Capstone Project Proposal

Introduction to flow Data Science by Springboard

Network Data Flow analysis

By Gabriel Fontenot

Introduction

The problem statement

Network engineers have been collecting flow data for decades regarding performance and behavior. The flow data sets collected include but not limited to device configurations, syslog (device messaging), and device snmptraps (fault management). Engineers and operations have for the most part used the collected flow data to assist with various troubleshooting and design exercises.

Using the statistical, proablitity and machine learning modules, the idea is to enhance the overall flow data analization experience and provide a conceise and targeted approach to understanding network and application behavior.

The host OS (distribution) is a constant and necesary part of the overall application structure. The intent of the analysis below is to understand how the OS is utilized by various components of the network to influence application behavior. This will include understanding the volument of traffic per OS and how each OS uses the network resourses.

The results inferred can further network engineers and architects understanding of applications and be considered when looking at network designs.

About the flow/host data set

The flow data sets subject to flow data analysis include a packet capture describing the overall network behavior (who is talking to who) and a description of the host communicating on the network during the specified time interval. The network flow data includes source and destination ip addresses along with host descriptions to assist in identifying traffic patterns and anomolies.

The intended flow data set has the following structures and descriptions.

Flows

1) host_name : Factor : the host name of the guest OS
2) last_software_update_at : int : the last time the host agent was updated
3) data_plane_disabled : logic : the status of the data plane forwarding
4) platform : Factor : the guest OS host distribution

5) agent_type : Factor : the current agent deployed on the guest os
6) current_sw_version : Factor : the current agent version as installed on the host
7) enable_pid_lookup : logic : the current status of the PID lookup functionality
8) last_config_fetch_at : int : the timeframe the guest OS last checked in

Sensors

1) start_timestamp : num : the timeframe the flow started
2) src_port : int : the source port for the selected flow
3) rev_pkts : int : the packet count for the reverse flows
4) rev_bytes : int : the byte count for the reverse flows
5) proto : Factor : the IP protocol used for the current flow
6) src_address : Factor : source IP ADDRESS of the host that initiated the flow
7) timestamp : Factor : the timestamp the flow was collected
8) fwd_bytes : int : the byte count for the forward direction of the flow
9) src_hostname : Factor : the host name of the source host in the flow
10) dst_address : Factor : destination IP ADDRESS of the host of which the source is communicating with
11) src_is_internal : Factor : the current state of the host relative to is location
12) dst_port : int : the destination port for teh selected flow
13) srtt_usec : int : the srtt latency associated with the current flow
14) vrf_id : int : the vrf id for the current flow
15) vrf_name : Factor : the vrf name for the current flow
16) fwd_pkts : int : the packet count for the forward direction of the flow
17) server_app_latency_usec : int : the application latency as derived from the application agent
18) total_network_latency_usec: int : the calculated network latency
19) total_pkts : int : the total fwd and rev packet count
20) platform_windowns : logic : the state of the platform, windows
21) total_bytes : int : the toal fwd and rev byte count

Evaluating the flow data

RStudio Libraries

The libraries listed below are utilized to provide functionality required to parse, analyze and display the appropriate flow data sets.

```
library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(rjson)
library(lubridate)
library(RColorBrewer)
library(knitr)
library(caTools)
```

Loading the data sets

The ‘csv’ files representing the flow analysis are loaded into the variable flowData and SensorData. The data is comprised of a network capture of multiple host communicating on the network during the define collection period.

Cleaning the flow/host data sets

The prepare the data for analysis, the data must first be normalized. The multiple step process includes the following steps.

1) Remove ROWs with non IPv4 addresses in Source and Destination

The collected data set contains IPv4 and IPv6 addresses. For the purpuse of this exercise, all IPv6 addresses will be removed.

2) Remove ROWs with no (missing) destination HOSTNAME

The collected data set contains hosts without a defined hostname. These flows will be identified and removed from the analysis as they are flows that do not contain agents.

3) Merge data sets, flows and hosts

To getter a holistic view of the flows, the flow data and the host data will be combined. This effort will allow for additional oppurtunities to corelate flow and host data.

4) Add new column Total Packet, total_pkts

To provide summarization for the packet count, a sumamry field of the fwd and rev packets will be added.

5) Add new column Total Bytes, total_bytes

To provide summarization for the byte count, a sumamry field of the fwd and rev bytes will be added.

6) Add new column Platform Type, platform_type

7) Add new column Time in Mins, time_min

8) Filter the data set if needed to represent a single application or set of hosts

9) Create a cleaned CSV file representing the data to be analyzed.

The cleaned data will be saved into a CSV file callsed flowDataSensor_clean.

