Cyber ML Capstone Project - Justin Oh

Justin Oh

11/22/2023

Introduction

Our ever-increasing global connectivity and heavy reliance on cloud services has forced humanity to become more technologically dependent than ever before. Approximately 5.25 billion people have access to and use the internet on a daily basis. In fact, from the year 2000 to 2022, the usage of the internet has increased by 1,355%.

Such dependence highlights just how critical it has become for individuals, companies, and nations to boost their cyber security. Technology continues to advance at a rate nearly impossible to keep up with, but we must not trail behind in our attempts to understand these developments and apply the lessons learned. The cost of leniency is steep as cyber criminals are finding resounding success in their hacking attempts. To put in perspective, the average cost of a data breach was $4.24 million in 2021, with the average time to identify a breach being 212 days.

While governments across the globe are allocating more resources to adequately combat cyber crimes, the vast majority of the world do not consider cyber security to be a significant issue, let alone a national security threat. In order to emphasize the importance for countries to increase investment in their cyber security programs, I have decided to pursue a personal project regarding the classification of benign and malicious URLs.

Goal of the Project

By utilizing Logistic Regression, K-nearest neighbors (KNN) and Random Forest algorithms, the objective of this project is to identify which model most accurately classifies the URLs as malicious or benign. Specificity is the metric we'll try to improve as we work through the models.

Preparing the Dataset

First, we will load all the necessary libraries and the dataset.

```r
# Required packages will install, please allow several minutes to complete.

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(purrr)) install.packages("purrr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(Hmisc)) install.packages("Hmisc", repos = "http://cran.us.r-project.org")
if(!require(forecast)) install.packages("forecast", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(class)) install.packages("class", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(psych)) install.packages("psych", repos = "http://cran.us.r-project.org")
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(NCmisc)) install.packages("NCmisc", repos = "http://cran.us.r-project.org")

# Packages for graphing and modeling
library(tidyverse)
library(purrr)
library(caret)
library(Hmisc)
library(forecast)
library(randomForest)
library(class)
library(data.table)
# Packages for reading the CSV file:
library(psych)
library(readr)
library(dplyr)
#Packages for testing what packages are used:
library(NCmisc)
```

Now, we will download the dataset from my GitHub repository and load it into RStudio:

```r
# The necessary data-set is able through my Github repository if the following code fails
download.file("https://raw.githubusercontent.com/justin-2028/Justin-2028-HarvardX-Capstone-Personal-Pro
train = read.csv("kaggleRCdataset.csv", header = TRUE)
```

We'll move on to first stage of the analysis process, which is data cleaning.

Data Description

URL: It is the anonymous identification of the URL analyzed in the study.

URL_LENGTH: It is the number of characters in the URL.

NUMBERSPECIALCHARACTERS: It is the number of special characters identified in the URL, such as, "/", "%", "#", "&", ". ", "=".

CHARSET: It is a categorical value and represents the character encoding standard (also called character set).

SERVER: It is a categorical value and represents the operative system of the server that has been derived from the packet response.

CONTENT_LENGTH: It represents the content size of the HTTP header.

WHOIS_COUNTRY: It is a categorical variable, and its values are the countries we got from the server response (specifically, through the API of WHOIS).

WHOIS_STATEPRO: It is a categorical variable, and its values are the states we got from the server response (specifically, through the API of WHOIS).

WHOIS_REGDATE: WHOIS provides the server registration date, so, this variable has date values with the format DD/MM/YYY HH:MM.

WHOISUPDATEDDATE: Through the WHOIS, the last update date from the server is represented through this variable.

TCPCONVERSATIONEXCHANGE: This variable is the number of TCP packets exchanged between the server and the honeypot client.

DISTREMOTETCP_PORT: It is the number of the ports detected and unique to TCP.

REMOTE_IPS: This variable represents the total number of IPs connected to the honeypot.

APP_BYTES: This is the number of bytes transferred.

SOURCEAPPPACKETS: This is the number of packets sent from the honeypot to the server.

REMOTEAPPPACKETS: This is the number of packets received from the server.

APP_PACKETS: This is the total number of IP packets generated during the communication between the honeypot and the server.

DNSQUERYTIMES: This is the number of DNS packets generated during the communication between the honeypot and the server.

TYPE: This is a categorical variable, and its values represent the type of web page analyzed, specifically, 1 is for malicious websites and 0 is for benign websites.

Data Cleaning:

Data cleaning is a vital stage of the analysis. It helps us understands the data and clean it in order to make it fit for our modelling purposes. Our data also has some issues that must be addressed beforehand. Thus, we will clean it to prepare it for modelling. There are some issues with the "WHOIS_COUNTRY" variable which we need to examine:

```
unique(train$WHOIS_COUNTRY)
```

```
##  [1] "None"           "US"             "SC"             "GB"
##  [5] "UK"             "RU"             "AU"             "CA"
##  [9] "PA"             "se"             "IN"             "LU"
## [13] "TH"             "[u'GB'; u'UK']" "FR"             "NL"
## [17] "UG"             "JP"             "CN"             "SE"
## [21] "SI"             "IL"             "ru"             "KY"
## [25] "AT"             "CZ"             "PH"             "BE"
## [29] "NO"             "TR"             "LV"             "DE"
## [33] "ES"             "BR"             "us"             "KR"
## [37] "HK"             "UA"             "CH"             "United Kingdom"
## [41] "BS"             "PK"             "IT"             "Cyprus"
## [45] "BY"             "AE"             "IE"             "UY"
## [49] "KG"
```

We can see that WHOIS_COUNTRY has different values for one country and that has to be corrected. For example: UK is shown as United Kingdom and GB. We need to correct that before moving forward.

```
train$WHOIS_COUNTRY <- as.character(train$WHOIS_COUNTRY)
train[train$WHOIS_COUNTRY == 'United Kingdom','WHOIS_COUNTRY'] <- "UK"
train[train$WHOIS_COUNTRY == "[u'GB'; u'UK']",'WHOIS_COUNTRY'] <- "UK"
train[train$WHOIS_COUNTRY == "GB",'WHOIS_COUNTRY'] <- "UK"
train[train$WHOIS_COUNTRY == "us",'WHOIS_COUNTRY'] <- "US"
train[train$WHOIS_COUNTRY == 'ru','WHOIS_COUNTRY'] <- "RU"
```

Most countries do not seem to have malicious data. This disparity in data can lead to over-fitting in the modelling. We'll have to deal with this issue here by creating a single category for such countries called "Other".

```
mc <- train[train$Type == 1,'WHOIS_COUNTRY']
others <- which(!(train$WHOIS_COUNTRY %in% mc))
train[others,'WHOIS_COUNTRY'] <- "Other"
train$WHOIS_COUNTRY <- as.factor(train$WHOIS_COUNTRY)
```

Now let's look at the values in CHARSET variable:

```
unique(train$CHARSET)
```

```
## [1] "iso-8859-1"   "UTF-8"        "us-ascii"     "ISO-8859-1"   "utf-8"
## [6] "None"         "windows-1251" "ISO-8859"     "windows-1252"
```

There is a similar problem in this variable as we saw in the WHOIS_COUNTRY variable:

```
train$CHARSET <- as.character(train$CHARSET)
train[train$CHARSET == 'iso-8859-1',"CHARSET"] <- "ISO-8859-1"
train[train$CHARSET == 'utf-8',"CHARSET"] <- "UTF-8"
train[train$CHARSET == 'windows-1251',"CHARSET"] <- "windows-12##"
train[train$CHARSET == 'windows-1252',"CHARSET"] <- "windows-12##"
train$CHARSET <- as.factor(train$CHARSET)
```

For the normalization of the SERVER variable, we will assign the values which do not have any malicious value in the data to the "Other" server value.

```
train$SERVER <- as.character(train$SERVER)
mserver <- train[train$Type == 1,"SERVER"]
others <- which(!(train$SERVER %in% mserver))
train[others,'SERVER'] <- "Other"
table(train$SERVER == "Other")
```

```
##
## FALSE  TRUE
##  1031   750
```

```
train$SERVER <- as.factor(train$SERVER)
```

