PSI_Capstone Pair Project

Ruei Li Jhang & Chia Hua Lin

2024-03-25

Contents

```
# Loading libraries

library(dplyr)
library(ggplot2)
library(ltm)
library(randomForest)
library(rpart)
library(e1071)
library(mlbench)
library(caTools)
library(caret)
library(pROC)
library(PRROC)
library(ROCR)
library(xgboost)
#library(kableExtra)

#devtools::install_github("kupietz/kableExtra")
```

Data Understanding

The data set was downloaded from Kaggle.com. According to the data description, the data focuses on malicious URL detection. This data helps in the development of cyber security systems that can detect any malicious attempt to gain access and send a sigal to systems to perform the relevant in return.

Data set link: https://www.kaggle.com/datasets/pilarpieiro/tabular-dataset-ready-for-malicious-url-detection/data

The data set includes URLs and 60 other calculated features. In our analysis, we will not use all the 60 features, but we will select the most important features. In the data description, a list of 6 important features has been provided and they are as follows:

- Basic URL Components
- Domain Information

- Content Analysis
- Host Reputation
- Network Features
- Behavioural Features

```
# # Loading the train data
# train_data <- read.csv('train_dataset.csv')
# train_data <- na.omit(train_data) # Remove rows with any NA/null values
# dim(train_data)

# # Loading test data
# test_data <- read.csv('test_dataset.csv')
# test_data <- na.omit(test_data) # Remove rows with any NA/null values
# dim(test_data)
```

The original train data has 6,728,848 observations and the test data contains 1,682,213.

Due to the computational resources, we will randomly select 200,000 observations for training and 100,000 observations for testing.

```
# # Set seed for reproducibility
# set.seed(2024)

# # Define the number of samples you want for each label category
# num_samples_per_label <- 1000000

# # Sample from each label category
# train_data_sample <- train_data %>%
#    group_by(label) %>%
#    sample_n(num_samples_per_label, replace = TRUE)

# # Test data
# test_data_sample <- test_data[sample(nrow(test_data), 1000000), ]

# # Write train_data2 to CSV
# write.csv(train_data_sample, file = "train_data_sample.csv", row.names = FALSE)
# write.csv(test_data_sample, file = "test_data_sample.csv", row.names = FALSE)
```

Loading the Save Data

```
# Loading the train data
train_data <- read.csv('train_data_sample.csv')
train_data <- na.omit(train_data) # Remove rows with any NA/null values
dim(train_data)
```

```
## [1] 200000     60
```

```
# Loading the test data
test_data <- read.csv('test_data_sample.csv')
test_data <- na.omit(test_data) # Remove rows with any NA/null values
dim(test_data)
```

```
## [1] 100000     60
```

Selecting Necessary Columns

We selected specific columns from the train_data data frame. The columns selected include various features related to URLs as:

- Whether the URL contains certain elements like login, client, server, admin, IP,
- Whether it is shortened,
- Its length, entropy,
- Its counts of various characters and components,
- Features related to the length
- Components of the path,
- Query, sub domain, and primary domain of the URL.
- Label column indicating some classification or labeling information associated with each URL.

```
# Train data
train_data <- train_data %>%
  dplyr::select(url, source, url_has_login, url_has_client, url_has_server,
                url_has_admin, url_has_ip, url_isshorted, url_len,
                url_entropy, url_count_dot, url_count_https, url_count_http,
                url_count_perc, url_count_hyphen, url_count_www,
                url_count_hash, url_count_semicolon, url_count_underscore,
                url_count_ques, url_count_equal, url_count_amp,
                url_count_letter, url_count_digit,
                url_count_sensitive_financial_words,
                url_count_sensitive_words, path_len, query_len,
                query_count_components, pdomain_len, subdomain_len,
                subdomain_count_dot, label
                )
```

Exploratory Data Analysis

During EDA, we are going to perform tasks such as:

- Summarizing the distribution of each variable (count and distribution)
  - Binary Variables
    - Select 6 columns to identify binary variables.Check if the column is numeric and if it possesses only two unique values. If these conditions were met, then the column contains binary data.
- Visualizing relationships between variables (scatter plots, box plots)
  - Distribution of Numerical Variables
    - We filtered out specific columns that mentioned selected variables and used histograms to analyze their distribution.
- Detecting outliers or anomalies (histograms)
  - Treating Outliers
    - We defined a function for identifying and managing outliers and took three parameters: the dataset data, specifying the vector of columns to process, and determining the threshold for anomaly detection criteria, set to 3 standard deviations from the mean. We replace outliers with missing(NA) and print a summary for each column, indicating the number of outliers. Then we ignore all the missing values(NA).
- Exploring correlations between variables
  - Correlation Between Numerical Variables and Target Variable, Label
  - Correlation Between Binary Variable and Target Variable, Label
- Identifying potential patterns or trends in the data

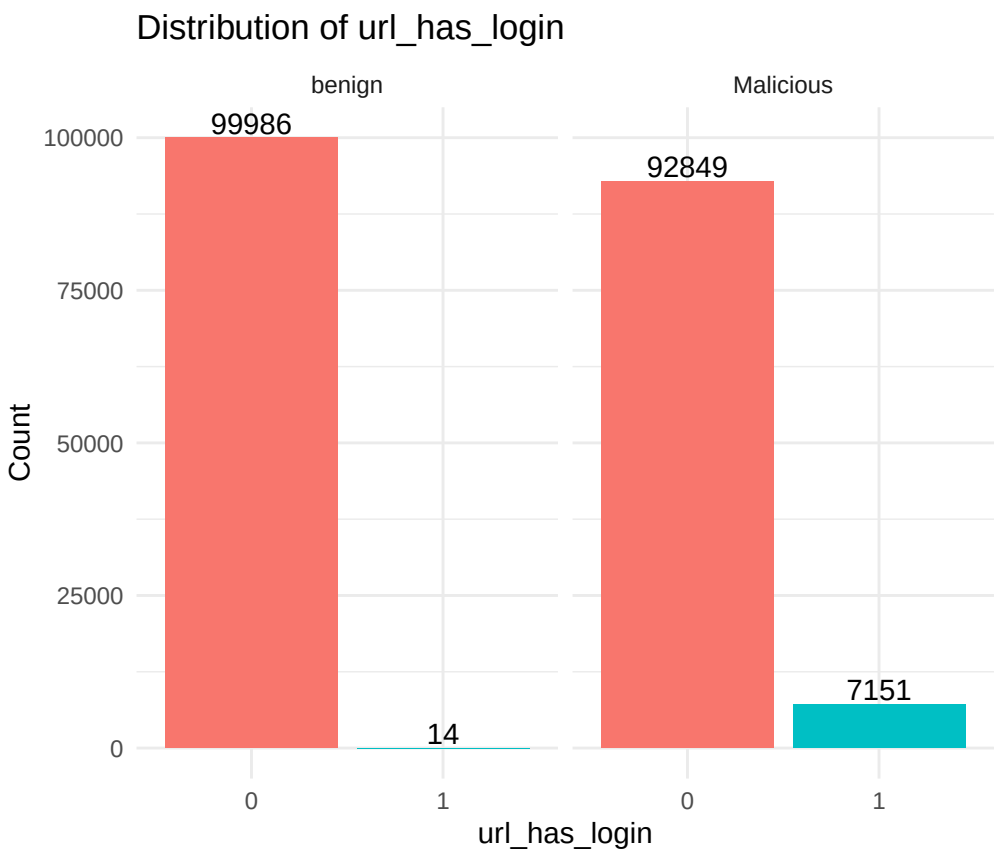
Binary Variables

Here, we're preparing for Exploratory Data Analysis (EDA) by selecting specific columns from the train_data dataframe. We are identifying binary variables within the train_dataset. The criteria involve checking if the column is numeric and if it possesses only two unique values. If these conditions are satisfied, indicating that the column indeed contains binary data, we add it to the binary_vars list.

```
# Initialize an empty list to store binary variables
binary_vars <- list()

# Loop through each column in the dataset
for (col in names(train_data)) {

  # Check if the column is numeric and has only two unique values
  if (is.numeric(train_data[[col]]) && length(unique(train_data[[col]])) == 2) {

    # If the condition is met, add the column to the list
    binary_vars[[col]] <- train_data[[col]]
  }
}

# Convert the list of binary variables to a dataframe
binary_df <- as.data.frame(binary_vars)

# Convert all columns in binary_df to factors
binary_df <- as.data.frame(lapply(binary_df, as.factor))

# Overview
head(binary_df, 5)
```

```
##   url_has_login url_has_client url_has_server url_has_admin url_has_ip
## 1             0              0              0             0          0
## 2             0              0              0             0          0
## 3             0              0              0             0          0
## 4             0              0              0             0          0
## 5             0              0              0             0          0
##   url_isshorted label
## 1             1     0
## 2             0     0
## 3             0     0
## 4             0     0
## 5             0     0
```

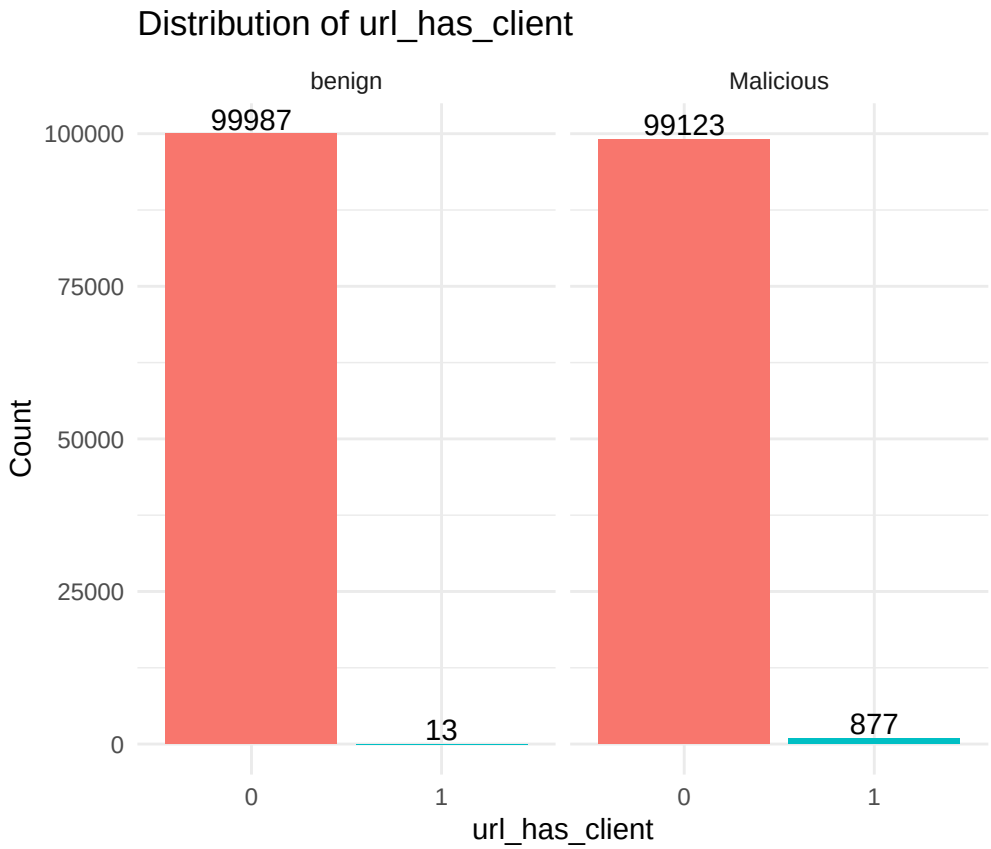
```
# Plotting the count of url_has_login
ggplot(binary_df, aes(x = url_has_login, fill = url_has_login)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_text(stat = "count", aes(label = after_stat(count)),
            vjust = -0.2, position = position_dodge(width = 0.8)) +
  labs(title = "Distribution of url_has_login",
       x = "url_has_login",
       y = "Count") +
  theme_minimal() +
  facet_wrap(~ label, labeller = labeller(label = c("0" = "benign", "1" = "Malicious")))
```



Distribution of url_has_login

For the URLs that are not malicious, 99986 had no login while 14 had login. For the malicious URLs, 92849 had nolog in while 7151 had login.

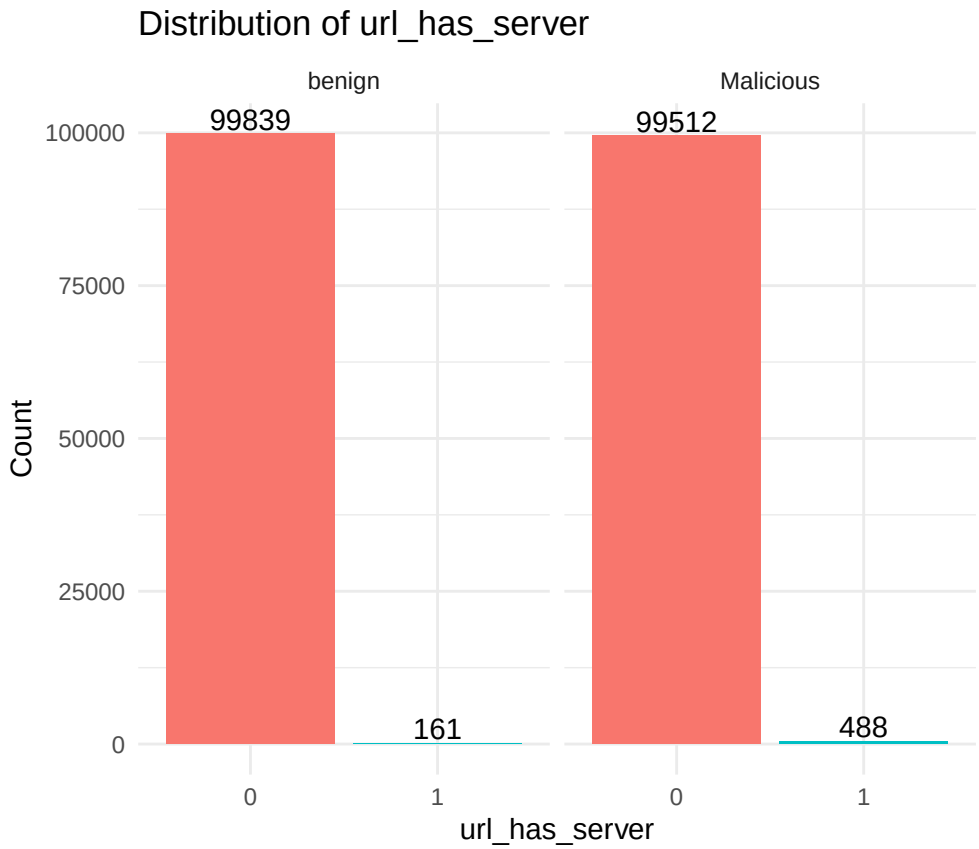
```
ggplot(binary_df, aes(x = url_has_client, fill = url_has_client)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_text(stat = "count", aes(label = after_stat(count)),
            vjust = -0.2, position = position_dodge(width = 0.8)) +
  labs(title = "Distribution of url_has_client",
       x = "url_has_client",
       y = "Count") +
  theme_minimal() +
  facet_wrap(~ label, labeller = labeller(label = c("0" = "benign", "1" = "Malicious")))
```



Distribution of url_has_client

For the URLs that are not malicious, 99987 had no clients while 13 had clients. For the malicious URLs, 99123 had no client while 877 had a client.

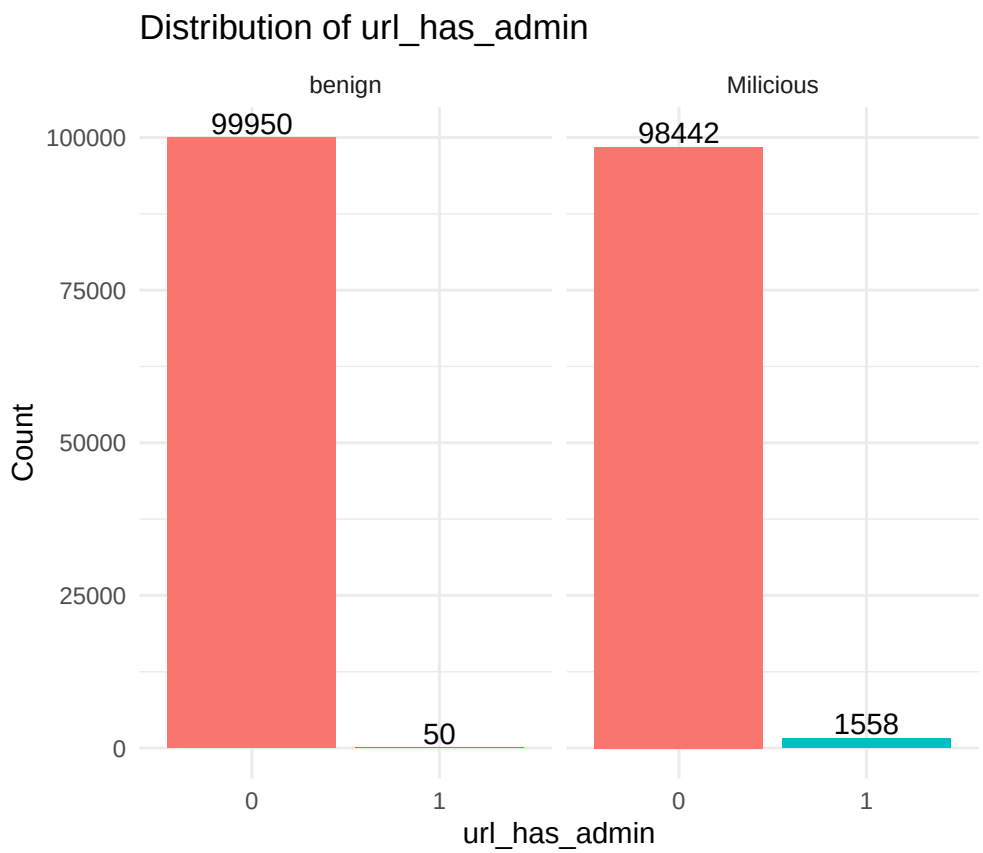
```
# Plotting the count of url_has_server
ggplot(binary_df, aes(x = url_has_server, fill = url_has_server)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_text(stat = "count", aes(label = after_stat(count)),
            vjust = -0.2, position = position_dodge(width = 0.8)) +
  labs(title = "Distribution of url_has_server",
       x = "url_has_server",
       y = "Count" ) +
  theme_minimal() +
  facet_wrap(~ label, labeller = labeller(label = c ("0" = "benign", "1" = "Malicious")))
```



Distribution of url_has_server

For the UELs that are not malicious, 99839 had no server while 161 had a server. For the malicious URLs, 99512 had no server while 488 had a server.

