DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

Chennai House Price Prediction

By
Semih
Oktay
Alptuğ

1) Project Description

This project aims to use machine learning techniques to predict house prices in the real estate market of Chennai, one of the important cities in India known for its dynamic real estate market. The goal of this project is to analyze price fluctuations in this market and adopt a data-driven approach to forecast future house prices.

1.1 Dataset

The project uses a comprehensive dataset to predict house prices in Chennai. This dataset encompasses various features of houses, serving as a significant resource to analyze factors that can influence house prices.

Location Information:

It includes the areas where houses are situated (AREA), representing the advantages and characteristics of these regions. For instance, areas like Karapakkam, Anna Nagar, Adyar, Velachery, and their respective traits.

House Features:

The dataset includes details about the features of houses, such as the interior square footage (INT_SQFT), the number of bedrooms (N_BEDROOM), bathrooms (N_BATHROOM), and the total number of rooms (N_ROOM).

Structure and Conditions:

Information about the types of structures (BUILDTYPE) and sales conditions (SALE_COND) are part of the dataset. It represents various types of structures like commercial, residential, or other types, along with sale conditions like abnormal, family, or normal sale.

Services and Environment:

Details regarding the available services (UTILITY_AVAIL) in houses and environmental factors (STREET) are included. For example, whether the house has services like electricity and water or the type of street it's located on.

Region and Quality Assessments:

Evaluation metrics such as the types of zones where houses are located (MZZONE) and scores indicating the quality of rooms, bathrooms, bedrooms, and an overall score (QS_ROOMS, QS_BATHROOM, QS_BEDROOM, QS_OVERALL) are part of the dataset.

Other Features:

Additional features such as the construction dates of houses (DATE_BUILD), selling prices (SALES_PRICE), registration fees (REG_FEE), commissions (COMMIS), among others, are also included in the dataset.

Here is the dataset:

	prt_id <chr>	area <chr>	int_sqft <int>	date_sale <chr>	dist_mainroad <int>	n_bedroom <int>	n_bathroom <int>	n_room <int>	sale_cond <chr>
1	p03210	karapakkam	1004	04-05-2011	131	1	1	3	abnormal
2	p09411	anna nagar	1986	19-12-2006	26	2	1	5	abnormal
3	p01812	adyar	909	04-02-2012	70	1	1	3	abnormal
4	p05346	velachery	1855	13-03-2010	14	3	2	5	family
5	p06210	karapakkam	1226	05-10-2009	84	1	1	3	abnormal
6	p00219	chrompet	1220	11-09-2014	36	2	1	4	partial
7	p09105	chrompet	1167	05-04-2007	137	1	1	3	partial
8	p09679	velachery	1847	13-03-2006	176	3	2	5	family
9	p03377	chrompet	771	06-04-2011	175	1	1	2	adjland
10	p09623	velachery	1635	22-06-2006	74	2	1	4	abnormal

1-10 of 10 rows | 1-10 of 22 columns

park_fadl	date_build	buildtype	utilify_avail	street	mzzone	qs_rooms	qs_bathroom	qs_bedroom	qs_overall
<crJ>	<tt,r>	<cnr·	<cnr>	<tr>	<tk,	<col>	<co>	<co)	<aol>
yes	I5-0\-1967	commercial	allpub	paved		40	J.9	u	4.J30
no	11-11-1995	commercial	allpub	gravel	rn	u	4.2	2.\	3765
yes	09-01-1992	commercial	elo	gravel	rl	4.1	J.8	2.2	3.090
no	I8-0J-1988	otners	nosewr	paved		n	J.9	3.6	4010
yes	I J-10-1919	otners	allpub	gravel		JO	1.5	4.1	J.290
no	12-09-2009	commercial	nosewa	noaccess	rn	ts	1.6	3.1	3.J20
no	12-04-1919	otner	allpub	noaccess	rl	I.6	1.1	2)	2610
no	I5-0J-1996	commercial	allpub	gravel	rm	1.4	4.5	2.1	3.260
no	14-04-1917	otners	nosewr	paved	rm	1.9	J]	4.0	J\50
no	16-06-1991	otners	elo	noaccess		3.1	J.I	J.3	J.160

1-10of10rows|11-10of22columns

utililpvail	wee!	mnone	uoom1	1_oatBroom	1_oearoom	1_overall	re Jee	comm11	1ale1_rrice
<tnr>	<tnr;•	((11>	<ao>	(QQ)	<ao,>	<QOI)	<int:>	<nt>	<int:>
allruo	ravea		to	J.	u	mo	1ioooo	I 1 00	lb00000
allruo	ravel	rn	0	0	ll	rn	1om1	JMM	m11m
elo	ravel	rl	ti	J.	u	1.0 0	rn	n11	IJI1 100
nom1r	rave□		V	Jj	H	torn	11m1	llM1	%1010
allruo	ravel		1.0	I.\	ti	mo	mooo	I OoJ	lOb110
noiewa	noatte11	rn	ti	lb	1.1	mo	0 011	I m	Im /\O
allruo	nome11		H	I.I	n	mo	mm	m11	m o
allruo	ravel	rm	1A	0	L.I	mo	oOio	m10	lb00110
noiewr	ravea	rm	i.	n	to	1.110	mm	ll1%	10 10
elo	noatte11		1.1	1.1	jj	J.IW	1111b	mm	o mo

1-10o(IOrowil1 -1o1ln columm

2) Preprocessing

2.1 Fill Null (N/A) Value

The preprocessing stage involves identifying null values in the dataset as the first step. If the column is numeric, the missing values are filled using the median of that column's data. For categorical data, the missing values are replaced with the mode of that column.

```r
cat("Number of missing values : ", sum(is.na(dataset)) , "\n")

# Determining columns with missing values
columns_with_na <- names(Filter(function(x) any(is.na(x)), dataset))

# Selecting numeric columns with missing values
numeric_columns_with_na <- columns_with_na[sapply(dataset[columns_with_na], is.numeric)]

cat("Columns with missing values : ", columns_with_na , "\n")

# Fill Null value with median(numeric value)
for (col in numeric_columns_with_na) {
  dataset[[col]][is.na(dataset[[col]])] <- median(dataset[[col]], na.rm = TRUE)
}

cat("Number of missing values after filling : ", sum(is.na(dataset)), "\n")
```

```
Number of missing values :  54
Columns with missing values :  n_bedroom n_bathroom qs_overall
Number of missing values after filling :  0
```

2.2 Rectification of Erroneous Entries in Data

In this section, we are rectifying incorrectly entered data in certain columns of our dataset with their accurate versions

"area" Column

```
# Convert all column names to lowercase
colnames(dataset) <- tolower(colnames(dataset))

# Convert all values in the 'area' column to lowercase
dataset$area <- tolower(dataset$area)

# Correcting spelling mistakes in the 'area' column
dataset$area <- str_replace_all(dataset$area, c('velchery' = 'velachery',
                                                'kknagar' = 'kk nagar',
                                                'tnagar' = 't nagar',
                                                'chormpet' = 'chrompet',
                                                'chrompt' = 'chrompet',
                                                'chrmpet' = 'chrompet',
                                                'ana nagar' = 'anna nagar',
                                                'ann nagar' = 'anna nagar',
                                                'karapakam' = 'karapakkam',
                                                'adyr' = 'adyar'))
```

"sale_cond" Column

```
dataset$sale_cond <- str_replace_all(dataset$sale_cond, c('adj land' = 'adjland',
                                                          'normal sale' = 'normal sale',
                                                          'partiall' = 'partial',
                                                          'ab normal' = 'abnormal'))
```

"park_facil" Column

```
dataset$park_facil <- str_replace_all(dataset$park_facil, c('noo' = 'no'))
```

"buildtype" Column

```
dataset$buildtype <- str_replace_all(dataset$buildtype, c('comercial' = 'commercial',
                                                          'others' = 'other'))
```

“utility_avail” Column

```
dataset$utility_avail <- str_replace_all(dataset$utility_avail, c('all pub' = 'allpub',
                                                                  'nosewr' = 'nosewa'))
```

“street” Column

```
dataset$street <- str_replace_all(dataset$street, c('pavd' = 'paved',
                                                    'noaccess' = 'no access'))
```

2.3 Change Type Columns

```
dataset$n_bedroom <- as.integer(dataset$n_bedroom)
dataset$n_bathroom <- as.integer(dataset$n_bathroom)
```

```
dataset$date_sale <- as.Date(dataset$date_sale, format = "%d-%m-%Y")
dataset$date_build <- as.Date(dataset$date_build, format = "%d-%m-%Y")
```

2.4 Create New Columns (Property Age , Total Price)

The 'Property Age' column represents the age of the house, calculated by subtracting the construction year of the house from the sale date.

```
dataset$property_age <- as.numeric(format(dataset$date_sale, "%Y")) - as.numeric(format(dataset$date_build, "%Y"))
```

The 'total_price' column was created by summing the realtor commission and title deed expenses with the house's sale price to form a new column.

```
dataset$total_price <- dataset$reg_fee + dataset$commis + dataset$sales_price
```

2.5 Delete Outlier

```
q1 <- quantile(dataset$sales_price, 0.25)
q3 <- quantile(dataset$sales_price, 0.75)
iqr <- q3 - q1
lower_bound <- q1 - 1.5 * iqr
upper_bound <- q3 + 1.5 * iqr

# Find outlier
outliers_indices <- which(dataset$sales_price < lower_bound | dataset$sales_price > upper_bound)

# Delete outlier row
cleaned_data <- dataset[-outliers_indices, ]
```

