Creating A Fire Alarm Using Household IOT Sensors

Gabi Rivera, Sahil Wadhwa, and Liam Richardson

Shiley-Marcos School of Engineering, University of San Diego

Table of Contents

Abstract

Traditional fire alarms are extremely useful, but have several shortcomings in terms of reliability. These shortcomings can be reduced by utilizing sensor fusion. To create a secondary fire alarm, we're able to utilize various household IOT sensors to function as a fire alarm. We trained and evaluated several models which would operate on IOT data to function as a fire alarm, these included a logistic regression model, KNN model, a MARS, a neural network and a NSC model. The most effective model ended up being the support vector machine which reported an accuracy of 97.7% on the test set with a notable 0 false positive fire alarms.

Keywords: IOT Sensors, Prediction models, Logistic Regression, NSC, Nearest Shrunken Centroid, KNN, K-Nearest Neighbors, MARS, Multivariate Adaptable Regression Spline, NN, Neural Network, SVM, Support Vector Machine, Fire Alarm

Creating A Fire Alarm Using Household IOT Sensors

Fire alarms are known to save lives, however there are several known issues which interfere with their efficacy. The first problem being false alarms which can make the fire department less likely to respond quickly just based off of a sensor-triggered alarm. The second issue is that very often people allow them to run out of battery and/or intentionally remove the battery due to the high false alarm rate (Iste). Hardware engineers working on IOT sensors have noted that if fused together, the ensemble of many household sensors now present in homes which are placed in thermostats, refrigerators etc could be used as a makeshift fire alarm. This can be useful because having a second "fire alarm" can address both the problems of intermittency, as well as increasing the confidence (Molino-Minero) of a reported fire being legitimate, provided that the artificial fire alarm is tuned properly to reject false positives. Using IOT data which was collected in time synchronous fashion, with a known calibrated fire alarm, as a fire was lit in a room, we were able to train several models to fuse data from IOT sensors to make an ensemble fire alarm.

Method

General Information of Dataset

The sensor-fusion smoke detection classification was retrieved from Kaggle under CCO Public Domain. There were two datasets associated in the package which are labeled *train_dataset.csv* and *test_dataset.csv*. The train dataset was decidedly used over the test dataset due to the presence of outcome variables. Having an original outcome data to compare with the binary classification train and test results is necessary to create a confusion matrix for performance evaluation. There were 5,000 observations, 14 numerical predictors, and a binary outcome that comprised the train dataset. The outcome variable is labeled *Fire.Alarm* with "Yes" and "No" classes pertaining to fire alarm triggered due to the presence of fire and the absence of fire alarm, consecutively. The other features are measurements of air temperature, air humidity, air pressure, total volatile organic compounds, carbon dioxide (CO_2) equivalent concentration, raw hydrogen (H_2) gas, raw ethanol, particulate matter < 1.0 μm (PM1.0), particulate matter 1.0 μm < 2.5 μm (PM2.5), sample counter, timestamp UTC seconds, concentration of particulate

matter < 0.5 µm (NC0.5), concentration of particulate matter 0.5 µm < 1.0 µm (NC1.0), and concentration of particulate matter 1.0 µm < 2.5 µm (NC2.5).

Exploratory Data Analysis

Preliminary data exploratory analysis was used to reduce the predictors down to 7 variables. The first approach was to observe the predictors distribution through a histogram plot followed by a box plot to visualize the outliers as shown in Figures 1-2. A box plot of the fire alarm class distribution was also plotted for each predictor as shown in Figure 3. Skewness score was then calculated to quantify non-normality in the distribution of each predictor as shown in Table 1. There are several predictors that clearly have non-normal distributions that were addressed during data preprocessing.

Most of the predictors removed were assessed through the correlation matrix analysis. The cutoff was set at 0.75 Pearson correlation coefficient score. This left the smoke alarm dataset with the temperature, total volatile organic compounds, CO_2 equivalent concentration, raw hydrogen, raw ethanol, concentration of particulate matter 0.5 µm < 1.0 µm (NC1.0), and concentration of particulate matter 1.0 µm < 2.5 µm (NC2.5). The complete list of predictors had a maximum of 1.0 correlation score while the reduced list of smoke alarm predictors has a maximum of 0.97 correlation score. Figure 4 shows the original dataset correlation plot against the reduced dataset correlation plot.

The second exploratory data analysis step was to extract valuable information about key relationship patterns observed from the entire dataset. A pairwise relationship plot was created to show the general overview of the variable's correlation and distribution defined by the outcome classes. As shown in Figure 5, absence of alarm (No = 0) proportionally has a higher frequency count compared to the frequency of triggered alarms (Yes = 1). This means that the outcome variable is disproportionate and it will need to be rebalanced during data splitting to better represent the minority class of interest.

The proportions of the outcome classes were also explored for each of the remaining predictors as shown in Figure 6. Temperature, raw ethanol, concentration of particulate matter 0.5 µm < 1.0 µm

(NC1.0), concentration of particulate matter 1.0 µm < 2.5 µm (NC2.5) are the features that show a higher proportion of triggered alarms (Yes = 1) over the absence of alarm (No = 0). The other 3 predictors are showing the opposite trend. In this case, it makes sense that triggered alarm class is highly associated with temperature, presence of ethanol, and particulate matter as those are natural indicators of fire or smoke. The same with CO_2's association with the absence of alarm (No =1) as the compound is a fire suppressant. Raw hydrogen and total volatile organic compounds are both highly flammable and so their high association to absence of alarm (No = 0) is unexpected.

Day of the week and hour of the day predictors were also explored to justify the removal of time variables along with the sample counter as unnecessary contributors. Day of the week and hour of the day are showing opposite patterns in the fire alarm class proportion. It is possible that certain days of the week are more strongly associated with triggered alarms (Yes = 1) as opposed to hours of the day being highly associated with absence of alarm (No = 0). This suggests that both features capture different aspects of the time-based pattern in the dataset. Figure 7 shows a follow up plot to visualize the frequency of fire alarm classes per hour of the day. The pattern is showing a bimodal distribution for triggered alarms (Yes = 1) starting early morning and midday ranges. For the day of the week, the frequency of fire alarm classes per day is shown in Figure 8. The result indicates bias in that alarms were triggered most frequently on Thursdays of the week in the month of June 2022.

Data Wrangling and Pre-Processing

During preprocessing, no missing values were identified across the dataset. The next data wrangling step performed was extracting time components from the UTC variable. Month and year were determined to have the same value throughout the UTC variable which was June 2022. These components were omitted along with hours and days later in the process as their temporal pattern contributions might introduce unintended bias into the model. As detailed in the EDA section, predictors with high Pearson correlation at >0.75 score were removed from the data frame to avoid features weighing similar information (Figure 4). A skewness assessment was then performed to measure the asymmetry level of

the feature's probability distribution. This is followed by the Box-Cox transformation to reshape non-normal predictors to satisfy normality assumption of statistical models and stabilize variance. As shown in Table 1, none of the skewed predictors benefited from Box-Cox Transformation and so all the variables were kept to their original shapes. The last transformation that was applied is Center & Scale to standardize the features for increased comparability and accommodate for algorithms that are especially sensitive to the scale of the data. The train and test set predictors are then transformed to have a mean of zero and standard deviation of 1.

Table 1

Skewness Before and After Box-Cox Transformation (acceptance<0.5)

Property	Original Skew	Skew after BoxCox	Lambda
Temperature.C.	-0.303	-0.3035	NA
TVOC.ppb.	4.248	-1.8453	0.1
eCO2.ppm.	8.395	2.4265	-2.0
Raw.H2	-2.335	-2.0295	2.0
Raw.Ethanol	-2.229	-1.8972	2.0
NC1.0	10.232	2.6113	-0.1
NC2.5	13.103	4.0405	-0.2

Data Splitting

Data splitting was performed by partitioning the preprocessed data set into 80% training set and 20% test set. Then the fire alarm outcome of the training set was evaluated for class imbalance. There was 2,900 absence of alarm (No = 0) count against 1,101 triggered alarms (Yes = 1) count. The major class has around 2.6 times more instances than the minor class. To correct for this class imbalance, random over-sampling technique was employed to generate a synthetic balanced train set by over-sampling the minority class (Yes = 1) and under-sampling the majority class (No = 0). This technique gives equal importance to both classes during modeling and gives the minority class of interest the opportunity for better representation. Table 2 shows the fire alarm class distribution result of the train set after the

application of random over-sampling. Reduced, transformed, and rebalanced train set was then fed into selected models for training and tuning. Reduced and transformed test set was later used to assess the performance of each model created.

Table 2

Rebalanced Fire Alarm Outcome Variable

Rebalanced Fire Alarm Counts	
Var1	Freq
No	2010
Yes	1991

Model Strategies

We created several different models in pursuit of the best performance. The models explored were: Logistic Regression, Nearest Shrunken Centroids, a Neural Network, a Support Vector Machine, K - Nearest Neighbors, and a Multivariate Adaptable Regression Spline. The format and tuning methodology for each model is as follows.

Nearest Shrunken Centroids

Area under the ROC was used as the optimization metric, and we tried 30 different values of shrinkage, between 0 and 25. The optimal shrinkage value came out to be 11. NSC works by computing the centroid of each datapoint, and then shrinking these centroids towards the overall mean to reduce noise; new rows are then classified into the class whose centroid is closest to theirs.

Logistic Regression

Area under the ROC was used as the optimization metric. For this model there were no hyperparameters to be tuned. Logistic regression is used only for binary classification (which is the case with this project) it creates a linear combination of the input features - whose weights are varied in training and then passes the output of that combination to a sigmoid function which squashes the values between 0 and 1.

Neural Network

For this model we attempted several different architectures. Ranging from 4 to 8 neurons in the hidden layer as well as 1-2 levels of the hidden layer. Because this was a classification problem, we opted to use a softmax activation function for the output layer, and a rectilinear activation function in the input layer. The learning rate was also varied with 10 intermediate runs with values starting at .1 and ending at .05. The optimal architecture ended up being a single hidden layer with 8 neurons, and a learning rate of .01 with accuracy being the metric used for optimization. To compensate for the small learning rate we ran one million training iterations.

Support Vector Machine

Accuracy was chosen as the optimization metric, tuning the support vector machine ended up requiring two hyperparameters to be tuned. The cost parameter, which controls how tightly the SVM will attempt to fit the model to the training data, this value was varied from .03 to 16. The sigma value which determines how heavily the model weighs outliers was also varied between .03 and 16. The optimal combination of these ended up being C = 14 and sigma=4. That being said, there were several other combinations which generated very similar performance, so it's likely that for this particular dataset the SVM's performance is not overly sensitive to these hyperparameters.

K Nearest Neighbors

Accuracy was used for optimization, the number of neighbors was explored between 1 & 20, surprisingly K=1 ended up being optimal, with euclidean distance being used to calculate distance to neighbors. KNN works by finding the closest N instances to a data point, and classifying the point as the majority class of its neighbors.

Multivariate Adaptable Regression Spline

Accuracy was used for optimization, the degree of the spline was locked to one which indicates minimal interactions between features, the number of terms was tuned between values of 2 and 38, with 11 terms being the optimal. The MARS model breaks up the data into sections where the relationship

between the input features and target variable is similar, and fits differently weighted logistic regression models to each section.

Validation and Testing

Data validation and testing was performed by using the test data to test out our models that we built based using our training data. This way we would be able to accurately predict how well our models were created as we would be predicting values with data the models had never seen before. We used the predict function to predict the values for the variable at hand, Fire.Alarm for each of the models. This gave us a value of Yes or No for each observation in the test data on whether the fire alarm would go off or not.

Results and Final Model Selection

The metric we decided to use to evaluate our seven models when being tested with the test data was accuracy and we calculated that using a confusion matrix. The accuracies of the seven models can be seen in Table 3 below:

Table 3

Accuracy Metric of Models

Model	Accuracy
NNET (Neural Network)	0.995
SVM (Support Vector Machine)	0.977
PLR (Penalized Logistic Regression)	0.758
KNN (K Nearest Neighbors)	0.755
LR (Logistic Regression)	0.752
NSC (Nearest Shrunken Centroids)	0.733
MARS (Multivariate Adaptable Regression Spline)	0.674

As you can glean from the results, the most accurate models were the neural network model and the support vector machine. Even though the neural network performed better than the SVM in terms of

accuracy, the SVM would most likely be the most useful if deployed as it had 0 false alarms for the positive class. This is crucial because the training data was collected over the timespan of a week. Therefore even a tiny percentage of false alarms when accumulated over a long period of time would be extremely annoying for users. On the other hand, the least effective model was the multivariate adaptable regression spline. Interestingly, the other four models were all very close to each other when it came to accuracy. The reason why SVM may have performed that well is because there is a clear separation between two groups or classes that we are looking at, which is Yes or No for the Fire.Alarm variable. The reason being that SVM works best when there is a clear distinction of two classes in a dataset. On the other hand, the Mars model may have not had the best time here as it is best used when working with high-dimensional data which was not the problem at hand here. It is slightly surprising that the neural network model worked that well as it thrives with data that has high variance but it is a very flexible model so maybe we shouldn't be surprised. In conclusion, it seems like the SVM model is the way to go when dealing with our problem at hand.

Discussion and Conclusion

Fire alarms are things that we don't notice day in and day out but they are much more important than we realize. It plays a crucial role in minimizing injuries and deaths that are unfortunately caused due to fires. If we are able to improve the use of fire alarms by improving the detection rate as well as reducing the false detection rate of fire alarms, then the world will become a much safer place and many more lives will be saved. We hope the two most accurate models that we built in SVM and Neural Net are models that will help find the solution to our problem at hand. However, it won't be that easy as these models were simply tested against a single dataset so it is important that we continue to keep updating and improving our model by testing it against other datasets that we are able to find. The next step should be to continue to test these two models against similar datasets that we are able to find and also to not give up on the other models so quickly as we never know how effective those models could be by just evaluating it on a singular dataset. We hope to bring this important issue to light with our work as we are dealing with the most important issue in the world, the issue of life or death.

