EAS508: Statistical learning and Data Mining I

Project_1

Team members:

- Sri Guna Kaushik Undru (srigunak)
- Dominic Parosh Yamarthi (dyamarth)
- Shri Harsha Adapala Thirumala (sadapala)
- Sushmitha Jakka (sjakka)

Q.1) You will use Health data set to hone your regression skills. The data set along with description of the variables has been uploaded to UB Learns under the title "project "toy" data set".

The response variable is given in X1.

Description of the variables is also given here:

X1	death rate per 1000 residents
X2	doctor availability per 100,000 residents
X3	hospital availability per 100,000 residents
X4	annual per capita income in thousands of dollars
X5	population density people per square mile

Solution:

Regression:

Regression is a statistical technique which is used to simulate the relationship between two or more variables. Finding the mathematical formula that best captures the link between the dependent variable, which is the variable being predicted, and one or more independent variables, which are the variables utilized to make the prediction, requires examining data.

Regression analysis aims to fit a regression model to the data so that it can be utilized for inference, prediction, or understanding of the connection between variables. A linear equation, which depicts a straight-line relationship between the variables, or a nonlinear equation, which indicates a curved relationship between the variables, are two types of equations that are frequently used to represent regression models.

In addition to forecasting future results, regression analysis can also be used to analyse the strength and direction of correlations between variables, identify key predictors, and assess the effects of interventions or treatments on outcomes. Numerous disciplines, including statistics, economics, the social sciences, business, finance, medicine, and machine learning, among others, frequently use regression.

For this problem we set the seed to 1 and call the “readxl” library so that we can read xlsx files and read the data set and store it into “health_df” and we print top data to get an idea of how the data set is organized by using “head(health_df)”.

```
set.seed(1)
library(readxl)
health_df = read_excel("data/Health.xlsx")
head(health_df)
```

Output :

```
X1    X2    X3    X4    X5
  <dbl> <dbl> <dbl> <dbl> <dbl>
1  8       78   284  9.10   109
2  9.30    68   433  8.70   144
3  7.5     70   739  7.20   113
4  8.90    96  1792  8.90    97
5 10.2     74   477  8.30   206
6  8.30   111   362 10.9    124
```

We find the dimensions of the given data set to find number of rows and number or columns.

```
cat("Dimesions of the data:",dim(health_df))
```

Output:

```
Dimesions of the data: 53 5
```

We, now check if there are any null or N/A values present in the given data set.

```
paste("Number of NA Values: ",sum(is.na(health_df)))
```

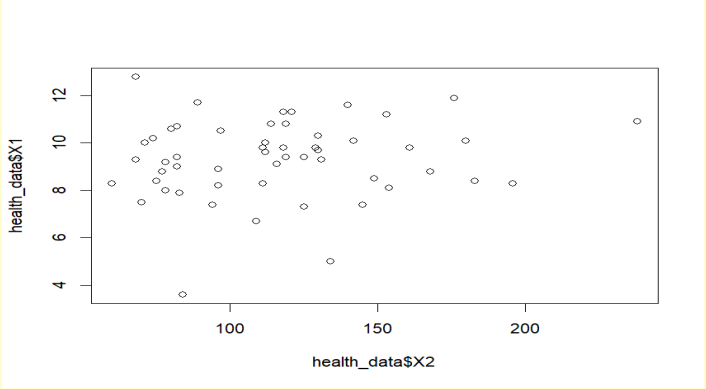
Output:

```
[1] "Number of NA Values:  0"
```

This implies that there is no null or N/A values in the data set.

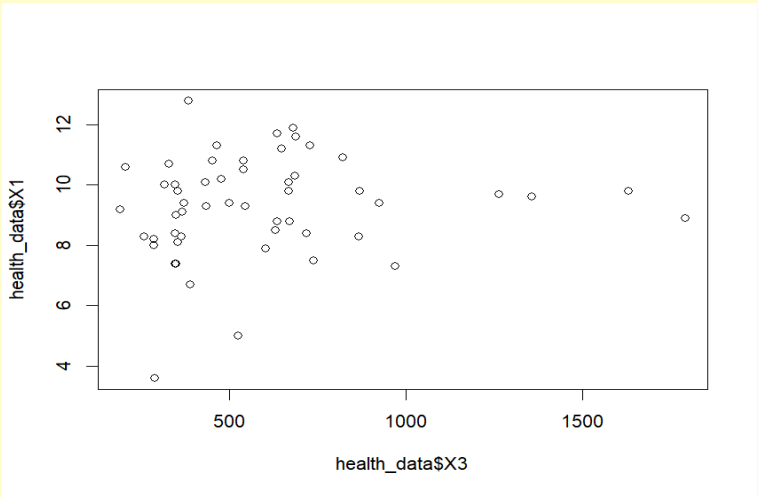
We will now plot the graph for X2 and X1 which are the columns in the data set.

```
plot(x = health_df$X2, y=health_df$X1)
```



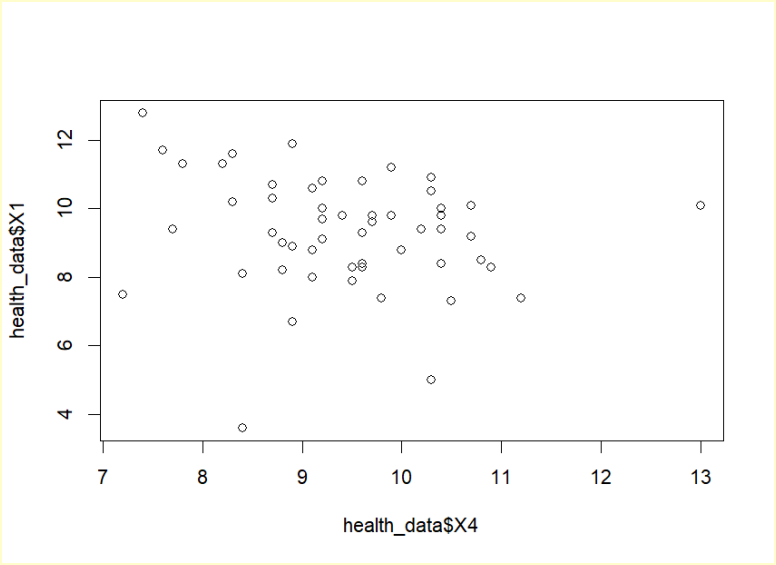
We will now plot the graph for X3 and X1 which are the columns in the data set

```
plot(x = health_df$X3, y=health_df$X1)
```



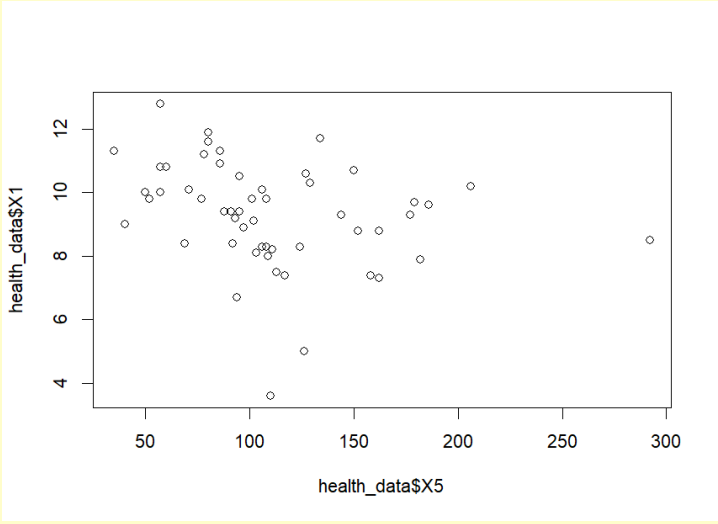
We will now plot the graph for X4 and X1 which are the columns in the data set.

```
plot(x = health_df$X4, y=health_df$X1)
```



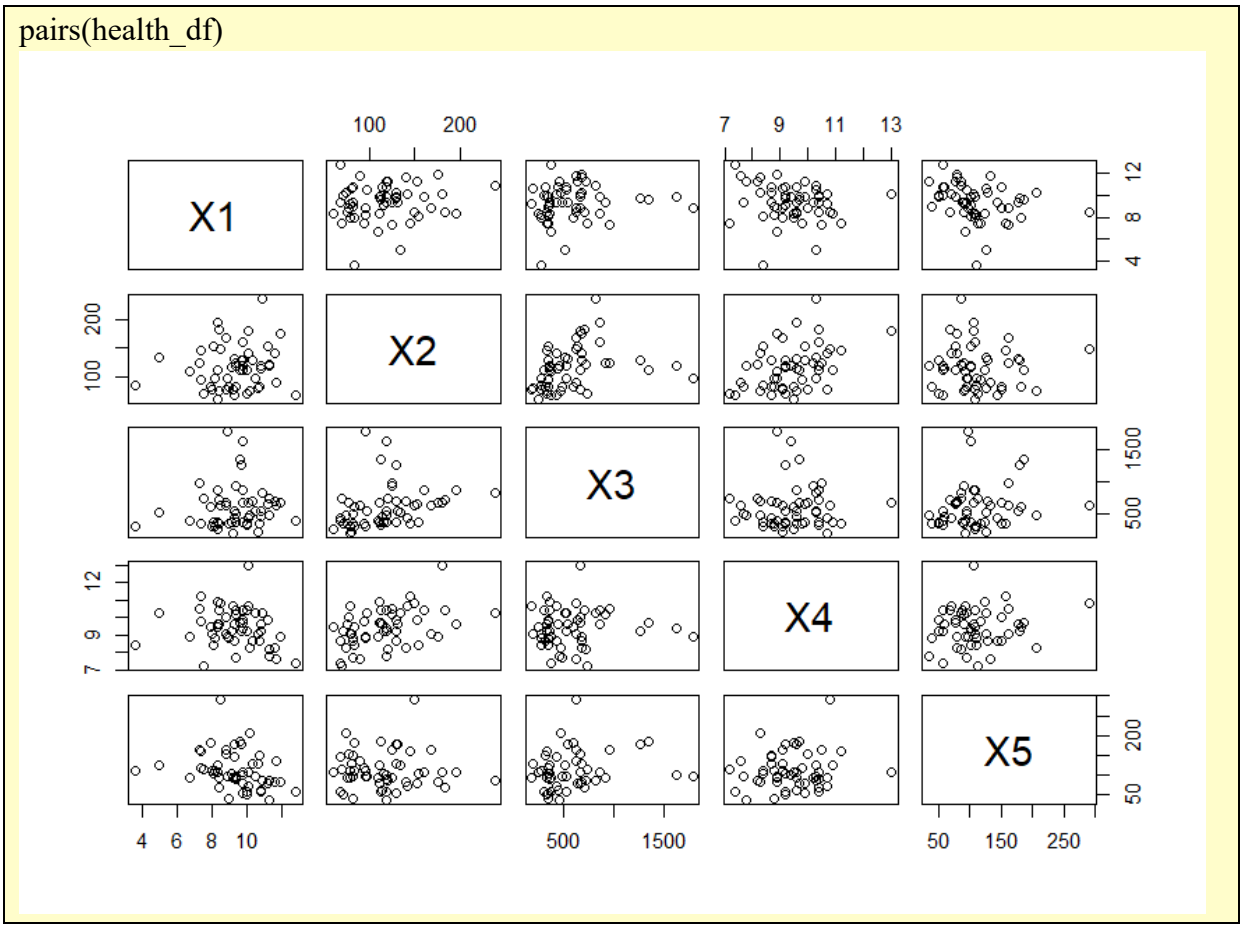
We will now plot the graph for X5 and X1 which are the columns in the data set.

```
plot(x = health_df$X5, y=health_df$X1)
```



We use pairs function to get a matrix of scatter plots.

```
pairs(health_df)
```



1) Linear Regression:

A statistical technique called linear regression is used to represent the relationship between two variables, where one of the variables is the dependent variable and the other is the independent variable. Finding the regression line or line of best fit that minimizes the residual sum of squares (RSS), the sum of the squared differences between the observed values of the dependent variable and the predicted values by the regression line, is the objective of linear regression.

A linear regression model's equation appears as follows:
$Y = \beta 0 + \beta 1 * X + \varepsilon$
where:

- Y is the dependent variable, also known as the response variable or the target variable.
- X is the independent variable, also known as the predictor variable or the feature variable.
- $\beta 0$ is the intercept term, which represents the predicted value of Y when X is equal to 0.

- β1 is the slope coefficient, which represents the change in Y for a unit change in X.
- ε is the error term or the residual, which represents the unexplained variation in Y that is not accounted for by the linear relationship with X.

The following are the main steps in performing linear regression:
Data collection and preparation: This includes acquiring data for the dependent and independent variables, as well as cleaning and preparing the data for analysis. Data plotting is constructing a scatter plot to depict the relationship between the dependent and independent variables.

Estimating the parameters entails estimating the values of 0 and 1 that best fit the data using a statistical method such as ordinary least squares (OLS).
Model evaluation: This includes evaluating the model's goodness of fit using statistical metrics such as R-squared, adjusted R-squared, and p-values, as well as graphical diagnostics such as residual plots.

Interpreting the data entails deriving judgments about the strength and direction of the link between the variables based on the estimated parameters.
Making forecasts: Once fitted and validated, the model can be used to forecast the dependent variable for new values of the independent variable.
Linear regression is a popular and straightforward method for estimating the relationship between two variables, with numerous applications in economics, finance, social sciences, healthcare, and engineering.

We perform linear regression on the given model by setting the seed to 1 and importing the library “boot”. For this data set we take X1 as the dependent variable and perform linear regression against X2, X3, X4 and X5. To estimate test error, we use K-fold cross-validation.

K-fold cross-validation (CV) is a technique used to evaluate the performance of a machine learning model by partitioning the data into k equally sized folds and iteratively training the model on k-1 folds while using the remaining one-fold as the validation set. This process is repeated k times, with each fold being used once as the validation set, and the average performance across all folds is computed to obtain an estimate of the model's performance. We generally choose k value to be 5.

```
set.seed(1)
library(boot)
lm_model = glm(X1 ~ X2 + X3 + X4 + X5, data = health_df)
lm_cv_model = cv.glm(data=health_df, lm_model, K = 5)
cv_error = lm_cv_model$delta[1]

print(paste("CV error:", cv_error))
```

Output:

```
[1] "CV error: 2.83255913388255"
```

After performing cross-validation the estimated test error for linearly regressed data is 2.83

We get the summary of the model we created by using “summary” function.

```
set.seed(1)
lm_model <- lm(X1 ~ X2 + X3 + X4 + X5, data = health_df)
summary(lm_model)
```

Output:

```
Call:
lm(formula = X1 ~ X2 + X3 + X4 + X5, data = health_df)

Residuals:
    Min      1Q  Median      3Q     Max
-5.6404 -0.7904  0.3053  0.9164  2.7906

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.2662552  2.0201467   6.072 1.95e-07 ***
X2           0.0073916  0.0069336   1.066   0.2917
X3           0.0005837  0.0007219   0.809   0.4228
X4          -0.3302302  0.2345518  -1.408   0.1656
X5          -0.0094629  0.0048868  -1.936   0.0587 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.601 on 48 degrees of freedom
Multiple R-squared:  0.1437,  Adjusted R-squared:  0.07235
F-statistic: 2.014 on 4 and 48 DF,  p-value: 0.1075
```

- The P - value of F - statistic signifies that no predictor has a significant effect on the response.
- The P-values of t-statistic says that only predictor X5 has effect on the response.

