Load Forecasting

B.M Njuguna

2023-06-26

Contents

1 Variable Checking

Features in the training set may contain nuisance properties such as linear dependency or colinearity which may affect the training process. Such occurences were investigated prior to training the models as follows.

1.1 Correlation

Correlation amongst the features may affect ML models.

```
# Correlation
cor_var2 <- findCorrelation(cor(train[, -c(1:3)]))
```

On checking the correlations, the following variables were correlated with each other;

x
Temparature
Wind Speed

Figure 1 represents the correlation plot for all the features in the data set. Based on the correlation plot, significant correlations among various meteorological variables were observed.

For example, there is a strong positive correlation of 0.73 between the temperature in the "toc" location (T2M_toc) and the temperature in the "san" location (T2M_san). This implies that as the temperature increases in the "toc" location, we tend to observe a corresponding increase in the temperature in the "san" location.

Furthermore, we found a moderate positive correlation of 0.55 between the specific humidity (QV2M_toc) and the total cloud liquid water content (TQL_toc) in the "toc" location. This suggests that an increase in specific humidity is associated with higher levels of cloud liquid water content in the same location.

Additionally, there are other notable correlations such as the negative correlation of -0.41 between the temperature in the "toc" location (T2M_toc) and the wind speed in the "toc" location (W2M_toc). This indicates that as the temperature increases, the wind speed tends to decrease in the same location.

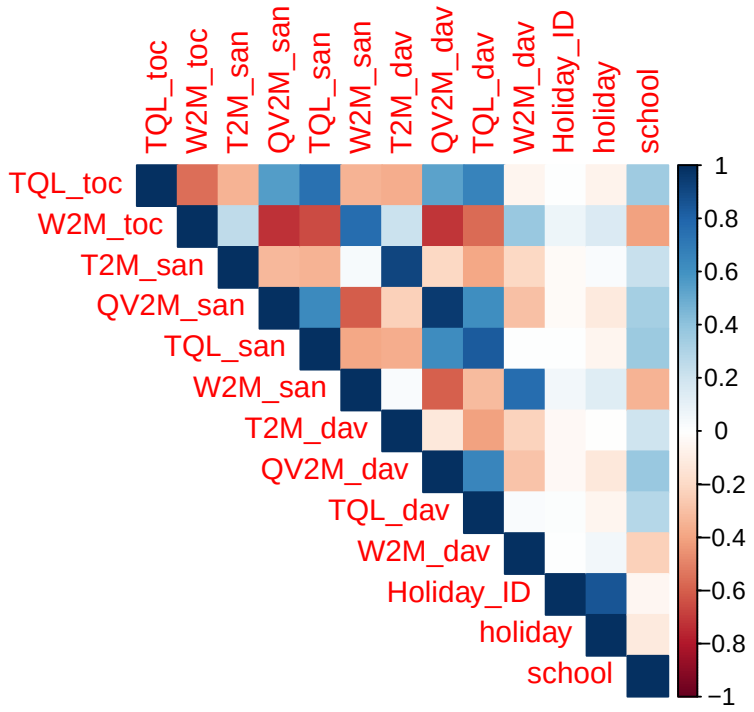


Figure 1: Correlation Plot

To minimize the correlation among variables that measure the same thing for different locations, an approach taken which was to calculate the average of these variables. Specifically, the variables T2M_toc, T2M_san, and T2M_dav, which represent temperature measurements at different locations, were averaged.In addition to averaging the temperature variables (T2M_toc, T2M_san, T2M_dav), other variables that exhibited high correlation and measured the same thing for different locations were also subjected to the averaging process. These variables include:

Specific humidity:

- QV2M_toc: Specific humidity at location Toc
- QV2M_san: Specific humidity at location San
- QV2M_dav: Specific humidity at location Dav

Total column liquid:

- TQL_toc: Total column liquid at location Toc
- TQL_san: Total column liquid at location San
- TQL_dav: Total column liquid at location Dav

Wind speed:

- W2M_toc: Wind speed at location Toc
- W2M_san: Wind speed at location San
- W2M_dav: Wind speed at location Dav

This process aimed to capture the overall representation of the measured phenomena (specific humidity, total column liquid, and wind speed) while reducing the influence of location-specific variations.

By averaging these variables, the focus shifts from individual locations to the collective behavior of the measured phenomena. This helps in reducing multicollinearity and consolidating the information into a single variable, which can be more informative and less redundant for subsequent analyses or modeling tasks. Figure 2 represents the correlation plot after averaging the correlated predictors.