Figures

Figure 1

Smoke Alarm Predictors: Data Distribution

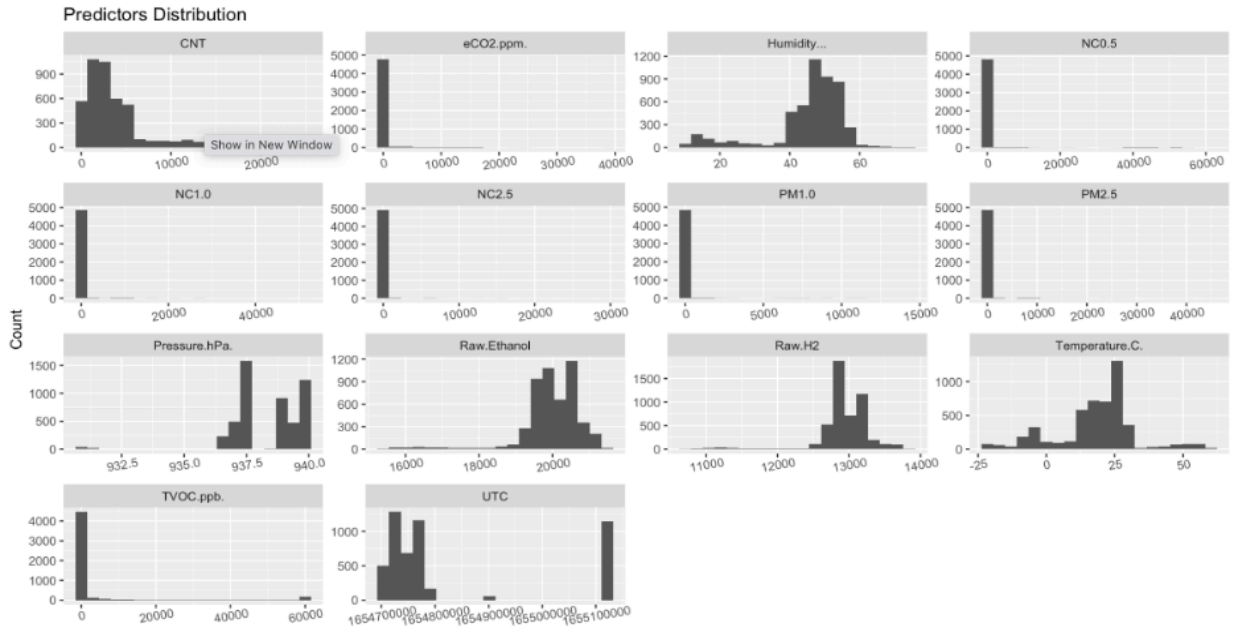


Figure 2

Smoke Alarm Predictors: Box Plots Showing outliers

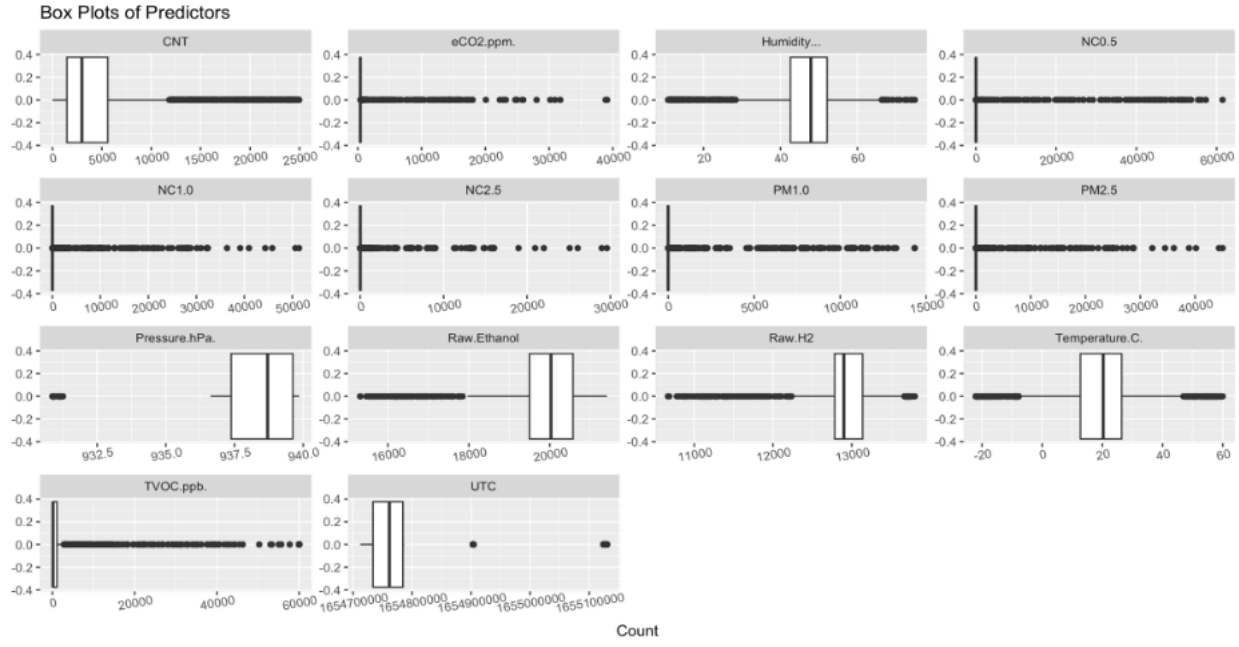


Figure 3

Fire Alarm Outcome Box Plots for each Predictors

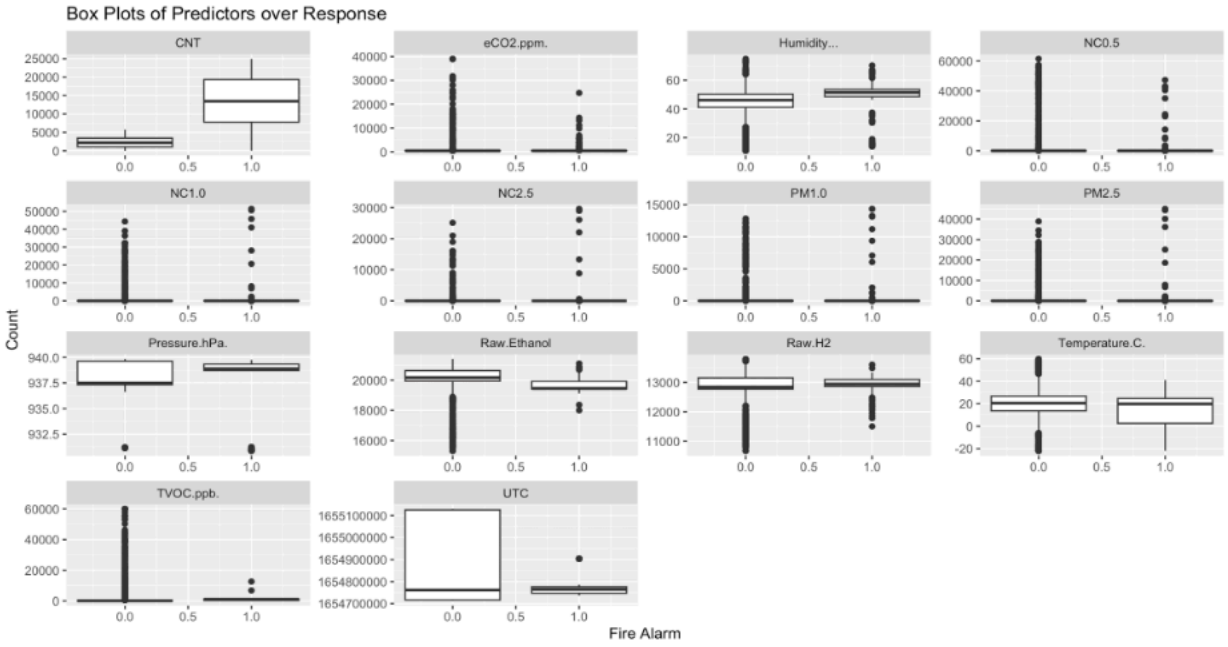


Figure 4

Correlation Plot Comparison: Original vs Reduced Dataset

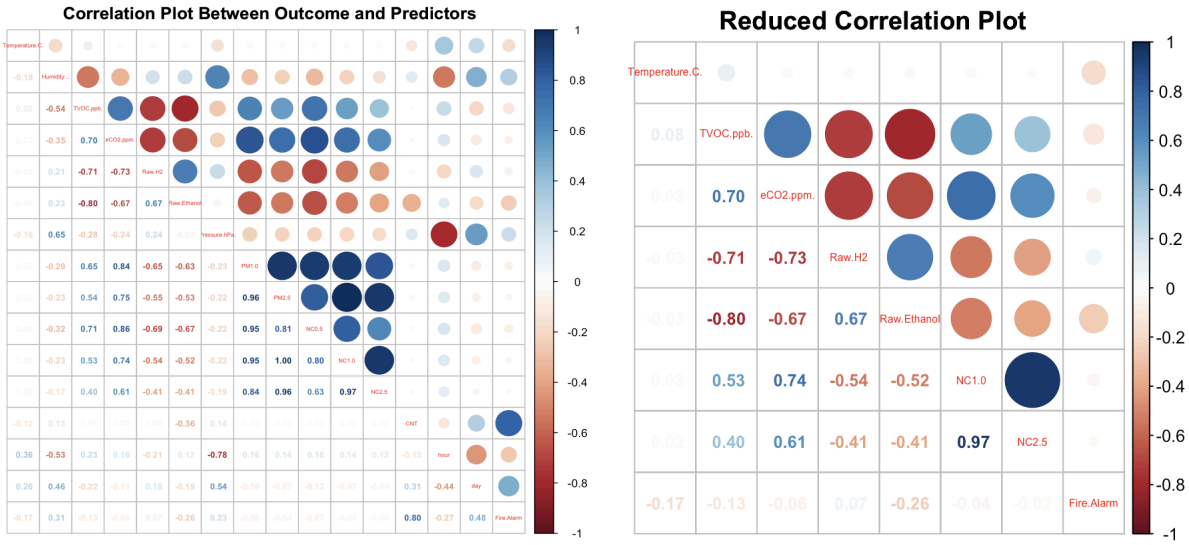


Figure 5

Reduced Smoke Alarm Dataset: Pairwise Relationship Plot

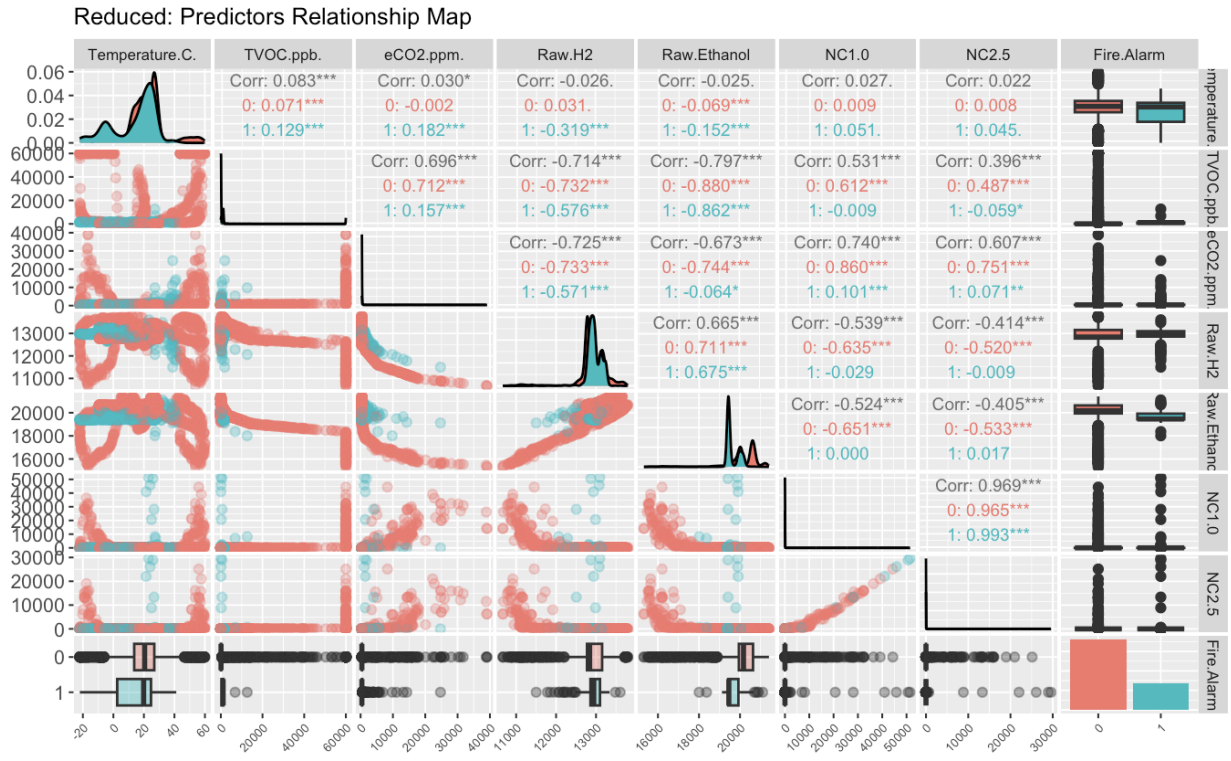


Figure 6

Reduced Smoke Alarm Dataset: Fire Alarm Class Proportion

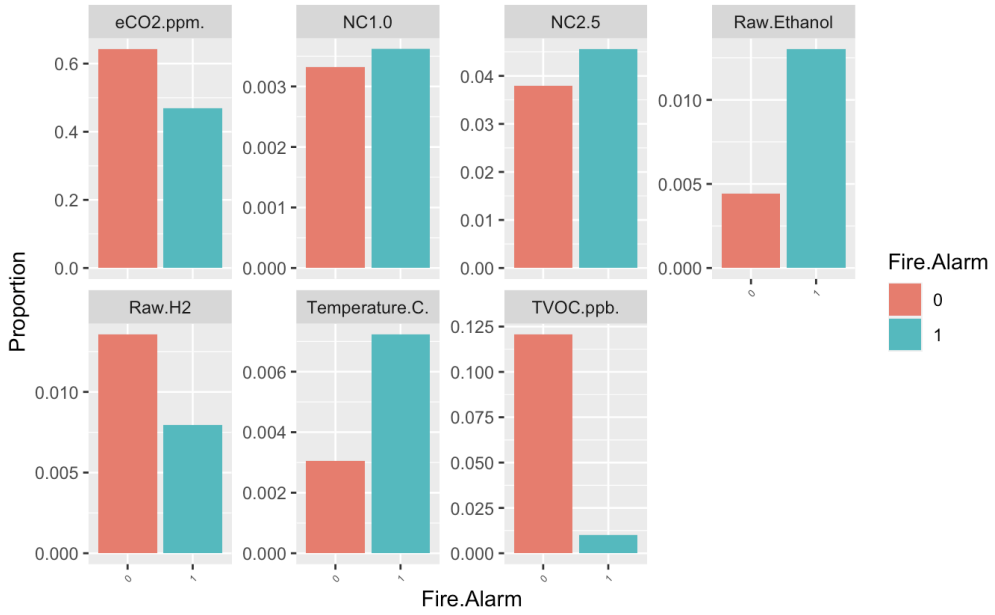


Figure 7

Frequency of Fire Alarm Classes Per Hour

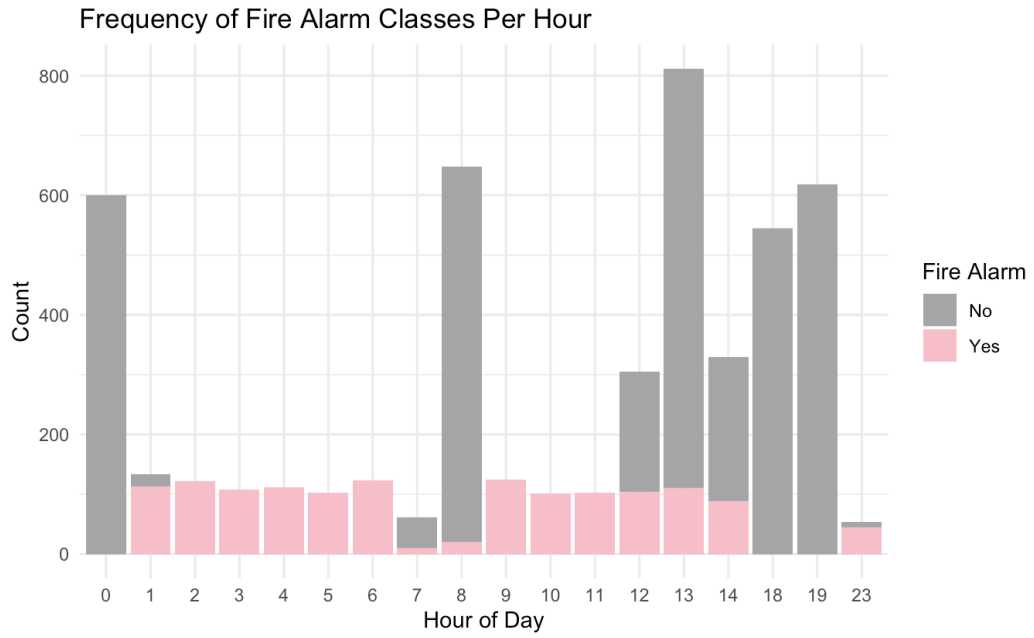
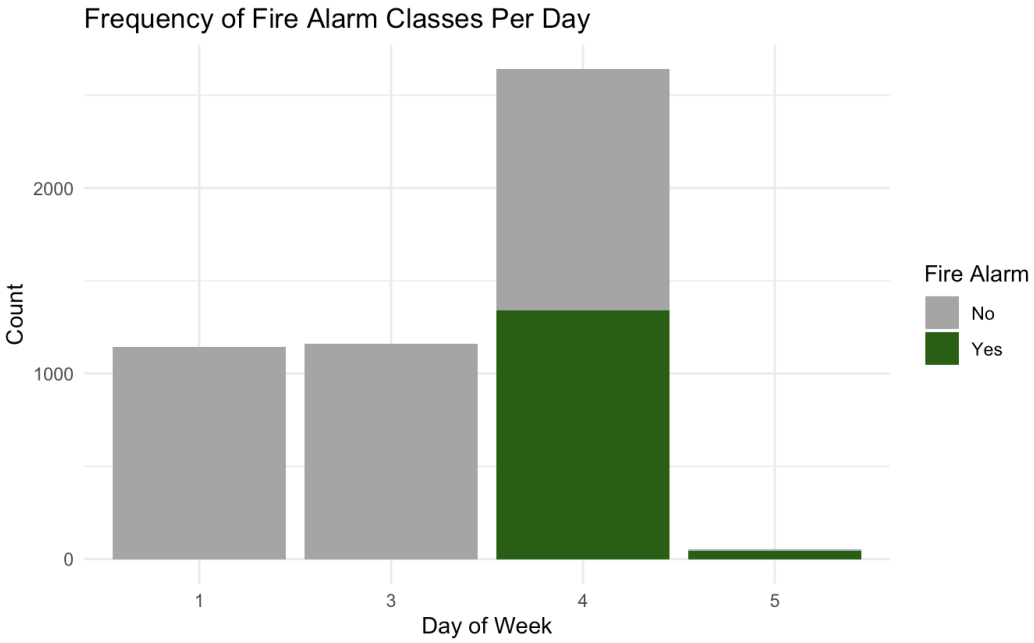


Figure 8

Frequency of Fire Alarm Classes Per Day



References

OpenAI. (n.d.). *ChatGPT*. Retrieved June 01, 2024, from https://www.openai.com/chatgpt

Istre, G. R., & Mallonee, S. (2000). *Smoke alarms and prevention of house-fire-related deaths and injuries*. The Western journal of medicine, 173(2), 92–93. https://doi.org/10.1136/ewjm.173.2.92

Molino-Minero-Re, E., Aguileta, A. A., Brena, R. F., & Garcia-Ceja, E. (2021). *Improved Accuracy in Predicting the Best Sensor Fusion Architecture for Multiple Domains*. Sensors (Basel, Switzerland), 21(21), 7007. https://doi.org/10.3390/s2121700

Appendix

R code attached

Final Project Report Team 7

Gabi Rivera Liam Richardson Sahil Wadhwa

Sensor-Fusion Smoke Detection Classification

The goal is to devise a Machine Learning model that that will detect smoke through the use of IoT data to trigger a fire alarm.

Information about the Features:

Air Temperature
Air Humidity
TVOC: Total Volatile Organic Compounds; measured in parts per billion (Source)
eCO2: co2 equivalent concentration; calculated from different values like TVCO
Raw H2: raw molecular hydrogen; not compensated (Bias, temperature, etc.)
Raw Ethanol: raw ethanol gas (Source)
Air Pressure
PM 1.0 and PM 2.5: particulate matter size < 1.0 μm (PM1.0). 1.0 μm < 2.5 μm (PM2.5)
Fire Alarm: ground truth is “1” if a fire is there
CNT: Sample counter
UTC: Timestamp UTC seconds
NC0.5/NC1.0 and NC2.5: Number concentration of particulate matter. This differs from PM because NC gives the actual number of particles in the air. The raw NC is also classified by the particle size: < 0.5 μm (NC0.5); 0.5 μm < 1.0 μm (NC1.0); 1.0 μm < 2.5 μm (NC2.5);

Pre-Processing:

```
# List of libraries
library(caret)
library(tidyverse)
library(naniar)
library(gt)
library(ggplot2)
library(dplyr)
library(tidyr)
library(GGally)
library(corrplot)
library(e1071)
library(tibble)
library(MASS)
library(mice)
library(reshape2)
library(ROSE)
library(pROC)
library(lubridate)
library(torch)
torch::install_torch()
```

Note:

- Train dataset has 5000 observations, 14 predictors, and an outcome variable.
- Test dataset has 12437 observations and 14 predictors.

```
# Upload datasets
train <- read.csv("train_dataset.csv")
test <- read.csv("test_dataset.csv")

# Use train_dataset moving forward
smokedf <- train
str(smokedf)
```

```
## 'data.frame':    5000 obs. of  15 variables:
##  $ UTC           : int  1655127646 1654734418 1654714047 1654715196 1655125243 165
4712431 1654717409 1655124985 1655130001 1654715540 ...
##  $ Temperature.C.: num  15.1 27.1 26.4 26 -1.2 ...
##  $ Humidity...   : num  43 54.8 45.8 48.4 41.4 ...
##  $ TVOC.ppb.     : int  199 0 144 180 76 58 60000 81 656 223 ...
##  $ eCO2.ppm.     : int  426 400 409 431 400 420 9147 447 400 467 ...
##  $ Raw.H2        : int  12775 13058 12784 12771 12791 12834 11410 12785 13732 1274
7 ...
##  $ Raw.Ethanol   : int  20524 19961 20580 20537 20673 20724 16840 20690 20533 2050
2 ...
##  $ Pressure.hPa. : num  937 940 937 937 938 ...
##  $ PM1.0         : num  1.55 0.21 1.97 1.93 1.9 ...
##  $ PM2.5         : num  1.61 0.22 2.05 2.01 1.97 ...
##  $ NC0.5         : num  10.66 1.46 13.59 13.31 13.08 ...
##  $ NC1.0         : num  1.663 0.228 2.118 2.075 2.04 ...
##  $ NC2.5         : num  0.038 0.005 0.048 0.047 0.046 ...
##  $ CNT           : int  3338 1087 1860 3009 935 244 5222 677 5693 3353 ...
##  $ Fire.Alarm    : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
class(smokedf)
```

```
## [1] "data.frame"
```

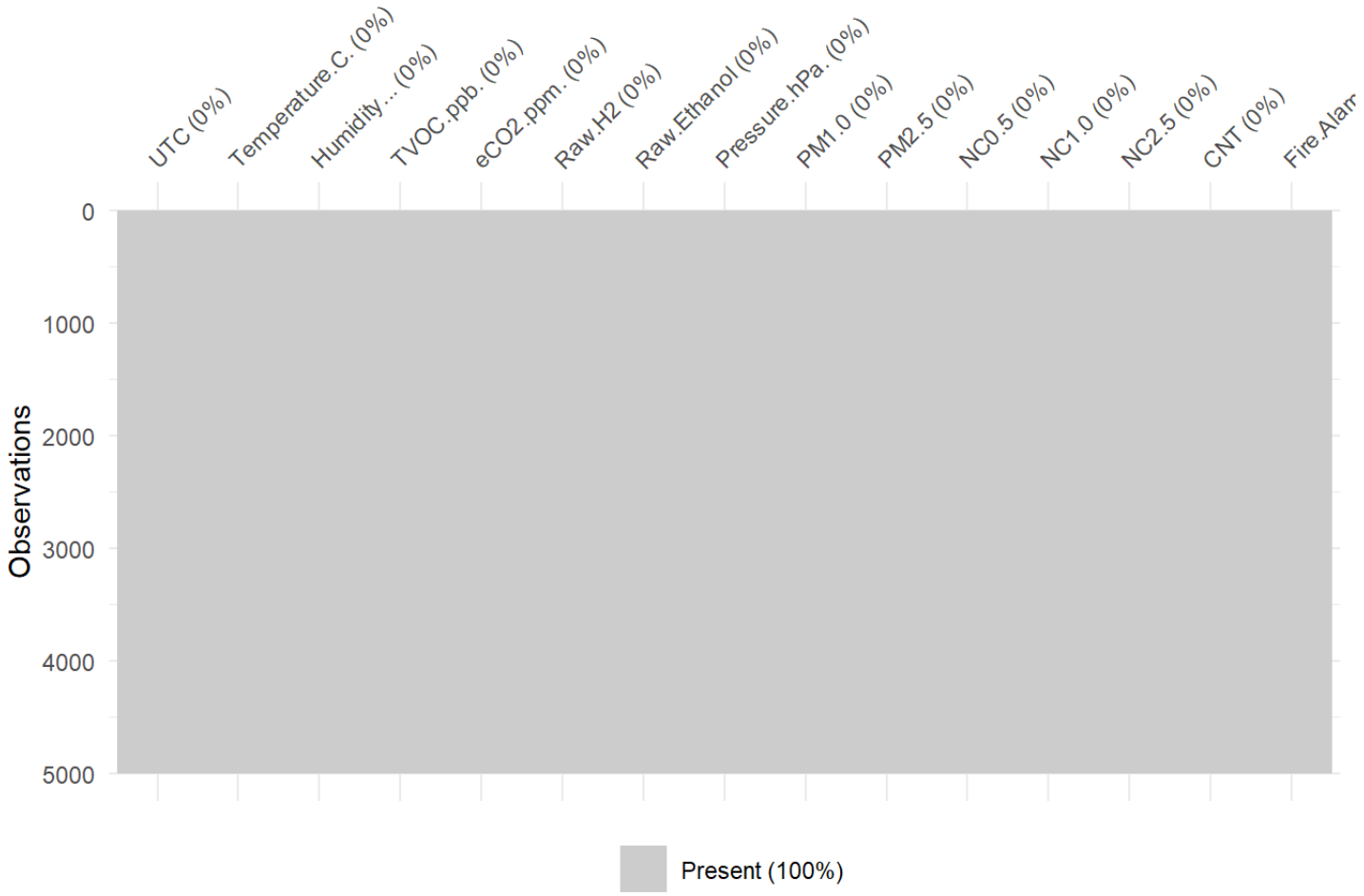
```
#summary(smokedf)
```

```
# Missing Values
smokedf[smokedf == ""] <- NA
na_value <- sapply(smokedf, function(x) sum(is.na(x)))
predictors_with_missing <- names(na_value[na_value > 0])

missing_values_table <- data.frame(Predictor = predictors_with_missing,
Missing_Values = na_value[predictors_with_missing])
missing_values_table |> head() |> gt() |>
  tab_header(title = "Predictors with Missing Values")
```