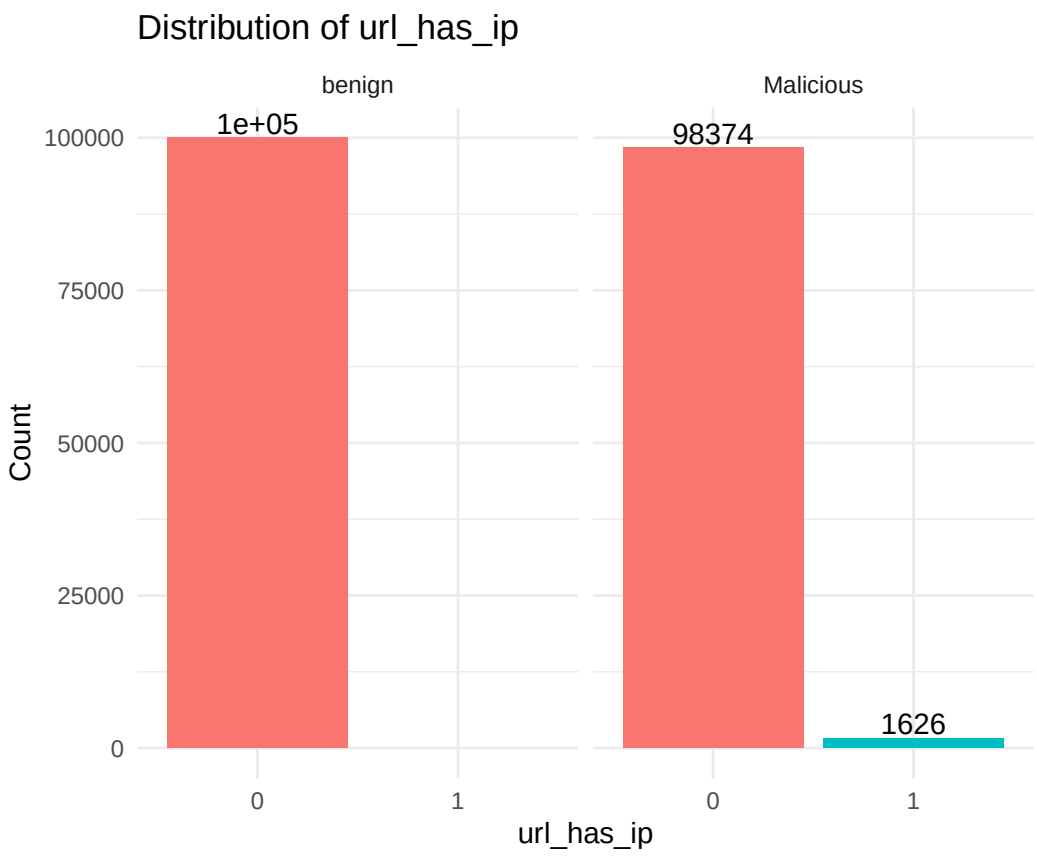
```
#plotting the count of url_has_admin
ggplot(binary_df, aes(x = url_has_admin, fill = url_has_admin)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_text(stat = "count", aes(label = after_stat(count)),
            vjust = -0.2, position = position_dodge(width = 0.8)) +
  labs(title = "Distribution of url_has_admin",
       x = "url_has_admin",
       y = "Count" ) +
  theme_minimal() +
  facet_wrap(~ label, labeller = labeller(label = c("0" = "benign", "1" = "Milicious")))
```



Distribution of url_has_admin

For the URLs that are not malicious, 99950 had no admin while 50 had admin. For the malicious URLs, 98442 had no admin while 1558 had admin.

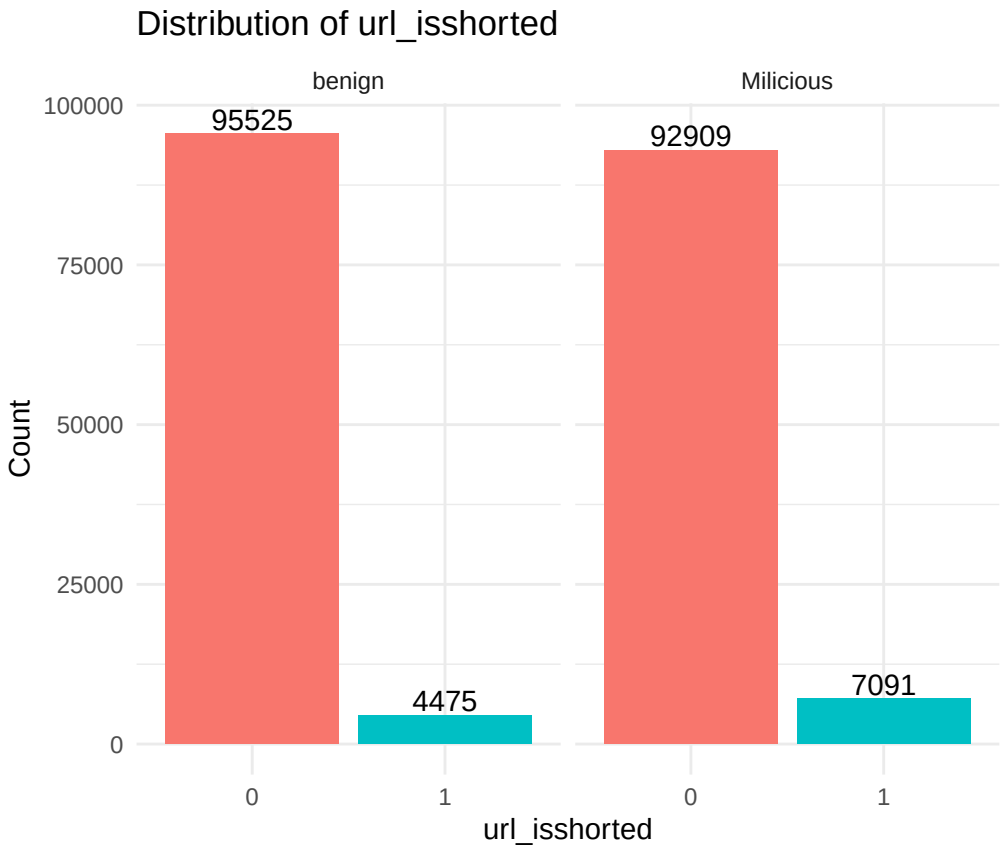
```
# Plotting the count of url_has_ip
ggplot(binary_df, aes(x = url_has_ip, fill = url_has_ip)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_text(stat = "count", aes(label = after_stat(count)),
            vjust = -0.2, position = position_dodge(width = 0.8)) +
  labs(title = "Distribution of url_has_ip",
       x = "url_has_ip",
       y = "Count" ) +
  theme_minimal() +
  facet_wrap(~ label, labeller = labeller(label = c("0" = "benign", "1" = "Malicious")))
```



Distribution of url__has__ip

For the URLs that are not malicious, all had no IP. For the malicious URLs, 98374 had no IP while 1626 had an IP.

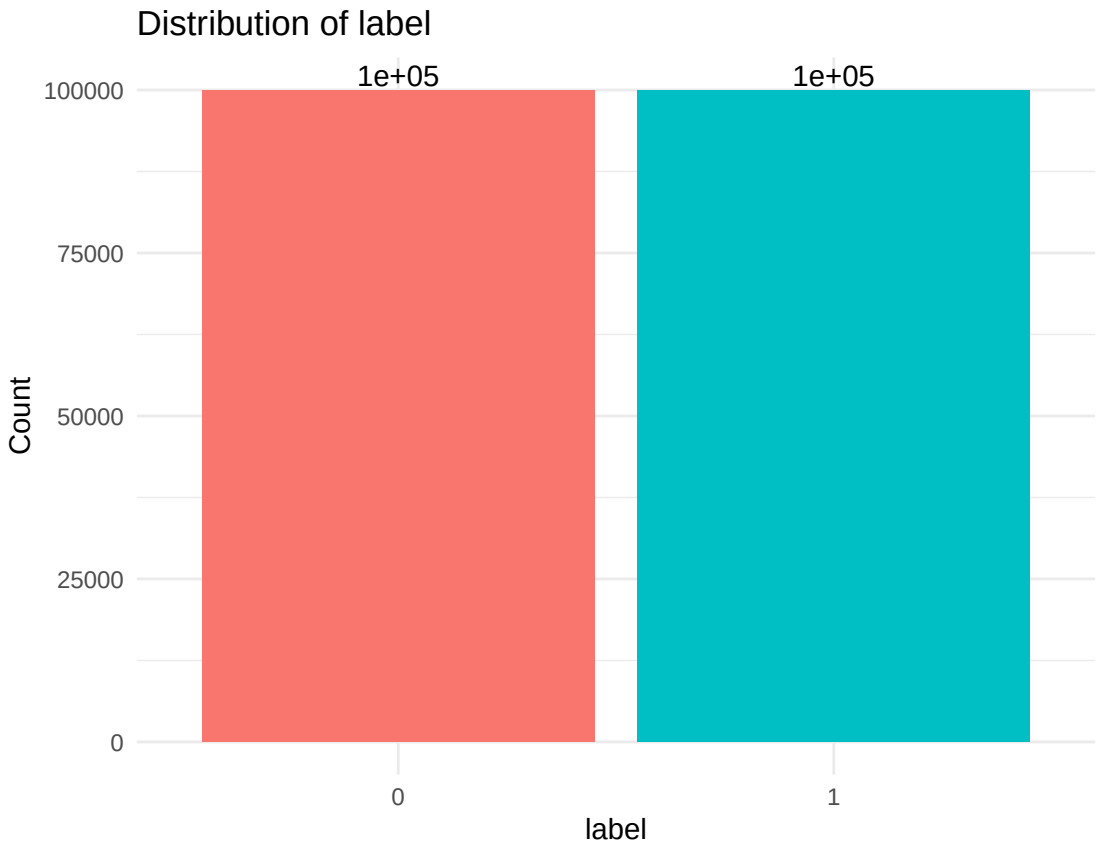
```
# Plotting the count of url_isshorted
ggplot(binary_df, aes(x = url_isshorted, fill = url_isshorted)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_text(stat = "count", aes(label = after_stat(count)),
            vjust = -0.2, position = position_dodge(width = 0.8)) +
  labs(title = "Distribution of url_isshorted",
       x = "url_isshorted",
       y = "Count" ) +
  theme_minimal() +
  facet_wrap(~ label, labeller = labeller(label = c("0" = "benign", "1" = "Milicious")))
```



Distribution of url_isshorted

For the URLs that are not malicious, 4475 were shortened while 95525 were not. For the malicious URLs, 7091 were shortened while 92909 were not.

```
# Plotting the count of label
ggplot(binary_df, aes(x = label, fill = label)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_text(stat = "count", aes(label = after_stat(count)),
            vjust = -0.2, position = position_dodge(width = 0.8)) +
  labs(title = "Distribution of label",
       x = "label",
       y = "Count" ) +
  theme_minimal()
```



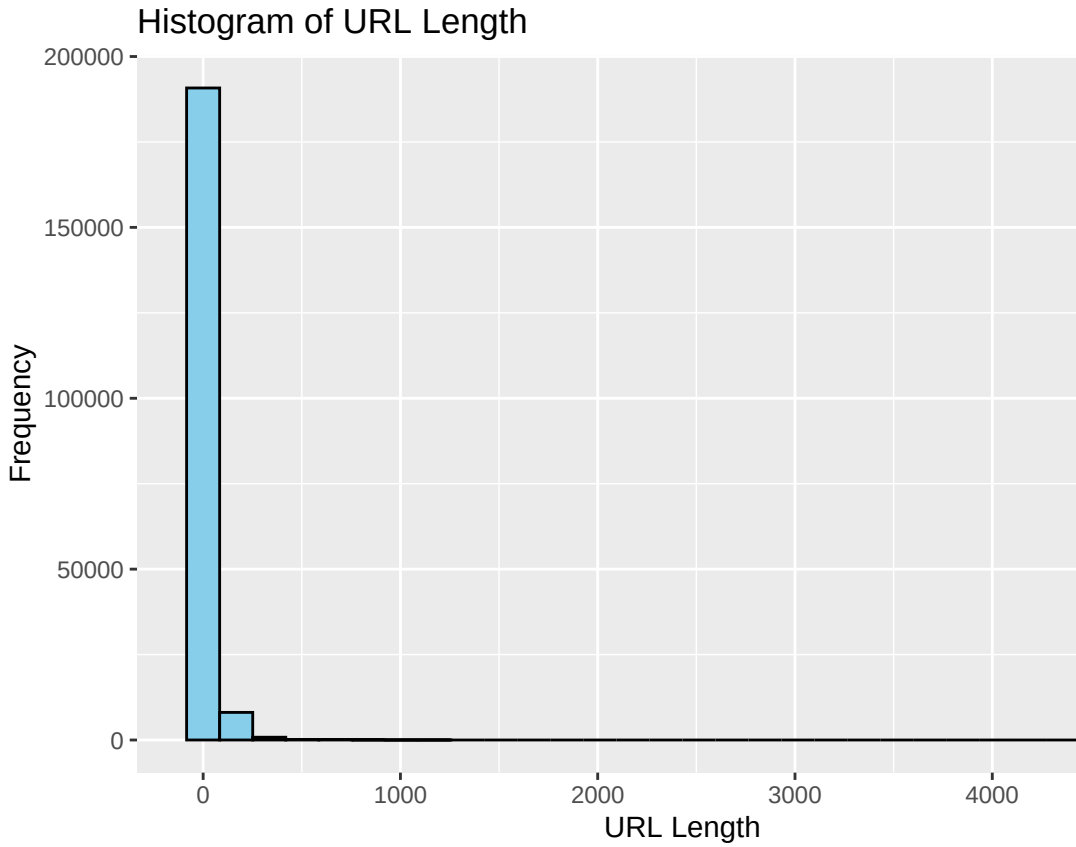
Distribution of Label

Our target variable which is Label, was balanced.

Distribution of Numerical Variables

```
# Selecting all columns except the ones mentioned in binary
numerical_columns <- train_data[, !names(train_data) %in% names(binary_df)]

# Remove the first two columns
numerical_columns <- numerical_columns %>%
  dplyr::select(-url, -source)
```

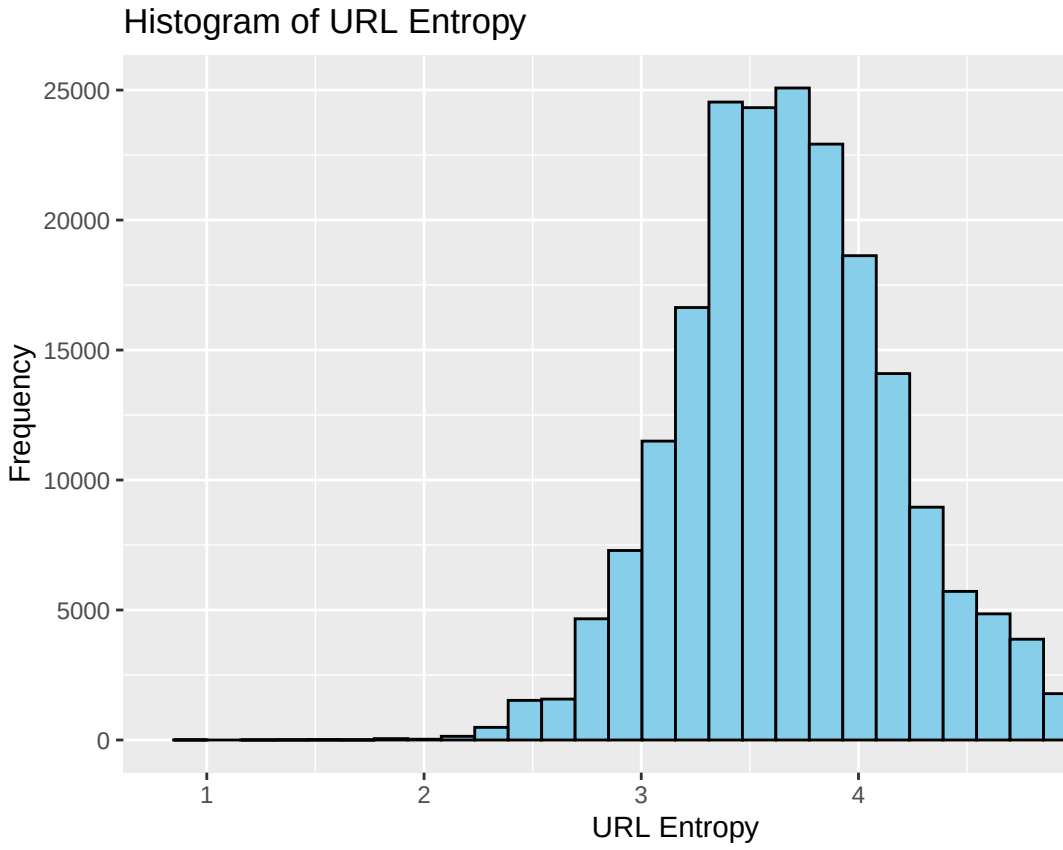
```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_len)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Length", x = "URL Length", y = "Frequency")
```



Distribution of URL_LEN

The URL len is positively skewed meaning the majority of the URLs have a shorter length. However, we can already see the presence of outliers in the URL length column. We will treat this by dropping all the outliers.