To find the confidence interval of linear regression model we use “confint”.

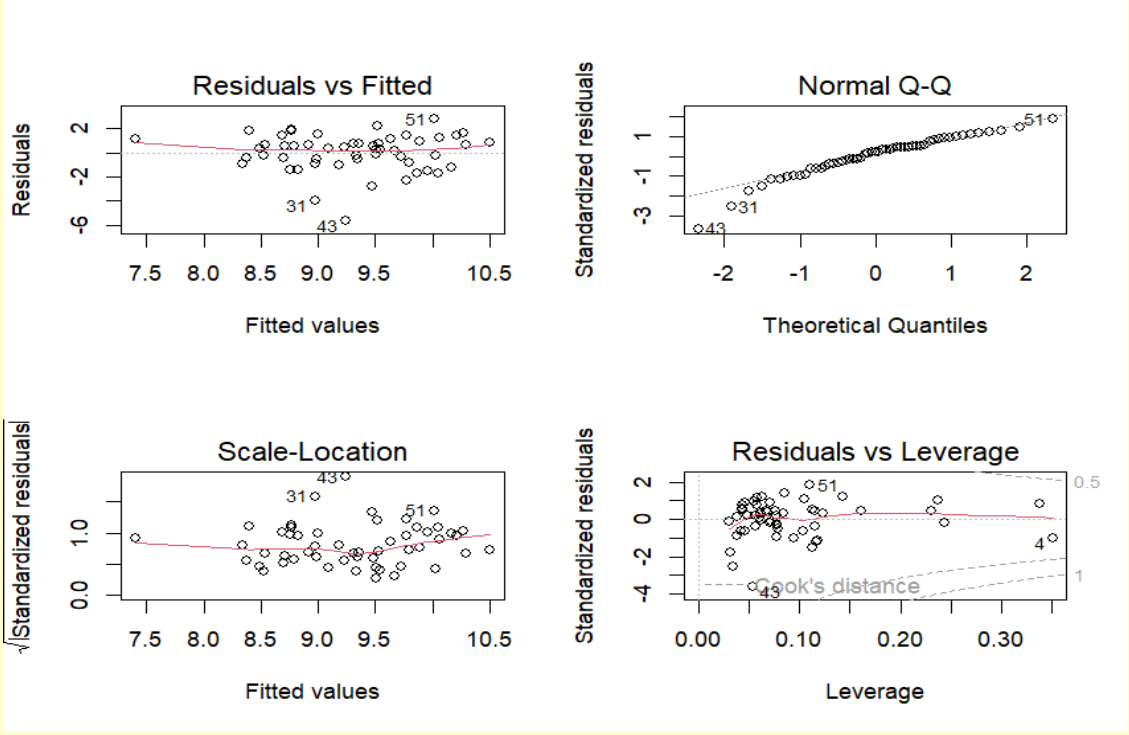
```
confint (lm_model)
```

Output:

```
                2.5 %       97.5 %
(Intercept)  8.2044779685 1.632803e+01
```

```
X2          -0.0065494163 2.133265e-02
X3          -0.0008677875 2.035219e-03
X4          -0.8018281444 1.413677e-01
X5          -0.0192884855 3.627158e-04
```

```
par(mfrow = c(2,2))
plot(lm_model)
```



Problems in the fit:

1. The Residual vs Fitted Values shows no significant non-linearity in the data.
2. The plot also says that there is constant variance in the errors, which shows that our assumption that errors ε has constant variance is right. This is further justified by the Fitted Values vs $\sqrt{\text{Standardized} - \text{Residuals}}$ plot.
3. The quantile-quantile plot showed errors deviating from the normality assumption at left end of the tail.
4. In the bottom-right plot we can see, some points have standardized residuals greater than 3, which makes them outliers.
5. The cut-off for the leverage point threshold is $\frac{2p}{n}$ where p is the number of predictors, which is one in this case, and n is the number of observations, which is 1503 in this case. Therefore, the cut-off is $\frac{(2)(3)}{51} = 0.151$. Hence, there are some leverage points in this dataset.
6. As some observations qualified as both outliers and leverage points, we can safely say we have some influential points.

Subset selection in linear regression is a technique used to select a subset of predictor variables from a larger set of variables to build a linear regression model. This is typically done to identify the most important variables that have a significant impact on the response variable, and to simplify the model by removing unnecessary variables.

Here we set the seed to 1 and load “leaps” library and perform best subset selection by using “regsubsets”.

```
set.seed(1)
# Load the leaps package
library(leaps)
```

Perform best subset selection using the regsubsets function

```
subset_model <- regsubsets(X1 ~ X2 + X3 + X4 + X5, data = health_df, nvmax = 4)

summary(subset_model)
```

Output:

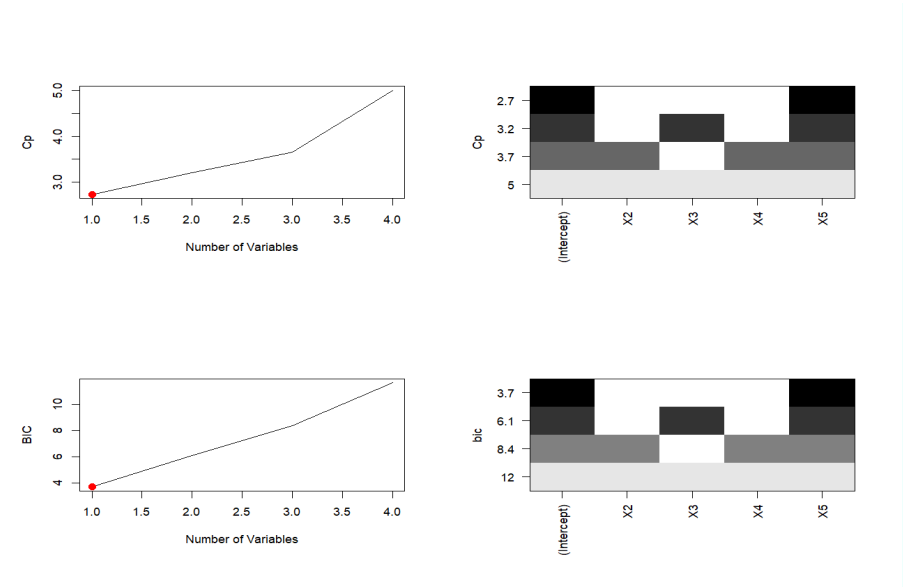
```
Subset selection object
Call: regsubsets.formula(X1 ~ X2 + X3 + X4 + X5, data = health_df,
   nvmax = 4)
4 Variables  (and intercept)
  Forced in Forced out
X2    FALSE     FALSE
X3    FALSE     FALSE
X4    FALSE     FALSE
X5    FALSE     FALSE
1 subsets of each size up to 4
Selection Algorithm: exhaustive
       X2  X3  X4  X5
1  ( 1 ) " " " " " " "*"
2  ( 1 ) " " "*" " " "*"
3  ( 1 ) "*" " " "*" "*"
4  ( 1 ) "*" "*" "*" "*"
```

Now draw plots for CP metric and BIC metric.

```
regfit_summary = summary(subset_model)

par(mfrow = c(2,2))

# Drawing plots for CP Metric
```

```
plot(regfit_summary$cp, xlab = "Number of Variables", ylab = "Cp",type = "l")
points(which.min(regfit_summary$cp), regfit_summary$cp[which.min(regfit_summary$cp)],
col = "red", cex = 2, pch = 20)
plot(subset_model, scale = "Cp")

# Drawing plots for BIC Metric
plot(regfit_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
points(which.min(regfit_summary$bic),
regfit_summary$bic[which.min(regfit_summary$bic)], col = "red", cex = 2, pch = 20)
plot(subset_model, scale = "bic")
```

Output:



- Plot shows that model with one predictor does better than the rest.
- And the subset selection method shows that one predictor is X5, this goes hand in hand with what we observed with t-test p-values.

Now creating a linear regression model with only the predictor we found to be best after sub-set selection.

```
best_subset_lm_model = glm(X1 ~ X5, data = health_df)
best_subset_lm_cv_model = cv.glm(data=health_df, best_subset_lm_model, K = 5)
cv_error = best_subset_lm_cv_model$delta[1]

print(paste("CV error:", cv_error))
```

Output:

```
[1] "CV error: 2.65318961211693"
```

On Performing Cross Validation, the estimated test error for best subset selected linear regression (with only one predictor) was: 2.65

2) Polynomial Regression:

Polynomial regression is a statistical method for modeling the connection between a dependent variable and one or more independent variables that is not linear but rather follows a polynomial curve. It is an extension of linear regression in which the relationship is represented as a straight line.

The link between the dependent variable and the independent variable(s) is treated as a polynomial function of a given degree in polynomial regression. The shape of the curve that is fitted to the data is determined by the degree of the polynomial. A polynomial of degree 2 produces a quadratic function

When there is evidence of a non-linear relationship between the variables, polynomial regression might be used. It provides greater flexibility in modelling complicated patterns in data that a basic linear regression cannot capture. Polynomial regression is widely used in statistics, machine learning, economics, and engineering to model real-world data with non-linear correlations.

The polynomial regression model is often estimated using the least squares method, which entails determining the polynomial coefficients that minimize the sum of squared residuals between the observed data points and the anticipated values from the model.

Polynomial regression has several benefits, such as the capacity to model non-linear relationships, but it also has significant drawbacks. Overfitting can be a problem, particularly when utilizing high-degree polynomials, and can lead to poor generalization performance. Furthermore, due to the non-linearity of the relationship between the variables, understanding the coefficients of a polynomial regression model might be more difficult than linear regression. To avoid these difficulties and assure the dependability of the polynomial regression model, proper model evaluation procedures such as cross-validation and model selection should be applied.

For our data set we perform polynomial regression by setting the seed to 1 and by using the for loop to get the cross-validation error value

```
set.seed(1)
options(warn = -1)
cv_error <- rep(0,10)
for (i in 1:10) {
  non_lin_reg <- glm(health_df$X1 ~ polym(health_df$X2, health_df$X3, health_df$X4,
health_df$X5, degree = i))
```

```
  cv_error[i] <- cv.glm(health_df, non_lin_reg, K = 5)$delta[1]
}

print(paste("Minimum CV error:", min(cv_error)))
print(paste("Best degree:", which.min(cv_error)))
```

Output:

```
[1] "Minimum CV error: 2.91484676876147"
[1] "Best degree: 1"
```

- On performing Cross Validation For different degrees of polynomial we found that 1st degree model has minimum test error.

- The test error was 2.91.

3) Ridge Regression

Ridge regression is a type of linear regression technique used for regression analysis, a statistical method used to model the relationship between a dependent variable and one or more independent variables. Ridge regression is an extension of ordinary least squares (OLS) regression in which a penalty term is added to the objective function to reduce model complexity and prevent overfitting.

The main idea behind ridge regression is to introduce a regularization term, often called a "ridge" or "L2" penalty, which adds a penalty to the regression coefficients based on their magnitudes. The regularization term is multiplied by a hyperparameter called the regularization strength, denoted as λ (lambda), which controls the trade-off between fitting the data and regularization. A larger value of λ results in stronger regularization, leading to smaller coefficients and a simpler model, while a smaller value of λ allows for larger coefficients and a more complex model.
Mathematically, the objective function of ridge regression can be represented as:
minimize: $||Y - X\beta||^2 + \lambda||\beta||^2$
where:

- Y is the vector of observed dependent variable values
- X is the matrix of independent variable values
- β is the vector of regression coefficients to be estimated
- λ is the regularization strength
- $||.||^2$ denotes the squared Euclidean norm, which is the sum of squared values

Ridge regression can be solved using various optimization techniques, such as closed-form solutions or iterative methods, and it has several advantages. It can handle multicollinearity (high correlation between independent variables) better than ordinary least squares regression, and it can provide more stable estimates of the regression coefficients. Ridge

regression is commonly used in situations where there are multiple correlated predictors, and the goal is to prevent overfitting and obtain a more robust and generalized model.

We develop a ridge regression model on our dataset by setting the seed to 1 and we set up a cross-validation grid and perform cross-validation using the “cv.glmnet” function. Select the best value of lambda.

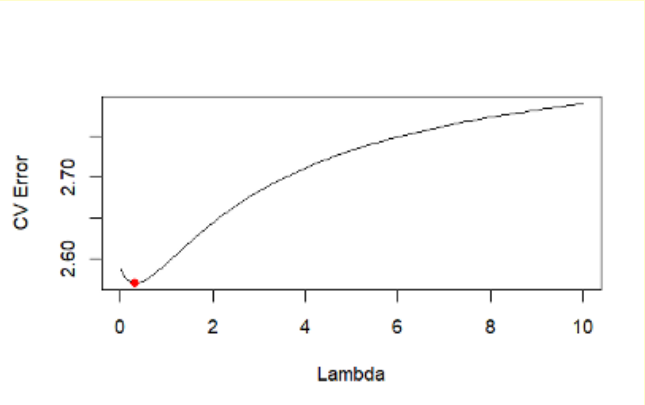
```
x = model.matrix(X1 ~ ., health_df)[, -1]
y = health_df$X1

library(glmnet)
set.seed(1)
# Set up the cross-validation grid
cv_grid = expand.grid(lambda = seq(from = 0.001, to = 10, length = 100))

# Perform cross-validation using the cv.glmnet function
rr_cv_model = cv.glmnet(x, y, alpha = 0, lambda = cv_grid$lambda, nfolds = 5)

# Select the best value of lambda
best_lambda = rr_cv_model$lambda.min

# Print the minimum cross-validation error
print(paste("Best Lambda Value:", best_lambda))
```

Output:

```
Loading required package: Matrix

Loaded glmnet 4.1-7

[1] "Best Lambda Value: 0.304"
```

```
cv_error <- rr_cv_model$cvm

# Plot the cross-validation error curve
plot(rr_cv_model$lambda, cv_error, type = "l", xlab = "Lambda", ylab = "CV Error")
```

```
points(best_lambda, min(cv_error), col = "red", pch = 19)
```



Now we get the minimum cross-validation error.

```
# Get the minimum cross-validation error
min_cv_error = min(rr_cv_model$cvm)

# Print the minimum cross-validation error
print(paste("Minimum CV error:", min_cv_error, 'for the lambda value of', best_lambda))
```

Output:

```
[1] "Minimum CV error: 2.57026825735826 for the lambda value of 0.304"
```

We got minimum CV error of 2.57 for the Lambda value of 0.304.

```
set.seed(1)
rr_best_model = glmnet(x, y, alpha = 0, lambda = best_lambda)
coef(rr_best_model)
```

Output:

```
5 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept) 11.7231398460
X2           0.0057530764
X3           0.0005035695
X4          -0.2636432444
X5          -0.0080863202
```

We can see that coefficient values shrinking towards zero.