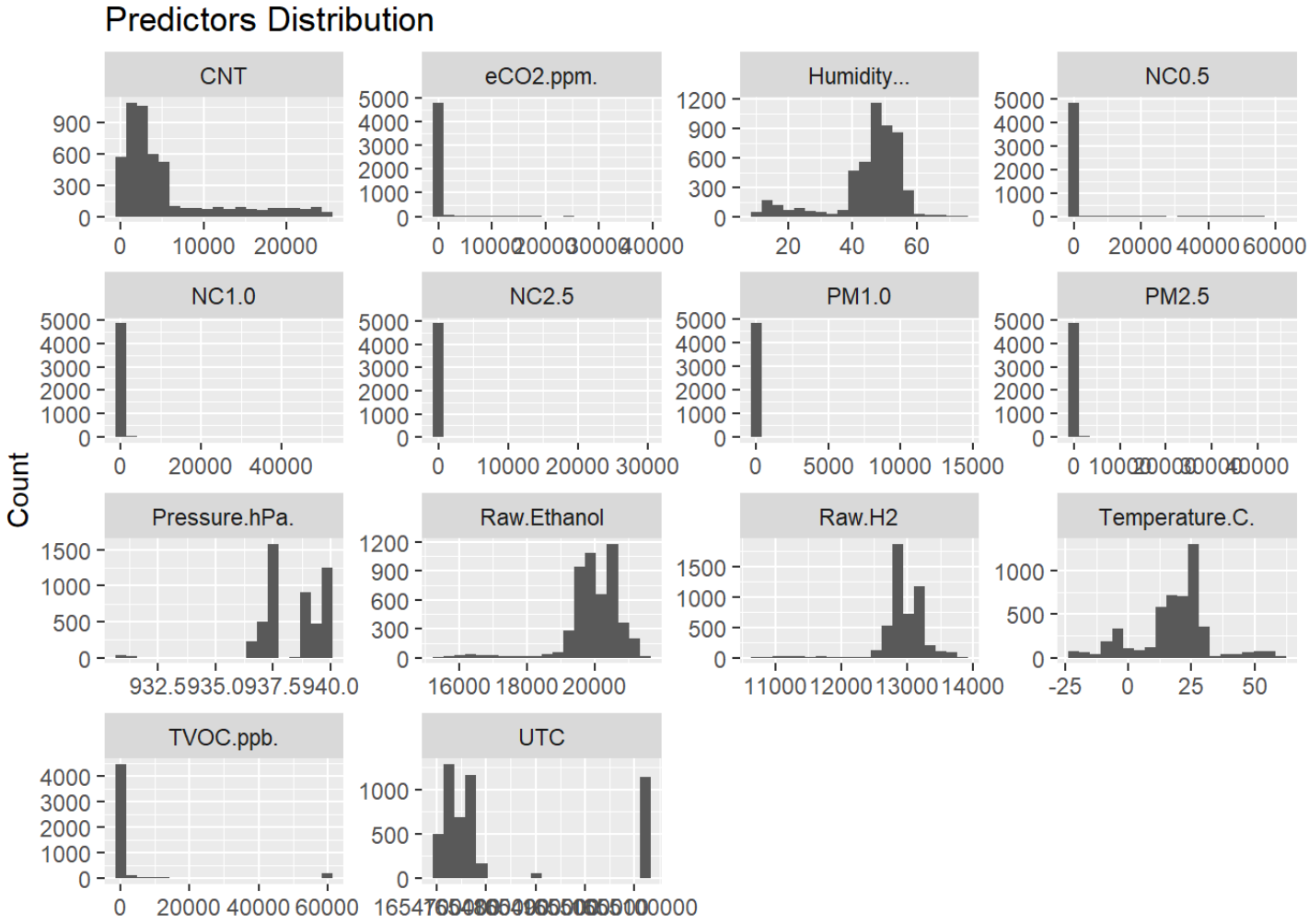
Predictors with Missing Values

Predictor	Missing_Values

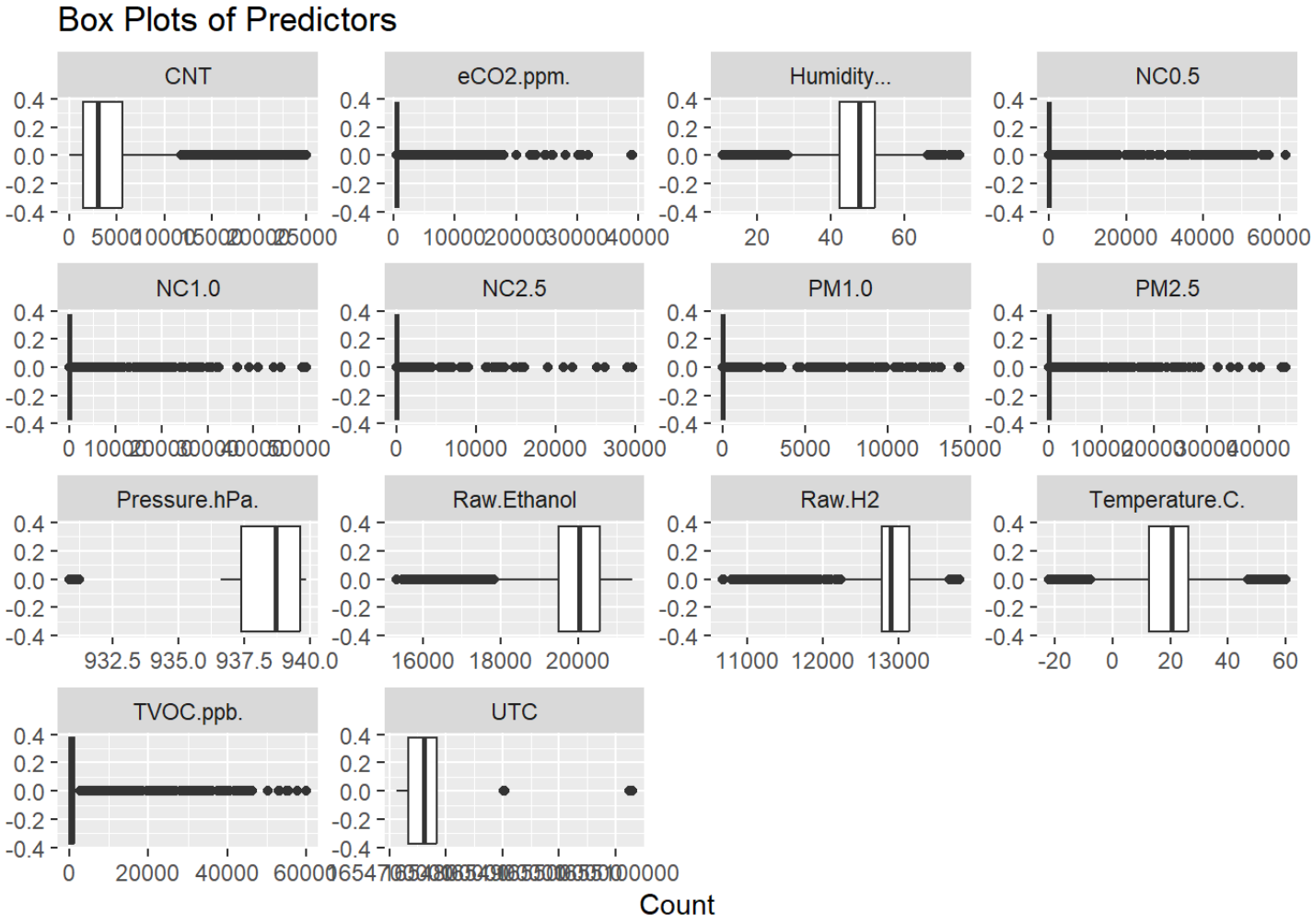
```
vis_miss(smokedf)
```



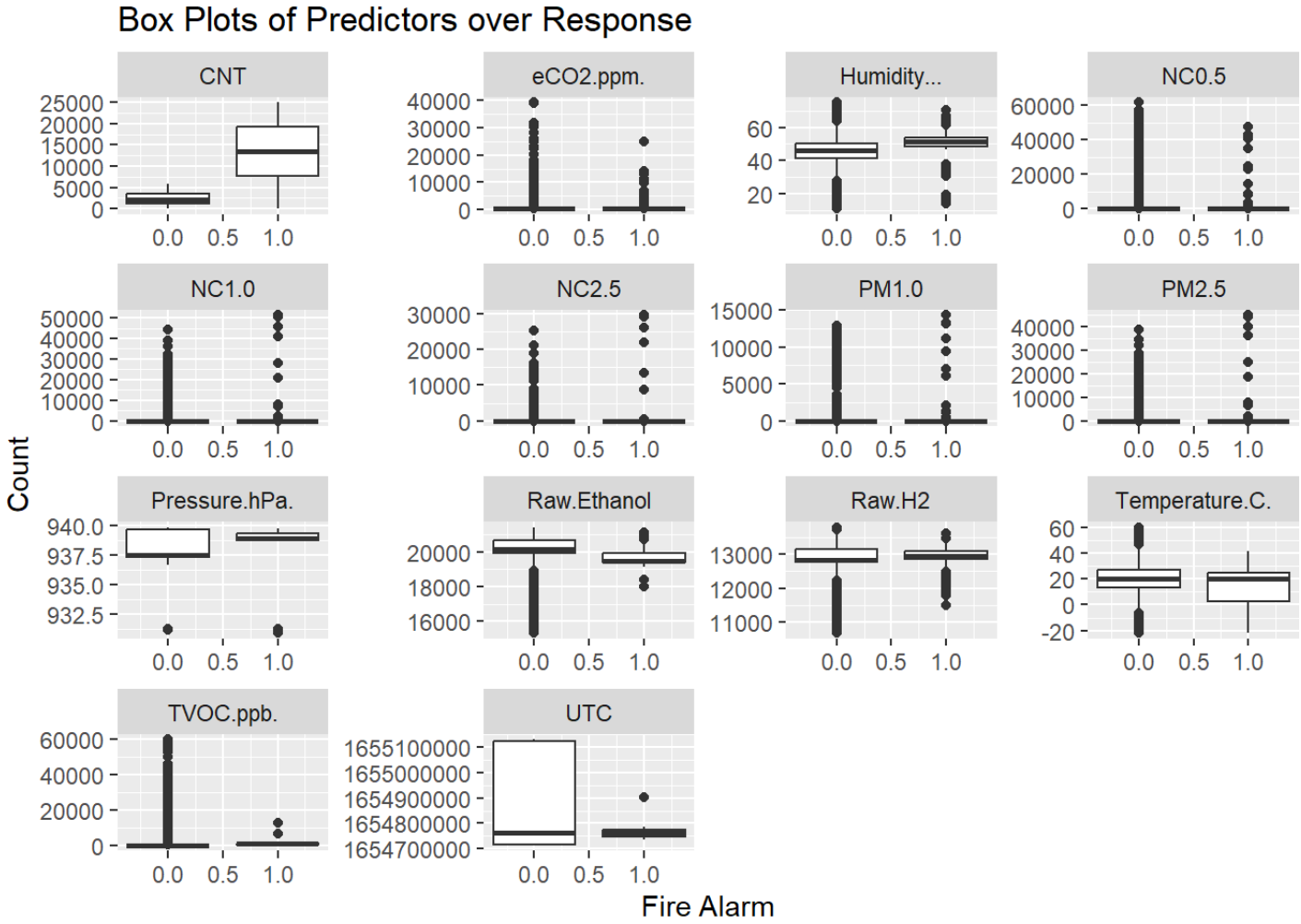

```
# Histogram of each predictors
smokedf |>
  pivot_longer(-Fire.Alarm, names_to = 'Element', values_to = 'value') |>
  ggplot(aes(x = value)) +
  geom_histogram(bins = 20) +
  facet_wrap(~Element, scales = 'free', ncol = 4) +
  theme(axis.text.x = element_text(angle = 0)) +
  labs(title = 'Predictors Distribution', x = NULL, y = "Count")
```




```
# Create box plots for each predictors:
smokedf |>
  pivot_longer(-Fire.Alarm, names_to = 'Element', values_to = 'value') |>
  ggplot(aes(value)) +
  geom_boxplot() +
  facet_wrap(~Element, scales = "free", ncol = 4) +
  theme(axis.text.x = element_text(angle = 0)) +
  labs(title = 'Box Plots of Predictors', x = "Count")
```

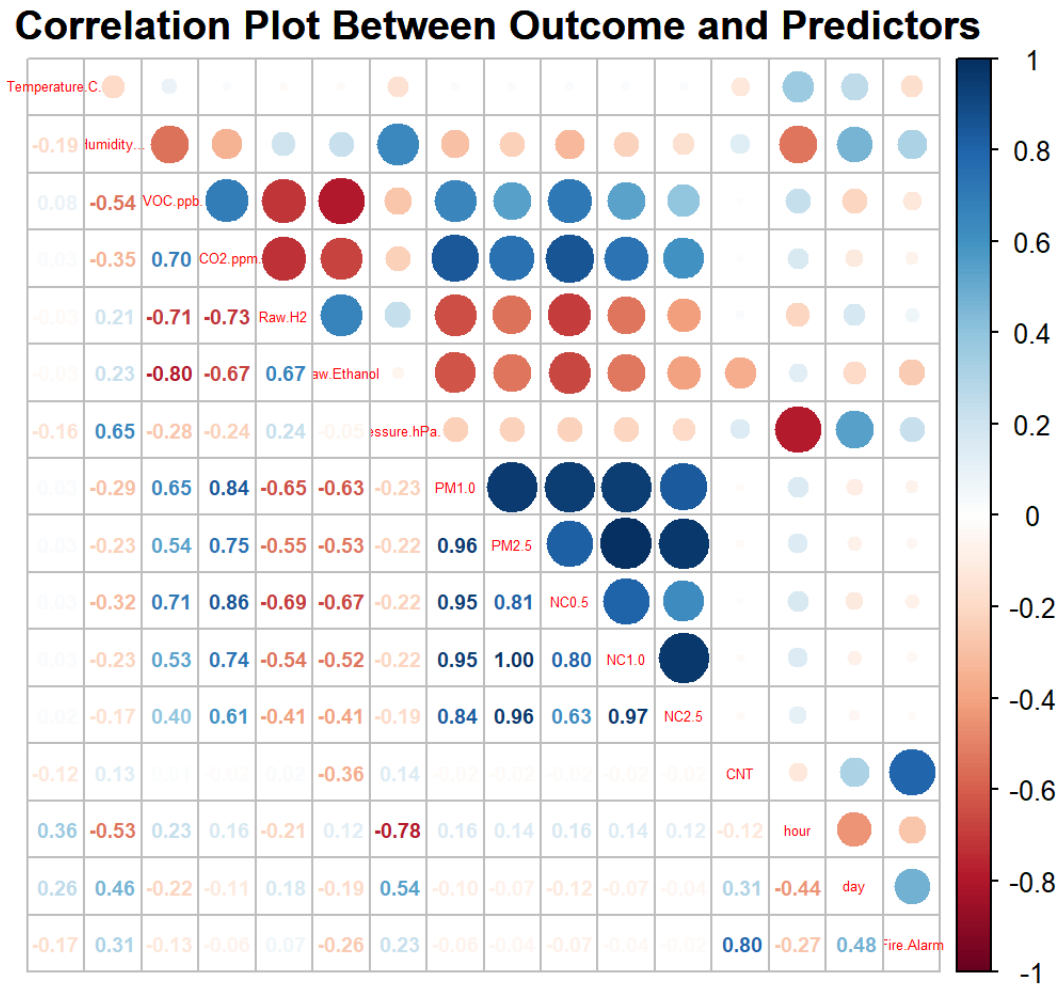



```
# Create box plots of response for each predictors:
smokedf |>
  pivot_longer(-Fire.Alarm, names_to = 'Element', values_to = 'value') |>
  ggplot(aes(Fire.Alarm, value, group = Fire.Alarm)) +
  geom_boxplot() +
  facet_wrap(~Element, scales = "free", ncol = 4) +
  theme(axis.text.x = element_text(angle = 0)) +
  labs(title = 'Box Plots of Predictors over Response', x = "Fire Alarm", y = "Coun
t")
```




```
# Convert UTC format into hours of day and day of week
smokedf$UTC_datetime <- as.POSIXct(smokedf$UTC, origin = "1970-01-01", tz = "UTC")
# all from 2022 and June

smokedf$hour <- hour(smokedf$UTC_datetime)
smokedf$day <- wday(smokedf$UTC_datetime, week_start = 1)

# Reorder variables and remove UTC_datetime
other_columns <- setdiff(names(smokedf), "Fire.Alarm")
smokedf <- smokedf[, c(other_columns, "Fire.Alarm")]
smokedf <- subset(smokedf, select = c(-UTC_datetime, -UTC))
```

```
# Correlation plot including outcome variable
smokedf |>
mutate(Fire.Alarm = as.numeric(Fire.Alarm)) |>
cor() |>
corrplot.mixed(title = "Correlation Plot Between Outcome and Predictors",
tl.cex = .4, number.cex = 0.6, mar = c(1, 1, 1, 1))
```




```
# Remove unnecessary predictors
smokedf_red <- subset(smokedf, select = c(-hour, -day, -CNT))
```

```
# Remove highly correlated predictors
correlation_matrix <- cor(smokedf_red[, -1])
highly_correlated <- findCorrelation(correlation_matrix, cutoff = 0.75)
highly_correlated_names <- colnames(smokedf_red[, -1])[highly_correlated]
smokedf_reduced <- smokedf_red[, -highly_correlated]

# Correlation plot including outcome variable
smokedf_reduced |>
  mutate(Fire.Alarm = as.numeric(Fire.Alarm)) |>
  cor() |>
  corrplot.mixed(title = "Reduced Correlation Plot",
                 tl.cex = .5, number.cex = 0.8, mar = c(1, 1, 1, 1))
```

