MICROSOFT MALWARE PREDICTION

TABLE OF CONTENTS

INDEX

ABSTRACT

The malware industry continues to be a well-organized, well-funded market dedicated to evading traditional security measures. Once a computer is infected by malware, criminals can hurt consumers and enterprises in many ways. With more than one billion enterprise and consumer customers worldwide, Microsoft takes this problem seriously and has always invested dedicatedly in improving security measures.

As one part of their overall strategy for doing so, Microsoft challenged the data science community keeping Kaggle as their platform to let data scientists across the globe develop techniques to predict if a machine will soon be hit with Malware. Microsoft provided Kagglers with unprecedented malware dataset of 12 Million plus records and a total size of 9GB (Training + Test) to encourage effective predictive techniques for predicting malware occurrences.

I completed the challenge on Kaggle with 0.65 as private score and 0.63 as public score. For the purpose of this report, keeping in mind the limitations of my personal laptop, I will be using close to 2.5 million records for Modelling and EDA is done on 9 million records.

The aim of this project is to find out, If I can help protect more than one billion machines from damage BEFORE it happens?

EXPLANATORY DATA ANALYSIS

Reading the dataset as a data table is almost 10 times more faster than reading it as a data frame using write.csv

```
```{r }

#Read Training Data
mltrain.df <- fread('train.csv',na.strings = c("","NA"))
```
```

```
|--------------------------------------------------|
|==================================================|
|--------------------------------------------------|
|==================================================|
```

```
```{r}
#Shape of the dataframe
shape = as.array(dim(mltrain.df))
shape
```
```

```
[1] 8921483      83
```

```
```{r}
#Check the size of dataset.
print(object.size(mltrain.df), units='auto')
```
```

```
4.7 Gb
```

Meta Data as given by Microsoft Inc.

Attribute	Description
MachineIdentifier	Individual machine ID
ProductName	Defender state information e.g. win8defender
EngineVersion	Defender state information e.g. 1.1.12603.0
AppVersion	Defender state information e.g. 4.9.10586.0
AvSigVersion	Defender state information e.g. 1.217.1014.0
IsBeta	Defender state information e.g. false
RtpStateBitfield	NA
IsSxsPassiveMode	NA
DefaultBrowsersIdentifier	ID for the machine's default browser
AVProductStatesIdentifier	ID for the specific configuration of a user's antivirus software
AVProductsInstalled	NA
AVProductsEnabled	NA
HasTpm	True if machine has tpm

CountryIdentifier	ID for the country the machine is located in
CityIdentifier	ID for the city the machine is located in
OrganizationIdentifier	ID for the organization the machine belongs in, organization ID is mapped to both specific companies and broad industries
GeoNameIdentifier	ID for the geographic region a machine is located in
LocaleEnglishNameIdentifier	English name of Locale ID of the current user
Platform	Calculates platform name (of OS related properties and processor property)
Processor	This is the process architecture of the installed operating system
OsVer	Version of the current operating system
OsBuild	Build of the current operating system
OsSuite	Product suite mask for the current operating system.
OsPlatformSubRelease	Returns the OS Platform sub-release (Windows Vista, Windows 7, Windows 8, TH1, TH2)
OsBuildLab	Build lab that generated the current OS. Example: 9600.17630.amd64fre.winblue_r7.150109-2022
SkuEdition	The goal of this feature is to use the Product Type defined in the MSDN to map to a 'SKU-Edition' name that is useful in population reporting
IsProtected	This is a calculated field derived from the Spynet Report's AV Products field. Returns: a. TRUE if there is at least one active and up-to-date antivirus product running on this machine
AutoSampleOptIn	This is the SubmitSamplesConsent value passed in from the service, available on CAMP 9
PuaMode	Pua Enabled mode from the service
SMode	This field is set to true when the device is known to be in 'S Mode', as in, Windows 10 S mode, where only Microsoft Store apps can be installed
IeVerIdentifier	NA
SmartScreen	This is the SmartScreen enabled string value from registry. If the value exists but is blank, the value "ExistsNotSet" is sent in telemetry.
Firewall	This attribute is true (1) for Windows 8.1 and above if windows firewall is enabled, as reported by the service.
UacLuaenable	This attribute reports whether or not the "administrator in Admin Approval Mode" user type is disabled or enabled in UAC.
Census_MDC2FormFactor	A grouping based on a combination of Device Census level hardware characteristics. The logic used to define Form Factor is rooted in business and industry standards and aligns with how people think about their device.
Census_DeviceFamily	AKA DeviceClass. Indicates the type of device that an edition of the OS is intended for.
Census_OEMNameIdentifier	NA
Census_OEMModelIdentifier	NA
Census_ProcessorCoreCount	Number of logical cores in the processor
Census_ProcessorManufacturerIdentifier	NA
Census_ProcessorModelIdentifier	NA

Census_ProcessorClass	A classification of processors into high/medium/low. Initially used for Pricing Level SKU. No longer maintained and updated
Census_PrimaryDiskTotalCapacity	Amount of disk space on primary disk of the machine in MB
Census_PrimaryDiskTypeName	Friendly name of Primary Disk Type - HDD or SSD
Census_SystemVolumeTotalCapacity	The size of the partition that the System volume is installed on in MB
Census_HasOpticalDiskDrive	True indicates that the machine has an optical disk drive (CD/DVD)
Census_TotalPhysicalRAM	Retrieves the physical RAM in MB
Census_ChassisTypeName	Retrieves a numeric representation of what type of chassis the machine has. A value of 0 means xx
Census_InternalPrimaryDiagonalDisplaySizeInInches	Retrieves the physical diagonal length in inches of the primary display
Census_InternalPrimaryDisplayResolutionHorizontal	Retrieves the number of pixels in the horizontal direction of the internal display.
Census_InternalPrimaryDisplayResolutionVertical	Retrieves the number of pixels in the vertical direction of the internal display
Census_PowerPlatformRoleName	Indicates the OEM preferred power management profile. This value helps identify the basic form factor of the device
Census_InternalBatteryType	NA
Census_InternalBatteryNumberOfCharges	NA
Census_OSVersion	Numeric OS version Example - 10.0.10130.0
Census_OSArchitecture	Architecture on which the OS is based. Derived from OSVersionFull. Example - amd64
Census_OSBranch	Branch of the OS extracted from the OsVersionFull. Example - OsBranch = fbl_partner_eeap where OsVersion = 6.4.9813.0.amd64fre.fbl_partner_eeap.140810-0005
Census_OSBuildNumber	OS Build number extracted from the OsVersionFull. Example - OsBuildNumber = 10512 or 10240
Census_OSBuildRevision	OS Build revision extracted from the OsVersionFull. Example - OsBuildRevision = 1000 or 16458
Census_OSEdition	Edition of the current OS. Sourced from HKLM\Software\Microsoft\Windows NT\CurrentVersion@EditionID in registry. Example: Enterprise
Census_OSSkuName	OS edition friendly name (currently Windows only)
Census_OSInstallTypeName	Friendly description of what install was used on the machine i.e. clean
Census_OSInstallLanguageIdentifier	NA
Census_OSUILocaleIdentifier	NA
Census_OSWUAutoUpdateOptionsName	Friendly name of the WindowsUpdate auto-update settings on the machine.
Census_IsPortableOperatingSystem	Indicates whether OS is booted up and running via Windows-To-Go on a USB stick.

Census_GenuineStateName	Friendly name of OSGenuineStateID. 0 = Genuine
Census_ActivationChannel	Retail license key or Volume license key for a machine.
Census_IsFlightingInternal	NA
Census_IsFlightsDisabled	Indicates if the machine is participating in flighting.
Census_FlightRing	The ring that the device user would like to receive flights for. This might be different from the ring of the OS which is currently installed if the user changes the ring after getting a flight from a different ring.
Census_ThresholdOptIn	NA
Census_FirmwareManufacturerIdentifier	NA
Census_FirmwareVersionIdentifier	NA
Census_IsSecureBootEnabled	Indicates if Secure Boot mode is enabled.
Census_IsWIMBootEnabled	NA
Census_IsVirtualDevice	Identifies a Virtual Machine (machine learning model)
Census_IsTouchEnabled	Is this a touch device ?
Census_IsPenCapable	Is the device capable of pen input ?
Census_IsAlwaysOnAlwaysConnectedCapable	Retreives information about whether the battery enables the device to be AlwaysOnAlwaysConnected .
Wdft_IsGamer	Indicates whether the device is a gamer device or not based on its hardware combination.
Wdft_RegionIdentifier	NA

The dataset size is 4.7GB which may pose a memory issues while modelling or rendering plots. Also, analytics isn't about just running the algorithms straight out of libraries but about the features we use for prediction. Hence, the following steps are significant before running the model.

