43
```

Figure 17: Backward elimination (nameno+distance)

```
> summary(regressor4)

Call:
lm(formula = price ~ nameno, data = rides2)

Residuals:
    Min      1Q  Median      3Q     Max
-12.813  -5.185  -0.359   3.525  60.897

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 25.55660    0.16685  153.18   <2e-16 ***
nameno      -1.29067    0.02078  -62.12   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 7.295 on 8554 degrees of freedom
Multiple R-squared:  0.3108,    Adjusted R-squared:  0.3108
F-statistic:  3858 on 1 and 8554 DF,  p-value: < 2.2e-16
```

```
> summ(regressor4, confint = TRUE, ci.width=.5)
MODEL INFO:
Observations: 8556
Dependent Variable: price
Type: OLS linear regression

MODEL FIT:
F(1,8554) = 3858.27, p = 0.00
R² = 0.31
Adj. R² = 0.31
```

Figure 18: Backward elimination (nameno)

Linear Regression	R^2	RMSE	MAE
All variables	0.92	2.455	1.762
nameno+distance+time_stamp	0.46	6.468	5.169
nameno+distance	0.43	6.628	5.227
nameno	0.31	7.239	5.654

Table 2: Summary for Linear Regression

Through the backward selection method, it can be observed that once the input variables are reduced, the accuracy of the model dropped tremendously till only 0.31 when only the “nameno” variable is chosen. The last experiment yield an accuracy of a third of the original, supporting the indication of the importance of the “nameno” variable in the price prediction model.

4.2 Ridge Regression

The implementation of ridge regression method will be using glmnet package. Ridge regression is a linear regression extension in which the loss function is modified to reduce model complexity. This is accomplished by including a penalty parameter equal to the square of the magnitude of the coefficients.