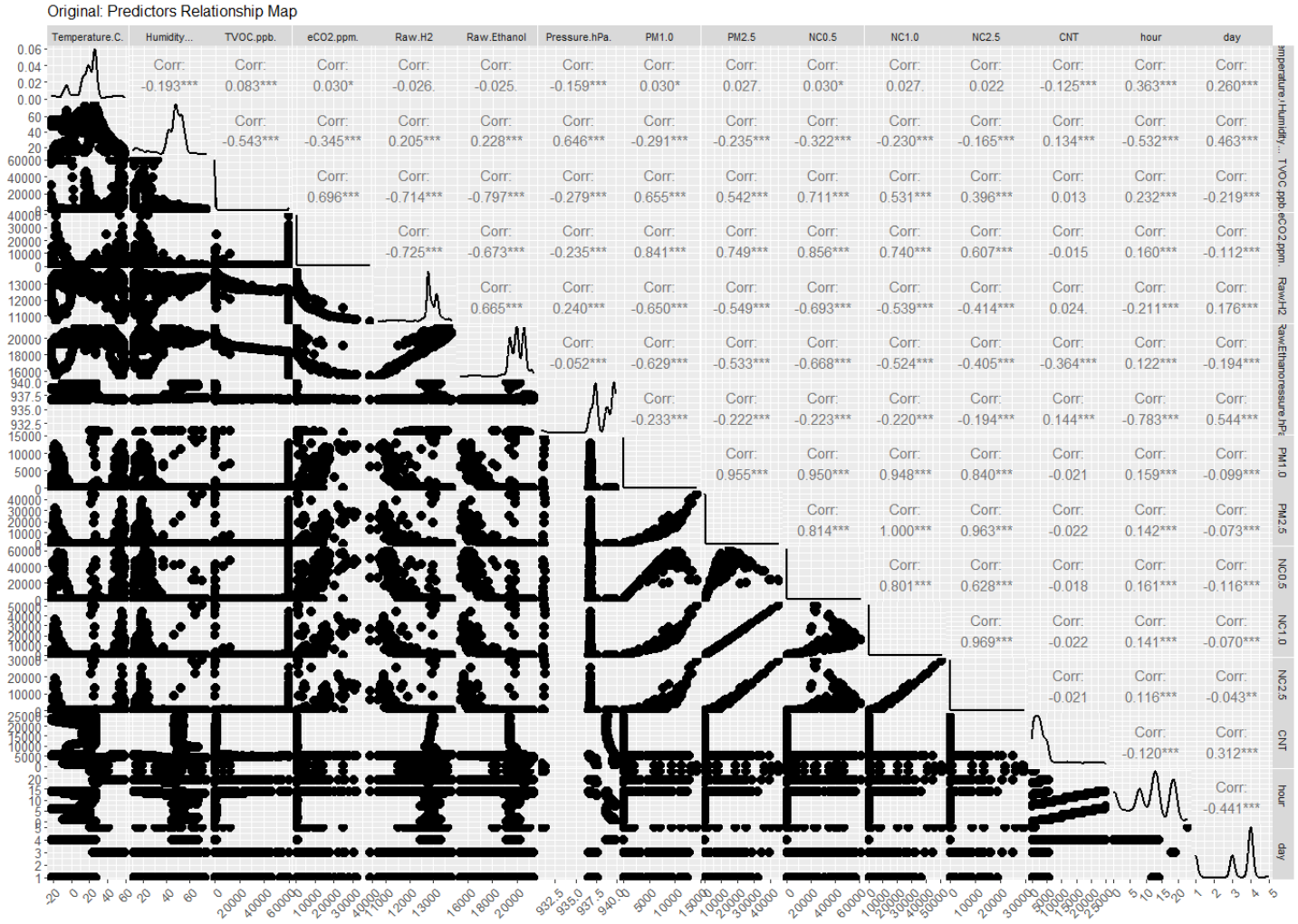

```
# Calculate skewness of each predictors
# Skew values less than ±0.5 should be considered 'normal enough'
skew_fa <- smokedf_reduced |>
  dplyr::select(-Fire.Alarm) |>
  map_dbl(skewness) |> round(3)

skew_fa_tibble <- tibble(Variable = names(skew_fa), Skewness = skew_fa)
#skew_fa_tibble |> gt()

# Determine zero values in the dataframe
zero_counts <- sapply(smokedf, function(x) sum(x == 0, na.rm = TRUE))
```

```
# Perform Box Cox Tranformation Analysis and determine predictors with >50% improveme
nt
bct_test <- function(x, property = 'skew') {
  stopifnot(property %in% c('skew', 'lambda'))

  x <- x[which(! is.na(x))]
  x2 <- x + ifelse(any(x == 0), 0.0001, 0)

  bct <- BoxCoxTrans(x2)
  x_trans <- predict(bct, x2 )
if (property == 'skew') return(e1071::skewness(x_trans))
return(bct$lambda)}

skew_smfa_bct <- smokedf_reduced |>
  dplyr::select(-Fire.Alarm) |>
  map_dbl(bct_test)

bct_analysis <- tibble(
  Property = names(skew_fa),
  `Original Skew` = skew_fa,
  `Skew after BoxCox`= round(skew_smfa_bct,4),
  Lambda = smokedf_reduced |> dplyr::select(-Fire.Alarm) |>
    map_dbl( ~ bct_test(.x, 'lambda')))

bct_keep <- bct_analysis
bct_keep |>
gt::gt()
```

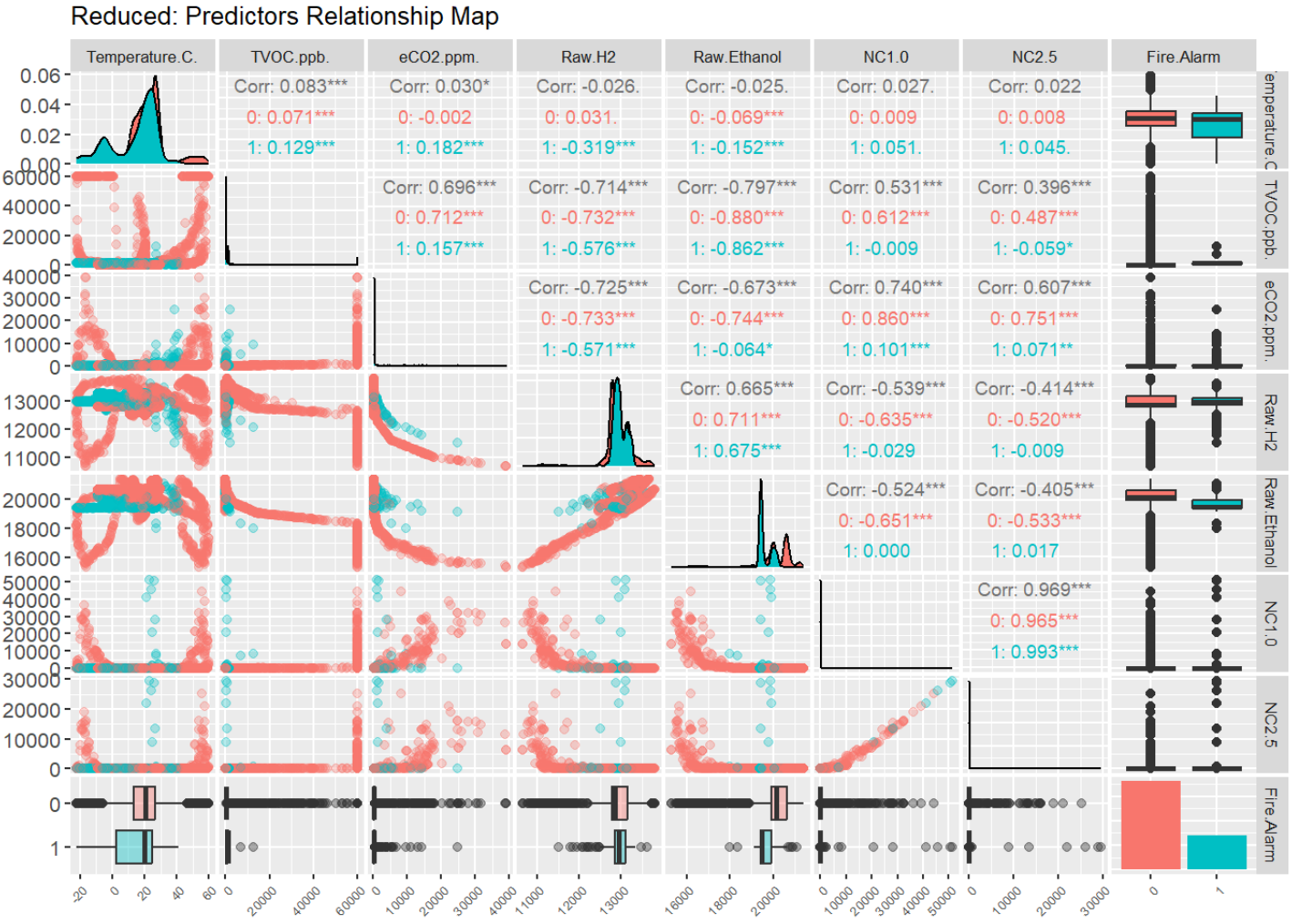
Property	Original Skew	Skew after BoxCox	Lambda
Temperature.C.	-0.303	-0.3035	NA
TVOC.ppb.	4.248	-1.8453	0.1
eCO2.ppm.	8.395	2.4265	-2.0
Raw.H2	-2.335	-2.0295	2.0
Raw.Ethanol	-2.229	-1.8972	2.0
NC1.0	10.232	2.6113	-0.1
NC2.5	13.103	4.0405	-0.2

Exploratory Data Analysis

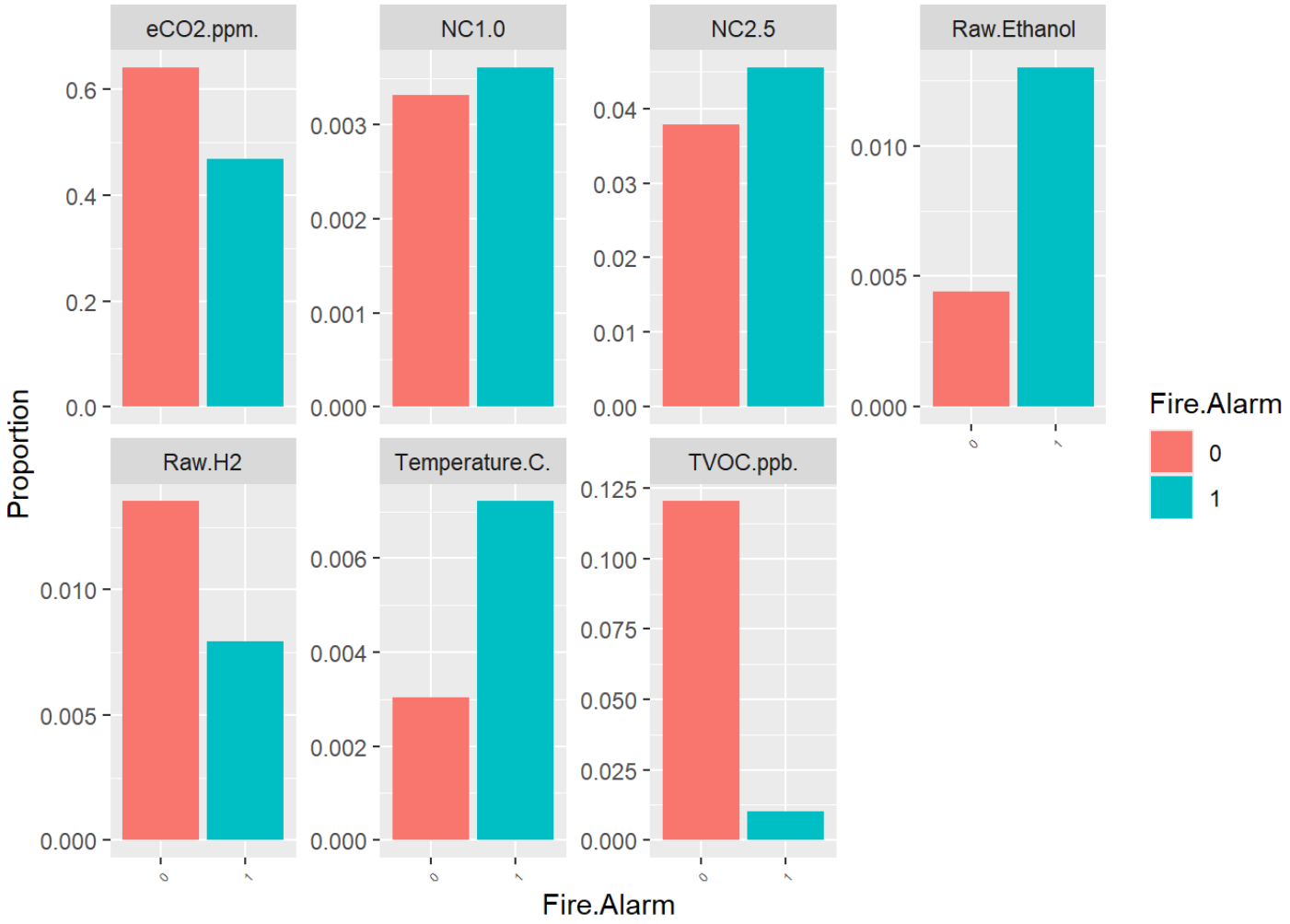
```
# General relationship plot: Multivariate Analysis (Original Data)
smokedf |>
  dplyr::select(-Fire.Alarm) |>
  ggpairs(title = 'Original: Predictors Relationship Map', progress = TRUE,
          upper = list(continuous = wrap("cor", size = 2))) +
  theme_grey(base_size = 5) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 5),
        axis.text.y = element_text(size = 5),
        panel.spacing = unit(0.01, "lines"))
```



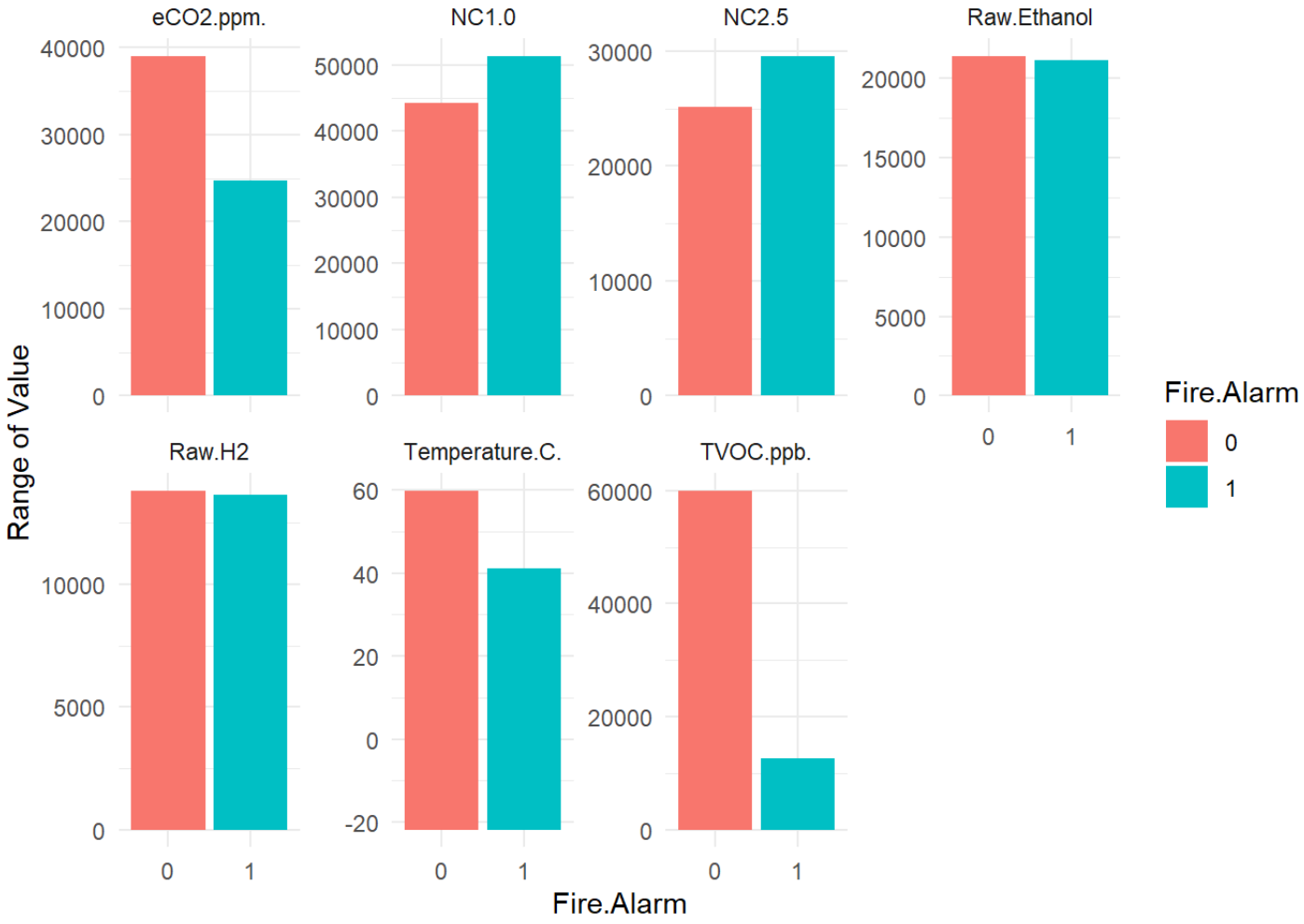
```
# General relationship plot: Multivariate Analysis (Reduced Data)
smokedf_reduced$Fire.Alarm <- as.factor(smokedf_reduced$Fire.Alarm)

smokedf_reduced |>
  ggpairs(title = 'Reduced: Predictors Relationship Map', progress = TRUE,
          upper = list(continuous = wrap("cor", size = 2.5)),
          lower = list(continuous = wrap("points", alpha = 0.3),
                       combo = wrap("box_no_facet", alpha = 0.4)),
          columns = 1:8, ggplot2::aes(colour = Fire.Alarm)) +
  theme_grey(base_size = 8) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 5),
        axis.text.y = element_text(size = 8),
        panel.spacing = unit(0.1, "lines"))
```



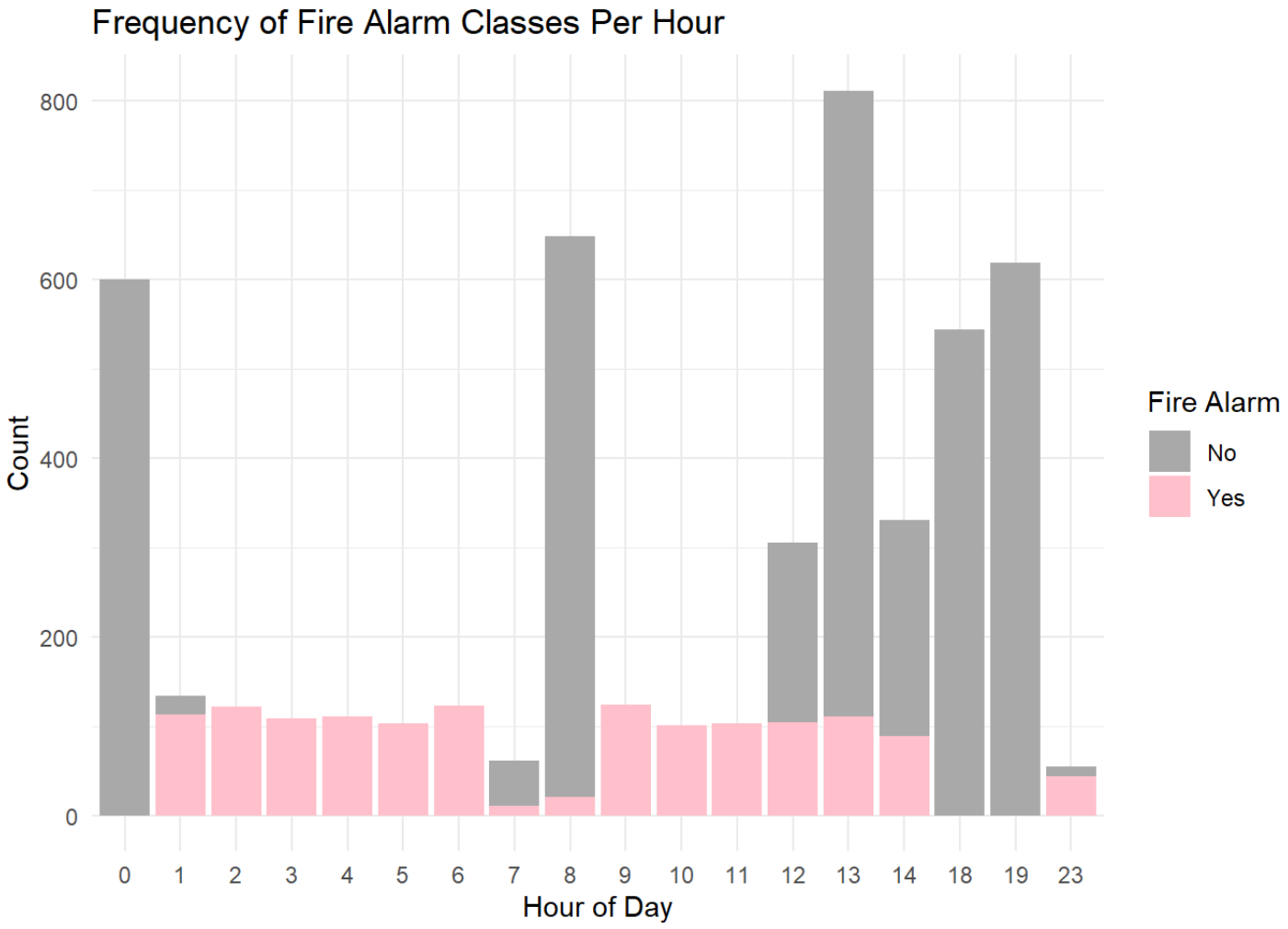
```
# Explore proportion of Fire.Alarm classes against each predictors
proportions <- smokedf_reduced |>
  pivot_longer(-Fire.Alarm, names_to = 'variable', values_to = 'value') |>
  group_by(Fire.Alarm, variable, value) |>
  summarize(Count = n(), .groups = 'drop') |>
  group_by(variable, Fire.Alarm) |>
  mutate(Proportion = Count / sum(Count))

# Create the bar plot
proportions |>
  ggplot(aes(x = Fire.Alarm, y = Proportion, fill = Fire.Alarm)) +
  geom_bar(stat = "identity", position = "dodge") +
  facet_wrap(~ variable, scales = "free_y", ncol = 4) +
  labs(x = "Fire.Alarm", y = "Proportion") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 5))
```