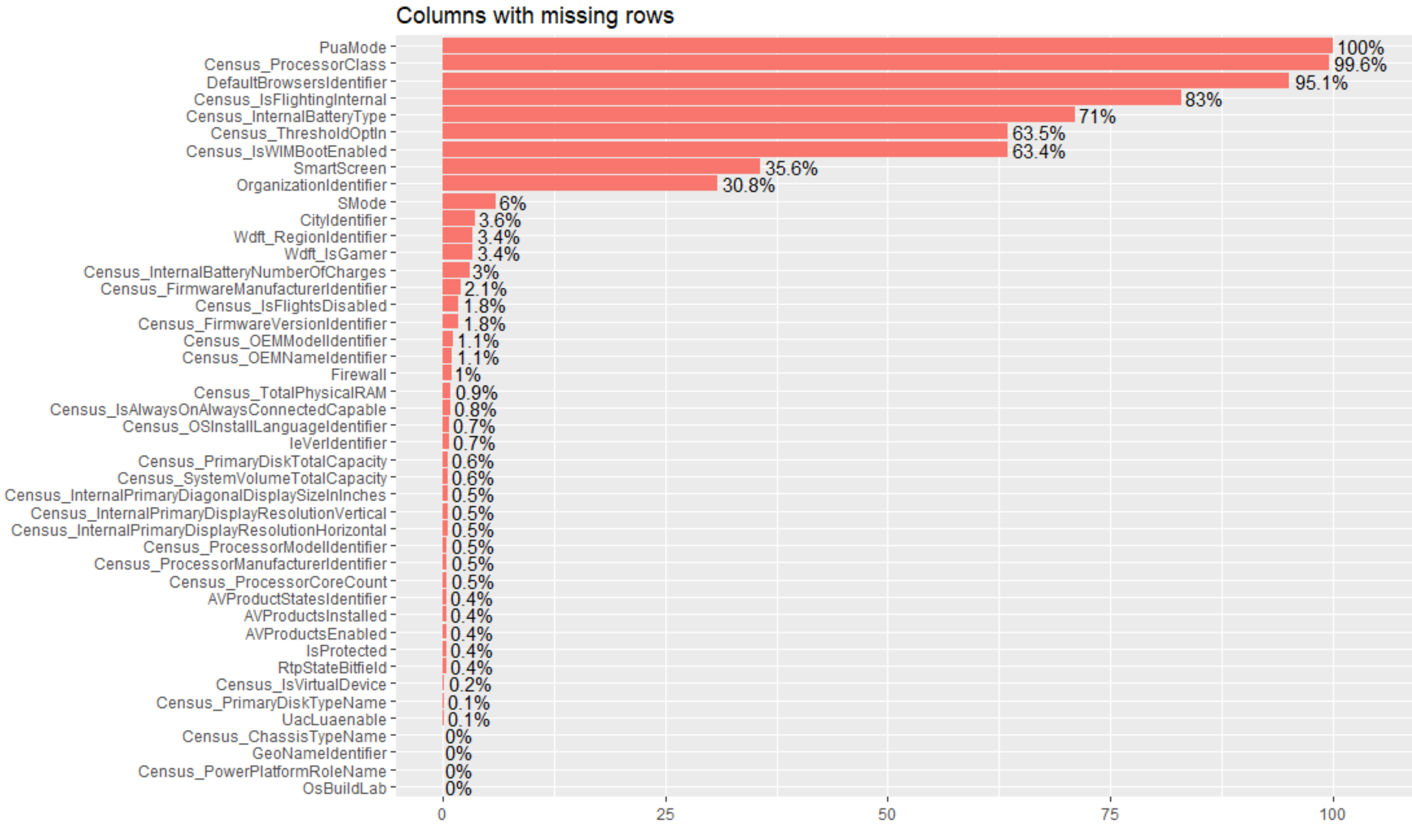
Before starting, check dedicated memory allocation for R processing(default 12173 MB) and increase the limit to 70384 MB.

```
memory.limit()

#Increase memory allocation for R
memory.limit(70384)
```

```
[1] 12173
[1] 70384
```

Inspection of Missing values:



Variables “PuaMode”, “Census_ProcessorClass” and “DefaultBroswersIdentifier” has more than 90% missing records. Imputations for that many records will most likely result in skewing the dataset. Therefore, it would be best to drop these columns altogether. Also, variable “MachineIdentifier” is just an identifier for individual machine and plays no role in modelling either.

```
#Remvoing column with more than 90% missing values and Machine identifier which plays no role in
modeling
mltrain.df <- mltrain.df[,-c('DefaultBrowsersIdentifier','MachineIdentifier', 'PuaMode',
'Census_ProcessorClass')]
```

Inspection of skewed variables:

Skewness is the degree of distortion from symmetrical bell curve or normal distribution. The rule of thumb is to drop the variables if they are 100% skewed. It means, the entire column has the same value throughout all records. In such cases the variable doesn't capture any variation and therefore is totally useless in modelling.

The dataset is huge to be able to perform the skewness analysis at once. Therefore, check skewness for numeric variables first and then for categorical variables.

```
#Fitering only numeric columns
library(microbenchmark)
numcols<-Filter(is.numeric,mltrain.df)


#Create empty dataframe to check the distinct value of each numeric column
x= data.frame("Variable" = character(), 'Distinct' = integer(), stringsAsFactors = FALSE)
```

Create an empty data frame "x" to capture the number of unique elements in variables.

Also, it is mandatory to convert data table to data frame. Data tables doesn't support named column indexes. For instance accessing data table variable using mltrain.df[, i] gives an error where 'i' may equal any integer.

```
#Convert from Data table to dataframe {named column index doesnt work in data table}
mltrain.df<-as.data.frame(mltrain.df)
numcols<-as.data.frame(numcols)
```

Now, fill the empty data frame with variable and its corresponding unique values.

```
#Fill Empty data frame
for (k in 1:52)
{
  x[k,1]<-names(numcols[k])
  x[k,2]<-length(unique(numcols[,k]))
}
```

Several variables in the data frame have more than 10 unique values and do not pose a threat of absolute positive or negative skewness. Filter only the variables with less than 5 unique values and check for skewness.

```
#Columns with less than 5 unique values
z=x$Variable[x$Distinct<5]
```

Calculate and print if the variables inside variable 'Z' are skewed.

```r
#Print if these columns are skewed towards one value
library(formattable)

for (i in z)
{
  print(i)
  print(format(round(prop.table(table(numcols[,i]))*100),2))
  print('-------------------\n')
}
```

```
[1] "IsBeta"

    0     1 
"100"   "0" 
[1] "-------------------\n"
[1] "IsSxsPassiveMode"

   0    1 
"98"  "2" 
[1] "-------------------\n"
[1] "HasTpm"

   0    1 
 "1" "99" 
[1] "-------------------\n"
[1] "IsProtected"

   0    1 
 "5" "95" 
[1] "-------------------\n"
[1] "AutoSampleOptIn"
```

[1] "AutoSampleOptIn"

```
    0     1 
"100"   "0" 
[1] "-------------------\n"
[1] "SMode"

    0     1 
"100"   "0" 
[1] "-------------------\n"
[1] "Firewall"

   0    1 
 "2" "98" 
[1] "-------------------\n"
[1] "Census_HasOpticalDiskDrive"

   0    1 
"92"  "8" 
[1] "-------------------\n"
[1] "Census_IsPortableOperatingSystem"

    0     1 
"100"   "0" 
[1] "-------------------\n"
[1] "Census_IsFlightingInternal"

    0     1 
"100"   "0" 
[1] "-------------------\n"
[1] "Census_IsFlightsDisabled"

    0     1 
"100"   "0" 
```

```
[1] "Census_ThresholdOptIn"

    0     1 
"100"   "0" 
[1] "--------------------\n"
[1] "Census_IsSecureBootEnabled"

   0    1 
"51" "49" 
[1] "--------------------\n"
[1] "Census_IsWIMBootEnabled"

    0     1 
"100"   "0" 
[1] "--------------------\n"
[1] "Census_IsVirtualDevice"

   0    1 
"99"  "1" 
[1] "--------------------\n"
[1] "Census_IsTouchEnabled"

   0    1 
"87" "13" 
[1] "--------------------\n"
[1] "Census_IsPenCapable"

   0    1 
"96"  "4" 
[1] "--------------------\n"
[1] "Census_IsAlwaysOnAlwaysConnectedCapable"

   0    1 
"94"  "6" 
```

```
[1] "Wdft_IsGamer"

   0    1 
"72" "28" 
[1] "--------------------\n"
[1] "HasDetections"

   0    1 
"50" "50" 
[1] "--------------------\n"
```

Following variables have 100% skewed values and therefore play no role in Modelling:

IsBeta, AutoSampleOptIn, SMode, Census_IsPortableOperatingSystem, Census_ISFlightingInternal, Census_IsFlightsDisabled, Census_ThresholdOptIn, Census_IsWIMBootEnabled

```
#Remove skewed numeric columns
mltrain.df<-subset(mltrain.df, select = -c(IsBeta,
AutoSampleOptIn,SMode,Census_IsPortableOperatingSystem,Census_IsFlightingInternal,Census_IsFlightsDisabl
ed,Census_ThresholdOptIn,Census_IsWIMBootEnabled))
```