```
l_ridge <- glmnet(x, y, family="gaussian", alpha=0)
plot(l_ridge, xvar = 'lambda', label=T)
```

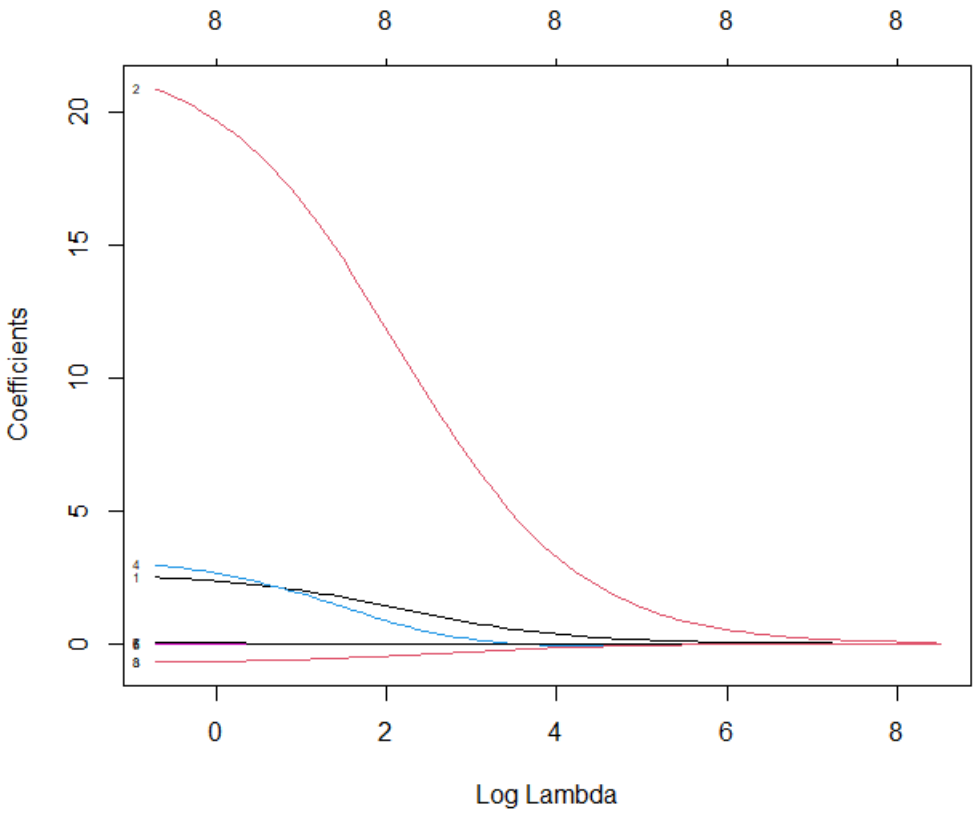
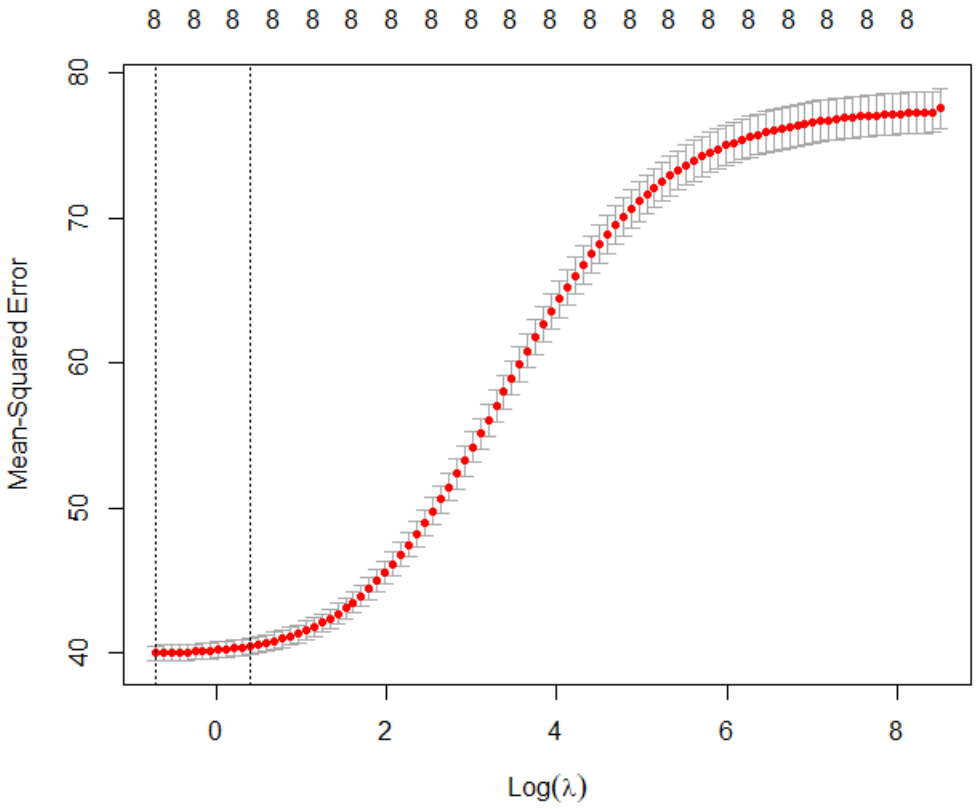


Figure 19: Code snippet and the output for the glmnet() function

The glmnet uses 100 lamda values by default for the model. One of the major differences between linear and regularized regression models is that the latter involves tuning a hyperparameter, lambda. The optimal lambda value is found by using the cv.glmnet() function.

```
> cv_out_ridge = cv.glmnet(x, y, alpha =0)
> plot (cv_out_ridge)
> names(cv_out_ridge) # outputs created by cv_out_ridge
 [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"       "nzero"      "call"
 [8] "name"       "glmnet.fit" "lambda.min" "lambda.1se" "index"
```