4) Lasso Regression

Lasso regression, also known as L1 regularization, is a type of linear regression technique that incorporates regularization to improve the predictive accuracy and interpretability of the model. In lasso regression, a penalty term is added to the ordinary least squares (OLS) objective function, which encourages the model to use a smaller number of predictors or features in the final model.

The key idea of lasso regression is to introduce a penalty term based on the absolute values of the regression coefficients. The penalty term is proportional to the sum of the absolute values of the coefficients multiplied by a hyperparameter, often denoted as lambda (λ). The hyperparameter λ controls the strength of the regularization, with larger values of λ resulting in stronger regularization and smaller values of λ resulting in weaker regularization.

The lasso regression model seeks to minimize the following objective function:

OLS objective function + λ * (sum of absolute values of coefficients)

The regularization term in lasso regression has a unique property that it can set some of the regression coefficients to exactly zero. This allows for feature selection, where less important features can be automatically excluded from the model. This property makes lasso regression particularly useful when dealing with high-dimensional data, where there are many predictors and some of them may be irrelevant or redundant.

Lasso regression can be used for various purposes, including prediction, variable selection, and interpretation. It is commonly used in machine learning and statistics for tasks such as linear regression, logistic regression, and generalized linear models. Lasso regression is implemented in many statistical software packages and is widely used in practice for building predictive models with sparse or high-dimensional data.

To perform Lasso regression on the given data set by finding best lambda value

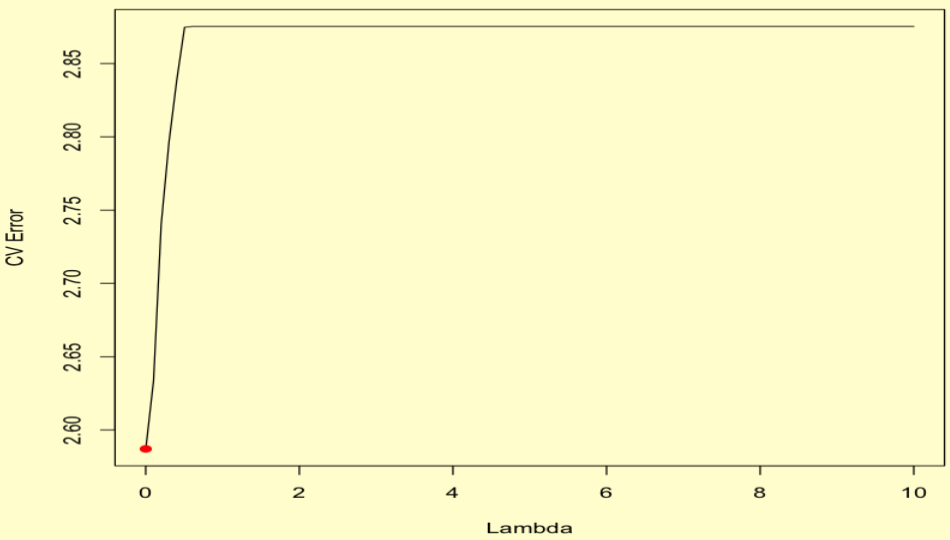
```
# Set up the cross-validation grid
cv_grid = expand.grid(lambda = seq(from = 0.001, to = 10, length = 100))

# Perform cross-validation using the cv.glmnet function
lasso_cv_model = cv.glmnet(x, y, alpha = 1, lambda = cv_grid$lambda, nfolds = 5)

# Select the best value of lambda
best_lambda = lasso_cv_model$lambda.min

# Print the minimum cross-validation error
print(paste("Best Lambda Value:", best_lambda))
```

Output:

```
[1] "Best Lambda Value: 0.001"
```

We plot the cross-validation error curve

```
cv_error <- lasso_cv_model$cvm

# Plot the cross-validation error curve
plot(lasso_cv_model$lambda, cv_error, type = "l", xlab = "Lambda", ylab = "CV Error")
points(best_lambda, min(cv_error), col = "red", pch = 19)
```



We now get minimum of cross-validation error.

```
# Get the minimum cross-validation error
min_cv_error = min(lasso_cv_model$cvm)

# Print the minimum cross-validation error
print(paste("Minimum CV error:", min_cv_error))
```

Output:

```
[1] "Minimum CV error: 2.58709373465739"
```

- We got minimum CV error of 2.587 for the Lambda value of 0.001.

We now find the coefficient values.

```
set.seed(1)
lasso_best_model = glmnet(x, y, alpha = 1, lambda = best_lambda)
coef(lasso_best_model)
```

Output:

```
5 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept) 12.2564960667
X2           0.0073550643
X3           0.0005812564
X4          -0.3288262318
X5          -0.0094429565
```

- We can see that coefficient values shrinking towards zero.

5) Regression tree

A regression tree, also known as a decision tree for regression, is a type of machine learning model used for predicting numerical or continuous values. It is a tree-based model that recursively splits the data into subsets based on the values of input features, and then fits a regression model to each subset to predict the target variable.

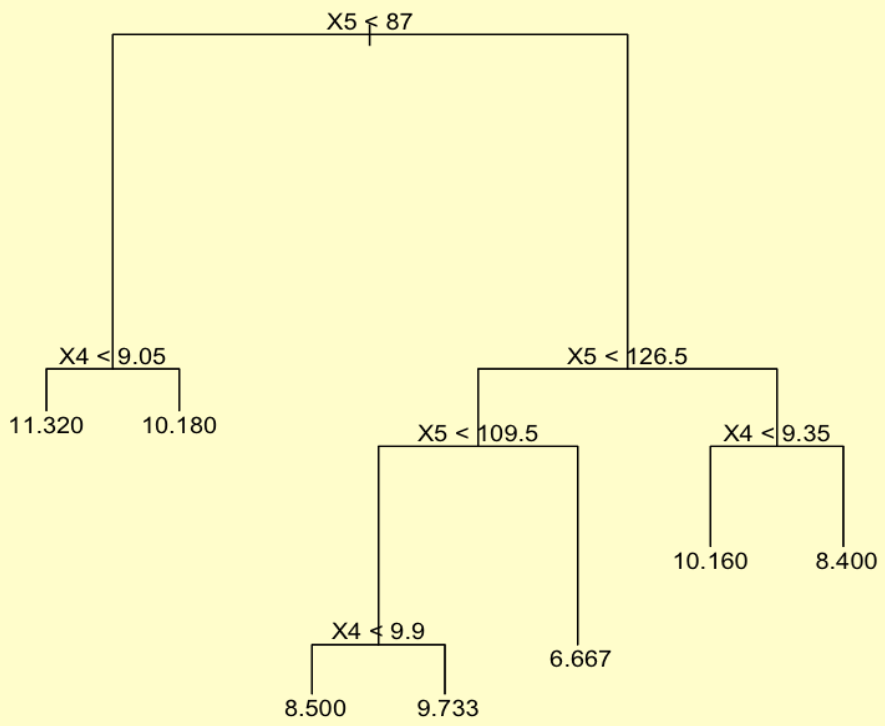
The basic idea behind a regression tree is to repeatedly partition the input feature space into non-overlapping regions, or "leaves", such that the target variable within each leaf is as homogeneous as possible. This is achieved by making splits in the feature space that maximize the reduction in the variability of the target variable at each step, typically using metrics such as mean squared error or mean absolute error. The process of recursively splitting the data and building the tree continues until a stopping criterion is met, such as reaching a maximum depth or minimum number of samples in a leaf node.

Once the tree is constructed, it can be used for making predictions on new input data by following the decision path from the root node to a leaf node and using the mean or median of the target variable within that leaf as the predicted value. Regression trees are relatively simple to interpret and visualize, and they can capture non-linear relationships and interactions between features in the data. However, they can be prone to overfitting, especially when the tree becomes too deep or when there are noisy or sparse data. To mitigate this, techniques such as pruning, regularization, or ensemble methods like random forests or boosting can be used in conjunction with regression trees.

We do regression trees method on our dataset and plot the tree.

```
library(tree)
set.seed(1)
tree_model <-tree(X1~.,health_df)
plot(tree_model)
```

```
text(tree_model, pretty = 0)
```

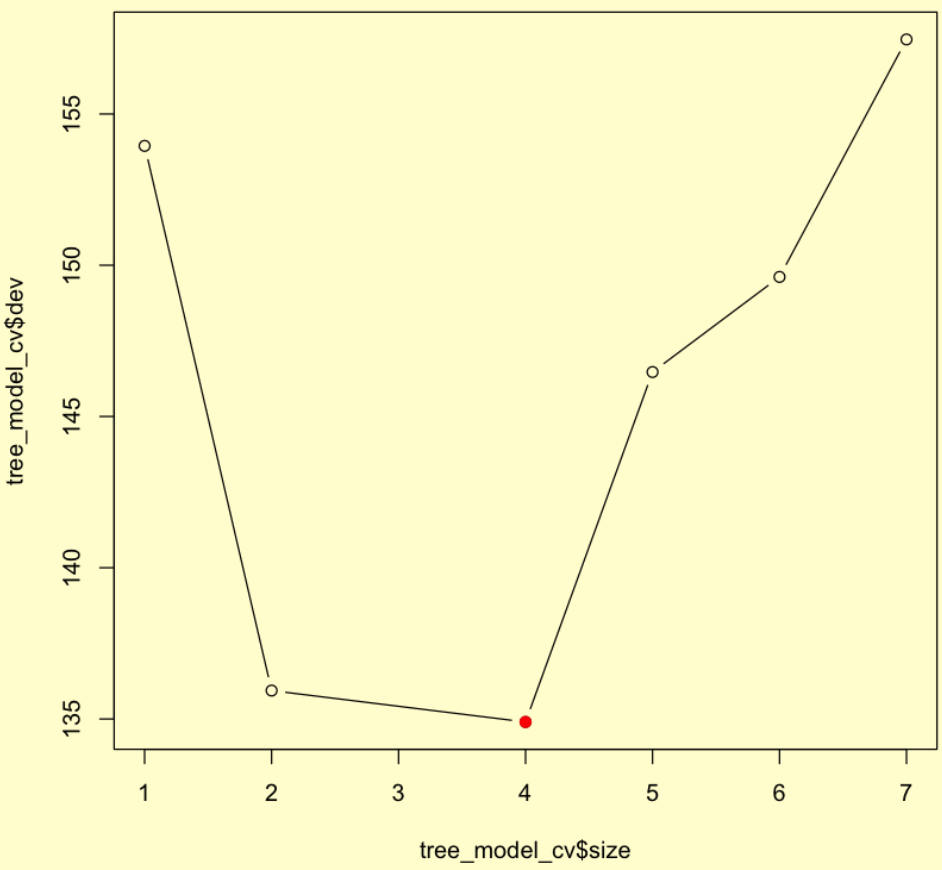


- From the tree we can see that Split with X5 has explained large amount of variance in the data.

Now we prune the tree.

```
set.seed(1)
tree_model_cv <- cv.tree(tree_model, FUN = prune.tree, method = "deviance")

plot(tree_model_cv$size, tree_model_cv$dev, type = "b")
```

```
points(tree_model_cv$size[which.min(tree_model_cv$dev)], min(tree_model_cv$dev), col =
"red", pch = 19)
```



Evaluate cross-validation error.

```
# Extract cross-validation error and optimal tree size
print(paste('Minimum CV Error is :', min(tree_model_cv$dev), 'Occured for the tree size of',
tree_model_cv$size[which.min(tree_model_cv$dev)]))
```

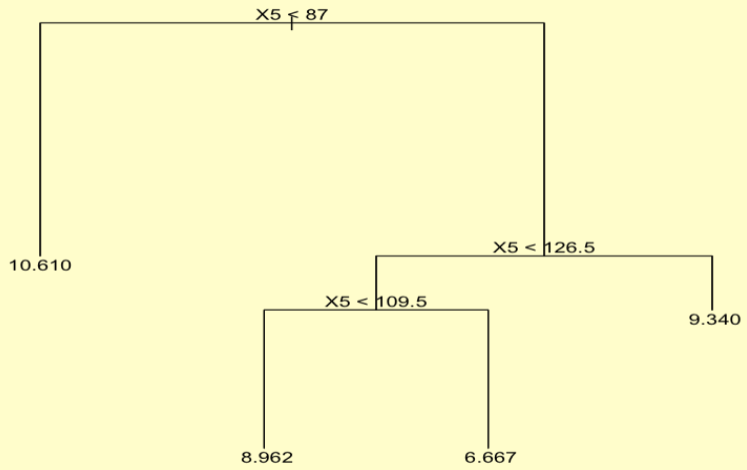
Output:

```
[1] "Minimum CV Error is : 134.900923234265 Occured for the tree size of 4"

"Minimum CV Deviance is : 134.900923234265 Occured for the tree size of 4"
```