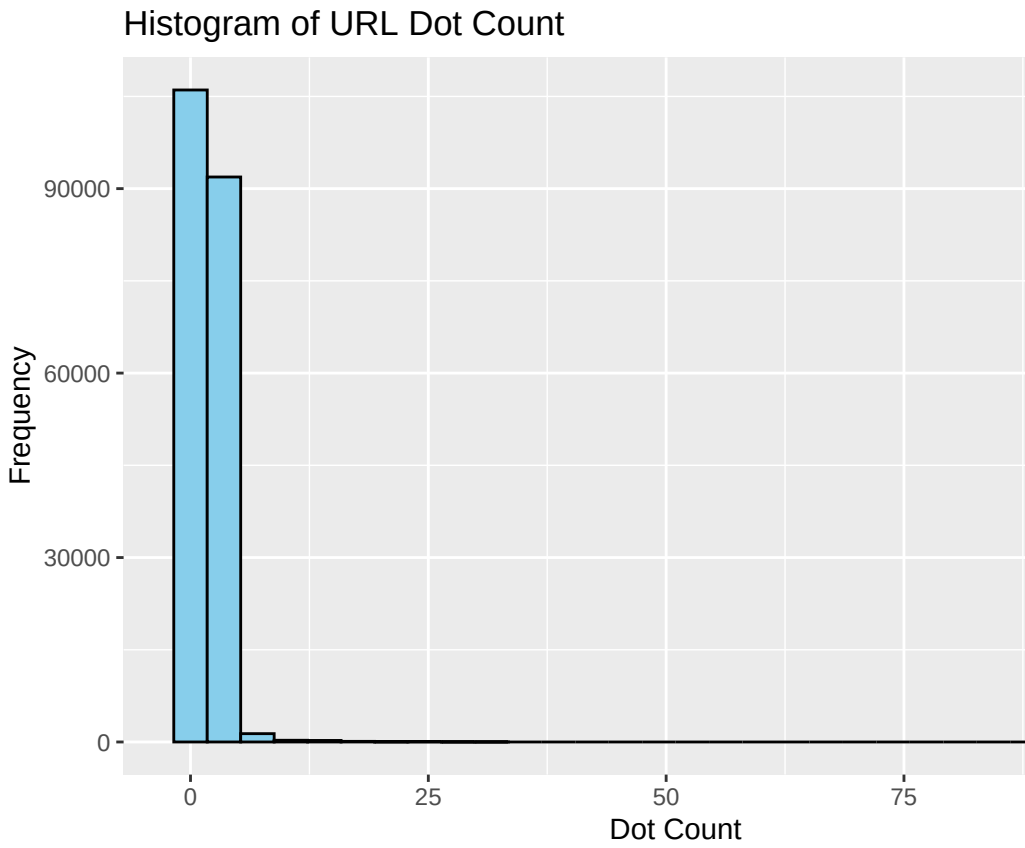
```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_entropy)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Entropy", x = "URL Entropy", y = "Frequency")
```



Distribution of url_entropy

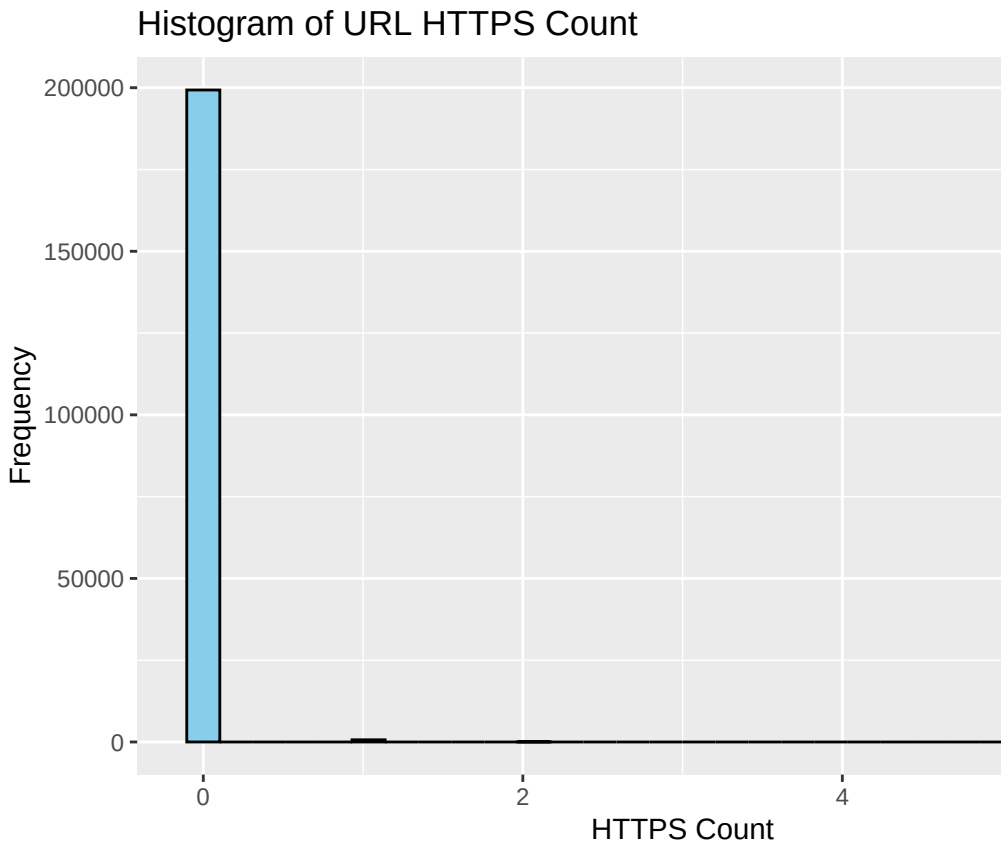
URL entropy is almost normally distributed with a slight negative skew.

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_dot)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Dot Count", x = "Dot Count", y = "Frequency")
```



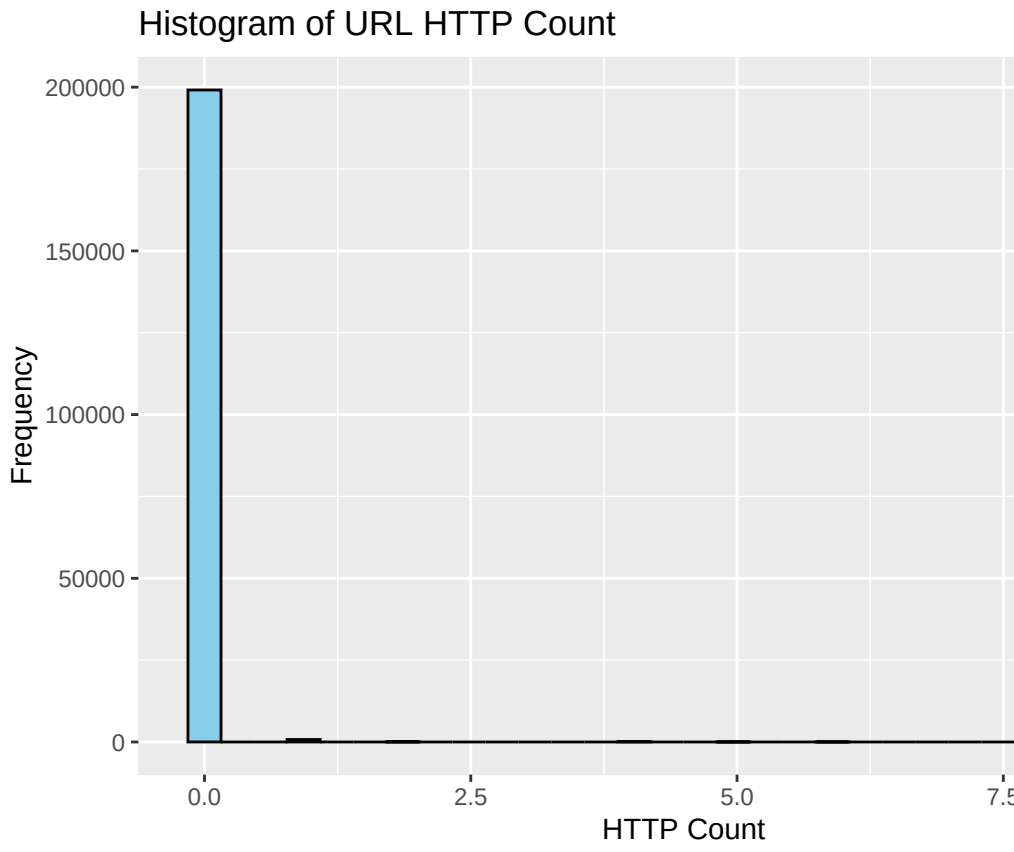
Distribution of url_count_dot

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_https)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL HTTPS Count", x = "HTTPS Count", y = "Frequency")
```



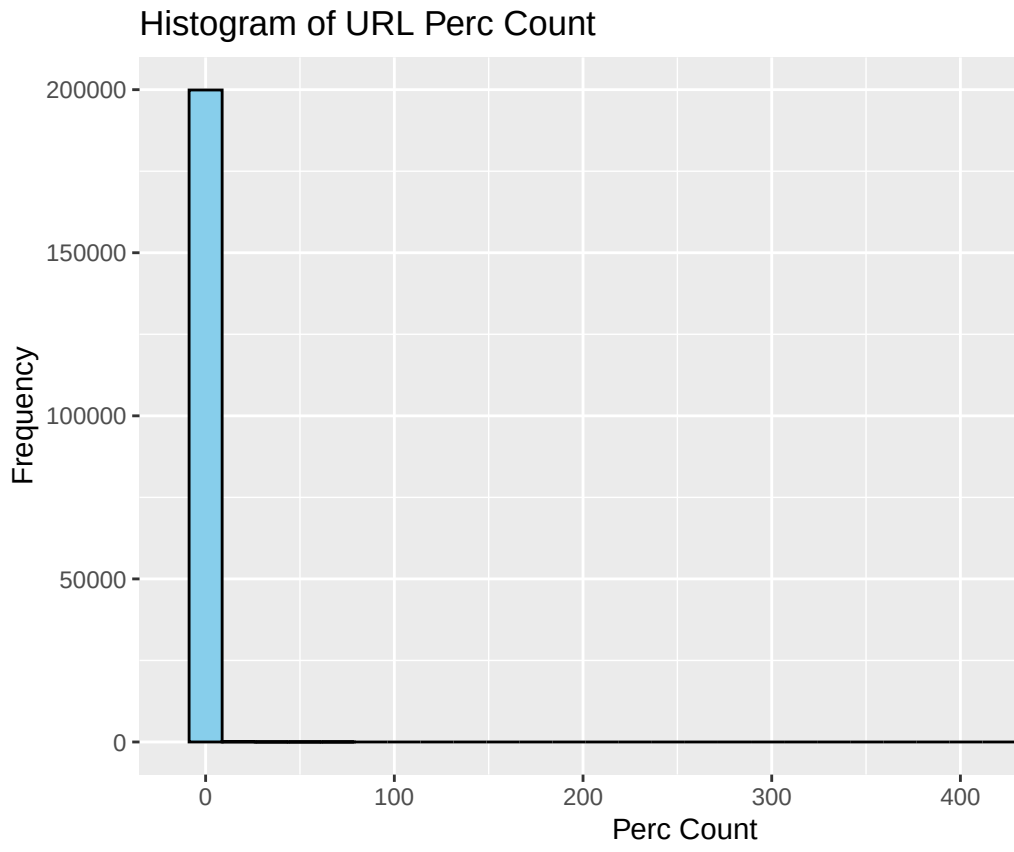
Distribution of url_count_https

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_http)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL HTTP Count", x = "HTTP Count", y = "Frequency")
```



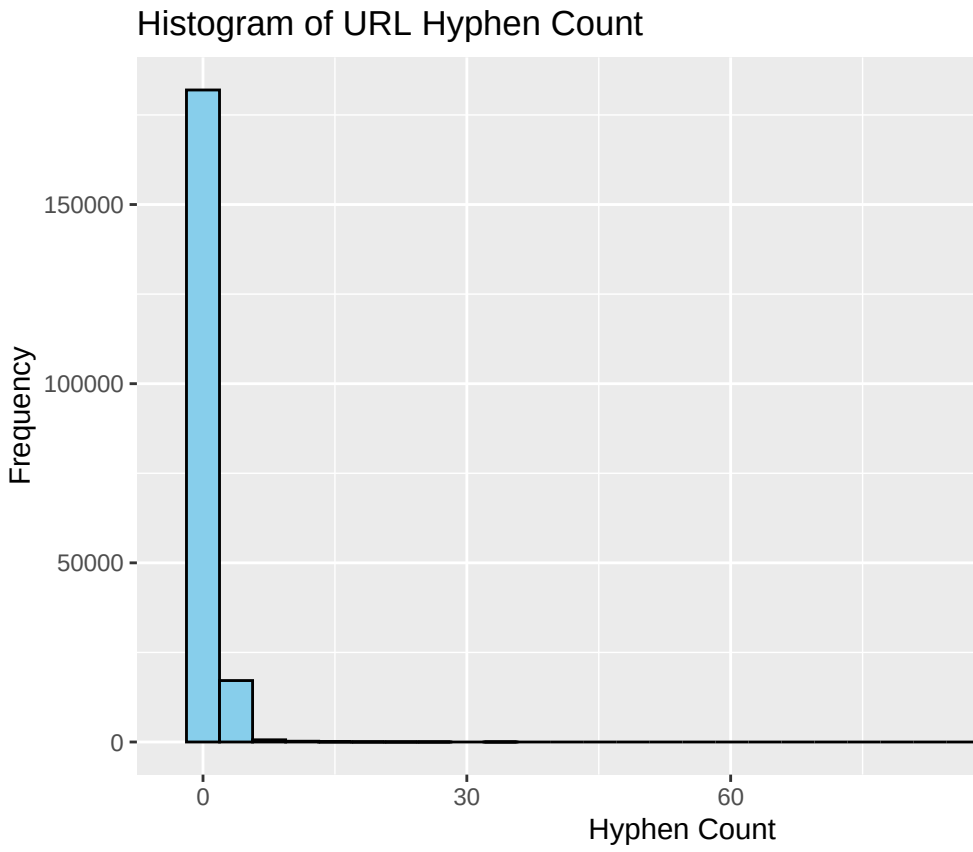
Distribution of url_count_http

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_perc)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Perc Count", x = "Perc Count", y = "Frequency")
```



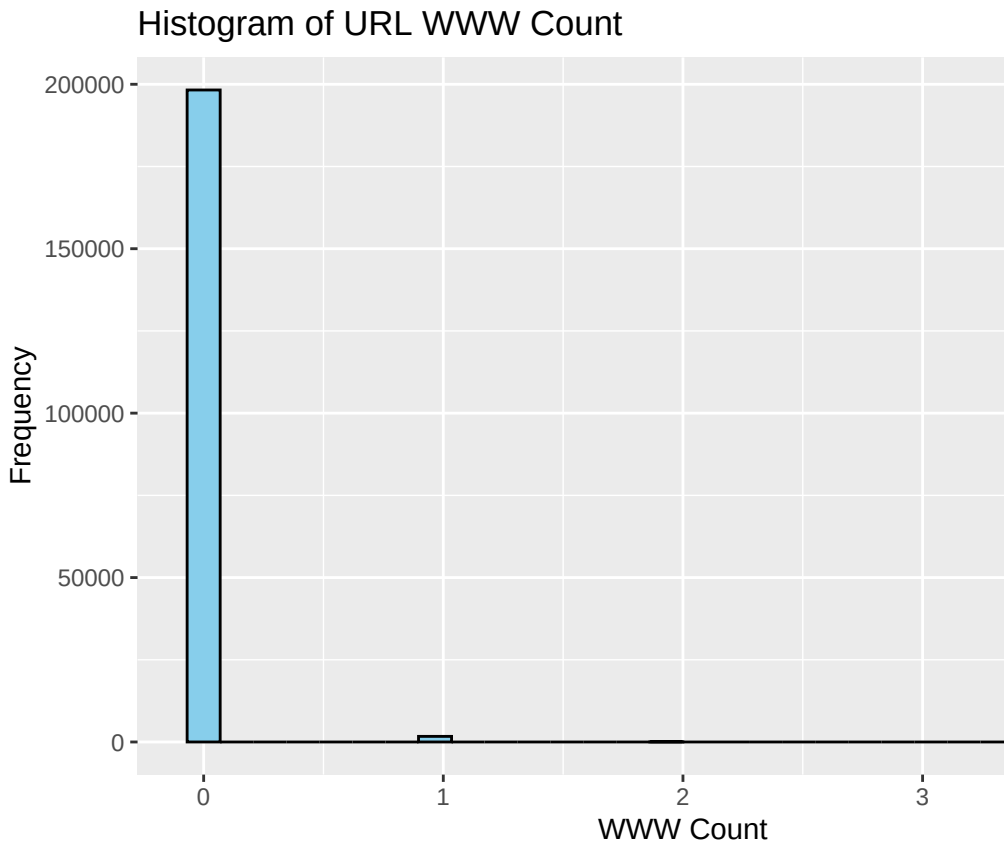
Distribution of url_count_perc

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_hyphen)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Hyphen Count", x = "Hyphen Count", y = "Frequency")
```



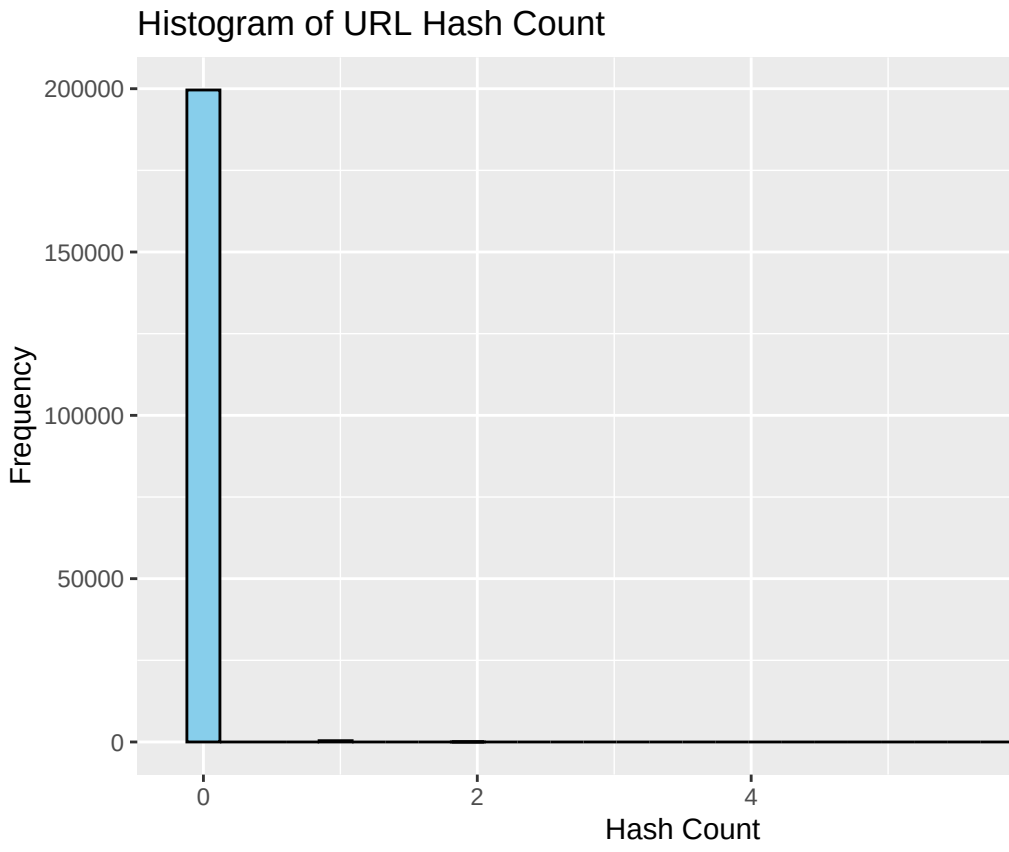
Distribution of url_count_hyphen

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_www)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL WWW Count", x = "WWW Count", y = "Frequency")
```



