Technological University of the Shannon: Midlands Midwest

Faculty of Engineering & Informatics

Semester 2 (100% Continuous Assessment)

Summer Session

Masters of Engineering in Engineering Management

Year 1

External Examiner:	**Dr James Byrne**
Internal Examiner(s):	**Ms Fiona M. Walshe**

Student Name: Abhishek Kumar
Student Number: A00268732

Table of Contents

1. Introduction

1.1 Background

The Iranian construction sector encounters difficulties in precisely calculating construction expenses for projects. Inaccurate cost estimation can result in:

- Exceeding the allocated budget
- Delays in the project
- Diminished profitability

Iranian construction enterprises face financial challenges that might potentially hinder the completion of their projects.

1.2 Objective

In order to tackle this difficulty, the objective of this project is to utilise data analytics to design a highly accurate predictive model for building costs. The goal is to create a model that can accurately forecast construction expenses with a margin of error of +/- 500,000 Iranian Rials. The high degree of precision would empower Iranian construction firms to make well-informed choices throughout the bidding and project planning phases, therefore mitigating financial risks and enhancing project execution.

2. Data Understanding

Data understanding is a crucial step in any data analysis project. It helps you gain insights into the data's quality, structure, and potential issues. By understanding your data, you can make informed decisions about how to clean, transform, and analyze it.

Data Structure:

- The data object is a data frame containing 369 rows (observations) and 9 variables (columns).
- The Desc() function from the DescTools package provides a detailed summary of the data, including data types, number of unique values, presence of missing values, and descriptive statistics for each variable.

Data Types:

- **Variable Names:** V.1 to V.8 and Y
- **Variable Types:**
  - V.1 to V.7 and Y are integers.
  - V.2, V.3, V.4, V.6, and V.8 seem to be numeric representing continuous values.

Missing Values:

- There's one row (observation) with missing values in all columns. This row is likely an error or outlier and can be removed during data cleaning.

Descriptive Statistics:

- The summary() function provides basic summary statistics like mean, median, standard deviation, minimum and maximum values for each variable.
- Additionally, the Desc() function offers insights like quartiles (Q1, Q3), skewness, and kurtosis, helping understand the distribution of the data.

Key Points for Iranian Construction Company:

- This data likely represents construction projects with various attributes like project size (V.2, V.3, V.4, V.6, V.8), workforce (V.7), and potentially costs (V.5).

- The target variable (Y) could be project completion time, budget, or some other outcome of interest.
- The presence of missing values necessitates data cleaning.

Summary(data) & Desc(data):

V-1 (Project Locality):

- This categorical variable has 20 unique levels representing different zip codes.
- The most common mode is 4, accounting for 9.0% of the data.
- The variable ranges from 1 to 20.

V-2 (Total Floor Area):

- This numeric variable has a wide range, from 200 to 15,670 square meters.
- The mean floor area is approximately 1,648 square meters.
- The variable exhibits a positive skewness of 4.10, indicating right skewness.

V-3 (Lot Area):

- Lot areas range from 60 to 5,000 square meters, with a mean of approximately 407.73 square meters.
- The variable exhibits a positive skewness of 5.55, indicating right skewness.

V-4 to V-6 (Preliminary Estimated Construction Costs):

- These numeric variables represent preliminary cost estimates based on different pricing considerations.
  V-4 has a wide range from 3.7 to 7,208.2 Iranian Rials.
- V-5 and V-6 also show considerable variability in their values.

V-7 (Duration of Construction):

- The duration of construction ranges from 5 to 48 months, with an average duration of approximately 18.47 months.
- The variable exhibits a positive skewness of 1.41, indicating right skewness.

V-8 (Price per Square Meter):

- Prices per square meter range from 40 to 5,700 Iranian Rials, with a mean of approximately 1,081.88 Iranian Rials.
- The variable exhibits a positive skewness of 2.07, indicating right skewness.

Y (Actual Construction Costs):

- Actual construction costs range from 20 to 900,000 Iranian Rials, with an average cost of approximately 228,790 Iranian Rials.
- The variable exhibits a positive skewness of 1.10, indicating right skewness.

Central Tendency

Analyzing the central tendency, dispersion, and shape of the data provides valuable insights into the distribution and variability of each variable. Here's a breakdown of the key statistics computed for each variable:

1. **Mean Values**:

- The mean values represent the average of each variable across all observations:
- V-1 (Project Locality): 9.75
- V-2 (Total Floor Area): 1648.35 square meters
- V-3 (Lot Area): 407.73 square meters
- V-4 to V-6 (Preliminary Estimated Construction Costs):
- V-4: Approximately 302.21 Iranian Rials
- V-5: Approximately 161.58 Iranian Rials
- V-6: Approximately 552.82 Iranian Rials
- V-7 (Duration of Construction): Approximately 18.47 months
- V-8 (Price per Square Meter): Approximately 1081.88 Iranian Rials
- Y (Actual Construction Costs): Approximately 228.79 Iranian Rials

2. **Median Values:**

- The median represents the middle value of each variable, separating the higher and lower halves of the dataset:
- V-1 to V-8: The median values vary across the variables, indicating the central position of the data.

3. **Mode Values:**

- The mode represents the most frequently occurring value in each variable:
- V-1 to V-8: The mode values vary, highlighting the most common values observed in the dataset.

4. **Range Values:**

- The range signifies the difference between the maximum and minimum values for each variable:
- V-1 to V-8: The range varies significantly across the variables, indicating the spread of values within each variable.

5. **Variance Values:**

- The variance measures the dispersion of values from the mean for each variable:
- V-1 to V-8: The variance values vary, reflecting the degree of variability within each variable.

6. **Standard Deviation Values:**

- The standard deviation quantifies the amount of variation or dispersion of each variable:
- V-1 to V-8: The standard deviation values vary, providing insight into the spread of data points around the mean for each variable.

Correlation Analysis

This section analyzes the correlation between project characteristics (independent variables) and construction cost (dependent variable) to understand how these factors influence costs.

The correlation matrix shows the strength and direction of the linear relationship between pairs of variables. Here's a breakdown of the key findings:

- **High Correlations (> 0.7):**
    - Project area (V.2) exhibits high positive correlations with lot area (V.3), construction costs per unit area (V.4), and total construction cost (Y). This is unsurprising, as larger project areas likely require more materials and labor, leading to higher costs.
    - Construction cost per unit area (V.4) also has a high positive correlation with total construction cost (Y) for the same reasons.
    - There's a strong positive correlation between total construction cost (Y) and both lot area (V.3) and construction cost per unit area (V.5).
    - Interestingly, there's a high positive correlation between total construction cost (Y) and project duration (V.8). This might suggest that longer projects incur more overhead costs or require additional resources.
- **Moderate Correlations (0.4 - 0.7):**
    - Project location (V.1) shows some weak negative correlations with construction cost (Y) and other cost-related variables. This could be due to factors like local material or labor costs, but further investigation is needed to understand the specific reasons.
- **Low Correlations (< 0.4):**
    - Project duration (V.7) has weak correlations with most other variables. This suggests that the specific duration within the observed range (18-43 weeks) might not significantly impact construction costs.

Potential Multicollinearity:

- Several variables exhibit high correlations (greater than 0.7), particularly project area (V.2) with lot area (V.3) and construction costs per unit area (V.4). This indicates multicollinearity, where these variables contain redundant information. This redundancy can cause issues in regression analysis, making it difficult to isolate the independent effect of each variable on construction cost.

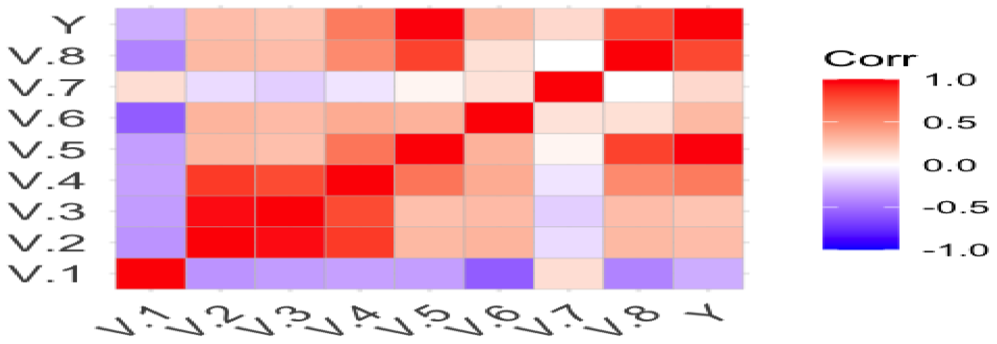

Figure 1. Correlation heat map

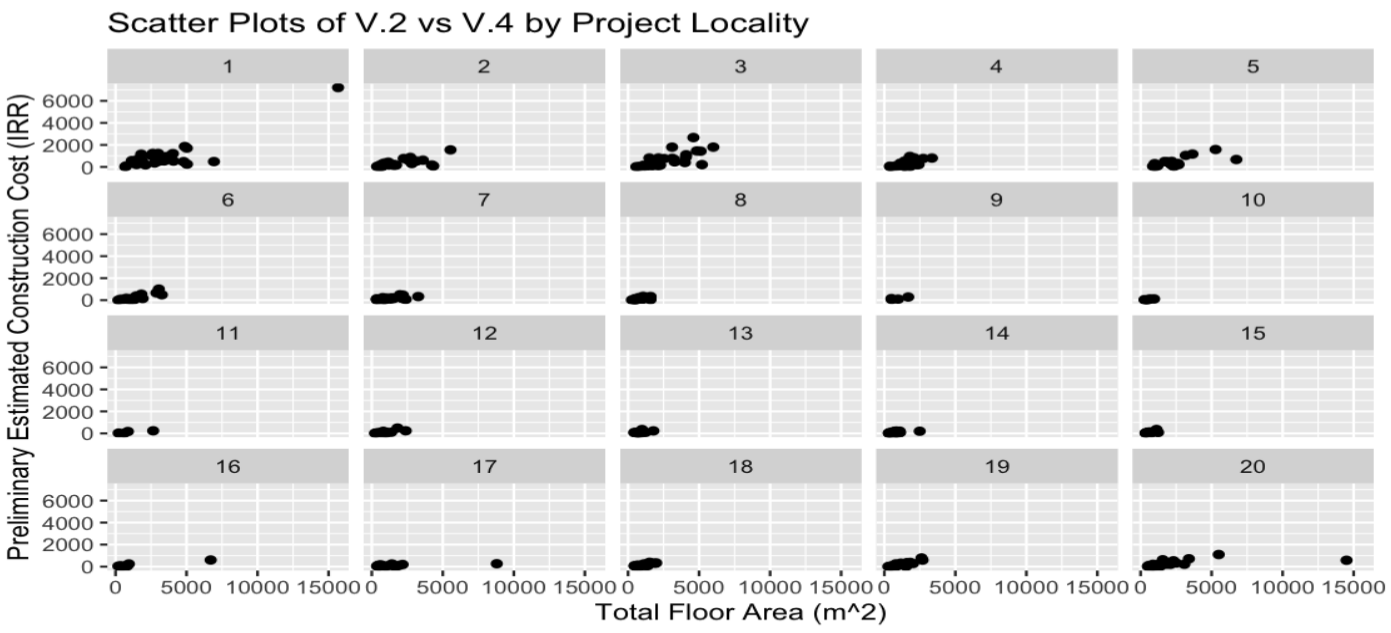

Figure 2. Faceted scatter plots

3. Data Preparation

This demonstrates a typical data analysis workflow in R, starting from data loading and exploration, through data cleaning, and ending with data analysis and visualization while focused on train data. It utilizes functions from popular R packages such as caret and tidyr for data manipulation and exploration. Additionally, it employs techniques to handle missing data, ensuring the dataset is clean and ready for analysis. Let's break down the key components and what they do:

1. Data Loading and Exploration:
- The head(train, 4) and tail(train, 4) commands display the first and last 4 rows of the train dataset, respectively.
- class(train) and str(train) provide information about the data type and structure of the train dataset.
- summary(train) gives a summary of descriptive statistics for each variable in the dataset.