3) Examination of Data Column Distributions

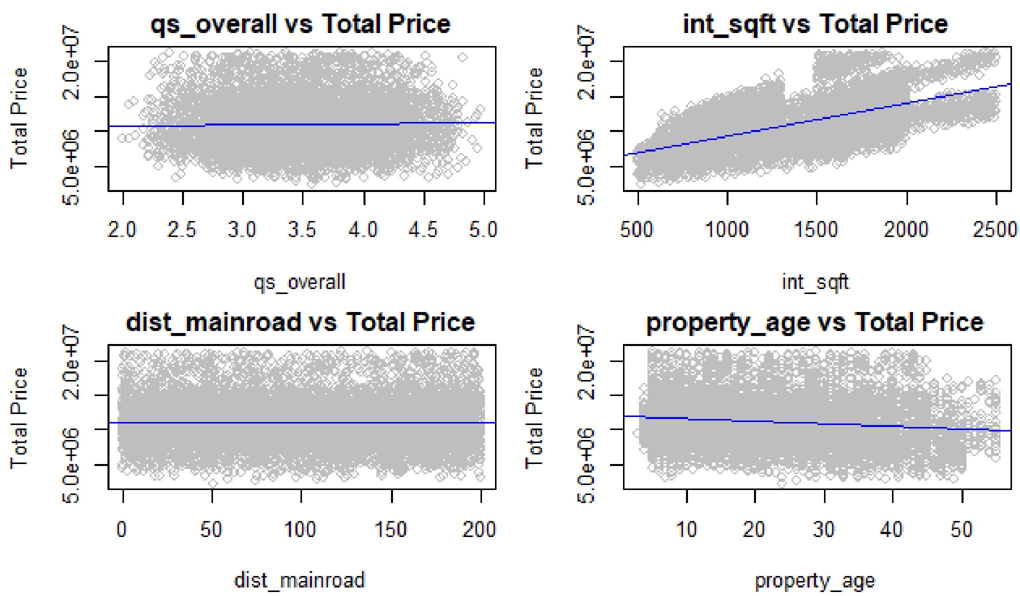
In this section, we delved into understanding the distribution of each column in our dataset. Histograms were generated for columns containing numerical data, depicting the frequency and distribution of values within those columns. For columns with categorical data, we visualized their distribution using bar graphs.

This analysis aided in comprehending the characteristics of each feature within the dataset. It allowed us to visually assess differences between features, understand the overall structure of the dataset, and evaluate the distribution of variables. This visual exploration significantly contributed to a deeper understanding of the dataset's analysis.
The distribution of the data appears to fit a normal distribution.

~~~
library(ggplot2)

for (col in names(dataset)) {
  if(is.numeric(dataset[[col]])) {
    graph <- ggplot(dataset, aes(x = !!sym(col))) +
      geom_histogram(fill = "lightblue", color = "grey", bins = 30) +
      labs(title = paste("Distribution of", col))
    print(graph)
  } else {
    graph <- ggplot(dataset, aes(x = !!sym(col))) +
      geom_bar(fill = "blue", color = "grey") +
      labs(title = paste("Distribution of", col))
    print(graph)
  }
}
```
~~~

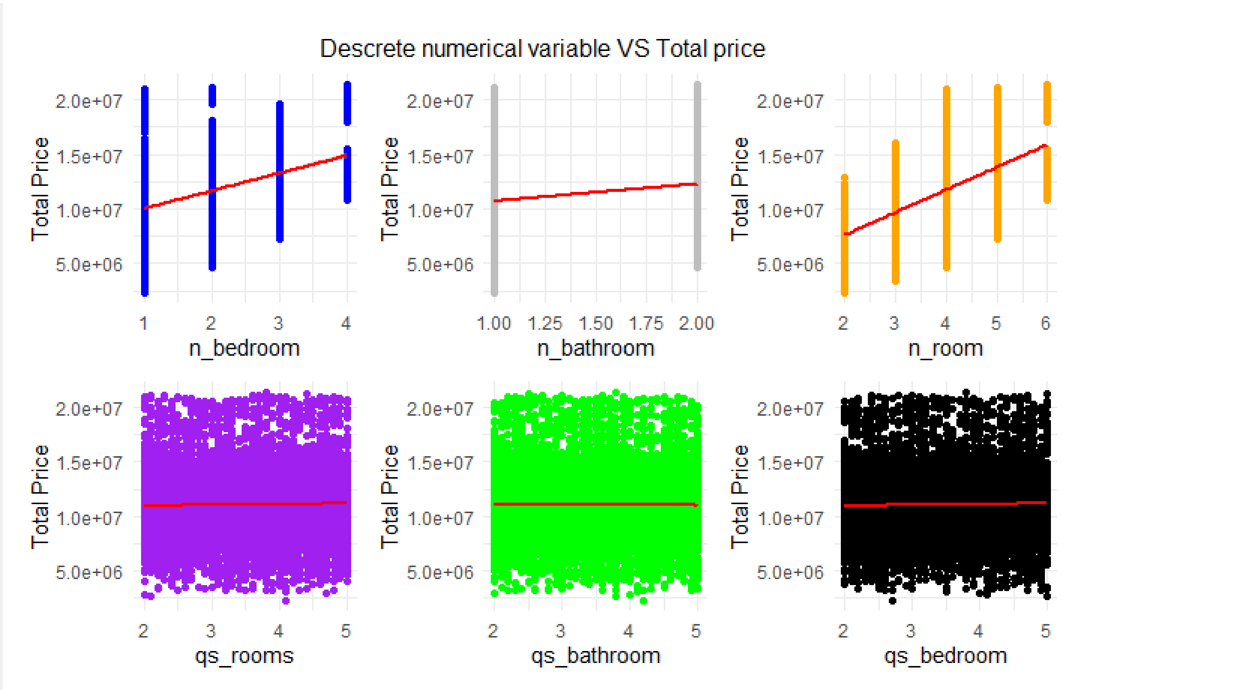
4) The Impact of Numerical Variables on House Prices

The distributions in each graph illustrate how the respective numerical variables impact house prices. For instance, it emphasizes that 'int_sqft' has a more pronounced effect on house prices, while 'property_age' demonstrates limited impact. Furthermore, 'qs_overall' and 'dist_mainroad' show no significant correlation with house prices. Columns lacking correlation were subsequently removed from the dataset.



Continous numerical variable VS Total price

Similar operations are conducted here, but this time, the impact of discrete and continuous columns on the total price is investigated. As seen in the graph, it is evident that the columns 'qs_rooms', 'qs_bathroom', and 'qs_bedroom' have no discernible impact on the target column. Therefore, we remove these columns from our dataset.



5) One-Hot Encoding

At this stage, we are transforming our categorical columns in the dataset into a numeric format by applying one-hot encoding, thereby creating new columns.

5.1 “buildtype” Column

```
filtered_data <- dataset[dataset$buildtype %in% c("commercial", "other", "house"), ]

# One-hot encoding
one_hot_encoded <- model.matrix(~ buildtype - 1, data = filtered_data)

#delete buildtype
dataset <- dataset[ , !(names(dataset) %in% "buildtype")]

dataset <- cbind(dataset, one_hot_encoded)
```

buildtype <chr>	buildtypecommercial <dbl>	buildtypehouse <dbl>	buildtypeother <dbl>
commercial	1	0	0
commercial	1	0	0
other	0	0	1
other	0	0	1
commercial	1	0	0
other	0	0	1
commercial	1	0	0
other	0	0	1
other	0	0	1
commercial	1	0	0

5.2 "area" Column

```
area_levels <- c("karapakkam", "anna nagar", "adyar", "velachery", "chrompet", "kk nagar", "t nagar")
one_hot_encoded_area <- matrix(0, nrow = nrow(dataset), ncol = length(area_levels))
colnames(one_hot_encoded_area) <- paste("area", area_levels, sep = "_")

for (i in 1:nrow(dataset)) {
  area_index <- match(dataset[i, "area"], area_levels)
  if (!is.na(area_index)) {
    one_hot_encoded_area[i, area_index] <- 1
  }
}

one_hot_encoded_area_df <- as.data.frame(one_hot_encoded_area)

#delete are column and add one_hot_encoded_area columns
dataset <- cbind(dataset[, !(names(dataset) %in% "area")], one_hot_encoded_area_df)
```

area <chr>
karapakkam
adyar
velachery
karapakkam
chrompet
chrompet
velachery
chrompet
velachery
chrompet

area_karapakkam <dbl>	area_anna nagar <dbl>	area_adyar <dbl>	area_velachery <dbl>
1	0	0	0
0	0	1	0
0	0	0	1
1	0	0	0
0	0	0	0
0	0	0	0

The transformation was applied to other columns as well; however, only the first four columns are displayed due to space limitations in the image.

5.3 “sale_cond” Column

```
dataset$sale_cond <- factor(dataset$sale_cond, levels = c('partial', 'family', 'abnormal', 'normal sale', 'adjland'))

#One hot encoding
one_hot_encoded_sale_cond <- model.matrix(~ sale_cond - 1, data = dataset)

# Delete sade_cond
dataset <- dataset[, !(names(dataset) %in% "sale_cond")]

# Add One-hot encoded
dataset <- cbind(dataset, one_hot_encoded_sale_cond)
```

sale_cond <chr>
abnormal
abnormal
family
abnormal
partial
partial
family
adjland
abnormal
adjland

sale_condpartial <dbl>	sale_condfamily <dbl>	sale_condabnormal <dbl>	sale_condnormal sale <dbl>	sale_condadjland <dbl>
0	0	1	0	0
0	0	1	0	0
0	1	0	0	0
0	0	1	0	0
1	0	0	0	0
1	0	0	0	0

5.4 “park_facil” Column