Now, checking the skewness for categorical variables of dataset.

```
#Filter only character columns
charcols <- Filter(is.character,mltrain.df)

#Create empty dataframe to check the distinct value of each numeric column
x= data.frame("Variable" = character(), 'Distinct' = integer(), stringsAsFactors = FALSE)

#Convert from Data table to dataframe {named column index doesnt work in data table}
charcols<-as.data.frame(charcols)

for (k in 1:27)
{
  x[k,1]<-names(charcols[k])
  x[k,2]<-length(unique(charcols[,k]))
}


z=x$Variable[x$Distinct<=5]
```

```
#Print if these columns are skewed towards one value
for (i in z)
{
  print(i)
  print(format(round(prop.table(table(charcols[,i]))*100),2))
  print('------------------\n')
}
```

```
[1] "Platform"

  windows10 windows2016    windows7    windows8 
       "97"         "0"         "1"         "2" 
[1] "------------------\n"
[1] "Processor"

arm64   x64   x86 
  "0"  "91"   "9" 
[1] "------------------\n"
[1] "Census_DeviceFamily"

        Windows Windows.Desktop  Windows.Server 
            "0"           "100"             "0" 
[1] "------------------\n"
[1] "Census_PrimaryDiskTypeName"

        HDD         SSD     UNKNOWN Unspecified 
       "65"        "28"         "4"         "3" 
[1] "------------------\n"
[1] "Census_OSArchitecture"

amd64 arm64   x86 
 "91"   "0"   "9" 
```

Variable Census_DeviceFamily is 100% skewed.

```
#Remove skewed character columns
mltrain.df<-subset(mltrain.df, select = -c(Census_DeviceFamily))
```

NULL Value Imputation :

Remaining NULL values in the dataset are imputed using Mode. Mode is used because the variables of dataset are all discrete factors and not continuous.

First, filtered out the variables with missing values and stored it in data frame 'miss'. Created two Mode functions, one for the variables whose mode is not null and another for the variables whose mode is Null.

```
#Shape of the dataframe
shape = as.array(dim(mltrain.df))

#Find columns with highest missing values in percent
miss<-(sort((colSums(is.na(mltrain.df))/shape[1])*100, decreasing=TRUE))
miss<-data.frame(dimnames(as.array(miss[miss>0])))

miss$c..Census_InternalBatteryType....SmartScreen....OrganizationIdentifier...<-as.character(miss$c..Cen
sus_InternalBatteryType....SmartScreen....OrganizationIdentifier...)


#Mode Function for imputation
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

#Mode Function when mode of variable is Null
Modena<-function(x){
  ux <- unique(na.omit(x))
  ux[which.max(tabulate(match(x, ux)))]
}
```

Ran for-loop to impute null values with mode depending upon whether the mode is null or not-null.

```
#Na Value Imputations
for (i in 1:nrow(miss))
{
  if (is.na(Mode(mltrain.df[,miss[i,1]])))
    {
    mltrain.df[,miss[i,1]][is.na(mltrain.df[,miss[i,1]])] <- Modena(mltrain.df[,miss[i,1]])

  }
  else
  {
    mltrain.df[,miss[i,1]][is.na(mltrain.df[,miss[i,1]])]<-Mode(mltrain.df[,miss[i,1]])
  }
}

remove(miss)
```

Inspect 100% Correlated Variable Pairs:

Highly correlated variable pairs essentially provide the same information or variation for modelling. It is safe to remove one variable out of the correlated pairs without loss of information or variation.

To find Correlation amongst all variables, convert character variables to numeric variables.

```
numcols<-Filter(is.numeric,mltrain.df)
charcols <- Filter(is.character,mltrain.df)


#Convert Character columns to Factors
charcols[sapply(charcols, is.character)] <- lapply(charcols[sapply(charcols, is.character)], as.factor)

#Convert Factor Columns to numeric
charnumcols <- charcols %>% mutate_if(is.factor, as.numeric)


remove(charcols)

#Dataframe to calculate correlation
train <- cbind(charnumcols,numcols)

remove(numcols)
remove(charnumcols)
```

Calculated correlation between every possible variable pair combination. Filtered the list of variable pairs having more than 95% correlation. Variable pairs with more than 99.99% correlation are (OsVer-Platform), (Census_OSSkuName - Census_OSEdition), (Census_OSArchitecture - Processor).

It is safe to drop one variable each from aforementioned pairs.

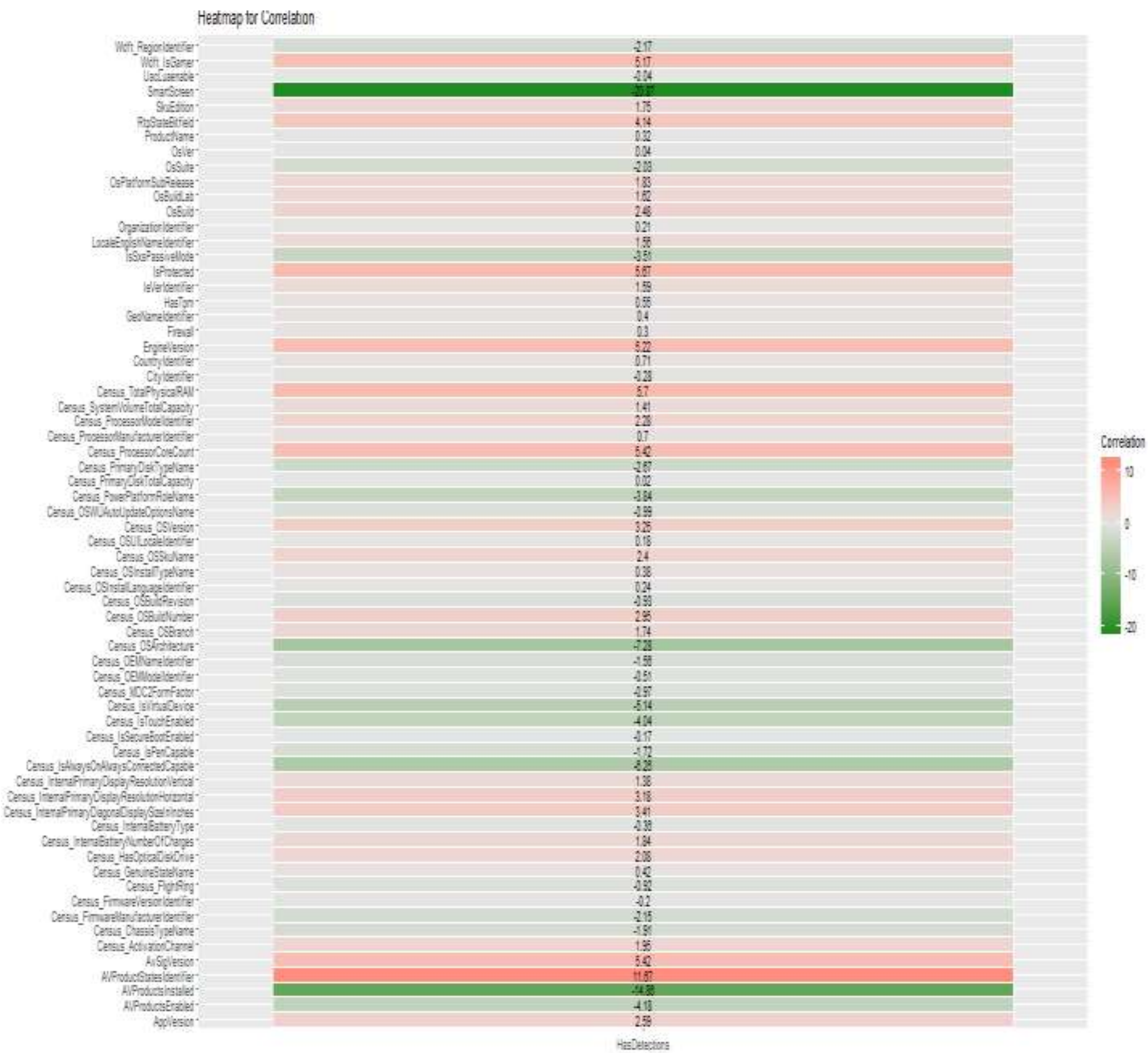
```
#find out highly correlated columns and eventually remove one from the pair having more than 99%
correlation
library(tidyr)
library(tibble)

cormat <- train %>% as.matrix %>% cor %>% abs %>% as.data.frame %>% rownames_to_column(var = 'var1') %>%
gather(var2, value, -var1)

corrmat<-cormat[order(-cormat$value),]

remove(cormat)
#remove(train)

#More than 90 correlation
dfcor<-corrmat[which(corrmat$value>0.95),]
remove(corrmat)

#Approximately 100% Correlated pairs
#(OsVer-Platform),(Census_OSSkuName-Census_OSEdition), (Census_OSArchitecture-Processor)

mltrain.df<-subset(mltrain.df, select = -c(Platform, Census_OSEdition, Processor))
remove(dfcor)
train<- subset(train, select = -c(Platform, Census_OSEdition, Processor))
```

Using feature engineering the data size has been reduce from 4.7 GB to 3.1 GB. Number of variables left are 67.

```
#Print reduced dataset Size
print(object.size(mltrain.df), units='auto')
```

```
3.1 Gb
```

Heatmap to check correlation with target variable:



The maximum correlation of approximately 20% of Malware detection is with variable SmartScreen. The data doesn't seem to provide any strong connection between malware detection and the features of machine.