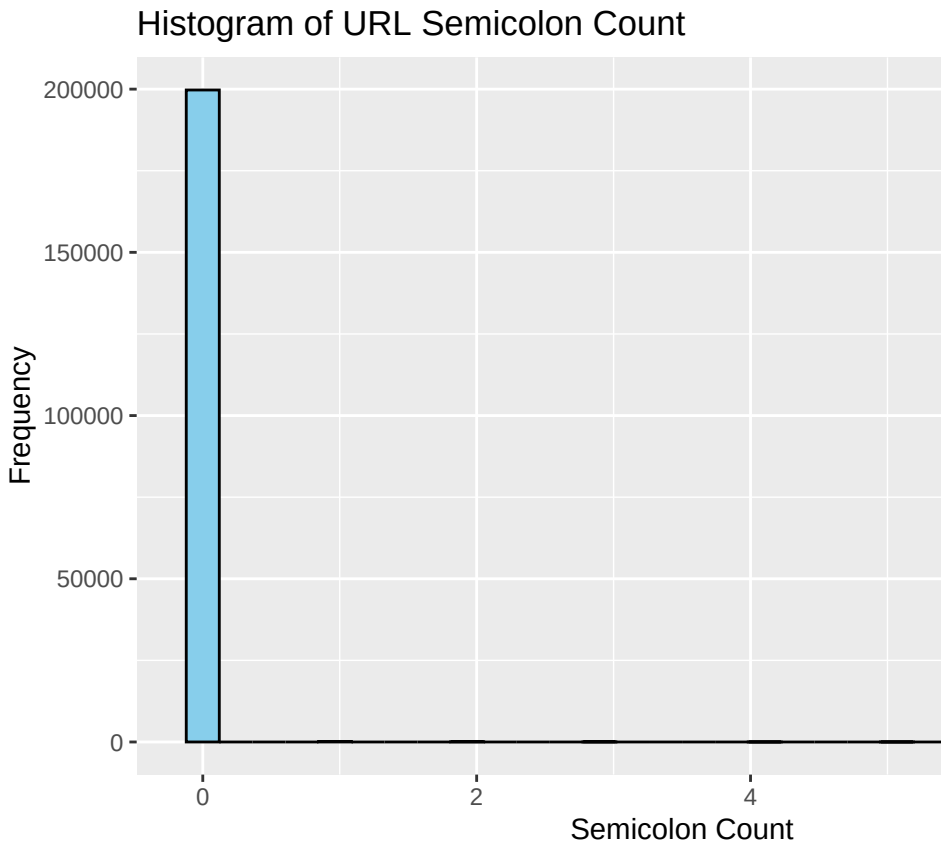
Distribution of url_count_www

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_hash)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Hash Count", x = "Hash Count", y = "Frequency")
```



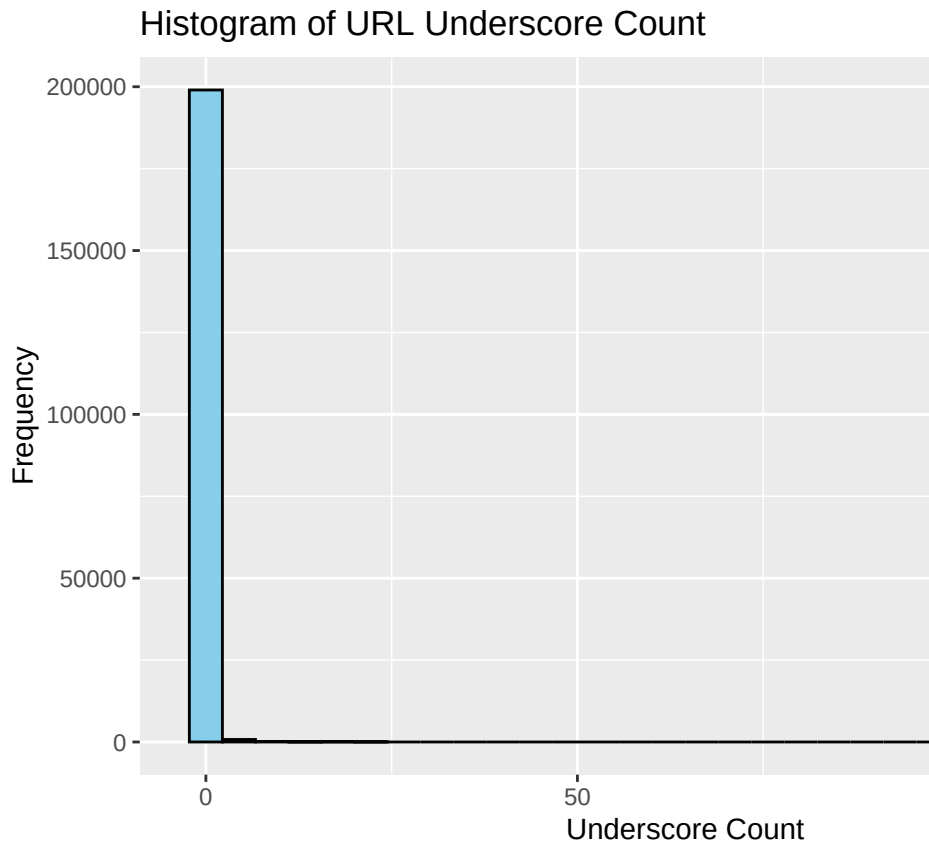
Distribution of url_count_hash

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_semicolon)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Semicolon Count", x = "Semicolon Count", y = "Frequency")
```



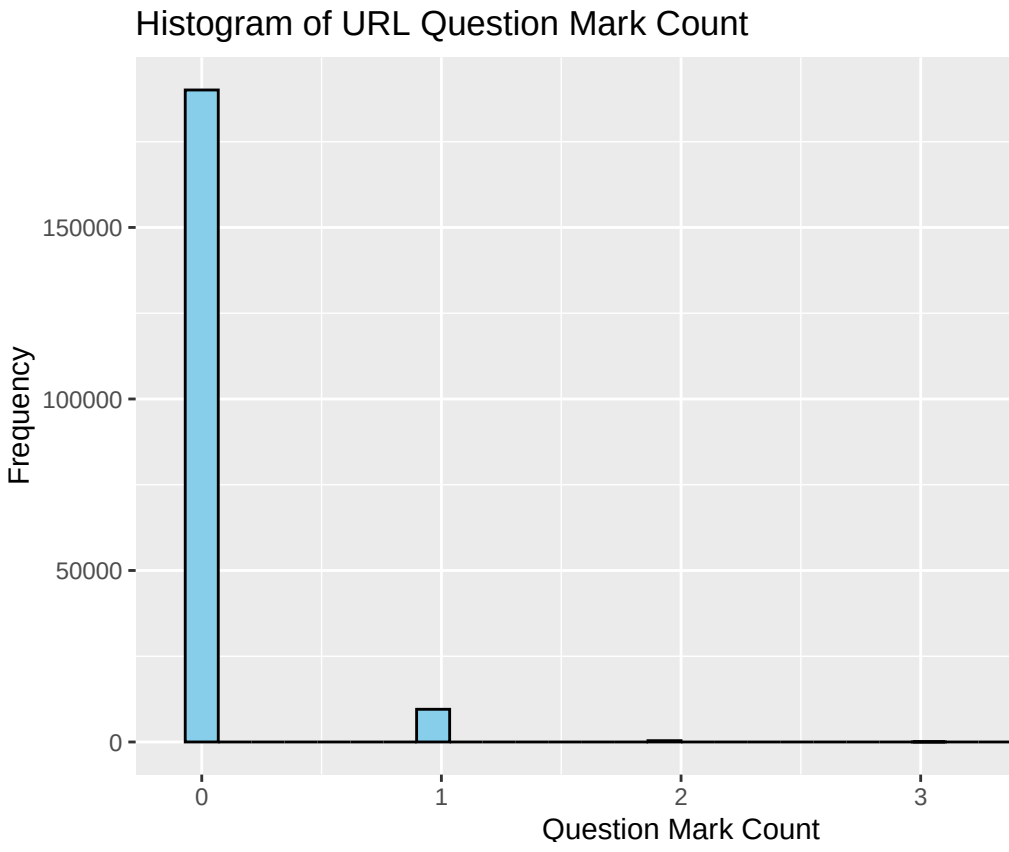
Distribution of url_count_semicolon

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_underscore)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Underscore Count", x = "Underscore Count", y = "Frequency")
```



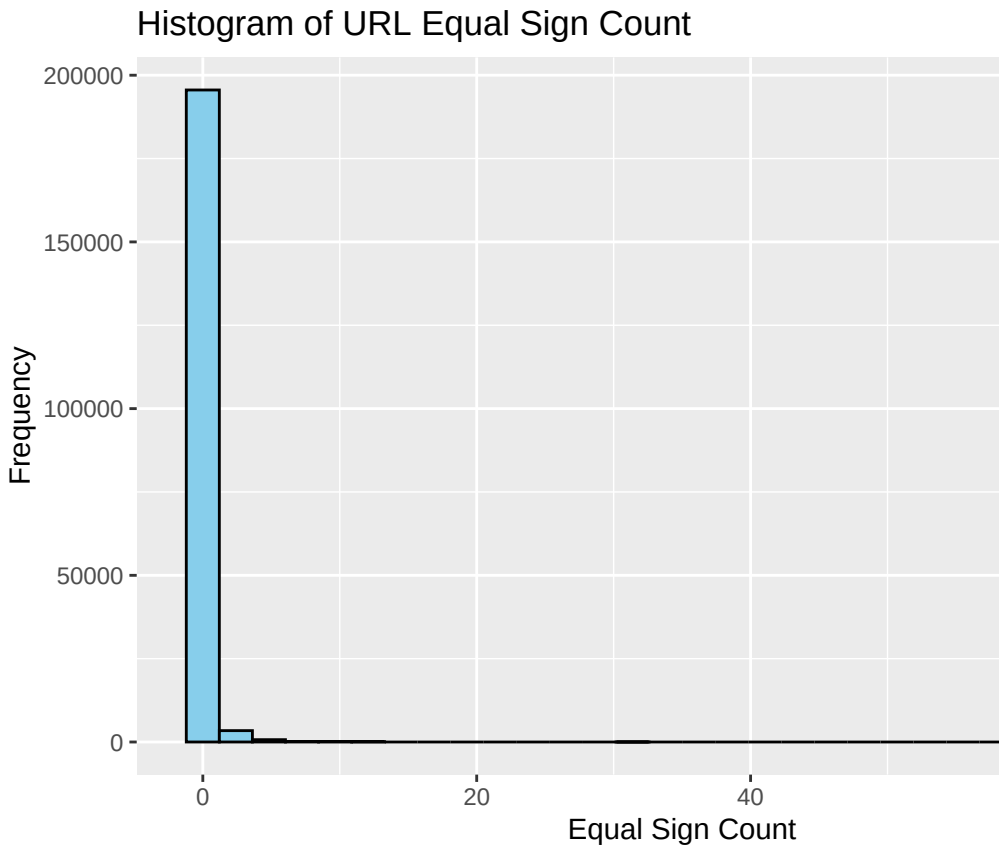
Distribution of url_count_underscore

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_ques)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Question Mark Count", x = "Question Mark Count", y = "Frequency")
```



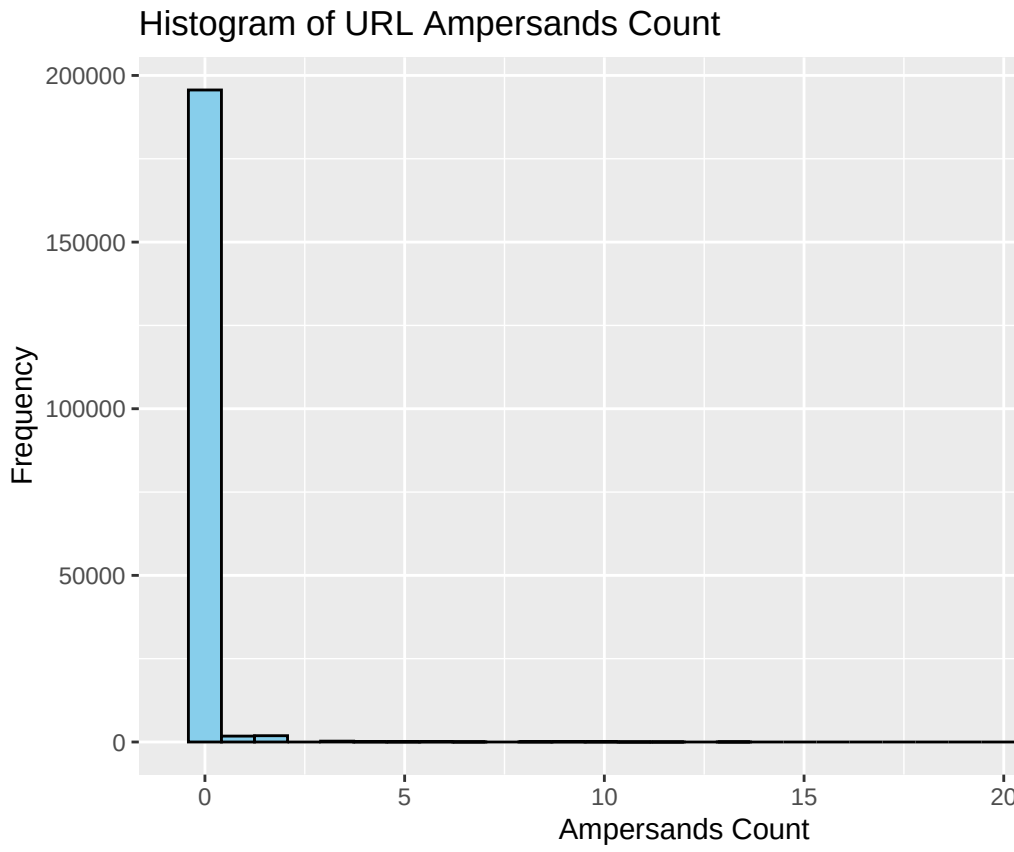
Distribution of url_count_ques

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_equal)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Equal Sign Count", x = "Equal Sign Count", y = "Frequency")
```



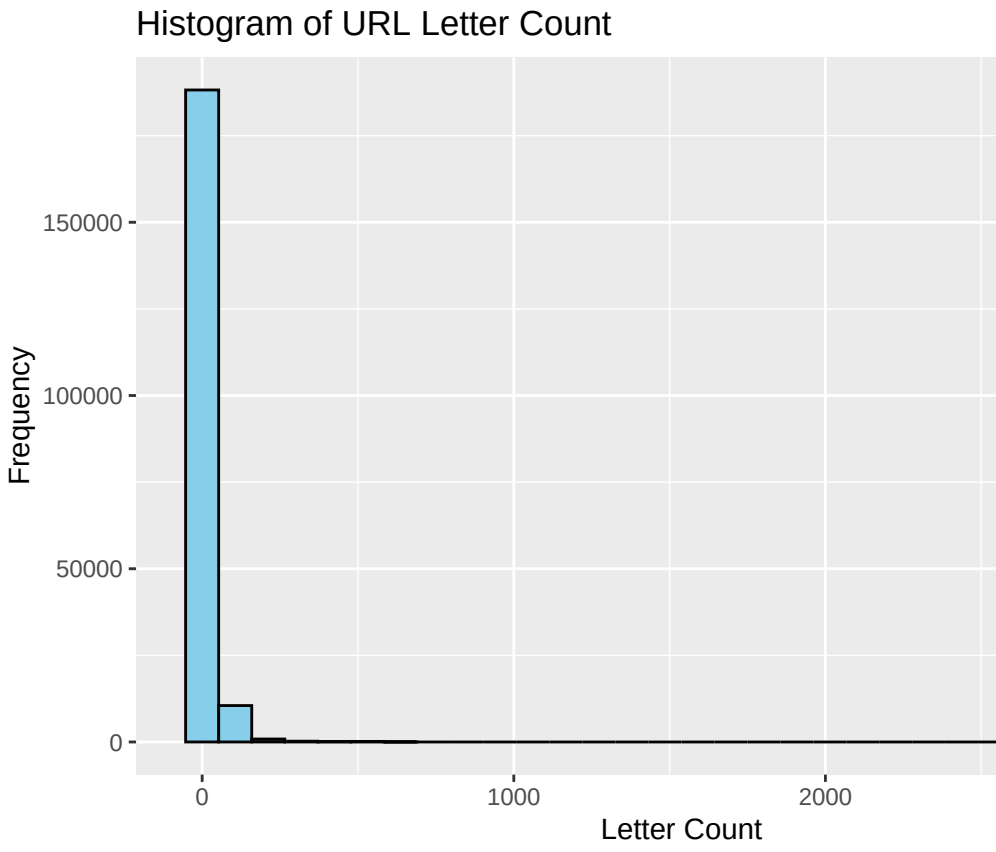
Distribution of url_count_equal

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_amp)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Ampersands Count", x = "Ampersands Count", y = "Frequency")
```



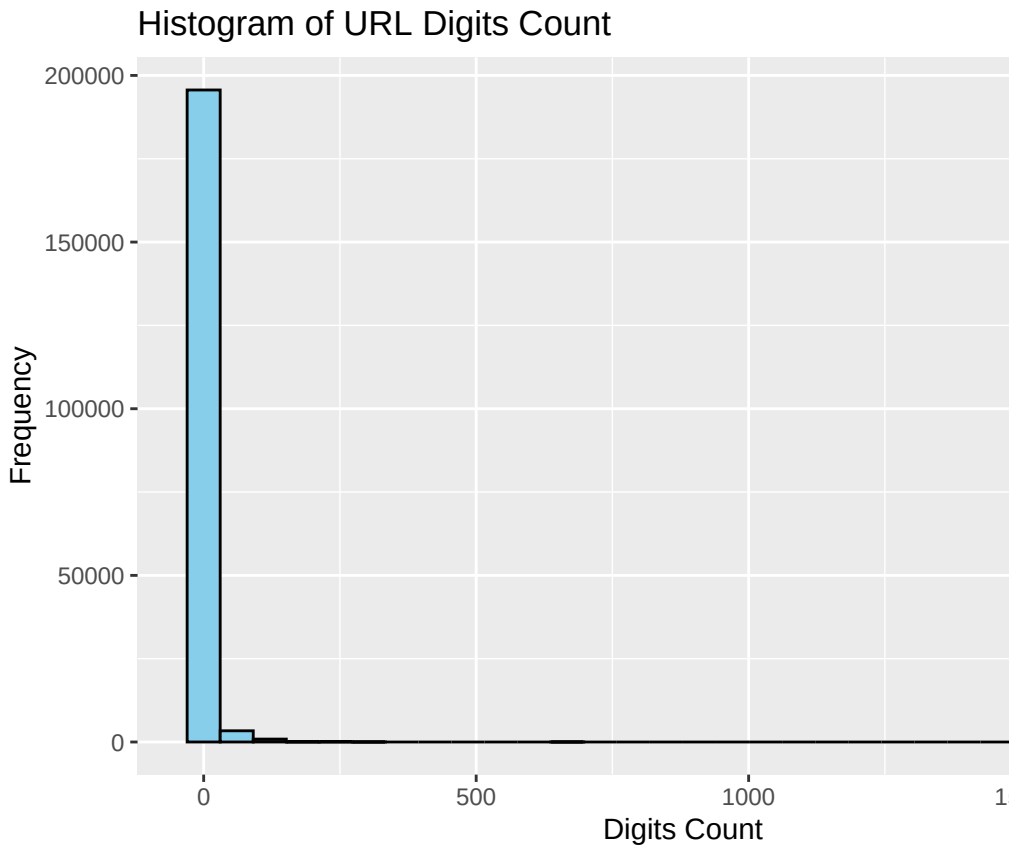
Distribution of url_count_amp

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_letter)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Letter Count", x = "Letter Count", y = "Frequency")
```



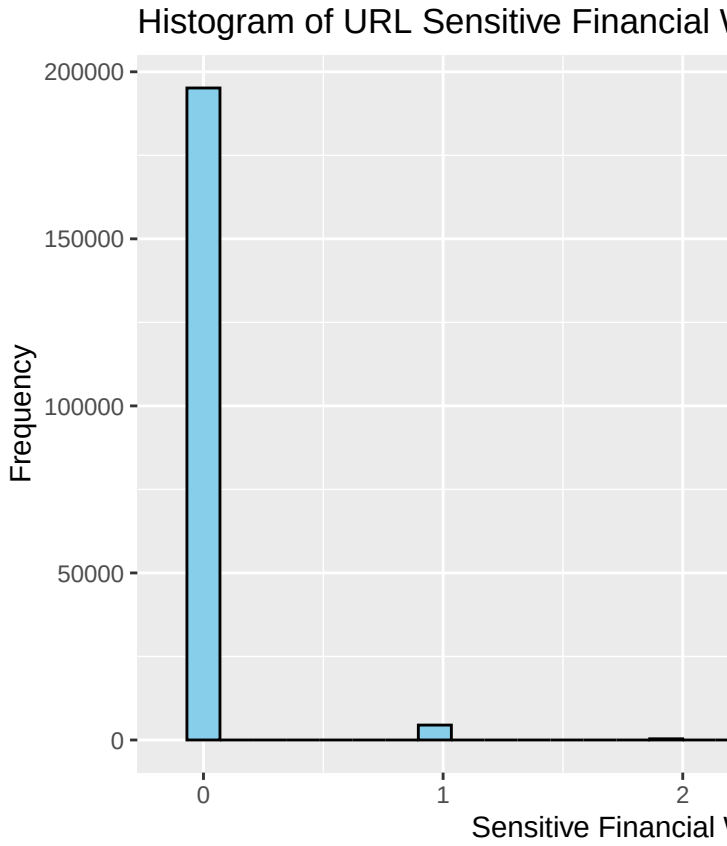
Distribution of url_count_letter

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_digit)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Digits Count", x = "Digits Count", y = "Frequency")
```