```
> lambda_min <- cv_out_ridge$lambda.min
> lambda_min
[1] 0.4914391
```

The plot for cv.glmnet() function is also done and the optimal lambda value comes out to be 0.491 and will be used to build the ridge regression model. After that, the predict function is used to generate predictions on the training set and test set. Finally, the eval_results() function is then used to calculate and print the evaluation results

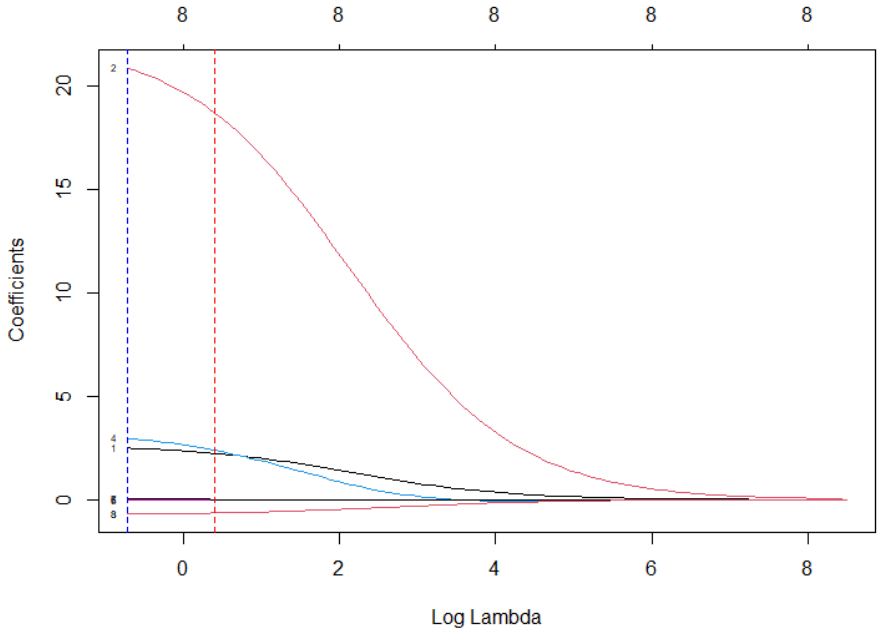


Figure 20: The ridge regression output after plotted with the optimal lambda value and the lambda for error within 1 standard deviation

```r
# Compute R^2 from true and predicted values
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE/nrow(df))


  # Model performance metrics
  data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  )

}

# Prediction and evaluation on train data
predictions_train <- predict(l_ridge, s = lambda_min, newx = x)
eval_results(y, predictions_train, training_set)

# Prediction and evaluation on test data
predictions_test <- predict(l_ridge, s = lambda_min, newx = x_test)
eval_results(y_test, predictions_test, test_set)
```

Figure 21: Evaluation metrics of ridge regression

```
> # Prediction and evaluation on train data
> predictions_train <- predict(l_ridge, s = lambda_min, newx = x)
> eval_results(y, predictions_train, training_set)
      RMSE   Rsquare
1 6.310831 0.4865036
> # Prediction and evaluation on test data
> predictions_test <- predict(l_ridge, s = lambda_min, newx = x_test)
> eval_results(y_test, predictions_test, test_set)
      RMSE   Rsquare
1 6.324961 0.4722199
```

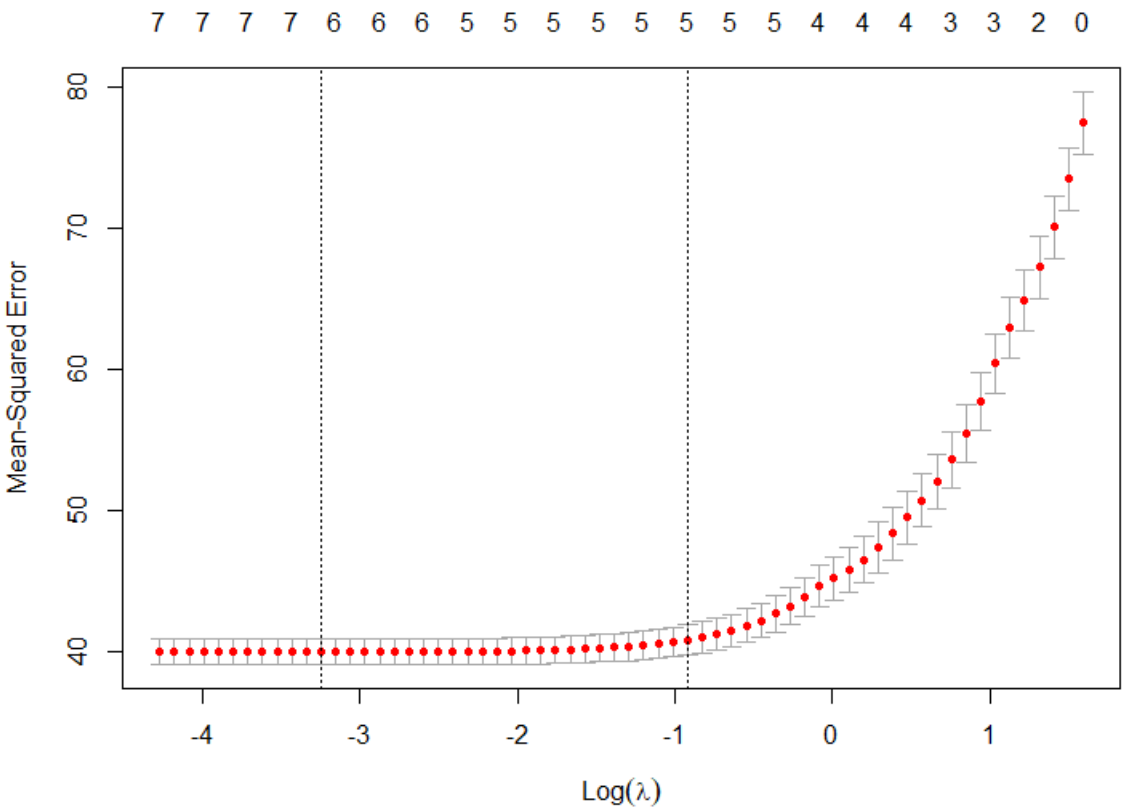
Figure 22: Output of the evaluation metrics

The above output shows that the RMSE and R-squared values for the ridge regression model on the training data are 6.31 million and 48.6 percent, respectively. For the test data, the results for these metrics are 6.32 million and 47.2 percent, respectively. There is a drop in the performance compared with linear regression model.

4.3 Lasso Regression

The implementation of lasso regression method will also be using glmnet package. The loss function in lasso is modified to reduce model complexity by limiting the sum of the absolute values of the model coefficients.

To build the lasso model, similar to ridge regression model, the optimal lambda value is identified by using the cv.glmnet() function.