Plot the pruned tree.

```
pruned_tree = prune.tree(tree_model, best =
tree_model_cv$size[which.min(tree_model_cv$dev)])
```

```
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```



- From the pruned tree we can see that Split with only X5 has explained significant amount of variance in the data. It further proves our observation from t-statistic and subset selection that only X5 has impact on response.

Divergence v MSE

In regression trees, divergence refers to the difference or dissimilarity between the predicted values and the actual target values at a particular node of the tree. Mean squared error (MSE) is a commonly used splitting criterion in regression trees, which measures the average squared difference between the predicted and actual target values. The relationship between divergence and MSE in regression trees can be summarized as follows:

1. Higher divergence results in higher MSE: If the predicted values at a node are highly divergent from the actual target values, it means that the tree is not accurately capturing the underlying patterns in the data. This will result in higher MSE, as the squared differences between the predicted and actual values will be larger.
2. Lower divergence results in lower MSE: On the other hand, if the predicted values at a node are closer to the actual target values, it means that the tree is accurately capturing the patterns in the data. This will result in lower MSE, as the squared differences between the predicted and actual values will be smaller.
3. MSE is minimized during tree construction: During the construction of a regression tree, the goal is to recursively split the data into subsets at each node in a way that minimizes the MSE of the predicted values for each subset. This is achieved by selecting splitting variables and splitting points that result in the lowest MSE for the resulting subsets.

In summary, in regression trees, higher divergence between predicted and actual values leads to higher MSE, while lower divergence leads to lower MSE. The goal during tree construction is to minimize MSE by selecting optimal splitting variables and splitting points.

In a regression tree, divergence can be quantified by calculating the difference or dissimilarity between the predicted values and the actual target values at a particular node. One common measure of divergence used in regression trees is the mean squared error (MSE), which is calculated as the average of the squared differences between the predicted and actual target values.

5.1) Random Forest

Random Forest is an ensemble learning technique used for both classification and regression tasks in machine learning. It is based on the concept of decision trees, which are tree-like structures used for decision-making or prediction. However, instead of using a single decision tree, Random Forest combines multiple decision trees to make predictions in a more robust and accurate way.

The key idea behind Random Forest is to create a collection of decision trees that are trained on different subsets of the training data, obtained through random sampling with replacement (known as bootstrapping). This process creates a diverse set of decision trees, each with its own slightly different perspective on the data. During prediction, the output of each individual decision tree is combined to obtain the final prediction. Random Forest has several advantages over using a single decision tree. Some of the main advantages are:

1. Robustness: Random Forest reduces overfitting by averaging the predictions of multiple decision trees, which helps to improve the model's generalization performance and makes it more robust to noisy data.

2. Stability: Random Forest is less sensitive to variations in the input data compared to a single decision tree. Small changes in the training data are less likely to result in large changes in the predictions, making the model more stable.

3. Feature Importance: Random Forest can provide estimates of feature importance, which can help identify the most important features for making accurate predictions. This can be useful for feature selection and feature engineering tasks.

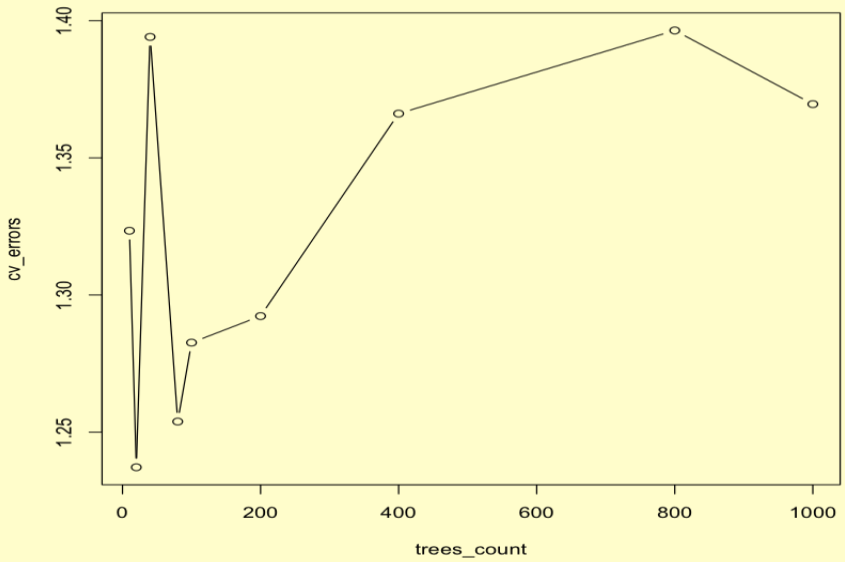
4. Handling Missing Values: Random Forest can handle missing values in the input data without the need for explicit imputation, as it can still make predictions using the available features.

5. Scalability: Random Forest can handle large datasets with a large number of features efficiently, making it suitable for a wide range of applications.

Random Forest is widely used in various domains, such as finance, healthcare, image recognition, and natural language processing, due to its flexibility, accuracy, and robustness. However, it is also important to carefully tune hyperparameters such as the number of trees in

the forest, tree depth, and other parameters to optimize its performance for a specific problem.
We now implement random forest technique for our dataset by initially importing the necessary libraries and fit the random forest model and plot.

```
library(randomForest)
library(rfUtilities)
predictors = dim(health_df)[2] - 1
trees_count = rep(1:1000)
cv_errors = c()
set.seed(1)
for (rf_tree_count in trees_count){

   # Fitting random forest model
   rf_model = randomForest(X1 ~ ., data = health_df, m_try = sqrt(predictors), ntree =
rf_tree_count)

   # Perform Cross Validation
   cv_results = rf.crossValidation(rf_model, health_df, n=5)

   cv_errors = c(cv_errors, mean(cv_results$model.mse))
}

plot(trees_count, cv_errors, type = "b")
```



Calculate minimum cross-validation error.

```
paste("Minimum CV error:", min(cv_errors), "at number of trees", which.min(cv_errors))
```

Output:

```
'Minimum CV error: 1.23720639482381 at number of trees 2'
```

we got 'Minimum CV error: 1.23720639482381 at number of trees 2 and number of predictors used is ($\sqrt{\text{Predictorscount}}$)

```
set.seed(1)
best_rf_model = randomForest(X1 ~ ., data = health_df, m_try = sqrt(predictors), ntree =
which.min(cv_errors))
best_rf_model
```

Outputs:

```
Call:
 randomForest(formula = X1 ~ ., data = health_df, m_try = sqrt(predictors),      ntree =
which.min(cv_errors))
               Type of random forest: regression
                     Number of trees: 2
No. of variables tried at each split: 1

          Mean of squared residuals: 6.651972
                    % Var explained: -145.29
```

Cross-Validation errors for all the methods attempted.

Linear Regression	Linear Regression with subset selection	Polynomial Regression	Ridge Regression	Lasso Regression	Regression Trees	Random Forest
2.83	2.635	2.914	2.570	2.5870	134.9009	1.2372

Q.2) You will be using one of the three data sets located at https://archive.ics.uci.edu/ml/datasets.php?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table.This is a popular data set repository maintained by University of California at Irvine. Pick regression and attribute numerical in the left pane. The three suggested data sets are:

Airfoil self-noise data set

Combined cycle power plant

Real estate valuation data set

Solution:

For this problem we chose Air foil self-noise data set from the link that is provided. The name of the data set is "airfoil_self_noise.dat".

Abstract:

This is a NASA data set, obtained from a series of aerodynamic and acoustic tests of two and three-dimensional airfoil blade sections conducted in an anechoic wind tunnel.

Source:

Provide the names, email addresses, institutions, and other contact information of the donors and creators of the data set.
Donor:
Dr Roberto Lopez
robertolopez '@' intelnics.com
Intelnics

Creators:
Thomas F. Brooks, D. Stuart Pope and Michael A. Marcolini
NASA.

Data Set Information:

The NASA data set comprises different size NACA 0012 airfoils at various wind tunnel speeds and angles of attack. The span of the airfoil and the observer position were the same in all of the experiments.

Attribute Information:

This problem has the following inputs:
1. Frequency, in Hertzs.
2. Angle of attack, in degrees.
3. Chord length, in meters.
4. Free-stream velocity, in meters per second.
5. Suction side displacement thickness, in meters.
The only output is:
6. Scaled sound pressure level, in decibels.

Regression:

Regression is a statistical technique which is used to simulate the relationship between two or more variables. Finding the mathematical formula that best captures the link between the dependent variable, which is the variable being predicted, and one or more independent variables, which are the variables utilized to make the prediction, requires examining data.

Regression analysis aims to fit a regression model to the data so that it can be utilized for inference, prediction, or understanding of the connection between variables. A linear equation, which depicts a straight-line relationship between the variables, or a nonlinear equation, which indicates a curved relationship between the variables, are two types of equations that are frequently used to represent regression models.

In addition to forecasting future results, regression analysis can also be used to analyse the strength and direction of correlations between variables, identify key predictors, and assess the effects of interventions or treatments on outcomes. Numerous disciplines, including statistics, economics, the social sciences, business, finance, medicine, and machine learning, among others, frequently use regression.

For this problem we first read the dataset and store it in “airfoil_df” and see the first 6 rows of the data .

```
airfoil_df <- read.table("data/airfoil_self_noise.dat")

head(airfoil_df)
```

Output:

A data.frame: 6 × 6

	V1	V2	V3	V4	V5	V6
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	800	0	0.3048	71.3	0.00266337	126.201
2	1000	0	0.3048	71.3	0.00266337	125.201
3	1250	0	0.3048	71.3	0.00266337	125.951
4	1600	0	0.3048	71.3	0.00266337	127.591
5	2000	0	0.3048	71.3	0.00266337	127.461
6	2500	0	0.3048	71.3	0.00266337	125.571

We find the dimensions of the given data set.

```
cat("Dimesions of the data:",dim(airfoil_df))
```

Output:

```
Dimesions of the data: 1503 6
```

We find the number of null or N/A values in the given dataset.

```
paste("Number of NA Values: ",sum(is.na(airfoil_df)))
```

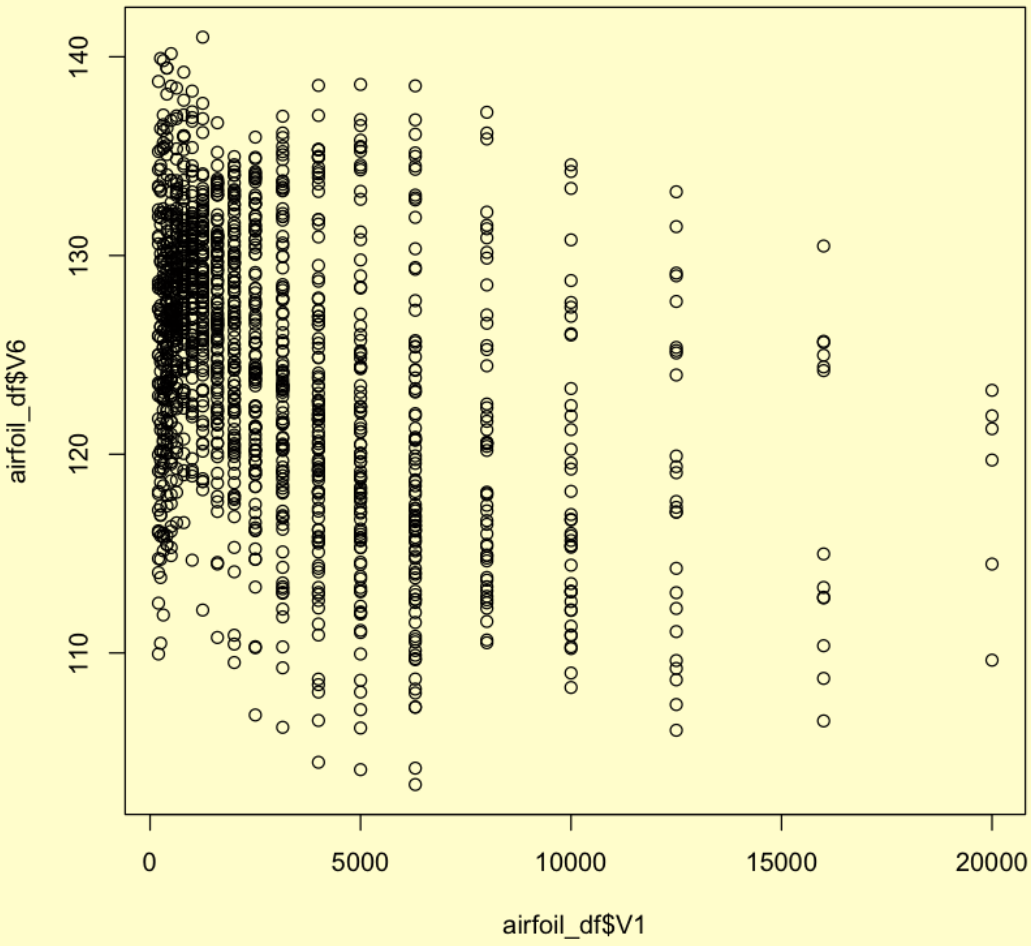
Output:

```
'Number of NA Values: 0'
```

So, there are no null or N/A values in the given dataset.

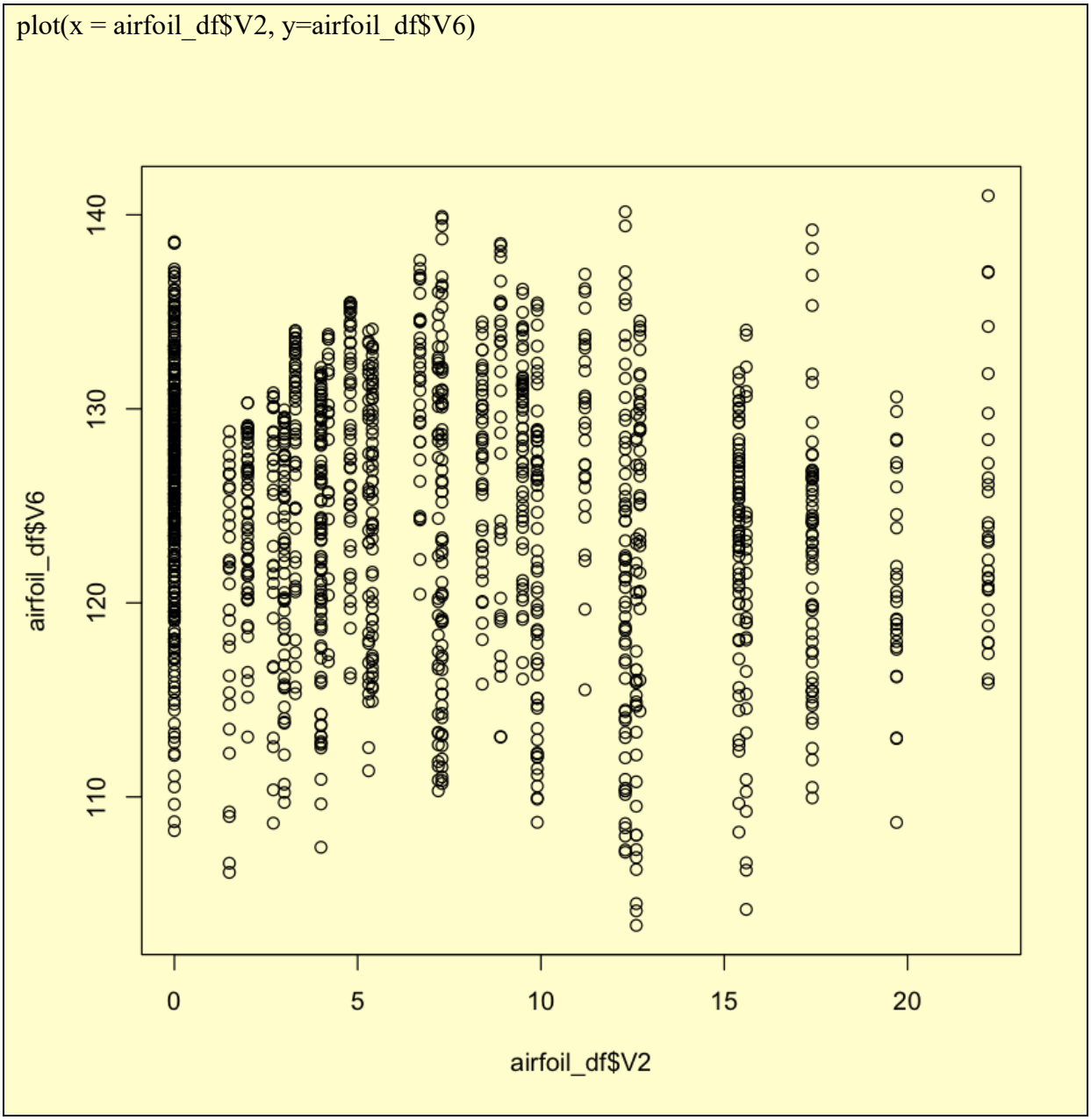
We now plot the graph between V1 and V6, which are the columns of the given dataset.