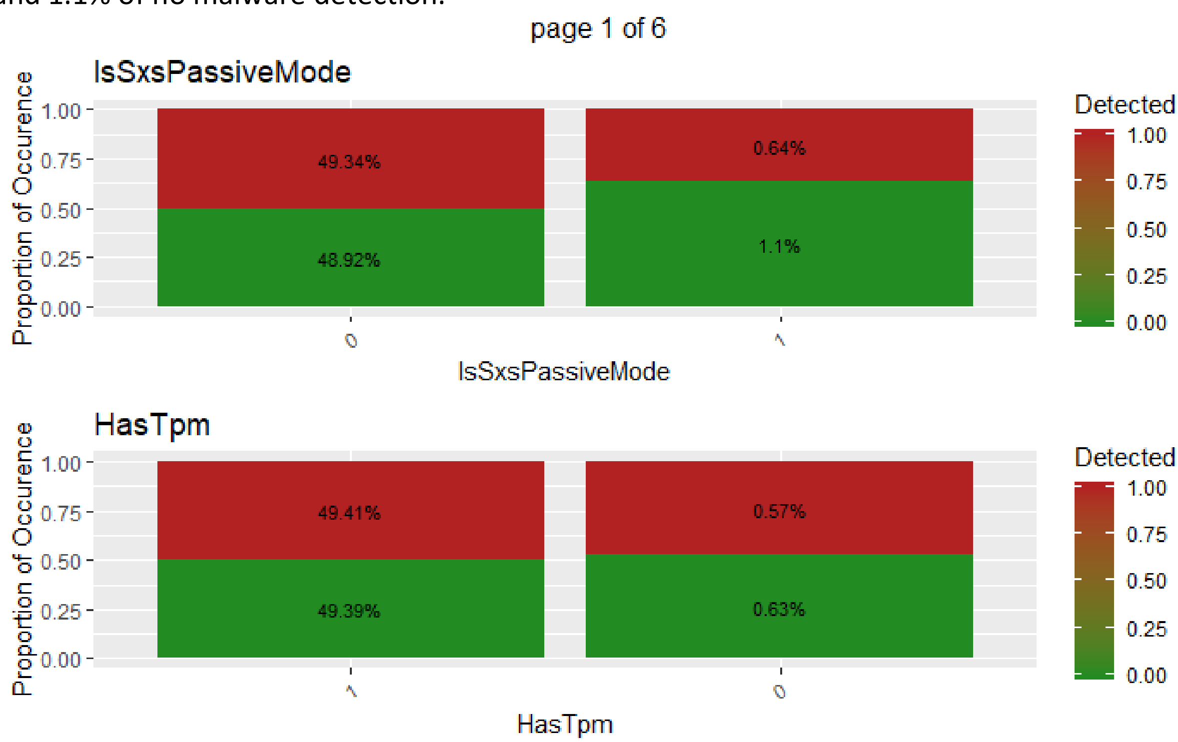
Distribution of "HasDetection" in variables:

Restricted the analysis to variables with less than 11 unique factors.

```
mat1<-vector()

for (i in colnames(mltrain.df))
  {
  if (length(unique(mltrain.df[,i]))<11)
  {
    mat1<-append(mat1,i)

  }

}

#Create new data frame with variables having less than 11 unique factors
dt <- data.frame(mltrain.df[,mat1])
```

Binary Columns:

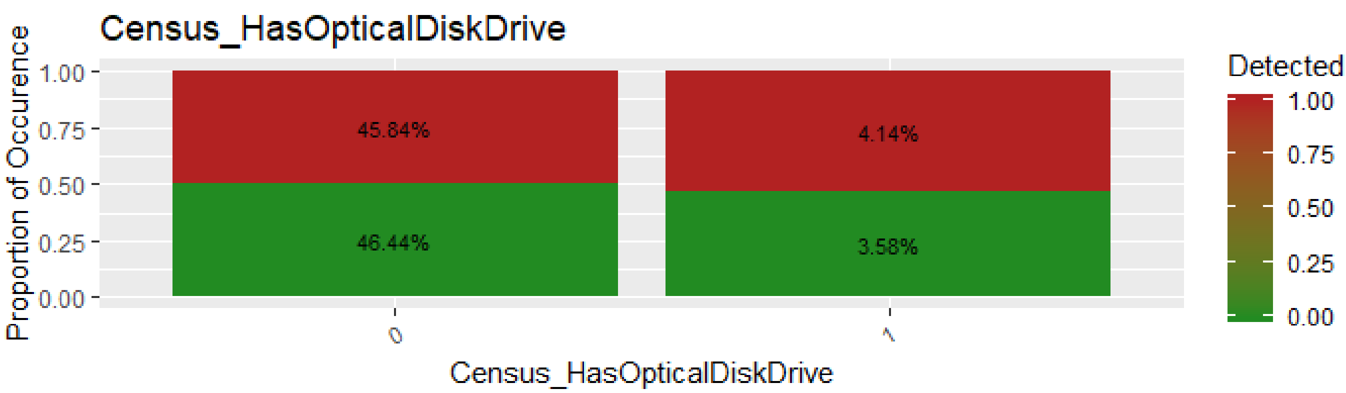
From the below graph, for "IsSxsPAssiveMode" as 0, there are 49.34% of Malware detection and 48.92% of no malware detection. Whereas for "IsSxsPassiveMode" as 1, there is only 0.64% of Malware detection and 1.1% of no malware detection.



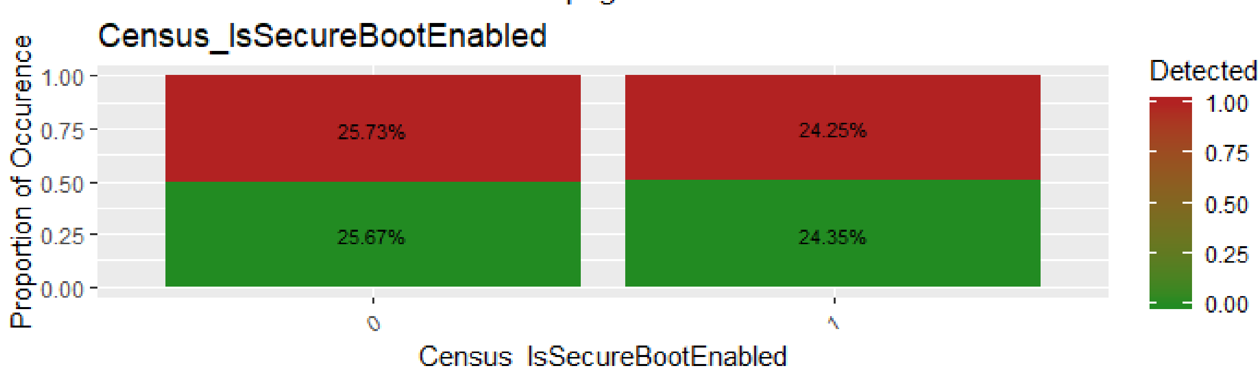
The peculiar thing to observe in the below graph is even though there is some sort of protection active and updated antivirus installed on the machine, that's where the greatest number of malware attacks happened.