10) List object variables

Counting records and variables

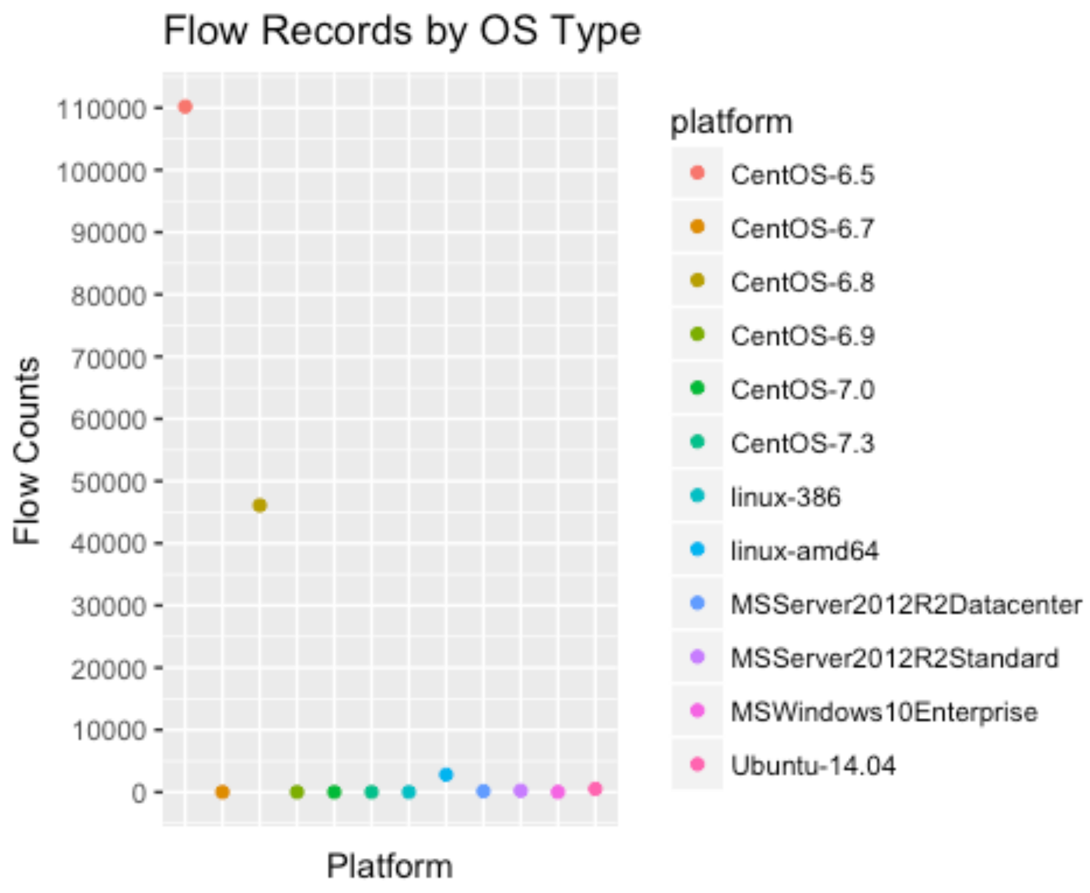
Using R methods and functionality, determine various flow data set counts. Counting various aspects of your flow data set can prepare you for dealing with the content during analysis.

- Count the number of rows and columns in the flow data set, SensorData.csv.
- Count the number of rows in the flow data set that have a platform of "CentOS-6.5"

Understanding the data set as related to host OS.

Using the combined data sets, we can get a better understanding of the guest OS distribution accross the collected flows. This first step in the overall data analysis provides insight into the overall distribution.

```r
a <- data %>% group_by(platform) %>% tally

a %>% ggplot() +
  geom_point(aes(x=platform, y=n, color = platform)) +
  scale_y_continuous(breaks=number_ticks(10)) +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank()) +
  xlab("Platform") +
  ylab("Flow Counts") +
  ggtitle(paste("Flow Records by OS Type"))
```



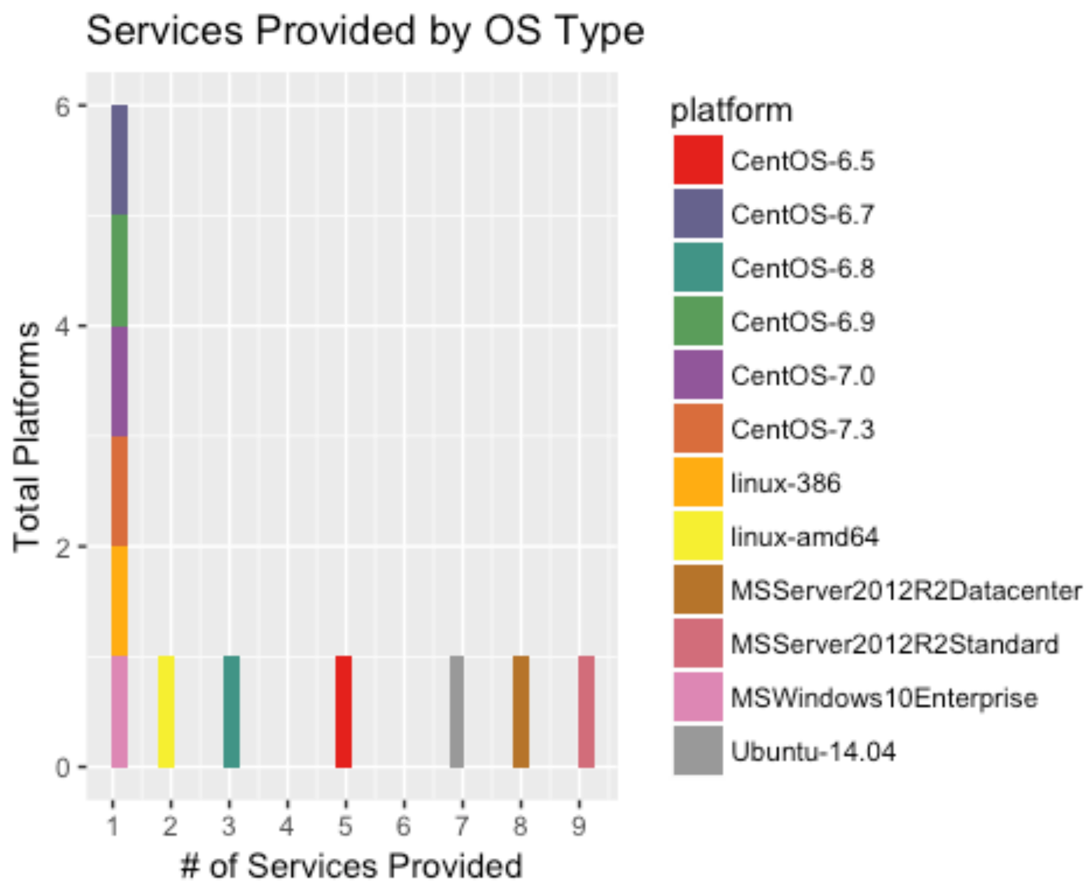
```
#data$timestamp <- factor(data$timestamp, levels=unique(data$timestamp))
```