```
plot(x = airfoil_df$V1, y=airfoil_df$V6)
```



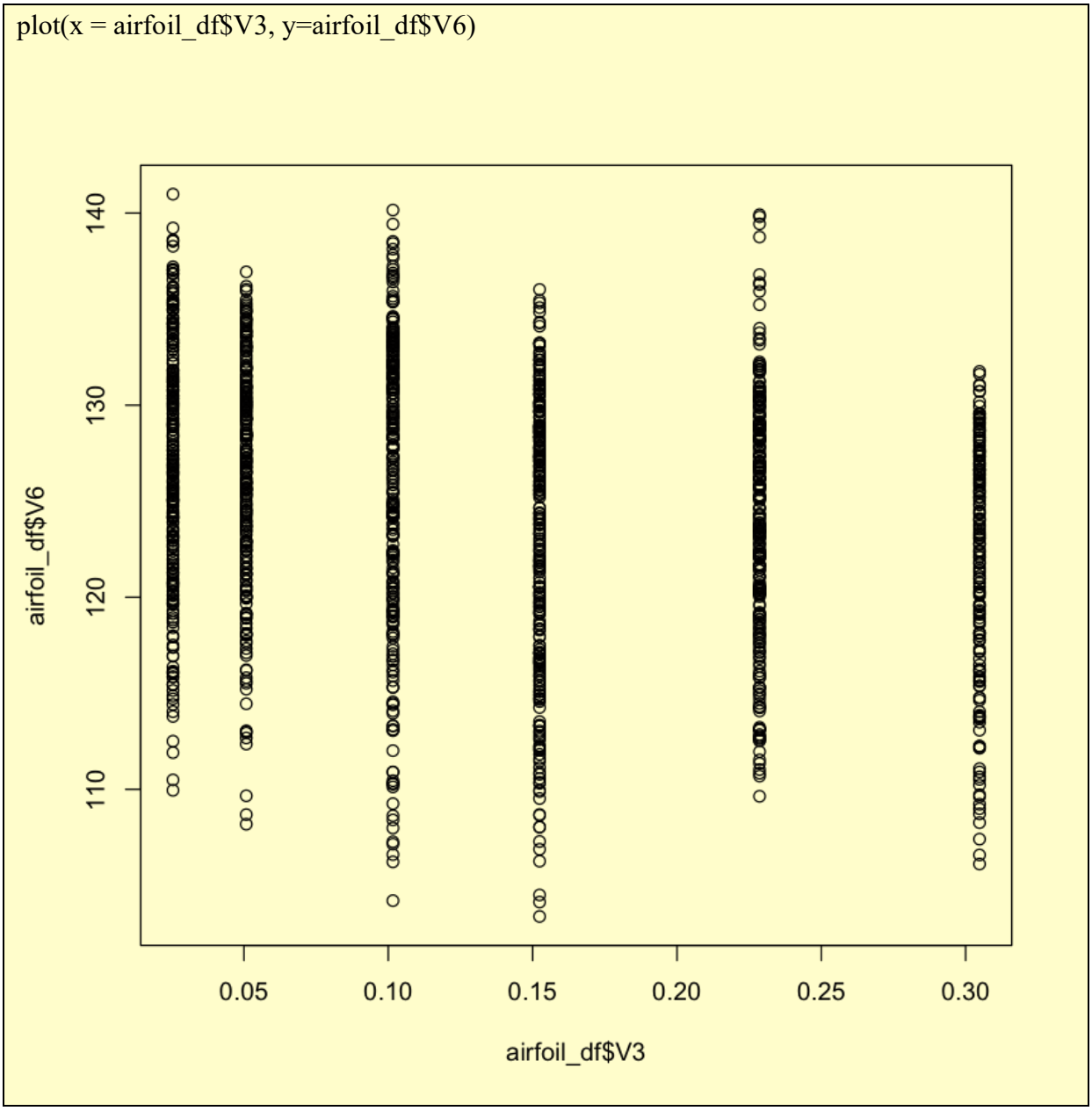
We now plot the graph between V2 and V6, which are the columns of the given dataset.

```
plot(x = airfoil_df$V2, y=airfoil_df$V6)
```



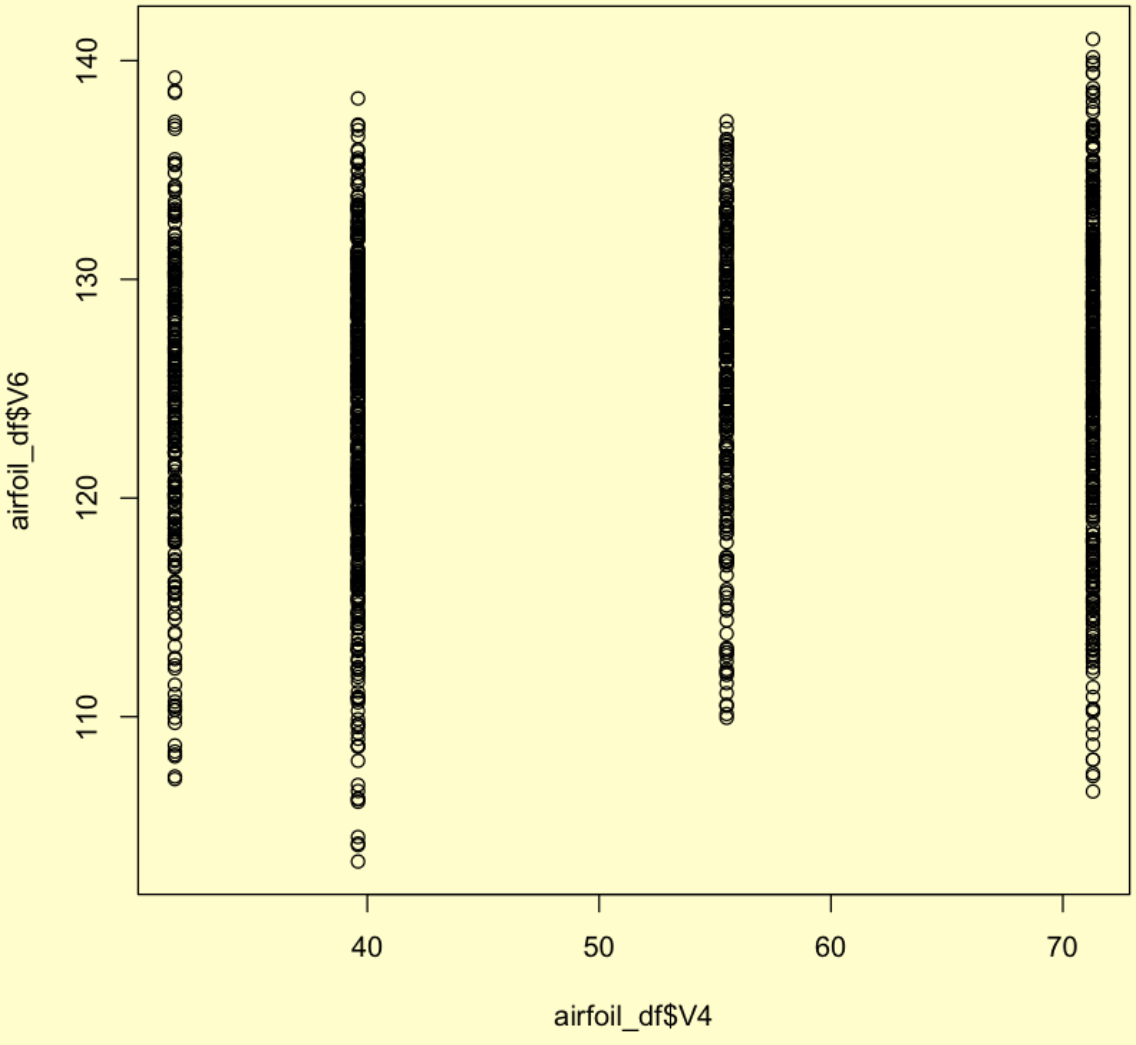
We now plot the graph between V3 and V6, which are the columns of the given dataset.

```
plot(x = airfoil_df$V3, y=airfoil_df$V6)
```



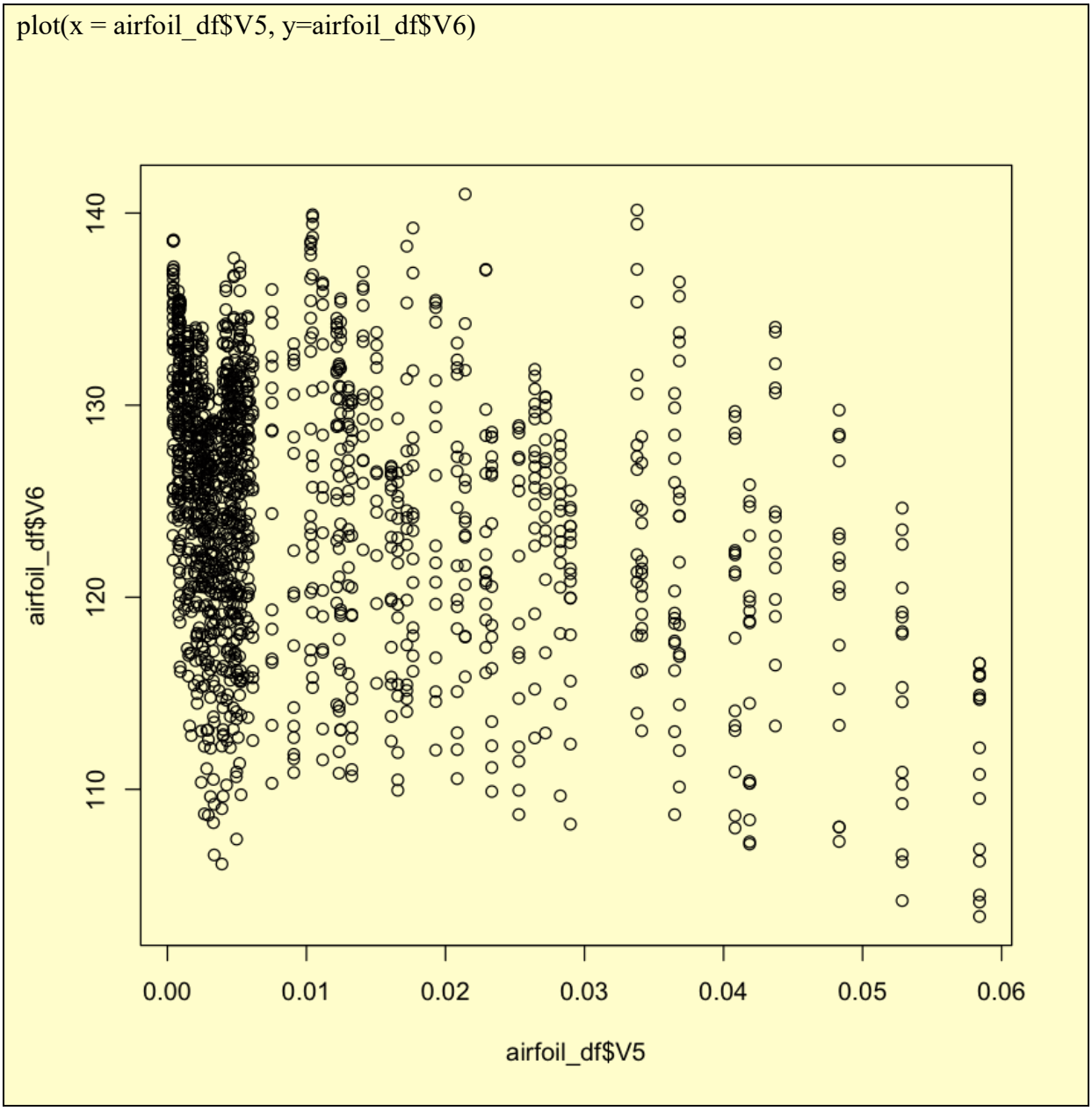
We now plot the graph between V4 and V6, which are the columns of the given dataset.