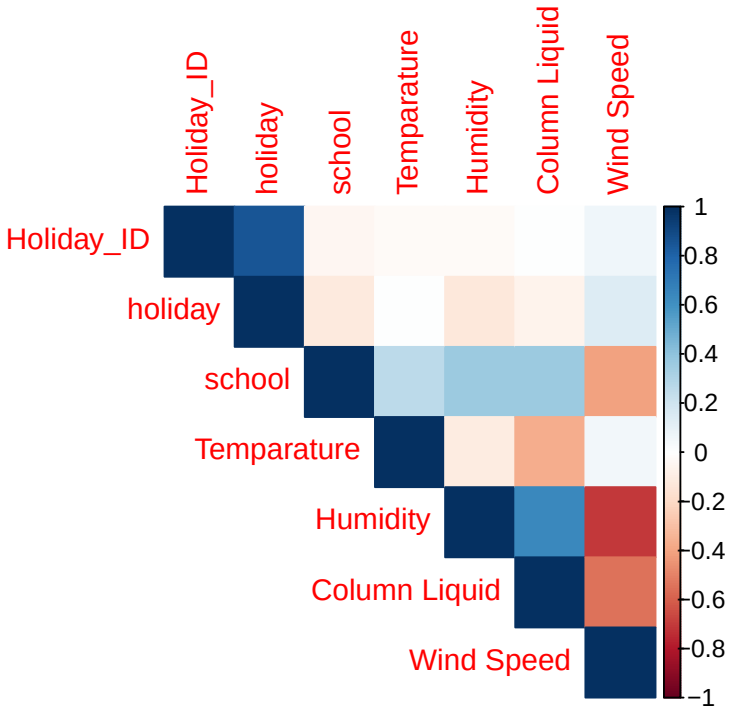


Figure 2: Correlation Plot after removing the correlated features

The only remaining correlation was between the holiday and holiday_ID column as well as column liquid and Wind Speed. However, the correlation amongst these variables was taken care of by further steps that were taken.

1.2 Near zero Variance Features.

Near zero variance features refer to variables in a dataset that have very little or almost no variation across observations. These features have limited or no ability to discriminate or provide useful information for

modeling or analysis purposes. Identifying and removing near zero variance features is important because they can lead to overfitting and adversely affect the performance of machine learning models.In the training set, the variable Holiday_ID was a near varianze feature and hence it was removed from the training set.

```
NZV <- nearZeroVar(train[,-1])
colnames(train[, NZV+1])
```

```
## character(0)
```

1.3 Linear Combination

Linear combination among features refers to a situation where one or more features in a dataset can be expressed as a linear combination of other features. This means that the values of certain features can be obtained by multiplying other features by certain coefficients and summing them together.

Detecting linear combinations among features is important because they can introduce multicollinearity in regression models, leading to unstable parameter estimates and reduced interpretability of the model. To identify and address linear combinations among features, you can use techniques such as variance inflation factor (VIF) and principal component analysis (PCA). Also, there were no linear combinations among the features in the training set.

1.4 Distribution of features

Most of the machine learning models assume that the features in the training set are normally distributed. Thus, it is of essence to check whether the variables are normally distributed. If they are not, then necessary transformation such as *Box-Cox transformation* may be applied to the features to make them less skewed.

Figure **??** represents the holiday variable which was a categorical variable. It was highly imbalanced with more than 80% labelled "0". Figure **??** represents the school variable which was also a categorical variable. This variable was also imbalanced with close to 75% labeled "1". The other variables were continous variables.

Figure **??** shows the distribution of the temperature variable. The histogram shows that the variable was approximately normally distributed with a minimal skweness towards the right. From Figure **??** it clear that humidity was skewed to the left and it had approximately a bi modal distribution. Figure **??** shows that the variable `column liquid` was not normally distributed and had an approximate bi modal distribution. Finally, figure **??** shows that wind speed was right skewed.

To reduce the skewness of the continous variables, the **Yeo-Johnson transformation** was used. Note, the Box-Cox transformation can be used but Yeo-Johnson is preferred since it can handle both negative and positive values. Figure **??** shows an example of feature before transformation and after transformation.

1.5 Centering and Scaling

The features were also centered and scaled.

1.6 Dummy Variables

Dummy variables were created for the categorical variables.

The code below shows all the above steps which were used in preprocessing the data for model training.