Having missing values in the data is a problem. There are many ways to deal with the missing values. The first method is to remove the missing values, but if the number of rows having missing values is high, it can result in data loss. Another method to deal with missing values is using imputations. This method helps when rows for missing values is high enough to not be removed, so you change the missing values to the mean of the variable. If the number of rows with missing values is more than 60% or 70% of total number of rows, then imputing the mean or median won't be helpful and it's better to remove that column from the analysis. We'll look at the number of NAs in our data:

```
colSums(is.na(train))
```

```
##                       URL                 URL_LENGTH NUMBER_SPECIAL_CHARACTERS
##                         0                          0                         0
##                   CHARSET                     SERVER            CONTENT_LENGTH
##                         0                          0                       812
##             WHOIS_COUNTRY             WHOIS_STATEPRO             WHOIS_REGDATE
##                         0                          0                         0
##         WHOIS_UPDATED_DATE TCP_CONVERSATION_EXCHANGE      DIST_REMOTE_TCP_PORT
##                         0                          0                         0
##                REMOTE_IPS                  APP_BYTES        SOURCE_APP_PACKETS
##                         0                          0                         0
##        REMOTE_APP_PACKETS           SOURCE_APP_BYTES          REMOTE_APP_BYTES
##                         0                          0                         0
##               APP_PACKETS            DNS_QUERY_TIMES                      Type
##                         0                          1                         0
```

We can see that there are 812 NAs in the content length variable. This is almost half of the total number of rows. We can remove this column as imputation won't be helpful in the modelling process and information from this variable isn't significant. There is 1 NA in DNS_QUERY_TIMES which can be solved by imputation:

```
train$DNS_QUERY_TIMES=impute(train$DNS_QUERY_TIMES, 0)
train$CONTENT_LENGTH=impute(train$CONTENT_LENGTH, mean)
```

We'll now remove the variables which will not be useful in the modelling process:

```
train$URL <- NULL
train$WHOIS_REGDATE <- NULL

train$CONTENT_LENGTH <- NULL
```

We can change our response variable to the factor variable:

```
train$type<- as.factor(train$Type)
```

Data Exploration:

The main purpose of exploring the data here is to get a better understanding of the dataset. First, we'll look at how many rows and columns we have in the dataset:

```
dim(train)
```

```
## [1] 1781   19
```

We have 1781 rows and 21 variables. Let's look at the first few and last few rows of the data set to see how the table looks like:

```
head(train, 10)
```

```
##    URL_LENGTH NUMBER_SPECIAL_CHARACTERS    CHARSET                SERVER
## 1          16                         7 ISO-8859-1                 nginx
## 2          16                         6      UTF-8                 Other
## 3          16                         6   us-ascii Microsoft-HTTPAPI/2.0
## 4          17                         6 ISO-8859-1                 nginx
## 5          17                         6      UTF-8                 Other
## 6          18                         7      UTF-8                 nginx
## 7          18                         6 ISO-8859-1              Apache/2
## 8          19                         6   us-ascii Microsoft-HTTPAPI/2.0
## 9          20                         5      UTF-8          nginx/1.10.1
## 10         20                         5      UTF-8          nginx/1.10.1
##    WHOIS_COUNTRY WHOIS_STATEPRO WHOIS_UPDATED_DATE TCP_CONVERSATION_EXCHANGE
## 1           None           None               None                         7
## 2           None           None               None                        17
## 3           None           None               None                         0
## 4             US             AK    12/09/2013 0:45                        31
## 5             US             TX    11/04/2017 0:00                        57
## 6          Other           Mahe     3/10/2016 3:45                        11
## 7             US             CO     1/07/2016 0:00                        12
## 8             US             FL    19/03/2017 0:00                         0
## 9           None           None               None                         0
## 10          None           None               None                         0
##    DIST_REMOTE_TCP_PORT REMOTE_IPS APP_BYTES SOURCE_APP_PACKETS
## 1                     0          2       700                  9
## 2                     7          4      1230                 17
## 3                     0          0         0                  0
## 4                    22          3      3812                 39
## 5                     2          5      4278                 61
## 6                     6          9       894                 11
## 7                     0          3      1189                 14
## 8                     0          0         0                  0
## 9                     0          0         0                  2
## 10                    0          0         0                  2
##    REMOTE_APP_PACKETS SOURCE_APP_BYTES REMOTE_APP_BYTES APP_PACKETS
## 1                  10             1153              832           9
## 2                  19             1265             1230          17
## 3                   0                0                0           0
## 4                  37            18784             4380          39
## 5                  62           129889             4586          61
## 6                  13              838              894          11
## 7                  13             8559             1327          14
## 8                   0                0                0           0
## 9                   3              213              146           2
## 10                  1               62              146           2
##    DNS_QUERY_TIMES Type type
## 1                2    1    1
## 2                0    0    0
## 3                0    0    0
## 4                8    0    0
## 5                4    0    0
## 6                0    0    0
## 7                2    0    0
## 8                0    0    0
## 9                2    1    1
## 10              2    1    1
```

```
tail(train, 10)
```

```
##      URL_LENGTH NUMBER_SPECIAL_CHARACTERS    CHARSET           SERVER
## 1772        170                        17      UTF-8           Apache
## 1773        173                        34      UTF-8           Apache
## 1774        178                        16      UTF-8           Apache
## 1775        183                        29 ISO-8859-1            Other
## 1776        194                        17      UTF-8           Apache
## 1777        194                        16      UTF-8           Apache
## 1778        198                        17      UTF-8           Apache
## 1779        201                        34      UTF-8            Other
## 1780        234                        34 ISO-8859-1 cloudflare-nginx
## 1781        249                        40      UTF-8            Other
##      WHOIS_COUNTRY WHOIS_STATEPRO WHOIS_UPDATED_DATE TCP_CONVERSATION_EXCHANGE
## 1772            ES      Barcelona     2/09/2016 0:00                         0
## 1773            ES      Barcelona     2/09/2016 0:00                         1
## 1774            ES      Barcelona     2/09/2016 0:00                         0
## 1775            US             NY    18/11/2014 0:00                        22
## 1776            ES      Barcelona     2/09/2016 0:00                         0
## 1777            ES      Barcelona     2/09/2016 0:00                         0
## 1778            ES      Barcelona     2/09/2016 0:00                         0
## 1779            US             FL    15/07/2015 0:00                        83
## 1780            US             CA     9/12/2016 0:00                         0
## 1781            US      Wisconsin    20/11/2013 0:00                        19
##      DIST_REMOTE_TCP_PORT REMOTE_IPS APP_BYTES SOURCE_APP_PACKETS
## 1772                    0          0         0                  0
## 1773                    1          1        90                  1
## 1774                    0          0         0                  0
## 1775                    2          7      2062                 30
## 1776                    0          0         0                  0
## 1777                    0          0         0                  0
## 1778                    0          0         0                  0
## 1779                    2          6      6631                 87
## 1780                    0          0         0                  0
## 1781                    6         11      2314                 25
##      REMOTE_APP_PACKETS SOURCE_APP_BYTES REMOTE_APP_BYTES APP_PACKETS
## 1772                  2              124                0           0
## 1773                  5              416               90           1
## 1774                  3              186                0           0
## 1775                 26             8161             2742          30
## 1776                  3              186                0           0
## 1777                  3              186                0           0
## 1778                  2              124                0           0
## 1779                 89           132181             6945          87
## 1780                  0                0                0           0
## 1781                 28             3039             2776          25
##      DNS_QUERY_TIMES Type type
## 1772               0    1    1
## 1773               0    1    1
## 1774               0    1    1
## 1775               8    0    0
## 1776               0    1    1
## 1777               0    1    1
## 1778               0    1    1
## 1779               4    0    0
## 1780               0    0    0
## 1781               6    0    0
```