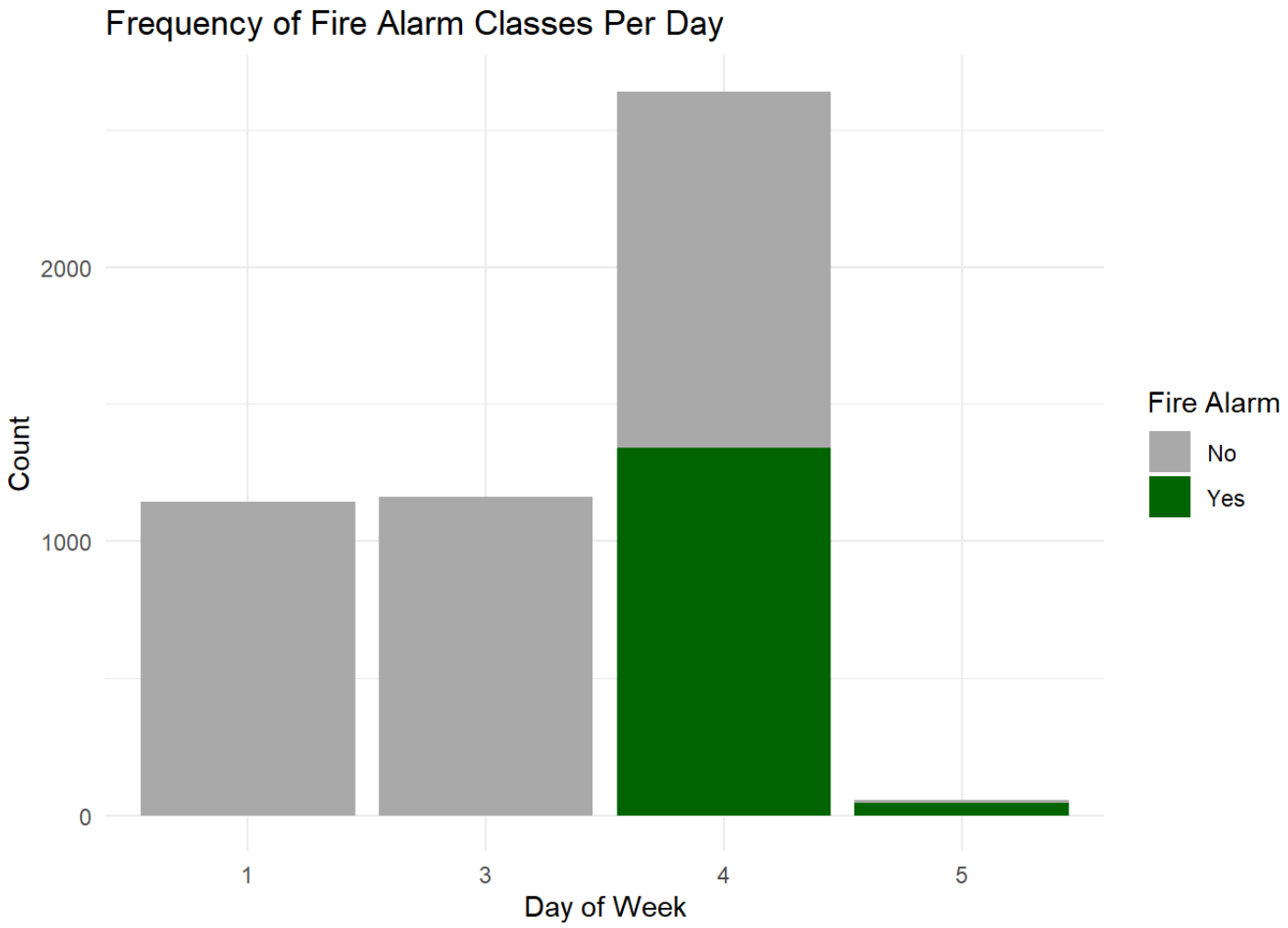
```
# Explore the the value range of Fire.Alarm classes against each predictors
smokedf_reduced |>
  pivot_longer(-Fire.Alarm, names_to = 'variable', values_to = 'value') |>
  ggplot(aes(x = Fire.Alarm, y = value, fill = Fire.Alarm)) +
  geom_bar(stat = "identity", position = "dodge") +
  facet_wrap(~ variable, scales = "free_y", ncol = 4) +
  labs(x = "Fire.Alarm", y = "Range of Value") +
  theme_minimal()
```



```
# Prevelent hour of the day
smokedf$Fire.Alarm1 <- factor(smokedf$Fire.Alarm, levels = c(0, 1), labels = c("No",
"Yes"))

smokedf|>
  ggplot(aes(x = factor(hour), fill = Fire.Alarm1)) +
  geom_bar(position = "stack") +
  labs(title = 'Frequency of Fire Alarm Classes Per Hour', x = "Hour of Day",
       y = "Count", fill = "Fire Alarm") +
  scale_x_discrete(labels = function(x) as.numeric(x)) +
  scale_fill_manual(values = c("Yes" = "pink", "No" = "darkgray")) +
  theme_minimal()
```



```
# Prevelent day of the week
smokedf |>
  ggplot(aes(x = factor(day), fill = Fire.Alarm1)) +
  geom_bar(position = "stack") +
  labs(title = 'Frequency of Fire Alarm Classes Per Day', x = "Day of Week",
       y = "Count", fill = "Fire Alarm") +
  scale_x_discrete(labels = function(x) as.numeric(x)) +
  scale_fill_manual(values = c("Yes" = "darkgreen", "No" = "darkgray")) +
  theme_minimal()
```



Training and Test Sets

```
set.seed(503)
smokedf_reduced$Fire.Alarm <- ifelse(smokedf_reduced$Fire.Alarm == 0, "No", "Yes")
smokedf_predictors <- smokedf_reduced[ ,-1]
smokedf_yield <- smokedf_reduced[ ,1]

# Split the data into training and test sets
smokedf_index <- createDataPartition(smokedf_yield, p = 0.8, list = FALSE)
smokedf_train <- smokedf_predictors[smokedf_index, ]
smokedf_test <- smokedf_predictors[-smokedf_index, ]
```

```
# Center and scaling using preProcess
smokedf_prep <- preProcess(smokedf_train, method = c("center", "scale"))

# Apply the transformation to the training data
smokedf_train_transformed <- predict(smokedf_prep, smokedf_train)
smokedf_test_transformed <- predict(smokedf_prep, smokedf_test)

# Rebalance train dataset
cat("Number of non-triggered fire alarm (No) = ",
    sum(smokedf_train_transformed$Fire.Alarm == "No"), "\n")
```

```
## Number of non-triggered fire alarm (No) =  2900
```

```
cat("Number of triggered fire alarm (Yes) =",
    sum(smokedf_train_transformed$Fire.Alarm == "Yes"))
```

```
## Number of triggered fire alarm (Yes) = 1101
```

```
train_balanced <- ROSE(Fire.Alarm ~ ., data = smokedf_train_transformed)$data

rebalanced <- as.data.frame(table(train_balanced$Fire.Alarm))
rebalanced |> gt() |>
  tab_header(title = "Rebalanced Fire Alarm Counts")
```

Rebalanced Fire Alarm Counts	
Var1	Freq
No	2010
Yes	1991

Model Strategies

Logistic Regression

```
set.seed(503)
ctrl <- trainControl(method = "cv", summaryFunction = twoClassSummary,
                     classProbs = TRUE, savePredictions = TRUE)

# Create Logistic Regression
lrFit <- train(Fire.Alarm ~ .,
               data = train_balanced,
               method = "glm",
               metric = "ROC",
               trControl = ctrl)
# Logistic regression final model using train data set
lrFit$finalModel
```

```
##
## Call:  NULL
##
## Coefficients:
## (Intercept)    TVOC.ppb.    eCO2.ppm.       Raw.H2  Raw.Ethanol        NC1.0
NC2.5
##     -0.7295      -1.8087      -0.1028       0.6705      -2.5635      -0.1305
0.1557
##
## Degrees of Freedom: 4000 Total (i.e. Null);  3994 Residual
## Null Deviance:       5546
## Residual Deviance: 3934  AIC: 3948
```

```
lrFit
```

```
## Generalized Linear Model
##
## 4001 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3601, 3601, 3601, 3601, 3600, 3601, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8330091  0.7323383  0.8302563
```

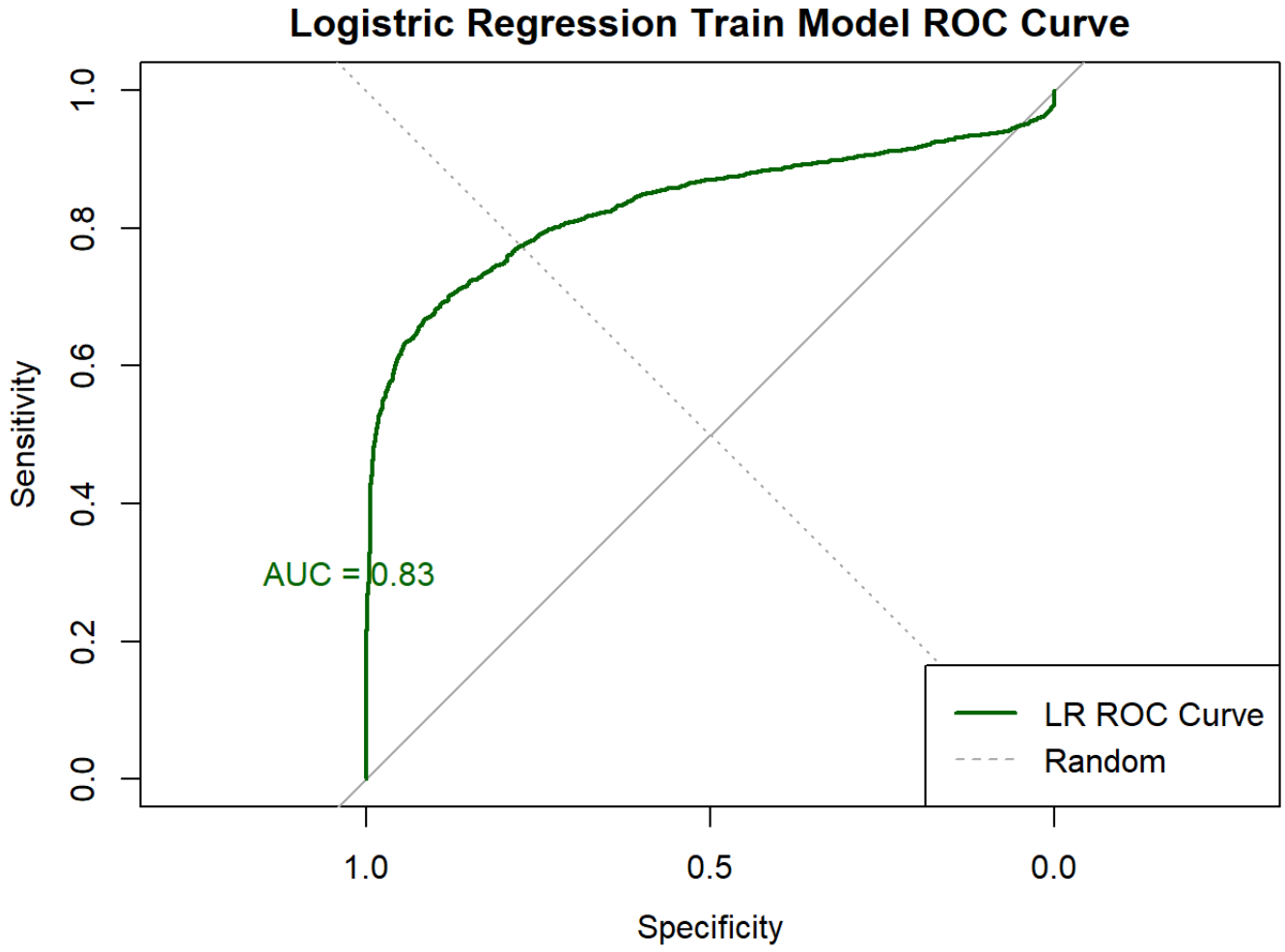
```
# Confusion matrix
lrCM <- confusionMatrix(lrFit, norm = "none")
lrCM
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##           Reference
## Prediction   No  Yes
##        No  1472  338
##        Yes  538 1653
##
##  Accuracy (average) : 0.7811
```

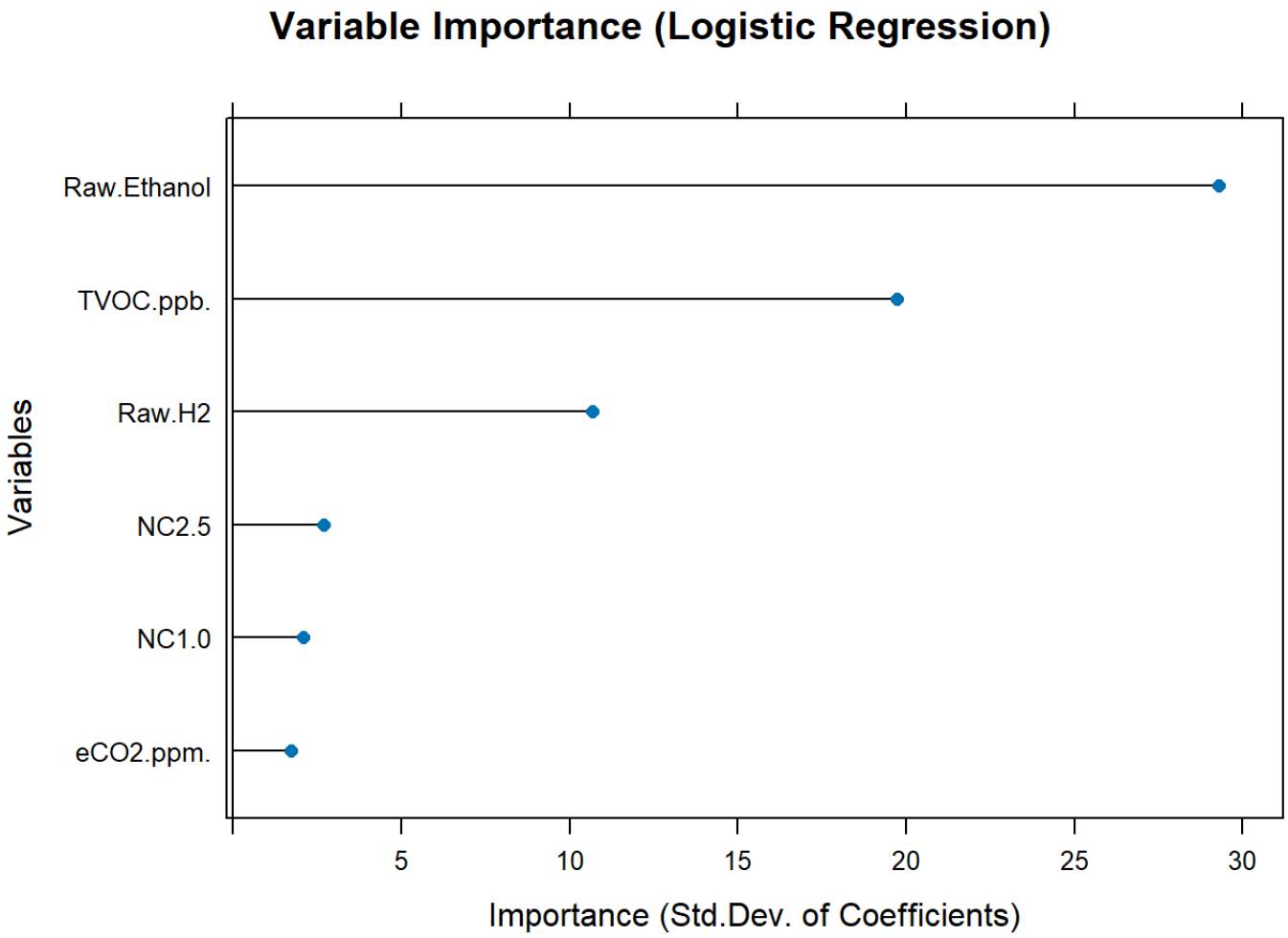
```
# Logistic Regression ROC curve and AUC score
lrRoc <- roc(response = lrFit$pred$obs,
             predictor = lrFit$pred$Yes,
             levels = rev(levels(lrFit$pred$obs)))
```

```
## Setting direction: controls > cases
```

```
auc_score_lr <- auc(lrRoc )

plot(lrRoc, main = "Logistic Regression Train Model ROC Curve",
     col = "darkgreen", lwd = 2)
abline(a = 0, b = 1, lty = 3, col = "darkgray")
text(0.5, 0.3, paste0("AUC = ", round(auc_score_lr, 2)), adj = 2.6, col = "darkgree
n")
legend("bottomright", legend = c("LR ROC Curve", "Random"),
       col = c("darkgreen", "darkgray"), lty = c(1, 2), lwd = c(2, 1))
```




```
# Variable importance
lrImp <- varImp(lrFit, scale = FALSE)
plot(lrImp, main = "Variable Importance (Logistic Regression)",
     xlab = "Importance (Std.Dev. of Coefficients)", ylab = "Variables")
```