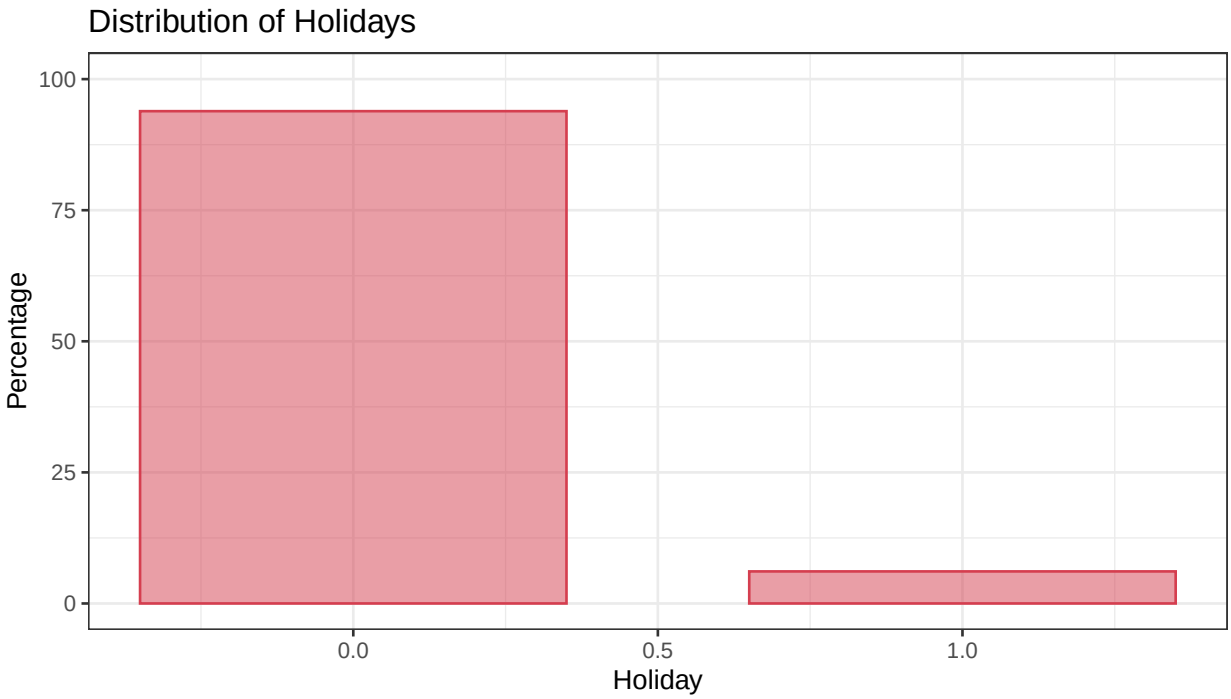


Figure 3: (#fig:figure3)Distribution of Holidays.