Looking at the structure of the dataset is very important as it enables further understanding of the variables. It's important to make sure the variables have the correct classes.

```
str(train)
```

```
## 'data.frame':    1781 obs. of  19 variables:
##  $ URL_LENGTH               : int  16 16 16 17 17 18 18 19 20 20 ...
##  $ NUMBER_SPECIAL_CHARACTERS: int  7 6 6 6 6 7 6 6 5 5 ...
##  $ CHARSET                  : Factor w/ 6 levels "ISO-8859","ISO-8859-1",..: 2 5 4 2 5 5 2 4 5 5 ...
##  $ SERVER                   : Factor w/ 29 levels "Apache","Apache/1.3.27 (Unix) PHP/4.4.1",..: 19 2
##  $ WHOIS_COUNTRY            : Factor w/ 19 levels "BR","CA","CN",..: 10 10 10 18 18 11 18 18 10 10 .
##  $ WHOIS_STATEPRO           : chr  "None" "None" "None" "AK" ...
##  $ WHOIS_UPDATED_DATE       : chr  "None" "None" "None" "12/09/2013 0:45" ...
##  $ TCP_CONVERSATION_EXCHANGE: int  7 17 0 31 57 11 12 0 0 0 ...
##  $ DIST_REMOTE_TCP_PORT     : int  0 7 0 22 2 6 0 0 0 0 ...
##  $ REMOTE_IPS               : int  2 4 0 3 5 9 3 0 0 0 ...
##  $ APP_BYTES                : int  700 1230 0 3812 4278 894 1189 0 0 0 ...
##  $ SOURCE_APP_PACKETS       : int  9 17 0 39 61 11 14 0 2 2 ...
##  $ REMOTE_APP_PACKETS       : int  10 19 0 37 62 13 13 0 3 1 ...
##  $ SOURCE_APP_BYTES         : int  1153 1265 0 18784 129889 838 8559 0 213 62 ...
##  $ REMOTE_APP_BYTES         : int  832 1230 0 4380 4586 894 1327 0 146 146 ...
##  $ APP_PACKETS              : int  9 17 0 39 61 11 14 0 2 2 ...
##  $ DNS_QUERY_TIMES          : 'impute' num  2 0 0 8 4 0 2 0 2 2 ...
##   ..- attr(*, "imputed")= int 1660
##  $ Type                     : int  1 0 0 0 0 0 0 0 1 1 ...
##  $ type                     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 2 2 ...
```

Now, let's look at the overall summary of the dataset to see the mean, median and variance for the numeric variables.

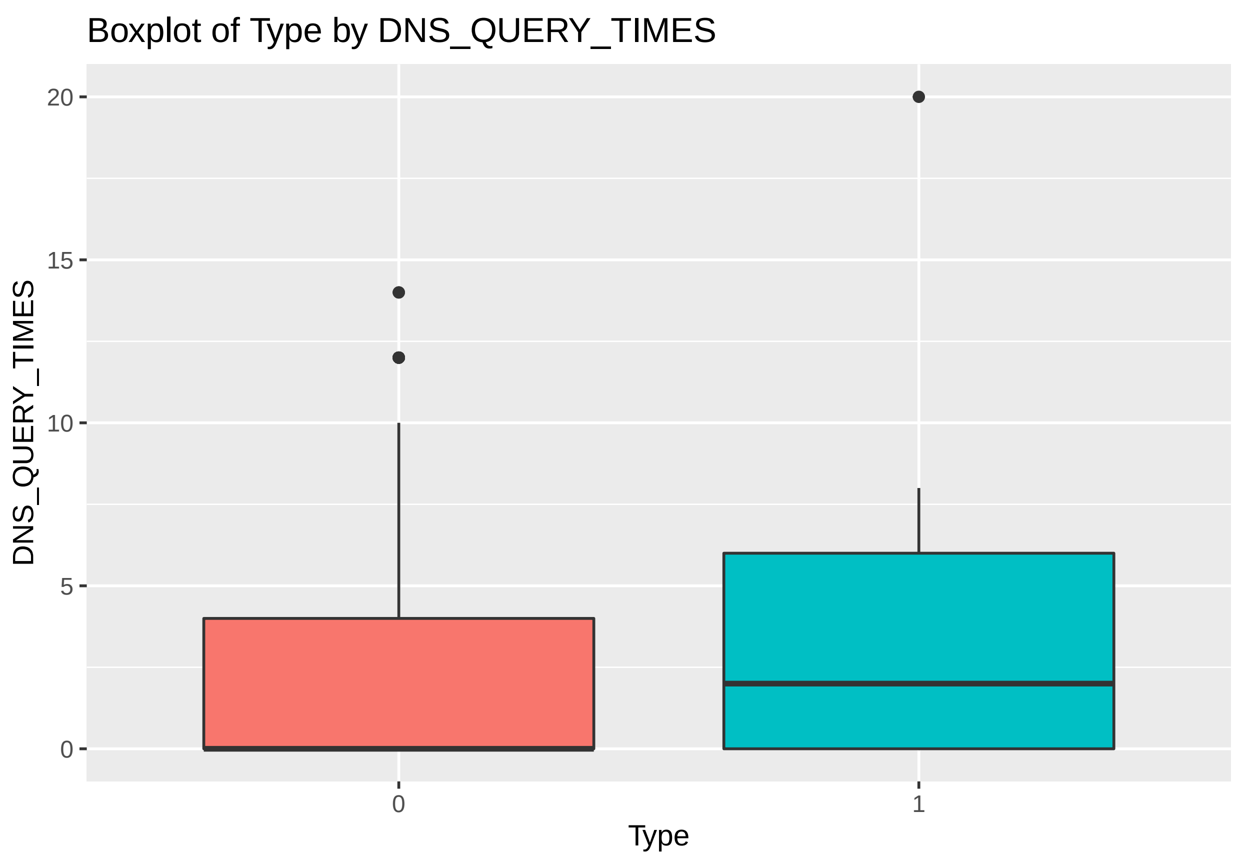
```
describe(train)
```

```
##                           vars    n     mean       sd median trimmed     mad
## URL_LENGTH                   1 1781    56.96    27.56     49   53.02   17.79
## NUMBER_SPECIAL_CHARACTERS    2 1781    11.11     4.55     10   10.42    2.97
## CHARSET*                     3 1781     3.96     1.37      5    4.07    0.00
## SERVER*                      4 1781    17.39    10.31     19   18.23   11.86
## WHOIS_COUNTRY*               5 1781    14.59     4.92     18   15.51    0.00
## WHOIS_STATEPRO*              6 1781    79.60    49.61    101   76.52   83.03
## WHOIS_UPDATED_DATE*          7 1781   328.22   174.83    341  330.29  212.01
## TCP_CONVERSATION_EXCHANGE    8 1781    16.26    40.50      7   10.55   10.38
## DIST_REMOTE_TCP_PORT         9 1781     5.47    21.81      0    2.22    0.00
## REMOTE_IPS                  10 1781     3.06     3.39      2    2.53    2.97
## APP_BYTES                   11 1781  2982.34 56050.57    672 1124.62  996.31
## SOURCE_APP_PACKETS          12 1781    18.54    41.63      8   12.63   11.86
## REMOTE_APP_PACKETS          13 1781    18.75    46.40      9   12.17   13.34
## SOURCE_APP_BYTES            14 1781 15892.55 69861.93    579 5264.55  858.43
## REMOTE_APP_BYTES            15 1781  3155.60 56053.78    735 1277.00 1089.71
## APP_PACKETS                 16 1781    18.54    41.63      8   12.63   11.86
## DNS_QUERY_TIMES             17 1781     2.26     2.93      0    1.80    0.00
## Type                        18 1781     0.12     0.33      0    0.03    0.00
## type*                       19 1781     1.12     0.33      1    1.03    0.00
##                           min     max   range  skew kurtosis      se
## URL_LENGTH                 16     249     233  1.80     4.99    0.65
## NUMBER_SPECIAL_CHARACTERS   5      43      38  1.88     5.23    0.11
## CHARSET*                    1       6       5 -0.67    -1.45    0.03
## SERVER*                     1      29      28 -0.60    -1.21    0.24
## WHOIS_COUNTRY*              1      19      18 -1.13     0.06    0.12
## WHOIS_STATEPRO*             1     182     181  0.21    -1.22    1.18
## WHOIS_UPDATED_DATE*         1     594     593 -0.09    -1.16    4.14
## TCP_CONVERSATION_EXCHANGE   0    1194    1194 17.58   451.64    0.96
## DIST_REMOTE_TCP_PORT        0     708     708 21.85   639.87    0.52
## REMOTE_IPS                  0      17      17  1.13     0.78    0.08
## APP_BYTES                   0 2362906 2362906 41.91  1761.45 1328.15
## SOURCE_APP_PACKETS          0    1198    1198 16.28   406.24    0.99
## REMOTE_APP_PACKETS          0    1284    1284 15.97   372.13    1.10
## SOURCE_APP_BYTES            0 2060012 2060012 18.24   459.14 1655.42
## REMOTE_APP_BYTES            0 2362906 2362906 41.89  1760.53 1328.23
## APP_PACKETS                 0    1198    1198 16.28   406.24    0.99
## DNS_QUERY_TIMES             0      20      20  1.12     0.80    0.07
## Type                        0       1       1  2.32     3.38    0.01
## type*                       1       2       1  2.32     3.38    0.01
```