The plot above shows CentOS-6.5 as the most popular OS utilized during the provided flow capture. The source of the traffic, host OS, is not evenly distributed accross OS's.

Services provided per OS

As we explore the flow data set further, we can further understand the services provided by the guest OS. This may be an indication of the guest OS (application) that is the most utlized on the network during the flow capture timeframe

```
data_port <- data %>% group_by(platform) %>% summarise(n_distinct(dst_port))
names(data_port)[2]<-paste("services_provided")

data_port %>% ggplot() +
  geom_histogram(aes(x=services_provided, fill=platform)) +
  scale_fill_manual(values = getPalette(colourCount)) +
  scale_x_continuous(breaks=number_ticks(7)) +
  ylab("Total Platforms") +
  xlab("# of Services Provided") +
  ggtitle(paste("Services Provided by OS Type"))
```

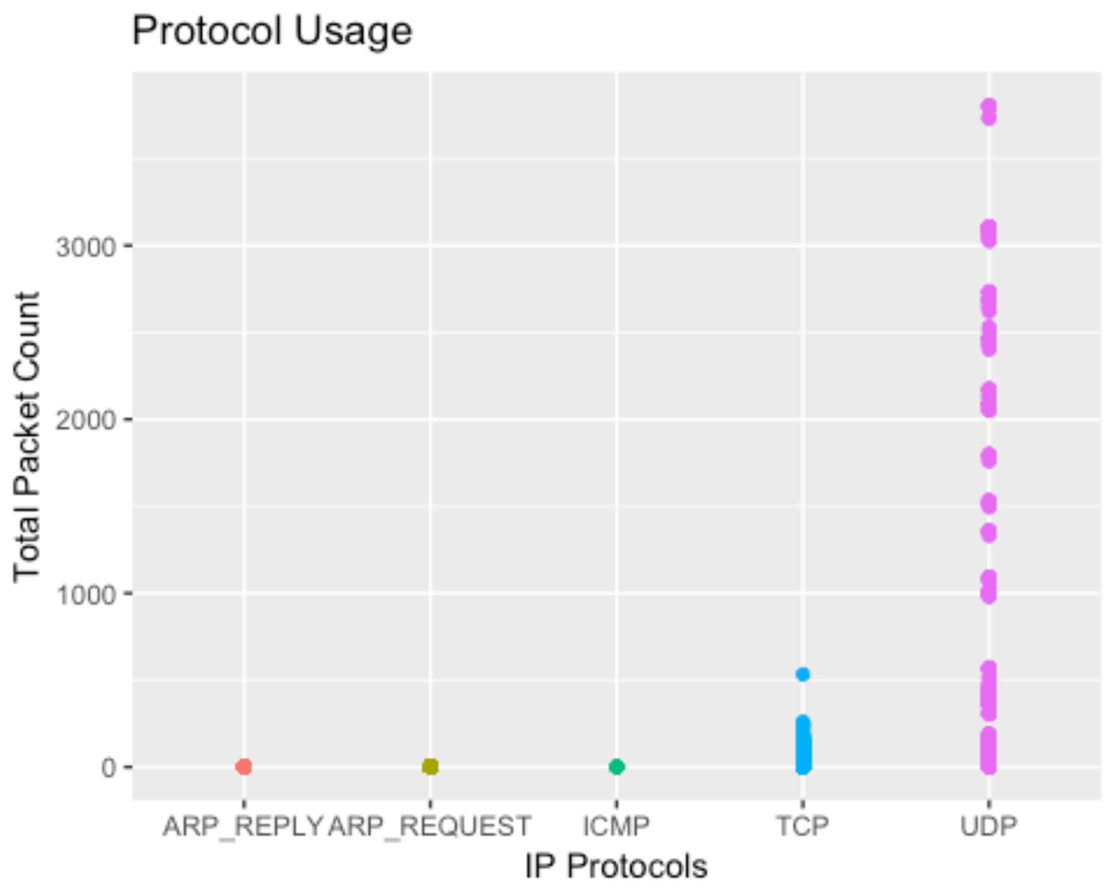


As indicated in the histogram above, we can determine that majority of the guest OS's are providing a single service. This can be interpreteded as the majority of the OSs analyzed within the flow analysis are providing serivce to a single application.