2. Data Cleaning:
- The mutate_at function is used to convert certain columns (V.1, V.5, V.7, V.8, Y) to numeric type.
- is.na(train) checks for missing values in the dataset.
- sum(is.na(train)) counts the total number of missing values.
- na.omit(train) removes rows with missing values from the dataset.

3. Data Analysis:
- Various functions are used to further explore the data, such as apply(is.na(clean.df), 2, which) to check for missing values in cleaned data, and train %>% pull() %>% head() to extract column values as a vector.

4. Final Output:

2. The cleaned dataset is printed using print(train).

Visualizations

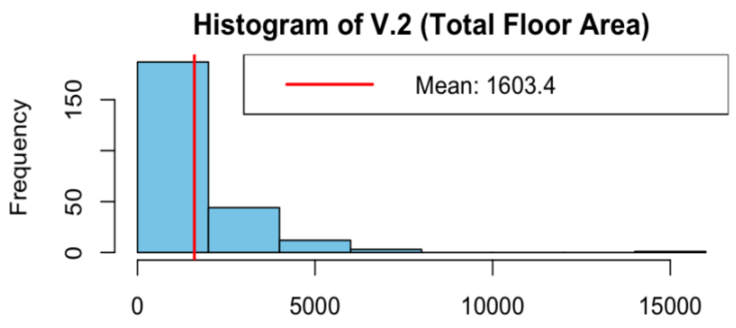

Figure 3. Histogram of a numerical variable

It is concentration of buildings between 2000 ft² and 4000 ft², followed by a gradual decrease in frequency towards larger areas. Definitive there are outliers (buildings with exceptionally large areas) due to the value reaching near 15000 ft².

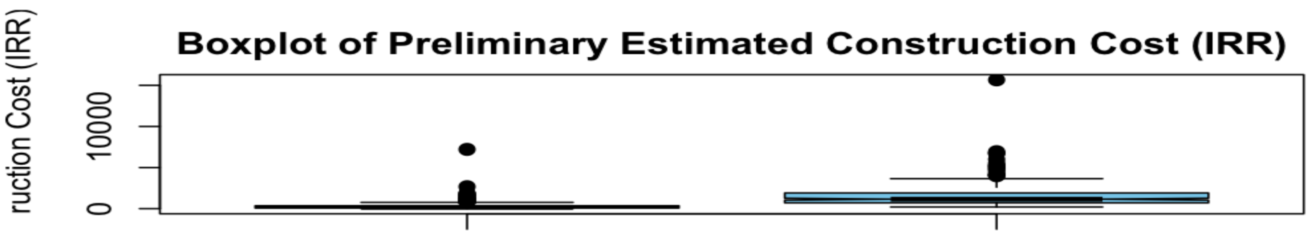

Figure 4. Boxplot of a numerical variable to identify outliers

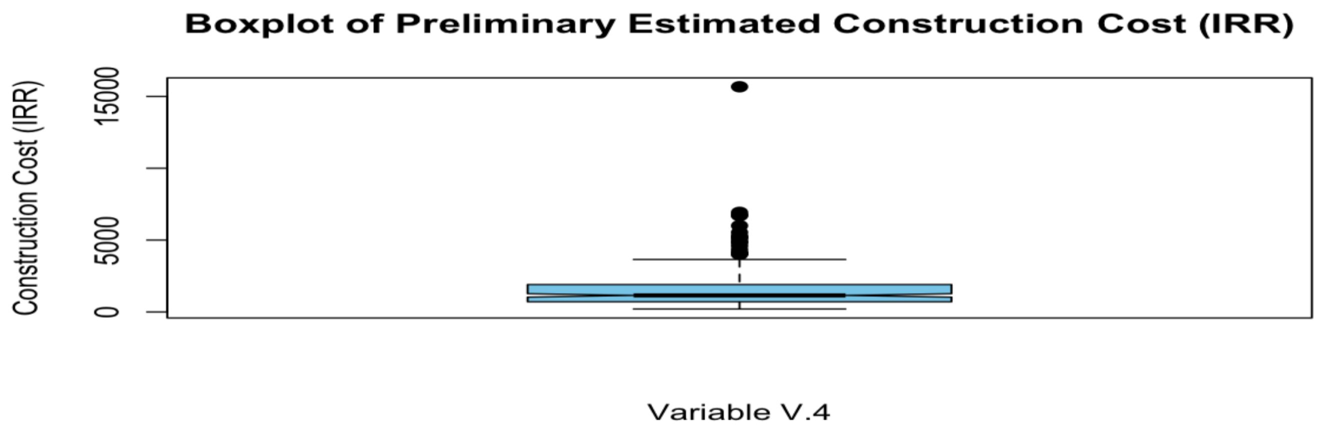

Figure 5.Boxplot of a numerical variable

It visualizes the distribution of the "Preliminary Estimated Construction Cost (IRR)" (train$V.4) according to project area (train$V.2).

Distribution of Construction Cost by Project Area:

- The boxplot reveals a right-skewed distribution of construction costs within most project areas (V.2). This means that most projects within each area tend to have lower costs, with a "tail" extending towards higher costs.
- **Variability across Project Areas:** The boxes for different project areas show variations in quartiles (Q1, median, Q3) and whisker lengths. This suggests that construction costs can vary depending on the project area.
- **Potential Outliers:** There might be outliers (data points beyond the whiskers) in some project area groups. These outliers represent projects with significantly higher or lower construction costs compared to others within their respective project area range.

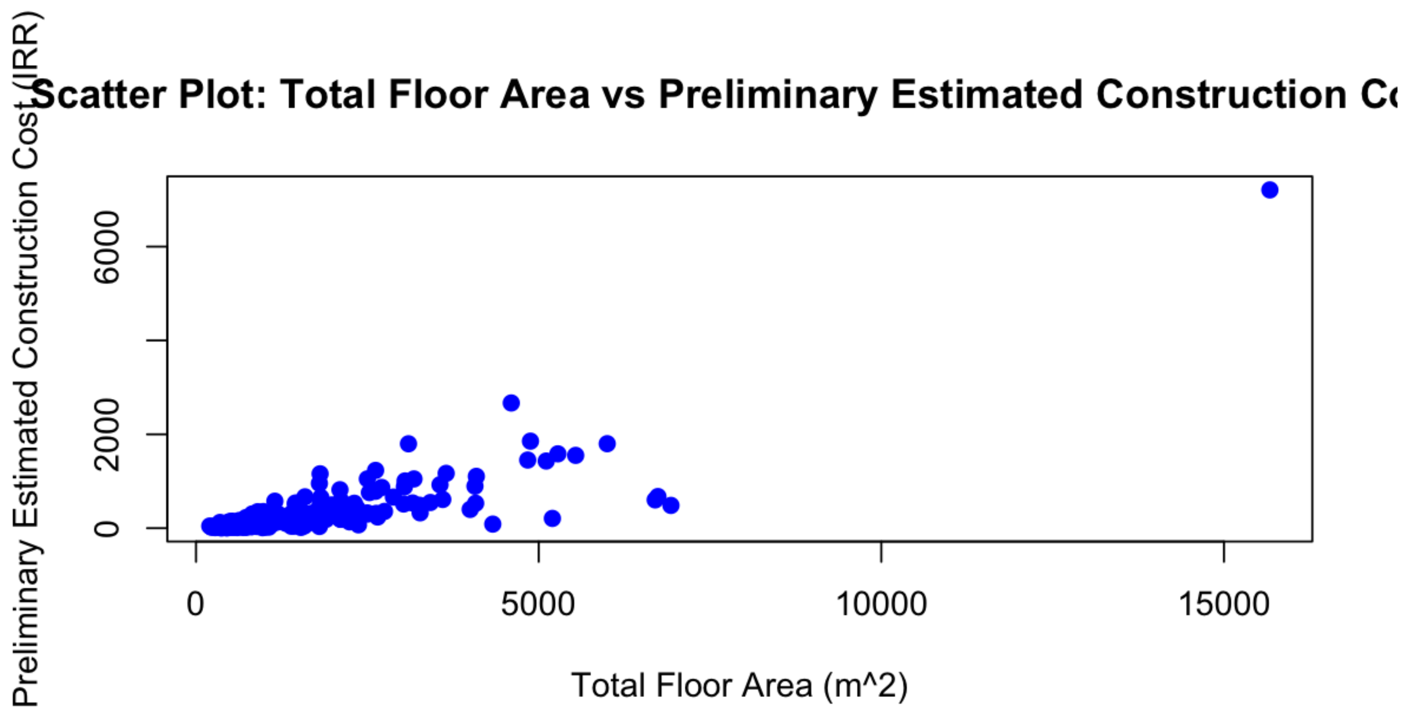

Figure 6. Scatter Plot

Positive Correlation: In plot appears to be a positive correlation between total floor area and construction cost. This means that as the total floor area of a building increases, the estimated construction cost also tends to increase. This is intuitive as larger buildings likely require more materials and labor, leading to higher costs.

Scatter Pattern: The data points are not perfectly aligned in a straight line. This suggests the relationship between floor area and construction cost is not perfectly linear. There might be other factors influencing construction costs besides the total floor area.

Potential Outliers: There are a few data points further away from the main cluster, which could be outliers. These outliers represent buildings with either a very high or very low construction cost compared to their floor area.