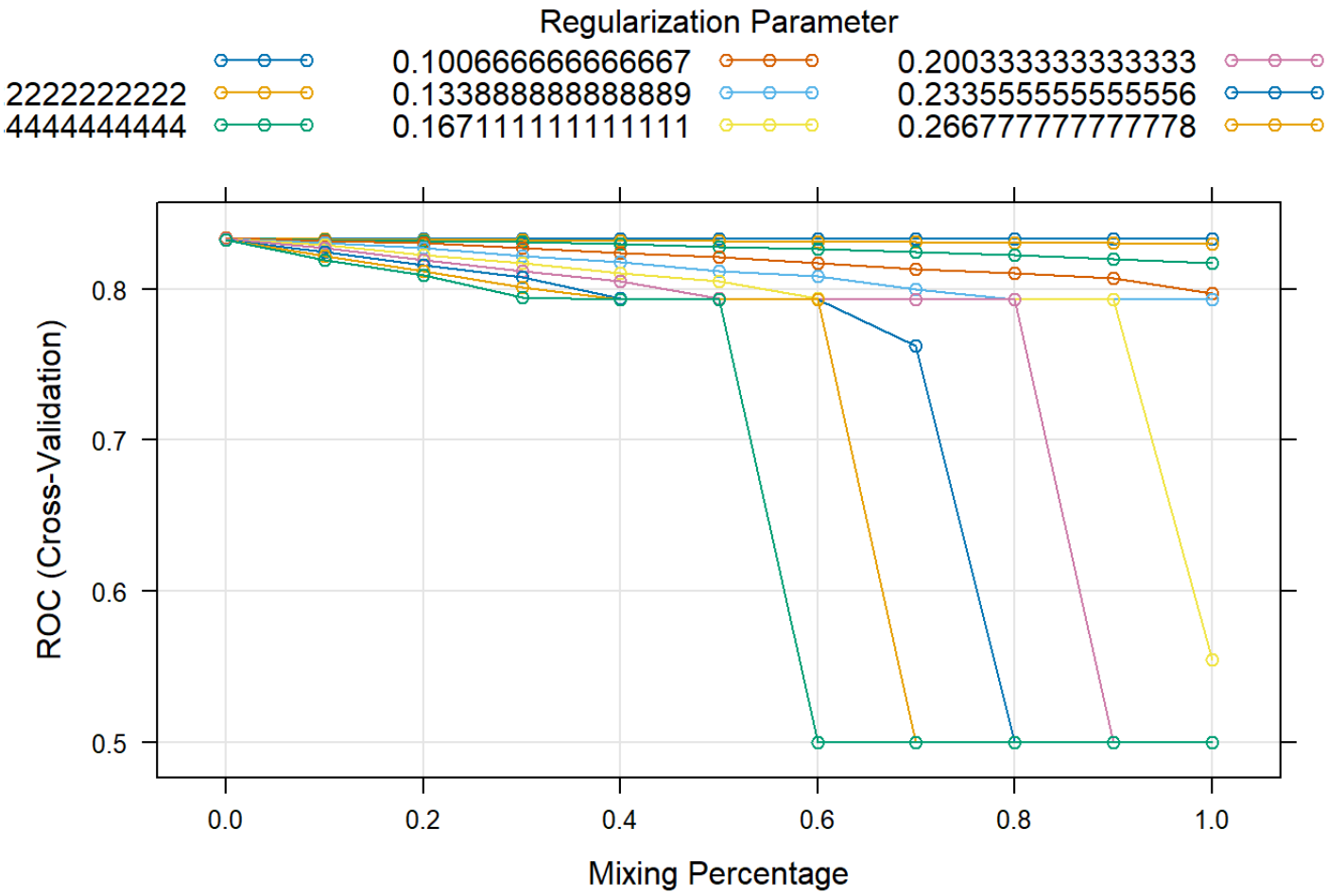


Penalized Logistic Regression

```
# Create Penalized Logistic Regression
glmnGrid <- expand.grid(alpha = seq(0, 1, by = 0.1),
lambda = seq(.001, .3, length = 10))
glmnFit <- train(Fire.Alarm ~ .,
                 data = train_balanced,
                 method = "glmnet",
                 tuneGrid = glmnGrid,
                 preProc = c("center", "scale"),
                 metric = "ROC",
                 trControl = ctrl)
optimal_a <- glmnFit$bestTune$alpha
optimal_l <- glmnFit$bestTune$lambda
glmnmodel <- train(Fire.Alarm ~ .,
                   data = train_balanced,
                   method = "glmnet",
                   preProc = c("center", "scale"),
                   metric = "ROC",
                   trControl = ctrl,
                   tuneGrid = expand.grid(alpha = optimal_a,
                                          lambda = optimal_l))

glmnmodel
```

```
## glmnet
##
## 4001 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## Pre-processing: centered (6), scaled (6)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3601, 3601, 3601, 3601, 3601, 3600, ...
## Resampling results:
##
##   ROC        Sens       Spec
##   0.8330415  0.7154229  0.8523291
##
## Tuning parameter 'alpha' was held constant at a value of 0
## Tuning parameter 'lambda'
##  was held constant at a value of 0.06744444
```

```
plot(glmnFit)
```



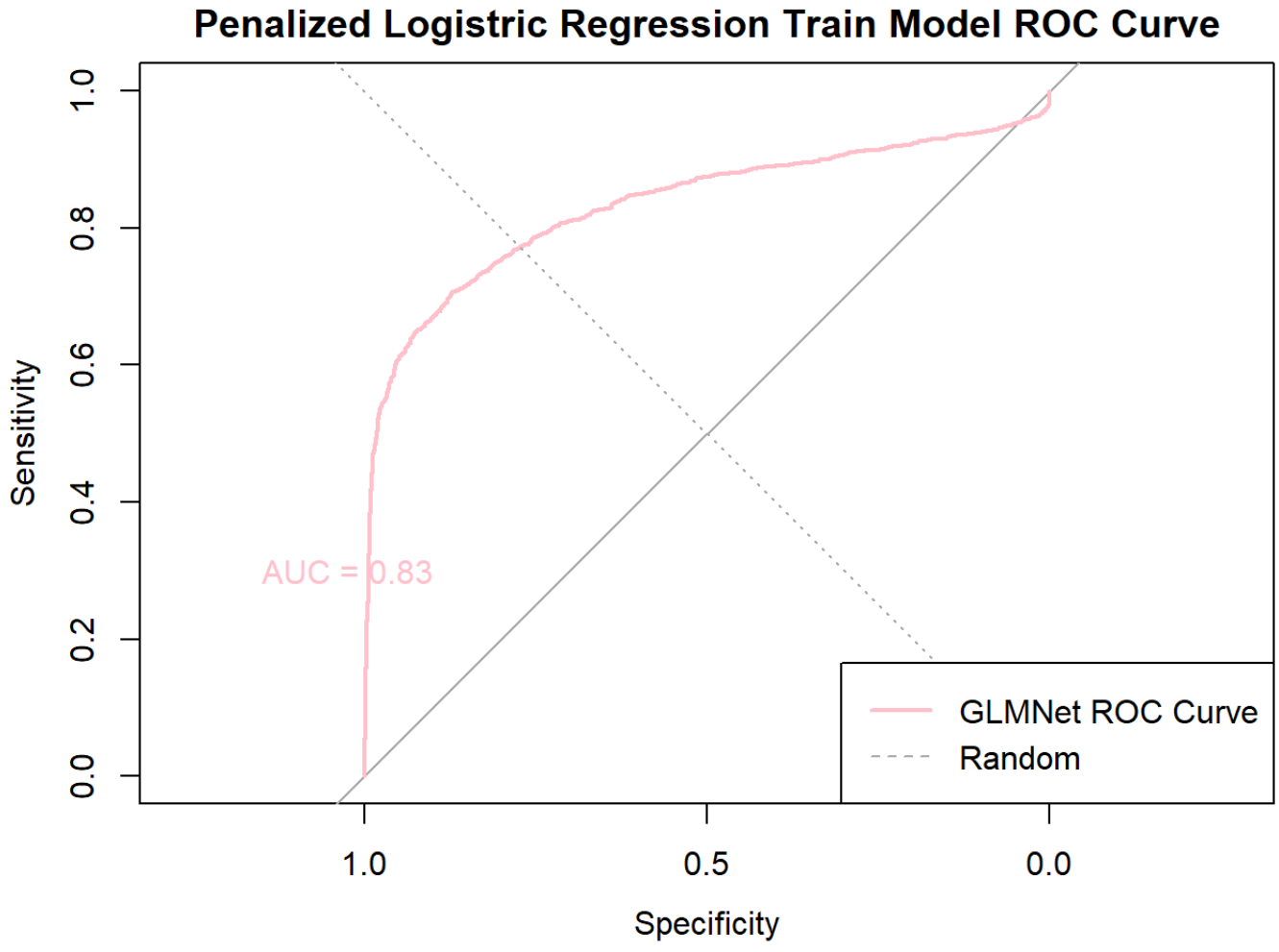

```
# Confusion matrix
glmnCM <- confusionMatrix(glmnmodel, norm = "none")
glmnCM
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##           Reference
## Prediction   No  Yes
##        No  1438  294
##        Yes  572 1697
##
##  Accuracy (average) : 0.7836
```

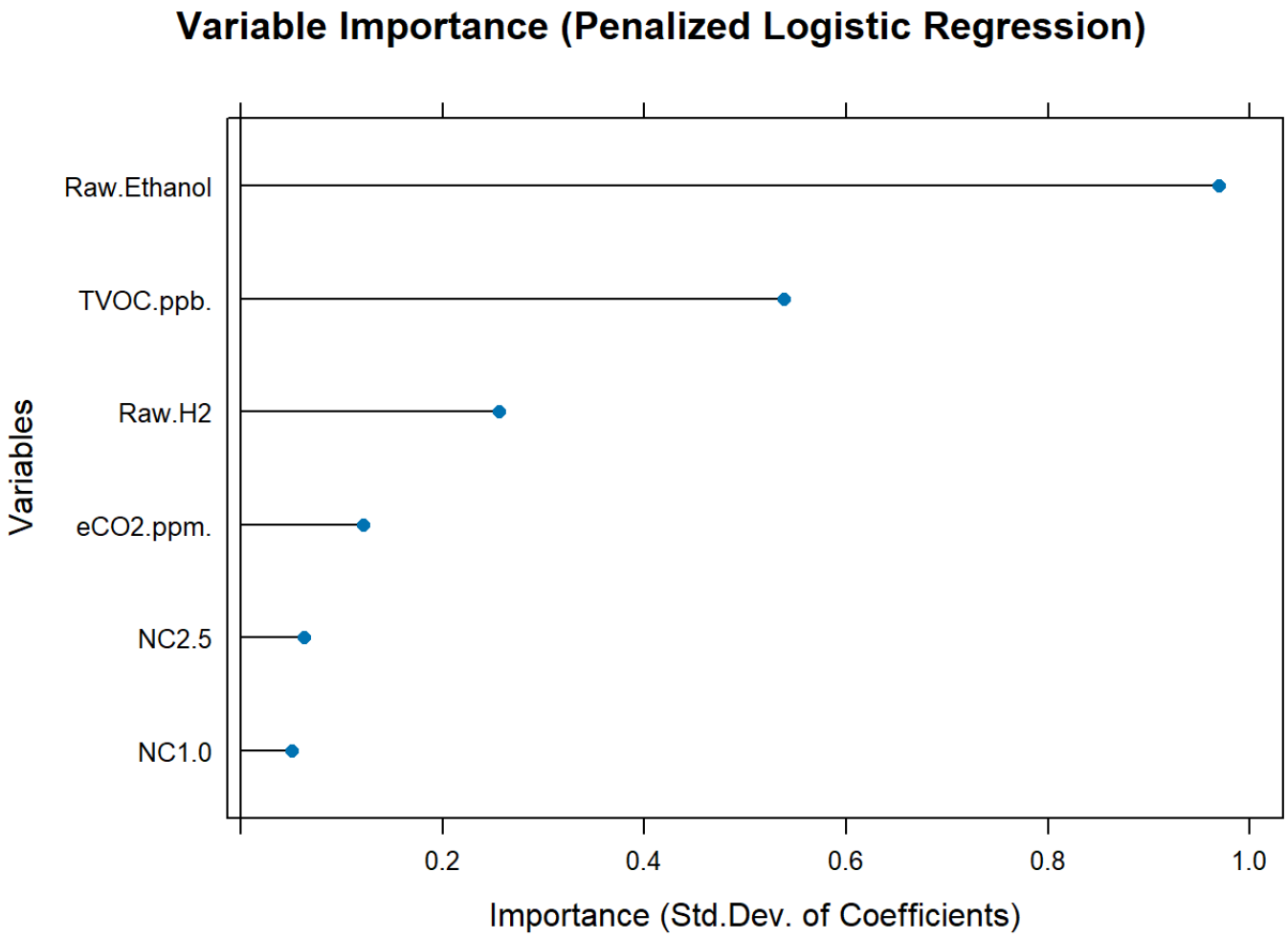
```
# Penalized Logistic Regression ROC curve and AUC score
glmnRoc <- roc(response = glmnmodel$pred$obs,
               predictor = glmnmodel$pred$Yes,
               levels = rev(levels(glmnmodel$pred$obs)))
```

```
## Setting direction: controls > cases
```

```
auc_score_glmn <- auc(glmnRoc )

plot(glmnRoc, main = "Penalized Logistic Regression Train Model ROC Curve",
     col = "pink", lwd = 2)
abline(a = 0, b = 1, lty = 3, col = "darkgray")
text(0.5, 0.3, paste0("AUC = ", round(auc_score_glmn, 2)), adj = 2.6, col = "pink")
legend("bottomright", legend = c("GLMNet ROC Curve", "Random"),
       col = c("pink", "darkgray"), lty = c(1, 2), lwd = c(2, 1))
```



```
# Variable importance
glmnImp <- varImp(glmnmodel, scale = FALSE)
plot(glmnImp, main = "Variable Importance (Penalized Logistic Regression)",
     xlab = "Importance (Std.Dev. of Coefficients)", ylab = "Variables")
```



Nearest Shrunken Centroids

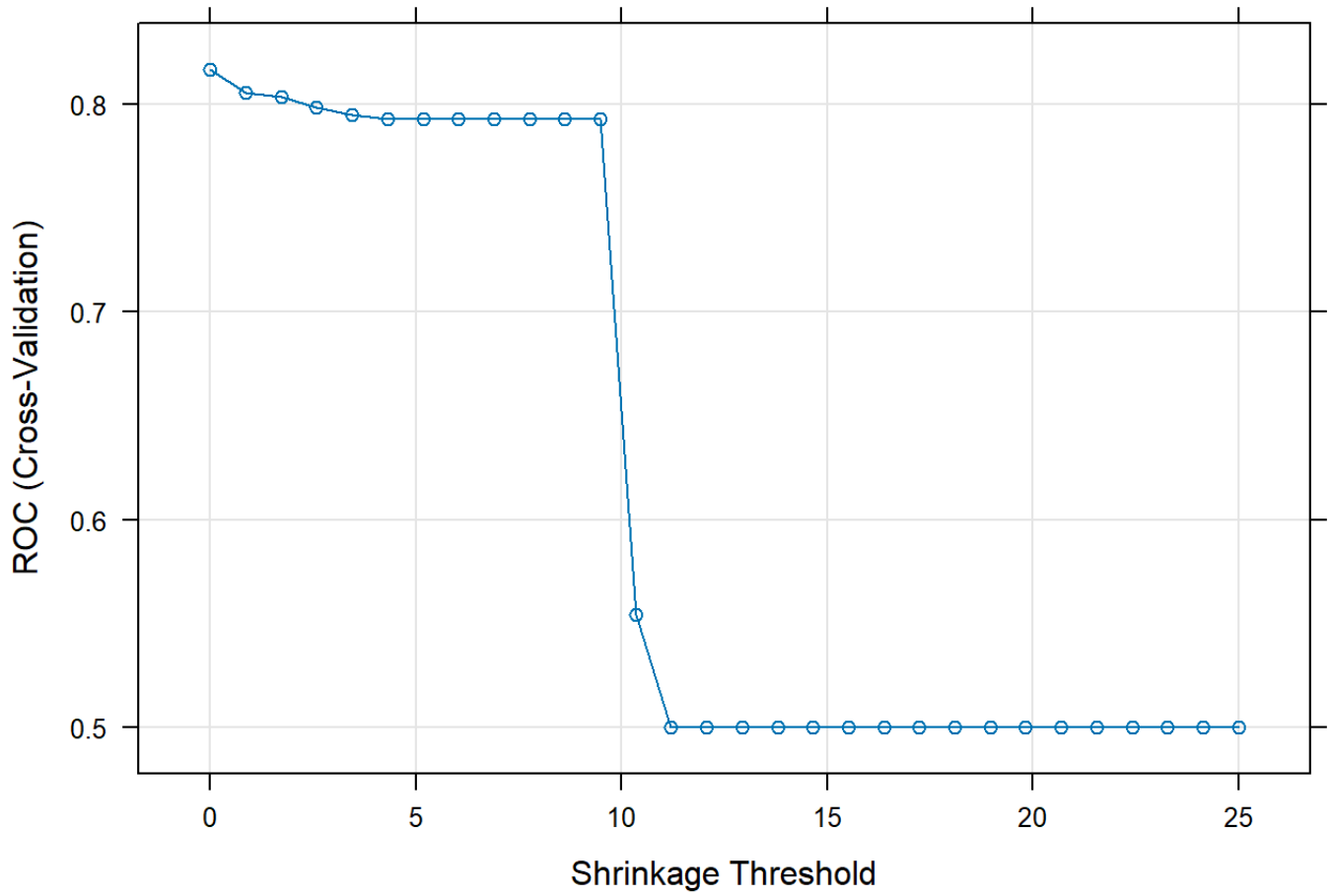
```
# Create Nearest Shrunken Centroids
nscGrid <- expand.grid(threshold = seq(0, 25, length = 30))
nscFit <- train(Fire.Alarm ~ .,
                data = train_balanced,
                method = "pam",
                tuneGrid = nscGrid,
                metric = "ROC",
                trControl = ctrl)
```

```
## 11111111111
```

```
nscFit
```

```
## Nearest Shrunken Centroids
##
## 4001 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3601, 3601, 3600, 3601, 3601, 3601, ...
## Resampling results across tuning parameters:
##
##   threshold  ROC        Sens       Spec
##    0.000000  0.8169255  0.7114428  0.82169598
##    0.862069  0.8055693  0.7109453  0.79407035
##    1.724138  0.8033393  0.7253731  0.77046482
##    2.586207  0.7983385  0.7303483  0.74585176
##    3.448276  0.7948267  0.7422886  0.73329648
##    4.310345  0.7931467  0.7492537  0.72425628
##    5.172414  0.7931467  0.7547264  0.71923618
##    6.034483  0.7931467  0.7616915  0.71270854
##    6.896552  0.7931467  0.7761194  0.70165578
##    7.758621  0.7931467  0.7950249  0.67102764
##    8.620690  0.7931467  0.8388060  0.54350251
##    9.482759  0.7931467  0.9218905  0.08994975
##   10.344828  0.5541339  1.0000000  0.00000000
##   11.206897  0.5000000  1.0000000  0.00000000
##   12.068966  0.5000000  1.0000000  0.00000000
##   12.931034  0.5000000  1.0000000  0.00000000
##   13.793103  0.5000000  1.0000000  0.00000000
##   14.655172  0.5000000  1.0000000  0.00000000
##   15.517241  0.5000000  1.0000000  0.00000000
##   16.379310  0.5000000  1.0000000  0.00000000
##   17.241379  0.5000000  1.0000000  0.00000000
##   18.103448  0.5000000  1.0000000  0.00000000
##   18.965517  0.5000000  1.0000000  0.00000000
##   19.827586  0.5000000  1.0000000  0.00000000
##   20.689655  0.5000000  1.0000000  0.00000000
##   21.551724  0.5000000  1.0000000  0.00000000
##   22.413793  0.5000000  1.0000000  0.00000000
##   23.275862  0.5000000  1.0000000  0.00000000
##   24.137931  0.5000000  1.0000000  0.00000000
##   25.000000  0.5000000  1.0000000  0.00000000
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was threshold = 0.
```

```
plot(nscFit)
```