Overall Protocol Usage

As we continue to understand how applications consume overall flows within the network, it's useful to understand overall protocol distribution. The section will focus on understanding how IP protocols are implemented for the flow capture period.

```
ggplot(data, aes(proto, total_pkts)) +
  geom_point(aes(colour = proto), na.rm=TRUE) +
  theme(legend.position="none") +
  ylab("Total Packet Count") +
  xlab("IP Protocols") +
  ggtitle(paste("Protocol Usage"))
```

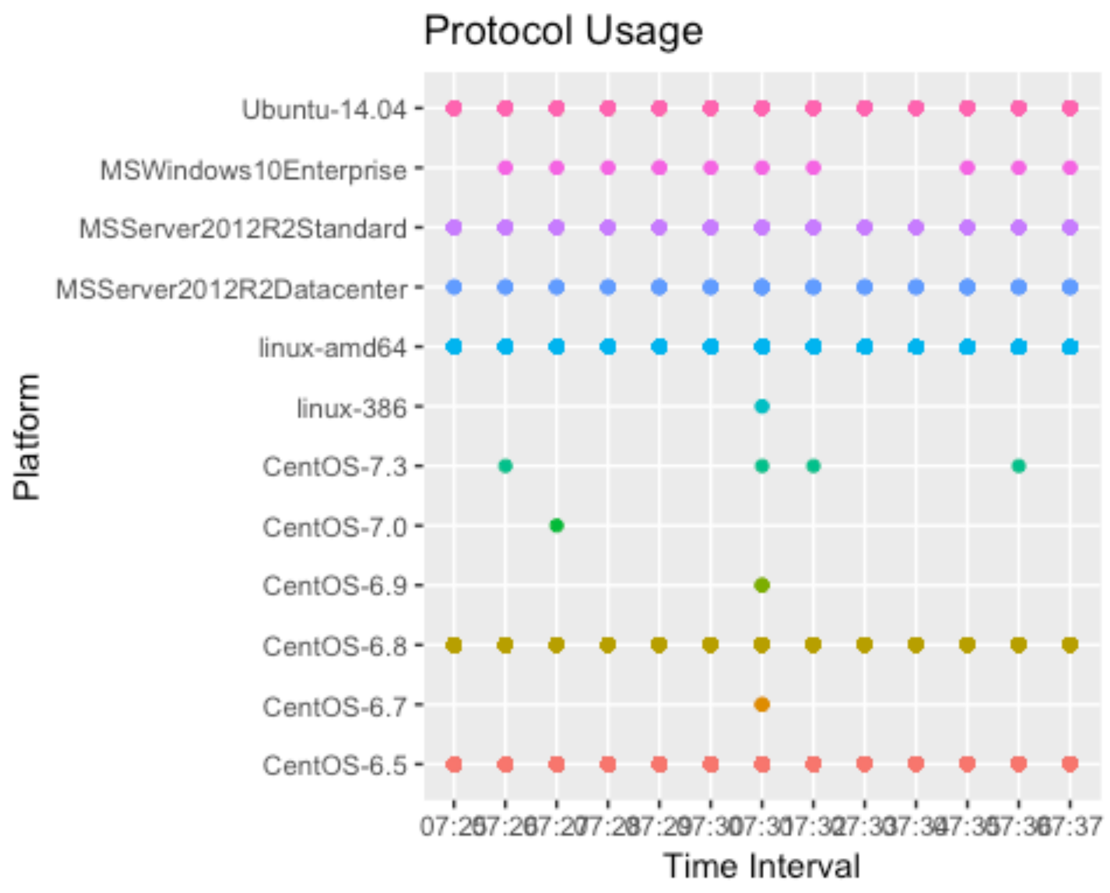


From the results we can cleary determine that UDP is predominately utilized via the applications analysed during the capture period. The range of packets part of the UDP flows extend from 1 to 3800+ packets per flow.

Platform usage by Time

By further evaluating the OS traffic over time, we can get an idea as to when the applications communicated accross the network. The network consumption can be used to understand traffic patterns and network usage.

```r
ggplot(data, aes(x=time_min, y=platform)) +
  geom_point(na.rm=TRUE, aes(colour = platform)) +
  theme(legend.position="none") +
  ylab("Platform") +
  xlab("Time Interval") +
  ggtitle(paste("Protocol Usage"))
```



As indicated in the table above, majority of the OS distributions commincate throught the interval of time with a couple of distriburtions only utilizing only network resources on accasion.