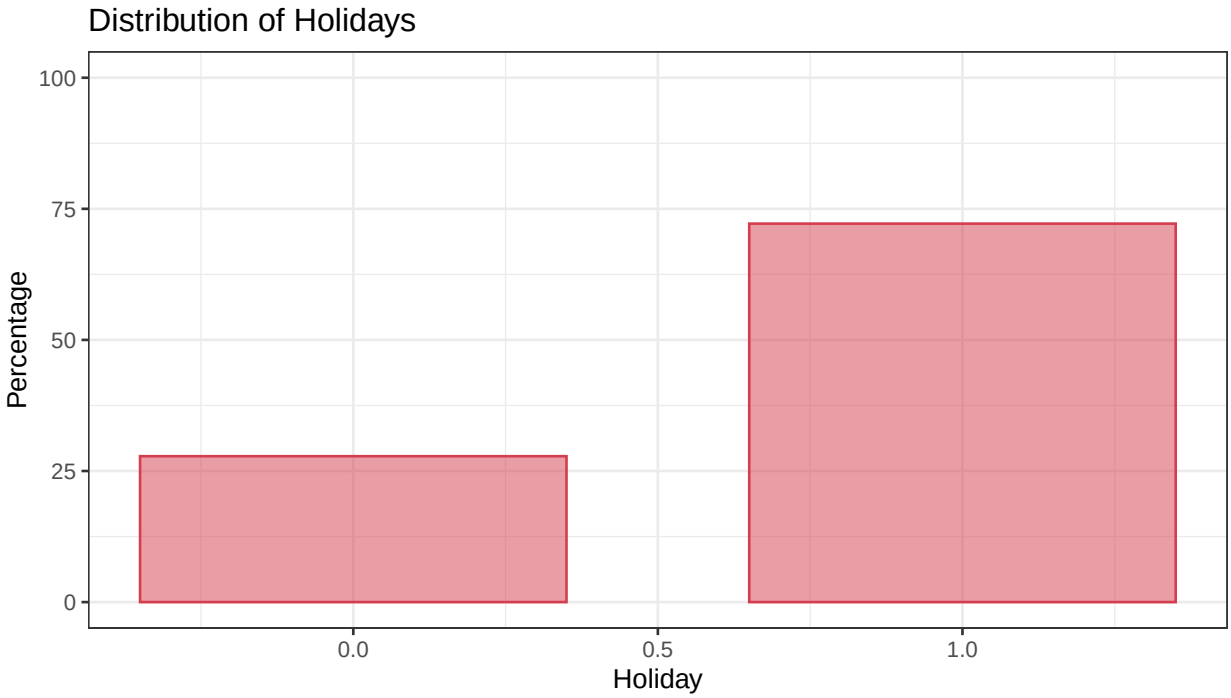


Figure 4: (#fig:figure4)Distribution of Shool feature

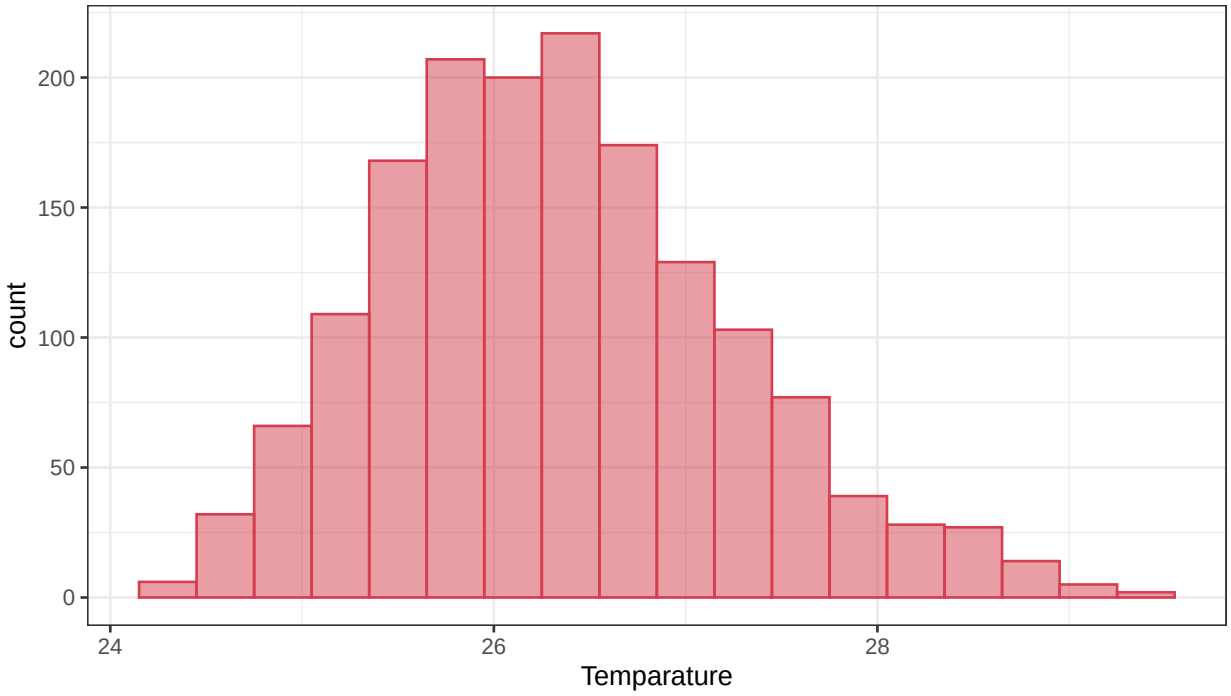


Figure 5: (#fig:figure5)Distribution of Temparature.

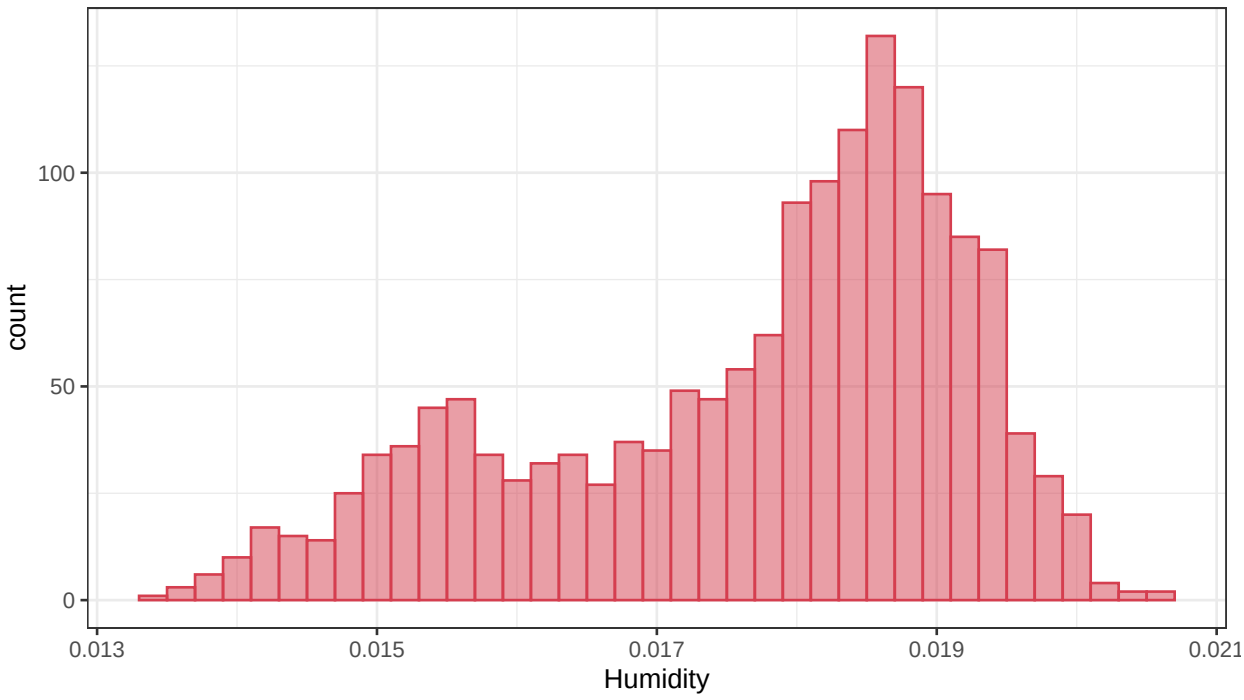


Figure 6: (#fig:figure6)Distribution of Humidity.