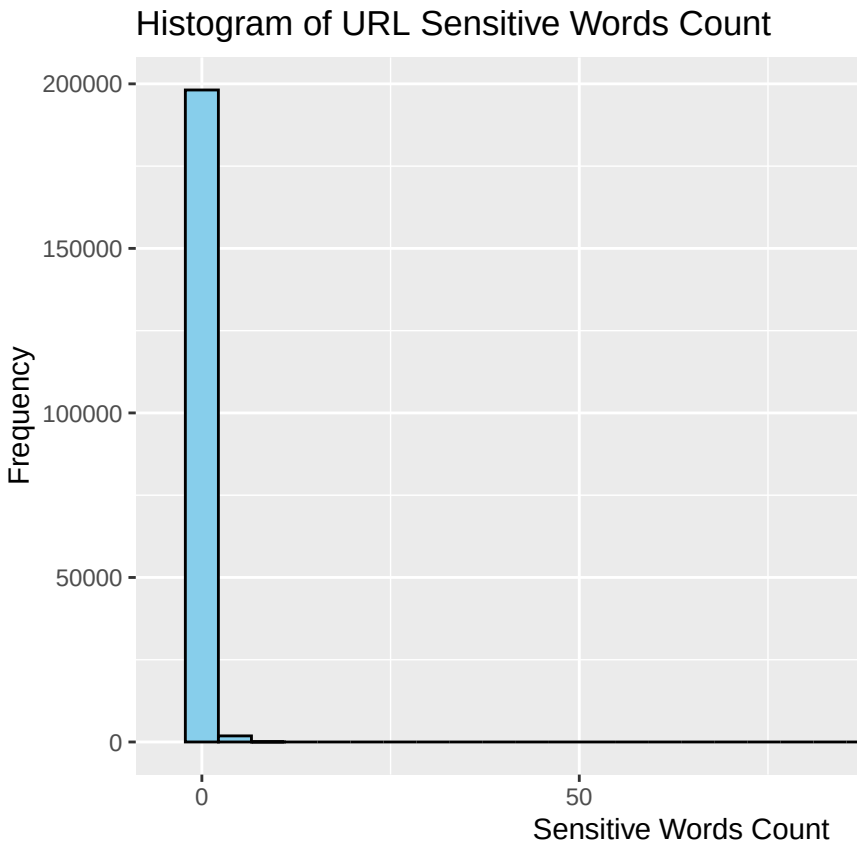
Distribution of url_count_digit

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_sensitive_financial_words)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Sensitive Financial Words Count",
       x = "Sensitive Financial Words Count", y = "Frequency")
```



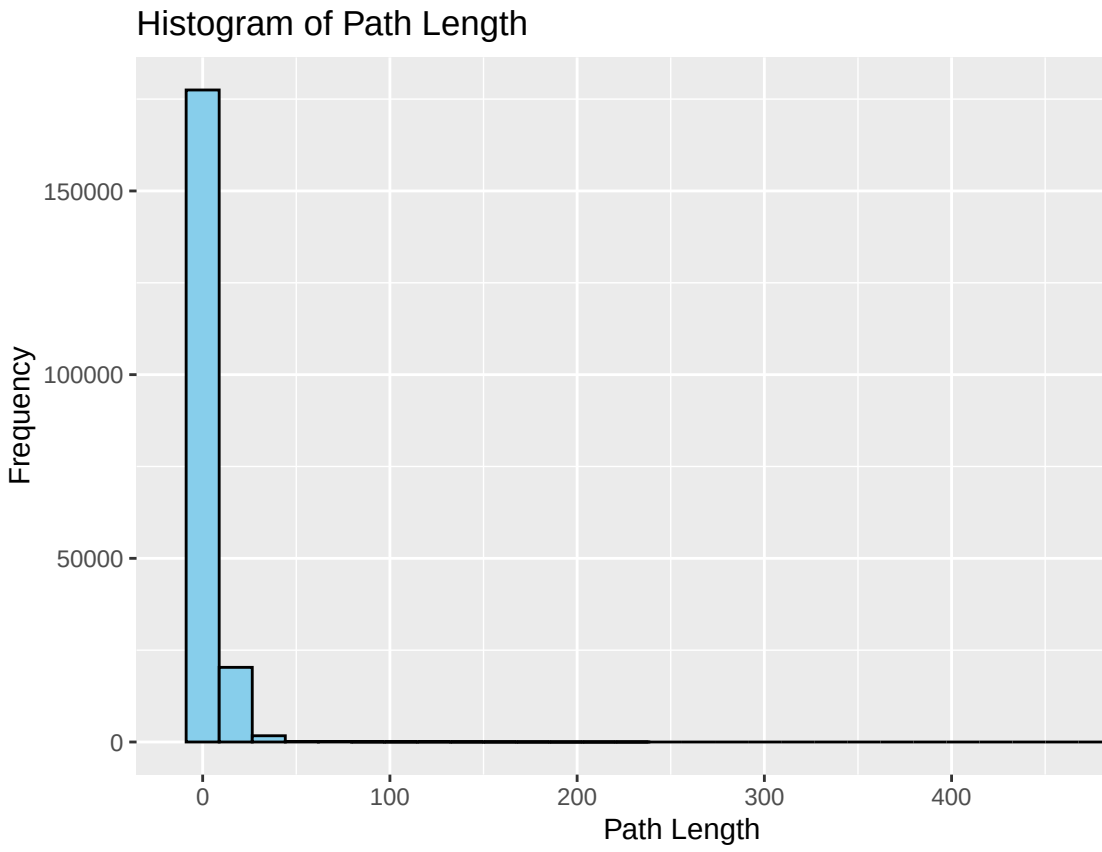
Distribution of url_count_sensitive_financial_words

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = url_count_sensitive_words)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of URL Sensitive Words Count",
       x = "Sensitive Words Count", y = "Frequency")
```



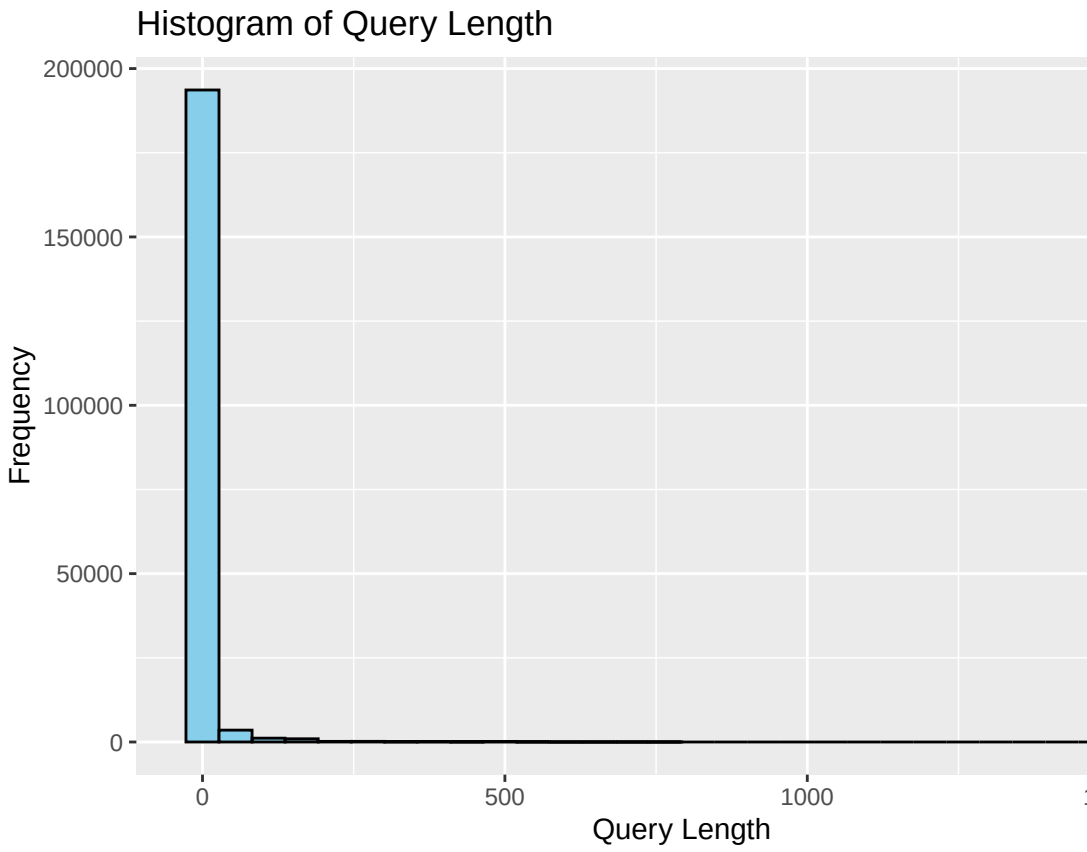
Distribution of url_count_sensitive_words

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = path_len)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Path Length", x = "Path Length", y = "Frequency")
```



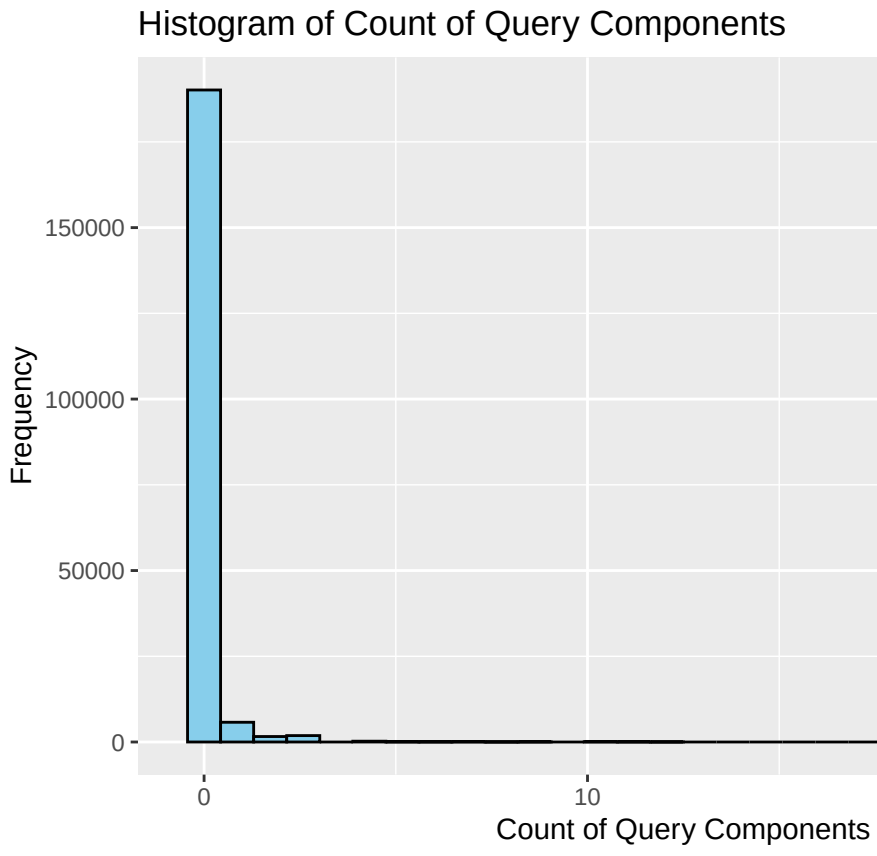
Distribution of path_len

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = query_len)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Query Length", x = "Query Length", y = "Frequency")
```



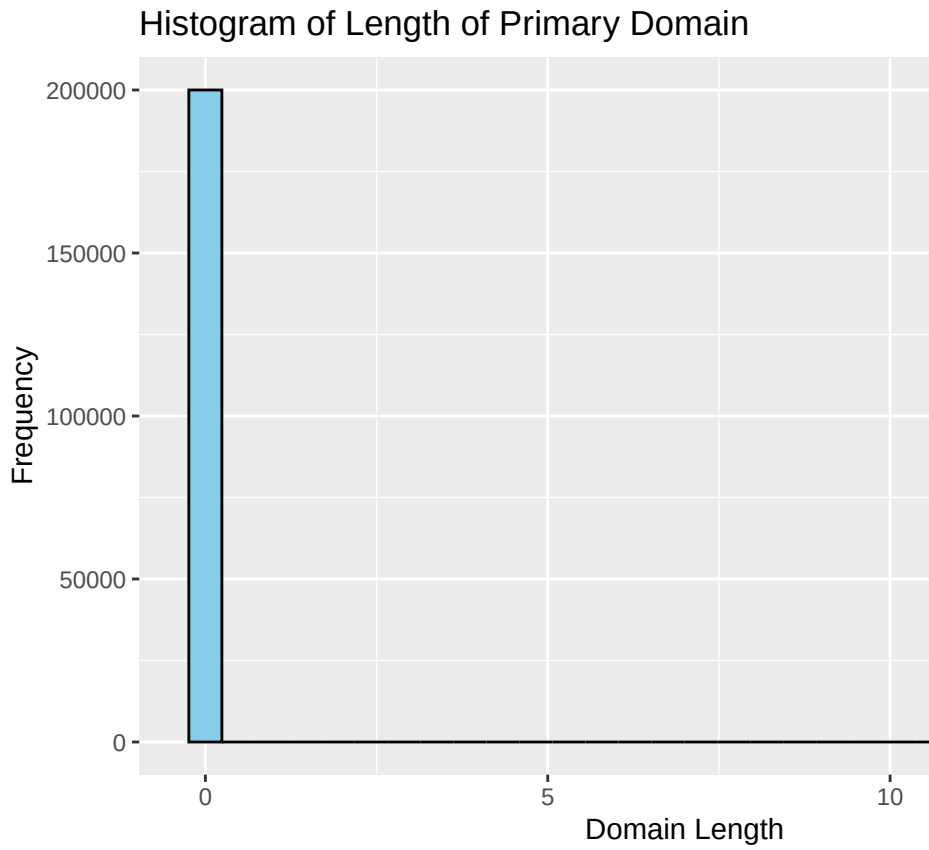
Distribution of query_len

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = query_count_components)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Count of Query Components",
       x = "Count of Query Components", y = "Frequency")
```



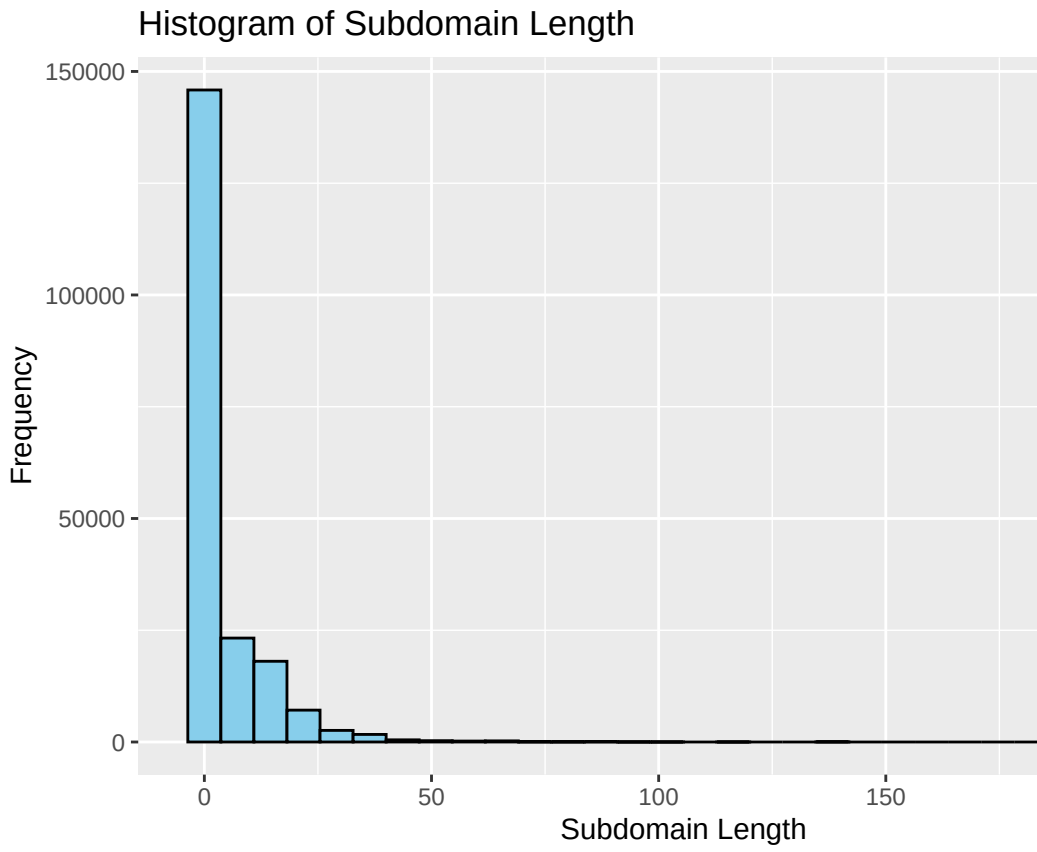
Distribution of query_count_components

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = pdomain_len)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Length of Primary Domain",
       x = "Domain Length", y = "Frequency")
```



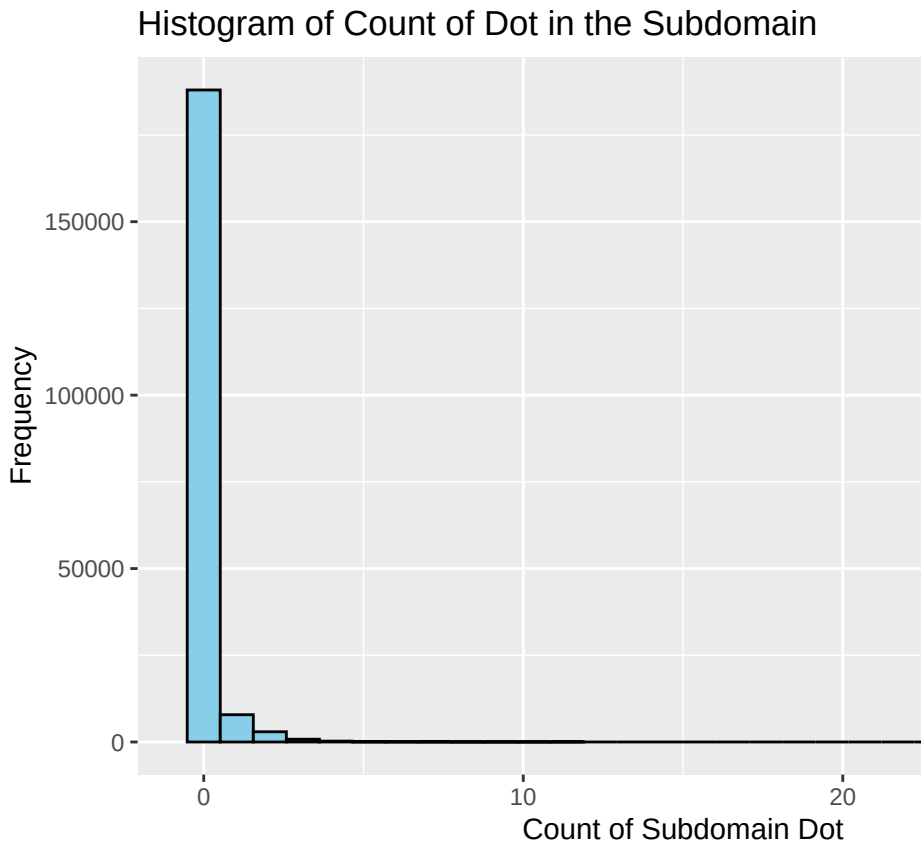
Distribution of primary domain length

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = subdomain_len)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Subdomain Length",
       x = "Subdomain Length", y = "Frequency")
```



Distribution of subdomain len

```
# Plotting histogram using ggplot2
ggplot(numerical_columns, aes(x = subdomain_count_dot)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Count of Dot in the Subdomain",
       x = "Count of Subdomain Dot", y = "Frequency")
```



Distribution of subdomain_count_dot

Other than URL entropy, all the variables are positively skewed with a clear sign of large outliers. These outliers could skew our analysis results and therefore, we will deal with them in the Data Preparation section.

Treating Outliers

We defined a function for identifying and managing outliers within a dataset. The function takes three parameters: the dataset data, a vector specifying the columns to be processed columns, and a threshold determining the outlier deterction criteria, set to 3 standard deviations from the mean. Detected outliers are replaced with NA values, and a summary is printed for each column, indicating the number of outliers removed. Then all the missing valuas (NA) are omitted.