```
dataset$park_facil <- factor(dataset$park_facil, levels = c('yes', 'no'))

# 'yes' 1 'no' 2
dataset$park_facil <- ifelse(dataset$park_facil == 'yes', 1, 0)

dataset$park_facil <- as.integer(dataset$park_facil)
```

park_facil <int>
1
1
0
1
0
0

park_facil <chr>
yes
yes
no
yes
no
no

5.5 "utility_avail" Column

```r
utility_levels <- c('elo', 'nosewa', 'nosewr', 'allpub')
utility_factors <- factor(dataset$utility_avail, levels = utility_levels)

one_hot_encoded_utility <- model.matrix(~ utility_avail - 1, data = dataset)

one_hot_encoded_utility_df <- as.data.frame(one_hot_encoded_utility)

#add one hot encoded columns
dataset <- cbind(dataset, one_hot_encoded_utility_df)

# Delete 'utility_avail'
dataset <- dataset[, !(names(dataset) %in% "utility_avail")]
```

utility_avail <chr>	utility_availelo <dbl>	utility_availnosewa <dbl>	utility_availnosewa <dbl>
allpub	0	0	0
elo	1	0	0
nosewa	0	0	1
allpub	0	0	0
nosewa	0	1	0
allpub	0	0	0

5.6 "street" Column

```r
street_levels <- c('no access', 'paved', 'gravel')
street_factors <- factor(dataset$street, levels = street_levels)

one_hot_encoded_street <- model.matrix(~ street - 1, data = dataset)

one_hot_encoded_street_df <- as.data.frame(one_hot_encoded_street)

dataset <- cbind(dataset, one_hot_encoded_street_df)

# Delete 'street'
dataset <- dataset[, !(names(dataset) %in% "street")]
```

street <chr>	streetgravel <dbl>	streetno access <dbl>	streetpaved <dbl>
paved	0	0	1
gravel	1	0	0
paved	0	0	1
gravel	1	0	0
no access	0	1	0
no access	0	1	0

5.7 "mzzone" Column

```
mzzone_levels <- c('a', 'c', 'i', 'rl', 'rh', 'rm')
mzzone_factors <- factor(dataset$mzzone, levels = mzzone_levels)

one_hot_encoded_mzzone <- model.matrix(~ mzzone - 1, data = dataset)

one_hot_encoded_mzzone_df <- as.data.frame(one_hot_encoded_mzzone)

dataset <- cbind(dataset, one_hot_encoded_mzzone_df)

# Delete 'mzzone'
dataset <- dataset[, !(names(dataset) %in% "mzzone")]
```

mzzone <chr>	mzzonea <dbl>	mzzonec <dbl>	mzzonei <dbl>	mzzonerh <dbl>	mzzonerl <dbl>	mzzonerm <dbl>
a	1	0	0	0	0	0
rl	0	0	0	0	1	0
i	0	0	1	0	0	0
c	0	1	0	0	0	0
rh	0	0	0	1	0	0
rl	0	0	0	0	1	0

After applying one-hot encoding, the total number of columns has increased to 36.

6) Scaling

In our project, we applied 2 different scaling methods to our dataset. One of these methods was Min-Max normalization, and the other was logarithmic transformation. Just as optimizing the model using different parameters is important, ensuring that the dataset is optimized for our models is crucial as well. Therefore, we conducted experiments on our dataset for both models and analyzed the outcomes we obtained.

6.1) Min max Normalizayion

```
# Min-Max normalization
min_max_normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

normalized_dataset <- as.data.frame(lapply(dataset, min_max_normalize))

head(normalized_dataset)
dataset <- normalized_dataset
```

After Min-Max normalization, some of our models also exhibited better results compared to the log transformation. For instance, the first image shows the outcome of our Random Forest model with default parameters after the log transformation, while the second image displays the outcome of our Random Forest model with default parameters after Min-Max normalization.

After Log Transform

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9812256	0.008168379	0.09037909	0.06644879

After Min max normalization

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9842605	0.002128213	0.04613256	0.03564307

1 row

When we experimented by changing the parameters of our Random Forest model, it was observed that we obtained better results than Min-Max normalization.

Log Transform

Description: df [16 x 6]

ntree <dbl>	mtry <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
300	15	0.9824098	0.007590077	0.08712105	0.06410768
300	20	0.9826705	0.007421479	0.08614801	0.06333586
500	2	0.9338219	0.063525213	0.25204208	0.19366765
500	5	0.9733074	0.013178055	0.11479571	0.08548213
500	15	0.9825425	0.007535124	0.08680509	0.06413908
500	20	0.9824639	0.007530793	0.08678014	0.06365282

11-16 of 16 rows Previous 1 2 Next

Min-max Normalization

Description: df [16 x 6]

ntree <dbl>	mtry <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
300	15	0.9845815	0.002058215	0.04536756	0.03482078
300	20	0.9849977	0.001991349	0.04462454	0.03429554
500	2	0.9259547	0.020798879	0.14421817	0.11222391
500	5	0.9771390	0.003466189	0.05887435	0.04631760
500	15	0.9848186	0.002026061	0.04501179	0.03456800
500	20	0.9851012	0.001979353	0.04448992	0.03414428

11-16 of 16 rows Previous 1 2 Next

In our linear regression model, the dataset resulting from the log transformation provided better results.

Log Transform

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9809795	0.008015242	0.08952788	0.06250183

1 row

Min-max Transform

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9632262	0.004798767	0.06927313	0.05060293

1 row

In our SVM model, a slight difference was observed where the logarithmic transformation yielded slightly better results.

Log Transform

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9958596	0.001736094	0.04166647	0.03214065

Min-max Normalization

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9955887	0.000577444	0.02403007	0.01901622

1 row

Our KNN model also yielded better results with the logarithmic transformation.

Log Transform

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9242683	0.03575501	0.18909	0.1474365

Min-max Transform

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.8723139	0.0170232	0.130473	0.09624709

1 row

6.2) Log Transform

Some of our columns needed normalization compared to other columns; for instance, our columns like 'sale_price' and 'total_price' consisted of 7 to 8-digit numbers. Consequently, even with a close estimation, metrics like MSE resulted in significantly large values. Additionally, after the logarithmic transformation, our KNN model showed a substantial increase in performance, reaching success rates around 0.92. While the improvement in other models wasn't as pronounced as in KNN, their performance also enhanced.

```
dataset$sales_price <- log(dataset$sales_price)
dataset$total_price <- log(dataset$total_price)
dataset$int_sqft <- log(dataset$int_sqft)
dataset$property_age <- log(dataset$property_age)
```

7) Cross Validation

In our project, the cross-validation method utilized was k-fold cross-validation, while in some models, the Repeated k-fold Cross-Validation method was employed.

```
trControl = trainControl(method = "repeatedcv", number = k, repeats = k_fold),
```

```
trControl = trainControl(method = "cv", number = k_fold),
```

8-9) Model Training

In our project, we utilized four different machine learning models:

1 - Random Forest

2 - Support Vector Machine

3 - KNN

4 - Linear Regression

We compared the default settings of these four models with variations in certain parameters, presenting the results in a tabular format. During the model training, we partitioned the data into 80% for training and 20% for testing. Furthermore, we employed cross-validation methods including k-fold cross-validation and repeated k-fold during this partitioning process. Additionally, we conducted these procedures separately for two scaling methods: Log transform and Min-Max normalization.

8) Log Transform Result

8.1 Random Forest

8.1.1 Random Forest with Default Parameter

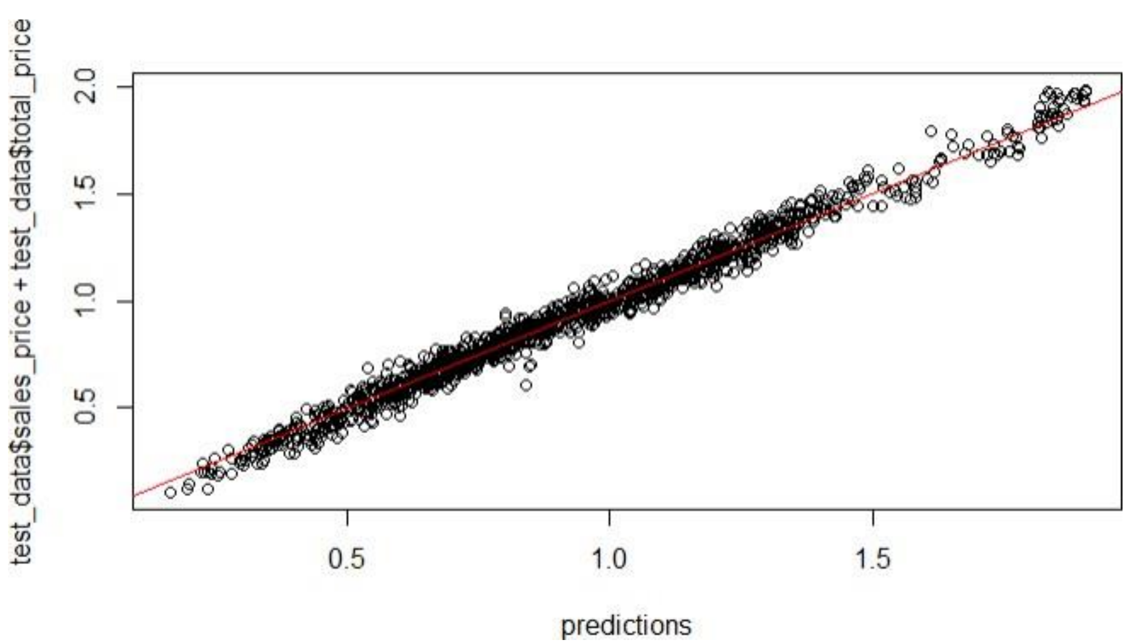
```
library(caret)
library(randomForest)
library(Metrics)