It's important to look at the relationship of numeric variables to our response variable type. First, we'll look at the DNS_QUERY_TIMES variable against type variable:

```r
ggplot(train, aes(x=as.factor(Type), y=DNS_QUERY_TIMES, fill=as.factor(Type))) +
  geom_boxplot() +
  theme(legend.position="none")+
  ggtitle("Boxplot of Type by DNS_QUERY_TIMES")+
  xlab('Type')
```

```
## Don't know how to automatically pick scale for object of type impute. Defaulting to continuous.
```



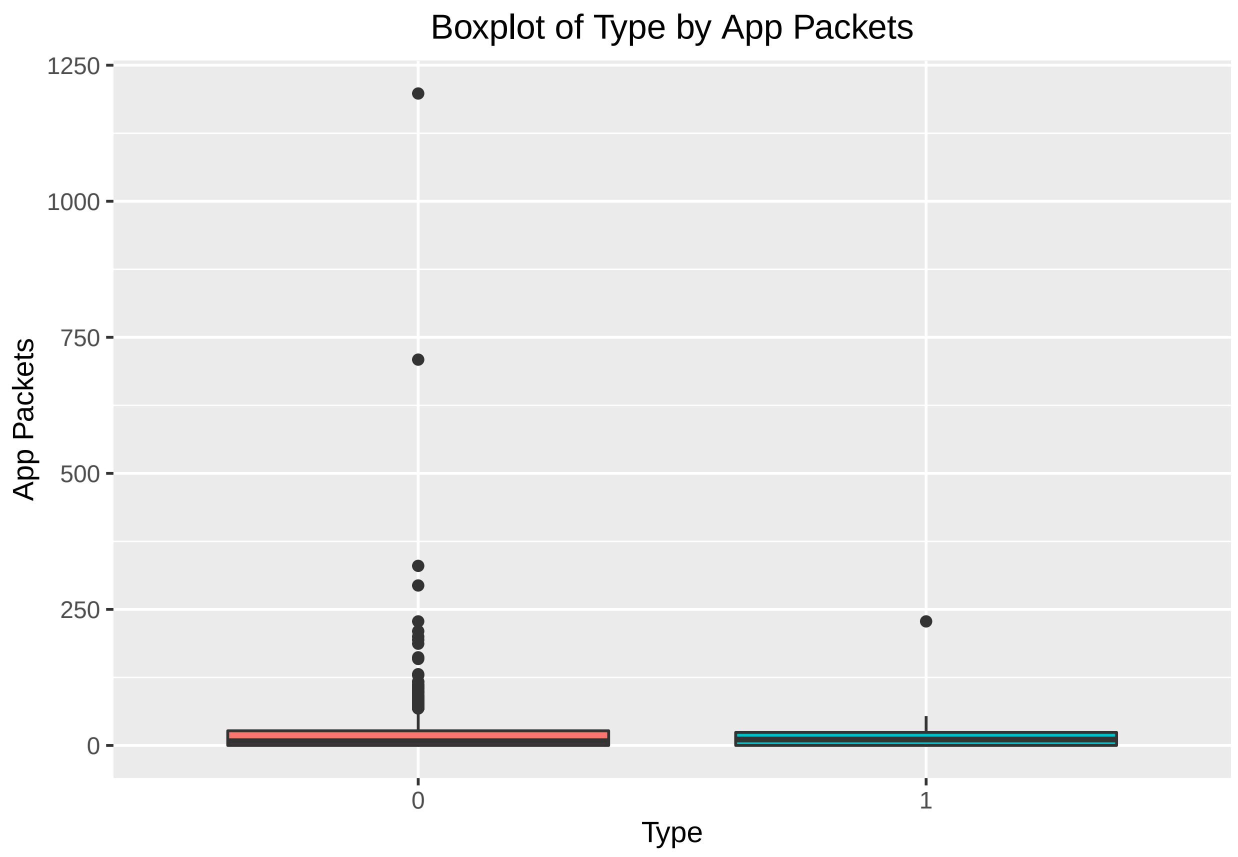
```
theme(plot.title = element_text(hjust = 0.5))
```

```
## List of 1
##  $ plot.title:List of 11
##   ..$ family       : NULL
##   ..$ face         : NULL
##   ..$ colour       : NULL
##   ..$ size         : NULL
##   ..$ hjust        : num 0.5
##   ..$ vjust        : NULL
##   ..$ angle        : NULL
##   ..$ lineheight   : NULL
##   ..$ margin       : NULL
##   ..$ debug        : NULL
##   ..$ inherit.blank: logi FALSE
##   ..- attr(*, "class")= chr [1:2] "element_text" "element"
##  - attr(*, "class")= chr [1:2] "theme" "gg"
##  - attr(*, "complete")= logi FALSE
##  - attr(*, "validate")= logi TRUE
```

There seems to be clear difference between distribution of DNS_QUERY_TIMES for Malicious and Benign URLs. This suggests that DNS_QUERY_TIMES can be a good predictor.

Now we'll look at the relationship for APP_PACKETS variable:

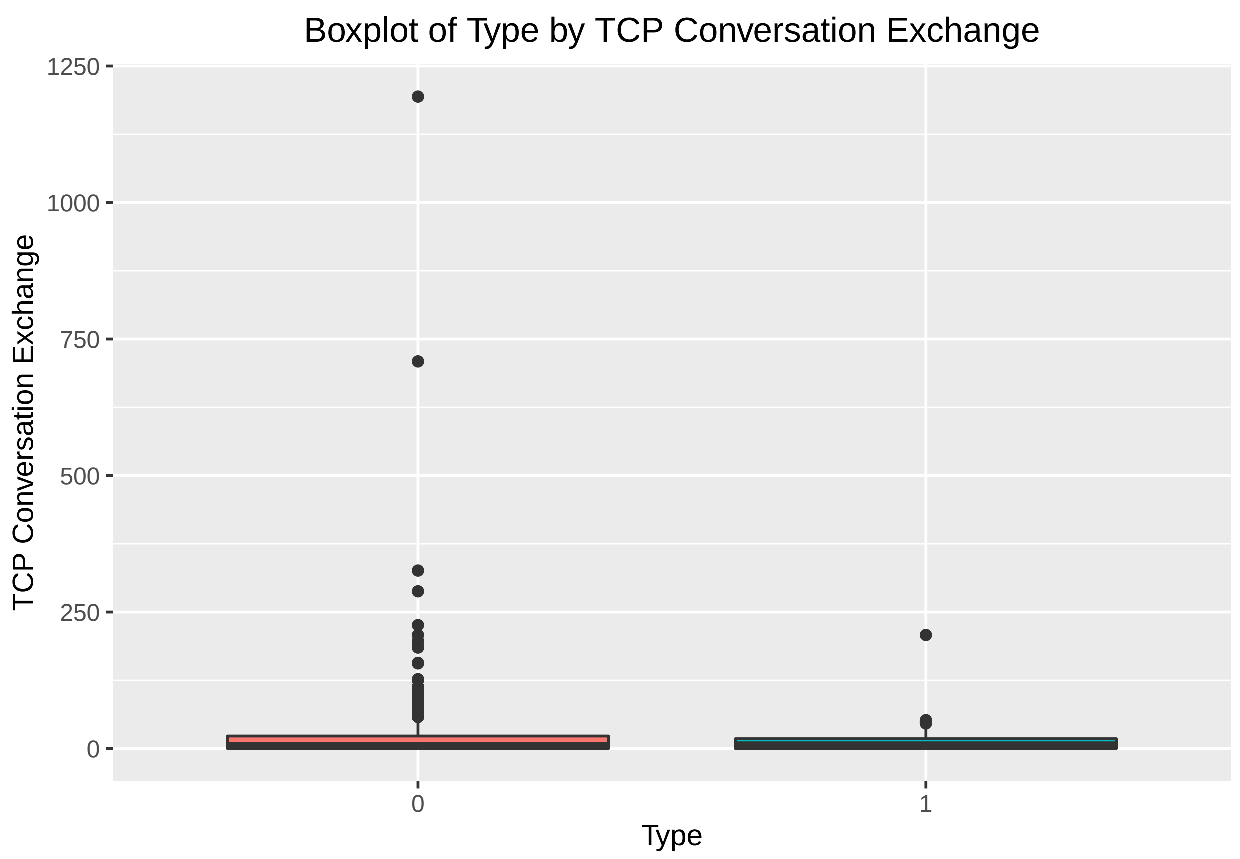
```r
train%>%ggplot(aes(x=as.factor(Type), y=APP_PACKETS, fill=as.factor(Type))) +
  geom_boxplot() +
  theme(legend.position="none")+
  ggtitle("Boxplot of Type by App Packets")+
  xlab('Type')+ylab('App Packets')+
  theme(plot.title = element_text(hjust = 0.5))
```