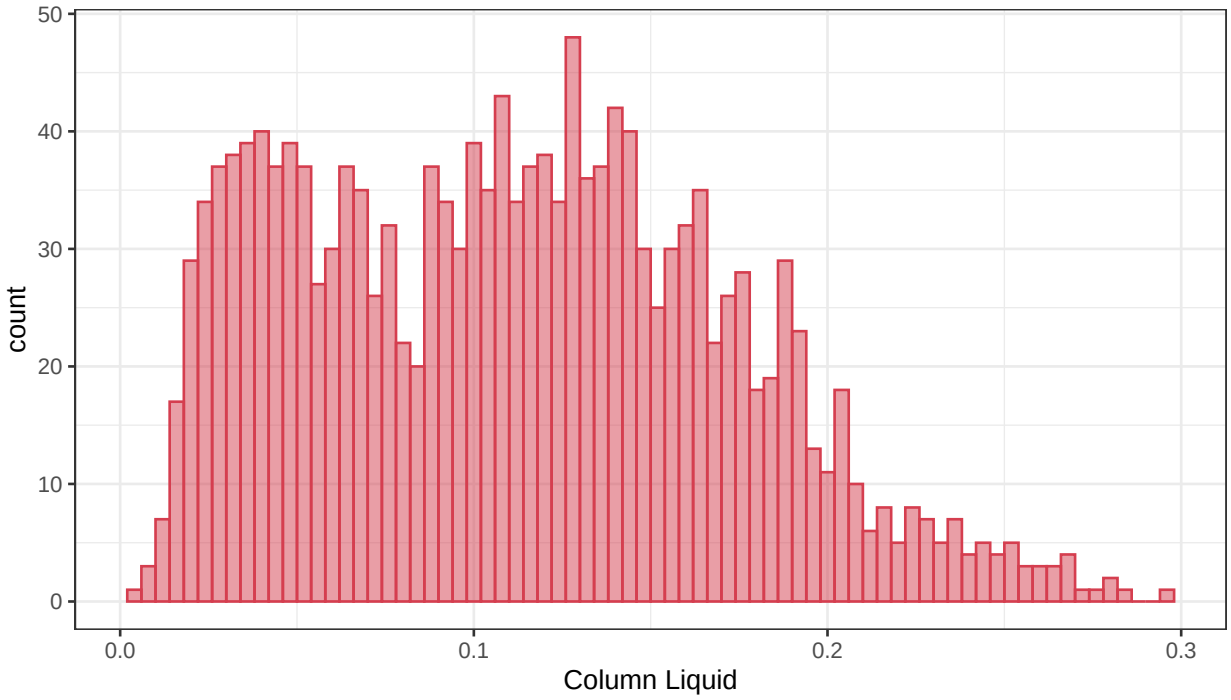


Figure 7: (#fig:figure7)Distribution of Column Liquid

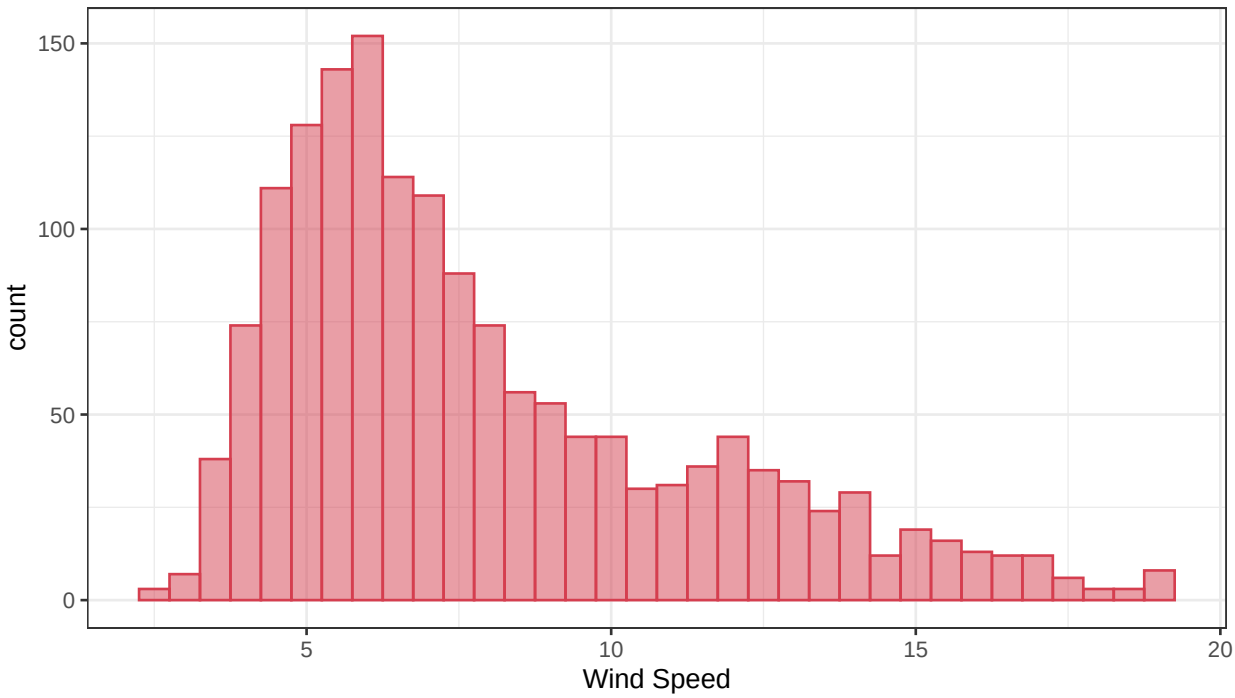


Figure 8: (#fig:figure8)Distribution of Wind Speed

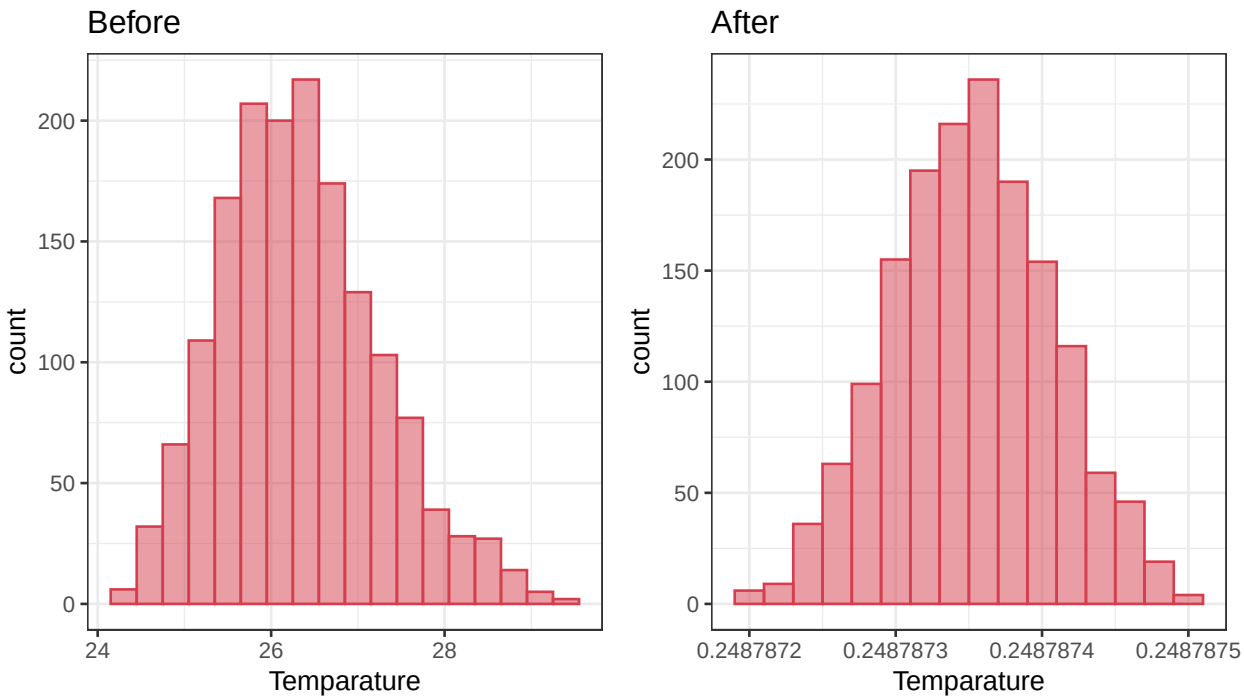


Figure 9: (#fig:figure9)Distribution of Temparature Before and After Transformation

```
train_init <- recipe(nat_demand ~., data = train)

train_step1 <- train_init %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_dummy(all_factor_predictors()) %>%
  step_center(all_numeric_predictors()) %>%
  step_scale(all_numeric_predictors())

train_step2 <- prep(train_step1, training = train)
train.df <- juice(train_step2)
test.df <- bake(train_step2, test)
```

2 Model Training and Parameter Tuning

Model training refers to the process of creating a predictive model using a given dataset. It involves selecting an appropriate algorithm, configuring its parameters, and fitting the model to the training data. The goal is to enable the model to learn patterns and relationships within the data so that it can make accurate predictions on new, unseen data.

Parameter tuning, on the other hand, involves finding the optimal values for the model's hyperparameters. Hyperparameters are settings that are not learned from the data but are specified by the user. Tuning these parameters helps improve the model's performance by fine-tuning its behavior and adapting it to the specific problem at hand.

To train the model and perform parameter tuning, the resampling method was employed using the trainControl function. This function was called with specific settings to configure the model training process. The

method parameter was set to "cv" to indicate the use of cross-validation. Cross-validation involves dividing the dataset into multiple folds to evaluate the model's performance on different subsets of the data.