#Create Train and test set.
set.seed(42)
trainIndex <- createDataPartition(dataset$total_price, p = 0.8, list = FALSE)
train_data <- dataset[trainIndex, ]
test_data <- dataset[-trainIndex, ]
```

```r
#Create Random Forest model
rf_model <- randomForest(
  x = train_data[, -which(names(train_data) %in% target_columns)],
  y = train_data$sales_price + train_data$total_price
)

#Prediciton for random forest model
predictions <- predict(rf_model, newdata = test_data[, -which(names(test_data) %in% target_columns)])

# Calculate Error
errors <- predictions - (test_data$sales_price + test_data$total_price)

# Calculated Metrics
r_squared <- cor(predictions, test_data$sales_price + test_data$total_price)^2
mse <- mean(errors^2)
rmse <- sqrt(mse)
mae <- mean(abs(errors))

# Print Metrics
print(paste("R-kare Skoru:", r_squared))
print(paste("MSE:", mse))
print(paste("RMSE:", rmse))
print(paste("MAE:", mae))

plot(predictions, test_data$sales_price + test_data$total_price)
abline(0, 1, col = "red")

# Create Table
results <- data.frame(
  `R2` = r_squared,
  `MSE` = mse,
  `RMSE` = rmse,
  `MAE` = mae
)
```

Result

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9842605	0.002128213	0.04613256	0.03564307

Regression Graph



8.1.2 Random Forest based on ntree and mtry parameters

```
#Create Train and test set
set.seed(42)
trainIndex <- createDataPartition(dataset$total_price, p = 0.8, list = FALSE)
train_data <- dataset[trainIndex, ]
test_data <- dataset[-trainIndex, ]

# Parameter Values
ntree_values <- c(50, 100, 300, 500)
mtry_values <- c(2, 5, 15 , 20)

metrics <- data.frame()

for (ntree in ntree_values) {
  for (mtry in mtry_values) {
    # Create Model
    rf_model <- randomForest(
      x = train_data[, -which(names(train_data) %in% target_columns)],
      y = train_data$sales_price + train_data$total_price,
      ntree = ntree,
      mtry = mtry
    )

    predictions <- predict(rf_model, newdata = test_data[, -which(names(test_data) %in% target_columns)])

    r_squared <- cor(predictions, test_data$sales_price + test_data$total_price)^2

    mse <- mean((test_data$sales_price + test_data$total_price - predictions)^2)

    rmse <- sqrt(mse)

    mae <- mean(abs(test_data$sales_price + test_data$total_price - predictions))

    metrics <- rbind(metrics, data.frame(ntree = ntree, mtry = mtry, R_Squared = r_squared, MSE = mse, RMSE = rmse, MAE = mae))
  }
}
```

Result

Description: df [16 x 6]

ntree <dbl>	mtry <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
50	2	0.9211987	0.067983313	0.26073610	0.19996137
50	5	0.9725328	0.013231114	0.11502658	0.08640300
50	15	0.9811689	0.008085738	0.08992073	0.06579502
50	20	0.9818037	0.007748094	0.08802326	0.06501823
100	2	0.9250735	0.064723604	0.25440834	0.19526509
100	5	0.9721829	0.013786009	0.11741384	0.08762275
100	15	0.9821284	0.007678622	0.08762774	0.06425683
100	20	0.9823239	0.007556541	0.08692837	0.06404707
300	2	0.9292085	0.064238582	0.25345331	0.19538404
300	5	0.9732764	0.013081165	0.11437292	0.08523582

1-10 of 16 rows

Previous 1 2 Next

Description: df [16 x 6]

ntree <dbl>	mtry <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
300	15	0.9824098	0.007590077	0.08712105	0.06410768
300	20	0.9826705	0.007421479	0.08614801	0.06333586
500	2	0.9338219	0.063525213	0.25204208	0.19366765
500	5	0.9733074	0.013178055	0.11479571	0.08548213
500	15	0.9825425	0.007535124	0.08680509	0.06413908
500	20	0.9824639	0.007530793	0.08678014	0.06365282

11-16 of 16 rows Previous 1 2 Next

8.1.3 Random Forest based on ntree , mtry and k_fold parameters

Result

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
50	2	5	0.9211987	0.067983313	0.26073610	0.19996137
50	2	10	0.9218441	0.067295480	0.25941372	0.19848218
50	5	5	0.9691847	0.014568628	0.12070057	0.09010003
50	5	10	0.9718730	0.013588517	0.11656980	0.08756457
50	15	5	0.9821615	0.007712954	0.08782343	0.06519105
50	15	10	0.9816072	0.007895447	0.08885633	0.06523930
50	20	5	0.9810447	0.008090964	0.08994979	0.06599325
50	20	10	0.9819050	0.007710772	0.08781100	0.06431601
100	2	5	0.9280581	0.064258851	0.25349329	0.19469466
100	2	10	0.9258452	0.069007769	0.26269330	0.20323839

1-10 of 32 rows Previous 1 2 3 4 Next

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
100	5	5	0.9724107	0.013558800	0.11644226	0.08690117
100	5	10	0.9720796	0.013458221	0.11600957	0.08627295
100	15	5	0.9810658	0.008151597	0.09028620	0.06656414
100	15	10	0.9822436	0.007624655	0.08731927	0.06454228
100	20	5	0.9818595	0.007756532	0.08807118	0.06491858
100	20	10	0.9824796	0.007502193	0.08661520	0.06393937
300	2	5	0.9282952	0.065151793	0.25524849	0.19677570
300	2	10	0.9290833	0.065823745	0.25656139	0.19899090
300	5	5	0.9727465	0.013329879	0.11545510	0.08548728
300	5	10	0.9724436	0.013502597	0.11620068	0.08631529

11-20 of 32 rows Previous 1 2 3 4 Next

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
300	15	5	0.9825396	0.007527314	0.08676009	0.06406077
300	15	10	0.9826037	0.007491979	0.08655622	0.06373573
300	20	5	0.9826869	0.007419464	0.08613631	0.06346357
300	20	10	0.9826720	0.007419433	0.08613613	0.06350300
500	2	5	0.9301979	0.064851585	0.25465974	0.19595990
500	2	10	0.9288779	0.066196498	0.25728680	0.19770694
500	5	5	0.9732347	0.013066652	0.11430946	0.08553121
500	5	10	0.9738344	0.012932465	0.11372099	0.08498769
500	15	5	0.9822828	0.007630594	0.08735327	0.06408144
500	15	10	0.9825920	0.007514127	0.08668406	0.06376360

21-30 of 32 rows Previous 1 2 3 4 Next

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
500	20	5	0.9826313	0.007436806	0.08623692	0.06331519
500	20	10	0.9824180	0.007518826	0.08671117	0.06369990

31-32 of 32 rows Previous 1 2 3 4 Next

8.1.4 Random Forest based on ntree , mtry ,k_fold , maxnodes , maxdepth parameters

```
for (ntree in ntree_values) {
  for (mtry in mtry_values) {
    for (k_fold in k_fold_values) {
      for (maxnodes in c(10, 20, 30)) {
        for (maxdepth in c(5, 10, 15)) {
          rf_model <- randomForest(
            x = train_data[, -which(names(train_data) %in% target_columns)],
            y = train_data$sales_price + train_data$total_price,
            ntree = ntree,
            mtry = mtry,
            maxnodes = maxnodes,
            maxdepth = maxdepth
          )
```

Result

Description: df [288 x 9]

ntree <dbl>	mtry <dbl>	k_fold <dbl>	maxnodes <dbl>	maxdepth <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
500	15	5	30	15	0.8812029	0.05283845	0.2298662	0.1813471
500	15	10	10	5	0.8040510	0.09272521	0.3045081	0.2381953
500	15	10	10	10	0.8048066	0.09209831	0.3034770	0.2379861
500	15	10	10	15	0.8057245	0.09214314	0.3035509	0.2375047
500	15	10	20	5	0.8574510	0.06438483	0.2537417	0.1994943
500	15	10	20	10	0.8571976	0.06478411	0.2545272	0.2005870
500	15	10	20	15	0.8587350	0.06425417	0.2534841	0.1996474
500	15	10	30	5	0.8802552	0.05304895	0.2303236	0.1819457
500	15	10	30	10	0.8803260	0.05295387	0.2301171	0.1813966
500	15	10	30	15	0.8815448	0.05261584	0.2293814	0.1812126

261-270 of 288 rows   Previous 1 ... 24 25 26 27 28 29 Next

Due to the extensive changes in parameters, only the section displaying the best values is included, resulting in 28 pages of tables.

Conclusion

Changing different parameters sometimes leads to better results for our model, while in some cases, using default values for certain parameters yields better outcomes. For instance, increasing the ntree value enhances model performance, but it's beneficial to maintain a balanced number for the mtry parameter. Additionally, restricting maxnodes and maxdepth to specific numbers decreases our model's performance, while increasing the k_fold value improves the model's performance.

8.2 Support Vector Machine (SVM)

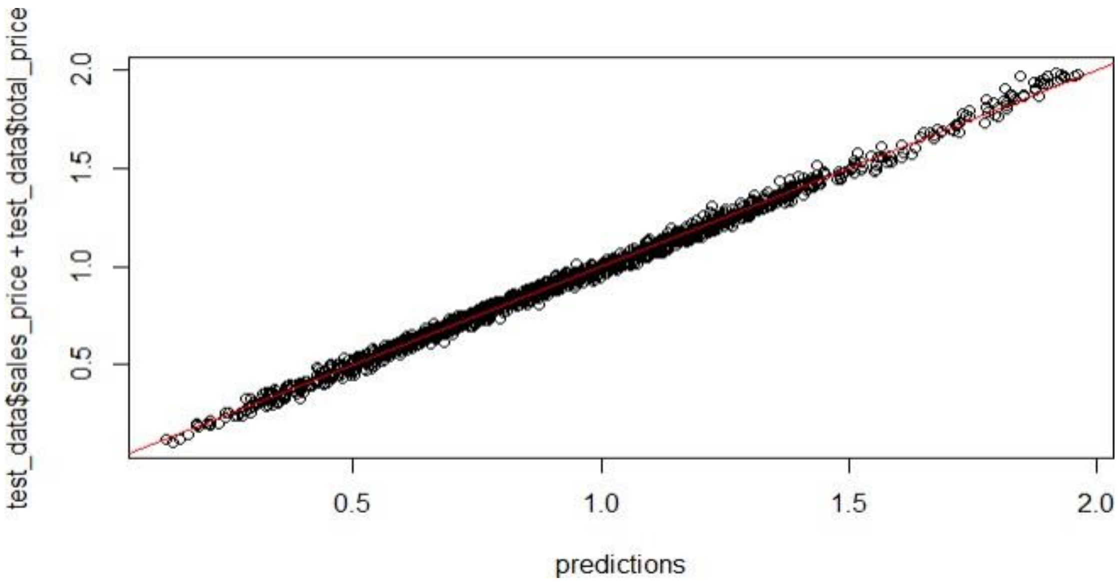
8.2.1 SVM with Default Parameters