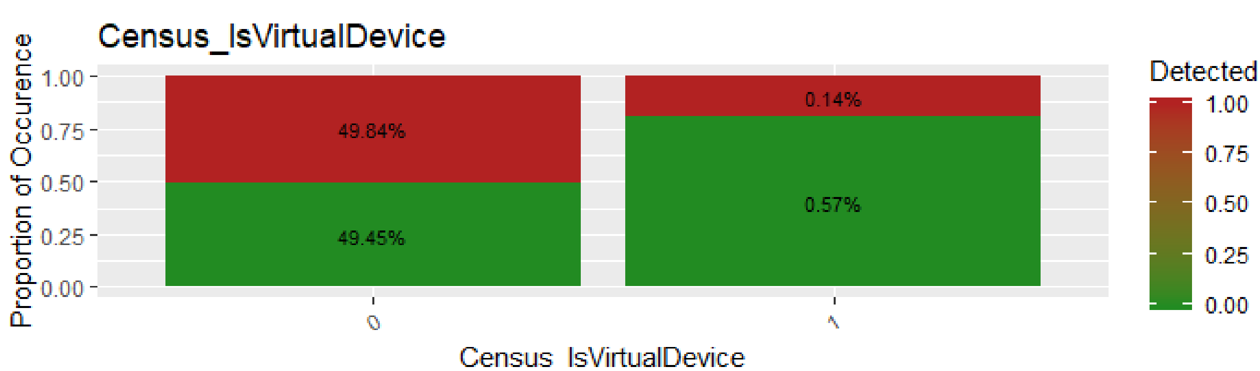
When the "IsProtected" value is 1(installed antivirus), machines experienced 47.91% of malware detection and 46.67% no detection.