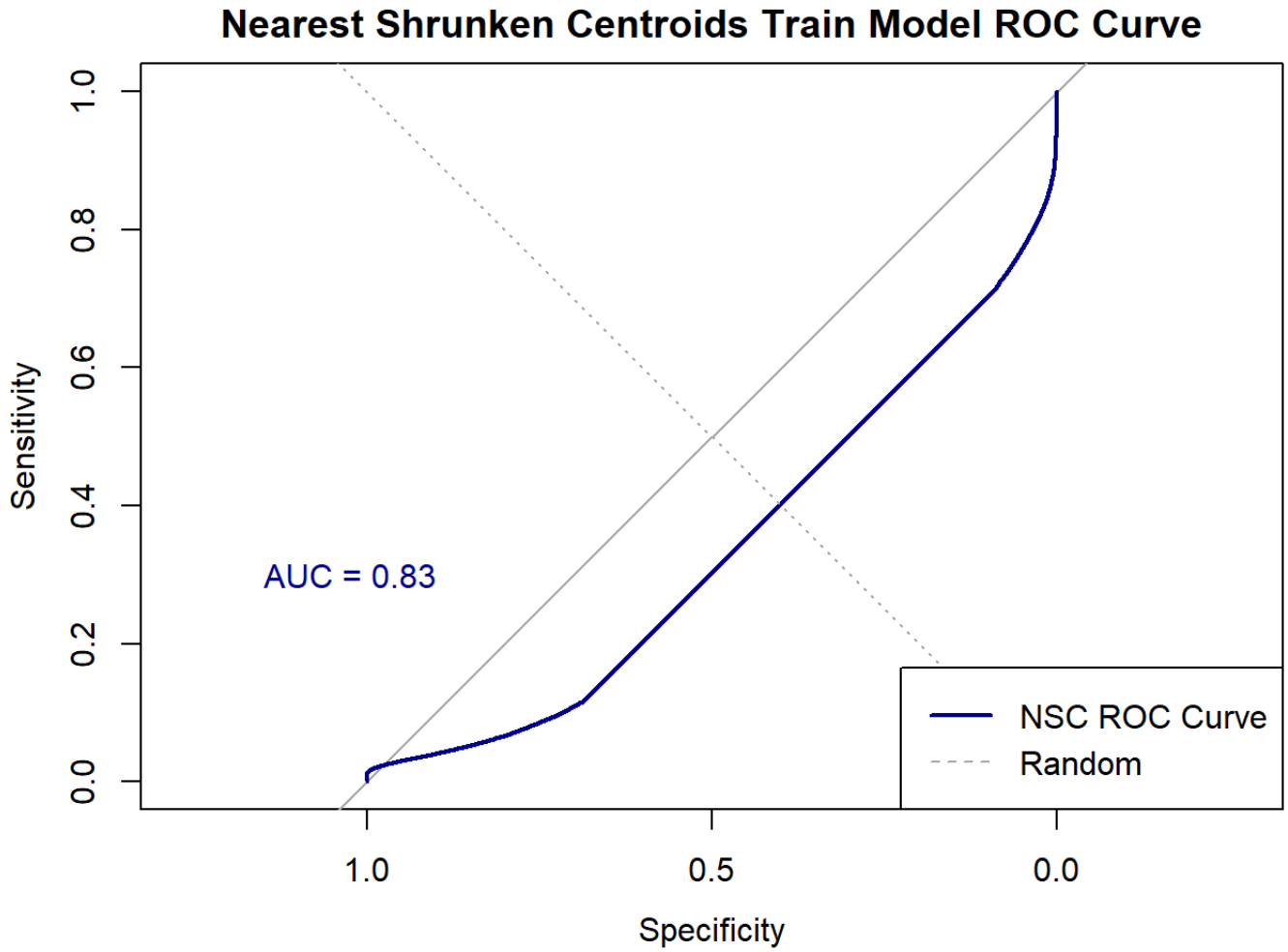

```
# Confusion matrix
nscCM <- confusionMatrix(nscFit, norm = "none")
nscCM
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##            Reference
## Prediction   No  Yes
##        No  1430  355
##        Yes  580 1636
##
##  Accuracy (average) : 0.7663
```

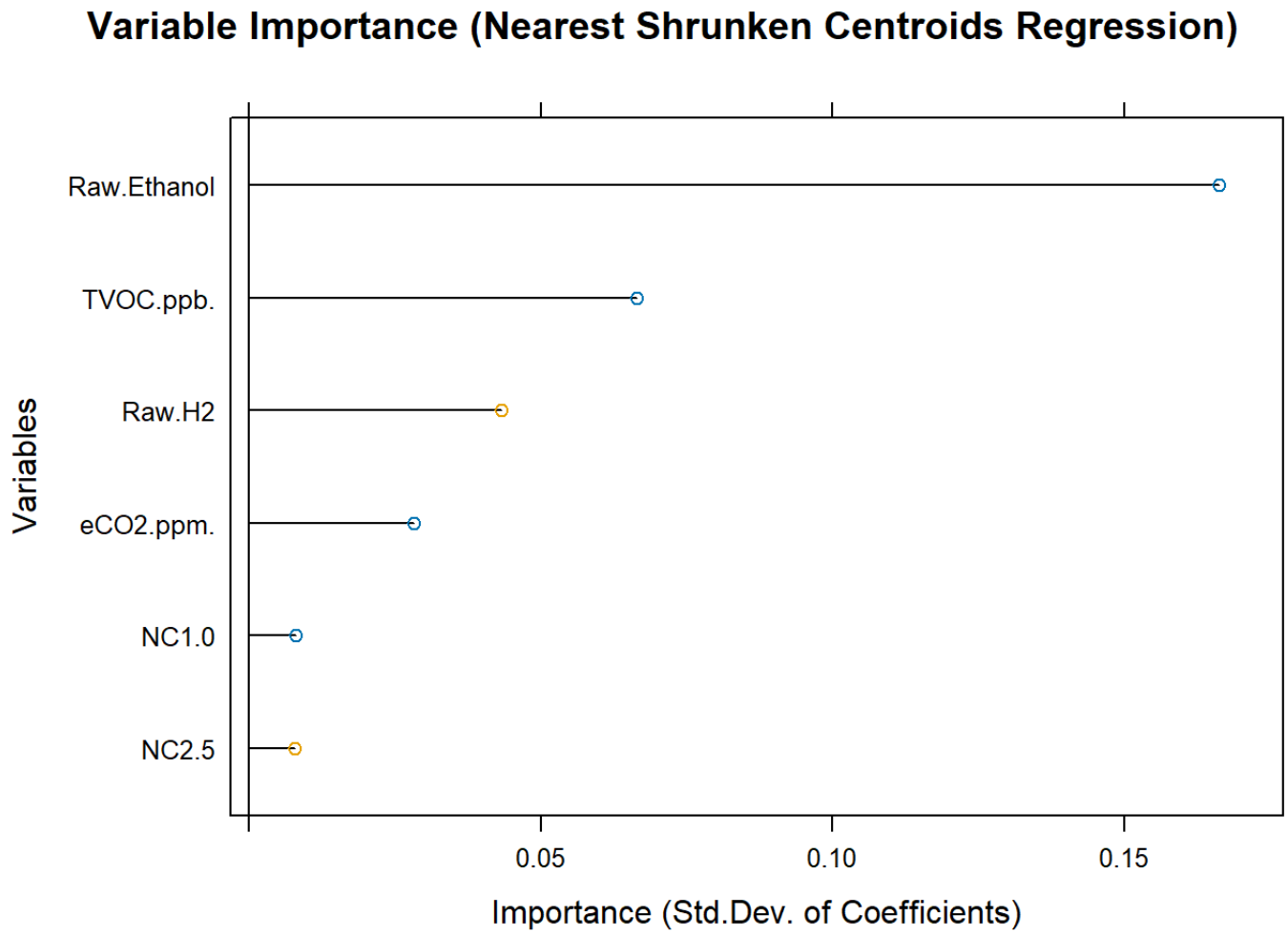
```
# Nearest Shrunken Centroids ROC curve and AUC score
nscRoc <- roc(response = nscFit$pred$obs,
              predictor = nscFit$pred$Yes,
              levels = rev(levels(nscFit$pred$obs)))
```

```
## Setting direction: controls < cases
```

```
auc_score_nsc <- auc(nscRoc)

plot(nscRoc, main = "Nearest Shrunken Centroids Train Model ROC Curve",
     col = "darkblue", lwd = 2)
abline(a = 0, b = 1, lty = 3, col = "darkgray")
text(0.5, 0.3, paste0("AUC = ", round(auc_score_glmn, 2)), adj = 2.6, col = "darkblu
e")
legend("bottomright", legend = c("NSC ROC Curve", "Random"),
       col = c("darkblue", "darkgray"), lty = c(1, 2), lwd = c(2, 1))
```



```
# Variable importance
nscImp <- varImp(nscFit, scale = FALSE)
plot(nscImp, main = "Variable Importance (Nearest Shrunken Centroids Regression)",
     xlab = "Importance (Std.Dev. of Coefficients)", ylab = "Variables")
```

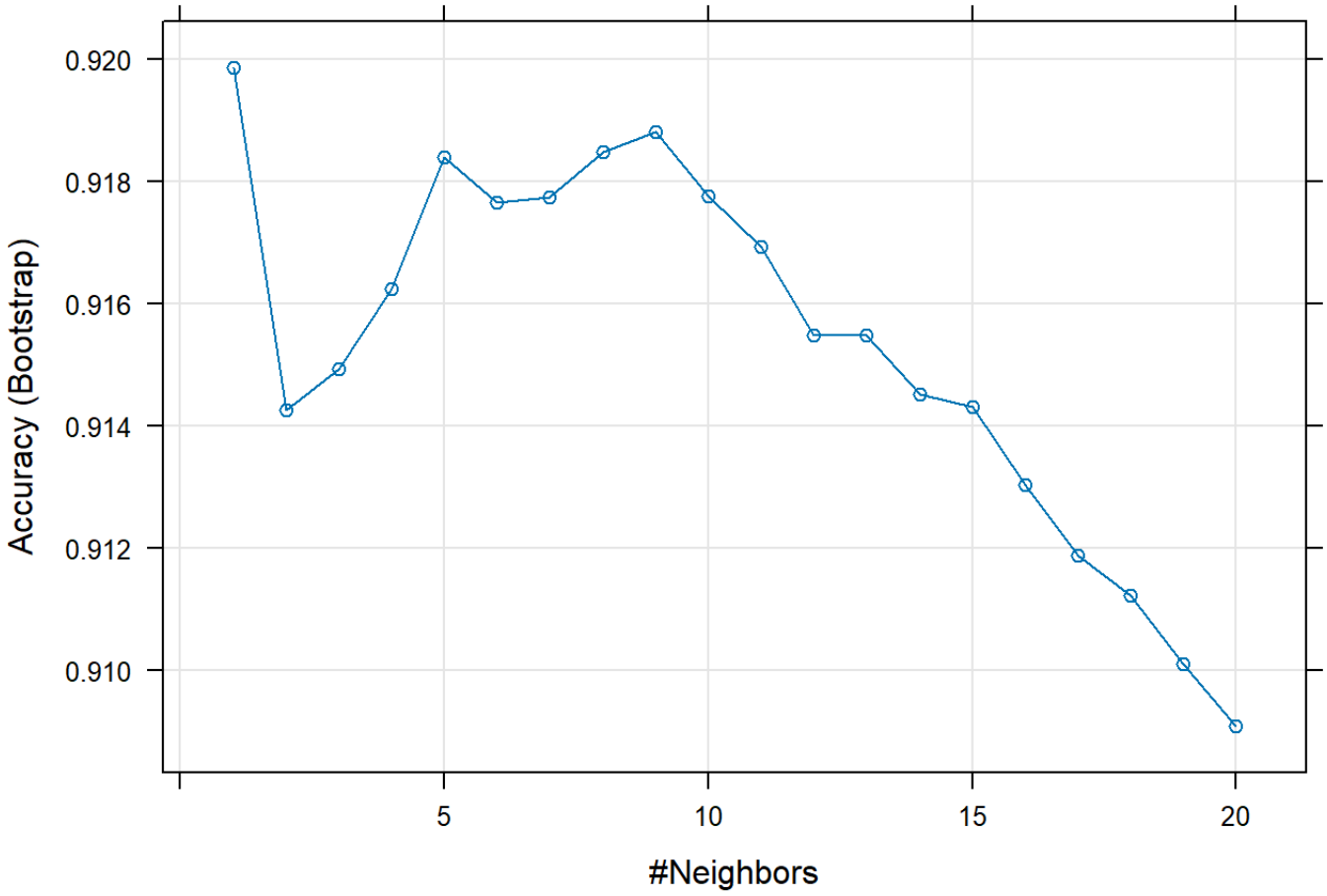


KNN (K Nearest Neighbors)

```
set.seed(100)
knnTune <- train(Fire.Alarm ~ ., data = train_balanced,
                 method = "knn",
                 preProc = c("center","scale"),
                 tuneGrid = data.frame(k = 1:20))

knnTune
```

```
## k-Nearest Neighbors
##
## 4001 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## Pre-processing: centered (6), scaled (6)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 4001, 4001, 4001, 4001, 4001, 4001, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9198642  0.8397044
##    2  0.9142667  0.8285112
##    3  0.9149387  0.8298600
##    4  0.9162315  0.8324609
##    5  0.9183972  0.8367881
##    6  0.9176535  0.8352959
##    7  0.9177381  0.8354796
##    8  0.9184842  0.8369690
##    9  0.9188040  0.8375999
##   10  0.9177642  0.8355247
##   11  0.9169206  0.8338586
##   12  0.9154862  0.8309959
##   13  0.9154893  0.8310095
##   14  0.9145090  0.8290464
##   15  0.9143135  0.8286654
##   16  0.9130402  0.8261226
##   17  0.9118809  0.8237931
##   18  0.9112209  0.8224717
##   19  0.9101137  0.8202736
##   20  0.9090833  0.8182137
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```
plot(knnTune)
```



MARS (Multivariate Adaptable Regression Spline)

```
set.seed(100)
marsTune <- train(Fire.Alarm ~ ., data = train_balanced,
                  method = "earth",
                  tuneGrid = expand.grid(degree = 1, nprune = 2:38))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

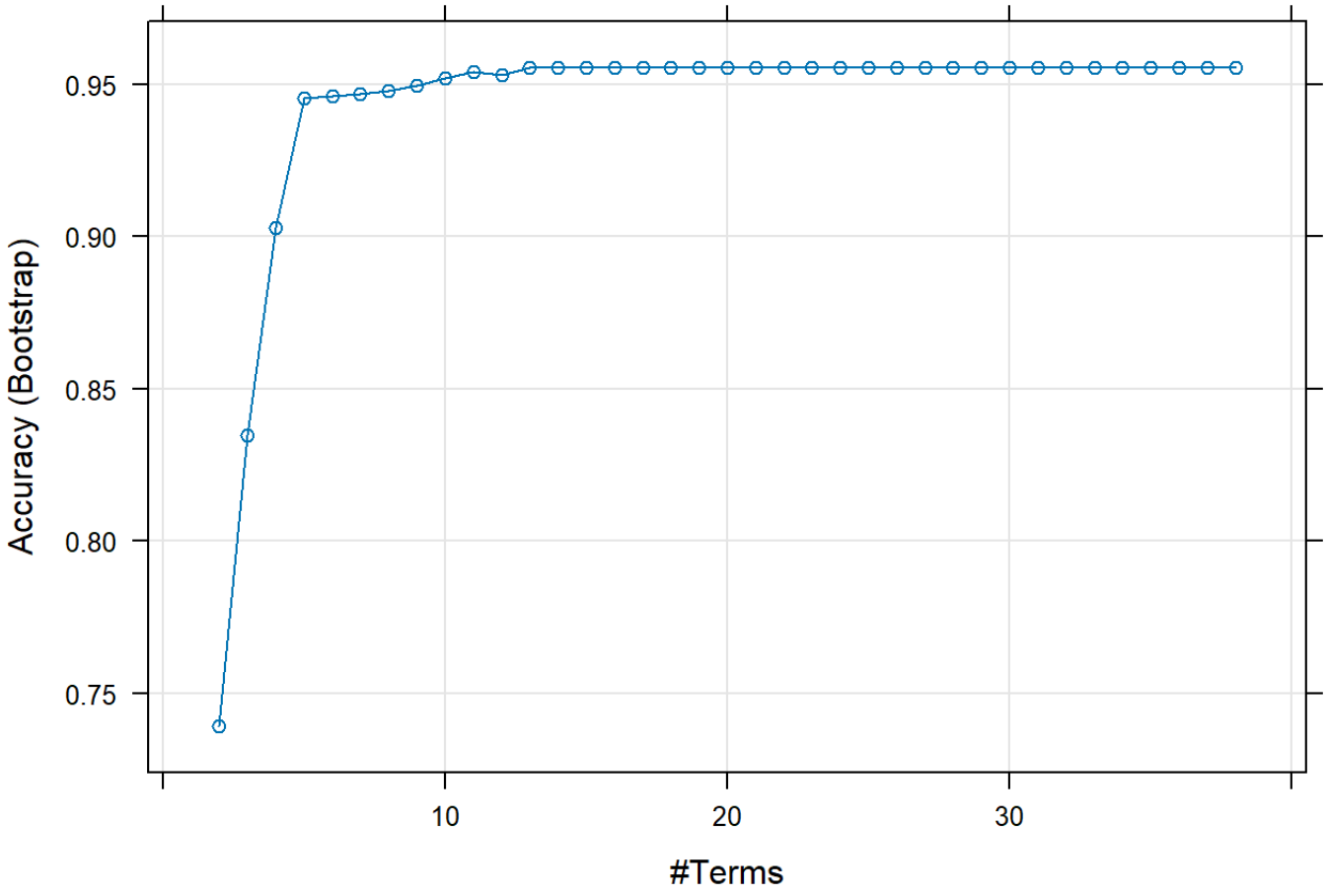
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
marsTune
```

```
## Multivariate Adaptive Regression Spline
##
## 4001 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 4001, 4001, 4001, 4001, 4001, 4001, ...
## Resampling results across tuning parameters:
##
##   nprune  Accuracy   Kappa
##    2      0.7390587  0.4781796
##    3      0.8346021  0.6692391
##    4      0.9028691  0.8057944
##    5      0.9455815  0.8911364
##    6      0.9460950  0.8921540
##    7      0.9467634  0.8934922
##    8      0.9478473  0.8956588
##    9      0.9495619  0.8990874
##   10      0.9519621  0.9038932
##   11      0.9543307  0.9086305
##   12      0.9532345  0.9064347
##   13      0.9556691  0.9113057
##   14      0.9556417  0.9112505
##   15      0.9556417  0.9112505
##   16      0.9556417  0.9112505
##   17      0.9556417  0.9112505
##   18      0.9556417  0.9112505
##   19      0.9556417  0.9112505
##   20      0.9556417  0.9112505
##   21      0.9556417  0.9112505
##   22      0.9556417  0.9112505
```

```
##   23      0.9556417  0.9112505
##   24      0.9556417  0.9112505
##   25      0.9556417  0.9112505
##   26      0.9556417  0.9112505
##   27      0.9556417  0.9112505
##   28      0.9556417  0.9112505
##   29      0.9556417  0.9112505
##   30      0.9556417  0.9112505
##   31      0.9556417  0.9112505
##   32      0.9556417  0.9112505
##   33      0.9556417  0.9112505
##   34      0.9556417  0.9112505
##   35      0.9556417  0.9112505
##   36      0.9556417  0.9112505
##   37      0.9556417  0.9112505
##   38      0.9556417  0.9112505
##
## Tuning parameter 'degree' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nprune = 13 and degree = 1.
```

```
plot(marsTune)
```