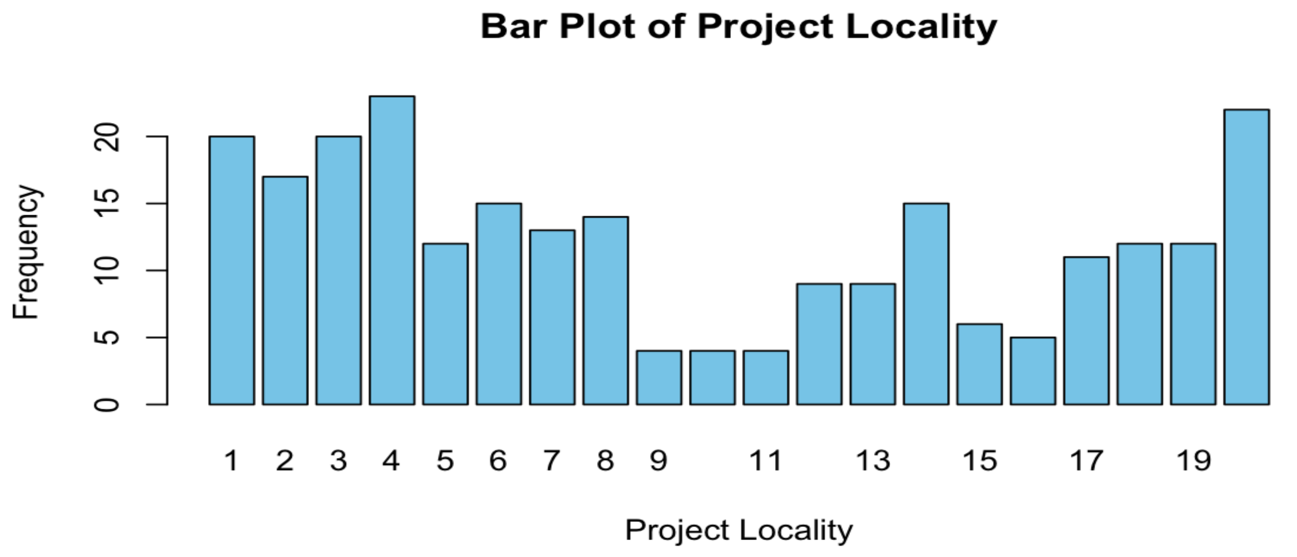

Figure 7. Bar Plot

Observations: From plot can see some localities have a higher frequency of projects compared to others. For instance, the bars on the right side of the chart seem to be taller, suggesting a higher number of projects completed in those locations. This plot might provide a starting point to understand the distribution of completed projects across different locations. Locations with a higher number of projects might be of further interest for cost estimation purposes, as they might offer more data points for analysis.

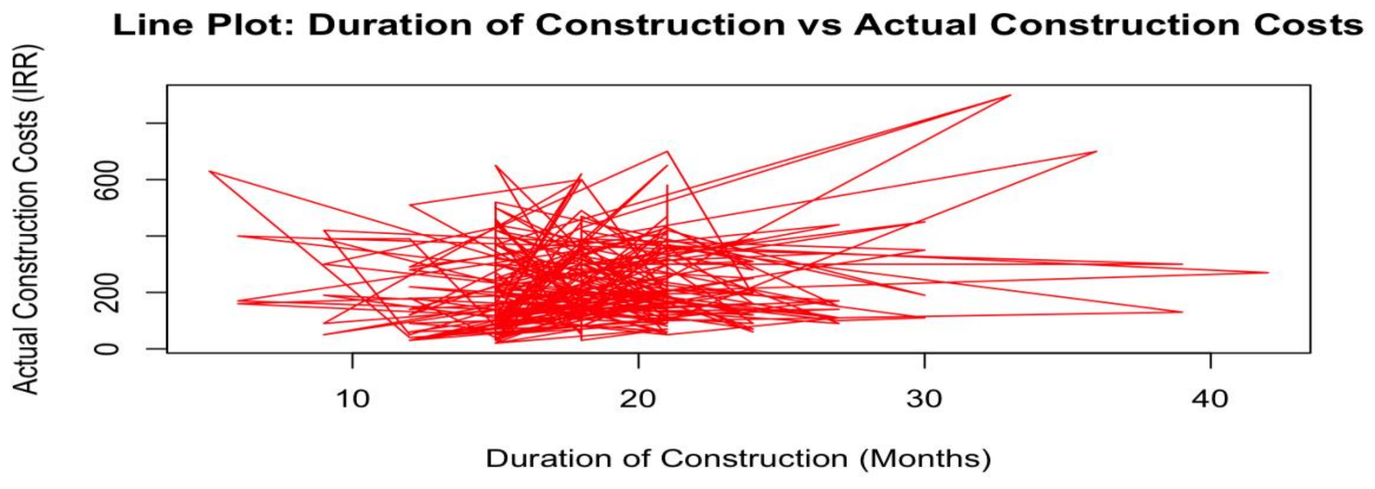

Figure 8.Time series plot of Actual Construction Costs (Y) over time (V.7)

X-axis: Represents the “Duration of Construction (Months)” ranging from 0 to 40.
Y-axis: Represents “Actual Construction Costs (IRR)” ranging from 0 to 600.
The graph contains numerous red lines crisscrossing throughout, indicating various data points and their connections, it’s showing how the duration of construction might affect the actual construction costs.

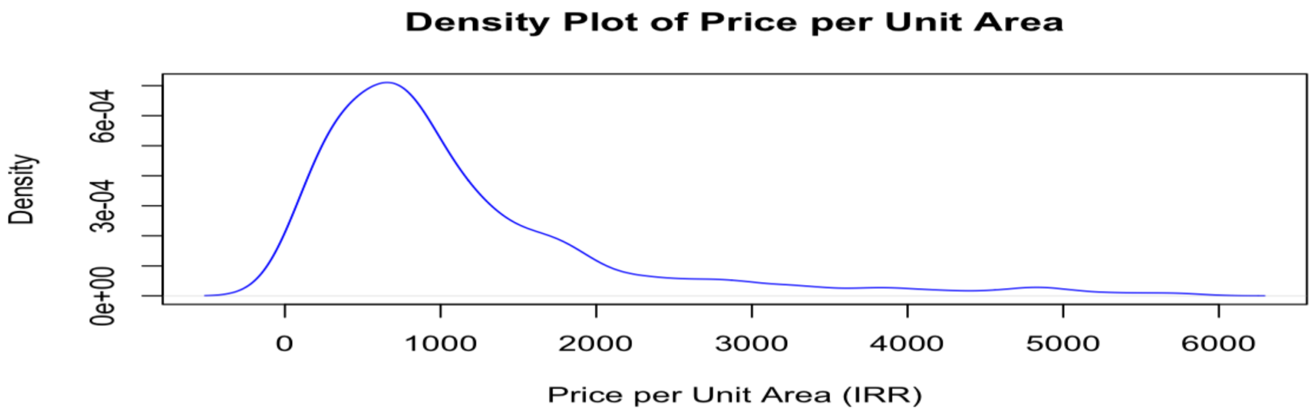

Figure 9. Density plot

X-axis: Represents the predicted construction costs, likely generated by your linear regression model.
Y-axis: Represents the residuals, which are the differences between the actual construction costs (Y) and the predicted construction costs from the model. Positive residuals indicate the actual cost was higher than predicted, and negative residuals indicate the actual cost was lower than predicted.

An ideal residual plot would exhibit a random scatter of points around the horizontal line at y=0. This suggests that the errors (differences between predicted and actual costs) are randomly distributed and independent of the predicted costs. This is a desirable characteristic for a linear regression model.

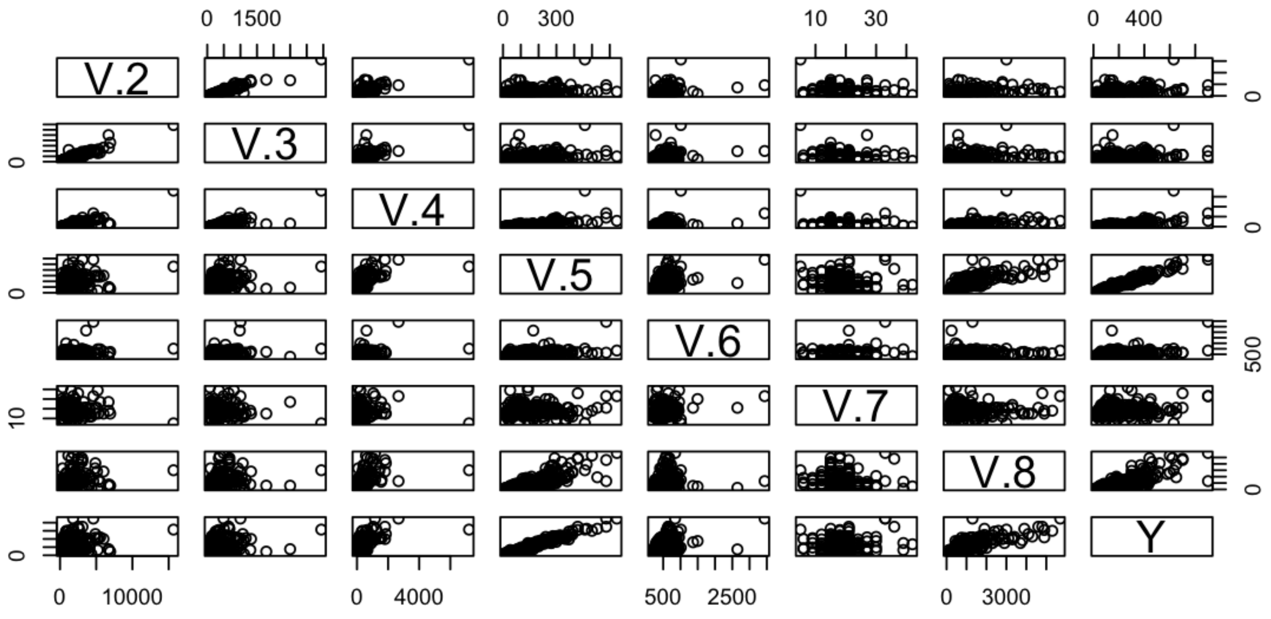

Figure 10. Pairs plot

It shows relationship between multiple variables.