page 2 of 6

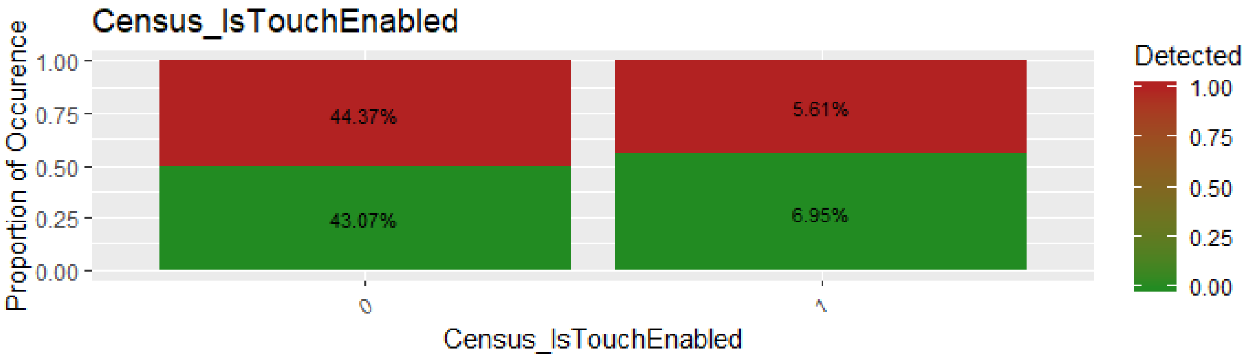


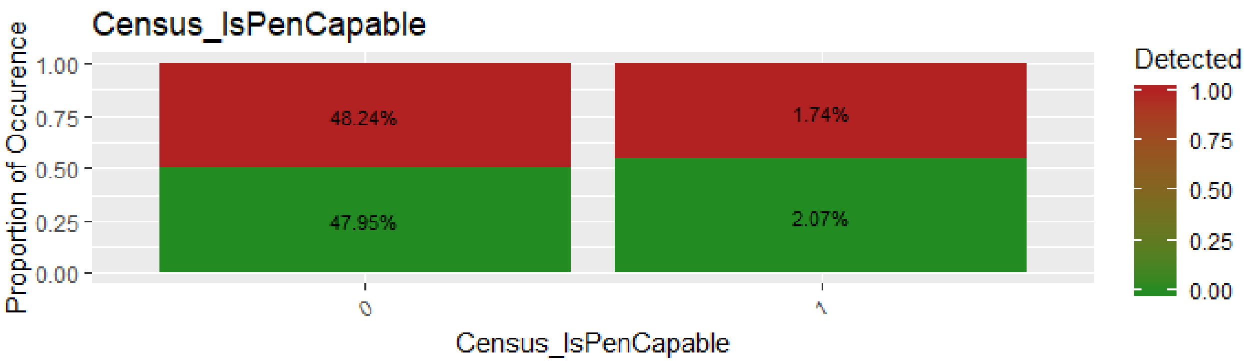
page 3 of 6



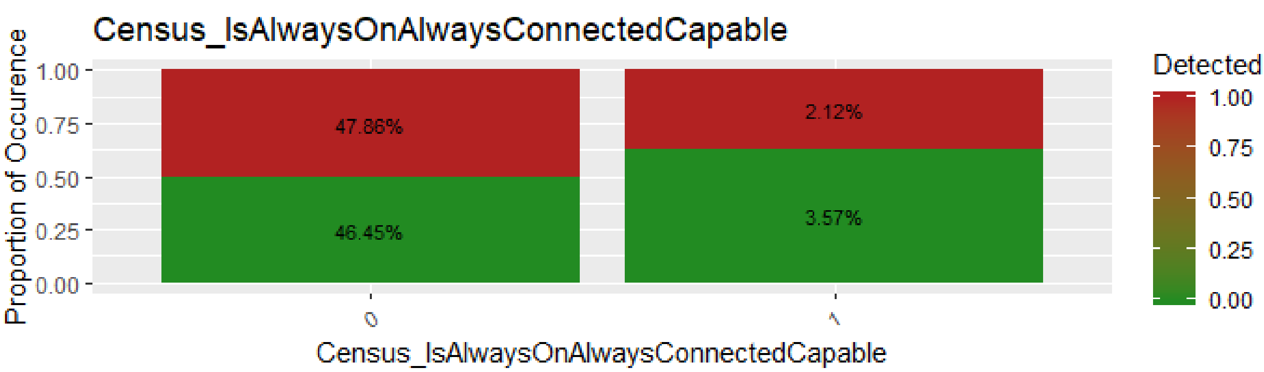


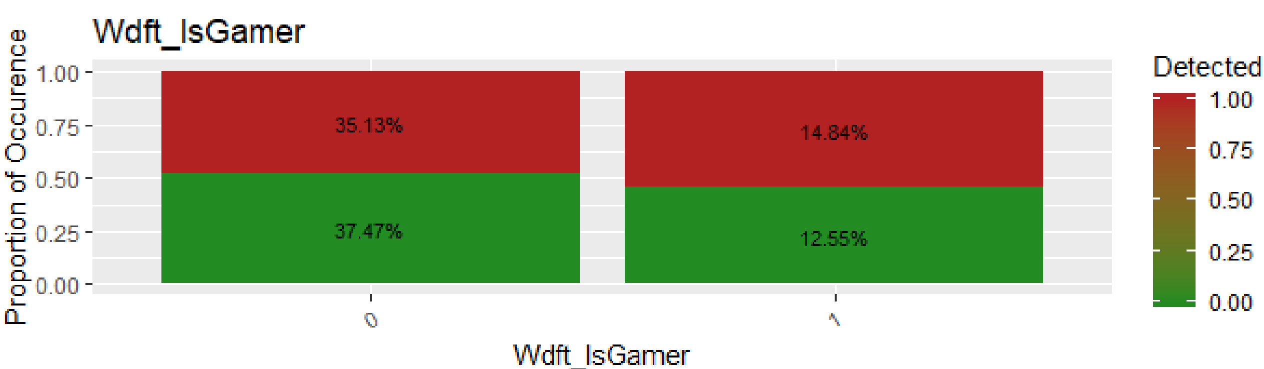
page 4 of 6

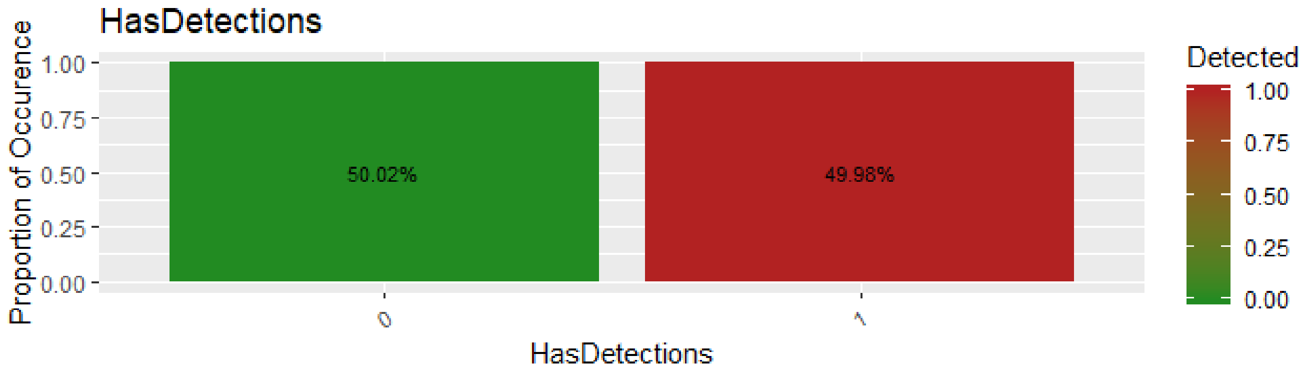




page 5 of 6



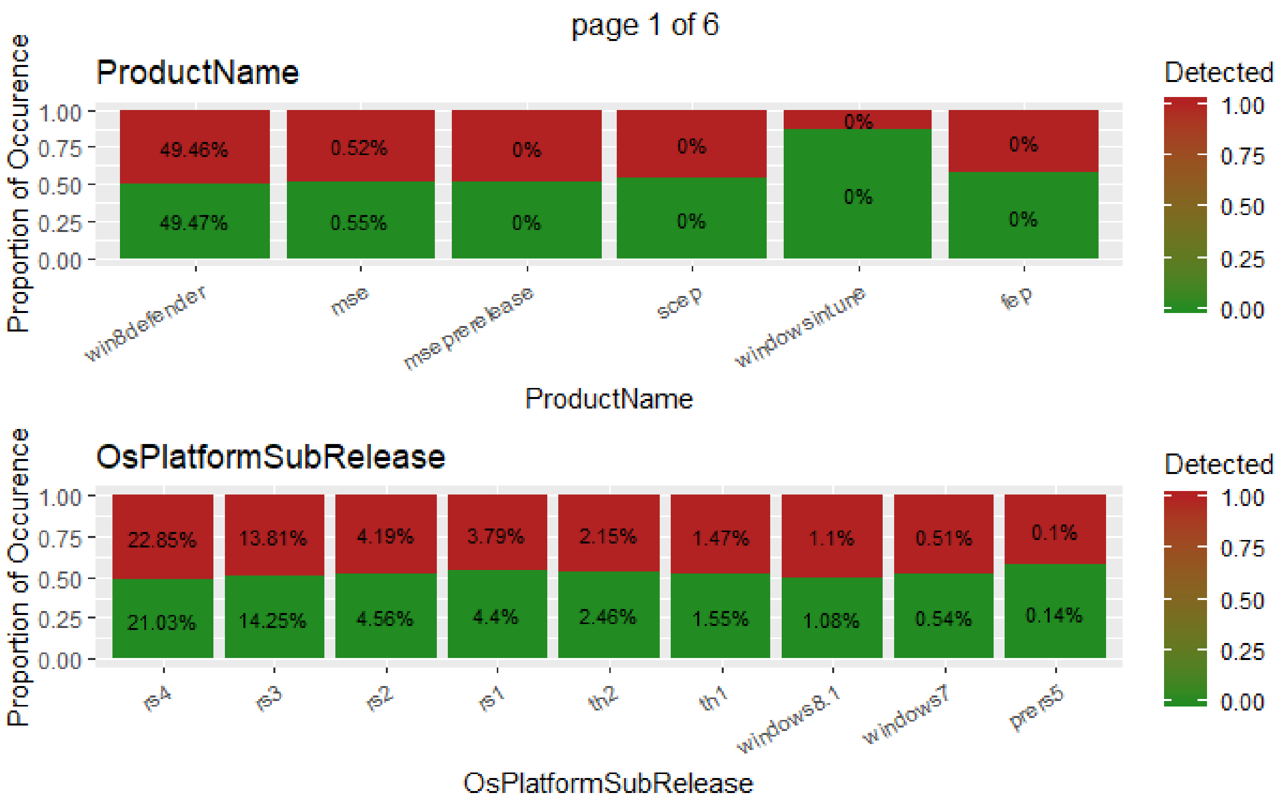


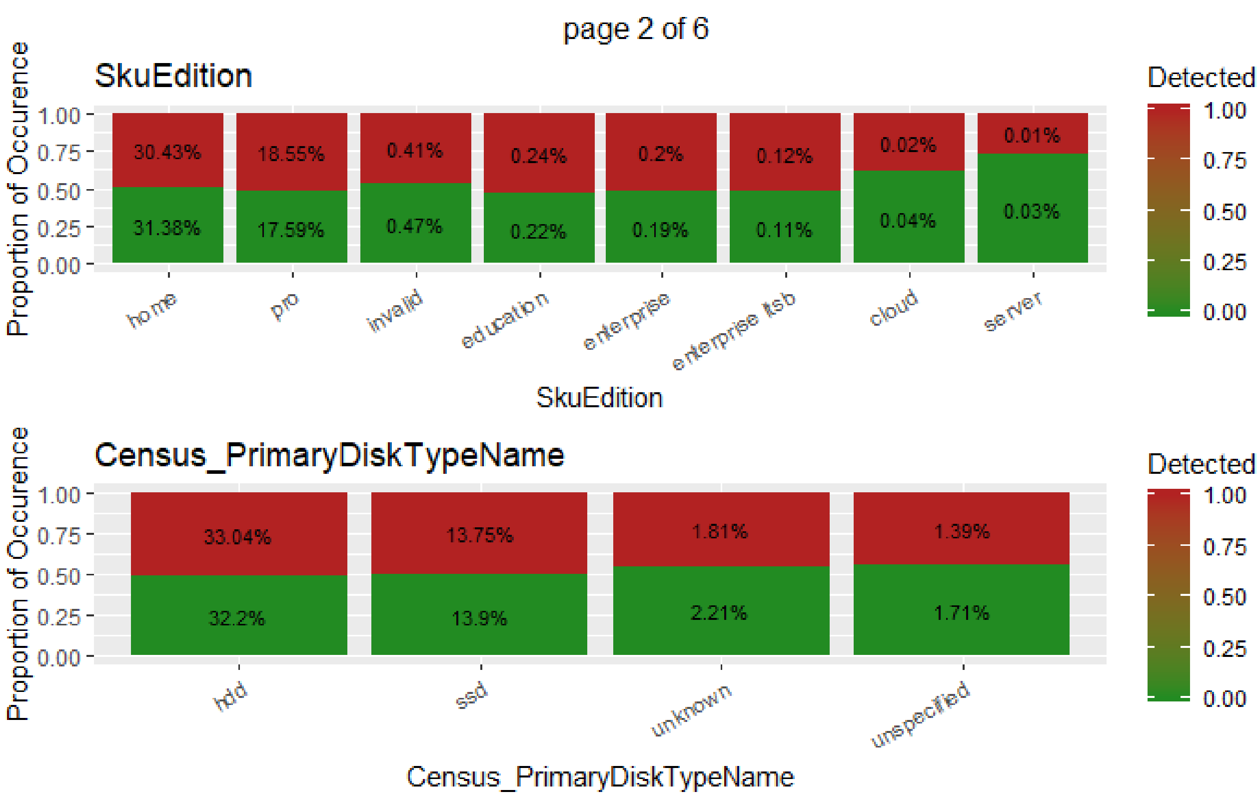

HasDetections
Proportion of Occurence
1.00
0.75
0.50
0.25
0.00
50.02%
49.98%
0
1
HasDetections
Detected
1.00
0.75