```
library(caret)
library(e1071)
library(Metrics)

# Create Train and Test set
set.seed(42)
trainIndex <- createDataPartition(dataset$total_price, p = 0.8, list = FALSE)
train_data <- dataset[trainIndex, ]
test_data <- dataset[-trainIndex, ]

#Create Model
svm_model <- train(
  x = train_data[, -which(names(train_data) %in% target_columns)],
  y = train_data$sales_price + train_data$total_price,
  method = "svmRadial",
  trControl = trainControl(method = "cv", number = 10),
  metric = "RMSE"
)

predictions <- predict(svm_model, newdata = test_data[, -which(names(test_data) %in% target_columns)])

errors <- predictions - (test_data$sales_price + test_data$total_price)

r_squared <- cor(predictions, test_data$sales_price + test_data$total_price)^2
mse <- mean(errors^2)
rmse <- sqrt(mse)
mae <- mean(abs(errors))

print(paste("R2:", r_squared))
print(paste("MSE:", mse))
print(paste("RMSE:", rmse))
print(paste("MAE:", mae))

plot(predictions, test_data$sales_price + test_data$total_price)
abline(0, 1, col = "red")
```

Result

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9958596	0.001736094	0.04166647	0.03214065

Regression Graph



8.2.2 SVM based on C, Sigma parameters

```
C_values <- c(0.1, 0.5 , 1, 10)
sigma_values <- c(0.001, 0.01, 1, 10)

metrics <- data.frame()

for (C in C_values) {
  for (sigma in sigma_values) {
    svm_model <- train(
      x = train_data[, -which(names(train_data) %in% target_columns)],
      y = train_data$sales_price + train_data$total_price,
      method = "svmRadial",
      trControl = trainControl(method = "cv", number = 10),
      metric = "RMSE",
      tuneGrid = data.frame(C = C, sigma = sigma)
    )
```

Result

Description: df [16 × 6]

C <dbl>	Sigma <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.1	1e-03	0.97106408	0.021643343	0.14711677	0.10186406
0.1	1e-02	0.98942287	0.004735437	0.06881451	0.04790824
0.1	1e+00	0.10217675	0.410551005	0.64074254	0.50648579
0.1	1e+01	0.04085508	0.417560564	0.64618926	0.51357016
0.5	1e-03	0.98186118	0.007930857	0.08905536	0.06136237
0.5	1e-02	0.99570666	0.001800935	0.04243742	0.03147775
0.5	1e+00	0.12403495	0.385357818	0.62077195	0.47892434
0.5	1e+01	0.04527730	0.410218333	0.64048289	0.50579768
1.0	1e-03	0.98481351	0.006435169	0.08021951	0.05522598
1.0	1e-02	0.99642140	0.001501163	0.03874484	0.02931876

1-10 of 16 rows   Previous 1 2 Next

C <dbl>	Sigma <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
1.0	1e+00	0.14156298	0.370278011	0.60850473	0.46396171
1.0	1e+01	0.05242834	0.403885425	0.63551981	0.50102154
10.0	1e-03	0.99442454	0.002342930	0.04840383	0.03458297
10.0	1e-02	0.99690122	0.001303514	0.03610421	0.02784705
10.0	1e+00	0.14974126	0.361876359	0.60156160	0.45812656
10.0	1e+01	0.05374585	0.399515363	0.63207228	0.49907610

11-16 of 16 rows Previous 1 2 Next

Conclusion

Our model achieved the best result with a C value of 10 and a Sigma value of 0.01.

8.3 KNN

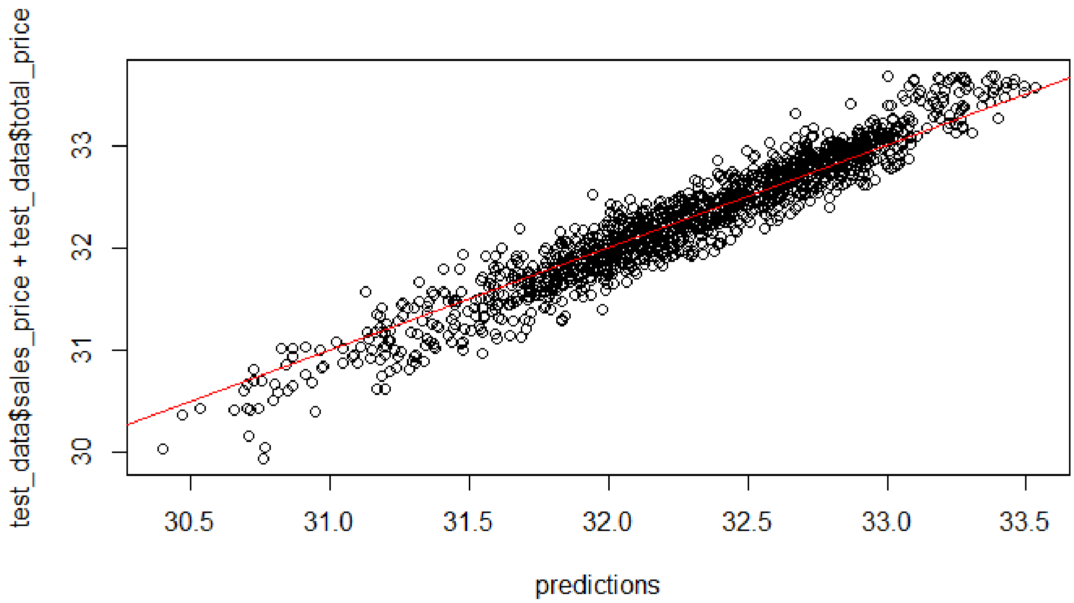
8.3.1 KNN with Default Parameters

```
set.seed(42)
trainIndex <- createDataPartition(dataset$total_price, p = 0.8, list = FALSE)
train_data <- dataset[trainIndex, ]
test_data <- dataset[-trainIndex, ]

knn_model <- train(
  x = train_data[, -which(names(train_data) %in% target_columns)],
  y = train_data$sales_price + train_data$total_price,
  method = "knn",
  trControl = trainControl(method = "cv", number = 10),
  metric = "RMSE",
)

predictions <- predict(knn_model, newdata = test_data[, -which(names(test_data) %in% target_columns)])

errors <- predictions - (test_data$sales_price + test_data$total_price)

r_squared <- cor(predictions, test_data$sales_price + test_data$total_price)^2
mse <- mean(errors^2)
rmse <- sqrt(mse)
mae <- mean(abs(errors))

print(paste("R-kare Skoru:", r_squared))
print(paste("MSE:", mse))
print(paste("RMSE:", rmse))
print(paste("MAE:", mae))

plot(predictions, test_data$sales_price + test_data$total_price)
abline(0, 1, col = "red")
```

Result

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9242683	0.03575501	0.18909	0.1474365

Regression Graph



8.3.2 KNN based on K, k_values parameters