```
# Removing outliers
remove_outliers <- function(data, columns, threshold = 3) {
  # Iterate over each specified column
  for (col in columns) {
    # Calculate mean and standard deviation
    mean_value <- mean(data[[col]], na.rm = TRUE)
    sd_value <- sd(data[[col]], na.rm = TRUE)

    # Identify outliers
    outliers <- data[data[[col]] > mean_value + threshold * sd_value |
                     data[[col]] < mean_value - threshold * sd_value, ]

    # Treat outliers (replace with NA)
    data[[col]][data[[col]] > mean_value + threshold * sd_value |
                data[[col]] < mean_value - threshold * sd_value ] <- NA

    # Print summary
```

```
    cat("Outliers removed for column:", col, "\n")
    cat("Number of outliers removed:", nrow(outliers), "\n\n")
  }

  return(data)

}

# Call the function
train_data_cleaned <- remove_outliers(train_data, names(numerical_columns), threshold = 3)
```

```
## Outliers removed for column: url_len
## Number of outliers removed: 3083
##
## Outliers removed for column: url_entropy
## Number of outliers removed: 561
##
## Outliers removed for column: url_count_dot
## Number of outliers removed: 2073
##
## Outliers removed for column: url_count_https
## Number of outliers removed: 700
##
## Outliers removed for column: url_count_http
## Number of outliers removed: 861
##
## Outliers removed for column: url_count_perc
## Number of outliers removed: 116
##
## Outliers removed for column: url_count_hyphen
## Number of outliers removed: 3354
##
## Outliers removed for column: url_count_www
## Number of outliers removed: 1743
##
## Outliers removed for column: url_count_hash
## Number of outliers removed: 399
##
## Outliers removed for column: url_count_semicolon
## Number of outliers removed: 268
##
## Outliers removed for column: url_count_underscore
## Number of outliers removed: 995
##
## Outliers removed for column: url_count_ques
## Number of outliers removed: 9926
##
## Outliers removed for column: url_count_equal
## Number of outliers removed: 3067
##
## Outliers removed for column: url_count_amp
## Number of outliers removed: 2600
##
## Outliers removed for column: url_count_letter
```

```
## Number of outliers removed: 2450
##
## Outliers removed for column: url_count_digit
## Number of outliers removed: 2332
##
## Outliers removed for column: url_count_sensitive_financial_words
## Number of outliers removed: 4853
##
## Outliers removed for column: url_count_sensitive_words
## Number of outliers removed: 1870
##
## Outliers removed for column: path_len
## Number of outliers removed: 2378
##
## Outliers removed for column: query_len
## Number of outliers removed: 2762
##
## Outliers removed for column: query_count_components
## Number of outliers removed: 4121
##
## Outliers removed for column: pdomain_len
## Number of outliers removed: 2
##
## Outliers removed for column: subdomain_len
## Number of outliers removed: 3567
##
## Outliers removed for column: subdomain_count_dot
## Number of outliers removed: 4172
```

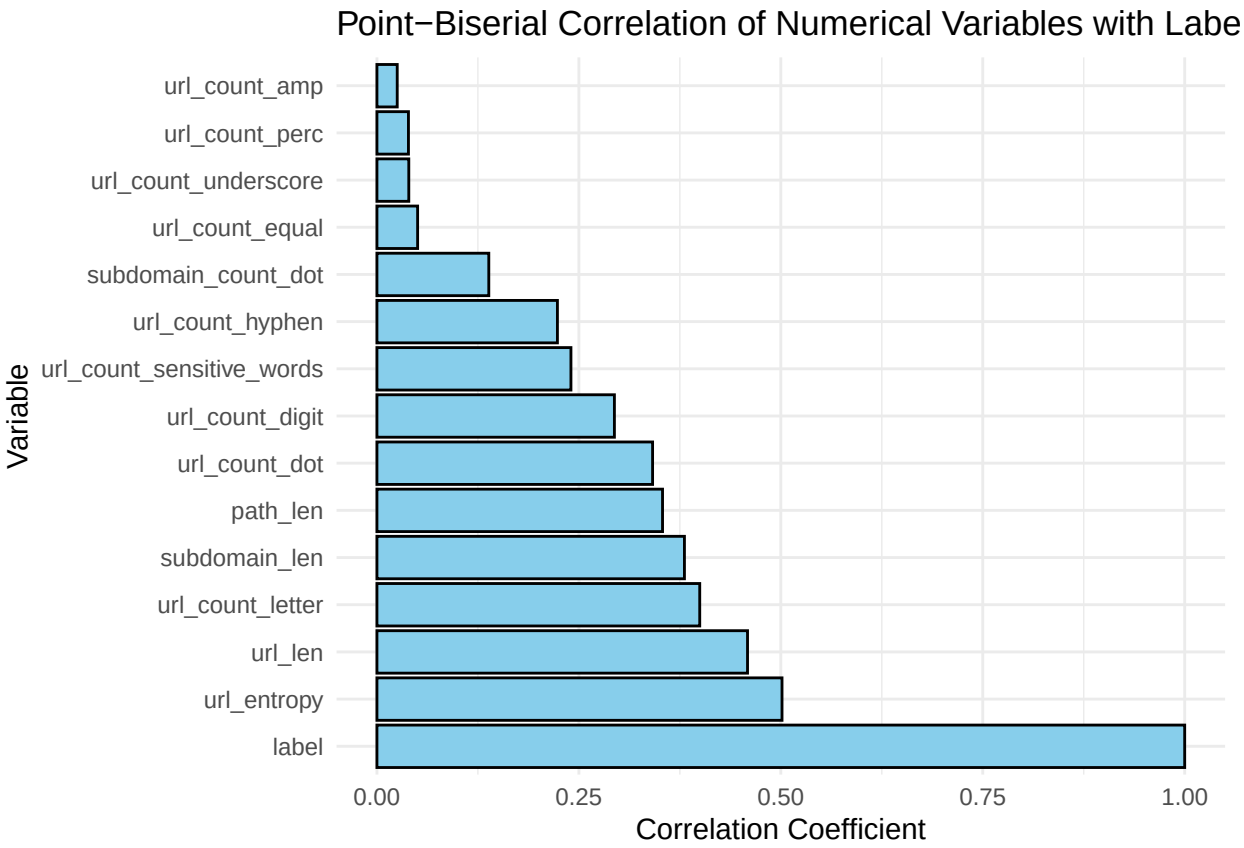
```r
# Remove nulls
train_data_cleaned <- na.omit(train_data_cleaned)
```

Relationship Between Our Target Variable, Label, and All Other Variables

Correlation Between Numerical Variables and Target Variable, Label Then, we perform correlation analysis between numerical variables and a binary label in a dataset. We define a function named correlation_plot that calculates point-biserial correlation coefficients for each numerical variable with respect to the binary label. This function iterates over the numerical columns, computes the correlation coefficient using the biserial.cor function, and appends the results to the global dataframe correlation_data. Finally, the function generates a bar plot of correlations ordered from highest to lowest, providing insights into the strength and direction of association between each numerical variable and the binary label.

```r
# Select binary variables
binary_cols <- names(binary_df)

# Remove the label
binary_cols <- setdiff(binary_cols, "label")

# Initialize an empty dataframe to store correlation coefficients
correlation_data <- data.frame(variable = character(),
                               correlation = numeric(),
                               stringsAsFactors = FALSE)

correlation_plot <- function(data, label_column) {
```

```r
  # Select numerical columns
  data2 <- select_if(data, is.numeric)

  # Remove the binary variables
  numerical_cols <- data2[, !(names(data2) %in% binary_cols)]

  # Iterate over numerical columns
  for (col in names(numerical_cols)) {

    # Calculate point-biserial correlation
    corr <- biserial.cor(data[[col]], as.numeric(data[[label_column]]),
                         use = c("all.obs"), level = 2)

    # Store correlation coefficient in the global dataframe
    correlation_data <<- bind_rows(correlation_data,
                                   data.frame(variable = col,
                                              correlation = corr))
  }

  # Plot correlation coefficients with bars ordered from max to min correlation
  ggplot(na.omit(correlation_data), aes(x = reorder(variable, -correlation), y = correlation)) +
    geom_bar(stat = "identity", fill = "skyblue", color ="black") +
    coord_flip() +
    labs(title = "Point-Biserial Correlation of Numerical Variables with Label",
         x = "Variable",
         y = "Correlation Coefficient") +
    theme_minimal()
}

# Calling the function
correlation_plot(train_data_cleaned, "label")
```




```
# Showing the dataframe
print(correlation_data)
```

```
##                                 variable correlation
## 1                                url_len  0.45878250
## 2                            url_entropy  0.50146422
## 3                           url_count_dot  0.34126717
## 4                         url_count_https         NaN
## 5                          url_count_http         NaN
## 6                          url_count_perc  0.03901660
## 7                        url_count_hyphen  0.22343556
## 8                           url_count_www         NaN
## 9                          url_count_hash         NaN
## 10                    url_count_semicolon         NaN
## 11                   url_count_underscore  0.03944986
## 12                         url_count_ques         NaN
## 13                        url_count_equal  0.05039230
## 14                          url_count_amp  0.02513906
## 15                       url_count_letter  0.39964435
## 16                        url_count_digit  0.29400054
## 17 url_count_sensitive_financial_words         NaN
## 18           url_count_sensitive_words  0.24018686
## 19                               path_len  0.35356281
## 20                              query_len         NaN
## 21              query_count_components         NaN
## 22                             pdomain_len         NaN
## 23                          subdomain_len  0.38067851
```

```
## 24                 subdomain_count_dot  0.13849100
## 25                               label  1.00000000
```

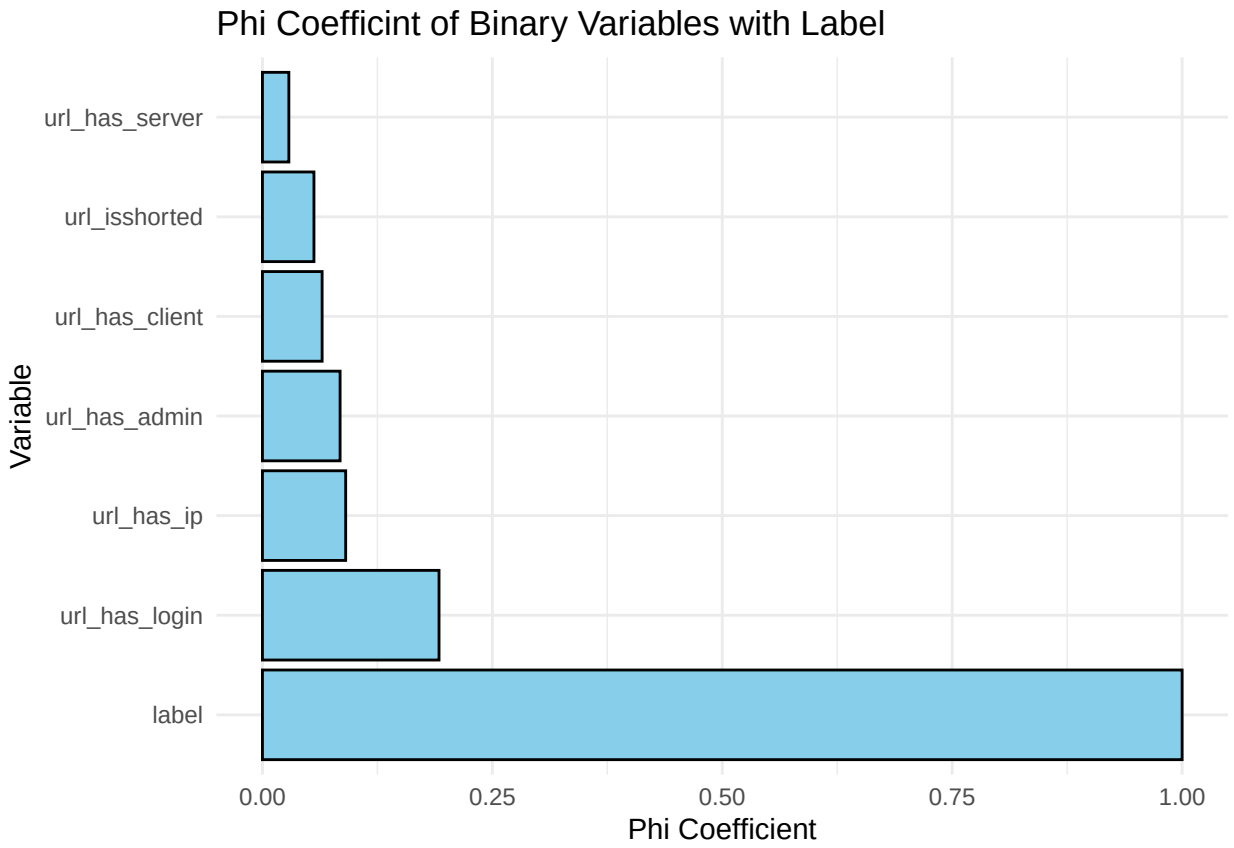
The plot above visualizes the data in the data frame printed above. We can see that we have numerical columns that have medium strength correlation with our target variable which is label. Some of the columns had NaNs so we will not be moving forward with any column that had no correlation with our target variable.

It is important to note that all our variables are positively correlated to our target variable

URL entropy has the strongest correlation with our target variable label, followed by URL Length, URL letter count and sub domain length in that order.

Correlation Between Binary Variable and Target Variable Label In here, we computed the Phi coefficient, a measure of association between two binary variables, specifically between binary variables and a binary label within a dataset. The compute_phi_coefficients function iterates over each column in the dataset, checking if the column is binary(i.e., a factor with two levels). If the condition is met, the Phi coefficient is calculated between the binary variable and the binary label.The results are appended to the result dataframe.Finally, the function generates a bar plot visualizing the Phi coefficients for each binary variable with respect to the label. The function returns both the result dataframe and the genarated plot.

```r
# Initialize empty dataframe
result <- data.frame(variable = character(),
                     label = character(),
                     phi_coefficient = numeric(),
                     stringsAsFactors = FALSE)

# Compute phi coefficient
compute_phi_coefficients <- function(data) {
  for (col in names(data)) {
    if (is.binary <- is.factor(data[[col]]) && length(levels(data[[col]])) == 2) {
      phi <- cor(as.numeric(data[[col]]), as.numeric(data$label))
      result <- rbind(result, data.frame(variable = col,
                                         label = "label",
                                         phi_coefficient = phi,
                                         stringsAsFactors = FALSE))
    }
  }

  # Plot correlation coefficients
  plot <- ggplot(na.omit(result), aes(x = reorder(variable, -phi_coefficient), y = phi_coefficient)) +
    geom_bar(stat = "identity", fill = "skyblue", color = "black") +
    coord_flip() +
    labs(title = "Phi Coefficint of Binary Variables with Label",
         x = "Variable",
         y = "Phi Coefficient") +
    theme_minimal()

  return(list(result = result, plot = plot))  # Return both result dataframe and plot
}

# Call the function
output <- compute_phi_coefficients(binary_df)

# Plot the bar chart separately
print(output$plot)
```




```
# result dataframe
result_df <- output$result
result_df
```

```
##         variable label phi_coefficient
## 1  url_has_login label      0.19200594
## 2 url_has_client label      0.06490411
## 3 url_has_server label      0.02874857
## 4  url_has_admin label      0.08442989
## 5     url_has_ip label      0.09053529
## 6  url_isshorted label      0.05603592
## 7          label label      1.00000000
```

We can also see the phi coefficient between all the binary variables and our target variable, label. Phi coefficients is a measure of association between two dichotomous variables.It measures how much two binary variables are associated where 1 is perfect association while 0 means no association at all.

The URL with log in has the strongest association with the target variable, followed by Url with IP, then URL with admin in that order. Since all of them had at least a weak association, we are going to use all the 6 binary variables in our next.

Data Preparation

We will then perform preparation like cleaning, and transforming the data to make it suitable for analysis and modeling. This may include handling missing values, encoding categorical variables, scaling features, and splitting the data into training and testing sets.

Cleaning

In here, we performed data preprocessing steps to prepare the training and test datasets for model training and evaluation. Firstly, we select columns without null values from the correlation_data dataframe and stroe the names of these columns in the num_cols vector. Then, we identify necessary columns for modeling, including both binary columns(binary_cols)and numerical columns(num_cols). Next, we set the seed for reproducibility using set.seed(2004) and downsapmle the train_data_cleaned dataset to 50,000 samples per label group to balace the dataset. We then selected only the necessary columns(necessary_cols) from both the training(train_data_cleaned) and test(test_data) datasets, ensuring consistency in the variables used for modeling between the training and test datasets.