NN (Neutral Network)

```
# X_test <- df <- subset(test_df, select = -c(UTC, CNT, Fire.Alarm))

df <- read.csv("train_dataset.csv")  # Make sure to set the correct path
y_failure_type <- factor(df$Fire.Alarm)
# Remove unnecessary columns
X <-  subset(df, select = -c(UTC, CNT, Fire.Alarm))

if(any(!sapply(X, is.numeric))) {
  X[] <- lapply(X, function(x) as.numeric(as.factor(x)))
}

# Standardize numeric features
preProcValue <- preProcess(X,method = c("center", "scale"))
X_processed <- predict(preProcValue, X)

# Splitting the data into training and testing sets
set.seed(42)
trainIndex <- createDataPartition(y_failure_type, p = 0.7, list = FALSE)
```

```
X_train <- X_processed[trainIndex, ]
X_test  <- X_processed[-trainIndex, ]
y_train <- y_failure_type[trainIndex]
y_test  <- y_failure_type[-trainIndex]

# Convert data to torch tensors
X_train_tensor <- torch_tensor(as.matrix(X_train), dtype = torch_float())
X_test_tensor <- torch_tensor(as.matrix(X_test), dtype = torch_float())
y_train_tensor <- torch_tensor(as.integer(y_train), dtype = torch_long())
y_test_tensor <- torch_tensor(as.integer(y_test), dtype = torch_long())

class_counts <- table(y_train)
total_counts <- sum(class_counts)
manual_class_weights <- c(1.0, 3.0)  # Adjust these values as needed
class_weights <- torch_tensor(manual_class_weights, dtype = torch_float())

# Define the neural network class
Net <- nn_module(
  initialize = function(input_size, num_classes = 2) {
    self$fc1 <- nn_linear(input_size, 8)
    self$relu <- nn_relu()
    self$fc2 <- nn_linear(8, num_classes)
  },
  forward = function(x) {
    x <- self$fc1(x)
    x <- self$relu(x)
    x <- self$fc2(x)
    x
  }
)
```

```
input_size <- ncol(X_train)

# Create an instance of the Net class
nn_model <- Net(input_size)

# Define a loss function and an optimizer
criterion <- nn_cross_entropy_loss(weight = class_weights)
optimizer <- optim_sgd(nn_model$parameters, lr = 0.01)

# Training loop
num_epochs <- 1000000
for (epoch in 1:num_epochs) {
  nn_model$train()

  # Forward pass
  outputs <- nn_model(X_train_tensor)
  loss <- criterion(outputs, y_train_tensor)

  # Backward and optimize
  optimizer$zero_grad()
  loss$backward()
  optimizer$step()

  if (epoch %% 10000 == 0) {
    cat(sprintf("Epoch [%d/%d], Loss: %.4f\n", epoch, num_epochs, loss$item()))
  }
}
```

```
## Epoch [10000/1000000], Loss: 0.1533
## Epoch [20000/1000000], Loss: 0.1242
## Epoch [30000/1000000], Loss: 0.1134
## Epoch [40000/1000000], Loss: 0.1079
## Epoch [50000/1000000], Loss: 0.1046
## Epoch [60000/1000000], Loss: 0.1017
## Epoch [70000/1000000], Loss: 0.0998
## Epoch [80000/1000000], Loss: 0.0982
## Epoch [90000/1000000], Loss: 0.0970
## Epoch [100000/1000000], Loss: 0.0959
## Epoch [110000/1000000], Loss: 0.0948
## Epoch [120000/1000000], Loss: 0.0937
## Epoch [130000/1000000], Loss: 0.0926
## Epoch [140000/1000000], Loss: 0.0914
## Epoch [150000/1000000], Loss: 0.0903
## Epoch [160000/1000000], Loss: 0.0892
## Epoch [170000/1000000], Loss: 0.0880
## Epoch [180000/1000000], Loss: 0.0869
```

```
## Epoch [660000/1000000], Loss: 0.0315
## Epoch [670000/1000000], Loss: 0.0308
## Epoch [680000/1000000], Loss: 0.0302
## Epoch [690000/1000000], Loss: 0.0296
## Epoch [700000/1000000], Loss: 0.0290
## Epoch [710000/1000000], Loss: 0.0284
## Epoch [720000/1000000], Loss: 0.0278
## Epoch [730000/1000000], Loss: 0.0273
## Epoch [740000/1000000], Loss: 0.0268
## Epoch [750000/1000000], Loss: 0.0262
## Epoch [760000/1000000], Loss: 0.0257
## Epoch [770000/1000000], Loss: 0.0252
## Epoch [780000/1000000], Loss: 0.0248
## Epoch [790000/1000000], Loss: 0.0243
## Epoch [800000/1000000], Loss: 0.0238
## Epoch [810000/1000000], Loss: 0.0234
## Epoch [820000/1000000], Loss: 0.0230
## Epoch [830000/1000000], Loss: 0.0225
## Epoch [840000/1000000], Loss: 0.0221
## Epoch [850000/1000000], Loss: 0.0217
## Epoch [860000/1000000], Loss: 0.0213
## Epoch [870000/1000000], Loss: 0.0209
## Epoch [880000/1000000], Loss: 0.0206
## Epoch [890000/1000000], Loss: 0.0202
## Epoch [900000/1000000], Loss: 0.0199
## Epoch [910000/1000000], Loss: 0.0195
## Epoch [920000/1000000], Loss: 0.0192
## Epoch [930000/1000000], Loss: 0.0189
## Epoch [940000/1000000], Loss: 0.0185
## Epoch [950000/1000000], Loss: 0.0182
## Epoch [960000/1000000], Loss: 0.0179
## Epoch [970000/1000000], Loss: 0.0176
## Epoch [980000/1000000], Loss: 0.0173
## Epoch [990000/1000000], Loss: 0.0170
## Epoch [1000000/1000000], Loss: 0.0168
```

```
# Evaluation on the test set
# Evaluation on the test set
```

SVM (Support Vector Machine)

```
svm_tune_grid=expand.grid(C = 2^(-5:2), sigma = 2^(-5:2))
svm_model <- train(X_train, y_train, method = 'svmRadial',
                   trControl = trainControl(method = 'cv', number = 10),
                   tuneGrid = svm_tune_grid,
                   tuneLength = 10)

svm_predictions <- predict(svm_model, X_test)
```

Validation and Testing

Logistic Regression

```
# Predict test data using logistic regression
test_predictors <- smokedf_test_transformed[, -8]
test_org <- data.frame(Fire.Alarm = smokedf_test_transformed$Fire.Alarm)
test_org$Fire.Alarm <- factor(test_org$Fire.Alarm)

lr_predictions <- predict(lrFit, newdata = test_predictors, type = "raw")
lr_predictions <- data.frame(Fire.Alarm = lr_predictions)

# Confusion Matrix
lrCM_test <- confusionMatrix(data = as.factor(lr_predictions$Fire.Alarm),
                             reference = test_org$Fire.Alarm, positive = "Yes")
lrCM_test
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  502  56
##        Yes 214 227
##
##                Accuracy : 0.7297
##                  95% CI : (0.701, 0.7571)
##     No Information Rate : 0.7167
##     P-Value [Acc > NIR] : 0.1904
##
##                   Kappa : 0.4306
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8021
##             Specificity : 0.7011
##          Pos Pred Value : 0.5147
##          Neg Pred Value : 0.8996
##              Prevalence : 0.2833
##          Detection Rate : 0.2272
##    Detection Prevalence : 0.4414
##       Balanced Accuracy : 0.7516
##
##        'Positive' Class : Yes
##
```

Penalized Logistic Regression

```
# Predict test data using penalized logistic regression
glmn_predictions <- predict(glmnmodel, newdata = test_predictors, type = "raw")
glmn_predictions <- data.frame(Fire.Alarm = glmn_predictions)

# Confusion Matrix
glmnCM_test <- confusionMatrix(data = as.factor(glmn_predictions$Fire.Alarm),
                               reference = test_org$Fire.Alarm, positive = "Yes")
glmnCM_test
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  468  39
##        Yes 248 244
##
##                Accuracy : 0.7127
##                  95% CI : (0.6836, 0.7406)
##     No Information Rate : 0.7167
##     P-Value [Acc > NIR] : 0.6257
##
##                   Kappa : 0.4217
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8622
##             Specificity : 0.6536
##          Pos Pred Value : 0.4959
##          Neg Pred Value : 0.9231
##              Prevalence : 0.2833
##          Detection Rate : 0.2442
##    Detection Prevalence : 0.4925
##       Balanced Accuracy : 0.7579
##
##        'Positive' Class : Yes
##
```

Nearest Shrunken Centroid

```
# Predict test data using nearest shrunken centroid
nsc_predictions <- predict(nscFit, newdata = test_predictors, type = "raw")
nsc_predictions <- data.frame(Fire.Alarm = nsc_predictions)

# Confusion Matrix
nscCM_test <- confusionMatrix(data = as.factor(nsc_predictions$Fire.Alarm),
                              reference = test_org$Fire.Alarm, positive = "Yes")
nscCM_test
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  485  60
##        Yes 231 223
##
##                Accuracy : 0.7087
##                  95% CI : (0.6794, 0.7367)
##     No Information Rate : 0.7167
##     P-Value [Acc > NIR] : 0.7258
##
##                   Kappa : 0.3935
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7880
##             Specificity : 0.6774
##          Pos Pred Value : 0.4912
##          Neg Pred Value : 0.8899
##              Prevalence : 0.2833
##          Detection Rate : 0.2232
##    Detection Prevalence : 0.4545
##       Balanced Accuracy : 0.7327
##
##        'Positive' Class : Yes
##
```

KNN (K Nearest Neighbors)

```
#Predictors for KNN

test_predictors <- smokedf_test_transformed[, -8]
test_org <- data.frame(Fire.Alarm = smokedf_test_transformed$Fire.Alarm)
test_org$Fire.Alarm <- factor(test_org$Fire.Alarm)

knn_predictions <- predict(knnTune, newdata = test_predictors, type = "raw")
knn_predictions <- data.frame(Fire.Alarm = knn_predictions)
```

MARS (Multivariate Adaptable Regression Spline)

```
#Predictors for Mars

mars_predictions <- predict(marsTune, newdata = test_predictors, type = "raw")
mars_predictions <- data.frame(Fire.Alarm = mars_predictions)
```

Performance Evaluation

LR, PLR, and NSC Performance

```
roc_test_lr <- roc(response = as.factor(test_org$Fire.Alarm),
                   predictor = as.numeric(lr_predictions$Fire.Alarm))
```

```
## Setting levels: control = No, case = Yes
```

```
## Setting direction: controls < cases
```

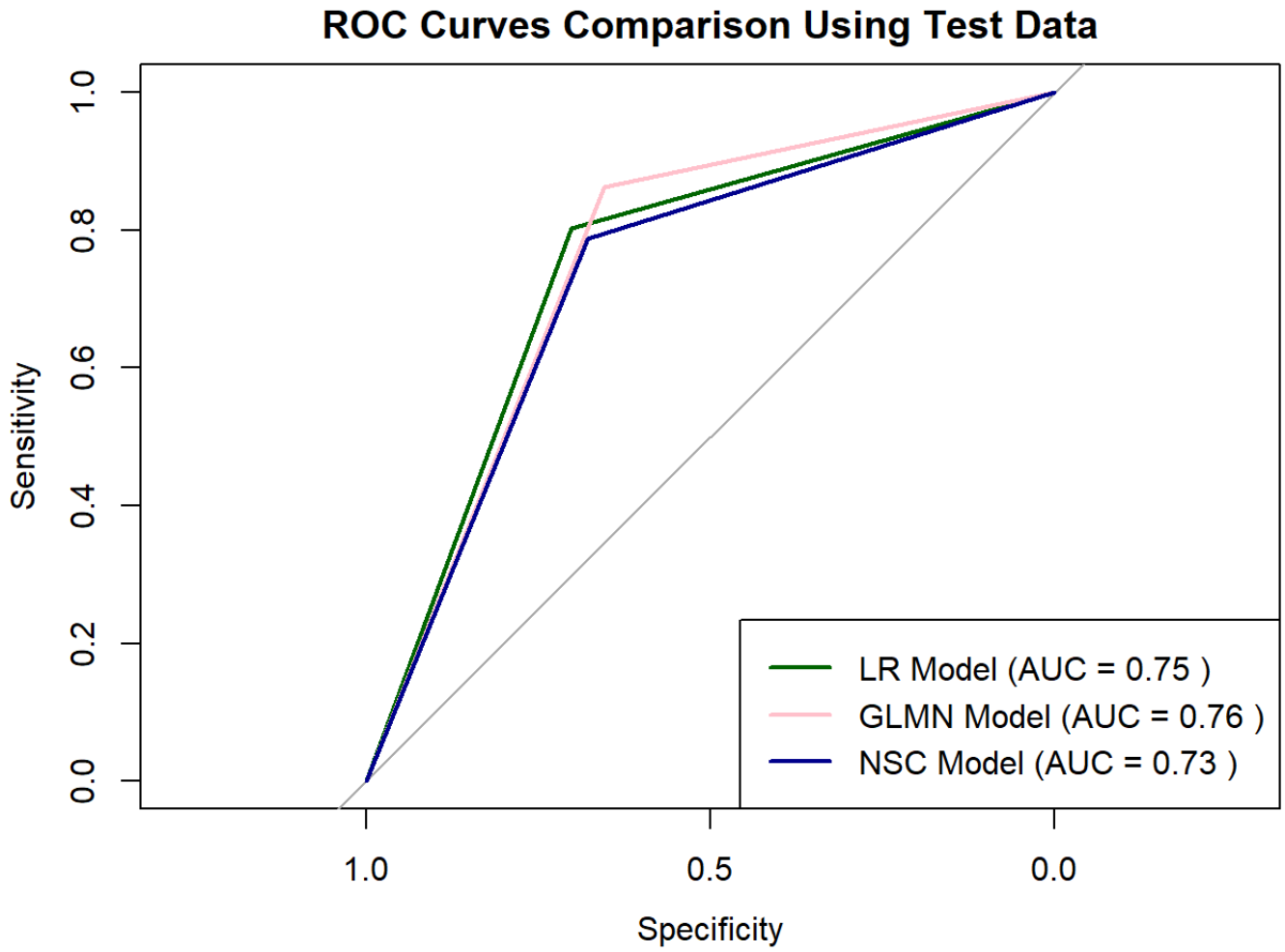
```
roc_test_glmn <- roc(response = as.factor(test_org$Fire.Alarm),
                   predictor = as.numeric(glmn_predictions$Fire.Alarm))
```

```
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
```

```
roc_test_nsc <- roc(response = as.factor(test_org$Fire.Alarm),
                   predictor = as.numeric(nsc_predictions$Fire.Alarm))
```

```
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
```

```
auc_test_lr <- auc(roc_test_lr)
auc_test_glmn <- auc(roc_test_glmn)
auc_test_nsc <- auc(roc_test_nsc)

# Plot ROC curve
plot(roc_test_lr, col = "darkgreen", main = "ROC Curves Comparison Using Test Data",
lwd = 2)
plot(roc_test_glmn, col = "pink", add = TRUE, lwd = 2)
plot(roc_test_nsc, col = "darkblue", add = TRUE, lwd = 2)
legend("bottomright", legend = c(paste("LR Model (AUC =", round(auc_test_lr, 2),
")"),
                                 paste("GLMN Model (AUC =", round(auc_test_glmn, 2),
")"),
                                 paste("NSC Model (AUC =", round(auc_test_nsc, 2),
")")),
       col = c("darkgreen", "pink", "darkblue"), lwd = 2)
```



LR, PLR, and NSC Performance Evaluation

```
# Model's ROC score
lrFit_ROC <- round(lrFit$results$ROC,3)
glmnmodel_ROC <- round(glmnmodel$results$ROC,3)
nscFit_ROC <- round(nscFit$results$ROC,3)

# Confusion matrix train accuracy score
lrCM_train_accuracy <- 0.781
glmnCM_train_accuracy <- 0.781
nscCM_train_accuracy <- 0.764

# Confusion matrix test AUC score
lrCM_AUC <- round(auc(roc_test_lr),3)
glmnCM_AUC <- round(auc(roc_test_glmn),3)
nscCM_AUC <- round(auc(roc_test_nsc),3)

# Confusion matrix test accuracy score
lrCM_test_accuracy <- round(lrCM_test$overall["Accuracy"],3)
glmnCM_test_accuracy <- round(glmnCM_test$overall["Accuracy"],3)
```

```
nscCM_test_accuracy <- round(nscCM_test$overall["Accuracy"],3)

# Confusion matric test class balanced accuracy score
lrCM_test_bal_accuracy <- round(lrCM_test$byClass["Balanced Accuracy"],3)
glmnCM_test_bal_accuracy <- round(glmnCM_test$byClass["Balanced Accuracy"],3)
nscCM_test_bal_accuracy <- round(nscCM_test$byClass["Balanced Accuracy"],3)

# Adjust nscFit_ROC
nscFit_ROC_adjusted <- nscFit_ROC[1]

# Now, create the data frame with adjusted nscFit_ROC
results_table <- data.frame(
  Model = c("LR", "Penalized LR",
            "NSC"),
  ROC_Score = c(lrFit_ROC, glmnmodel_ROC, nscFit_ROC_adjusted),
  Train_Accuracy = c(lrCM_train_accuracy, glmnCM_train_accuracy,
                     nscCM_train_accuracy),
  Test_AUC = c(lrCM_AUC, glmnCM_AUC, nscCM_AUC),
  Test_Accuracy = c(lrCM_test_accuracy, glmnCM_test_accuracy,
                    nscCM_test_accuracy),
  Test_Balanced_Accuracy = c(lrCM_test_bal_accuracy,
                             glmnCM_test_bal_accuracy,
                             nscCM_test_bal_accuracy))

# Create a gt table
results_table |> gt() |>
  tab_header(title = "Model Comparison",
             subtitle = "Comparison of performance metrics for different models") |>
  cols_label(Model = "Model",
             ROC_Score = "Model ROC",
             Train_Accuracy = "Train Accuracy",
             Test_AUC = "Test AUC",
             Test_Accuracy = "Test Accuracy",
             Test_Balanced_Accuracy = "Test Balanced Accuracy")
```

Model Comparison

Comparison of performance metrics for different models

Model	Model ROC	Train Accuracy	Test AUC	Test Accuracy	Test Balanced Accuracy
LR	0.833	0.781	0.752	0.730	0.752
Penalized LR	0.833	0.781	0.758	0.713	0.758
NSC	0.817	0.764	0.733	0.709	0.733

KNN Performance Evaluation

```
cm_knn <- confusionMatrix(data = as.factor(knn_predictions$Fire.Alarm),
                          reference = test_org$Fire.Alarm, positive = "Yes")
cm_knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  473   2
##        Yes 243 281
##
##               Accuracy : 0.7548
##                 95% CI : (0.7268, 0.7812)
##    No Information Rate : 0.7167
##    P-Value [Acc > NIR] : 0.003841
##
##                  Kappa : 0.5197
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9929
##             Specificity : 0.6606
##          Pos Pred Value : 0.5363
##          Neg Pred Value : 0.9958
##              Prevalence : 0.2833
##          Detection Rate : 0.2813
##    Detection Prevalence : 0.5245
##       Balanced Accuracy : 0.8268
##
##        'Positive' Class : Yes
##
```

```
#Accuracy of 0.755
```

MARS Performance Evaluation

```
cm_mars <- confusionMatrix(data = as.factor(mars_predictions$Fire.Alarm),
                          reference = test_org$Fire.Alarm, positive = "Yes")

cm_mars
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  393   3
##        Yes 323 280
##
##                Accuracy : 0.6737
##                  95% CI : (0.6436, 0.7027)
##     No Information Rate : 0.7167
##     P-Value [Acc > NIR] : 0.9987
##
##                   Kappa : 0.4011
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9894
##             Specificity : 0.5489
##          Pos Pred Value : 0.4643
##          Neg Pred Value : 0.9924
##              Prevalence : 0.2833
##          Detection Rate : 0.2803
##    Detection Prevalence : 0.6036
##       Balanced Accuracy : 0.7691
##
##        'Positive' Class : Yes
##
```

```
#Accuracy of 0.674
```

NNET Performance Evaluation

```
nn_model$eval()
outputs_test <- nn_model(X_test_tensor)
predicted <- outputs_test$argmax(dim = 2)$sub(1L)  # Adjust predictions to match fact
or levels (1-based indexing)

predicted <- as.array(predicted)
predicted <- as.integer(predicted)


nnet_confusion <- confusionMatrix(factor(predicted, levels = levels(y_test)), y_test)
print(nnet_confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1083    6
##          1    1  409
##
##                Accuracy : 0.9953
##                  95% CI : (0.9904, 0.9981)
##     No Information Rate : 0.7231
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9883
##
##  Mcnemar's Test P-Value : 0.1306
##
##             Sensitivity : 0.9991
##             Specificity : 0.9855
##          Pos Pred Value : 0.9945
##          Neg Pred Value : 0.9976
##              Prevalence : 0.7231
##          Detection Rate : 0.7225
##    Detection Prevalence : 0.7265
##       Balanced Accuracy : 0.9923
##
##        'Positive' Class : 0
##
```

SVM Performance Evaluation

```
confusion_svm <- confusionMatrix(svm_predictions, y_test)
print(confusion_svm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1080   22
##          1    4  393
##
##                Accuracy : 0.9827
##                  95% CI : (0.9747, 0.9886)
##     No Information Rate : 0.7231
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9561
##
##  Mcnemar's Test P-Value : 0.0008561
##
##             Sensitivity : 0.9963
##             Specificity : 0.9470
##          Pos Pred Value : 0.9800
##          Neg Pred Value : 0.9899
##              Prevalence : 0.7231
##          Detection Rate : 0.7205
##    Detection Prevalence : 0.7352
##       Balanced Accuracy : 0.9716
##
##        'Positive' Class : 0
##
```