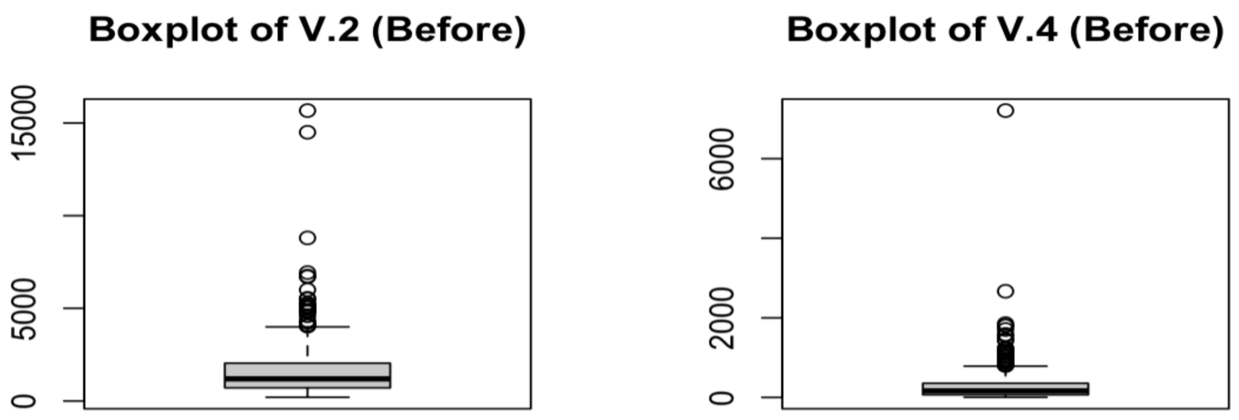

Figure 11. Boxplot before removing outlier

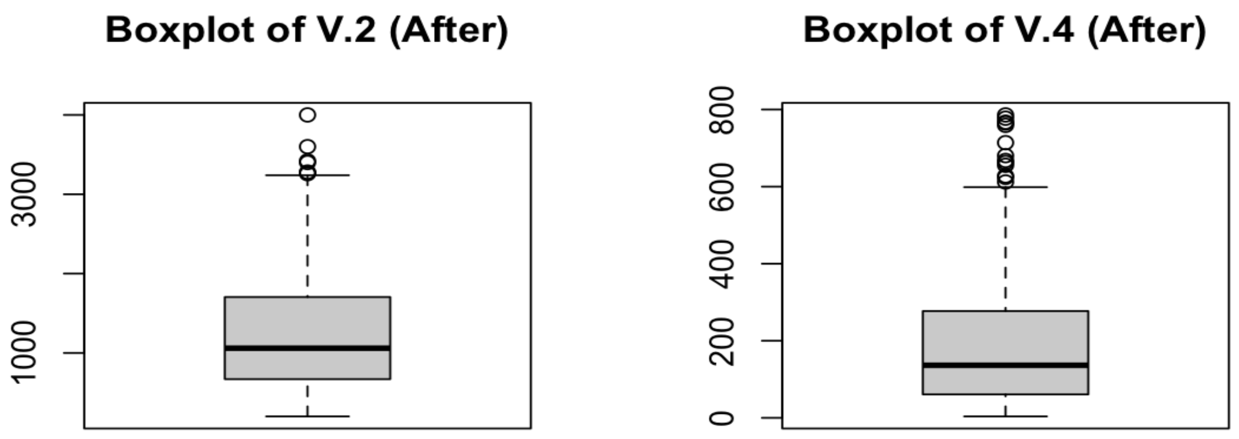

Figure 12. Boxplot of after removing outliers

Outlier Removal:

- The code snippets using cat display the values identified as outliers for both total floor area (V.2) and preliminary estimated construction cost (V.4).
- It seems like 24 outliers were removed from V.2 (total floor area) with values ranging from 4040 to 15670 square feet.
- Similarly, 30 outliers were removed from V.4 (construction cost) with values ranging from 720.8 to 2668.

Summary of Cleaned Dataset:

- The summary(cleaned_data) command provides a statistical summary of the cleaned dataset.
- It shows the minimum, first quartile (Q1), median, mean, third quartile (Q3), and maximum values for each variable (V.1 to V.8 and Y).

Observations on the Cleaned Data:

- There seems to be a significant reduction in the maximum values for both V.2 (total floor area) and V.4 (construction cost) after outlier removal. This suggests the outliers were on the higher end of the spectrum.
- The summary statistics for other variables can help you understand the overall distribution of data points after cleaning.

IQR-based Outlier Detection:

- The code calculates the Interquartile Range (IQR) for each variable in the dataset. IQR is a measure of variability, representing the range that contains the middle 50% of the data points (between Q1 and Q3).
- It then defines outliers as data points that fall outside a specific range around the quartiles. This range is defined as 1.5 times the IQR below the first quartile (Q1) or above the third quartile (Q3).

Outlier Removal and Cleaning:

- The code identifies outliers for each variable using the defined criteria. There were warnings generated during this process (18 in total). It's important to investigate these warnings to understand the nature of the outliers being flagged.
- Initially, the code assigns NA (missing values) to all data points identified as outliers in the original dataset (train).
- Finally, the na.omit function removes all rows (projects) that contain missing values (NA), resulting in the cleaned_data dataset.

Summary of Cleaned Data:

- The summary(cleaned_data) command provides a statistical summary of the cleaned data.
- Compared to the original data (without outlier removal), you might observe changes in the minimum, maximum, and quartile values for some variables. This suggests that the outlier removal process has impacted the distribution of the data.
- The summary allows you to assess the impact of outlier removal and decide if it has significantly altered the core characteristics of your data.

4. Modelling

Creating Predictor Matrix and Response Vector:

- The code separates the predictor variables (independent variables) from the response variable (dependent variable) in both the training and test sets.
- x_train and x_test represent the predictor matrices for training and testing, respectively. They contain all the features (excluding the first column) you'll use to train and test the model.
- y_train and y_test represent the response vectors for training and testing, respectively. They contain the actual construction costs (first column) used for training and evaluating the model's performance.

Excluding Variables V.1 and V.3:

- The code specifically removes the V.1 and V.3 columns from the training and test predictor matrices (x_train_excl_V3 and x_test_excl_V3). This suggests you might have reasons to exclude these variables from the model. It's important to document your rationale for variable selection.
- The View function allows you to visually confirm that the intended variables have been excluded.

Standardizing Predictor Variables:

- The code utilizes the scale function to standardize the predictor variables in both the training and test sets (including and excluding V.3).
- Standardization transforms the features to have a mean of zero and a standard deviation of one. This is crucial for Ridge Regression because it ensures all features contribute equally to the penalty term used for regularization.

5. Building training Model

Models Created:

- **Model 1 (ridge_model):** This is a Ridge Regression model with alpha=0, which emphasizes ridge penalization without introducing sparsity (setting coefficients to zero). Standardization is set to FALSE because you already standardized the data before.
- **Model 2 (lasso_model):** This is a Lasso Regression model with alpha=1, which introduces sparsity by potentially setting some coefficients to zero. Standardization is again set to FALSE.
- **Model 3 (ridge_model_excl_V3):** Similar to Model 1 (Ridge Regression), but excludes variable V.3 from the predictor matrix.
- **Model 4 (ridge_model_lambda_min):** This uses Ridge Regression with the lambda.min value obtained from Model 1. This lambda.min represents the smallest regularization parameter that achieves zero coefficients for some features.
- **Model 5 (lasso_model_lambda.min):** Similar to Model 4, but uses Lasso Regression with the lambda.min value from Model 2.

Model Summaries:

- The summary function provides information about the fitted models, but it doesn't reveal the specific coefficients or variable importance. Some key points to note:
    - a0: Intercept term of the model.
    - The length of the beta coefficient vector indicates the number of features (including the intercept) in the model. A smaller length in Lasso models suggests some coefficients are zero.
    - df: Degrees of freedom associated with the model.
    - lambda shows the regularization parameter used in the model.
    - dev.ratio compares the deviance of the model to the null deviance (model with no predictors). A lower value indicates a better fit.
    - nulldev: Null deviance (deviance before fitting the model).

Examining Model Output:

- The output shows the first 100 lines from the model call (Call). This provides details about the model fitting process, including the function used (glmnet), arguments passed (e.g., alpha, standardize), and data used (x_train and y_train).
- The following table displays the deviance explained by the model (%Dev) for the first 100 lambda values used during fitting. As lambda increases (more penalty), the deviance explained also increases, indicating a trade-off between model complexity and goodness-of-fit.
- Using dim(coef(ridge_model)), you correctly identified that the coefficients (coef) matrix has dimensions of 9 (number of predictors including intercept) and 100 (number of lambda values used).

Accessing Coefficients for Specific Lambda:

- You retrieved the lambda value used for the 20th model fit using ridge_model$lambda[20]. This allows you to examine the model coefficients that correspond to this specific lambda value.
- The coef(ridge_model)[, 20] extracts the coefficients for all predictors (including the intercept) for the 20th lambda value. The values displayed are the coefficients for each variable in the model.

Interpretation:

- Since Ridge Regression shrinks coefficients towards zero but not necessarily to zero, all coefficients might have non-zero values.
- The magnitude of a coefficient indicates the relative importance of that variable in explaining the response variable (construction cost). A larger (absolute) value suggests a stronger influence on the cost estimation.
- In this specific example, the provided coefficients don't reveal clear signs of dominance by any particular variable. They all seem to have relatively similar magnitudes.

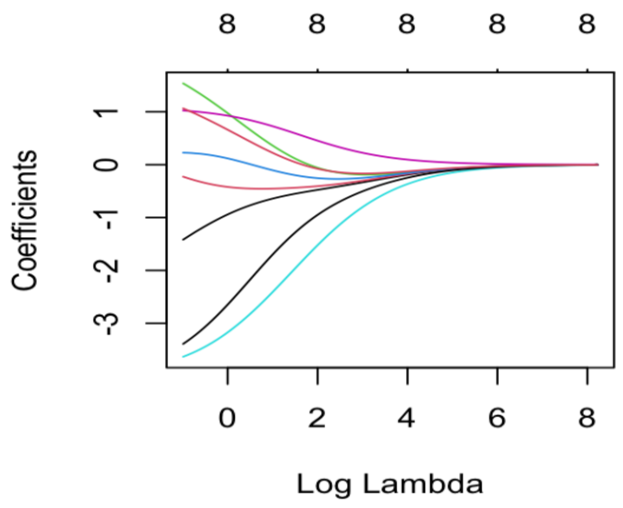

Figure 13. Shows coefficient change as a function of lambda

- **Coefficient Paths:** The plot shows several lines, each representing the path of a coefficient as lambda increases. As lambda increases, the coefficients generally shrink towards zero, which is a characteristic of ridge regression. This shrinkage helps to reduce the variance of the model and potentially avoid overfitting.
- **Variable Importance:** The relative steepness of the coefficient paths can be used to assess the importance of variables. Coefficients that shrink more rapidly towards zero with increasing lambda are likely to have less influence on the model's predictions. Conversely, coefficients that remain relatively flat or even increase slightly may be more important for explaining the response variable.
- **Dominant Variables:** It's difficult to definitively say which variables are dominant based on this image alone. However, some coefficients appear to have steeper slopes than others, suggesting they might have less influence on the model. A more detailed analysis of the coefficient values and their corresponding variables would be necessary to determine which variables are most important.