```
> # two lambda values may be noted. 'lambda.min', 'lambda.1se'- lambda for error within 1 standard deviation
> lambda_min <- cv_out_lasso$lambda.min
> lambda_min
[1] 0.03894571
> lambda_1se<- cv_out_lasso$lambda.1se
> lambda_1se
[1] 0.3986214
```

For lasso regression, the value of alpha is 1. The output of the optimal lambda turns out to be 0.0389. After getting the lambda values, they are plotted together with the lasso model.

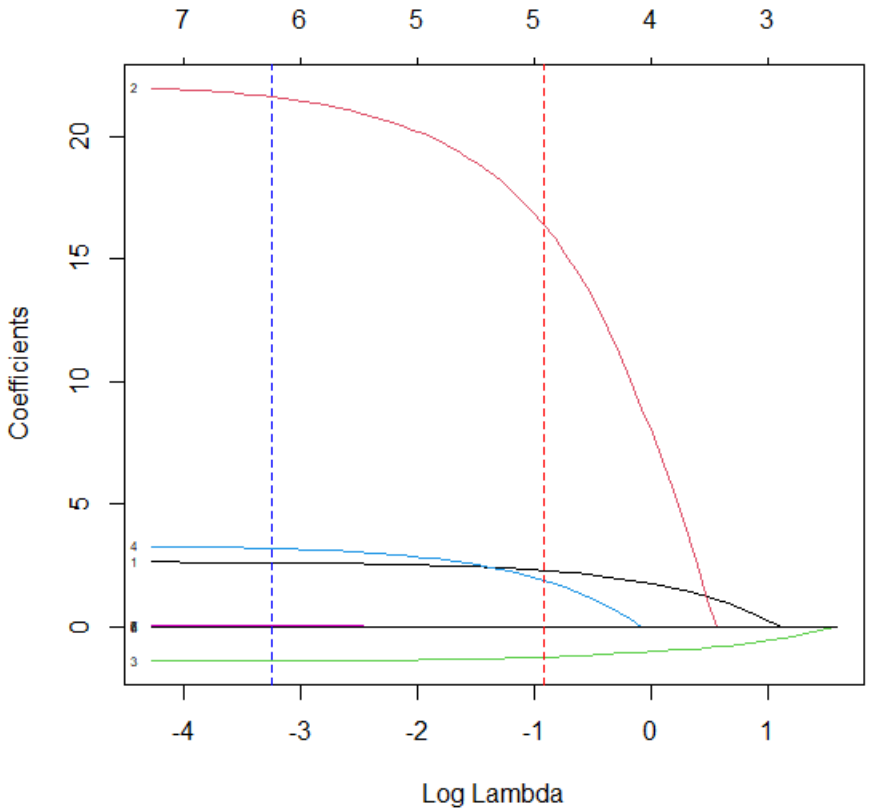


Figure 23: The lasso regression output after plotted with the optimal lambda value and the lambda for error within 1 standard deviation