```
k_values <- c(3, 5, 7 ,9, 11,13)
k_fold_values <- c(5, 7, 10, 15)

metrics <- data.frame()

for (k in k_values) {
  for (k_fold in k_fold_values) {
    print(k)
    knn_model <- train(
      x = train_data[, -which(names(train_data) %in% target_columns)],
      y = train_data$sales_price + train_data$total_price,
      method = "knn",
      trControl = trainControl(method = "cv", number = k_fold),
      metric = "RMSE",
      tuneGrid = data.frame(k = k),
    )
```

Result

k <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
3	5	0.8902965	0.04610926	0.2147307	0.1602240
3	7	0.8902965	0.04610926	0.2147307	0.1602240
3	10	0.8902965	0.04610926	0.2147307	0.1602240
5	5	0.9090311	0.03935628	0.1983842	0.1499091
5	7	0.9090311	0.03935628	0.1983842	0.1499091
5	10	0.9090311	0.03935628	0.1983842	0.1499091
7	5	0.9183904	0.03674460	0.1916888	0.1475414
7	7	0.9183904	0.03674460	0.1916888	0.1475414
7	10	0.9183904	0.03674460	0.1916888	0.1475414
9	5	0.9242683	0.03575501	0.1890900	0.1474365

1-10 of 12 rows   Previous 1 2 Next

k	k_fold	R_Squared	MSE	RMSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
7	10	0.9183904	0.03674460	0.1916888	0.1475414
7	15	0.9183904	0.03674460	0.1916888	0.1475414
9	5	0.9242683	0.03575501	0.1890900	0.1474365
9	7	0.9242683	0.03575501	0.1890900	0.1474365
9	10	0.9242683	0.03575501	0.1890900	0.1474365
9	15	0.9242683	0.03575501	0.1890900	0.1474365
11	5	0.9264972	0.03565688	0.1888303	0.1481121
11	7	0.9264972	0.03565688	0.1888303	0.1481121
11	10	0.9264972	0.03565688	0.1888303	0.1481121
11	15	0.9264972	0.03565688	0.1888303	0.1481121

11-20 of 24 rows Previous 1 2 3 Next

k	k_fold	R_Squared	MSE	RMSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
13	5	0.9291423	0.03582236	0.1892680	0.1484458
13	7	0.9291423	0.03582236	0.1892680	0.1484458
13	10	0.9291423	0.03582236	0.1892680	0.1484458
13	15	0.9291423	0.03582236	0.1892680	0.1484458

8.3.3 KNN based on K, k_values parameters

k	R_Squared	MSE	RMSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
3	0.8902965	0.04610926	0.2147307	0.1602240
3	0.8902965	0.04610926	0.2147307	0.1602240
3	0.8902965	0.04610926	0.2147307	0.1602240
3	0.8902965	0.04610926	0.2147307	0.1602240
5	0.9090311	0.03935628	0.1983842	0.1499091
5	0.9090311	0.03935628	0.1983842	0.1499091
5	0.9090311	0.03935628	0.1983842	0.1499091
5	0.9090311	0.03935628	0.1983842	0.1499091
7	0.9183904	0.03674460	0.1916888	0.1475414
7	0.9183904	0.03674460	0.1916888	0.1475414

1-10 of 24 rows Previous 1 2 3 Next

k	R_Squared	MSE	RMSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
7	0.9183904	0.03674460	0.1916888	0.1475414
7	0.9183904	0.03674460	0.1916888	0.1475414
9	0.9242683	0.03575501	0.1890900	0.1474365
9	0.9242683	0.03575501	0.1890900	0.1474365
9	0.9242683	0.03575501	0.1890900	0.1474365
9	0.9242683	0.03575501	0.1890900	0.1474365
11	0.9264972	0.03565688	0.1888303	0.1481121
11	0.9264972	0.03565688	0.1888303	0.1481121
11	0.9264972	0.03565688	0.1888303	0.1481121
11	0.9264972	0.03565688	0.1888303	0.1481121

11-20 of 24 rows Previous 1 2 3 Next

k	R_Squared	MSE	RMSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
13	0.9291423	0.03582236	0.1892680	0.1484458
13	0.9291423	0.03582236	0.1892680	0.1484458
13	0.9291423	0.03582236	0.1892680	0.1484458
13	0.9291423	0.03582236	0.1892680	0.1484458

Conclusion

In our KNN model, as the value of k increases, the performance of our model improves.

8.4 Linear Regression

8.4.1 Linear Regression Default Parameter

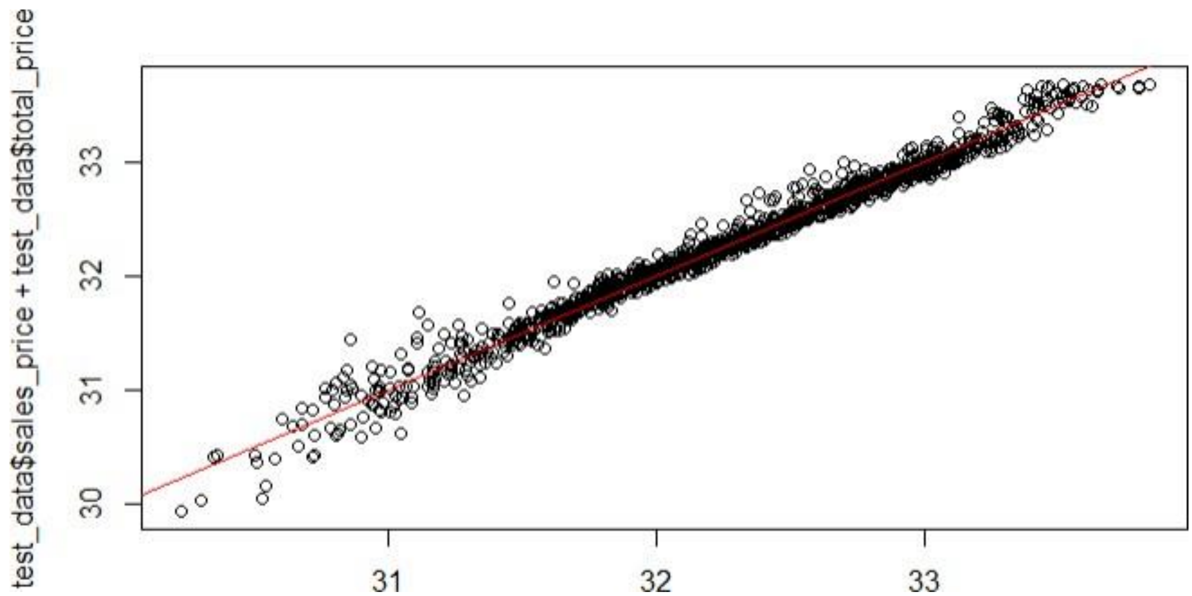
```r
set.seed(42)
trainIndex <- createDataPartition(dataset$total_price, p = 0.8, list = FALSE)
train_data <- dataset[trainIndex, ]
test_data <- dataset[-trainIndex, ]

lm_model <- train(
  x = train_data[, -which(names(train_data) %in% target_columns)],
  y = train_data$sales_price + train_data$total_price,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10),
  metric = "RMSE"
)

predictions <- predict(lm_model, newdata = test_data[, -which(names(test_data) %in% target_columns)])

errors <- predictions - (test_data$sales_price + test_data$total_price)

r_squared <- cor(predictions, test_data$sales_price + test_data$total_price)^2
mse <- mean(errors^2)
rmse <- sqrt(mse)
mae <- mean(abs(errors))

print(paste("R-kare Skoru:", r_squared))
print(paste("MSE:", mse))
print(paste("RMSE:", rmse))
print(paste("MAE:", mae))

plot(predictions, test_data$sales_price + test_data$total_price)
abline(0, 1, col = "red")