There are a lot of outliers in this variable and there seems to be no difference between malicious and benign URLs for the APP_PACKETS variable. Our model will explain if this variable has potential to be a good predictor.

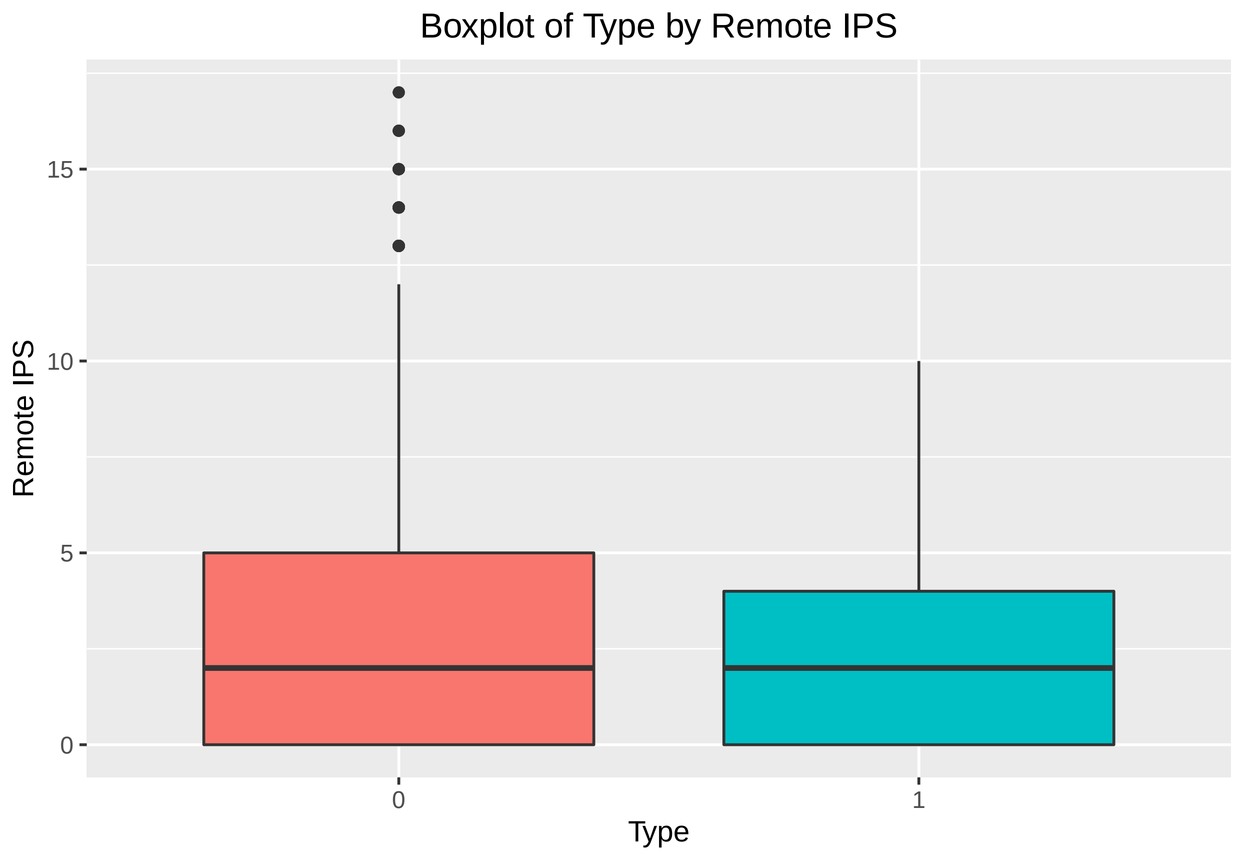
Let's look at the TCP_CONVERSATION_EXCHANGE variable:

```r
train%>%ggplot(aes(x=as.factor(Type), y=TCP_CONVERSATION_EXCHANGE, fill=as.factor(Type))) +
  geom_boxplot() +
  theme(legend.position="none")+
  ggtitle("Boxplot of Type by TCP Conversation Exchange")+
  xlab('Type')+ylab('TCP Conversation Exchange')+
  theme(plot.title = element_text(hjust = 0.5))
```



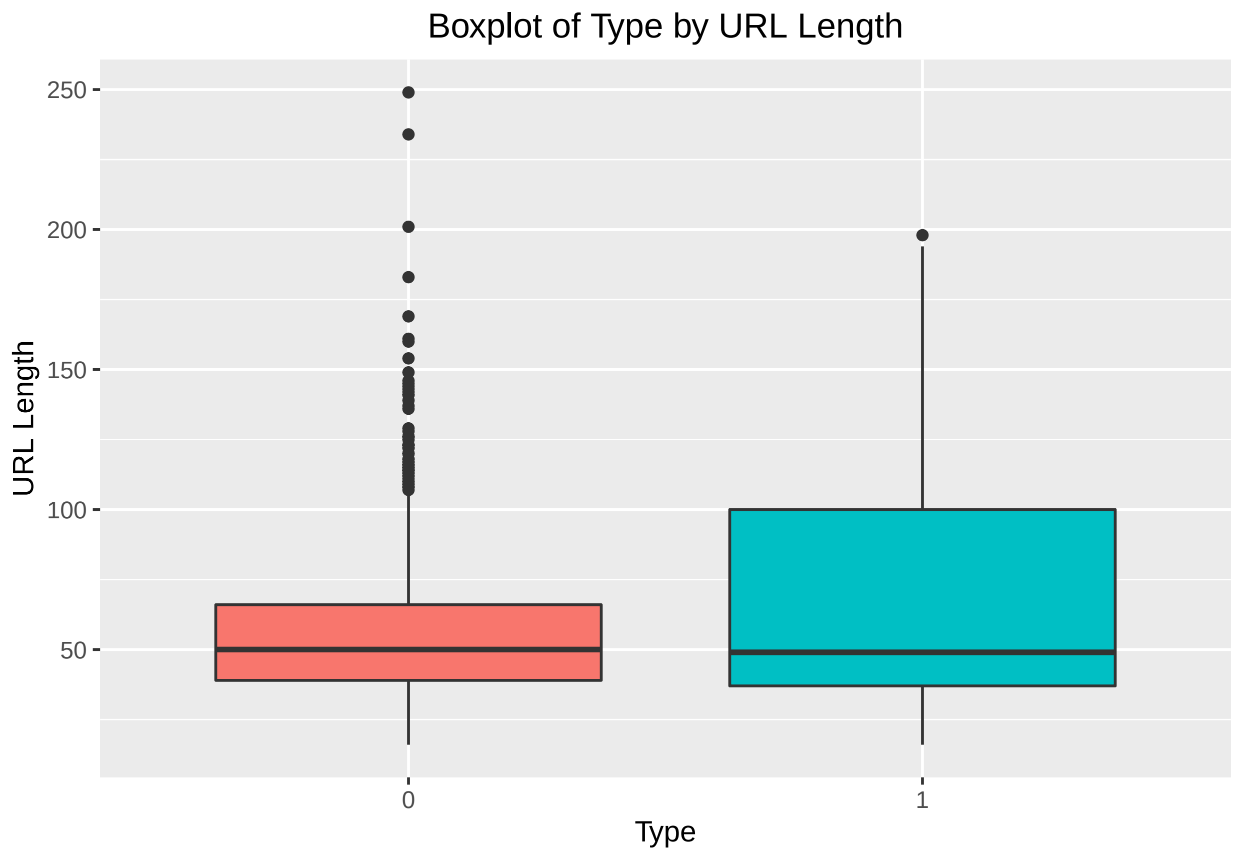
The distribution is similar to the App Packets and there are a lot of outliers in the dataset. Now let's look at the remote IPS variable:

```
train%>%ggplot(aes(x=as.factor(Type), y=REMOTE_IPS, fill=as.factor(Type))) +
  geom_boxplot() +
  theme(legend.position="none")+
  ggtitle("Boxplot of Type by Remote IPS")+
  xlab('Type')+ylab('Remote IPS')+
  theme(plot.title = element_text(hjust = 0.5))
```