```
# Select the columns without nulls
num_cols <- as.character(na.omit(correlation_data)$variable)

# Necessary columns
necessary_cols <- c(binary_cols, num_cols)

set.seed(2024)
train_data_cleaned <- train_data_cleaned %>% group_by(label) %>% sample_n(50000, replace = TRUE)

# Select only the necessary columns from train_data_clenaned
train <- train_data_cleaned[necessary_cols]

## Cleaning Test data
# Select only the necessary columns from train_data_cleaned
test <- test_data[necessary_cols]
```

Transforming to Right Data Types

For the training data, we first converted binary columns to factors. We converted the target variable label to a factor, with "0" encoded as "Benign" and "1" as "Malicious". We then shuffle the rows of the training data for randomness using sample and set the seed for reproducibility. Finally, we reset the row indices using row.names(). We did the same for testing set.

```
### Train Data ###
# Convert columns to factors
train[binary_cols] <- lapply(train[binary_cols], factor)

# Convert our target variable to factor
train <- train %>%
  mutate(label = as.factor(ifelse(label == '0', "Benign", "Malicious")))

# Shuffle the rows of train_data
set.seed(2024)
train <- train[sample(nrow(train)), ]

# Reset index using row.names()
row.names(train) <- NULL


### Test data ###
test[binary_cols] <- lapply(test[binary_cols], factor)

# Convert our target variable to factor
test <- test %>%
```

```
  mutate(label = as.factor(ifelse(label == '0', "Benign", "Malicious")))

# Shuffle the rows of train_data
set.seed(2024)
test <- test[sample(nrow(test)), ]

# Reset index using row.names()
row.names(test) <- NULL
```

Modeling

The data set was already split when we downloaded from Kaggle.com, but when selecting, we selected 200,000 observations for training before cleaning and pre-processing, and 100,000 for testing

We sampled a balanced training set.

Decision Tree

```
# decision tree classifier object
dt_classifier <- rpart(label ~ ., data = train, method = "class")

# Make predictions
dt_predictions <- predict(dt_classifier, newdata = test, type = "class")

# Calculate accuracy
accuracy <- sum(dt_predictions == test$label) / nrow(test)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.80613"
```

```
# Confusion matrix
confusion_matrix <- confusionMatrix(dt_predictions, test$label)

# Print Confusion matrix
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Benign Malicious
##   Benign     62674      3589
##   Malicious  15798     17939
##
##                Accuracy : 0.8061
##                  95% CI : (0.8037, 0.8086)
##     No Information Rate : 0.7847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5241
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7987
##             Specificity : 0.8333
```

```
##          Pos Pred Value : 0.9458
##          Neg Pred Value : 0.5317
##              Prevalence : 0.7847
##          Detection Rate : 0.6267
##    Detection Prevalence : 0.6626
##       Balanced Accuracy : 0.8160
##
##        'Positive' Class : Benign
##
```

```
# AUC using ROC curve
roc_dt <- roc(as.numeric(test$label), as.numeric(dt_predictions))
```

```
## Setting levels: control = 1, case = 2
```

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

```
# AUC for Decision Tree Classifier
auc_dt <- auc(roc_dt)
print(paste("AUC for Decision Tree Classifier:", round(auc_dt, 4)))
```

```
## [1] "AUC for Decision Tree Classifier: 0.816"
```

```
# Creating a dataframe for Decision Tree Classifier
dt_metrics_df <- data.frame(
  Model = "Decision Tree Classifier",
  Accuracy = round(as.numeric(confusion_matrix$overall['Accuracy']) * 100, 2),
  Sensitivity = round(as.numeric(confusion_matrix$byClass['Sensitivity'])  * 100, 2),
  Specificity = round(as.numeric(confusion_matrix$byClass['Specificity'])  * 100, 2),
  AUC = round(auc_dt  * 100, 2))
```

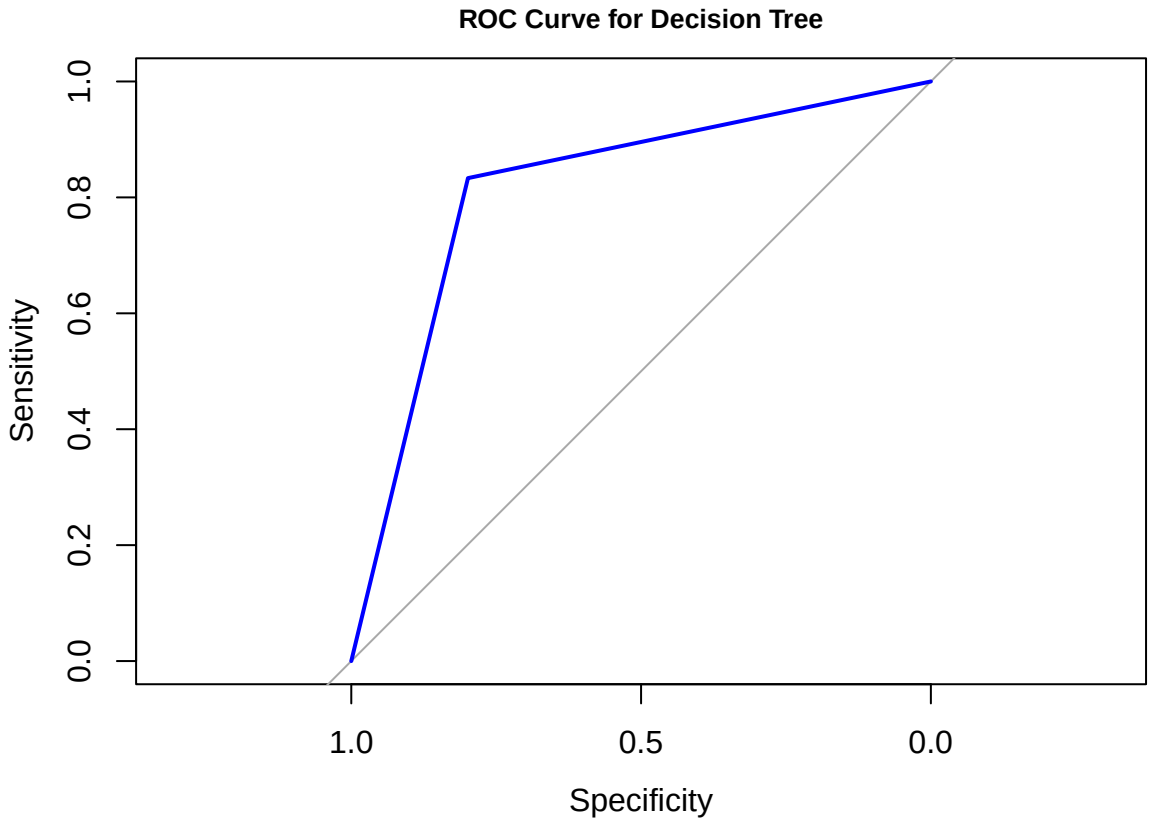
ROC Curve For Decision Tree

```
# Binary outcome for the ROC curve
binary_outcome_dt <- test$label

# Create an ROC object for Decision Tree
roc_dt <-roc(binary_outcome_dt, as.numeric(dt_predictions))
```

```
## Setting levels: control = Benign, case = Malicious
```

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

```
# Plot ROC curve for Decision Tree
plot(roc_dt, col = "blue",
     main = "ROC Curve for Decision Tree",
     lwd = 2,
     cex.main = 0.8)
```



The accuracy of the Decision Tree Classifier model is 0.861, indicating that it correctly predicts the class of the data points with an accuracy of 80.61%.

We can also see that the model correctly predicted62674 cases as Benign and 17939 cases as Malicious. It misclassiied 15798 Benign cases as Malicious and 3598 Malicious cases Malicious case as Benign.

The sensitivity of the model is 79.87, indicating the proportion of actual positives(Benign) correctly identified as positive(benign).

The specificity of the model is 83.33, indicating the proportion of actual negatives(Malicious) correctly identified as negative(Malicious).

The Area Under the Curve for the Decision Tree Classifier model is 0.816. This means that the model's ability to distinguish between positive and negative classes stands at 81.6%.

Logistic Regression

```
# Fit the Logistic Regression model
set.seed(2024)
log_reg <- glm(label ~ ., data = train, family = "binomial")

# Make Predictions
log_reg_predictions <- predict(log_reg, test)

# Convert predictions to class labels
log_reg_predictions <- as.factor(ifelse(log_reg_predictions > 0.5, 'Malicious', 'Benign'))

# Compute the confusion matrix
confusion_matrix_log_reg <- confusionMatrix(log_reg_predictions, test$label)
```

```r
# Print the confusion matrix
print(confusion_matrix_log_reg)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Benign Malicious
##   Benign     69895      6097
##   Malicious   8577     15431
##
##                Accuracy : 0.8533
##                  95% CI : (0.8511, 0.8554)
##     No Information Rate : 0.7847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5831
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8907
##             Specificity : 0.7168
##          Pos Pred Value : 0.9198
##          Neg Pred Value : 0.6427
##              Prevalence : 0.7847
##          Detection Rate : 0.6989
##    Detection Prevalence : 0.7599
##       Balanced Accuracy : 0.8037
##
##        'Positive' Class : Benign
##
```

```r
# Convert predictions to numeric
log_reg_predictions_numeric <- as.numeric(ifelse(log_reg_predictions == 'Malicious', 1, 0))
binary_label <- ifelse(test$label == 'Malicious', 1, 0)

# Calculate ROC curve
roc_log_reg <- roc(binary_label, log_reg_predictions_numeric)
```

```
## Setting levels: control = 0, case = 1
```

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

```r
# AUC for XGBoost
auc_log_reg <- auc(roc_log_reg)

print(paste("AUC for Logistic Regression:", round(auc_log_reg, 4)))
```

```
## [1] "AUC for Logistic Regression: 0.8037"
```

```r
# append Logistic Regression metrics to the existing dataframe
log_reg_metrics <- data.frame(
  Model = "Logistic Regression",
  Accuracy = round(as.numeric(confusion_matrix_log_reg$overall['Accuracy']) * 100, 2),
  Sensitivity = round(as.numeric(confusion_matrix_log_reg$byClass['Sensitivity']) * 100, 2),
  Specificity = round(as.numeric(confusion_matrix_log_reg$byClass['Specificity']) * 100, 2),
  AUC = round(auc_log_reg * 100, 2)
```

```
)
```

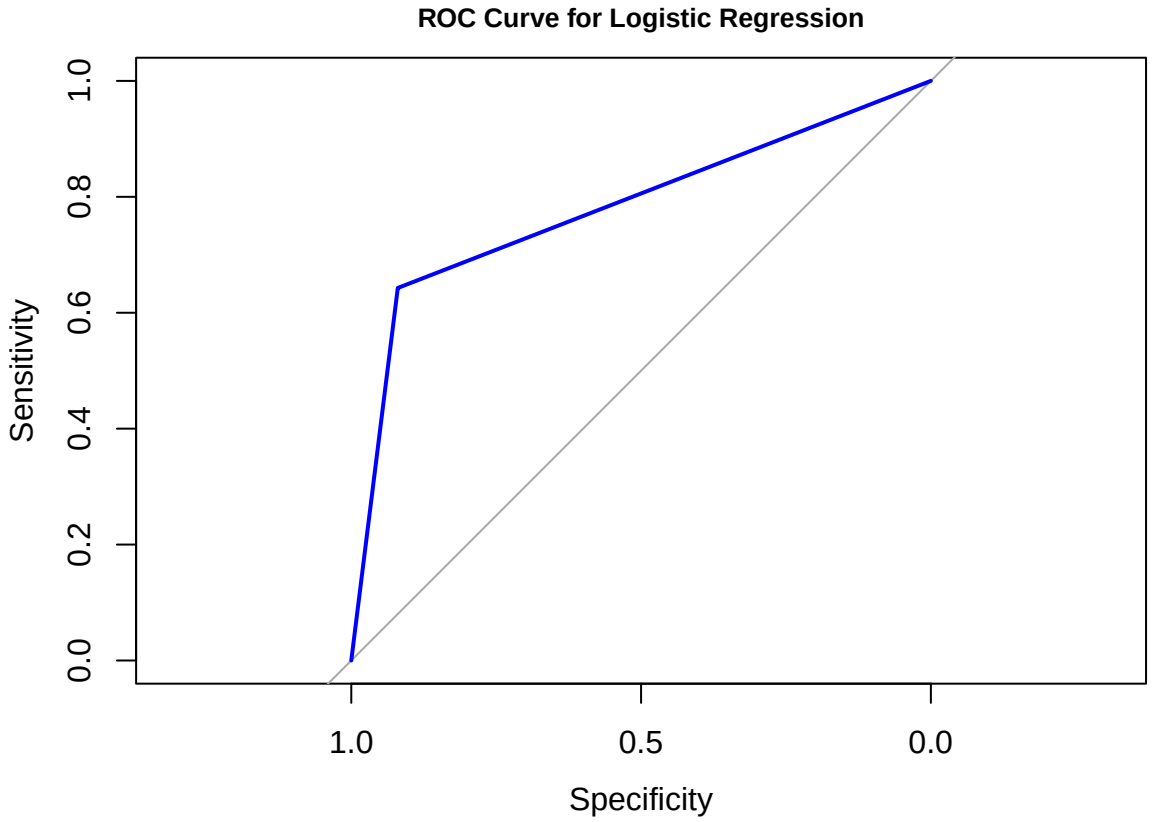
ROC For Logistic Regression

```
# Binary outcome for the ROC curve
binary_outcome_log_reg <- ifelse(test$label == "Alive", 1, 0)

# ROC Curve
roc_log_reg <- roc(log_reg_predictions_numeric, binary_label)
```

```
## Setting levels: control = 0, case = 1
```

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

```
# Plotting ROC Curve for KNN
plot(roc_log_reg, col ="blue",
     main = "ROC Curve for Logistic Regression",
     lwd = 2,
     cex.main = 0.8)
```



The accuracy of the model is 0.8533, which indicates that is correctly predicts the class of the data points with an accuracy of 85.33%.

From the confusion matrix, we can see that the model correctly predicted 69895 Benign cases and 15431 Malicious cases. However, it misclassified 8577 Benign cases as Malicious and 6097 Malicious cases as Benign.

The sensitivity of the model is 0.8907, which indicates the proportion (89.07%) of actual positives (Benign) correctly identified as positive.

The specificity of the model is 0.7168, which indicates the proportion (71.68%) of actual negatives (Malicious)

correctly identified as negative.

The Area Under the Curve is 0.8037, which means, the model's ability to distinguish between positive and negative classes scored 80.37%.

XGBoost Model

```
# XGBoost with iteration_range
# Convert the class labels to 0 and 1 for binary classification
train$label <- ifelse(train$label == "Malicious", 1, 0)
test$label <- ifelse(test$label == "Malicious", 1, 0)

# Convert entire train and test datasets to numeric
train <- as.data.frame(lapply(train, as.numeric))
test <- as.data.frame(lapply(test, as.numeric))

# Convert the training and test data to DMatrix format
dtrain <- xgb.DMatrix(data = as.matrix(train[, -which(names(train) == "label")]), label = train$label)
dtest <- xgb.DMatrix(data = as.matrix(test[, -which(names(test) == "label")]), label = test$label)

# Define XGBoost parameters
params <- list(
  # Binary classification problem
  objective = "binary:logistic",

  # Evaluation metric (logarithmic loss)
  eval_metric = "logloss",

  # Learning rate
  eta = 0.3,

  # Maximum depth of trees
  max_depth = 6,

  # Minimum sum of instance weight needed
  min_child_weight = 1,

  # Subsample ratio of the training data
  subsample = 1,

  # Subsample ratio of columns when constructing each tree
  colsample_bytree = 1
)

set.seed(2023)
# Train the XGBoost model
xgb_model <- xgboost(data = dtrain, nrounds = 100, verbose = 0, params = params)

# Make predictions on the test data
xgb_predictions <- predict(xgb_model, dtest)

# Convert predictions to class labels (0 or 1)
xgb_predictions <- as.factor(ifelse(xgb_predictions > 0.5, 'Malicious', 'Benign'))
```

```r
# Compute the confusion matrix
test$label <- as.factor(ifelse(test$label == 1, 'Malicious', 'Benign'))
train$label <- as.factor(ifelse(train$label == 1, 'Malicious', 'Benign'))
confusion_matrix <- confusionMatrix(xgb_predictions, test$label)

# Print the confusion matrix
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Benign Malicious
##   Benign     68598      3799
##   Malicious   9874     17729
##
##                Accuracy : 0.8633
##                  95% CI : (0.8611, 0.8654)
##     No Information Rate : 0.7847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6329
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8742
##             Specificity : 0.8235
##          Pos Pred Value : 0.9475
##          Neg Pred Value : 0.6423
##              Prevalence : 0.7847
##          Detection Rate : 0.6860
##    Detection Prevalence : 0.7240
##       Balanced Accuracy : 0.8489
##
##        'Positive' Class : Benign
##
```

```r
# Convert predictions to numeric
xgb_predictions_numeric <- as.numeric(ifelse(xgb_predictions == 'Malicious', 1, 0))
binary_label <- ifelse(test$label == 'Malicious', 1, 0)

# Calculate ROC curve
roc_xgb <- roc(binary_label, xgb_predictions_numeric)
```

```
## Setting levels: control = 0, case = 1
```

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

```r
# AUC for XGBoost
auc_xgb <- auc(roc_xgb)

print(paste("AUC for XGBoost:", round(auc_xgb, 4)))
```

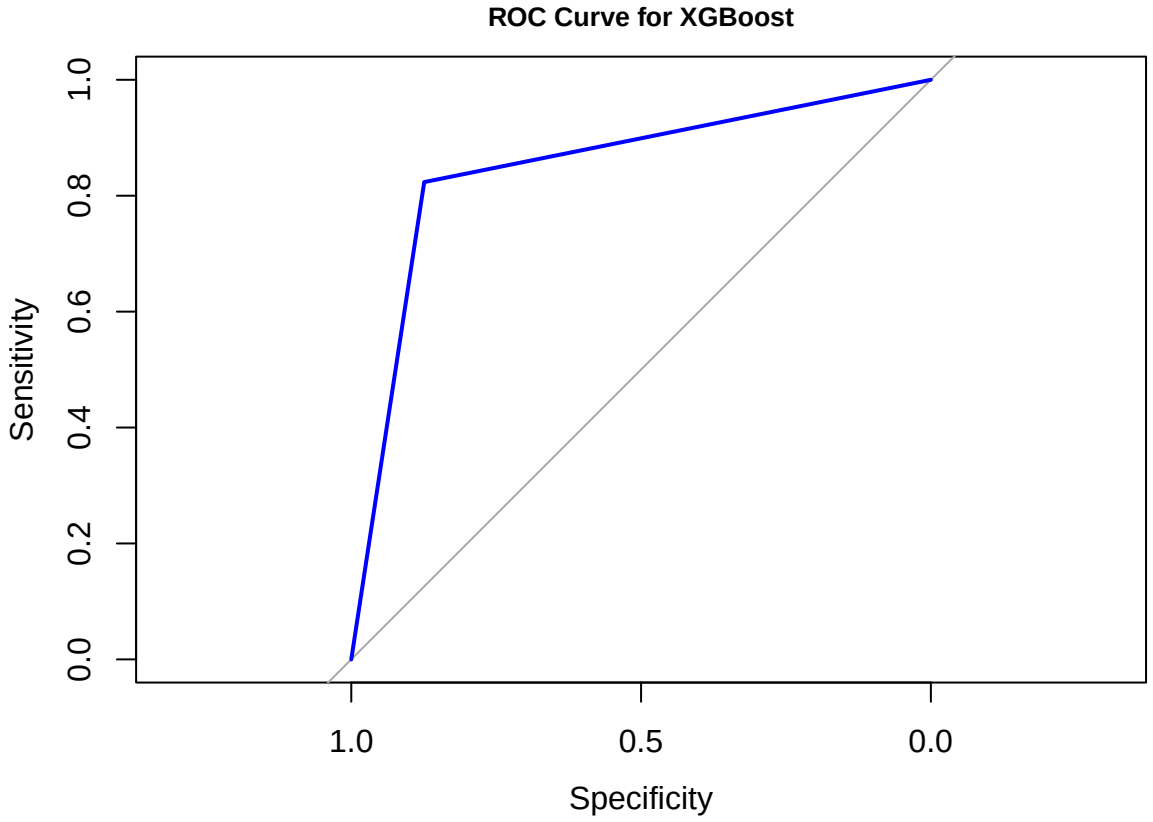
```
## [1] "AUC for XGBoost: 0.8489"
```