results <- data.frame(
  `R2` = r_squared,
  `MSE` = mse,
  `RMSE` = rmse,
  `MAE` = mae
)
```

Result

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9809795	0.008015242	0.08952788	0.06250183

1 row

Regression Graph



8.4.2 Lasso Regression

Alpha	Lambda	R_Squared	MSE	RMSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0.0	0.1	0.9760579	0.01184151	0.1088187	0.07606386
0.0	1.0	0.9264400	0.09321280	0.3053077	0.23650215
0.0	0.1	0.9760579	0.01184151	0.1088187	0.07606386
1.0	0.1	0.8400253	0.09642099	0.3105173	0.24018499
1.0	1.0	NA	0.41847511	0.6468965	0.51639289
1.0	0.1	0.8400253	0.09642099	0.3105173	0.24018499
0.1	0.1	0.9691291	0.01660699	0.1288681	0.09341997
0.1	1.0	0.8394652	0.18772303	0.4332702	0.33952643
0.1	0.1	0.9691291	0.01660699	0.1288681	0.09341997

9 rows

9) Min-Max Normalization

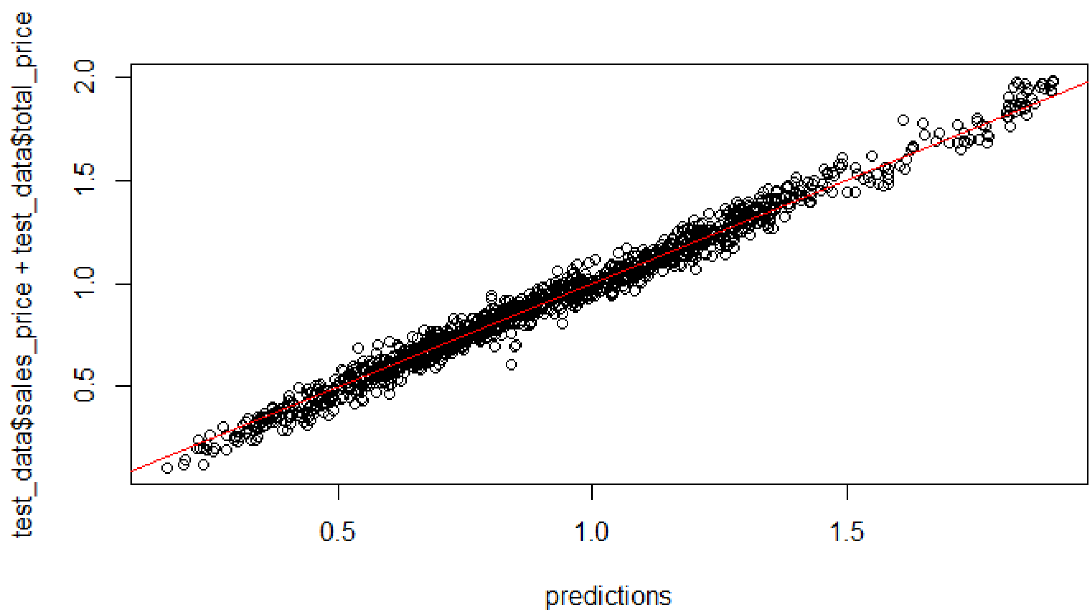
9.1 Random Forest

9.1.1 Random Forest Default Parameter

Result

R2	MSE	RMSE	MAE
<dbl>	<dbl>	<dbl>	<dbl>
0.9842605	0.002128213	0.04613256	0.03564307

Regression Graph



9.1.2 Random Forest based on ntree and mtry parameters

Result

Description: df [16 x 6]

ntree <dbl>	mtry <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
50	2	0.9197454	0.022400669	0.14966853	0.11626615
50	5	0.9750583	0.003748842	0.06122779	0.04819769
50	15	0.9838893	0.002146881	0.04633445	0.03559541
50	20	0.9842329	0.002096236	0.04578467	0.03541373
100	2	0.9257852	0.020073917	0.14168245	0.10890068
100	5	0.9754859	0.003700619	0.06083272	0.04773822
100	15	0.9842545	0.002108523	0.04591865	0.03527450
100	20	0.9842282	0.002097063	0.04579370	0.03513553
300	2	0.9288936	0.019904204	0.14108226	0.11002484
300	5	0.9773144	0.003478332	0.05897738	0.04618869

1-10 of 16 rows

Previous 1 2 Next

Description: df [16 x 6]

ntree <dbl>	mtry <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
300	15	0.9845815	0.002058215	0.04536756	0.03482078
300	20	0.9849977	0.001991349	0.04462454	0.03429554
500	2	0.9259547	0.020798879	0.14421817	0.11222391
500	5	0.9771390	0.003466189	0.05887435	0.04631760
500	15	0.9848186	0.002026061	0.04501179	0.03456800
500	20	0.9851012	0.001979353	0.04448992	0.03414428

11-16 of 16 rows Previous 1 2 Next

9.1.3 Random Forest based on ntree , mtry and k_fold parameters

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
50	2	5	0.9197454	0.022400669	0.14966853	0.11626615
50	2	10	0.9250977	0.019564522	0.13987323	0.10907932
50	5	5	0.9752340	0.003691721	0.06075954	0.04727501
50	5	10	0.9733052	0.004001189	0.06325495	0.04918287
50	15	5	0.9833999	0.002213550	0.04704838	0.03634472
50	15	10	0.9839696	0.002136041	0.04621733	0.03557919
50	20	5	0.9842301	0.002089717	0.04571343	0.03508345
50	20	10	0.9840088	0.002117323	0.04601438	0.03519626
100	2	5	0.9250234	0.020797826	0.14421452	0.11145339
100	2	10	0.9236668	0.020702455	0.14388348	0.11180502

1-10 of 32 rows Previous 1 2 3 4 Next

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
100	5	5	0.9763414	0.003601832	0.06001526	0.04710450
100	5	10	0.9754338	0.003633792	0.06028094	0.04699669
100	15	5	0.9842050	0.002103107	0.04585965	0.03520983
100	15	10	0.9845326	0.002067639	0.04547130	0.03502749
100	20	5	0.9844282	0.002064285	0.04543440	0.03493248
100	20	10	0.9844778	0.002060489	0.04539261	0.03495562
300	2	5	0.9335015	0.019292228	0.13889646	0.10735402
300	2	10	0.9280574	0.020920611	0.14463959	0.11242246
300	5	5	0.9768220	0.003511384	0.05925693	0.04684671
300	5	10	0.9773383	0.003471246	0.05891728	0.04637476

11-20 of 32 rows Previous 1 2 3 4 Next

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
300	15	5	0.9848533	0.002025648	0.04500720	0.03452782
300	15	10	0.9849339	0.002012978	0.04486622	0.03448568
300	20	5	0.9846465	0.002036167	0.04512391	0.03468468
300	20	10	0.9848299	0.002014904	0.04488768	0.03442697
500	2	5	0.9293018	0.020851116	0.14439915	0.11208406
500	2	10	0.9316795	0.020318815	0.14254408	0.11059594
500	5	5	0.9770718	0.003478795	0.05898131	0.04649602
500	5	10	0.9769371	0.003525227	0.05937362	0.04671912
500	15	5	0.9848207	0.002028223	0.04503580	0.03464048
500	15	10	0.9848909	0.002021787	0.04496428	0.03451824

21-30 of 32 rows Previous 1 2 3 4 Next

ntree <dbl>	mtry <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
500	20	5	0.9849902	0.001994974	0.04466513	0.03424287
500	20	10	0.9849335	0.002002155	0.04474545	0.03439120

9.1.4 Random Forest based on ntree , mtry , k_fold , maxnodes and maxdepth parameters

ntree <dbl>	mtry <dbl>	k_fold <dbl>	maxnodes <dbl>	maxdepth <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
500	15	5	30	15	0.8926436	0.01468770	0.1211928	0.09791773
500	15	10	10	5	0.8038754	0.02840976	0.1685519	0.13351107
500	15	10	10	10	0.8034613	0.02840661	0.1685426	0.13344461
500	15	10	10	15	0.8087610	0.02812976	0.1677193	0.13305001
500	15	10	20	5	0.8659775	0.01886665	0.1373559	0.10963564
500	15	10	20	10	0.8654613	0.01891232	0.1375221	0.10964013
500	15	10	20	15	0.8652186	0.01886792	0.1373605	0.10963885
500	15	10	30	5	0.8919452	0.01477645	0.1215584	0.09812583
500	15	10	30	10	0.8924020	0.01476413	0.1215077	0.09829855
500	15	10	30	15	0.8913166	0.01485196	0.1218686	0.09844158

261-270 of 288 rows   Previous 1 ... 24 25 26 27 28 29 Next

Due to the extensive changes in parameters, only the section displaying the best values is included, resulting in 28 pages of tables.

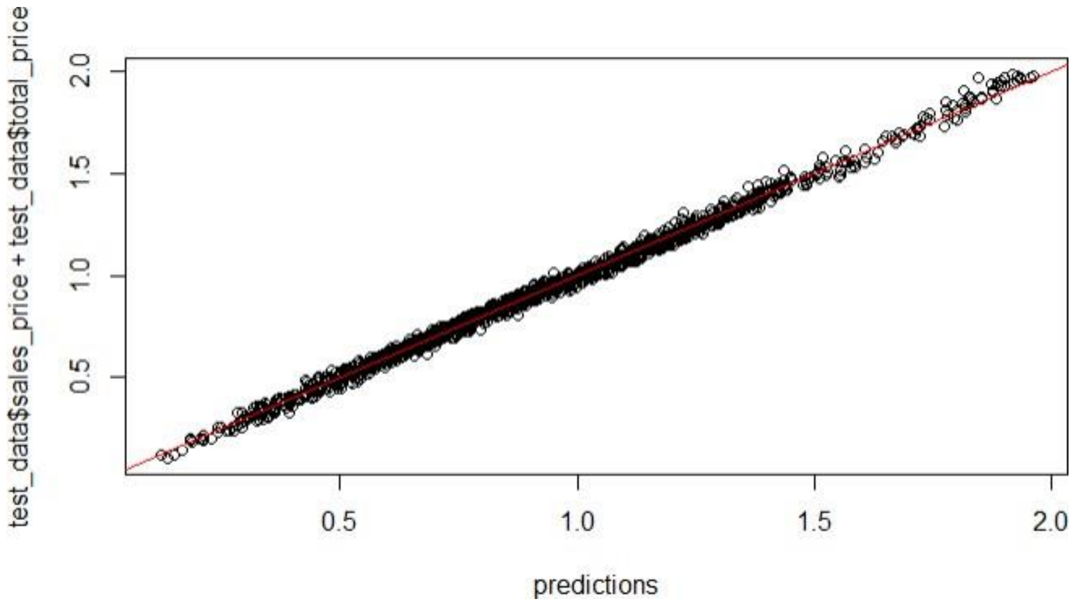
9.2 Support Vector Machine

9.2.1 SVM Default Parameters

Result

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9955887	0.000577444	0.02403007	0.01901622

Regression Graph



9.2.2 SVM based on C, Sigma parameters