0.50
0.25
0.00


Firewall
Proportion of Occurence
1.00
0.75
0.50
0.25
0.00
48.94%
48.94%
1.04%
1.08%
1
0
Firewall
Detected
1.00
0.75
0.50
0.25
0.00

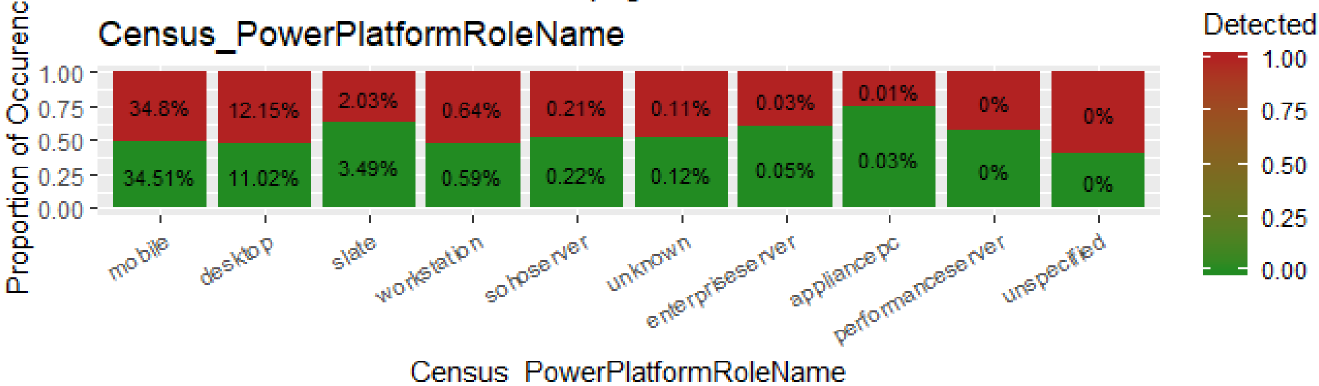
Character and Identifier Variables:

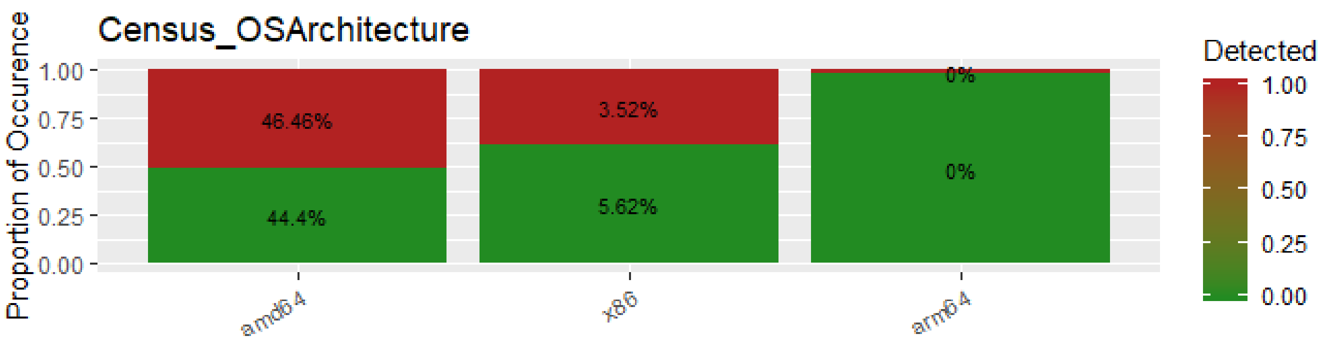
Note: The 0% in some bars in below graph are only approximate percent proportion. It signifies that the number of occurrence of ”windowsintune” is very low (lower than 0.001% of total occurrence).