```r
# Creating a dataframe for XGBoost model
xgb_metrics_df <- data.frame(
```

```
  Model = "XGBoost",
  Accuracy = round(as.numeric(confusion_matrix$overall['Accuracy']) * 100, 2),
  Sensitivity = round(as.numeric(confusion_matrix$byClass['Sensitivity']) * 100, 2),
  Specificity = round(as.numeric(confusion_matrix$byClass['Specificity']) * 100, 2),
  AUC = round(auc_xgb * 100, 2)
)
```

ROC For XGBoost Model

```
# Binary outcome for the ROC curve
binary_outcome_XGBoost <- test$label

# Create an ROC object for XGBoost
roc_XGBoost <- roc(binary_outcome_XGBoost, as.numeric(xgb_predictions))

# Plot ROC curve for XGBoost
plot(roc_XGBoost, col = "blue",
     main = "ROC Curve for XGBoost",
     lwd = 2, cex.main = 0.8)
```



The accuracy of the model is 0.8633, which indicates that it correctly predicts the class of the data points with an accuracy of 86.33%.

From the confusion matrix, we can see that the model correctly predicted 68598 Benign cases and 17729 Malicious cases. However, it misclassified 9874 Benign cases as Malicious and 3799 Malicious cases as Benign.

The sensitivity of the model is 0.8742, which indicates the proportion (87.42%) of actual positives (Benign) correctly identified as positive.

The specificity of the model is 0.8235, which indicates the proportion (82.35%) of actual negatives (Malicious) correctly identified as negative.

The Area Under the Curve is 0.8489, which means, the model's ability to distinguish between positive and negative classes scored 84.89%.

Random Forest Model

```
# Random Forest
set.seed(2024)
train <- train %>% group_by(label) %>% sample_n(5000, replace = TRUE)
train <- train[sample(nrow(train)), ]
row.names(train) <- NULL

rf_model <- randomForest(label ~ ., data = train)

# Predicting
rf_pred <- predict(rf_model, newdata = test)

# Binary outcome for the ROC curve
binary_outcome <- ifelse(test$label == "Malicious", 1, 0)
rf_pred_outcome <- ifelse(rf_pred == "Malicious", 1, 0)

# Calculating ROC curve
roc_rf <- roc(binary_outcome, rf_pred_outcome)
```

```
## Setting levels: control = 0, case = 1
```

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

```
auc_rf <- auc(roc_rf)

# Accuracy and AUC for Random Forest
conf_matrix_rf <- confusionMatrix(rf_pred, test$label)
conf_matrix_rf
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Benign Malicious
##   Benign     67173      3787
##   Malicious  11299     17741
##
##                Accuracy : 0.8491
##                  95% CI : (0.8469, 0.8514)
##     No Information Rate : 0.7847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6037
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8560
##             Specificity : 0.8241
##          Pos Pred Value : 0.9466
##          Neg Pred Value : 0.6109
```

```
##              Prevalence : 0.7847
##          Detection Rate : 0.6717
##    Detection Prevalence : 0.7096
##       Balanced Accuracy : 0.8401
##
##        'Positive' Class : Benign
##
```

```r
# Getting the accuracy
accuracy_rf <- conf_matrix_rf$overall["Accuracy"]
print(paste("Accuracy for Random Forest:", round(accuracy_rf, 4)))
```

```
## [1] "Accuracy for Random Forest: 0.8491"
```

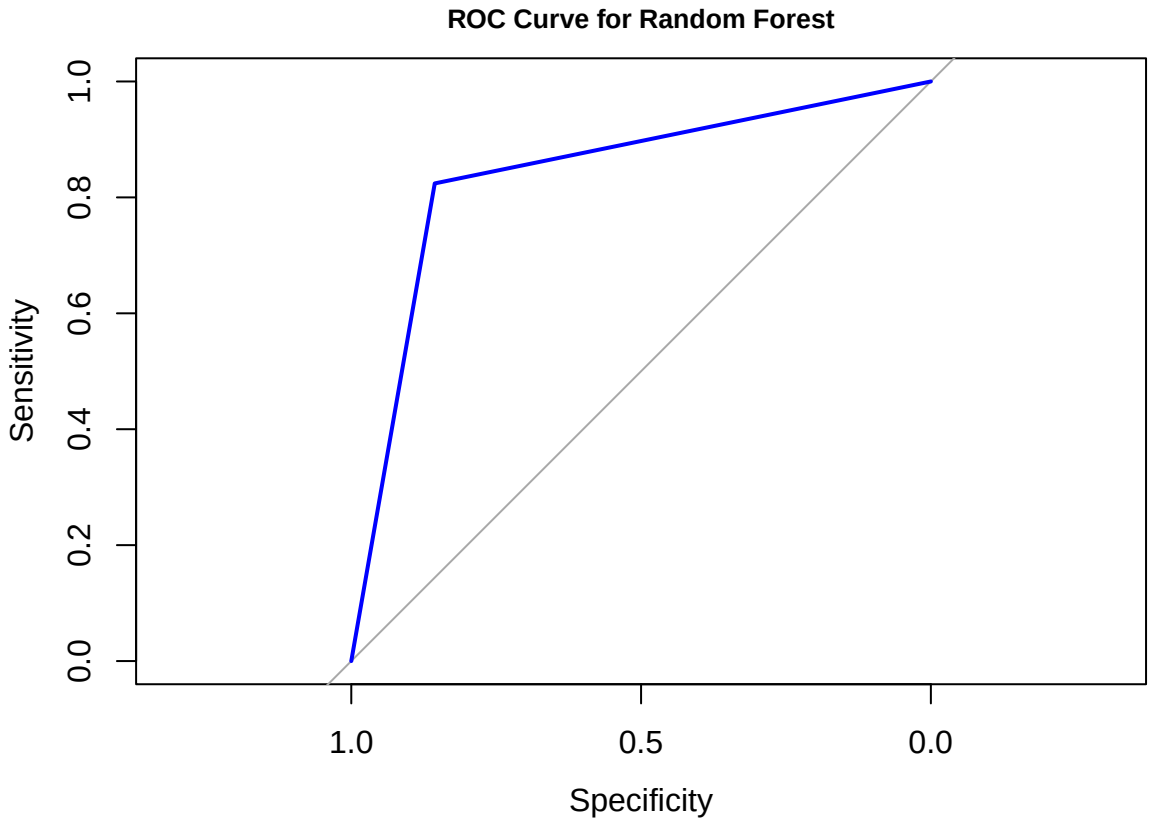
```r
print(paste("ACU for Random Forest:", round(auc_rf, 4)))
```

```
## [1] "ACU for Random Forest: 0.8401"
```

```r
# Creating a dataframe for Random Forest model
rf_metrics_df <- data.frame(
  Model = "Random Forest",
  Accuracy = round(as.numeric(conf_matrix_rf$overall['Accuracy']) * 100, 2),
  Sensitivity = round(as.numeric(conf_matrix_rf$byClass['Sensitivity']) * 100, 2),
  Specificity = round(as.numeric(conf_matrix_rf$byClass['Specificity']) * 100, 2),
  AUC = round(auc_rf * 100, 2)
)
```

ROC For Random Forest Model

```r
# Plot ROC curve for Random Forest
plot(roc_rf, col = "blue",
     main = "ROC Curve for Random Forest",
     lwd = 2,
     cex.main = 0.8)
```



For the Random Forest model, We got an accuracy of 0.8491, which means the algorithm correctly classified 84.91% of all instances into their respective classes. The sensitivity is 0.8560, implying that the model correctly identifies 85.60% of the actual positive cases (Benign) as positive. The model has a specificity of 0.8241, meaning that it identifies 82.41% of the actual negative cases (Malicious) as negative. The positive predictive value of 0.9466 means that among the instances predicted as positive by the model, 94.66% are truly positive. The negative predictive value of 0.6109 indicates that among the instances predicted as negative by the model, 61.09% are truly negative. Finally, the model managed to get an area under the curve score of 0.8401, which means the ability of the model to differentiate between the Benign and Malicious cases stood as 84.01%.

Support Vector Machine (SVM)

```
set.seed(2023)

# Fit SVM model
svm_model <- svm(label ~ ., data = train, kernel = "radial")

# Making predictions
svm_pred <- predict(svm_model, newdata = test)

# Confusion matrix
conf_matrix_svm <- confusionMatrix(svm_pred, test$label)

# Print metrict for SVM
conf_matrix_svm
```

```
## Confusion Matrix and Statistics
##
```

```
##            Reference
## Prediction  Benign Malicious
##   Benign     66335      3899
##   Malicious  12137     17629
##
##                Accuracy : 0.8396
##                  95% CI : (0.8374, 0.8419)
##     No Information Rate : 0.7847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5832
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8453
##             Specificity : 0.8189
##          Pos Pred Value : 0.9445
##          Neg Pred Value : 0.5923
##              Prevalence : 0.7847
##          Detection Rate : 0.6633
##    Detection Prevalence : 0.7023
##       Balanced Accuracy : 0.8321
##
##        'Positive' Class : Benign
##
```

```
# Birary outcome for the ROC curve
binary_outcome_svm <- ifelse(test$label == "Malicious", 1, 0)
binary_svm_pred <- ifelse(svm_pred == "Malicious", 1, 0)

# Calculating ROC curve
roc_svm <- roc (binary_outcome_svm, binary_svm_pred)
```

```
## Setting levels: control = 0, case = 1
```

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

```
auc_svm <- auc(roc_svm)
print(paste("AUC for SVM:", round(auc_svm, 4)))
```

```
## [1] "AUC for SVM: 0.8321"
```

```
# Creating a dataframe for SVM model
svm_metrics_df <- data.frame(
  Model = "SVM",
  Accuracy = round(as.numeric(conf_matrix_svm$overall['Accuracy']) * 100, 2),
  Sensitivity = round(as.numeric(conf_matrix_svm$byClass['Sensitivity']) * 100, 2),
  Specificity = round(as.numeric(conf_matrix_svm$byClass['Specificity']) * 100, 2),
  AUC = round(auc_svm * 100, 2)
)
```

ROC For SVM

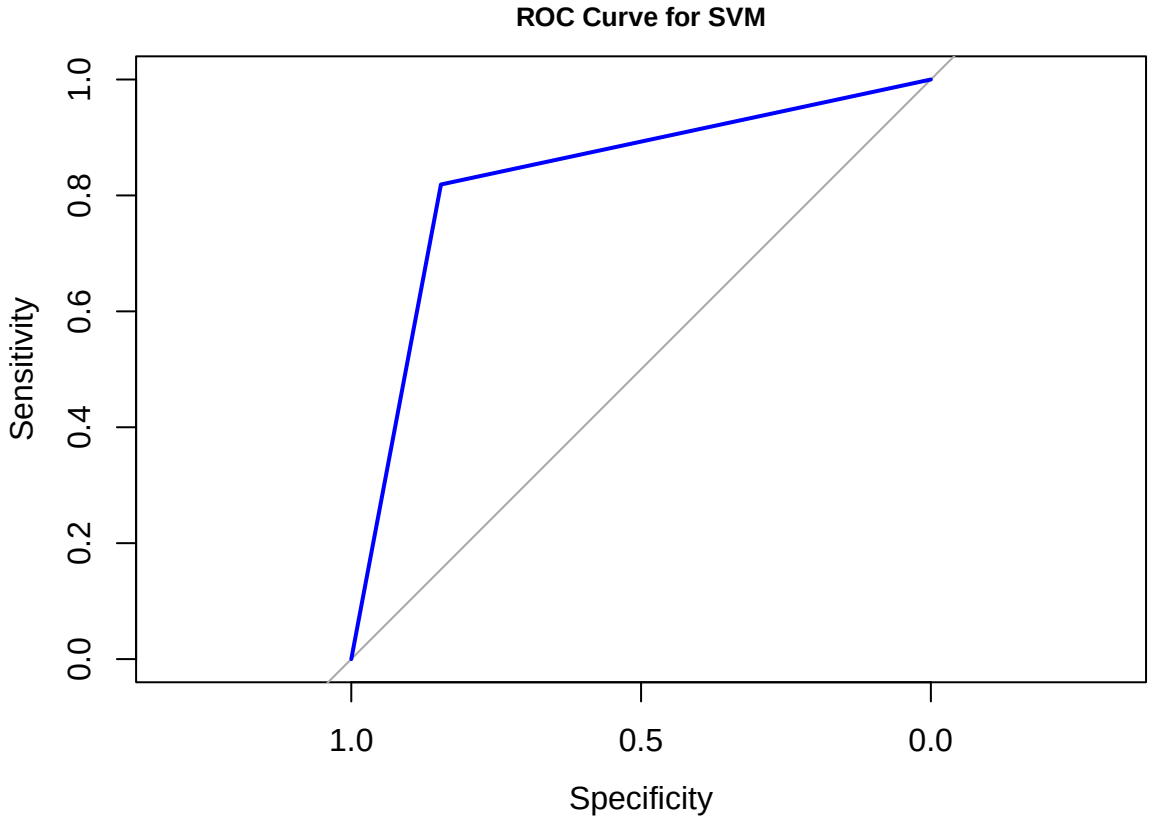
```
# Binary outcome for the ROC curve
binary_outcome_svm <- ifelse(test$label == "Malicious", 1, 0)
```

```r
# Create an ROC object for SVM
roc_svm <- roc(binary_outcome_svm, binary_svm_pred)
```

```
## Setting levels: control = 0, case = 1
```

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

```r
# Plot ROC curve for SVM
plot(roc_svm, col = "blue",
     main = "ROC Curve for SVM",
     lwd = 2,
     cex.main = 0.8)
```



The accuracy of the SVM model is 0.8396, indicating that it correctly predicts the class of the data points with an accuracy of 83.96% From the confusion matrix, we can see that the model correctly predicted 66335 Benign cases and 17629 Malicious cases, but it misclassified 12137 Benign cases as Malicious, and 3899 Malicious cases as Benign. The sensitivity of the model is 84.53%, indicating the proportion of the actual positives correctly identified as positive. The specificity of the model is 81.89%, indicating the proportion of the actual negatives correctly identified as negative. The area under the curve for the SVM model is 0.8321, meaning, the model's ability to in distinguishing between alive and dead classes is placed as about 83.21%.

Evaluation

```r
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
```

Model	Accuracy	Sensitivity	Specificity	AUC
Decision Tree Classifier	80.61	79.87	83.33	81.60
Logistic Regression	85.33	89.07	71.68	80.37
XGBoost	86.33	87.42	82.35	84.89
Random Forest	84.91	85.60	82.41	84.01
SVM	83.96	84.53	81.89	83.21

```
##
##     group_rows
```

```
# Combine all the dataframes
combined_metrics_df <- rbind(dt_metrics_df, log_reg_metrics, xgb_metrics_df, rf_metrics_df, svm_metrics

# Print the combined dataframe
# Combined_metrics_df

kable(combined_metrics_df) %>%
  kable_styling(latex_options = 'striped')
```

Considering all the models' performance as seen in the analysis, it seems like XGBoost has the highest accuracy (86.33%), relatively high sensitivity (87.42%), specificity (82.35%), and ACU (84.89%), making it the best choice for striking a balance between these metrics.

Deployment

In here, we have developed an interactive program for URL label prediction based on user-inputted features. Firstly, a function named get_user_input is defined to prompt the user to input values for each feature associated with the URL. These features are stored in the input_features vector. Subsequently, an empty dataframe named user_input_data is created to collect the user-provided values for each feature. Through a loop iterating over the input_features. user input are obtained and stored in the corresponding columns of user_input_data. Following this, a function called predict_label is established to predict the label, either "Malicious" or "Benign", based on the user-inputted features using a pre-trained model(xgb_model). The program them predicts the label for the provided user input and prints an appropriate message based on the predicted label, cautioning the user if a potential malicious URL is detected or signaling that the URL is safe to proceed. This program facilitates quick assessment and classification of URLs based on user-provided features, offering a simple yet effective means of URL label prediction.

```
# Function to get user input
get_user_input <- function(feature_name) {
  cat(paste("Enter value for", feature_name, ": "))
  as.numeric(readline(prompt = ""))
}

# Ask user to input values for each feature and save as dataframe
input_features <- c("url_has_login", "url_has_client", "url_has_server",
                    "url_has_admin", "url_has_ip", "url_isshorted",
                    "url_len", "url_entropy", "url_count_dot",
                    "url_count_perc", "url_count_hyphen", "url_count_underscore",
                    "url_count_equal", "url_count_amp", "url_count_letter",
                    "url_count_digit", "url_count_sensitive_words", "path_len",
                    "subdomain_len", "subdomain_count_dot")

user_input_data <- data.frame(matrix(nrow = 1, ncol = length(input_features)))
colnames(user_input_data) <- input_features
```

```r
for (i in seq_along(input_features)) {
  user_input_data[, i] <- get_user_input(input_features[i])
}
```

```
## Enter value for url_has_login :
## Enter value for url_has_client :
## Enter value for url_has_server :
## Enter value for url_has_admin :
## Enter value for url_has_ip :
## Enter value for url_isshorted :
## Enter value for url_len :
## Enter value for url_entropy :
## Enter value for url_count_dot :
## Enter value for url_count_perc :
## Enter value for url_count_hyphen :
## Enter value for url_count_underscore :
## Enter value for url_count_equal :
## Enter value for url_count_amp :
## Enter value for url_count_letter :
## Enter value for url_count_digit :
## Enter value for url_count_sensitive_words :
## Enter value for path_len :
## Enter value for subdomain_len :
## Enter value for subdomain_count_dot :
```

```r
# Function to predict label
predict_label <- function(model, input_data) {
  pred <- predict(model, as.matrix(input_data))
  ifelse(pred > 0.5, "Malicious", "Benign")
}

# Predict label using the trained model and user inputs
predicted_label <- predict_label(xgb_model, user_input_data)


# Print appropriate message based on predicted label
if (any(predicted_label == "Malicious")) {
  cat("\nPotential Malicious URL Detected! Take Caution.\n")
} else {
  cat("\nThe URL is safe. Proceed.\n")
}
```

```
##
## The URL is safe. Proceed.
```

```r
# Print appropriate message based on predicted label
if (any(predicted_label == "Malicious")) {
  cat("\nPotential Malicious URL Detected! Take Caution.\n")
} else {
  cat("\nProceed.\n")
}
```

```
##
## Proceed.
```