```
> #~~~~~ Lasso Regression Evaluation ~~~~~#
> lasso_model <- glmnet(x, y, alpha = 1, lambda = lambda_min, standardize = TRUE)
> # Prediction and evaluation on train data
> predictions_train <- predict(l_lasso, s = lambda_min, newx = x)
> eval_results(y, predictions_train, training_set)
      RMSE   Rsquare
1 6.305444 0.4873798
> # Prediction and evaluation on test data
> predictions_test <- predict(l_lasso, s = lambda_min, newx = x_test)
> eval_results(y_test, predictions_test, test_set)
      RMSE   Rsquare
1 6.327408 0.4718115
```

The above output shows that the RMSE and R-squared values on the training data are 6.31 million and 48.7 percent, respectively. The results on the test data are 6.33 million and 47.2 percent, respectively.

4.4 Net Elastic Regression

```
##### Net Elastic Regression #####
# Set training control
train_cont <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 5,
                           search = "random",
                           verboseIter = TRUE)

# Train the model
elastic_reg <- train(price ~ .,
                     data = training_set,
                     method = "glmnet",
                     preProcess = c("center", "scale"),
                     tuneLength = 10,
                     trControl = train_cont)

elastic_reg <- train(price ~ nameno+distance+time_stamp+surge_multiplier,
                     data = training_set,
                     method = "glmnet",
                     preProcess = c("center", "scale"),
                     tuneLength = 10,
                     trControl = train_cont)

# Best tuning parameter
elastic_reg$bestTune

# Make predictions on training set
predictions_train <- predict(elastic_reg, x)
eval_results(y, predictions_train, training_set)

# Make predictions on test set
predictions_test <- predict(elastic_reg, x_test)
eval_results(y_test, predictions_test, test_set)
```

Figure 24: Code snippet for Net Elastic Regression Model

Elastic net regression combines ridge and lasso regression properties. The caret package, which automatically selects the optimal value of the parameters alpha and lambda, makes it simple to build the model.

The training control object train_cont is first created to specify how the repeated cross validation will run. After that, the elastic regression model is built where possible values of alpha and lambda are tested to select the optimum value. 10 different combinations of alpha and lambda values are to be tested is specified using the argument tuneLength.

```
Aggregating results
Selecting tuning parameters
Fitting alpha = 0.898, lambda = 0.0407 on full training set
> # Best tuning parameter
> elastic_reg$bestTune
      alpha     lambda
9 0.8981337 0.04066913
> # Make predictions on training set
> predictions_train <- predict(elastic_reg, x)
> eval_results(y, predictions_train, training_set)
      RMSE   Rsquare
1 6.501182 0.4561642
> # Make predictions on test set
> predictions_test <- predict(elastic_reg, x_test)
> eval_results(y_test, predictions_test, test_set)
    RMSE   Rsquare
1 6.4018 0.4544343
```

Figure 25: Output for Net Elastic Regression

The optimum alpha and lambda values obtained are 0.898 and 0.0407 respectively. The values are then implemented to the model and evaluated. The RMSE and R-squared values for the elastic net regression model on the training data are 6.50 million and 45.6 percent respectively. On the test data, the results for these metrics are 6.40 million and 45.4 percent respectively.

The results for linear, ridge, lasso and elastic regression are then all tabulated as shown in Table 3. (Only the test set evaluations are tabulated for better comparison.)

Regression Models	R^2	RMSE	MAE
Linear Regression	0.92	2.455	1.762
Ridge Regression	0.472	6.324	-
Lasso Regression	0.472	6.327	-
Elastic Regression	0.454	6.402	-

Table 3: Comparison of Regression Models

4.5 Support Vector Machine

The implementation of SVM method will be using the library (e1071) through the function svm for regression of the “price”, the target variable, whereby all other variables were fed, considered as the input variables. SVM model are implemented on the dataset using the Radial Basis Function (RBF) kernel. Figure 14 depicts a sample summary of the svm RBF model, displaying the trained model's parameters as well as the number of support vectors, which are the data points closest to the hyperplane.

The predict function is then applied to the test data in order for the model to predict the "price" data. It should be noted that in this process, the target variable is not supplied to the prediction. The accuracy of the model is then determined by comparing the predicted “price” values with the actual values in the test set. The accuracy is then obtained through the root mean squared error (RMSE) and the mean absolute error (MAE). The results outcome from the different experiments are then tabulated and shown in Table 4.

4.3.1 RBF kernel (80:20 split)