Conclusion

TBD

MISC

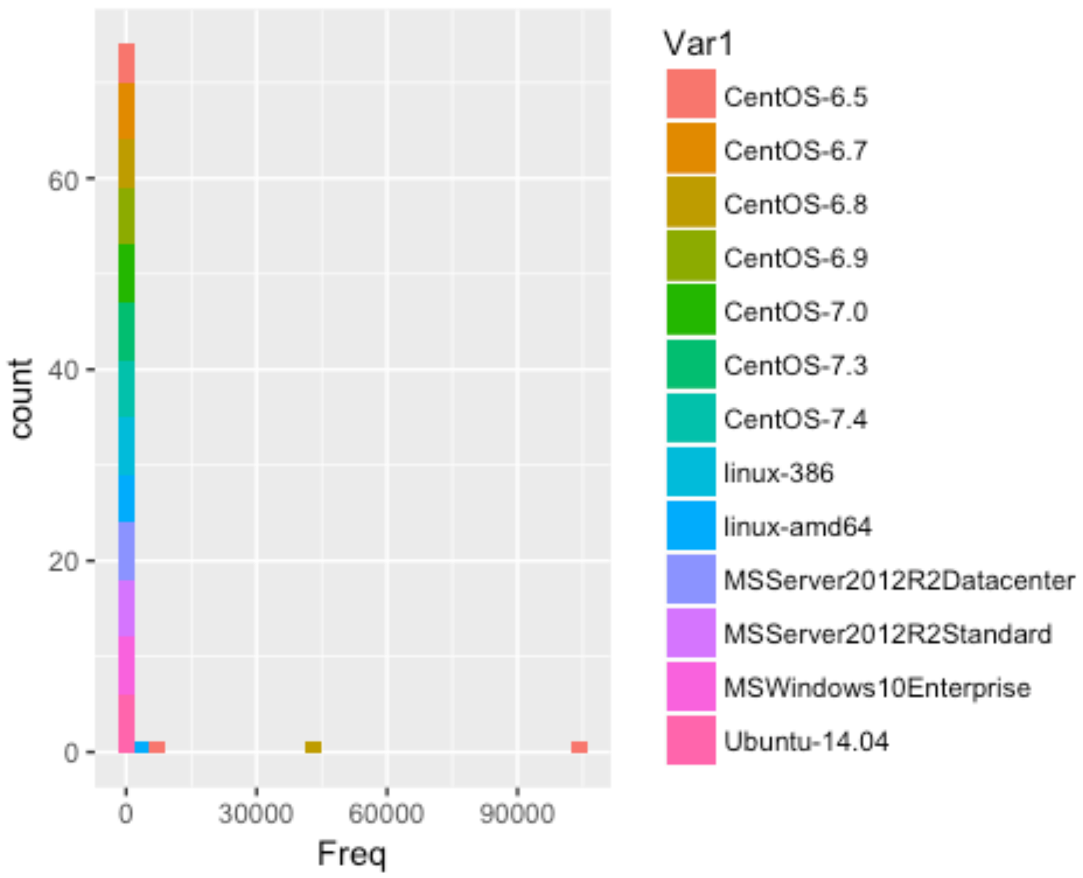
```
myTable <- table(data$platform, data$proto)

kable(myTable, caption = "My First Table")
```

My First Table

	ARP_REPLY	ARP_REQUEST	ICMP	ICMP6	TCP	UDP
CentOS-6.5	0	32	0	0	104219	5948
CentOS-6.7	0	3	0	0	0	0
CentOS-6.8	909	908	0	0	44282	0

CentOS-6.9	0	3	0	0	0	0
CentOS-7.0	0	1	0	0	0	0
CentOS-7.3	0	3	1	0	0	0
CentOS-7.4	0	0	0	0	0	0
linux-386	0	2	0	0	0	0
linux-amd64	0	26	0	0	2789	0
MSServer2012R2Datacenter	24	2	0	0	39	88
MSServer2012R2Standard	56	28	7	0	44	97
MSWindows10Enterprise	0	22	0	0	0	0
Ubuntu-14.04	3	107	0	0	406	0

```
myFrame <- as.data.frame((myTable))
myFrame %>% ggplot() +
  geom_histogram(aes(x=Freq, fill=Var1))
```



Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable.

In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

```
#Logistic Regrsssion

# Generate a TEST AND TRAINING SET
set.seed(100)
split = sample.split(data$platform_type, SplitRatio = 0.55)
dataTrain = subset(data, split == TRUE)
nrow(dataTrain)

## [1] 88027

dataTest = subset(data, split == FALSE)
nrow(dataTest)

## [1] 72022

# Determine the total for each field of the dependent variable
table(dataTrain$platform_type)

##
##     0     1
##   224 87803

# Update the dependent variable field to convert it to a vector
data$platform_type <- as.factor(data$platform_type)

# Logistic Regression
platformLog = glm(platform_type ~ proto + total_pkts, data = dataTrain,
family = binomial)
summary(platformLog)

##
## Call:
## glm(formula = platform_type ~ proto + total_pkts, family = binomial,
##     data = dataTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.6802   0.0195   0.0378   0.0418   1.8930
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       2.227004   0.148657  14.981  < 2e-16 ***
## protoARP_REQUEST  0.609847   0.236678   2.577 0.009975 **
## protoICMP        -3.887210   1.105447  -3.516 0.000437 ***
## protoTCP          4.663631   0.213033  21.892  < 2e-16 ***
## protoUDP          0.148689   0.194340   0.765 0.444213
## total_pkts        0.050768   0.006653   7.631 2.33e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3123.7  on 88026  degrees of freedom
## Residual deviance: 2063.0  on 88021  degrees of freedom
## AIC: 2075
##
## Number of Fisher Scoring iterations: 12
```

```
# Confidence Interval
confint(platformLog)
```

```
##                        2.5 %      97.5 %
## (Intercept)       1.9465091  2.53048821
## protoARP_REQUEST  0.1516072  1.08286176
## protoICMP        -6.8504937 -2.03757385
## protoTCP          4.2462729  5.08380967
## protoUDP         -0.2369815  0.52635869
## total_pkts        0.0386604  0.06487838
```