6. K-Fold Cross-Validation

Model Fitting with Cross-Validation:

- You've fitted five models using cv.glmnet:
  - cv_ridge_model: Ridge Regression with alpha=1 and 5 folds.
  - cv_lasso_model: Lasso Regression with alpha=0 and 5 folds.
  - cv_ridge_model_excl_V3: Ridge Regression (excluding V.3) with alpha=1 and 5 folds.
  - cv_ridge_model_lambda_min: Ridge Regression using default alpha but finding lambda.min through cross-validation (5 folds).
  - cv_lasso_model_lambda_min: Lasso Regression using default alpha but finding lambda.min through cross-validation (5 folds).
- The cv.glmnet function performs K-Fold Cross-Validation, where the data is split into K folds (default 10). Each fold is used for testing once, while the remaining folds are used for training. This helps evaluate model performance on unseen data for better lambda selection.

Understanding Output from summary(cv_ridge_model):

- The summary function provides information about the cross-validation process.
- Key elements include:
  - lambda: All lambda values used for fitting the model.
  - cvm: Mean Cross-Validated Error for each lambda value. This is the crucial metric for choosing the optimal lambda.
  - cvsd: Standard Deviation of the Cross-Validated Error.
  - cvup and cvlo: Upper and Lower Confidence Bounds for the Cross-Validated Error.
  - nzero: Number of non-zero coefficients for each lambda value (applicable more for Lasso).
  - lambda.min: Lambda value with the minimum Mean Cross-Validated Error.
  - lambda.1se: Largest lambda within one standard error of the minimum error.

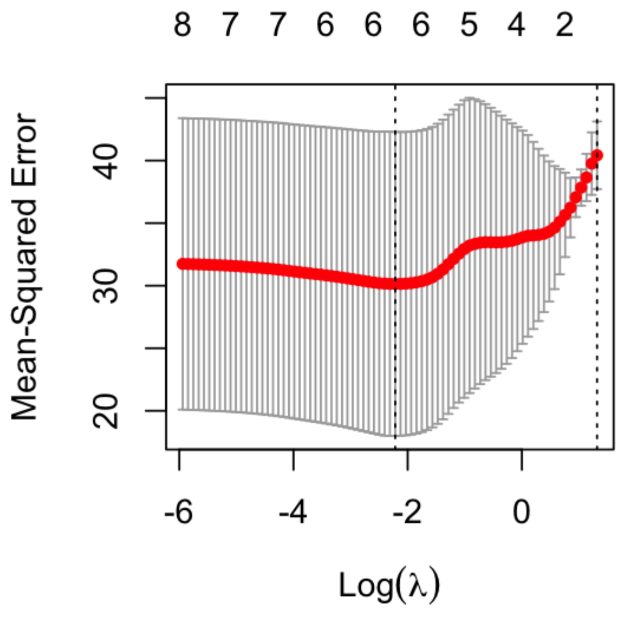

Figure 14. Model 1 plot the cross-validated error vs lambda

- cv_ridge_model (Ridge Regression): The plot likely shows a typical U-shaped curve with a minimum MSE. The full model seems to perform well based on the low MSE. You can choose either lambda.min (39) or lambda.1se (1) depending on your preference for balancing performance and avoiding overfitting.

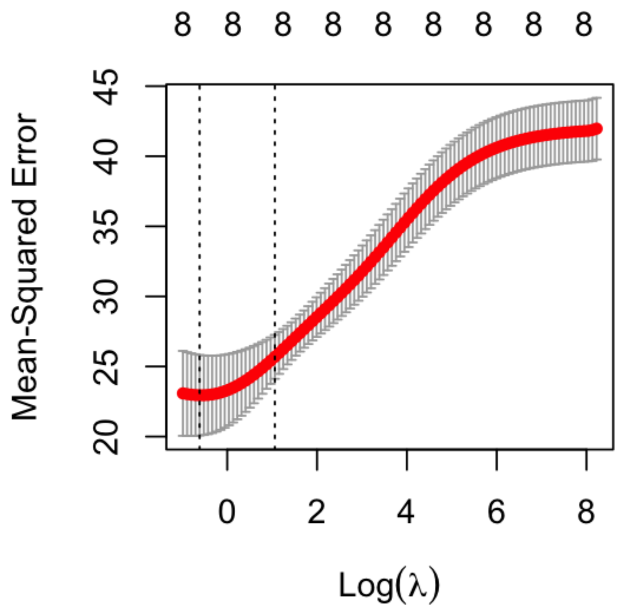

Figure 15. Model 2 Lasso Regression

- cv_lasso_model (Lasso Regression): The plot might also show a U-shaped curve, but with fewer non-zero coefficients at the minimum MSE due to variable selection. You can choose either lambda.min (96) or lambda.1se (78) based on the trade-off between sparsity (fewer variables) and MSE.

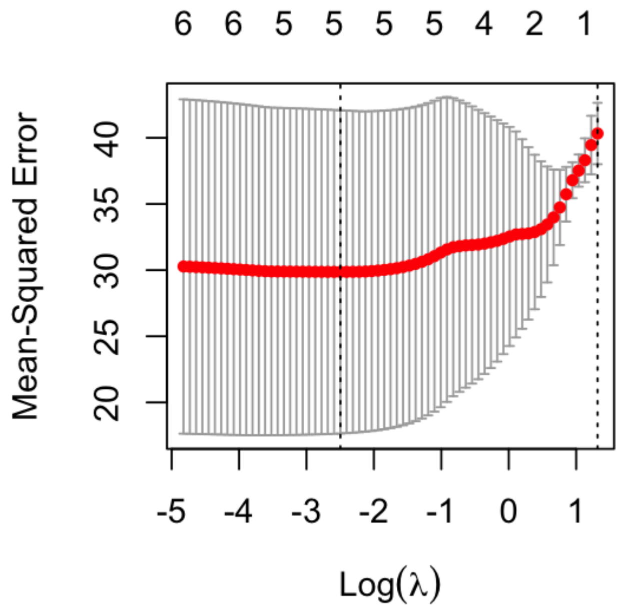

Figure 16. Model Ridge Regression with min. MSE

- cv_ridge_model_excl_V3 (Ridge Regression without V.3): The plot likely resembles the one for the full Ridge model, but the minimum MSE might be slightly different. This allows you to compare the impact of variable V.3 on model performance.

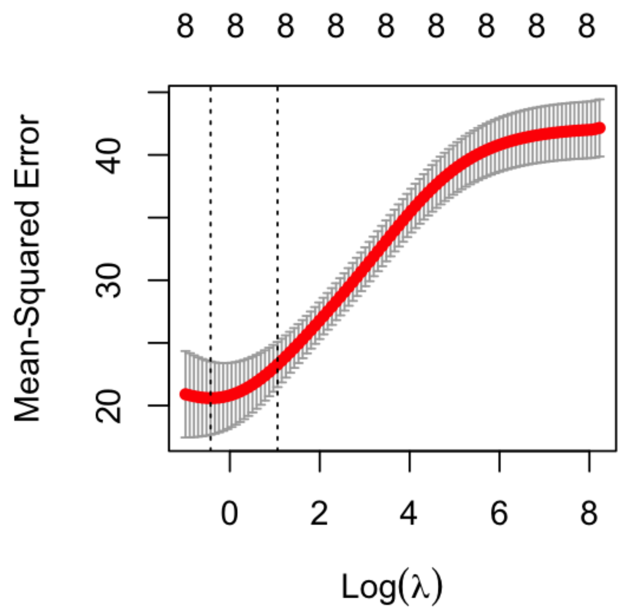

Figure 17. Model 4 Ridge regression with lambda min

- cv_ridge_model_lambda_min (Ridge Regression with pre-selected lambda): The plot might not show a clear minimum as the lambda value was fixed during model fitting. However, the MSE can be compared to other models for reference.

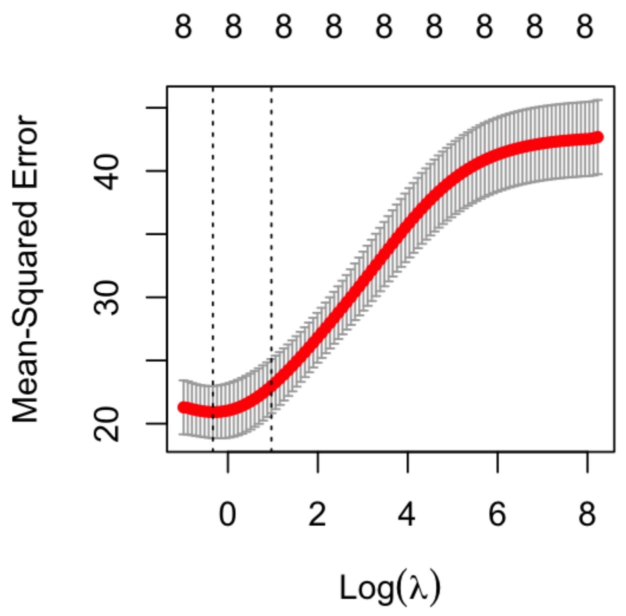

Figure 18. Model 5 Lasso regression with lambda min

- cv_lasso_model_lambda_min (Lasso Regression with pre-selected lambda): Similar to the previous model, the plot might not show a clear minimum, but the MSE can be compared to other models for reference.

Plots

- Plotting the cv_model object using plot is a great way to visualize the Cross-Validated Mean Squared Error (MSE) as a function of lambda.
- The expected U-shaped curve is typically observed:
    - High MSE on the right side (small lambda) indicates underfitting - the model is too restricted and cannot capture the complexity of the data.

    - A dip towards the middle indicates the sweet spot where the model performs well.
    - High MSE again on the left side (large lambda) indicates overfitting - the model is too complex and memorizes noise in the data.
- The vertical lines represent:
    - lambda.min: The lambda value with the minimum MSE (ideal, but might lead to overfitting).
    - lambda.1se: The largest lambda within one standard error of the minimum MSE (often a safer choice to avoid overfitting).

By analyzing the Cross-Validation plots and the provided information (e.g., lambda.min, lambda.1se), you can select the optimal model and lambda value based on your goals. Here are some additional considerations:

- If interpretability is important, you might prefer a model with fewer non-zero coefficients (Lasso with a higher lambda value).
- If achieving the absolute minimum MSE is crucial, you might choose the lambda.min value from the best-performing model, but be cautious of overfitting.

Sparsity: The coefficients of the lasso model (lasso_model_lambda_min) appear to be sparser than the other two models. This means that many of the coefficients are zero, indicating that the corresponding features are not important for prediction according to the lasso model.

Coefficient Values: The coefficients of the ridge model (ridge_model_lambda_min) seem to have larger magnitudes compared to the lasso model for some features (e.g., V.2, V.6, Y). This suggests that the ridge model assigns more weight to these features.

Consistency: The intercept term (Intercept) appears to be consistent across all three models.