Description: df [16 x 6]

C <dbl>	Sigma <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.1	1e-03	0.95356079	0.0105220903	0.10257724	0.06522408
0.1	1e-02	0.98298642	0.0024309724	0.04930489	0.03150775
0.1	1e+00	0.11212456	0.1278170649	0.35751513	0.28339224
0.1	1e+01	0.04104926	0.1301842916	0.36081060	0.28755642
0.5	1e-03	0.96711155	0.0046881794	0.06847028	0.04360503
0.5	1e-02	0.99499831	0.0006498829	0.02549280	0.02029121
0.5	1e+00	0.13180698	0.1189368408	0.34487221	0.26845862
0.5	1e+01	0.04389622	0.1272534478	0.35672601	0.28389484
1.0	1e-03	0.97280869	0.0037050717	0.06086930	0.03932049
1.0	1e-02	0.99594406	0.0005316558	0.02305766	0.01856891

1-10 of 16 rows   Previous 1 2 Next

Description: df [16 x 6]

C <dbl>	Sigma <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
1.0	1e+00	0.15079487	0.1139259658	0.33752921	0.26055402
1.0	1e+01	0.05221213	0.1253853569	0.35409795	0.28191404
10.0	1e-03	0.99299986	0.0009093829	0.03015598	0.02350918
10.0	1e-02	0.99665309	0.0004488779	0.02118674	0.01702865
10.0	1e+00	0.16498226	0.1107405716	0.33277706	0.25697140
10.0	1e+01	0.05424876	0.1241311342	0.35232249	0.28152442

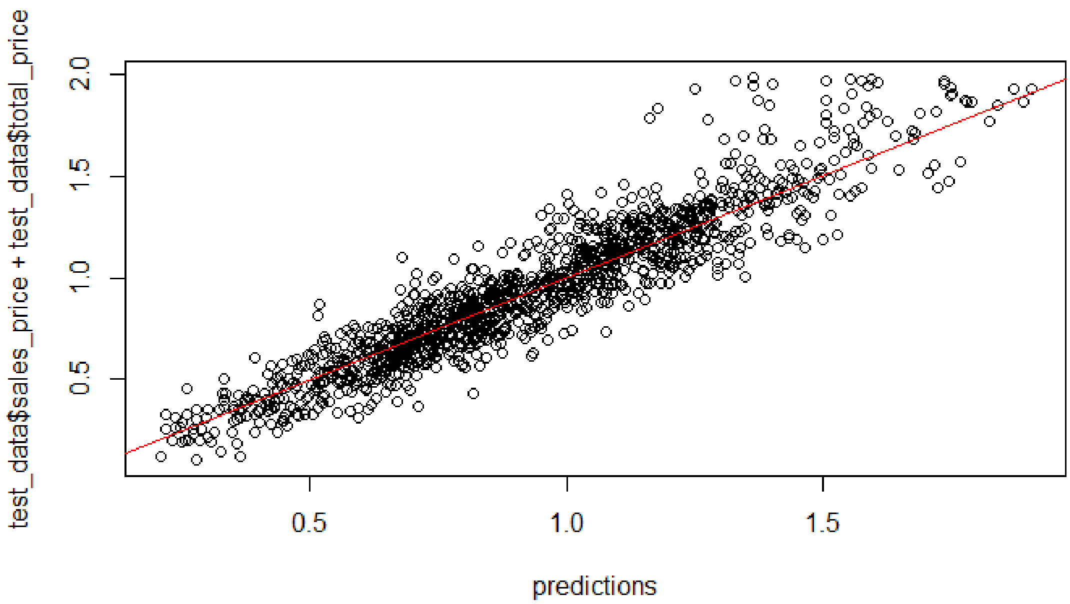
9.3 KNN

9.3.1 KNN with Default Parameters

Result

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.8723139	0.0170232	0.130473	0.09624709

Regression Graph



9.3.2 KNN based on K, k_values parameters

Result

k <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
3	5	0.8553037	0.01884594	0.1372805	0.10165864
3	7	0.8553037	0.01884594	0.1372805	0.10165864
3	10	0.8553037	0.01884594	0.1372805	0.10165864
3	15	0.8553037	0.01884594	0.1372805	0.10165864
5	5	0.8723139	0.01702320	0.1304730	0.09624709
5	7	0.8723139	0.01702320	0.1304730	0.09624709
5	10	0.8723139	0.01702320	0.1304730	0.09624709
5	15	0.8723139	0.01702320	0.1304730	0.09624709
7	5	0.8793048	0.01677938	0.1295352	0.09733357
7	7	0.8793048	0.01677938	0.1295352	0.09733357

1-10 of 24 rows Previous 1 2 3 Next

k <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
7	10	0.8793048	0.01677938	0.1295352	0.09733357
7	15	0.8793048	0.01677938	0.1295352	0.09733357
9	5	0.8847866	0.01662909	0.1289538	0.09910891
9	7	0.8847866	0.01662909	0.1289538	0.09910891
9	10	0.8847866	0.01662909	0.1289538	0.09910891
9	15	0.8847866	0.01662909	0.1289538	0.09910891
11	5	0.8861686	0.01684247	0.1297786	0.09988134
11	7	0.8861686	0.01684247	0.1297786	0.09988134
11	10	0.8861686	0.01684247	0.1297786	0.09988134
11	15	0.8861686	0.01684247	0.1297786	0.09988134

11-20 of 24 rows Previous 1 2 3 Next

k <dbl>	k_fold <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
13	5	0.8865284	0.01726890	0.1314112	0.10109138
13	7	0.8865284	0.01726890	0.1314112	0.10109138
13	10	0.8865284	0.01726890	0.1314112	0.10109138
13	15	0.8865284	0.01726890	0.1314112	0.10109138

9.3.3 Changing the cross-validation method in KNN

Result

k <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
3	0.8553037	0.01884594	0.1372805	0.10165864
3	0.8553037	0.01884594	0.1372805	0.10165864
3	0.8553037	0.01884594	0.1372805	0.10165864
3	0.8553037	0.01884594	0.1372805	0.10165864
5	0.8723139	0.01702320	0.1304730	0.09624709
5	0.8723139	0.01702320	0.1304730	0.09624709
5	0.8723139	0.01702320	0.1304730	0.09624709
5	0.8723139	0.01702320	0.1304730	0.09624709
7	0.8793048	0.01677938	0.1295352	0.09733357
7	0.8793048	0.01677938	0.1295352	0.09733357

1-10 of 24 rows Previous 1 2 3 Next

k <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
7	0.8793048	0.01677938	0.1295352	0.09733357
7	0.8793048	0.01677938	0.1295352	0.09733357
9	0.8847866	0.01662909	0.1289538	0.09910891
9	0.8847866	0.01662909	0.1289538	0.09910891
9	0.8847866	0.01662909	0.1289538	0.09910891
9	0.8847866	0.01662909	0.1289538	0.09910891
11	0.8861686	0.01684247	0.1297786	0.09988134
11	0.8861686	0.01684247	0.1297786	0.09988134
11	0.8861686	0.01684247	0.1297786	0.09988134
11	0.8861686	0.01684247	0.1297786	0.09988134

11-20 of 24 rows Previous 1 2 3 Next

k <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
13	0.8865284	0.01726890	0.1314112	0.10109138
13	0.8865284	0.01726890	0.1314112	0.10109138
13	0.8865284	0.01726890	0.1314112	0.10109138
13	0.8865284	0.01726890	0.1314112	0.10109138

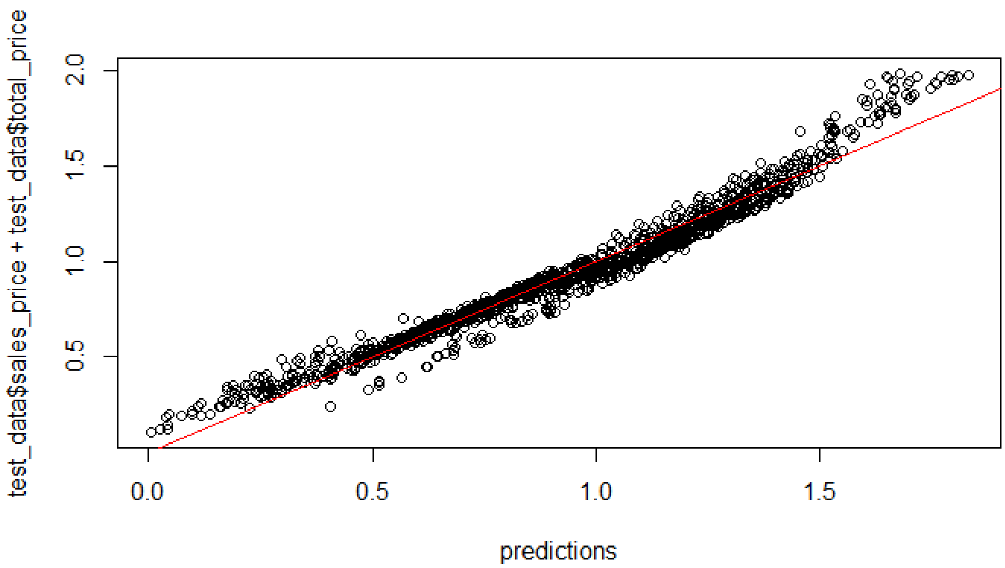
9.4 Linear Regression

9.4.1 Linear Regression with Default Parameter

Result

R2 <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.9632262	0.004798767	0.06927313	0.05060293

Regression Graph



9.4.1 Lasso Regression

Result

Alpha <dbl>	Lambda <dbl>	R_Squared <dbl>	MSE <dbl>	RMSE <dbl>	MAE <dbl>
0.0	0.1	0.9543901	0.007610068	0.08723571	0.05967037
0.0	1.0	0.8766830	0.048475942	0.22017253	0.17236359
0.0	0.1	0.9543901	0.007610068	0.08723571	0.05967037
1.0	0.1	0.7368749	0.056820892	0.23837133	0.18538742
1.0	1.0	NA	0.129924826	0.36045086	0.29043654
1.0	0.1	0.7368749	0.056820892	0.23837133	0.18538742
0.1	0.1	0.9447951	0.010596728	0.10294041	0.07328013
0.1	1.0	0.7031756	0.094551907	0.30749294	0.24469871
0.1	0.1	0.9447951	0.010596728	0.10294041	0.07328013

9 rows

10) References

[1] Kunwarakash, "Chennai House Price Prediction," Kaggle, [Online]. Available: https://www.kaggle.com/code/kunwarakash/chennai-house-price-prediction/.

[2] "Support Vector Machines in Machine Learning," YouTube, [Online]. Available: https://www.youtube.com/watch?v=6EXPYzbfLCE.

[3] "Support Vector Machines Tutorial in R," DataCamp, [Online]. Available: https://www.datacamp.com/tutorial/support-vector-machines-r.

[4] "Linear Regression Tutorial in R," DataCamp, [Online]. Available: https://www.datacamp.com/tutorial/linear-regression-R.