```
plot(x = airfoil_df$V4, y=airfoil_df$V6)
```



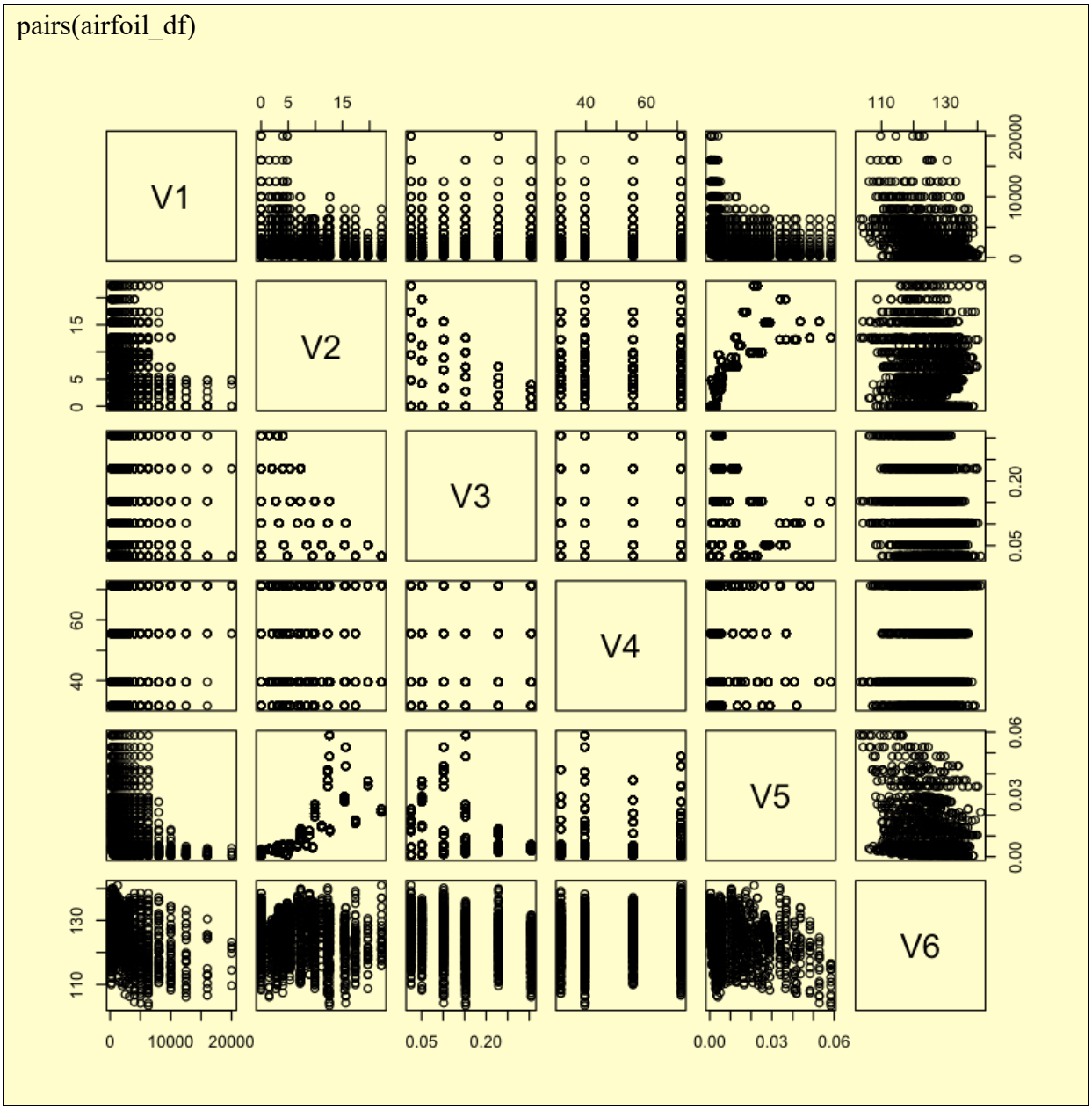
We now plot the graph between V5 and V6, which are the columns of the given dataset.

```
plot(x = airfoil_df$V5, y=airfoil_df$V6)
```



By using the “pairs” function we plot the scatter plot for each element with other element and create a matrix of scatterplots.

```
pairs(airfoil_df)
```



1) Linear Regression:

A statistical technique called linear regression is used to represent the relationship between two variables, where one of the variables is the dependent variable and the other is the independent variable. Finding the regression line or line of best fit that minimizes the residual sum of squares (RSS), the sum of the squared differences between the observed values of the dependent variable and the predicted values by the regression line, is the objective of linear regression.

A linear regression model's equation appears as follows:

$Y = \beta 0 + \beta 1 * X + \varepsilon$

where:

Y is the dependent variable, also known as the response variable or the target variable.

X is the independent variable, also known as the predictor variable or the feature variable.

β0 is the intercept term, which represents the predicted value of Y when X is equal to 0.

β1 is the slope coefficient, which represents the change in Y for a unit change in X.

ε is the error term or the residual, which represents the unexplained variation in Y that is not accounted for by the linear relationship with X.

The following are the main steps in performing linear regression:

Data collection and preparation: This includes acquiring data for the dependent and independent variables, as well as cleaning and preparing the data for analysis. Data plotting is constructing a scatter plot to depict the relationship between the dependent and independent variables.

Estimating the parameters entails estimating the values of 0 and 1 that best fit the data using a statistical method such as ordinary least squares (OLS).

Model evaluation: This includes evaluating the model's goodness of fit using statistical metrics such as R-squared, adjusted R-squared, and p-values, as well as graphical diagnostics such as residual plots.

Interpreting the data entails deriving judgments about the strength and direction of the link between the variables based on the estimated parameters.

Making forecasts: Once fitted and validated, the model can be used to forecast the dependent variable for new values of the independent variable.

Linear regression is a popular and straightforward method for estimating the relationship between two variables, with numerous applications in economics, finance, social sciences, healthcare, and engineering.

We perform linear regression on the given model by setting the seed to 1 and importing the library “boot”. For this data set we take V6 as the dependent variable and perform linear regression against V1, V2, V3, V4 and V5. To estimate test error, we use K-fold cross-validation.

K-fold cross-validation (CV) is a technique used to evaluate the performance of a machine learning model by partitioning the data into k equally sized folds and iteratively training the model on k-1 folds while using the remaining one-fold as the validation set. This process is repeated k times, with each fold being used once as the validation set, and the average performance across all folds is computed to obtain an estimate of the model's performance. We generally choose k value to be 5.

```
library(boot)
lm_model = glm(V6 ~ ., data = airfoil_df)
lm_cv_model = cv.glm(data=airfoil_df, lm_model, K = 5)
cv_error = lm_cv_model$delta[1]


print(paste("CV error:", cv_error))
```

Output:

```
[1] "CV error: 23.3703288570954"
```

On Performing Cross Validation, the estimated test error for linear regression was: 23.37.

For a better understanding after linear regression, we look at the summary.

```
lm_model <- lm(V6 ~ ., data = airfoil_df)
summary(lm_model)
```

Output:

```
Call:
lm(formula = V6 ~ ., data = airfoil_df)


Residuals:
    Min      1Q  Median      3Q     Max
-17.480  -2.882  -0.209   3.152  16.064


Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.328e+02  5.447e-01  243.87   <2e-16 ***
V1          -1.282e-03  4.211e-05  -30.45   <2e-16 ***
V2          -4.219e-01  3.890e-02  -10.85   <2e-16 ***
```

```
V3          -3.569e+01  1.630e+00  -21.89   <2e-16 ***
V4           9.985e-02  8.132e-03   12.28   <2e-16 ***
V5          -1.473e+02  1.501e+01   -9.81   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 4.809 on 1497 degrees of freedom
Multiple R-squared:  0.5157,  Adjusted R-squared:  0.5141
F-statistic: 318.8 on 5 and 1497 DF,  p-value: < 2.2e-16
```

The P - value of F - statistic says that all predictor has significant effect on the the response.

The P-values of t-statistic also justifies the same.

We get the confidence intervals of the coefficients in our data set.

```
confint (lm_model)
```

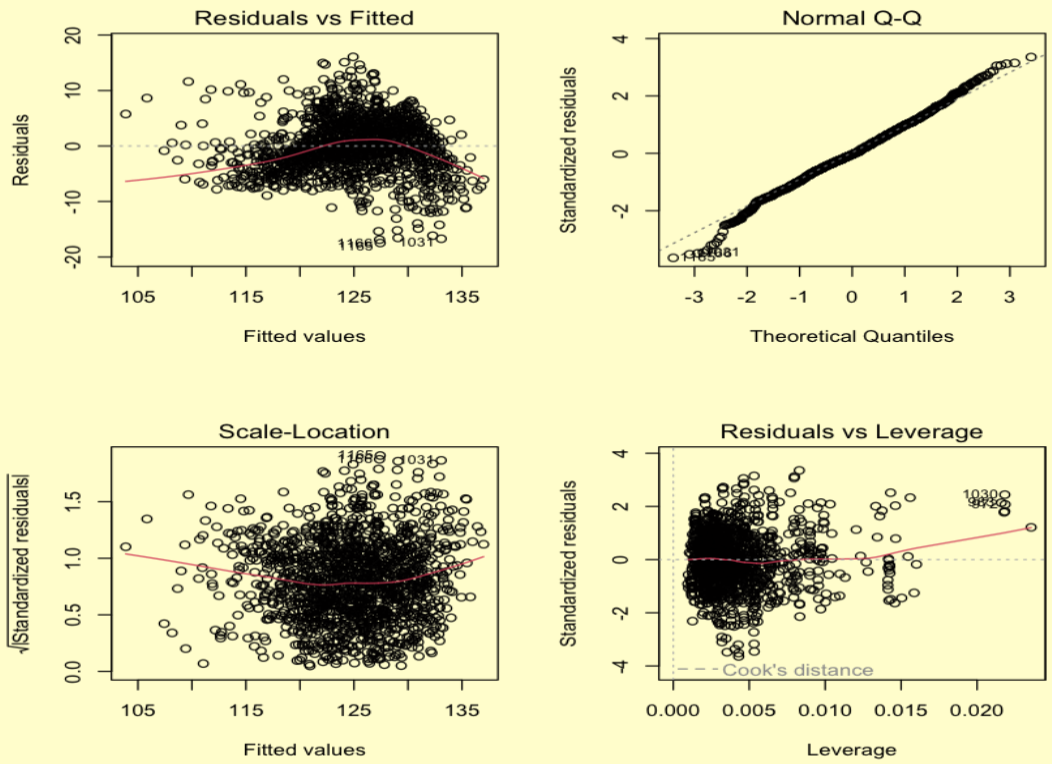
Output:

A matrix: 6 × 2 of type dbl

	2.5 %	97.5 %
(Intercept)	1.317653e+02	1.339023e+02
V1	-1.364799e-03	-1.199615e-03
V2	-4.982083e-01	-3.456151e-01
V3	-3.888617e+01	-3.248983e+01
V4	8.390221e-02	1.158059e-01
V5	-1.767525e+02	-1.178485e+02

We now plot the graphs attained after linear regression.

```
par(mfrow = c(2,2))
plot(lm_model)
```



Problems in the fit:

The Residual vs Fitted Values shows no significant non-linearity in the data.

The plot also says that there is constant variance in the errors, which shows that our assumption that errors ϵ has constant variance is right. This is further justified by the Fitted Values vs $\sqrt{\text{Standardized} - \text{residuals}}$ plot.

The quantile-quantile plot showed errors deviating from the normality assumption at left end of the tail.

In the bottom-right plot we can see, some points have standardized residuals greater than 3, which makes them outliers.

The cut-off for the leverage point threshold is $2p/n$where p is the number of predictors, which is one in this case, and n is the number of observations, which is 53 in this case. Therefore, the cut-off is (2∗5)/1503=0.00662∗51503=0.0066. Hence, there are some leverage points in this dataset.

As some observations qualified as both outliers and leverage points, we can safely say we have some influential points.

Subset selection in linear regression is a technique used to select a subset of predictor variables from a larger set of variables to build a linear regression model. This is typically

done to identify the most important variables that have a significant impact on the response variable, and to simplify the model by removing unnecessary variables.

Here we set the seed to 1 and load “leaps” library and perform best subset selection by using “regsubsets”

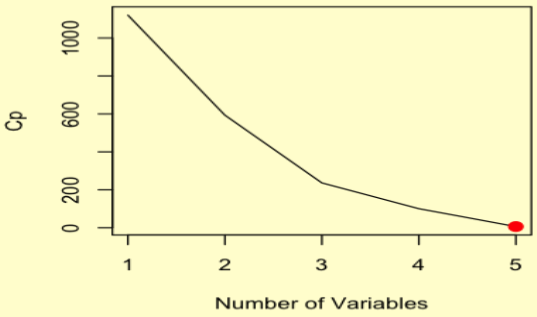
```
# Load the leaps package

library(leaps)

# Perform best subset selection using the regsubsets function

subset_model <- regsubsets(V6 ~ ., data = airfoil_df, nvmax = 6)

summary(subset_model)
```

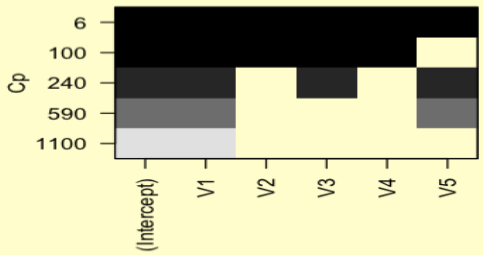
Output:

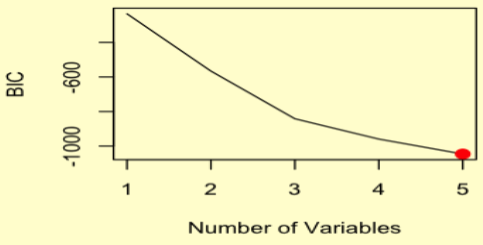
```
Subset selection object

Call: regsubsets.formula(V6 ~ ., data = airfoil_df, nvmax = 6)

5 Variables  (and intercept)

   Forced in Forced out

V1     FALSE     FALSE

V2     FALSE     FALSE

V3     FALSE     FALSE

V4     FALSE     FALSE

V5     FALSE     FALSE

1 subsets of each size up to 5

Selection Algorithm: exhaustive

         V1  V2  V3  V4  V5

1  ( 1 ) "*" " " " " " " " "

2  ( 1 ) "*" " " " " " " "*"

3  ( 1 ) "*" " " "*" " " "*"

4  ( 1 ) "*" "*" "*" "*" " "

5  ( 1 ) "*" "*" "*" "*" "*"
```

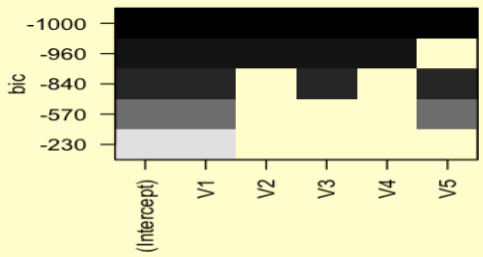
Now draw plots for CP metric and BIC metric.

```
regfit_summary = summary(subset_model)

par(mfrow = c(2,2))

# Drawing plots for CP Metric

plot(regfit_summary$cp, xlab = "Number of Variables", ylab = "Cp",type = "l")

points(which.min(regfit_summary$cp), regfit_summary$cp[which.min(regfit_summary$cp)],
col = "red", cex = 2, pch = 20)

plot(subset_model, scale = "Cp")

# Drawing plots for BIC Metric

plot(regfit_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")

points(which.min(regfit_summary$bic),
regfit_summary$bic[which.min(regfit_summary$bic)], col = "red", cex = 2, pch = 20)

plot(subset_model, scale = "bic")
```









Plot show that model with all predictor does better than the rest.

This goes hand in hand with what we observed with t-test p-values.

Now creating a linear regression model with only the predictor we found to be best after subset selection.