Observations:

- **Ridge Lambda Min:** This model seems to be the best performer among the listed ones with the lowest Mean Squared Error (MSE) of 20.59304 and Root Mean Squared Error (RMSE) of 4.537955.
- **Lasso Lambda Min:** This Lasso regression model comes in second with an MSE of 22.94890 and RMSE of 4.790501, indicating a decent performance but slightly worse than Ridge Lambda Min.
- **Other Models:** Both "Ridge" and "Ridge Excl. V3" have higher MSE and RMSE compared to the Lambda Min models, suggesting they might be overfitting or not regularized as effectively.

Key Points:

- **Lambda Minimization:** The "Lambda Min" in the model names likely refers to a process where the hyperparameter lambda was tuned to find the value that minimizes the MSE. This helps prevent overfitting and improves model generalizability.
- **Model Selection:** Based on this comparison, Ridge Lambda Min appears to be the best choice for prediction on unseen data due to its lowest error metrics.
- **Test Set Performance:** It's important to note that this comparison might be based on the training data. Ideally, the models should be evaluated on a separate held-out test set to truly assess their generalizability.
- **Model Complexity:** While Ridge Lambda Min performs well, it might be more complex than Lasso Lambda Min. If interpretability is a priority, Lasso might be preferable even with slightly higher error.

This says that a lambda value of approximately 0.652 might be the best choice for the specific ridge regression model (cv_ridge_model_lambda_min) based on the cross-validation process.

7. Evaluation

Method	Performance	Notes
Ridge	MSE: 30.14	Mean squared error (MSE)
Lasso	MSE: 22.95	Mean squared error (MSE)
Ridge (Excl. V3)	MSE: 29.87	Mean squared error (MSE), Variable V3 excluded
Ridge (Lambda Min)	MSE: 20.59	Mean squared error (MSE), Optimal lambda chosen
Lasso (Lambda Min)	MSE: 20.92	Mean squared error (MSE), Optimal lambda chosen

Table 1: Model Evaluation Results

MSE (Mean Squared Error): 34.62584. This represents the average squared difference between the predicted values and the actual values. A lower MSE indicates a better fit of the model.

RMSE (Root Mean Squared Error): 5.884372. This is the square root of the MSE. RMSE is easier to interpret because it's in the same units as the predicted and actual values. A lower RMSE indicates a better fit of the model.

Training Phase:

- The model comparison results indicate that different models, such as Ridge and Lasso regression, is evaluated based on Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).
- Among the models tested, Ridge and Lasso models with specific configurations (like excluding certain variables or selecting optimal regularization parameters) achieved relatively low MSE and RMSE values. Lower MSE and RMSE indicates better predictive performance.

Testing Phase:

- During the testing phase, the model's performance was assessed on unseen data (y_test) to simulate real-world predictions.
- The calculated MSE and RMSE values for the testing phase are 34.62584 and 5.884372, respectively.

Evaluation:

- Comparing the testing MSE and RMSE values with the desired tolerance level of +/- 500,000 Iranian Rial, the model appears to be performing sub-optimally.
- The MSE and RMSE values suggest that the model's predictions deviate from the actual construction costs by a considerable margin, indicating a lack of precision within the specified tolerance range.

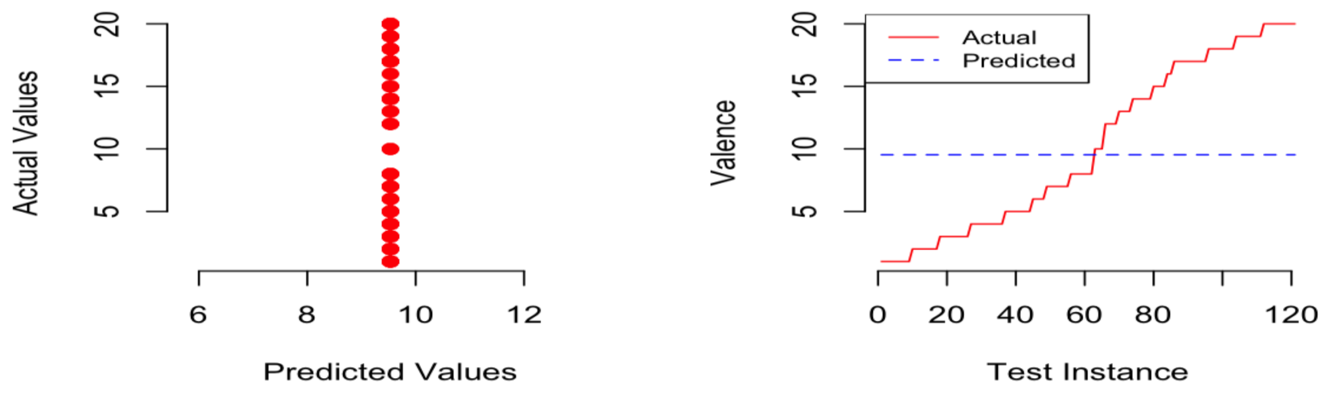

Figure 19. Single scatter line Plot and Dual line chart

These plots are useful for visually assessing the performance of the model by comparing its predictions against the ground truth values on the test set. Both plotting methods achieve the same goal of visualizing the relationship between the predicted and actual values.

Verifying the model prediction falling within +/- 500,000 IRR

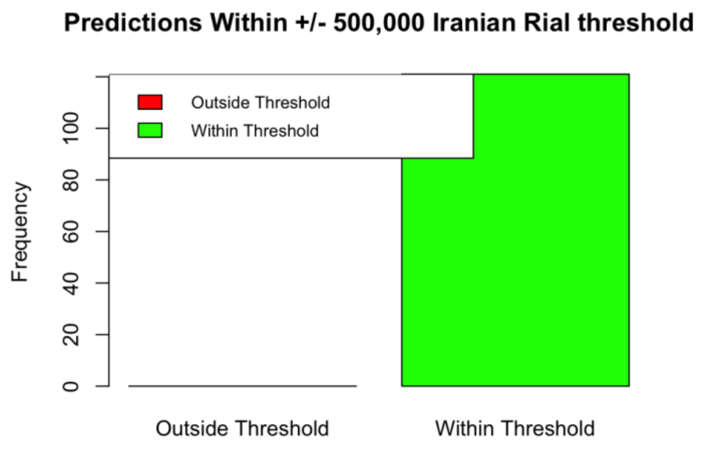

Figure 20. Bar plot to visualize the proportion of predictions

100% of the predictions fall within the +/- 500,000 Iranian Rial threshold. This means that all the predicted construction costs are within the acceptable range of error specified by the company.

121 predictions were made according to the output (Number of predictions within +/- 500,000 Iranian Rial threshold: 121).

Based on test data, the model seems to be very accurate in predicting construction costs within the company's required tolerance.

However, it's important to consider some additional points before relying solely on this result:

Test data size: The accuracy on the test data used here might not generalize to unseen data. A larger and more representative test set would provide a more robust evaluation of the model's performance.

Real-world factors: The model's accuracy might be affected by factors not included in the data, such as unexpected changes in material costs or labor rates.