```
> split = sample.split(rides2$price, SplitRatio = 0.8)
> training_set = subset(rides2, split == TRUE)
> test_set = subset(rides2, split == FALSE)
> # ~~~~~~~~~~~~~~~~~~~~  Default SVM Model using the RBF kernel ~~~~~~~~~~~~~~~~~~~~
> svm_rbf <- svm(price~., data = training_set)
> summary(svm_rbf)

Call:
svm(formula = price ~ ., data = training_set)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.025
    epsilon:  0.1


Number of Support Vectors:  3259
```

```
> pred = predict (svm_rbf, test_set)
> RMSE(pred, test_set$price) # Root mean squared error
[1] 2.082686
> MAE(pred, test_set$price) # Mean Absolute Error
[1] 1.243721
```

Figure 26: Output of SVM model using RBF kernel (80:20 split)

4.3.2 RBF kernel (70:30 split)

```
> split = sample.split(rides2$price, SplitRatio = 0.7)
> training_set = subset(rides2, split == TRUE)
> test_set = subset(rides2, split == FALSE)
> # ~~~~~~~~~~~~~~~~~~~~  Default SVM Model using the RBF kernel ~~~~~~~~~~~~~~~~~~~~~~
> svm_rbf <- svm(price~., data = training_set)
> summary(svm_rbf)

Call:
svm(formula = price ~ ., data = training_set)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.025
    epsilon:  0.1


Number of Support Vectors:  2865




> pred = predict (svm_rbf, test_set)
> RMSE(pred, test_set$price) # Root mean squared error
[1] 2.151315
> MAE(pred, test_set$price) # Mean Absolute Error
[1] 1.24983
```

Figure 27: Output of SVM model using RBF kernel (70:30 split)

4.3.3 RBF kernel (60:40 split)

```
> split = sample.split(rides2$price, SplitRatio = 0.6)
> training_set = subset(rides2, split == TRUE)
> test_set = subset(rides2, split == FALSE)
> # ~~~~~~~~~~~~~~~~~~~~~~  Default SVM Model using the RBF kernel ~~~~~~~~~~~~~~~~~~~~~~
> svm_rbf <- svm(price~., data = training_set)
> summary(svm_rbf)

Call:
svm(formula = price ~ ., data = training_set)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.025
    epsilon:  0.1


Number of Support Vectors:  2465




> pred = predict (svm_rbf, test_set)
> RMSE(pred, test_set$price) # Root mean squared error
[1] 2.052003
> MAE(pred, test_set$price) # Mean Absolute Error
[1] 1.258139
```

Figure 28: Output of SVM model using RBF kernel (60:40 split)

4.3.4 Parameter C & Gamma tuning

```
> # ~~~~~~~~~~~~~~~~~~~~ Parameter tuning ~~~~~~~~~~~~~~~~~~~~
> obj <- tune(svm, price~., data = training_set,
+             ranges = list(gamma = 2^(-1:1), cost = 2^(2:4))
+             tunecontrol = tune.control(sampling = "fix")
+ )
> summary(obj)

Parameter tuning of 'svm':

- sampling method: fixed training/validation set

- best parameters:
 gamma cost
   0.5    4

- best performance: 10.94673

- Detailed performance results:
  gamma cost    error dispersion
1   0.5    4 10.94673         NA
2   1.0    4 17.26520         NA
3   2.0    4 26.15644         NA
4   0.5    8 11.01380         NA
5   1.0    8 17.62561         NA
6   2.0    8 26.39096         NA
7   0.5   16 11.50846         NA
8   1.0   16 18.11355         NA
9   2.0   16 26.71591         NA

> plot(obj)
```

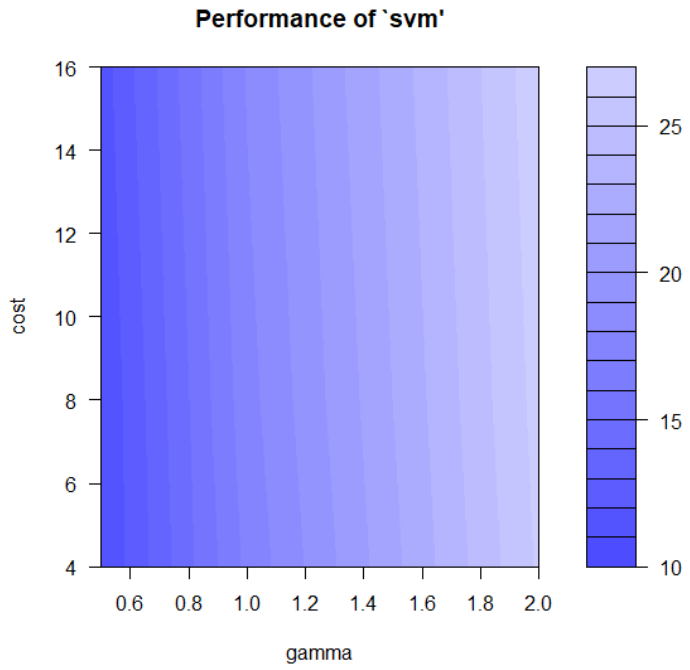


Figure 29: Output of parameter tuning for SVM model

By running the parameter tuning function, it is found that the best parameter for SVM model is gamma of 0.5 and C of 4. The values are then input to the SVM model and run again.

```
> svm_rbf2 <- svm(price~., data = training_set, cost=4, gamma=0.5)
> summary(svm_rbf2)

Call:
svm(formula = price ~ ., data = training_set, cost = 4, gamma = 0.5)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  4
      gamma:  0.5
    epsilon:  0.1


Number of Support Vectors:  3598
```

```
> pred2 = predict (svm_rbf2, test_set)
> RMSE(pred2, test_set$price) # Root mean squared error
[1] 2.070306
> MAE(pred2, test_set$price) # Mean Absolute Error
[1] 1.37807
```

Figure 30: The gamma and C parameter tuning for SVM (80:20 split)

```
> pred2 = predict (svm_rbf2, test_set)
> RMSE(pred2, test_set$price) # Root mean squared error
[1] 2.568679
> MAE(pred2, test_set$price) # Mean Absolute Error
[1] 1.487324
```

Figure 31: The gamma and C parameter tuning for SVM (70:30 split)

```
> pred2 = predict (svm_rbf2, test_set)
> RMSE(pred2, test_set$price) # Root mean squared error
[1] 2.591084
> MAE(pred2, test_set$price) # Mean Absolute Error
[1] 1.51207
```

Figure 32: The gamma and C parameter tuning for SVM (60:40 split)

Chapter 5: Discussion

Regression Models	R^2	RMSE	MAE
Linear Regression	0.92	2.455	1.762
Ridge Regression	0.472	6.324	-
Lasso Regression	0.472	6.327	-
Elastic Regression	0.454	6.402	-

Model	C	Gamma	Train-Test (80:20)		Train-Test (70:30)		Train-Test (60:40)	
			RMSE	MAE	RMSE	MAE	RMSE	MAE
SVM RBF	1	1	2.0827	1.2437	2.1513	1.2498	2.0520	1.2581
After Tuning	4	0.5	2.0703	1.3781	2.5687	1.4873	2.5910	1.5121

Table 4: Summary of the SVM model

After applying the best C and gamma values to the SVM model, it is found that it only improves the accuracy for the train-test split of 80:20 but not for 70:30 and 60:40. This may be due to when the training data is reduced, the learning process is not enough to learn and apply the best to the test set. Therefore, 80:20 split is a good ratio for train-test sample split.

The linear regression achieved a good accuracy with $R^2 = 0.92$. However, as a comparison with the other models, RMSE is a better measure to compare between the models. As for the RMSE measure, linear regression achieved RMSE of 2.455. On the other hand, the SVM RBF model with the train-test split of 80:20 ratio achieved the lowest RMSE at 2.087 and managed to reach RMSE of 2.0703 after doing parameter tuning.

Surprisingly, the ridge and lasso regression did not perform well with only RMSE of 6.32 while the hybrid of them, the elastic regression fare worse with only RMSE of 6.402. This indicates the importance of all the independent variables for the price prediction because the regularized regressions either penalized seemingly unimportant features or make them zero. This method was supposed to improve the model fit but end up it reduces the accuracy of the model.

Model	Train Test Split	Data Standardization	Parameter Tuning	Accuracy
Linear Regression				
Kunal, 2021	Yes	Yes	No	R^2 = 93.40%
Gabriel, 2021	Yes	Yes	No	R^2 = 92.87%
Hamed, 2021	Yes	Yes	No	RME = 4.533
Linear Regression	Yes	No	No	RMSE = 2.455
Random Forest				
Ravimunde, 2021	Yes	Yes	No	ACC: 97.47%
Hamed, 2021	Yes	Yes	No	RME = 3.638
Decision Tree				
Hamed, 2021	Yes	Yes	No	RME = 5.268
Regularized Regression				
Ridge Regression	Yes	No	Yes	RMSE = 6.325
Lasso Regression	Yes	No	Yes	RMSE = 6.327
Elastic Regression	Yes	No	Yes	RMSE = 6.402
Support Vector Machine (SVM)				
SVM RBF kernel	Yes	No	No	RMSE = 2.0827
SVM RBF kernel	Yes	No	Yes	RMSE = 2.0703

Table 5: Model Comparison Summary

As comparison with the other models in literature review, the linear regression in this study performed reasonably good with R^2 = 0.92 and RMSE of 2.455. But the regularized regressions did not performed so well with RMSE more than 6 whereby others were in the range of between 3 to 5. SVM model using the RBF kernel seems to be a good model where it achieved the lowest RMSE among all the models used for price prediction.

Chapter 6: Conclusion

6.1 Achievements

Objective	Completed?	Comments
Perform data merging between weather dataset and rides fare dataset	No	The epoch timestamp format for the weather and rides dataset do not match each other, therefore, it is not easy to combine them together as they are not exact match even after converting to a clearer time date format.
Perform feature selection function to find out the most important factor affecting the prices	Yes	It is found that the type of cab affects the price of the rides the most.
Perform parameter tuning for machine learning model used	Yes	The optimal alpha and lambda values is found and implemented in the ridge, lasso and elastic regression. The gamma and C values for SVM is also identified and compared to original SVM model.

6.2 Future Works & Recommendation

- Filter out respective cab types before doing price prediction for deeper analysis
- Relating the weather conditions during the time period of the rides and find the relationship between the weather and the price of the rides
- Running the models on a larger training set

6.3 Personal Reflection

Reflecting on the outcome of the report, the researcher is not too satisfied with the work done in the study as the aim and objectives of this work was fully achieved. The epoch time stamp format are not the same between the two datasets, making it tough to combine the datasets.

However, in overview, a better understanding about the machine learning workflow and application through R programming was obtained. Besides the machine learning algorithms and R programming knowledge obtained, this study also provided an opportunity to work on a real-life application in the ride sharing domain.

6.4 Acknowledgements

The researcher would like to express gratitude for the opportunity to work on this assignment which has helped him to familiarize with the workflow in machine learning. Moreover, online repositories like Kaggle, StackExchange, StackOverflow were also great help during the study.

References

Battineni, G., Chintalapudi, N. and Amenta, F. (2019) 'Machine learning in medicine: Performance calculation of dementia prediction by support vector machines (SVM)', *Informatics in Medicine Unlocked*, 16(June), p. 100200. doi: 10.1016/j.imu.2019.100200.

Chen, J. *et al.* (2019) 'A comparison of linear regression, regularization, and machine learning algorithms to develop Europe-wide spatial models of fine particles and nitrogen dioxide', *Environment International*, 130(February). doi: 10.1016/j.envint.2019.104934.

Godwin, A. (2021) *Ridge, LASSO, and ElasticNet Regression | by James Andrew Godwin | Towards Data Science, Towards data science*. Available at: https://towardsdatascience.com/ridge-lasso-and-elasticnet-regression-b1f9c00ea3a3 (Accessed: 6 September 2021).

Guo, S. *et al.* (2017) 'It can be cheaper: Using price prediction to obtain better prices from dynamic pricing in ride-on-demand services', *ACM International Conference Proceeding Series*, pp. 146–155. doi: 10.1145/3144457.3144476.

Kaggle (2019) *Uber & Lyft Cab prices | Kaggle*. Available at: https://www.kaggle.com/ravi72munde/uber-lyft-cab-prices (Accessed: 6 September 2021).

Kunal, A. *et al.* (2021) 'Prediction of Dynamic Price of Ride-On-Demand Services Using Linear Regression', *www.ijcait.com International Journal of Computer Applications & Information Technology*, 13(1), pp. 0–15. Available at: https://www.researchgate.net/publication/349484018.

Maulud, D. and Abdulazeez, A. M. (2020) 'A Review on Linear Regression Comprehensive in Machine Learning', *Journal of Applied Science and Technology Trends*, 1(4), pp. 140–

147. doi: 10.38094/jastt1457.
Stephanie (2016) *RMSE: Root Mean Square Error - Statistics How To*, *Statistics How To*. Available at: https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/ (Accessed: 6 September 2021).