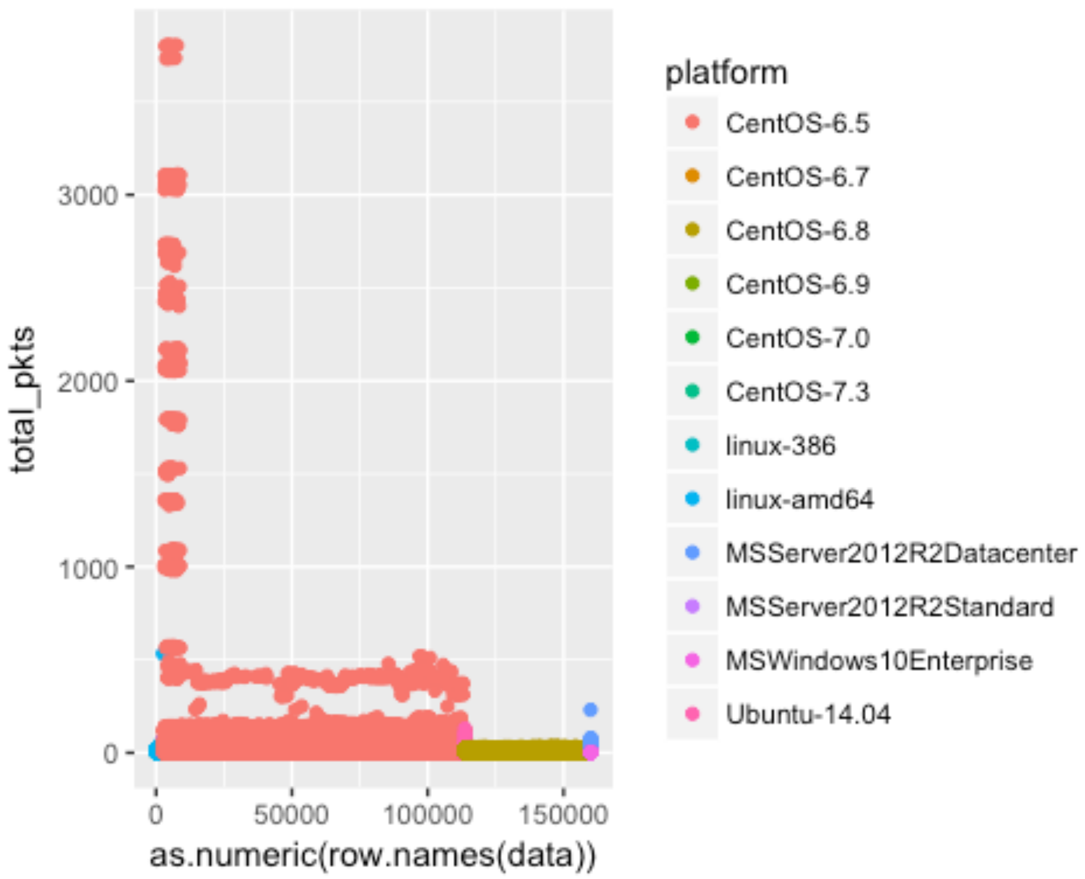
```
#exp(coef(platformLog))
#exp(cbind(OR = coef(platformLog), confint(platformLog)))

# Prediction
predictTrain = predict(platformLog, type="response")
summary(predictTrain)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1667  0.9991  0.9993  0.9975  0.9998  1.0000
```

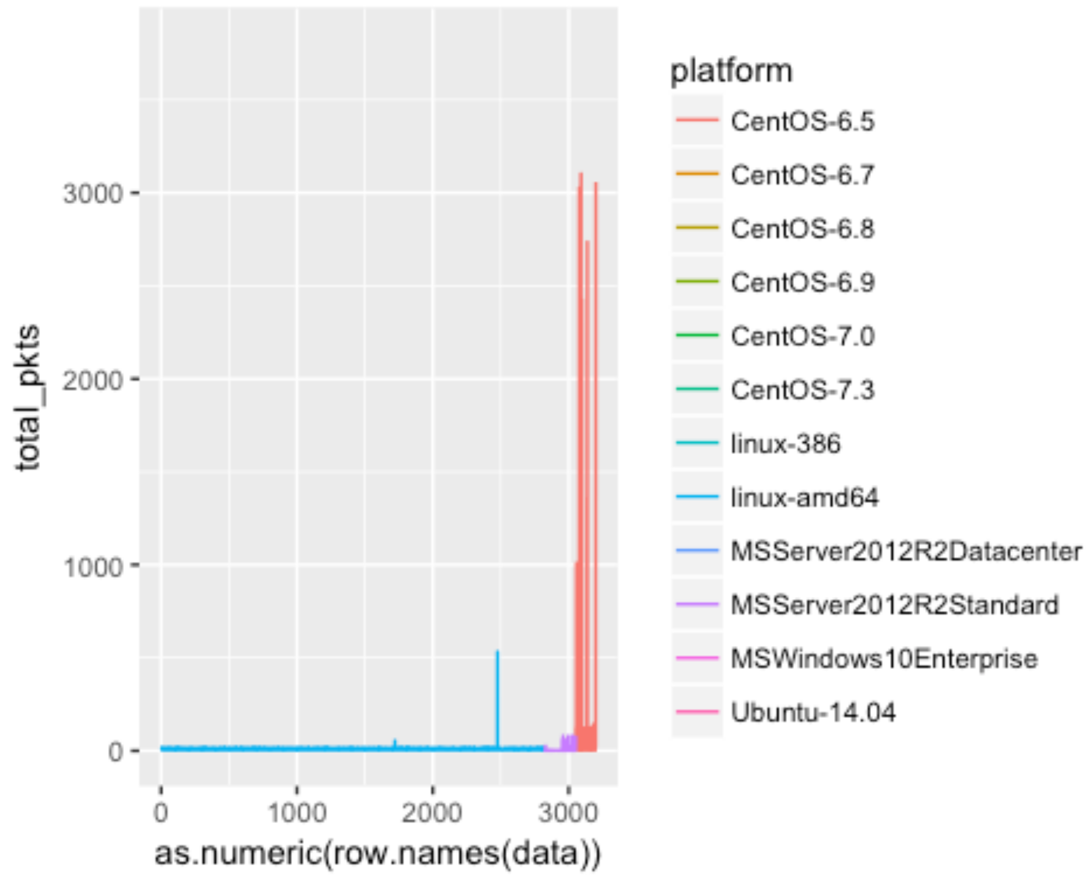
```
# Display the mean of the Training set
tapply(predictTrain, dataTrain$platform_type, mean)
```

```
##         0         1
## 0.9219886 0.9976479
```

```
# 1 is Windows Platform
# 0 is Linux


#ggplot(data, aes(x=timestamp, y=rev_pkts)) +
#  geom_line()

# describes the traffic usage per OS type based on total_pkts per capture
ggplot(data, aes(x = as.numeric(row.names(data)), total_pkts, color =
platform)) +
  geom_point()
```



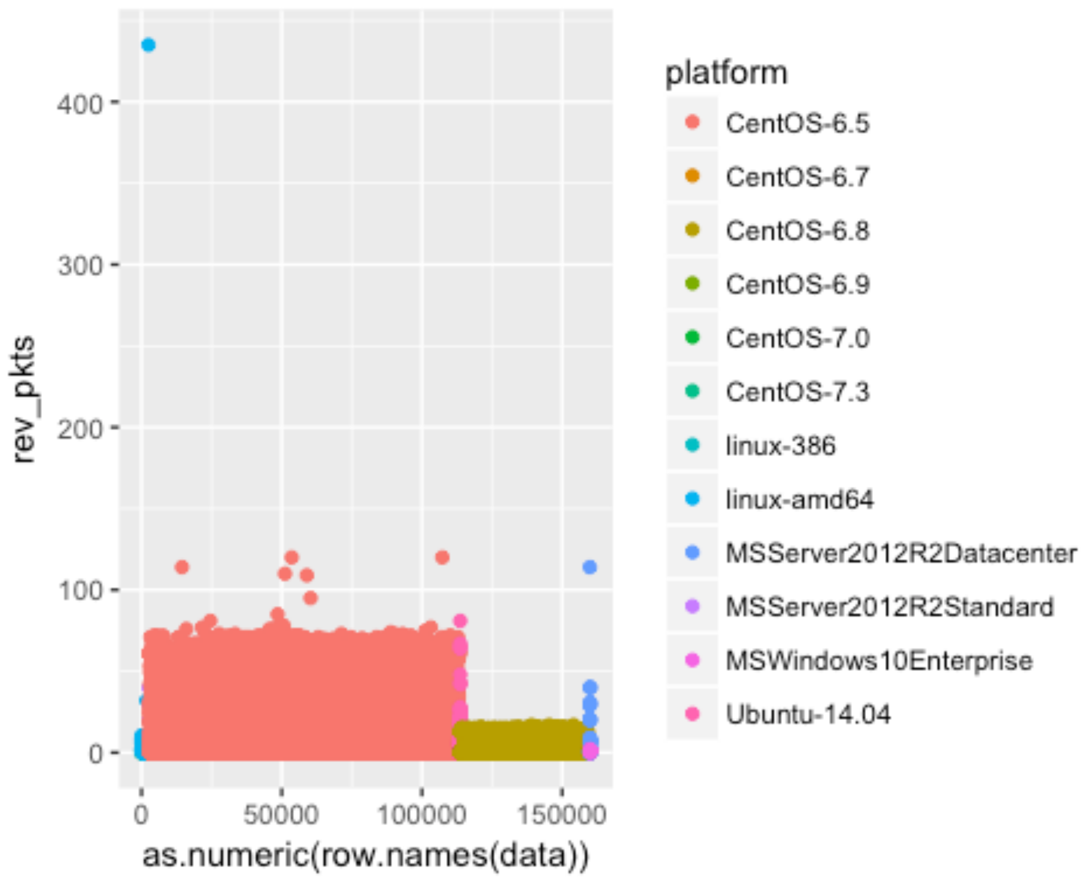
```
# describes the traffic usage per OS type based on total_pkts per capture
ZOOMed in
ggplot(data, aes(x = as.numeric(row.names(data)), total_pkts, color =
platform)) +
  geom_line() +
  xlim(0,3200)

## Warning: Removed 110053 rows containing missing values (geom_path).
```



K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k means clustering, we have to specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster.

K-Means is a clustering approach that belogs to the class of unsupervised statistical learning methods. K-means clustering is the most popular partitioning method. It requires the analyst to specify the number of clusters to extract. A plot of the within groups sum of squares by number of clusters extracted can help determine the appropriate number of clusters.

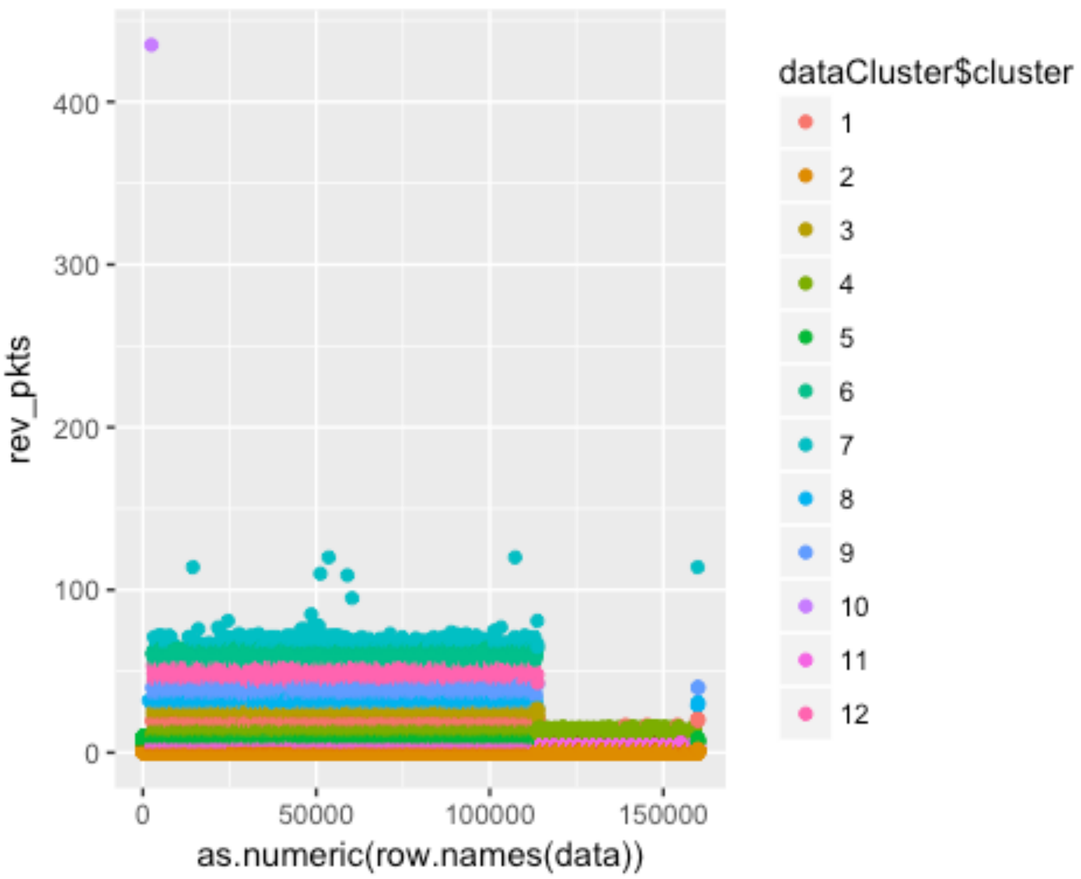
```
# K Means Clustering in R

#explore data
#head(data)

#plot data (before)
ggplot(data, aes(as.numeric(row.names(data)), rev_pkts, color = platform)) +
geom_point()
```



```
#clustering data
set.seed(20)
dataCluster <- kmeans(data[, 11], 12, nstart = 20)
#dataCluster

#compare cluster output with platform
table(dataCluster$cluster, data$platform)

##
##      CentOS-6.5 CentOS-6.7 CentOS-6.8 CentOS-6.9 CentOS-7.0 CentOS-7.3
##   1        4232          0          3          0          0          0
##   2       40250          3      34767          3          1          4
##   3        6824          0          0          0          0          0
##   4        8313          0       7387          0          0          0
##   5       11503          0          0          0          0          0
##   6       21018          0          0          0          0          0
##   7         491          0          0          0          0          0
##   8        2633          0          0          0          0          0
##   9        1990          0          0          0          0          0
##   10          0          0          0          0          0          0
##   11      11545          0       3942          0          0          0
##   12       1400          0          0          0          0          0
##
```

```
##      CentOS-7.4 linux-386 linux-amd64 MSServer2012R2Datacenter
##   1           0         0           0                        5
##   2           0         2        1871                      117
##   3           0         0           0                        0
##   4           0         0           0                        0
##   5           0         0         941                       17
##   6           0         0           0                        0
##   7           0         0           0                        1
##   8           0         0           1                        5
##   9           0         0           0                        3
##   10          0         0           1                        0
##   11          0         0           1                        5
##   12          0         0           0                        0
##
##      MSServer2012R2Standard MSWindows10Enterprise Ubuntu-14.04
##   1                       4                     0           15
##   2                     215                    22          114
##   3                       0                     0            7
##   4                       3                     0           23
##   5                       0                     0          209
##   6                       0                     0            1
##   7                       0                     0            3
##   8                       4                     0            1
##   9                       5                     0            1
##   10                      0                     0            0
##   11                      1                     0          140
##   12                      0                     0            2
```

```
#plot data (after)
dataCluster$cluster <- as.factor(dataCluster$cluster)
ggplot(data, aes(as.numeric(row.names(data)), rev_pkts, color =
dataCluster$cluster)) + geom_point()
```



rev_pkts
400
300
200
100
0
0
50000
100000
150000
as.numeric(row.names(data))
dataCluster$cluster
1
2
3
4
5
6
7
8
9
10
11
12