In this case, the number parameter was set to 10, which means the dataset was divided into 10 folds for cross-validation. Additionally, the savePredictions parameter was set to TRUE to save the predictions made during the cross-validation process.

The verboseIter parameter was also set to TRUE, allowing the display of progress information during each iteration of the training process. This can be helpful to monitor the model's performance and track its progress.

Finally, the returnResamp parameter was set to "all" to ensure that all performance metrics and model results were returned, providing a comprehensive evaluation of the trained model.

By using the trainControl function with these specified settings, the train_ctrl object was created, encapsulating the control parameters necessary for subsequent model training and evaluation.

```
train_ctrl <- trainControl(method = "cv",
                           number = 10,
                           savePredictions = TRUE,
                           verboseIter = TRUE,
                           returnResamp = "all")
```

The models trained are explained below

2.1 1. Random Forest

The random forest model was trained using the following procedure:

Model Configuration: The model was set up to use 10-fold cross-validation (method = "cv") for evaluating model performance. During cross-validation, the data was divided into 10 equally-sized folds, and the model was trained and evaluated 10 times, each time using a different fold as the validation set.

Parameter Grid: The model was trained with different values of the mtry parameter, which controls the number of features randomly sampled at each split of the decision tree in the random forest. The values explored were 2, 4, and 5. This exploration of different parameter values allows for model tuning and selection of the best-performing configuration.