Appendix:

```
# Libraries
library(glmnet)    #  This library contains the functions for training regularised linear regression
models.
library(DescTools)  #  Descriptive statistics library.
library(ggplot2)

data <- read.csv("data.csv")

set.seed(2)  #  Setting random seed for random sample reproducibility

# Gathering training indices: 67% for training/cross-val, the rest for test.
train_indices <- sample(1:nrow(data), nrow(data) * 0.67, replace = FALSE)

# Now we'll partition the data as per the test design/plan.
train <- data[train_indices,]
test <- data[-train_indices,]

################ END OF DO NOT EDIT THIS CODE, JUST RUN IT
############################################################

#### Enter your code below and then step through it to work towards completing the assignment.


################## Data Understanding Code #####################
Str(data)     # Display the structure of the data
View(data)
summary(data)   # Summary statistics of the data
dim(data)       # Dimensions of the data (rows and columns)
nrow(data)      # Number of rows in the data
ncol(data)      # Number of columns in the data


## DescTools::
# We've used Desc() to gather descriptive statistics and visualisations of data before.
Desc(data)

# Calculate the mean for each variable
mean_values <- sapply(data, mean, na.rm = TRUE)
print(mean_values)

# Calculate the median for each variable
median_values <- sapply(data, median, na.rm = TRUE)
print(median_values)

# Function to calculate mode
calculate_mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

# Calculate the mode for each variable
mode_values <- sapply(data, calculate_mode)
print(mode_values)
```

```
# Calculate the range for each variable
range_values <- sapply(data, function(x) max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
print(range_values)

# Calculate the variance for each variable
variance_values <- sapply(data, var, na.rm = TRUE)
print(variance_values)

# Calculate the standard deviation for each variable
std_dev_values <- sapply(data, sd, na.rm = TRUE)
print(std_dev_values)

# Count missing values for each variable
missing_values <- colSums(is.na(data))
print(missing_values)

# Identify rows with missing values
rows_with_missing <- which(rowSums(is.na(data)) > 0)
print(rows_with_missing)

# View the row with missing values
print(data[369, ])

# Remove the row with missing values
data <- data[-369, ]

### Correlation###
# Exploring associations using Pearson's correlation statistic
# is important if we might wish to explore the data using
# linear regression. cor(data) below facilitates calculations
# of correlation matrices based on input matrices or data
# frames.
train.corr <- cor(train)
print(train.corr)

ggcorrplot::ggcorrplot(train.corr)  #  Correlation heat map

# Faceted scatter plots of Total Floor Area (V.2) and Preliminary Estimated Construction Cost (V.4)
by Project Locality (V.1)
ggplot(data, aes(x = V.2, y = V.4)) +
  geom_point() +
  facet_wrap(~V.1) +
  labs(title = "Scatter Plots of V.2 vs V.4 by Project Locality", x = "Total Floor Area (m^2)", y =
"Preliminary Estimated Construction Cost (IRR)")

# Correlation matrix
correlation_matrix <- cor(data[, c("V.1","V.2", "V.3", "V.4", "V.5", "V.6", "V.7", "V.8", "Y")], use =
"complete.obs")
print(correlation_matrix)

# Identify potential multicollinearity (absolute correlation > 0.7)
col_to_check <- which(abs(correlation_matrix) > 0.7 & correlation_matrix != 1, arr.ind = TRUE)
if (any(col_to_check)) {
  print(paste0("High correlation detected between: ", rownames(correlation_matrix)[col_to_check[,
1]], " and ",
            rownames(correlation_matrix)[col_to_check[, 2]]))
  # Further investigation needed to decide if variable removal or other techniques are necessary
} else {
  print("No high correlations (> 0.7) detected among relevant variables.")
}

#################### Data Preparation Code #####################

library(tidyverse)  #  Streamlined data manipulation, visualization and analysis
library(caret)      #  Building, training, evaluating and tuning models

head(train, 4)
tail(train, 4)
class(train) # show the data type
str(train)  # Checking the structure/format of the data.
summary(train)

train <- mutate_at(train, vars(V.1, V.5, V.7, V.8, Y), as.numeric) # convert total to numeric variable
str(train)                              # let`s check the data structure again


is.na(train)                                   # classic way to check NA`s
sum(is.na(train))                                # counting NA`s
apply(is.na(train),2, which)                       # which indexes of NA`s (df only)
which(complete.cases(train))                         # identify observed complete values

train <- na.omit(train)

clean.vector <- na.omit(list(train))                # clean/remove a vector NA`s
clean.df <- na.omit(train)                        # clean/remove a dataframe NA`s
apply(is.na(clean.df),2, which)                    # make sure if there are missing values

any(is.na(clean.vector))

any(is.na(clean.df))

train %>% pull() %>% head()                # extract column values of `state` as a vector
print(train)

#Histogram of a numerical variable
hist(train$V.2,
    main = "Histogram of V.2 (Total Floor Area)",
    xlab = "Total Floor Area (m^2)", ylab = "Frequency",
    col = "skyblue", border = "black")
abline(v = mean(train$V.2),
      col = "red",
      lwd = 2) # Add a vertical line for the mean
legend("topright",
      legend = paste("Mean:",
                round(mean(train$V.2), 2)),
      col = "red",
      lwd = 2) # Add a legend for the mean
```

```
#Boxplot of a numerical variable to identify outliers
boxplot(train$V.4, train$V.2,
        main = "Boxplot of Preliminary Estimated Construction Cost (IRR)",
        xlab = "Variable V.4",
        ylab = "Construction Cost (IRR)",
        col = "skyblue",     # Customize colors
        border = "black",
        notch = TRUE,        # Add a notch
        pch = 19,            # Adjust outlier symbol
        horizontal = FALSE, # Horizontal orientation
        grid = TRUE          # Add grid lines
)
boxplot(train$V.2,
        main = "Boxplot of Preliminary Estimated Construction Cost (IRR)",
        xlab = "Variable V.4",
        ylab = "Construction Cost (IRR)",
        col = "skyblue",     # Customize colors
        border = "black",
        notch = TRUE,        # Add a notch
        pch = 19,            # Adjust outlier symbol
        horizontal = FALSE, # Horizontal orientation
        grid = TRUE          # Add grid lines
)
# scatter plot
plot(train$V.2, train$V.4,
     main = "Scatter Plot: Total Floor Area vs Preliminary Estimated Construction Cost",
     xlab = "Total Floor Area (m^2)",
     ylab = "Preliminary Estimated Construction Cost (IRR)",
     col = "blue",
     pch = 19)

#bar plot
barplot(table(train$V.1),
        main = "Bar Plot of Project Locality",
        xlab = "Project Locality",
        ylab = "Frequency",
        col = "skyblue")

# Time series plot of Actual Construction Costs (Y) over time (V.7)
plot(train$V.7, train$Y,
     type = "l",
     main = "Line Plot: Duration of Construction vs Actual Construction Costs",
     xlab = "Duration of Construction (Months)",
     ylab = "Actual Construction Costs (IRR)",
     col = "red")

# heatmap
heatmap(cor(train[, c("V.1","V.2", "V.3", "V.4", "V.5", "V.6", "V.7", "V.8", "Y")]),
        main = "Correlation Heatmap",
        xlab = "Variables",
        ylab = "Variables")

# Example density plot
plot(density(train$V.8),
     main = "Density Plot of Price per Unit Area",
    xlab = "Price per Unit Area (IRR)",
    col = "blue")

pairs(train[, c("V.2", "V.3", "V.4", "V.5", "V.6", "V.7", "V.8", "Y")])

heatmap(cor(train[, c("V.2", "V.3", "V.4", "Y")]),
        main = "Correlation Heatmap (Subset of Variables)",
        xlab = "Variables",
        ylab = "Variables")

#### Handling of the outliers########
#by examining each variable by boxplot, I have found outliers in 2 variables i.e. V.2 and V.4

# Boxplot before removing outliers
par(mfrow=c(1, 2))
boxplot(data$V.2, main="Boxplot of V.2 (Before)")
boxplot(data$V.4, main="Boxplot of V.4 (Before)")

# Identify outliers for V.2 and V.4 variables
outliers_V2 <- boxplot.stats(data$V.2)$out
outliers_V4 <- boxplot.stats(data$V.4)$out

# Remove outliers from the dataset
cleaned_data <- data[!(data$V.2 %in% outliers_V2 | data$V.4 %in% outliers_V4), ]

# Boxplot after removing outliers
par(mfrow=c(1, 2))
boxplot(cleaned_data$V.2, main="Boxplot of V.2 (After)")
boxplot(cleaned_data$V.4, main="Boxplot of V.4 (After)")

# Summary of removed outliers
cat("Outliers removed from V.2:", outliers_V2, "\n")
cat("Outliers removed from V.4:", outliers_V4, "\n")

# Remove rows with NA values
cleaned_data <- na.omit(cleaned_data)

# Summary of cleaned dataset
cat("Summary of cleaned dataset:\n")
summary(cleaned_data)

### IQR Method - to identify & handle outliers ###

# The IQR is used to identify and deal with outliers. It is a measure of
# the spread of the data values. It is a reliable measure of
# dispersion because it is not affected by extreme values of outliers.
#
# In the IQR method a range is defined by using first and third quartile and a
# multiplier which is usually set as 1.5. All the values below the lower
# limit and above the upper limit are considered as outliers.

# Calculate IQR for each variable
Q1 <- apply(train, 2, quantile, probs = 0.25)
Q3 <- apply(train, 2, quantile, probs = 0.75)
IQR <- Q3 - Q1
```

```r
outliers <- apply(train, 2, function(x) x < (Q1 - 1.5 * IQR) | x > (Q3 + 1.5 * IQR))

cleaned_data <- train
cleaned_data[outliers] <- NA
cleaned_data <- na.omit(cleaned_data)

summary(cleaned_data)

#################  Modelling Code ##################

# Create Predictor matrix & Response vector for both train & test set
x_train <- as.matrix(train[ , -1])   # predictor matrix for the training set.
y_train <- train[ , 1]            # response vector for the training set.

x_test <- as.matrix(test[ , -1])   # predictor matrix for the test set.
y_test <- test[ , 1]              # response vector for the test set.

# Excluding variables V.1 and V.3 from the model and adding predictor matrix
# in the training & test set

x_train_excl_V3 <- as.matrix(x_train[ , -3])
View(x_train_excl_V3)      # checking the correct variable has been removed
x_test_excl_V3 <- x_test[ , -3]
View(x_test_excl_V3)      # checking the correct variable has been removed

# a matrix is a two-dimensionsal collection of elements of the same data type
# (numeric, character, or logical) arranged into a fixed number of rows and columns.

## Standardizing the predictor variables ##

# Standardizing the predictor variables to have mean
# zero and unit variance. This is important for ridge regression
# because it ensures that the penalty term is applied equally to
# all the coefficients. Using scale function to do this.

x_train <- scale(x_train)  # standardize the training predictors
x_test <- scale(x_test)    # standardize the test predictors

x_train_excl_V3 <- scale(x_train_excl_V3)


##### Build Training Model - Cross Validation using Ridge Regression ####

# To perform ridge regression, I'll use functions from the glmnet package.

# I'll use the glmnet() function to fit the ridge regression model
# and specify alpha=0 to select Ridge Regression

# Setting alpha equal to 1 is equivalent to using Lasso Regression and
# setting alpha to some value between 0 and 1 is equivalent to using an elastic net.

# I'll use the default values of alpha and lambda and
# let the function choose the optimal values for us.
# Alpha = 0 corresponds to ridge regression,
# Alpha = 1 corresponds to lasso regression and
# 0 < alpha < 1 corresponds to elastic net regression,
# a combination of ridge and lasso.

# Ideally, producing multiple models and based on the lowest mean squared
# error and decide on the final model for implementation.

## Fitting the ridge regression model ##

# Model 1 for ridge regression
ridge_model <- glmnet(x_train, y_train, alpha = 0, standardize = FALSE)

# Model 2 for Lasso regression
lasso_model <- glmnet(x_train, y_train, alpha = 1, standardize = FALSE)

# Model 3 removing variable 3
ridge_model_excl_V3 <- glmnet(x_train_excl_V3, y_train, alpha = 0, standardize = FALSE)

# (The predictor value/ input features will need to be amended to exclude V3 in this example)

# Model 4 - Ridge Regression with lambda_min
ridge_model_lambda_min <- glmnet(x_train, y_train, alpha=0, lambda = ridge_model$lambda.min,
standardize = FALSE)

# Model 5 Lasso regression with lambda_min
lasso_model_lambda.min <- glmnet(x_train, y_train, alpha = 1, lambda =
lasso_model$lambda.min, standardize = FALSE)

# Note that by default, the glmnet() function standardizes the
# variables so that they are on the same scale. To turn off this default setting,
# use the argument standardize=FALSE.

# View summary model which will show  the length, class, mode, and dimensions of the elements.
summary(ridge_model)
summary(ridge_model_lambda_min)
summary(lasso_model_lambda.min)