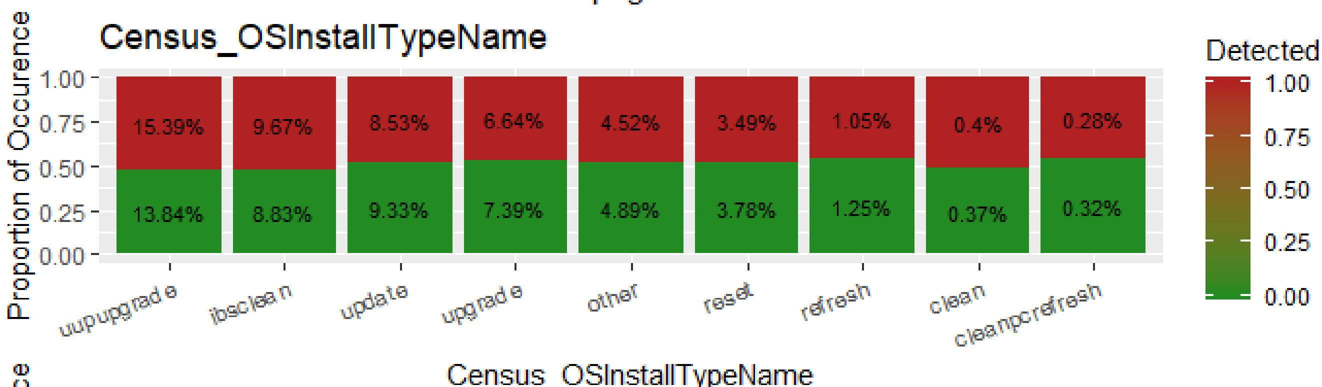


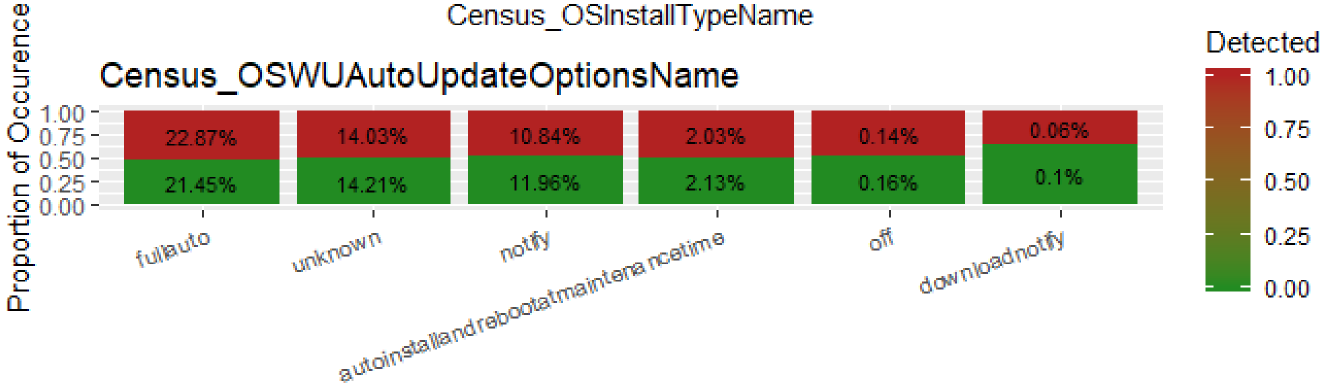
page 3 of 6



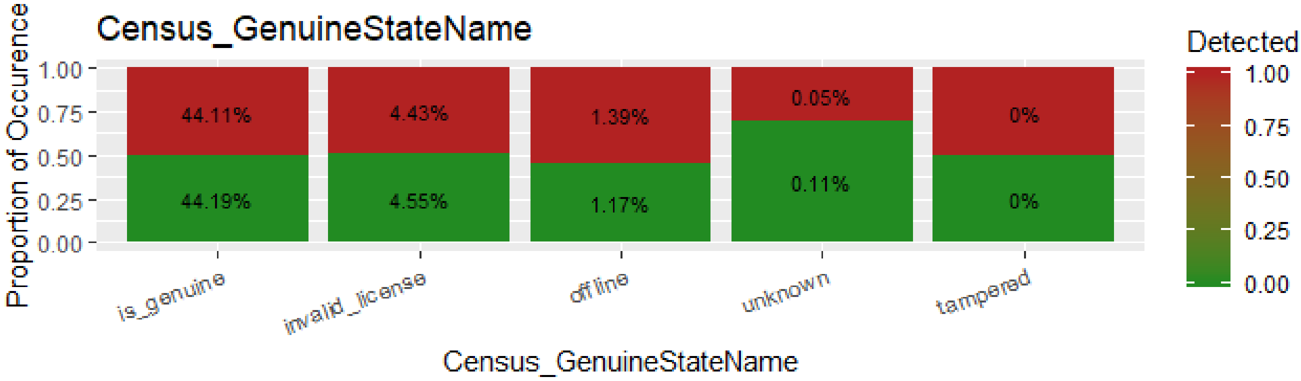


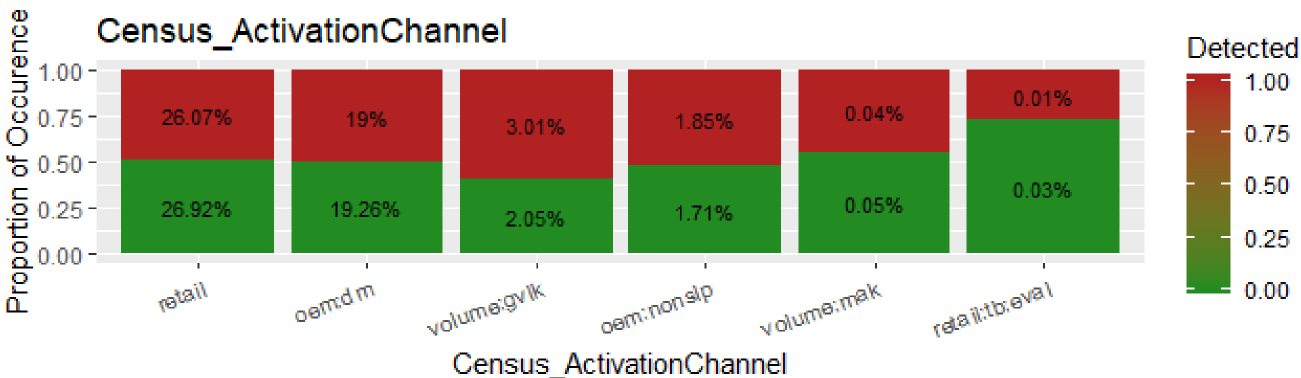
page 4 of 6



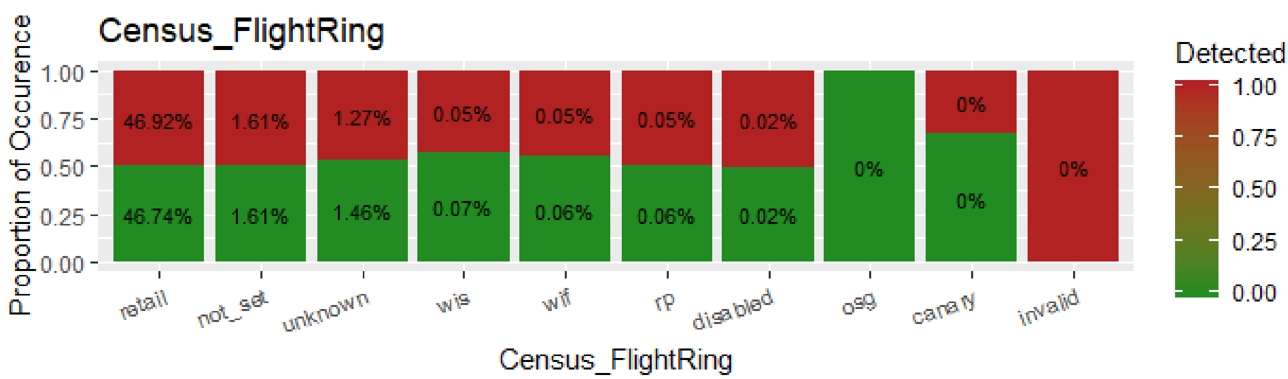


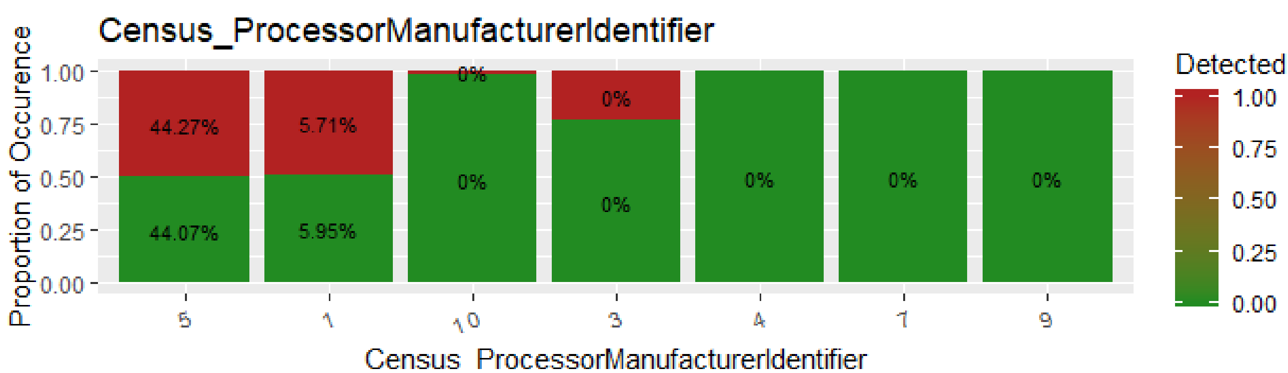
page 5 of 6





page 6 of 6





Interpretation:-

Looking at the Malware detection distribution in binary, character and Identifier columns, it is evident that the occurences are fairly random and each value of the variable has approximately uniform distribution of detection vs non-detection. It is important not to be confused by high percentage occurrence of "HasDetection values" in some of the bars(factor of variable). For example, "IsProtected" variable has two values 1 and 0. Most of "HasDetection" values fall inside 1(protectected machines). This is only because the maximum proportion of machines in the dataset are protected by some kind of antivirus. It is important to notice that within 1 of "IsProtected" the number of malware detections are approximately equal to non-detections. The same is the case with almost all the variables Therefore, these bar graphs right off the bat do not give us much reason as to why the machines get hit by malwares. We hope to achieve better results in modelling.

MODELLING

Since the dataset is enormous, for the purpose of this project report and keeping in mind the limitations of this computer, the training dataset is randomly sampled at 25% percent records. The number of records left are approximately 2.3 million. However, reducing the data size for training affected the accuracy. Accuracy on 9 million records was 65% and at 2.3 million records dropped to 63%.

This is a binary classification problem and for the purpose of predicting binary outcome of whether a machine will be hit by malware, decision tree methodology seems like the way to go. Decision trees are a method of splitting the data based on variables to either classify or predict some value. Each branch in decision tree divides the data into two groups. Decision trees are flexible and easy to understand. However, a single decision tree may overfit and is unlikely to generalize well. There are hyper parameters to restrict the tree from overfitting such as maximum depth but those might cause the tree to underfit as well. This is the reason why, generally it might not be the best idea to use a single decision tree. Instead, multiple decision trees are used together. Gradient boosting decision trees are one of the many methods that combine the predictions of many trees to make predictions that generalize well.

Gradient boosting decision tree finds the best split for each leaf by minimizing the cost function. This step requires algorithm to go to every feature of every data point. Thus, computational complexity increases exponentially with increasing features and data records.

LightGBM :

As the name goes, LightGBM isn’t as computationally complex as conventional gradient boosting decision trees. LightGBM is faster, takes lower memory to run and handles large data size efficiently. The computational complexity to build a tree is proportional to the number of splits that must be evaluated. Generally, small changes in the split do not make much of a difference in tree performance. LightGBM uses histogram-based method to take advantage of this fact by grouping features into a set of bins and perform splitting on bins instead of individual features. Since the features are binned before building the tree, it greatly speeds up the training process.

Since boosting type used is ‘gbdt’(gradient boosting decision trees). There is no need of cross validation because GBDTs make good generalization by combining the results of multiple decision trees. These reasons make LightGBM the perfect algorithm for this problem.

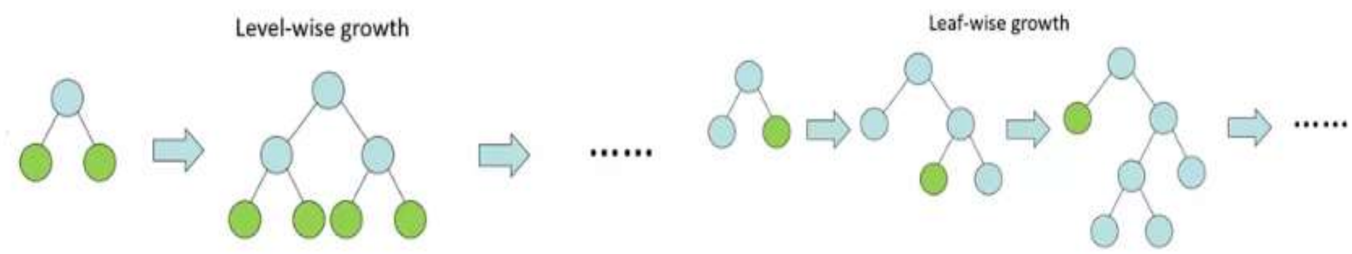
Hyperparameters: Selected the following parameters based on multiple runs that provided the best accuracy.

```
params = list(
  boosting_type = 'gbdt',
  objective = 'binary',
  metric = 'auc',
  nthread = 4,
  learning_rate = 0.05,
  max_depth = 5,
  num_leaves = 40,
  sub_feature = 0.1*9,
  sub_row = 0.1*9,
  bagging_freq = 1,
  lambda_l1 = 0.1,
  lambda_l2 = 0.1,
  verbosity =-1
)
```

Train the model based on parameter list

```
mod <- lgb.train(params = params,
                 nrounds = 4000,
                 data = x_train

)
```

While training the data, conventional decision trees and gradient boosting decision trees use level-wise growth strategy whereas LightGBM uses leaf-wise growth strategy. Leaf-wise growth strategy is more flexible and splits the leaf that reduces the loss most. This makes it a better choice for large datasets.



Create test matrix for predictions:

```
rm(x_train)
gc()

x_test <- data.matrix(valid.df)
```

Make Predictions on the test matrix and evaluate accuracy of the model.

```
#Make Predictions
pred <- predict(mod,data = x_test)

##Set decision boundary
pred.scale <- ifelse( pred >= 0.4, 1,0)

#Set Actuals and Precited to Factors
pred.scale<-as.factor(pred.scale)
valid.df$HasDetections<-as.factor(valid.df$HasDetections)

#Confusion Matrix
confusionMatrix(pred.scale, valid.df$HasDetections, positive = '1')
```

RESULT

Confusion Matrix yields the accuracy of 63%. The class of interest here is the machines that got hit by malware . Sensitivity of the model is 66%. Therefore, the model was able to predict 66% of the machines correctly that got hit by malware.

```
Confusion Matrix and Statistics

          Reference
Prediction      0      1
         0 180379 106334
         1 131800 205990

               Accuracy : 0.6187
                 95% CI : (0.6175, 0.6199)
    No Information Rate : 0.5001
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2374
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6595
            Specificity : 0.5778
         Pos Pred Value : 0.6098
         Neg Pred Value : 0.6291
             Prevalence : 0.5001
         Detection Rate : 0.3298
   Detection Prevalence : 0.5409
      Balanced Accuracy : 0.6187

       'Positive' Class : 1
```

SUMMARY

As seen from the malware presence distribution among variables, the presence of malware was approximately uniformly distributed inside within variable classes. Also, the heatmap rendered the maximum of 20% correlation between “SmartScreen” and Malware detection signifying that presence of malware isn’t strongly correlated with any of the 83 variables of the dataset. Keeping this in mind, LightGBM did a fairly good job by attaining sensitivity of 66% in correctly predicting the machine vulnerable to malware.

Note: Lessening the data size while modelling for the purpose of reproducing results on this computer hampered the accuracy statistics by 3 to 4%. The same steps and algorithm on full data fetched comparatively better results on Kaggle kernel.

REFERENCES & CITATIONS

Website

Kaggle Competitions. https://www.kaggle.com/c/microsoft-malware-prediction

GitHub. https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html

Machine Learning Explained. http://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/

A Medium Corporation. https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc