Pre-Owned Bike Price Prediction Using Machine Learning

MSc Research Project
Data Analytics

Keerthana Sathyanarayanan
Student ID: x21195234

School of Computing
National College of Ireland

Supervisor: Dr. Anh Duong Trinh

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Keerthana Sathyanarayanan
Student ID:	x21195234
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Anh Duong Trinh
Submission Due Date:	14/08/2023
Project Title:	Pre-Owned Bike Price Prediction Using Machine Learning
Word Count:	1130
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	*Keerthana Sathyanarayanan*
Date:	17th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	✓
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	✓
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Pre-Owned Bike Price Prediction Using Machine Learning

Keerthana Sathyanarayanan
x21195234

1 Introduction

This configuration manual can be used to achieve same objectives as the work conducted generating equivalent results. It includes hardware and software specifications, dataset source, model implementation code and model evaluation code appended.

2 Hardware and Software Specifications:

- CPU: Processor used for the research is 11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz 2.50 GHz.
- RAM: RAM used for the study is 16GB.
- Storage: 477 GB
- GPU: NVIDIA GeForce MX450
- Operating System: 64-bit operating system, x64-based processor. Windows 11 is used.
- Environment: R Studio.
- The programming language used is R programming.
- The package dependencies are tidyverse, corrplot, ggplot2, lubridate, gridExtra, caTools, GGally, randomForest, caret, ISLR, xgboost.

3 Dataset Used:

- The dataset used in this project is a bike price prediction dataset.
- It includes various features related to bikes, such as model name, model year, kms driven, owner location, mileage power, price.
- It comprises over 5063 records providing data from all across India.
- The source of the dataset is Kaggle. [1]

[1] https://www.kaggle.com/datasets/vinayjain449/bike-prediction-with-linear-regression

4 Research Question:

How effective are the machine learning models in accurately predicting the resale value of the used bikes?

5 Objective:

Using the five potential machine learning models, a comparative study to identify the best models for forecasting the price of used bikes is conducted. Models used include Linear Regression, Elastic Regression, Support Vector Regressor, Random Forest, and XG Boost.

6 Experiment Design:

6 shows the process flow of the experiment.

7 Implementation

- Step 1- Run the code from figure 2 to 6. The code in this section encompasses tasks to be performed before building the model. It includes, data loading, data exploration and preprocessing(checking missing values, handling categorical data by converting to numeric, handling outliers and computing correlation matrix), exploratory data analysis and data splitting.
- Step 2- Execute figure 7 to build linear regression model and obtain its MAPE and RMSE scores. The figure 8 shows the console output of linear regression model.
- Step 3- Implement figure 9 and 10 to build random forest model. The console output of random forest model is shown in the figure 11.
- Step 4- The SVR model can be built with the code from the figure 12 to 14. The figure 15 shows the console output with MAPE and RMSE scores.
- Step 5- Elastic Regression model is built with the following code shown in figure 17 and 18. The console output is shown in figure 16.
- Step 6- The XG Boost model execution can be seen in the figure 20. The MAPE and RMSE values are displayed in the console in the figure 19.

The seed value and the split ratio are set to different numbers to ensure stability of the model. The split ratios 80:20 and 70:30 are provided with three seed values 123, 321 and 1712. The no. of trials performed at each split is recorded and their corresponding MAPE and RMSE values are reported.

7.1 Linear Regression:

In Table 1 it shows the trails performed in each split in Linear Regression.

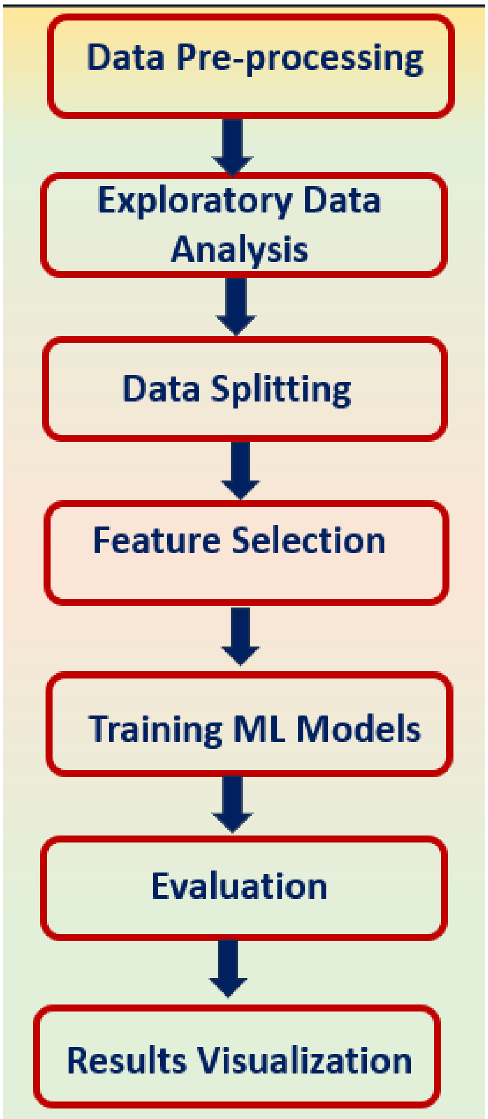

Figure 1: Work Flow Design

7.2 Random Forest:

In Table 2 it displays the trials run in each split of Random Forest.

7.3 Support Vector Regressor:

In Table 3 it displays the trials performed in each split of Support Vector Regressor.

7.4 Elastic Regression:

In Table 4 shows the trails run on each split in Elastic Regression.

7.5 XG Boost:

In Table 5 it displays the trials performed in each split of XG Boost.

```r
#Load required libraries
library(tidyverse)
library(corrplot)
library(ggplot2)
library(lubridate)
library(gridExtra)
library(caTools)
library(GGally)
library(randomForest)
library(caret)
library(ISLR)

#Read csv into dataframe
df=read.csv("Cleaned_bike_data.csv")

#Display first 10 rows of data frame
head(df,10)

#Display structure of data frame
str(df)

#Display summary of data frame
summary(df)
```

Figure 2: Model Implementation Code-1

```r

#Check for missing values
any(is.na(df))

#Count occurances of unique value
count(df)

#Categorical to numerical conversion
df$owner <- str_replace(df$owner,'first owner','1')
df$owner <- str_replace(df$owner,'second owner','2')
df$owner <- str_replace(df$owner,'third owner','3')
df$owner <- str_replace(df$owner,'fourth owner or more','4')
df$owner <- as.numeric(df$owner)

#Display first 10 rows of modified data frame
head(df,10)

#Display summary of model_year column
summary(df$model_year)

#Calculate quartiles and interquartile range for 'model_year'
q1 <- quantile(df$model_year, 0.25)
q3 <- quantile(df$model_year, 0.75)
iqr <- q3 - q1
```

Figure 3: Model Implementation Code-2

8 Evaluation

The Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE) are the two evaluation metrics used.

8.1 MAPE:

Table 6 displays the the MAPE value before and after hyperparameter tuning. The MAPE value of Random Forest is significantly decreased from 34 to 17.01. Next to Random Forest, XG Boost has the least MAPE score with 17.4. In figure21MAPE score is shown by a complete line graph of all models before and after hyperparameter tuning. The figure 23 illustrates complete performance comparison based on MAPE among all the models carried out. The choice of MAPE metric is supported with this reference Tayman and Swanson (1999).

```r

#Calculate lower and upper boundaries for outlier detection
lower_fence <- q1 - 1.5 * iqr
upper_fence <- q3 + 1.5 * iqr

# Select outliers from model_year column
outliers <- df[df$model_year < lower_fence | df$model_year > upper_fence, ]
# Removing outliers
df <- df[!(df$model_year %in% outliers$model_year), ]

#Display summary of model_year after outlier removal
summary(df$model_year)

#Display summary of price column
summary(df$price)

#Calculate quartiles and interquartile range for price column
q1 <- quantile(df$price, 0.25)
q3 <- quantile(df$price, 0.75)
iqr <- q3 - q1

#Calculate lower and upper boundaries for outlier detection
lower_fence <- q1 - 0.3 * iqr
upper_fence <- q3 + 1.5 * iqr
```

Figure 4: Model Implementation Code-3

```r
# Select outliers from price column
outliers1 <- df[df$price < lower_fence | df$price > upper_fence, ]
# Remove outliers from price column
df <- df[!(df$price %in% outliers1$price), ]

#Display summary of price after outlier removal
summary(df$price)

# Calculate correlation matrix for selected columns
cor1 <- cor(subset(df,select = c('model_year','kms_driven','owner','mileage','power','price')))

#Display summary of the matrix
summary(cor1)

#Adjust size of plots
options(repr.plot.width = 14, repr.plot.height = 8)

#Create correlation plot
corrplot(cor1, na.label = " ", method="color", tl.col = "black", tl.cex = 1)

#Create bar plot of owner column
ggplot(data = df, aes(x=reorder(owner, owner, function(x)-length(x)), fill = owner)) +
  geom_bar() + labs(x='Fuel') + labs(title = "Bar Graph of previous owners")

#Select columns for analysis
df1 <- subset(df,select = c('model_year','kms_driven','owner','mileage','power','price'))
```

Figure 5: Model Implementation Code-4

8.2 RMSE:

Chai and Draxler (2014) reported that RMSE is a better metric. Hence, RMSE is chosen as the evaluation metric for the study. Table 7 shows evaluation of models using RMSE metric. There is notable drop in RMSE number of Random Forest model after hyperparameter tuning. Before tuning it was 30392.1 and after tuning, it came down to 19405.75. But XG Boost has the least RMSE value with 18998.57. Figure22 displays the line graph of RMSE fluctuations before and after hyperparameter tuning. In illustration 24 all the five models are compared in aspect of performance having RMSE as the evaluation metric.

References

Chai, T. and Draxler, R. (2014). Root mean square error (rmse) or mean absolute error (mae)?– arguments against avoiding rmse in the literature, *Geoscientific Model Development* **7**: 1247–1250.

```
#Select columns for analysis
df1 <- subset(df,select = c('model_year','kms_driven','owner','mileage','power','price'))

#Set seed
set.seed(1712)

#Split data into train and test datasets
sample = sample.split(df1,SplitRatio = 0.7)
train_data =subset(df1,sample ==TRUE) # creates a training dataset named train1 with rows which are marked as TRUE
test_data=subset(df1, sample==FALSE)

#Display summary of data frame
summary(df1)
```

Figure 6: Model Implementation Code-5

```
df1 <- subset(df,select = c('model_year','kms_driven','owner','mileage','power','price'))

#Set seed
set.seed(1712)

#Split data into train and test datasets
sample = sample.split(df1,SplitRatio = 0.7)
train_data =subset(df1,sample ==TRUE) # creates a training dataset named train1 with rows which are marked as TRUE
test_data=subset(df1, sample==FALSE)

#Display summary of data frame
summary(df1)

#Build linear regression model
m1_lr <- lm(price ~ ., data = train_data)
summary(m1_lr)

#Make predictions using linear regression
pred=m1_lr$fitted.values
tally_table=data.frame(actual=train_data$price, predicted=pred)

#Calculate MAPE
mape=mean(abs(tally_table$actual-tally_table$predicted)/tally_table$actual)
mape

#Predict on test dataset and calculate errors
pred_er <- predict(m1_lr, test_data)
```

Figure 7: Model Implementation Code-6(LR)

Tayman, J. and Swanson, D. (1999). On the validity of mape as a measure of population forecast accuracy, *Population Research and Policy Review* **18**: 299–322.

```
> #Make predictions using linear regression
> pred=m1_lr$fitted.values
> tally_table=data.frame(actual=train_data$price, predicted=pred)
> #Calculate MAPE
> mape=mean(abs(tally_table$actual-tally_table$predicted)/tally_table$actual)
> mape
[1] 0.3364633
> #Predict on test dataset and calculate errors
> pred_er <- predict(m1_lr, test_data)
> error_er <- test_data$price - pred_er
> RMSE_er <- sqrt(mean(error_er^2))
> RMSE_er <- round(RMSE_er,2)
> RMSE_er
[1] 30392.1
```

Figure 8: Model Implementation Console Output-LR

```
pred_er <- predict(m1_lr, test_data)
error_er <- test_data$price - pred_er
RMSE_er <- sqrt(mean(error_er^2))
RMSE_er <- round(RMSE_er,2)
RMSE_er

#Build random forest model
m2_rf <- randomForest(
  price ~ .,
  data = train_data,
  ntree = 500,          # Number of trees in the forest
  mtry = 3,             # Number of variables randomly sampled as candidates at each split
  nodesize = 5,         # Minimum size of terminal nodes
  maxnodes = NULL,      # Maximum number of terminal nodes
  importance = TRUE,    # Calculate variable importance
  replace = TRUE        # Sample with replacement
)

#Plot feature importance for random forest
varImpPlot(m2_rf, main ='Feature Importance')

#Make predictions using random forest
predictions <- predict(m2_rf, newdata = test_data)
```

Figure 9: Model Implementation Code-7(RF)

```
predictions <- predict(m2_rf, newdata = test_data)

#Calculate MAPE for random forest
pred1= predict(m2_rf,newdata = test_data)
mape <- mean(abs((test_data$price - pred1) / test_data$price)) * 100
print(paste("MAPE:", round(mape, 2), "%"))
pred1= predict(m2_rf,newdata = test_data)
mape <- mean(abs((test_data$price - pred1) / test_data$price))
mape

#Predict testdata and calculate error
pred_er <- predict(m2_rf, test_data)
error_er <- test_data$price - pred_er
RMSE_er <- sqrt(mean(error_er^2))
RMSE_er <- round(RMSE_er,2)
RMSE_er
```

Figure 10: Model Implementation Code-8(RF)

```
> #Make predictions using random forest
> predictions <- predict(m2_rf, newdata = test_data)
> #Calculate MAPE for random forest
> pred1= predict(m2_rf,newdata = test_data)
> mape <- mean(abs((test_data$price - pred1) / test_data$price)) * 100
> print(paste("MAPE:", round(mape, 2), "%"))
[1] "MAPE: 17.01 %"
> pred1= predict(m2_rf,newdata = test_data)
> mape <- mean(abs((test_data$price - pred1) / test_data$price))
> mape
[1] 0.1701269
> #Predict testdata and calculate error
> pred_er <- predict(m2_rf, test_data)
> error_er <- test_data$price - pred_er
> RMSE_er <- sqrt(mean(error_er^2))
> RMSE_er <- round(RMSE_er,2)
> RMSE_er
[1] 19405.75
> 
```

Figure 11: Model Implementation Console Output-RF

```r
summary(df1)

#Extract input variables from training and test data
train_features <- train_data[, -which(names(train_data) == "price")]
test_features <- test_data[, -which(names(test_data) == "price")]

# Convert the target variable to a numeric vector
train_target <- train_data$price
test_target <- test_data$price

#Create grid of hyperparameters for SVR
hyperparameter_grid <- expand.grid(
  epsilon = c(0.1, 0.01, 0.001),
  cost = c(0.1, 1, 10),
  kernel = c("linear", "radial")
)

#Initialize variables to obtain best hyperparameters and corresponding metrics
best_rmse <- Inf
best_mape <- Inf
best_params <- NULL
```

Figure 12: Model Implementation Code-9(SVR)

```r
best_params <- NULL

#Loop through hyperparameter grid and train SVR
for (i in 1:nrow(hyperparameter_grid)) {
  params <- hyperparameter_grid[i, ]

  #Train SVR model with current hyperparameters
  svr_model <- svm(x = train_features, y = train_target,
                   kernel = params$kernel, cost = params$cost, epsilon = params$epsilon)

  #Make predictions on test set
  predictions <- predict(svr_model, newdata = test_features)

  #Calculate RMSE and MAPE
  rmse <- sqrt(mean((predictions - test_target)^2))
  mape <- mean(abs((predictions - test_target) / test_target)) * 100

  if (rmse < best_rmse) {
    best_rmse <- rmse
    best_mape <- mape
    best_params <- params
  }
}
```

Figure 13: Model Implementation Code-10(SVR)

```r
# Print best hyperparameters and corresponding metrics
cat("Best Hyperparameters:\n")
print(best_params)
cat("Best RMSE:", best_rmse, "\n")
cat("Best MAPE:", best_mape, "\n")

#Train an SVR model with the best hyperparameters on the full training data
svr_model <- svm(x = train_features, y = train_target)

#Make predictions on the test features using the trained SVR model
predictions <- predict(svr_model, newdata = test_features)

#Calculate RMSE
rmse <- sqrt(mean((predictions - test_target)^2))
print(paste("RMSE:", rmse))

#Calculate the MAPE
mape <- mean(abs((predictions - test_target) / test_target)) * 100
print(paste("MAPE:", mape))
```

Figure 14: Model Implementation Code-11(SVR)

```
Best Hyperparameters:
> print(best_params)
   epsilon cost kernel
16     0.1   10 radial
> cat("Best RMSE:", best_rmse, "\n")
Best RMSE: 23668.89
> cat("Best MAPE:", best_mape, "\n")
Best MAPE: 22.78684
> #Train an SVR model with the best hyperparameters on the full training data
> svr_model <- svm(x = train_features, y = train_target)
> #Make predictions on the test features using the trained SVR model
> predictions <- predict(svr_model, newdata = test_features)
> #Calculate RMSE
> rmse <- sqrt(mean((predictions - test_target)^2))
> print(paste("RMSE:", rmse))
[1] "RMSE: 23831.2244028002"
> #Calculate the MAPE
> mape <- mean(abs((predictions - test_target) / test_target)) * 100
> print(paste("MAPE:", mape))
[1] "MAPE: 22.8049390799974"
>
```

Figure 15: Model Implementation Console Output-SVR

```
Aggregating results
Selecting tuning parameters
Fitting alpha = 0.895, lambda = 0.13 on full training set
> #Make predictions using the trained elastic net regression model on the test d
ata
> pred_er <- predict(elastic_reg, test_data)
> #Calculate the prediction errors
> error_er <- test_data$price - pred_er
> #Calculate RMSE
> RMSE_er <- sqrt(mean(error_er^2))
> RMSE_er <- round(RMSE_er,2)
> #Display calculated RMSE
> RMSE_er
[1] 30379.65
> #Calculate MAPE
> mape <- mean(abs((test_data$price - pred_er) / test_data$price))
> mape
[1] 0.3376234
>
```

Figure 16: Model Implementation Console Output-ER

```
summary(df1)

#Create a trainControl object for cross-validation configuration
train_cont <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 5,
                           search = "random",
                           verboseIter = TRUE)


#Train elastic net regression model using the train() function
elastic_reg <- train(price ~.,
                     data = train_data,
                     method = "glmnet",
                     preProcess = c("center", "scale"),
                     tuneLength = 10,
                     trControl = train_cont)

#Make predictions using the trained elastic net regression model on the test data
pred_er <- predict(elastic_reg, test_data)

#Calculate the prediction errors
error_er <- test_data$price - pred_er

```

Figure 17: Model Implementation Code-12(ER)

```
error_er <- test_data$price - pred_er

#Calculate RMSE
RMSE_er <- sqrt(mean(error_er^2))
RMSE_er <- round(RMSE_er,2)

#Display calculated RMSE
RMSE_er

#Calculate MAPE
mape <- mean(abs((test_data$price - pred_er) / test_data$price))
mape
```

Figure 18: Model Implementation Code-13(ER)

Table 1: Data Split and No. of trials- Linear Regression

Split	Seed	Metrics	
		MAPE	RMSE
80:20	123	33.44	30159.58
	321	34.83	28553.16
	1712	33.64	30392.1
70:30	123	33.44	30159.58
	321	34.83	28553.16
	1712	33.64	30392.1

Table 2: Data Split and No. of trials- Random Forest

Split	Seed	Metrics	
		MAPE	RMSE
80:20	123	33.44	30159.58
	321	15.28	18053.13
	1712	17.01	19405.75
70:30	123	16.92	18910.54
	321	15.28	18053.13
	1712	17.01	19405.75

Table 3: Data Split and No. of trials- Support Vector Regressor

Split	Seed	Metrics	
		MAPE	RMSE
80:20	123	24.32	24148.12
	321	21.85	23705.36
	1712	22.78	23668.89
70:30	123	24.32	24148.12
	321	21.85	23705.36
	1712	22.78	23668.89

```
> #Make predictions with trained model on test data
> predictions <- predict(xgb_model, dtest)
> #Calculate RMSE between predicted and actual price
> rmse <- sqrt(mean((predictions - test_data$price)^2))
> print(paste("RMSE:", rmse))
[1] "RMSE: 18287.1009171189"
> #Calculate MAPE between predicted and actual price
> mape <- mean(abs((predictions - test_data$price) / test_data$price)) * 100
> print(paste("MAPE:", mape))
[1] "MAPE: 16.8051907193317"
> print(paste("MAPE:", round(mape, 2), "%"))
[1] "MAPE: 16.81 %"
```

Figure 19: Model Implementation Console Output-XGB

Table 4: Data Split and No. of trials- Elastic Regression

Split	Data	Metrics	
		MAPE	RMSE
80:20	123	35.21	30157.29
	321	34.38	28555.25
	1712	33.76	30379.65
70:30	123	35.21	30157.29
	321	34.38	28555.25
	1712	33.76	30379.65

Table 5: Data Split and No. of trials- XG Boost

Split	Seed	Metrics	
		MAPE	RMSE
80:20	123	17.83	18834.24
	321	16.81	18287.10
	1712	17.4	18998.57
70:30	123	17.83	18834.24
	321	16.81	18287.10
	1712	17.4	18998.57

```

# Convert the datasets to the DMatrix format required by XGBoost
dtrain <- xgb.DMatrix(data = as.matrix(train_data[, -which(names(train_data) == "price")]), label = train_data$price)
dtest <- xgb.DMatrix(data = as.matrix(test_data[, -which(names(test_data) == "price")]), label = test_data$price)

# Set the hyperparameters for XGBoost
params <- list(
  objective = "reg:squarederror",  # For regression tasks
  eta = 0.1,                       # Learning rate
  max_depth = 3,                   # Maximum depth of trees
  nrounds = 100                    # Number of boosting iterations
)

# Train the XGBoost model
xgb_model <- xgb.train(params, dtrain, nrounds = 100)

#Make predictions with trained model on test data
predictions <- predict(xgb_model, dtest)

#Calculate RMSE between predicted and actual price
rmse <- sqrt(mean((predictions - test_data$price)^2))
print(paste("RMSE:", rmse))

#Calculate MAPE between predicted and actual price
mape <- mean(abs((predictions - test_data$price) / test_data$price)) * 100
print(paste("MAPE:", mape))
print(paste("MAPE:", round(mape, 2), "%"))

```

Figure 20: Model Implementation Code-14(XGB)

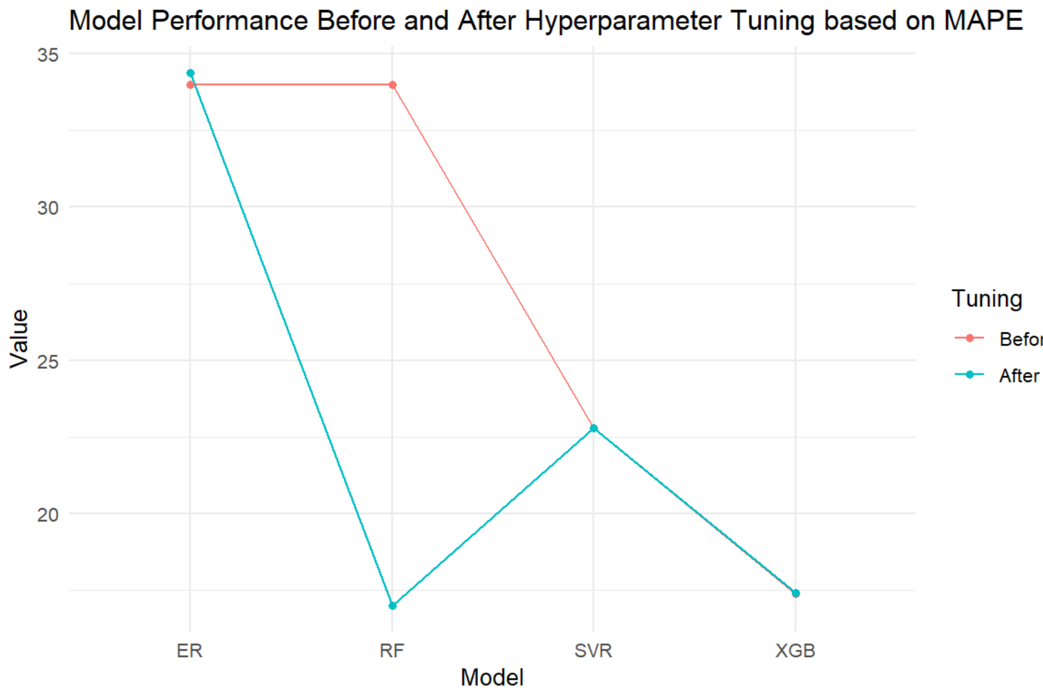

Figure 21: Hyperparameter Tuning: MAPE

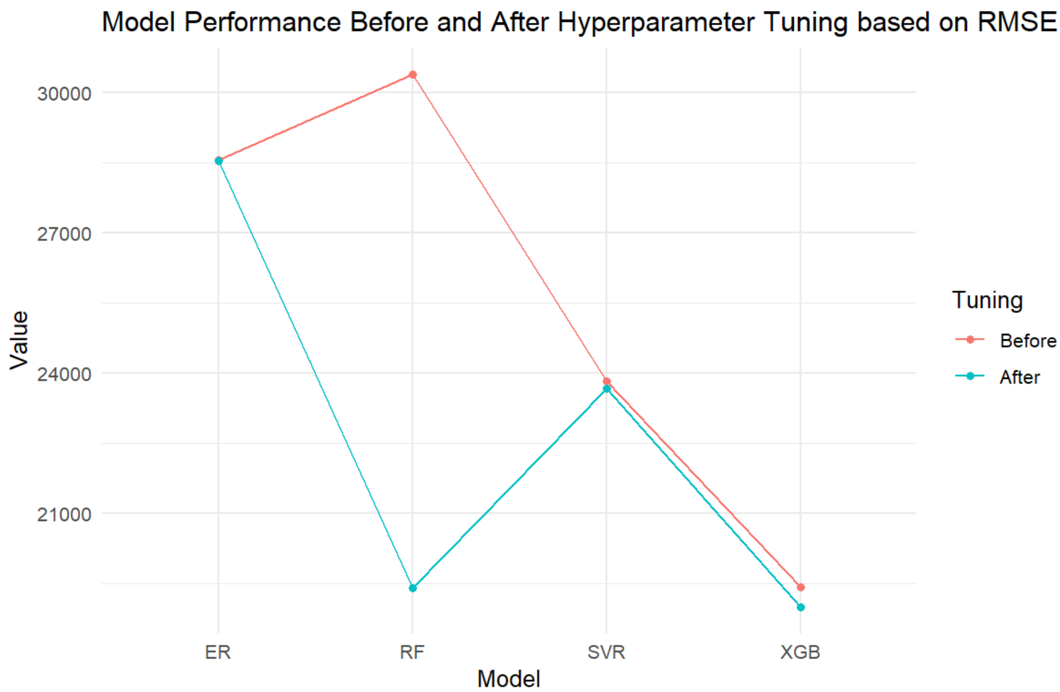

Figure 22: Hyperparameter Tuning: RMSE

Table 6: MAPE before and after hyperparameter tuning

Model	Before	After
Random Forest	34	17.01
Elastic Regression	34	34.37
Support Vector Regressor	22.80	22.78
XG Boost	17.37	17.4

Table 7: RMSE before and after hyperparameter tuning

Model	Before	After
Random Forest	30392.1	19405.75
Elastic Regression	28555.2	28552.68
Support Vector Regressor	23831	23668
XG Boost	19413	18998.57

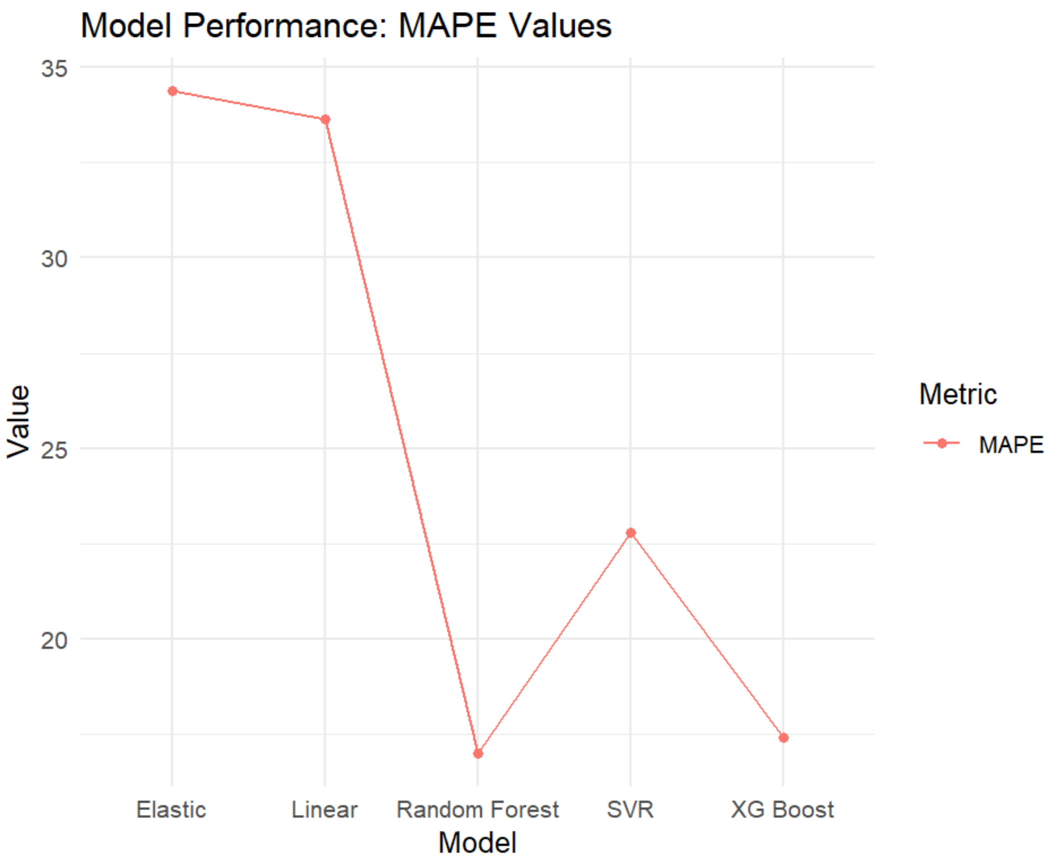

Figure 23: Model Performance Comparison based on MAPE

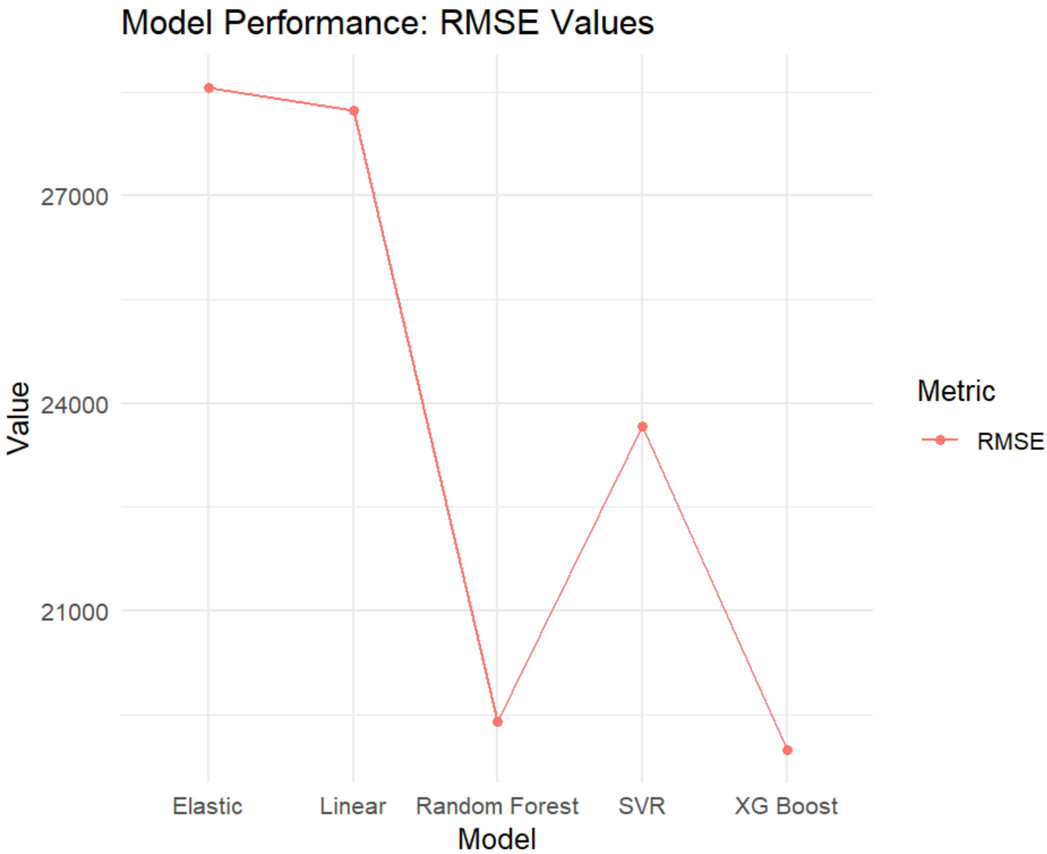

Figure 24: Model Performance comparison based on RMSE