Training the Model: The random forest model was trained using the train function. The response variable nat_demand was predicted using all other variables (.) in the train dataset. The method specified for training was random forest (method = "rf"). The training control parameters were provided by the train_ctrl object, which includes the cross-validation settings. The tuneGrid parameter was set to rf_ctrl, which specifies the grid of parameter values to be used during model tuning.

After the training process, the random forest model (rf_model) was obtained, capturing the knowledge learned from the training data. This model can now be used for various purposes, such as making predictions on new data or evaluating its performance on test data.

```
rf_ctrl <- expand.grid(mtry = c(2,4,5))
rf_model <- train(nat_demand ~.,
                  data = train,
                  method = "rf",
                  trControl = train_ctrl,
                  tuneGrid = rf_ctrl)
```

```
## Random Forest
```

```
##
## 1603 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1442, 1443, 1443, 1443, 1443, 1442, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     86.45528  0.1540638  69.09046
##   4     87.90474  0.1385600  70.36229
##   5     88.24340  0.1342131  70.70716
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

The results of the random forest regressor were as follows;

Model Information: The random forest model was trained on a dataset with 1603 samples and 6 predictor variables.

Pre-processing: No pre-processing steps were applied to the data before training the model.

Resampling: Cross-validation was performed with 10 folds. The summary of sample sizes indicates that each fold had approximately 1442-1443 samples.

Resampling Results: The model performance was evaluated using different values of the mtry parameter. For each mtry value, the Root Mean Square Error (RMSE), R-squared, and Mean Absolute Error (MAE) were calculated.

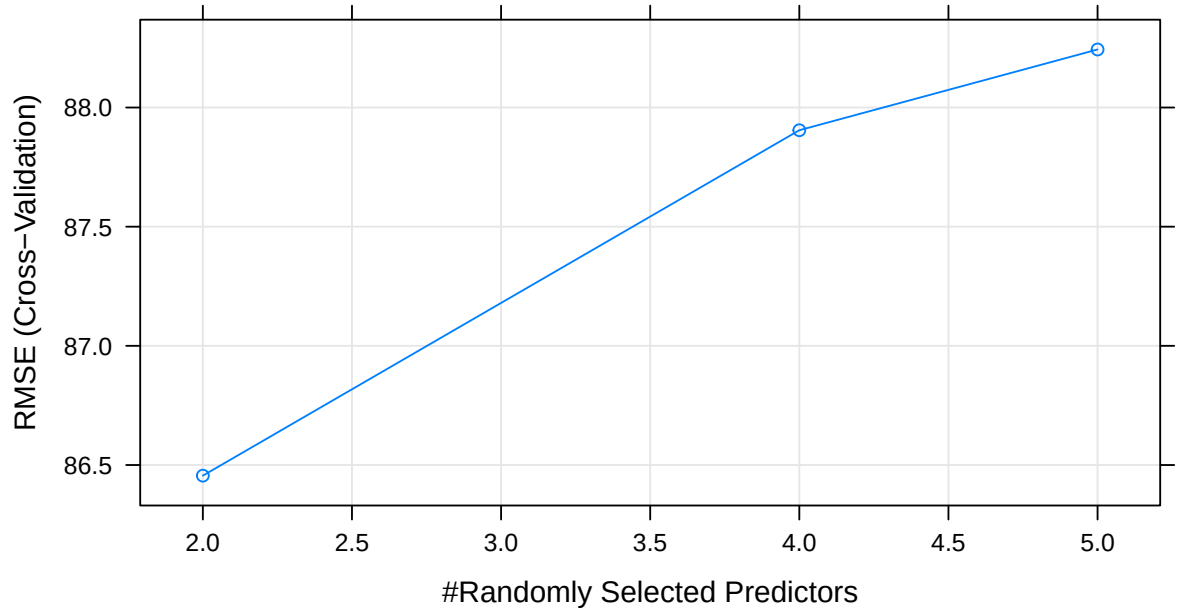
Results Summary: The table provides the performance metrics for each mtry value:

mtry = 2 resulted in an RMSE of 86.45528, an R-squared value of 0.1540638, and an MAE of 69.09046. mtry = 4 resulted in an RMSE of 87.90474, an R-squared value of 0.1385600, and an MAE of 70.36229. mtry = 5 resulted in an RMSE of 88.24340, an R-squared value of 0.1342131, and an MAE of 70.70716. Model Selection: The optimal model was selected based on the smallest RMSE value. Therefore, the final model used for prediction had an mtry value of 2.

In the context of the random forest algorithm, the mtry parameter determines the number of randomly selected predictor variables considered at each split when constructing individual decision trees in the ensemble.

These results provide insights into the performance of the random forest model with different mtry values. The model with mtry = 2 achieved the lowest RMSE, indicating better predictive accuracy compared to the other tested configurations. The R-squared values suggest that the model explains approximately 15% of the variance in the target variable. The MAE values represent the average absolute difference between the predicted and actual values, with lower values indicating better model performance. Figure **??** represents the desirable number of features for training the model.

```
plot(rf_model)
```



On predicting, the random forest regressor had a high RMSE of 85.25119.