```
best_subset_lm_model = glm(V6 ~ ., data = airfoil_df)
best_subset_lm_cv_model = cv.glm(data=airfoil_df, best_subset_lm_model, K = 5)
cv_error = best_subset_lm_cv_model$delta[1]
print(paste("CV error:", cv_error))
```

Output:

```
[1] "CV error: 23.2549261785911"
```

On Performing Cross Validation, the estimated test error for best subset selected linear regression (with only one predictor) is: 23.25.

2) Ridge Regression

Ridge regression is a type of linear regression technique used for regression analysis, a statistical method used to model the relationship between a dependent variable and one or more independent variables. Ridge regression is an extension of ordinary least squares (OLS) regression in which a penalty term is added to the objective function to reduce model complexity and prevent overfitting.

The main idea behind ridge regression is to introduce a regularization term, often called a "ridge" or "L2" penalty, which adds a penalty to the regression coefficients based on their magnitudes. The regularization term is multiplied by a hyperparameter called the regularization strength, denoted as λ (lambda), which controls the trade-off between fitting the data and regularization. A larger value of λ results in stronger regularization, leading to smaller coefficients and a simpler model, while a smaller value of λ allows for larger coefficients and a more complex model.

Mathematically, the objective function of ridge regression can be represented as:

minimize: $||Y - X\beta||^2 + \lambda||\beta||^2$

where:

Y is the vector of observed dependent variable values

X is the matrix of independent variable values

β is the vector of regression coefficients to be estimated

λ is the regularization strength

$||.||^2$ denotes the squared Euclidean norm, which is the sum of squared values

Ridge regression can be solved using various optimization techniques, such as closed-form solutions or iterative methods, and it has several advantages. It can handle multicollinearity (high correlation between independent variables) better than ordinary least squares regression, and it can provide more stable estimates of the regression coefficients. Ridge

regression is commonly used in situations where there are multiple correlated predictors, and the goal is to prevent overfitting and obtain a more robust and generalized model.

We develop a ridge regression model on our dataset by setting the seed to 1 and we set up a cross-validation grid and perform cross-validation using the “cv.glmnet” function. Select the best value of lambda.

```
x = model.matrix(V6 ~ ., airfoil_df)[, -1]

y = airfoil_df$V6

# Set up the cross-validation grid

cv_grid = expand.grid(lambda = seq(from = 0.0001, to = 10, length = 100))


# Perform cross-validation using the cv.glmnet function

rr_cv_model = cv.glmnet(x, y, alpha = 0, lambda = cv_grid$lambda, nfolds = 5)


# Select the best value of lambda

best_lambda = rr_cv_model$lambda.min


# Print the minimum cross-validation error

print(paste("Best Lambda Value:", best_lambda))
```

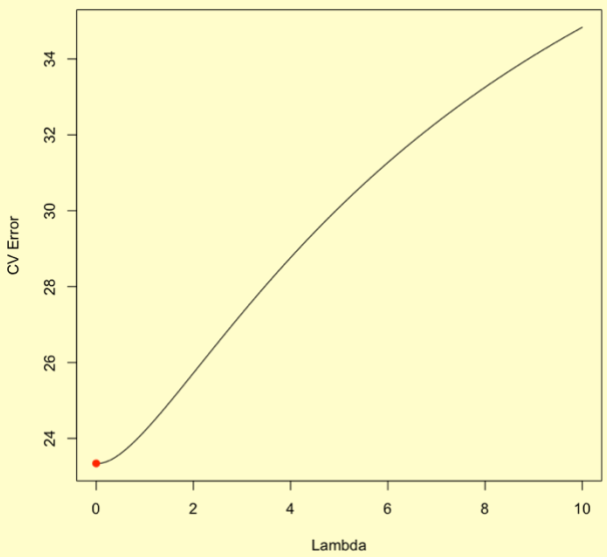
Output:

```
[1] "Best Lambda Value: 1e-04"
```

Now we plot the cross-validation error curve.

```
cv_error <- rr_cv_model$cvm


# Plot the cross-validation error curve

plot(rr_cv_model$lambda, cv_error, type = "l", xlab = "Lambda", ylab = "CV Error")
```

```
points(best_lambda, min(cv_error), col = "red", pch = 19)
```



Now we get minimum cross-validation error.

```
# Get the minimum cross-validation error
min_cv_error = min(rr_cv_model$cvm)


# Print the minimum cross-validation error
print(paste("Minimum CV error:", min_cv_error))
```

Output:

```
[1] "Minimum CV error: 23.3416951321391"
```

We got minimum CV error of 23.34 for the Lambda value of 1e-4.

```
rr_best_model = glmnet(x, y, alpha = 0, lambda = best_lambda)
coef(rr_best_model)
```

Output:

```
6 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept)  1.328315e+02
V1          -1.282033e-03
V2          -4.215147e-01
V3          -3.567978e+01
```

```
V4          9.983830e-02
V5         -1.474106e+02
```

3) Lasso Regression

Lasso regression, also known as L1 regularization, is a type of linear regression technique that incorporates regularization to improve the predictive accuracy and interpretability of the model. In lasso regression, a penalty term is added to the ordinary least squares (OLS) objective function, which encourages the model to use a smaller number of predictors or features in the final model.

The key idea of lasso regression is to introduce a penalty term based on the absolute values of the regression coefficients. The penalty term is proportional to the sum of the absolute values of the coefficients multiplied by a hyperparameter, often denoted as lambda (λ). The hyperparameter λ controls the strength of the regularization, with larger values of λ resulting in stronger regularization and smaller values of λ resulting in weaker regularization.

The lasso regression model seeks to minimize the following objective function:

OLS objective function + λ * (sum of absolute values of coefficients)

The regularization term in lasso regression has a unique property that it can set some of the regression coefficients to exactly zero. This allows for feature selection, where less important features can be automatically excluded from the model. This property makes lasso regression particularly useful when dealing with high-dimensional data, where there are many predictors and some of them may be irrelevant or redundant.

Lasso regression can be used for various purposes, including prediction, variable selection, and interpretation. It is commonly used in machine learning and statistics for tasks such as linear regression, logistic regression, and generalized linear models. Lasso regression is implemented in many statistical software packages and is widely used in practice for building predictive models with sparse or high-dimensional data.

To perform Lasso regression on the given data set by finding best lambda value

```
# Set up the cross-validation grid
cv_grid = expand.grid(lambda = seq(from = 0.001, to = 10, length = 100))


# Perform cross-validation using the cv.glmnet function
lasso_cv_model = cv.glmnet(x, y, alpha = 1, lambda = cv_grid$lambda, nfolds = 5)


# Select the best value of lambda
best_lambda = lasso_cv_model$lambda.min
```

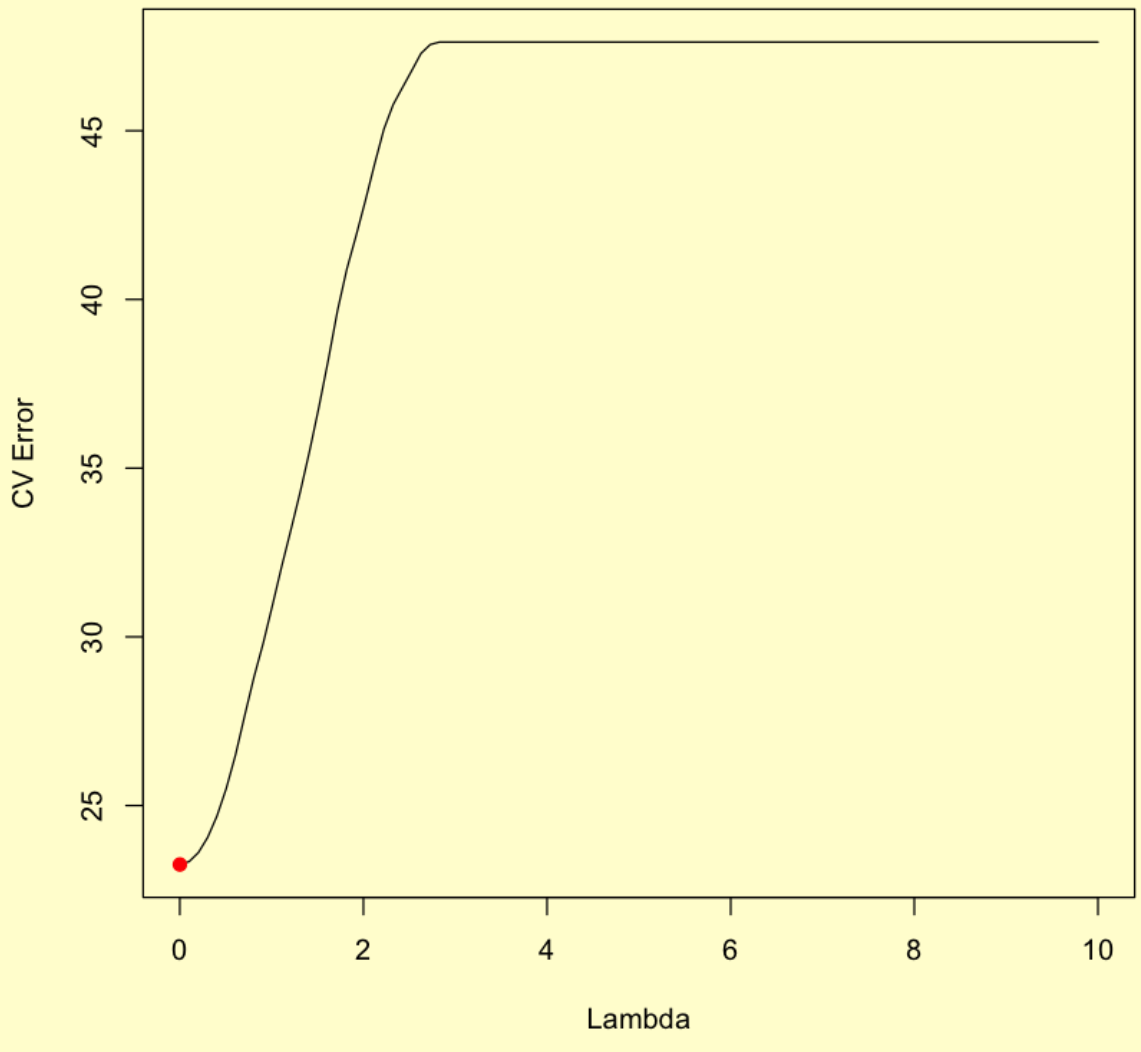
```
# Print the minimum cross-validation error
print(paste("Best Lambda Value:", best_lambda))
```

Output:

```
[1] "Best Lambda Value: 0.001"
```

Now plot the cross-validation error curve.

```
cv_error <- lasso_cv_model$cvm

# Plot the cross-validation error curve
plot(lasso_cv_model$lambda, cv_error, type = "l", xlab = "Lambda", ylab = "CV Error")
points(best_lambda, min(cv_error), col = "red", pch = 19)
```



Getting the minimum cross-validation error.

```
# Get the minimum cross-validation error
min_cv_error = min(lasso_cv_model$cvm)


# Print the minimum cross-validation error
print(paste("Minimum CV error:", min_cv_error))
```

Output:

```
[1] "Minimum CV error: 23.252373329225"
```

* We got minimum CV error of 23.23 for the Lambda value of 0.001.

We now find the coefficient values.

```
lasso_best_model = glmnet(x, y, alpha = 1, lambda = best_lambda)
coef(lasso_best_model)
```

Output:

```
6 x 1 sparse Matrix of class "dgCMatrix"
                    s0
(Intercept)  1.328277e+02
V1          -1.281433e-03
V2          -4.209949e-01
V3          -3.565375e+01
V4           9.974697e-02
V5          -1.474393e+02
```

4) Regression tree

A regression tree, also known as a decision tree for regression, is a type of machine learning model used for predicting numerical or continuous values. It is a tree-based model that recursively splits the data into subsets based on the values of input features, and then fits a regression model to each subset to predict the target variable.

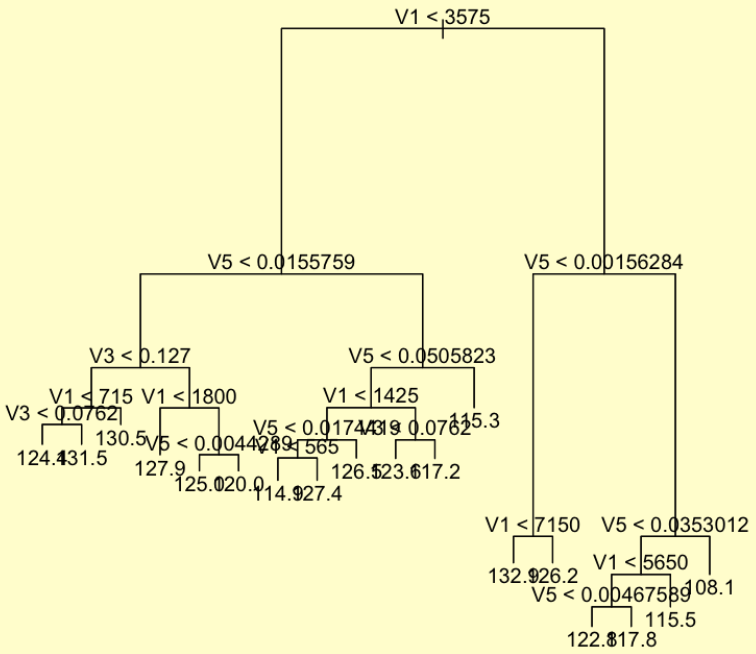
The basic idea behind a regression tree is to repeatedly partition the input feature space into non-overlapping regions, or "leaves", such that the target variable within each leaf is as homogeneous as possible. This is achieved by making splits in the feature space that maximize the reduction in the variability of the target variable at each step, typically using metrics such as mean squared error or mean absolute error. The process of recursively

splitting the data and building the tree continues until a stopping criterion is met, such as reaching a maximum depth or minimum number of samples in a leaf node.

Once the tree is constructed, it can be used for making predictions on new input data by following the decision path from the root node to a leaf node and using the mean or median of the target variable within that leaf as the predicted value. Regression trees are relatively simple to interpret and visualize, and they can capture non-linear relationships and interactions between features in the data. However, they can be prone to overfitting, especially when the tree becomes too deep or when there are noisy or sparse data. To mitigate this, techniques such as pruning, regularization, or ensemble methods like random forests or boosting can be used in conjunction with regression trees.

We do regression trees method on our dataset and plot the tree.