Remote IPS has a similar median for Malicious and Benign URLs. However, the interquartile range of Malicious links is less than that of Benign URLs. Now, we'll look at the length of the URLs to see the distribution of it for our response variable.

```
train%>%ggplot(aes(x=as.factor(Type), y=URL_LENGTH, fill=as.factor(Type))) +
  geom_boxplot() +
  theme(legend.position="none")+
  ggtitle("Boxplot of Type by URL Length")+
  xlab('Type')+ylab('URL Length')+
  theme(plot.title = element_text(hjust = 0.5))
```

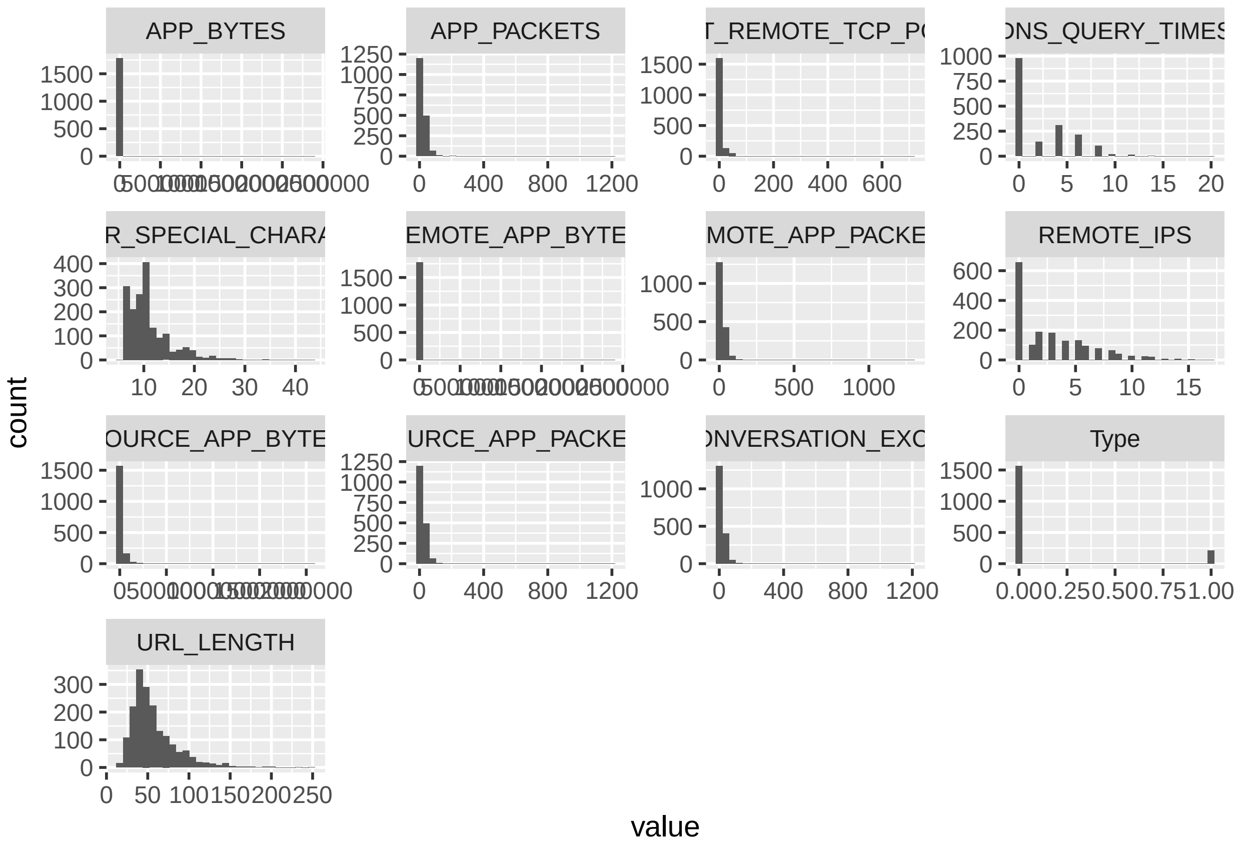


The shape of the distribution for Malicious and Benign URLs of URL length is quite different, so this can be a good identifier for the type of URL. Finally, we'll plot the distribution of all the numeric variables and see if there's a pattern:

```
train %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
  facet_wrap(~ key, scales = "free") +
  geom_histogram()
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It can be observed that most of the variables are skewed to the right and some variables have a spread out distribution. The variables with a spread out distribution can be helpful in prediction.

Creating the Test and Train Dataset:

First of all, we'll do the min-max normalization on the data and then create the training and test dataset. Our training dataset will be 80% of the observations.

```
minMax <- function(x) {
  return ((x - min(x)) / (max(x) - min(x))) }

train.subset <- train[c('URL_LENGTH','NUMBER_SPECIAL_CHARACTERS','TCP_CONVERSATION_EXCHANGE','DIST_REMO

train.subset.n <- as.data.frame(lapply(train.subset, minMax))
head(train.subset.n)
```

```
##    URL_LENGTH NUMBER_SPECIAL_CHARACTERS TCP_CONVERSATION_EXCHANGE
## 1 0.000000000                0.05263158               0.005862647
## 2 0.000000000                0.02631579               0.014237856
## 3 0.000000000                0.02631579               0.000000000
## 4 0.004291845                0.02631579               0.025963149
## 5 0.004291845                0.02631579               0.047738693
## 6 0.008583691                0.05263158               0.009212730
##   DIST_REMOTE_TCP_PORT REMOTE_IPS    APP_BYTES SOURCE_APP_PACKETS
## 1          0.000000000  0.1176471 0.0002962454        0.007512521
## 2          0.009887006  0.2352941 0.0005205455        0.014190317
## 3          0.000000000  0.0000000 0.0000000000        0.000000000
## 4          0.031073446  0.1764706 0.0016132677        0.032554257
## 5          0.002824859  0.2941176 0.0018104825        0.050918197
## 6          0.008474576  0.5294118 0.0003783477        0.009181970
##   REMOTE_APP_PACKETS Type
## 1        0.007788162    1
## 2        0.014797508    0
## 3        0.000000000    0
## 4        0.028816199    0
## 5        0.048286604    0
## 6        0.010124611    0
```

```
set.seed(123)
test_index <- sample(1:nrow(train.subset.n),size=nrow(train.subset.n)*0.2,replace = FALSE) #random sele

test.data <- train.subset[test_index,] # 20% will be test data
train.data <- train.subset[-test_index,] # remaining 80% will be training data

#Creating separate data frame for the 'Type' feature which is our target.
test.data_labels <-train.subset[test_index,9]
train.data_labels <- train.subset[-test_index,9]
```

Modelling and Testing:

We'll fit the Logistic Regression, KNN and Random forest models on the data and determine the performance of the models. The performance metrics that are important are Accuracy, Specificity and Sensitivity. Specificity tells us that how many Malicious URLs were correctly predicted by the model. That's why this is the most important metric for us as mentioned in the Introduction section.

Logistic Regression:

Now, we have our data ready for modelling. First, we'll use the logistic regression model. Logistic regression is quite a simple model which finds the logit function of the probability of the response variable using the equation used in linear modelling: $logit(p) = \beta_0 + \beta_i \times X_i + \epsilon_i$ We'll fit the logistic regression model on our data and predict it through our test data to check the accuracy and specificity of the results. Specificity is an important metric for us as it tells us how many of malicious URLs were correctly predicted by the algorithm. Let's fit the model and check the summary of the results:

```
glm_model <- glm(Type~.,data = train.data)

#Looking at the summary of the model
summary(glm_model)
```

```
##
## Call:
## glm(formula = Type ~ ., data = train.data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.51787  -0.14795  -0.06910   0.00344   1.60259
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -1.299e-01  2.277e-02  -5.707 1.40e-08 ***
## URL_LENGTH                -7.850e-03  7.452e-04 -10.534  < 2e-16 ***
## NUMBER_SPECIAL_CHARACTERS  6.315e-02  4.494e-03  14.050  < 2e-16 ***
## TCP_CONVERSATION_EXCHANGE -2.698e-02  4.135e-03  -6.525 9.45e-11 ***
## DIST_REMOTE_TCP_PORT      -3.598e-03  7.221e-04  -4.983 7.02e-07 ***
## REMOTE_IPS                -1.011e-02  3.114e-03  -3.246   0.0012 **
## APP_BYTES                  1.137e-06  2.370e-07   4.800 1.76e-06 ***
## SOURCE_APP_PACKETS         2.192e-02  3.506e-03   6.253 5.31e-10 ***
## REMOTE_APP_PACKETS         4.041e-03  1.576e-03   2.564   0.0105 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.08803696)
##
##     Null deviance: 153.51  on 1424  degrees of freedom
## Residual deviance: 124.66  on 1416  degrees of freedom
## AIC: 592.2
##
## Number of Fisher Scoring iterations: 2
```

We can see in the results of the model that all of the models are significant as their p-value is less than 0.05 level of significance. The AIC of the model is 592, which looks good for our purposes. Let's test the performance of the model on the test data:

```
predictions<- as.factor(ifelse(predict(glm_model, newdata = test.data,type = 'response')>0.5,1,0))

confusionMatrix(predictions,as.factor(test.data_labels))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 313  40
##          1   2   1
##
##                Accuracy : 0.882
##                  95% CI : (0.8439, 0.9136)
##     No Information Rate : 0.8848
##     P-Value [Acc > NIR] : 0.606
##
##                   Kappa : 0.0302
##
##  Mcnemar's Test P-Value : 1.135e-08
##
##             Sensitivity : 0.99365
##             Specificity : 0.02439
##          Pos Pred Value : 0.88669
##          Neg Pred Value : 0.33333
##              Prevalence : 0.88483
##          Detection Rate : 0.87921
##    Detection Prevalence : 0.99157
##       Balanced Accuracy : 0.50902
##
##        'Positive' Class : 0
##
```

Our logistic regression model gives us an accuracy of 88.2%, which isn't bad. However, we can see that the model is over-fitted on the data because the specificity of the model is very low (2.4%). The no information rate of the model is also very high. The results of the logistic regression model are not what we are looking for, so we'll test other models to achieve a higher accuracy.

KNN:

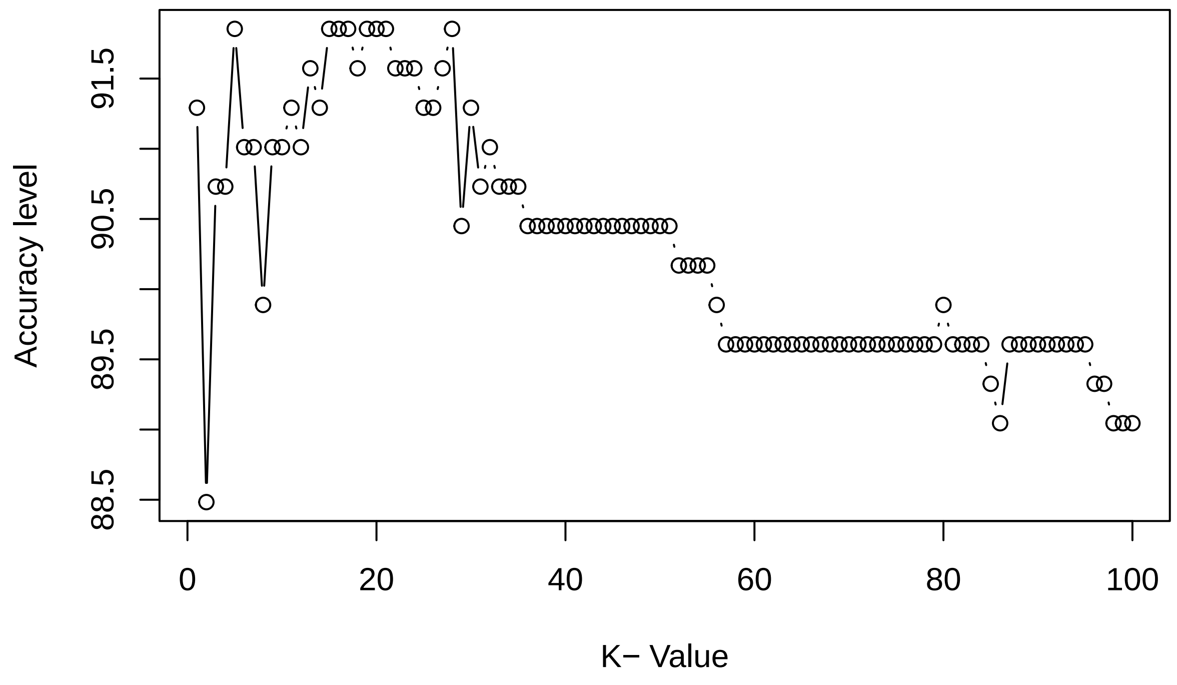
The KNN model checks all the data and classifies based on the similarity in the data. It is a non-parametric, supervised learning model and similar data is classified based on the nearest neighbor approach. It categorizes data based on similarity and classifies new cases based on their similarity with available categories. One important parameter in this model is the value of K. It represents the number of neighbors for assigning categories. We'll determine the optimal number of K through iterations. Let's fit KNN onto our data. We'll run a loop for KNN from 1 to 100 to test value of K and check accuracy at each point. The value of K which gives us highest accuracy will be selected.

```r
i=1                          # declaration to initiate for loop
k.optm=1                     # declaration to initiate for loop
for (i in 1:100){
  knn.mod <-  knn(train=train.data, test=test.data, cl=train.data_labels, k=i)
  k.optm[i] <- 100 * sum(test.data_labels == knn.mod)/NROW(test.data_labels)
  k=i
  cat(k,'=',k.optm[i],'\n')       # to print % accuracy
}
```

```
## 1 = 91.29213
## 2 = 88.48315
## 3 = 90.73034
## 4 = 90.73034
## 5 = 91.85393
## 6 = 91.01124
## 7 = 91.01124
## 8 = 89.88764
## 9 = 91.01124
## 10 = 91.01124
## 11 = 91.29213
## 12 = 91.01124
## 13 = 91.57303
## 14 = 91.29213
## 15 = 91.85393
## 16 = 91.85393
## 17 = 91.85393
## 18 = 91.57303
## 19 = 91.85393
## 20 = 91.85393
## 21 = 91.85393
## 22 = 91.57303
## 23 = 91.57303
## 24 = 91.57303
## 25 = 91.29213
## 26 = 91.29213
## 27 = 91.57303
## 28 = 91.85393
## 29 = 90.44944
## 30 = 91.29213
## 31 = 90.73034
## 32 = 91.01124
## 33 = 90.73034
## 34 = 90.73034
## 35 = 90.73034
## 36 = 90.44944
## 37 = 90.44944
## 38 = 90.44944
## 39 = 90.44944
## 40 = 90.44944
## 41 = 90.44944
## 42 = 90.44944
## 43 = 90.44944
## 44 = 90.44944
## 45 = 90.44944
## 46 = 90.44944
## 47 = 90.44944
## 48 = 90.44944
## 49 = 90.44944
## 50 = 90.44944
## 51 = 90.44944
## 52 = 90.16854
## 53 = 90.16854
## 54 = 90.16854
## 55 = 90.16854
## 56 = 89.88764
## 57 = 89.60674
## 58 = 89.60674
## 59 = 89.60674
## 60 = 89.60674
## 61 = 89.60674
## 62 = 89.60674
## 63 = 89.60674
## 64 = 89.60674
## 65 = 89.60674
## 66 = 89.60674
## 67 = 89.60674
## 68 = 89.60674
## 69 = 89.60674
## 70 = 89.60674
## 71 = 89.60674
## 72 = 89.60674
## 73 = 89.60674
## 74 = 89.60674
## 75 = 89.60674
## 76 = 89.60674
## 77 = 89.60674
## 78 = 89.60674
## 79 = 89.60674
## 80 = 89.88764
## 81 = 89.60674
## 82 = 89.60674
## 83 = 89.60674
## 84 = 89.60674
## 85 = 89.32584
## 86 = 89.04494
## 87 = 89.60674
## 88 = 89.60674
## 89 = 89.60674
## 90 = 89.60674
## 91 = 89.60674
## 92 = 89.60674
## 93 = 89.60674
## 94 = 89.60674
## 95 = 89.60674
## 96 = 89.32584
## 97 = 89.32584
## 98 = 89.04494
## 99 = 89.04494
## 100 = 89.04494
```

The series shows us the accuracy against value of K.