```
RMSE(predict(rf_model, test), test$nat_demand)
```

```
## [1] 85.25119
```

2.2 2. Lasso Model

The lasso model was trained in the similar manner to the above model.

```
lasso_ctrl <- expand.grid(alpha = 1,
                          lambda = seq(0, 1, by = 0.1))

lasso_model <- train(nat_demand ~.,
                  data = train,
                  method = "glmnet",
                  trControl = train_ctrl,
                  tuneGrid = lasso_ctrl)
```

The alpha parameter specifies the type of regularization to be used with 0 being lasso model or L1 regularization and 1 being Ridge regression or L2 regularization. In the Lasso model, the lambda parameter is the regularization parameter that controls the strength of regularization applied to the model. It determines the extent to which the coefficients are shrunk towards zero.. The results were as shown

```
lasso_model
```

```
## glmnet
##
## 1603 samples
```

```
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1442, 1443, 1443, 1443, 1443, 1443, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE      Rsquared   MAE
##   0.0     85.74622  0.1698901  68.35895
##   0.1     85.74677  0.1698855  68.35624
##   0.2     85.74833  0.1698494  68.34920
##   0.3     85.75045  0.1698078  68.34174
##   0.4     85.75157  0.1697839  68.33339
##   0.5     85.75225  0.1697765  68.32427
##   0.6     85.75423  0.1697505  68.31536
##   0.7     85.75689  0.1697236  68.30734
##   0.8     85.76100  0.1696757  68.30328
##   0.9     85.76623  0.1696228  68.30302
##   1.0     85.77399  0.1695234  68.30623
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.
```

The R-squared was still low but higher than that of random forest. Figure 10 represents a plot showing the best value for the regularization parameter.

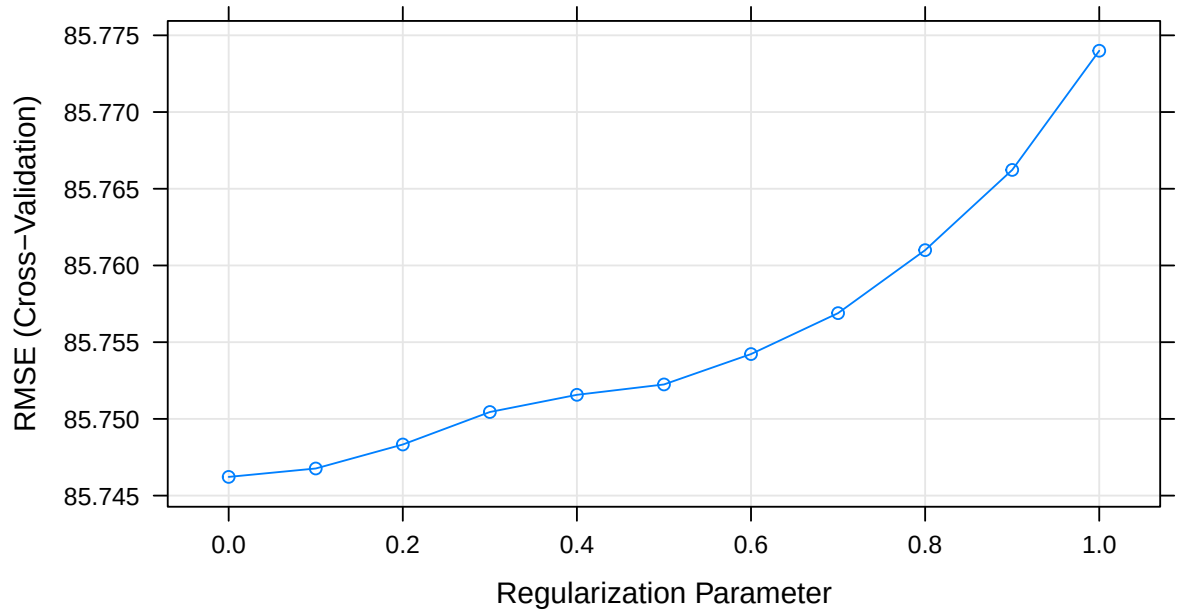


Figure 10: regularization parameter

On predicting, the lasso model had a high RMSE of 8.1809

```
RMSE(predict(lasso_model, test), test$nat_demand)
```

```
## [1] 85.18609
```

3 Conclusion

Both of the above-trained models, namely the Random Forest and Lasso models, exhibit a low R-squared value and high RMSE and MAE scores. These results suggest that the predictors utilized in these models are not effectively explaining the variability in the target variable.

In the case of the Random Forest model, the R-squared value of approximately 0.15 indicates that only about 15% of the variance in the target variable can be explained by the predictors included in the model. Additionally, the high RMSE and MAE scores imply that there is a substantial amount of error between the predicted and actual values.

Similarly, for the Lasso model, the low R-squared value and elevated RMSE and MAE scores reflect the limited capability of the predictors to accurately capture the variability in the target variable. The Lasso model's regularization parameter (lambda) was tuned using a range of values, but it appears that none of the tested lambda values were able to significantly improve the model's performance.

In conclusion, both models' poor performance in terms of R-squared, RMSE, and MAE suggests that the chosen predictors are insufficient in explaining the variation in the target variable. Further analysis or consideration of alternative predictors may be necessary to enhance the models' predictive accuracy.