```
set.seed(1)

tree_model <-tree(V6~.,airfoil_df)

plot(tree_model)

text(tree_model, pretty = 0)
```

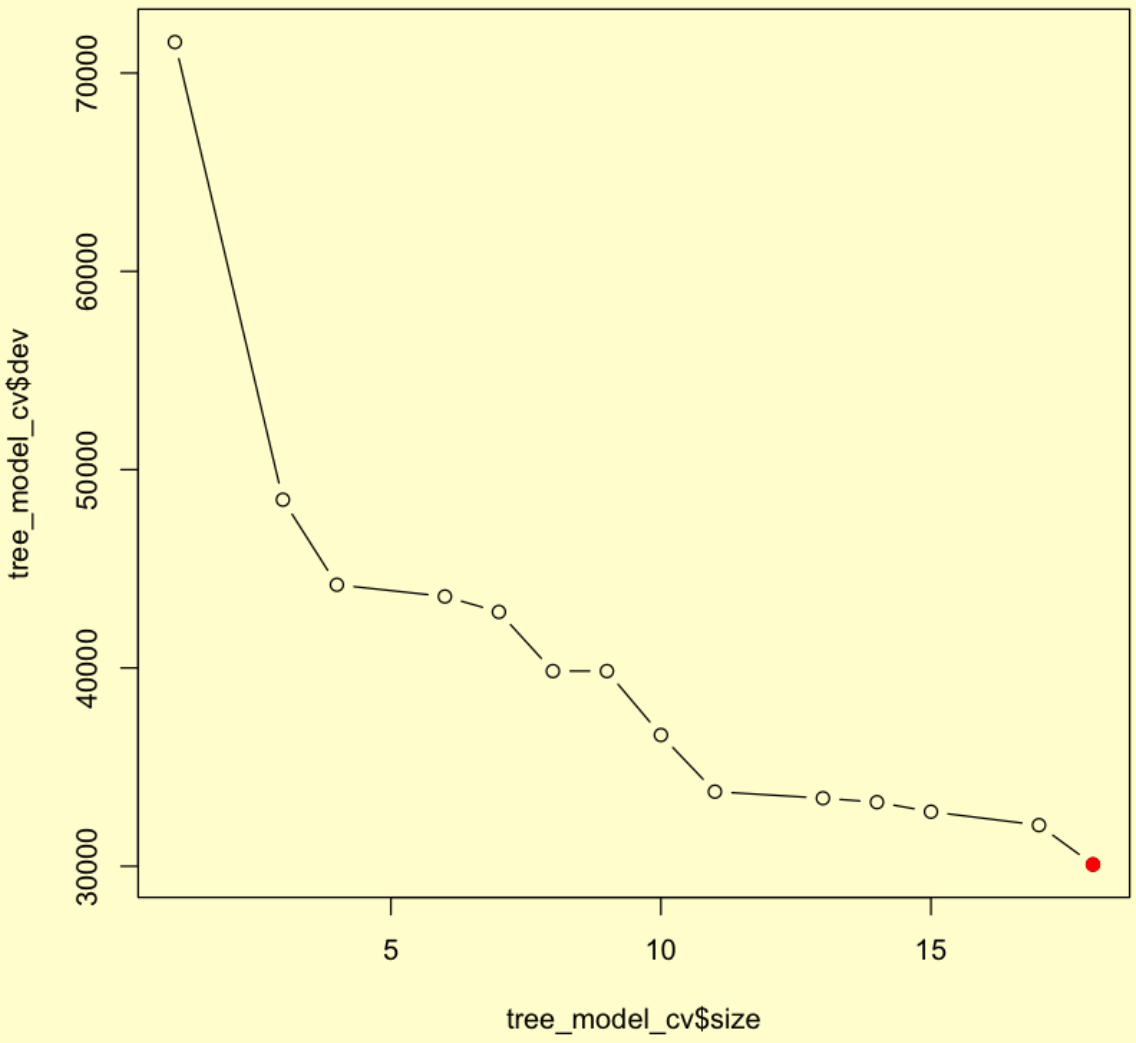


From the tree we can see that Split with V5 has explained large amount of variance in the data.

Now we prune the tree.

```
set.seed(1)
```

```
tree_model_cv <- cv.tree(tree_model, FUN = prune.tree, method = "deviance")
plot(tree_model_cv$size, tree_model_cv$dev, type = "b")
points(tree_model_cv$size[which.min(tree_model_cv$dev)], min(tree_model_cv$dev), col = "red", pch = 19)
```



Now we get cross-validation error and optimum size of the tree.

```
# Extract cross-validation error and optimal tree size
print(paste('Minimum CV Error is :', min(tree_model_cv$dev), 'Occured for the tree size of',
tree_model_cv$size[which.min(tree_model_cv$dev)]))
```

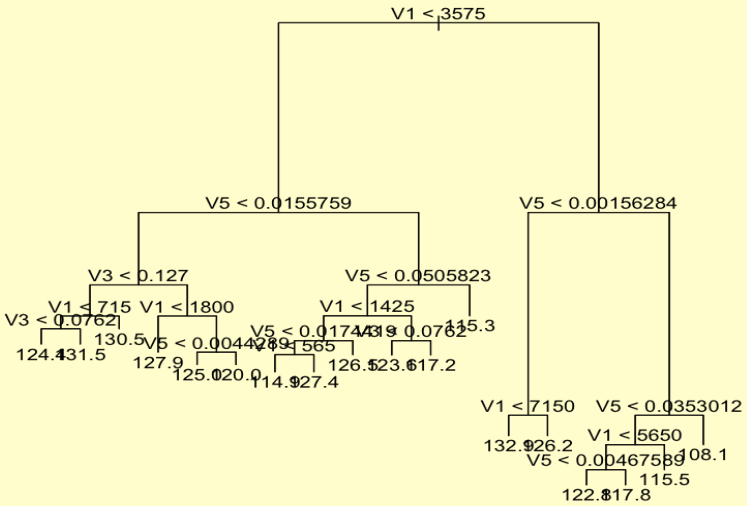
Output:

```
[1] "Minimum CV Error is : 30087.9680069964 Occured for the tree size of 18"
```

"Minimum CV Deviance is : 300087.900923234265 Occured for the tree size of 18"

Now we plot the pruned tree.

```
pruned_tree = prune.tree(tree_model, best =
tree_model_cv$size[which.min(tree_model_cv$dev)])


plot(pruned_tree)

text(pruned_tree, pretty = 0)
```



Divergence v MSE

In regression trees, divergence refers to the difference or dissimilarity between the predicted values and the actual target values at a particular node of the tree. Mean squared error (MSE) is a commonly used splitting criterion in regression trees, which measures the average squared difference between the predicted and actual target values. The relationship between divergence and MSE in regression trees can be summarized as follows:

Higher divergence results in higher MSE: If the predicted values at a node are highly divergent from the actual target values, it means that the tree is not accurately capturing the underlying patterns in the data. This will result in higher MSE, as the squared differences between the predicted and actual values will be larger.

Lower divergence results in lower MSE: On the other hand, if the predicted values at a node are closer to the actual target values, it means that the tree is accurately capturing the patterns in the data. This will result in lower MSE, as the squared differences between the predicted and actual values will be smaller.

MSE is minimized during tree construction: During the construction of a regression tree, the goal is to recursively split the data into subsets at each node in a way that minimizes the MSE

of the predicted values for each subset. This is achieved by selecting splitting variables and splitting points that result in the lowest MSE for the resulting subsets.

In summary, in regression trees, higher divergence between predicted and actual values leads to higher MSE, while lower divergence leads to lower MSE. The goal during tree construction is to minimize MSE by selecting optimal splitting variables and splitting points.

In a regression tree, divergence can be quantified by calculating the difference or dissimilarity between the predicted values and the actual target values at a particular node. One common measure of divergence used in regression trees is the mean squared error (MSE), which is calculated as the average of the squared differences between the predicted and actual target values.

5) Random Forest

Random Forest is an ensemble learning technique used for both classification and regression tasks in machine learning. It is based on the concept of decision trees, which are tree-like structures used for decision-making or prediction. However, instead of using a single decision tree, Random Forest combines multiple decision trees to make predictions in a more robust and accurate way.

The key idea behind Random Forest is to create a collection of decision trees that are trained on different subsets of the training data, obtained through random sampling with replacement (known as bootstrapping). This process creates a diverse set of decision trees, each with its own slightly different perspective on the data. During prediction, the output of each individual decision tree is combined to obtain the final prediction. Random Forest has several advantages over using a single decision tree. Some of the main advantages are:

1. Robustness: Random Forest reduces overfitting by averaging the predictions of multiple decision trees, which helps to improve the model's generalization performance and makes it more robust to noisy data.

2. Stability: Random Forest is less sensitive to variations in the input data compared to a single decision tree. Small changes in the training data are less likely to result in large changes in the predictions, making the model more stable.

3. Feature Importance: Random Forest can provide estimates of feature importance, which can help identify the most important features for making accurate predictions. This can be useful for feature selection and feature engineering tasks.

4. Handling Missing Values: Random Forest can handle missing values in the input data without the need for explicit imputation, as it can still make predictions using the available features.

5. Scalability: Random Forest can handle large datasets with a large number of features efficiently, making it suitable for a wide range of applications.

Random Forest is widely used in various domains, such as finance, healthcare, image recognition, and natural language processing, due to its flexibility, accuracy, and robustness. However, it is also important to carefully tune hyperparameters such as the number of trees in the forest, tree depth, and other parameters to optimize its performance for a specific problem.

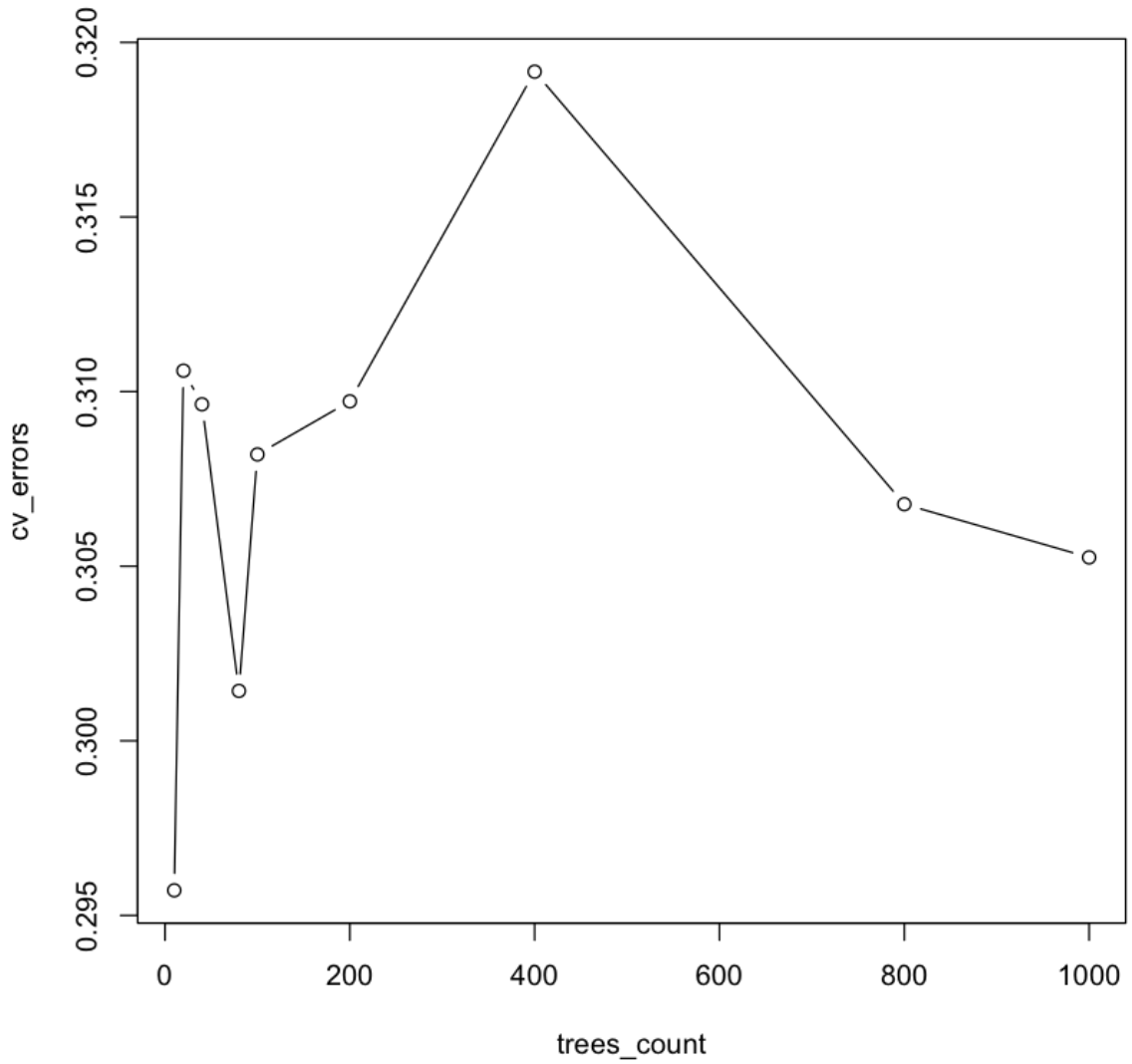
We now implement random forest model for our dataset by fitting it in random forest model and finding the cross-validation error.

```
predictors = dim(airfoil_df)[2] - 1

trees_count = c(10, 20, 40, 80, 100, 200, 400, 800, 1000)


cv_errors = c()

set.seed(1)

for (rf_tree_count in trees_count){


   # Fitting random forest model

   rf_model = randomForest(V6 ~ ., data = airfoil_df, m_try = sqrt(predictors), ntree =
rf_tree_count)


   # Perform Cross Validation

   cv_results = rf.crossValidation(rf_model, airfoil_df, n=5)


   cv_errors = c(cv_errors, mean(cv_results$model.mse))

}
```

Output:

```
running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations

running: regression cross-validation with 5 iterations
```

Now we plot cross-validation errors against trees count.



Now we find the minimum cross-validation error.

```
paste("Minimum CV error:", min(cv_errors), "at number of trees", which.min(cv_errors))
```

Output:

```
[1] Minimum CV error:0.295 at number of trees
```

we got 'Minimum CV error: 0.295 at number of trees 1 and number of predictors used is (-$\sqrt{\text{Predictorscount}}$)

```
set.seed(1)

best_rf_model = randomForest(V6 ~ ., data = airfoil_df, m_try = sqrt(predictors), ntree = which.min(cv_errors)

best_rf_model
```

Output:

```
Call:

 randomForest(formula = V6 ~ ., data = airfoil_df, m_try = sqrt(predictors),      ntree = which.min(cv_errors))

               Type of random forest: regression

                     Number of trees: 1

No. of variables tried at each split: 1


          Mean of squared residuals: 23.3251

                    % Var explained: 50.96
```

Cross-Validation errors for all the methods attempted.

Linear Regression	Linear Regression with subset selection	Ridge Regression	Lasso Regression	Regression Trees	Random Forest
23.37	23.25	23.34	23.23	300087.9	0.295