```
plot(k.optm, type="b", xlab="K- Value",ylab="Accuracy level")
```



As we can see from the plot and series, the optimal value of K is 17 as it yields the highest accuracy. Now we'll fit the KNN for the K value of 17:

```
knn.17 <- knn(train=train.data, test=test.data, cl=train.data_labels, k=17)
confusionMatrix(knn.17, as.factor(test.data_labels))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 311  25
##          1   4  16
##
##                Accuracy : 0.9185
##                  95% CI : (0.8851, 0.9448)
##     No Information Rate : 0.8848
##     P-Value [Acc > NIR] : 0.0241127
##
##                   Kappa : 0.4858
##
##  Mcnemar's Test P-Value : 0.0002041
##
##             Sensitivity : 0.9873
##             Specificity : 0.3902
##          Pos Pred Value : 0.9256
##          Neg Pred Value : 0.8000
##               Prevalence : 0.8848
##           Detection Rate : 0.8736
##     Detection Prevalence : 0.9438
##        Balanced Accuracy : 0.6888
##
##         'Positive' Class : 0
##
```

The accuracy of the model is 91.8 which is better than our previous model (logistic regression). The specificity of this KNN model is 39%, which is far better than logistic regression but it's still not what we are looking for. Now, we'll test our final model, Random Forest.

Random Forest:

The Random Forest model uses multiple decision trees to determine classifications. Random Forest classifies based on the general consensus provided from the decision trees involved. Let's fit the model and check the variable importance.

```
train.data$Type <- as.factor(train.data$Type)
modelrf <- randomForest(Type ~ ., data=train.data,ntree=1000)
(varimp.modelrf <- varImp(modelrf))
```

```
##                             Overall
## URL_LENGTH                 45.98348
## NUMBER_SPECIAL_CHARACTERS 45.04220
## TCP_CONVERSATION_EXCHANGE 26.51701
## DIST_REMOTE_TCP_PORT      38.72169
## REMOTE_IPS                21.56936
## APP_BYTES                 37.65647
## SOURCE_APP_PACKETS        28.64043
## REMOTE_APP_PACKETS        50.39126
```

We can see that URL_LENGTH and NUMBER_SPECIAL_CHARACTERS are the most important variables in this process. Let's test the output of the model on test data:

```
test.data$rf.pred <- predict(modelrf, newdata = test.data)
head(test.data[c("Type","rf.pred")], n=10)
```

```
##      Type rf.pred
## 415     0       0
## 463     0       0
## 179     0       0
## 526     0       0
## 195     0       0
## 938     0       0
## 1142    0       0
## 1323    0       0
## 1253    0       0
## 1268    0       0
```

```
confusionMatrix(as.factor(test.data$Type),as.factor(test.data$rf.pred))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 313   2
##          1   7  34
##
##                Accuracy : 0.9747
##                  95% CI : (0.9526, 0.9884)
##     No Information Rate : 0.8989
##     P-Value [Acc > NIR] : 2.792e-08
##
##                   Kappa : 0.869
##
##  Mcnemar's Test P-Value : 0.1824
##
##             Sensitivity : 0.9781
##             Specificity : 0.9444
##          Pos Pred Value : 0.9937
##          Neg Pred Value : 0.8293
##              Prevalence : 0.8989
##          Detection Rate : 0.8792
##    Detection Prevalence : 0.8848
##       Balanced Accuracy : 0.9613
##
##        'Positive' Class : 0
##
```

We can see that output of the Random Forest model is great on the test set. The overall accuracy of the model is 97% and specificity of the model is 94%. This result is great for our purposes and we'll use this model over the others.

Results

Over the course of this report, Logistic Regression, K-nearest neighbors (KNN) and Random Forest algorithms were utilized to find the model that best predicted whether a URL was malicious or benign.

We achieved a notable increase in accuracy from 88.2% to 97.47%, and an incredible jump in specificity from 2.4% to approximately 94%, with Logistic Regression being the former and Random Forests the latter.

Having accomplished both the highest accuracy and specificity amongst the three models, it is evident that the Random Forest algorithm is the best predictor of malicious or benign URLs.

Conclusion

Through this analysis, we were able to confirm that machine learning approaches regarding the prediction of Malicious and Benign URLs can be extremely helpful as they can decrease the risk of cyber crime and theft when applied successfully. In our analysis, the Random Forest model was able to secure a 94% Malicious URL prediction rate, which serves as a testament to the usefulness of such algorithms. Other advanced approaches (ex. neural network) can be more accurate in prediction. We saw that length of the URL and number of special characters in a URL are the most important when it comes to classification.

One important future implication is that a model trained through more detailed observations and with more advanced machine learning techniques can be deployed in real world situation. When considering the high prediction rates demonstrated by the models as a whole, their helpfulness becomes apparent, particularly in the cybersecurity sector.

While there is clear room for improvement, the algorithms used in this analysis found immense success in differentiating between Malicious and Benign URLs. Furthermore, we were able to determine that the Random Forest model produced the best results, making this venture an overall success.

Limitations

My Cyber ML Capstone Project encountered several limitations that I hope to address in the future, perhaps through the assistance of my peer reviewers and the feedback of the staff.

1) Software/Hardware Limitations: I have attempted several advanced machine learning models that could have outbested the Random Forest models, such as a potential incorporation of the cforest package, but the lackluster processing power and RAM of my laptop has currently made it difficult to do so.
2) Exploratory Limitations: A higher understanding of the Malicious and Benign URLs dataset would have allowed me to fine-tune the models further. I had much difficulty with the data wrangling portion of the project, so I hope to improve that in future endeavors.
3) Dataset Limitations: The Malicious and Benign URLs dataset contains only 1781 unique URLs. The small sample size undoubtedly limited the performance of our models. However, other datasets on Kaggle either faced the same predicament or lacked the variables I was looking for. I will continue to search for a higher-quality dataset so I can revisit and improve this project.

Future Works

As summarized above, my attempts at creating better models has been hampered by the limitations of my device and my understanding of the Malicious and Benign URLs dataset. However, I intend on trying the following in the near future.

1) Applying gradient boosting (through XGBoost) for regression purposes. As mentioned before, neural networks could be incorporated as well.
2) Redoing the project with a dataset with a much larger sample size. Could consider either broadening or narrowing the subject matter if necessary.

Acknowledgements

I would like to thank Professor Rafael Irizarry, as well as the countless educators, moderators, and supervisors that have contributed to the HarvardX Professional Data Science Certificate Program. Their collective efforts have provided one of the best remote learning experiences I have had to date, and I am extremely grateful to have studied the R programming language under such accredited curriculum. I would like to thank Christian Urcuquii for generously providing the Malicious and Benign URLs dataset on Kaggle for public use. I also want to share my gratitude and heartfelt thanks to my fellow peers, whose discussions on the interactive forums have given me invaluable insight throughout the journey. Lastly, I would like to acknowledge my parents, who have never stopped supporting my dreams.