# The corresponding values of lambda, beta, df, dev. ratio, and a0.
# The beta element is a sparse matrix, which means that it only stores
# the non-zero values of the coefficients.
# The df element is the degrees of freedom, the number of non-zero coefficients.
# The dev.ratio element is the fraction of deviance the model explains.
# The a0 element is the intercept term.

# Associated with each value of  λ is a vector of ridge regression coefficients,
# stored in a matrix that can be accessed by coef()
# coef(model)

# To check the coefficients using the dim()
dim(coef(ridge_model))

ridge_model

# Associated with each value of λ is a vector of ridge regression coefficients,
# stored in a matrix that can be accessed by coef(). In this case, it is a
# 5X100 matrix, with 5 rows (one for each predictor, plus an intercept) and
# 100 columns (one for each value of lambda).

ridge_model$lambda[20]

coef(ridge_model) [ , 20]

# Plotting the model object using plot function, shows coef change as a func of lambda

plot(ridge_model, xvar = "lambda", label = TRUE)# plot the coefficients vs lambda

# The x-axis is on a log scale (Log Lambda), so the smaller lambda
# values are on the right, and the larger values are on the left.

# The y-axis shows the values of the coefficients, and each line
# corresponds to a different predictor variable.

# In this example, when lambda log is 8, the coefficients are essentially zero.
# When we relax lambda the coefficients grow away from zero in a smooth way.
# The sum of squares of the coefficients are getting bigger and bigger until
# we reach a point where Lambda is effectively zero & the coefficients
# are regularized & so these would be the coefficients  that you get from an
# ordinary least squares fit of these variables.

### Perform k-fold cross-validation to find optimal lambda value ###

# Next, we'll identify the lambda value that produces the
# lowest test mean squared error (MSE) by using k-fold
# cross-validation a technique that splits the data into
# several subsets and uses some for training and some for testing.

# glmnet has the function cv.glmnet() that automatically
# performs k-fold cross validation. A fold is a subset of the
# data used for testing, while the rest is used for training.
# The default value is 10, meaning the data is split into 10 subsets,
# and each subset is used as a test set once.

# Model 1 - Ridge Regression cross validation
cv_ridge_model <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 5)
# Performs 5 fold cross validation on ridge model

# Model 2 - Lasso Regression cross validation
cv_lasso_model <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 5)
# Performs 5 fold cross validation on lasso model

# Model 3 - Ridge Regression with variable 3 excluded
cv_ridge_model_excl_V3 <- cv.glmnet(x_train_excl_V3, y_train, alpha = 1, nfolds = 5)

# Model 4 - Ridge Regression with lambda.min
# ridge_model_lambda_min <- glmnet(x_train, y_train, alpha=0, lambda =
cv_ridge_model$lambda.min)
cv_ridge_model_lambda_min <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 5)

# Model 5 - Lasso Regression with lambda.min
```

```
cv_lasso_model_lambda_min <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 5)

summary(cv_ridge_model)
summary(ridge_model_lambda_min)

# The cvm element is the mean cross-validated error for each value of lambda.

# The cvsd element is the standard deviation of the cross-validated error for
# each lambda value.

# The cvup and cvlo elements are the upper and lower confidence bounds
# for the cross-validated error for each lambda value.

# The nzero element is the number of non-zero coefficients for each value of lambda.

# The lambda.min element is the value of lambda that gives the minimum
# cross-validated error.

# The lambda.1se element is the largest value of lambda, giving a
# cross-validated error within one standard error of the minimum.

## plotting the cv_model object using the plot function ##

# Shows the cross-validated error changes as a function of lambda.

# The x-axis is on a log scale, so the smaller lambda values are on
# the right, and the larger values are on the left.

# The y-axis shows the values of the cross-validated error, and the
# error bars show the confidence bounds.

# The vertical dotted lines indicate the values of lambda that give
# the minimum cross-validated error and the largest error within
# one standard error of the minimum.

plot(cv_ridge_model) # plot the cross-validated error vs lambda


# This is a plot of the cross validated MSE and from the right hand side
# it dips downs. In the beginning the MSE is very high and the coefficients
# are restricted to be too small and then it starts to level off.
# This indicates that the full model is doing a good job.

# There are two vertical lines:
# The first one indicates the min. MSE.
# The second indicates the one standard error of the min. MSE.
# This is a more restricted model that can do as well as the min. MSE and
# we can decide to use this value instead of the min. MSE.

plot(cv_lasso_model)  # Model 2 Lasso Regression
plot(cv_ridge_model_excl_V3)  # Model 3 Ridge Regression with V3 removed
plot(cv_ridge_model_lambda_min) # Model 4 Ridge regression with lambda min.
plot(cv_lasso_model_lambda_min) # Model 5 Lasso regression with lambda min.

### This will show the results of the best results from each model. ###
```

```
cv_ridge_model
cv_lasso_model
cv_ridge_model_excl_V3
cv_ridge_model_lambda_min
cv_lasso_model_lambda_min

# This will pick the coefficient corresponding to the best model.
coef(cv_ridge_model)
coef(cv_lasso_model_lambda_min)
coef(ridge_model_lambda_min)

#### Let's store the validation results! They will be useful to compare against the test set results.
# Model 1 Ridge model MSE & RMSE
cross_validation_ridge_MSE <- min(cv_ridge_model$cvm)
cross_validation_ridge_RMSE <- sqrt(cross_validation_ridge_MSE)

# Model 2 Lasso model MSE & RMSE
cross_validation_lasso_MSE <- min(cv_lasso_model$cvm)
cross_validation_lasso_RMSE <- sqrt(cross_validation_lasso_MSE)

# Model 3 Ridge model excluding Var. 3 MSE & RMSE
cross_validation_ridge_excl_V3_MSE <- min(cv_ridge_model_excl_V3$cvm)
cross_validation_ridge_excl_V3_RMSE <- sqrt(cross_validation_ridge_excl_V3_MSE)

# Model 4 Ridge model with lambda min. MSE & RMSE
cross_validation_ridge_lambda_min <- min(cv_ridge_model_lambda_min$cvm)
cross_validation_ridge_lambda_min_RMSE <- sqrt(cross_validation_ridge_lambda_min)    # The
square root of the MSE of the ridge regression model excl V3

# Model 5 Lasso model with lambda min.
cross_validation_lasso_lambda_min <- min(cv_lasso_model_lambda_min$cvm)
cross_validation_lasso_lambda_min_RMSE <- sqrt(cross_validation_lasso_lambda_min)    # The
square root of the MSE of the ridge regression model excl V3

# Create a data frame to store the MSE and RMSE values
model_comparison <- data.frame(
  Model = c("Ridge", "Lasso", "Ridge Excl. V3", "Ridge Lambda Min", "Lasso Lambda Min"),
  MSE = c(cross_validation_ridge_MSE, cross_validation_lasso_MSE,
cross_validation_ridge_excl_V3_MSE, cross_validation_ridge_lambda_min,
cross_validation_lasso_lambda_min),
  RMSE = c(cross_validation_ridge_RMSE, cross_validation_lasso_RMSE,
cross_validation_ridge_excl_V3_RMSE, cross_validation_ridge_lambda_min_RMSE,
cross_validation_lasso_lambda_min_RMSE)
)

# Print the data frame
print(model_comparison)

#  min() can be used to get the smallest MSE from the evaluated lambda values.
#  The square root of the MSE of the ridge regression model

### Find optimal lambda value that minimizes test MSE ###

# From the above results we can see the Ridge Lamda Min has produced the lowest MSE.
#        Model        MSE              RMSE
# 1           Ridge  30.13858        5.489862
# 2           Lasso  22.94890        4.790501
# 3   Ridge Excl. V3  29.86710        5.465080
# 4 Ridge Lambda Min  20.59304        4.537955
# 5 Lasso Lambda Min  20.91619        4.573422

# I've selected the optimal value of lambda, it's time for me
# to assess the performance of the ridge regression model on the test set.

# I'll utilize the predict function to generate predictions of the response variable f
# or the test set, employing the ridge regression model with the chosen lambda value.

# Next, I'll compare these predicted values with the actual values and compute
# various metrics to gauge the accuracy of the predictions,
# including Mean Squared Error (MSE),
# Root Mean Squared Error (RMSE), and the
# coefficient of determination (R-squared).
# Using the predict function,

# I'll generate the predicted values for the test set using prediction func;

### To determine the optimal lambda value that minimizes cross-validated error, element.
# Lower values of lambda indicate stronger regularization,
# while higher values indicate weaker regularization
cv_ridge_model_lambda_min$lambda.min

####################  Final Evaluation ####################

test <- na.omit(test)   # Remove NA value from test set
anyNA(test)  #  If there are any missing values in the test set they will need to be handled.

# The test set predictor matrix has already been created earlier.
x_test <- as.matrix(test[ , -1])   # predictor matrix for the test set.
y_test <- test[ , 1]
x_test <- scale(x_test)


is.infinite(x_test)
is.infinite(y_test)

head(x_test)
head(y_test)

# Check if the test set is scaled similarly to the training set
summary(x_test)

# Make test set predictions.
predictions <- predict(ridge_model_lambda_min, s = ridge_model_lambda_min$lambda, newx =
x_test)

summary(predictions)    # Verify the prediction process

# Finally, let's evaluate how well (or poorly) we have done on the test set.
```

```r
test_MSE <- MSE(predictions, y_test)
test_RMSE <- RMSE(predictions, y_test)

test_MSE
test_RMSE

# Extracting the best lambda by indexing the glmnet lambda component
# then index it by order of RMSE, order puts them in ascending order
# smallest value and this will pick out the best lambda.

lam.best <- ridge_model_lambda_min$lambda[order(test_MSE)[1]]

lam.best <- min(ridge_model_lambda_min$lambda)
lam.best

coef(ridge_model_lambda_min, s = lam.best)

############## Plot of test set predictions vs actual values ######################

# Pad predictions vector with zeros if its length is less than y_test
if (length(predictions) < length(y_test)) {
  extra_zeros <- rep(0, length(y_test) - length(predictions))
  predictions <- c(predictions, extra_zeros)
} else if (length(predictions) > length(y_test)) {
  predictions <- predictions[1:length(y_test)]
}

# Check if the lengths are the same
length(predictions)
length(y_test)

# If they are not the same, you can adjust one of the vectors to match the length of the other.
# For example, you can trim or pad one of the vectors to match the length of the other.
# But in this case its same

# Ensuring both vectors have the same length
min_length <- min(length(predictions), length(y_test))
predictions <- predictions[1:min_length]
y_test <- y_test[1:min_length]

# Now plot the data
plot(x = predictions, y = y_test, frame = FALSE, pch = 19,
     col = "red", xlab = "Predicted Values", ylab = "Actual Values")

######Dual line chart for predicted vs. actual values########
test_instances <- seq_along(y_test)

plot(x = test_instances, y = y_test, frame = FALSE, pch = 19, type = "l",
     col = "red", xlab = "Test Instance", ylab = "Valence")

lines(x = test_instances, y = predictions, pch = 18, col = "blue", type = "l", lty = 2)

# Adding legend
legend("topleft", legend=c("Actual", "Predicted"), col=c("red", "blue"), lty = 1:2, cex=0.8)
```

```
###### verify if the model predictions fall within  +/- 500,000 Iranian Rial#############

# Calculate the absolute difference between predictions and actual values
abs_diff <- abs(predictions - y_test)

# Check if the absolute difference is within the specified threshold
within_threshold <- abs_diff <= 500000

# Count the number of predictions within the threshold
num_within_threshold <- sum(within_threshold)

# Calculate the percentage of predictions within the threshold
percentage_within_threshold <- (num_within_threshold / length(y_test)) * 100

print(percentage_within_threshold)

# Print the results
cat("Number of predictions within +/- 500,000 Iranian Rial threshold:", num_within_threshold, "\n")
cat("Percentage of predictions within +/- 500,000 Iranian Rial threshold:",
percentage_within_threshold, "%\n")

#####ploting the visual for it
# Convert the logical vector to a factor with labels "Within Threshold" and "Outside Threshold"
within_threshold_factor <- factor(within_threshold, levels = c(FALSE, TRUE), labels = c("Outside
Threshold", "Within Threshold"))

# Create a bar plot to visualize the proportion of predictions within the threshold
bar_colors <- c("red", "green")
bar_names <- c("Outside Threshold", "Within Threshold")
barplot(table(within_threshold_factor), col = bar_colors, main = "Predictions Within +/- 500,000
Iranian Rial threshold", ylab = "Frequency")

# Add a legend to the plot
legend("topleft", legend = bar_names, fill = bar_colors, cex = 0.8)
```