4-22-2021

Android Malware detection using predictive analytics.

Amira Mohammed Albalooshi
ama6472@rit.edu

Android Malware detection using predictive analytics.

by

Amira Mohammed Albalooshi

A Capstone Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Professional Studies:

Data Analytics

Department of Graduate Programs & Research

Rochester Institute of Technology

RIT Dubai

22-04 2021

RIT

Master of Science in Professional Studies:

City Science or Data Analytics

Graduate Capstone Approval

Student Name**: Amira Mohammed Abdulla Albalooshi**

Graduate Capstone Title**: Android Malware detection using predictive analytics**

Graduate Capstone Committee:

Name: Dr. Sanjay Modak **Date:**

Chair of committee

Name: Dr. Boutheina Tlili **Date:**

Member of committee/Mentor

Acknowledgements

I would like to express my appreciation to lots of people who helped me reaching here. Starting with my colleagues who were always there for support, my colleagues at my workplace who inspired me to start this journey, and special thanks to my family for their support and encouragement during this journey.

This paper would not be possible without my mentors' support, Dr. Sanjay Modak and Dr. Boutheina Tilili who were always there to support and guide me. Thank you to all of you.

Table of Contents

Abstract

The growth of android applications is causing a threat and a serious issue towards Android's security. The number of malware targeting the Android operating system is increasing daily. As a result, in recent days the traditional ways that are being used to detect malware are not able to defend alone against the rapid development of hackers attacking techniques and novel malware. This capstone project focuses on using predictive analytics toward detecting malware from the network traffic. In this capstone project, we aim to train and test our data to find the best machine learning model with the highest accuracy of detecting malware in the network traffic. Through a variety of machine learning algorithms and models, we focused on 5 models starting with the logistic regression that was successfully able to predict malware by 67%. Moving to the decision tree that was effectively able to predict malware by 69% which was exactly equal to the random forest prediction ability. The AdaBoost came about 84% exactness, and KNN came with the highest anticipation of 86% between all the models. This shows us the advantage of adopting predictive analytics in malware detection within the traditional approaches to build a strong and defendable Android operating system against malware.

1. Statement of the Problem

The Android operating system has become a glamorous and attractive target for cybercriminals. Although Android is applying its security mechanisms, it is still facing security threats because of the vulnerabilities it is having. This is due to the open-source operating system and multiple stores that applications can be downloaded from. For this, since the device is always connected to the internet, it makes it easier and more vulnerable to multiple sorts of malicious attacks. Although Google and anti-virus companies are making huge efforts into implementing the latest technologies to defend against malware targeting Android, it is still challenging to defend 100% successfully. Hackers are always one step ahead. Thereby, using predictive analytics might be used to assist with understanding the behavior of malware within the network traffic and either block it or notify the end user before it reaches the device and starts its malicious behavior by spreading into the device, collect information, spy on the user, etc.

2. Background of the Problem

With the widespread of smartphones, over one million Android applications around the world such as social media applications, mobile banking applications, and others are used and continue to play an increasingly important role in our everyday lives. Most of these apps have access to personal details of the users, such as their names, location, credit card numbers, and contact information. Almost all apps access the private data of the users, as this offers better-customized services for users. It could also result in lots of visible and non-visible issues. It is important to note that Android Operating System is Linux-based, which means that Linux operates its core functions (S.Sharma, 2014). The OS is mainly built for mobile devices and receives input through touch actions. It is considered as the popular open-source platform worldwide, allowing anyone with the skills required to modify and re-distribute its versions, and the ability to download from official and non-official stores and third-party apps.

Previously, attackers' main target was to attack computers to get some information. However, with the mobile era, the target has been moved or distributed between both computers and smartphones. With the use of mobile phones, people are holding a lot of data in their hands which is more valuable to the attacker to get. Therefore, targeting the mobile device is much more valuable than it was ever. Malware targets devices in different ways such as advertisements, scams, and spying, etc. The infected devices can misbehave or perform activities the owner has not authorized, which can come as a great inconvenience. Some cases have shown malware soliciting ransom payments from the users, usually after encrypting and blocking access to the device (T. Yang Y.Yang K.Qian, 2015). Other cases might be impossible to remove, as they embed themselves deep into the system. Security companies have created antivirus and anti-malware programs that help weed out the malware. However, the traditional ways of tackling malware are becoming harder than before because of the enormous growth of applications and the new mutations of the malware.

The traditional way of detecting android malware has become time-consuming and subject to human error. Static and dynamic tests of detecting android malware are effective but might not be efficient as the platform becomes popular and new applications are being downloaded by the user or uploaded to the store every day. Adopting new technologies is important to keep pace with malware mutations. To defend against Android malware, adopting new technologies is highly recommended to be proactive towards defending against different types of malware based on previous studies related to machine learning and the different models applied by researchers toward finding the best model that replaces the traditional methods or to integrate with it to build a strong system that detects malware while moving in the network before it reaches to the android operating system.

3. Project Definition and Goals

The main goal of this proposed project is to build a model that can detect Android malware using predictive analytics. In this project, we aim to use previous data on android network traffic and train it on different machine learning algorithms to find the best model that can detect malware in the network traffic by analyzing its behavior.

The predictive analytics includes acquiring android malware and its infection from a related dataset. The information will go through phases of cleaning and organizing to decrease the likelihood of the model making mistakes in future data. Using a training dataset about how the malware acts, the exercise will then turn towards developing the research model. The training phase is critical, as the entire models rely on their high accuracy. For this, we will be using Google Colab to apply data cleansing, exploratory data analysis, feature selection, data processing, and several machine learning algorithms such as logistic regression, decision tree, random forest, AdaBoost, and ANN model to find the best model that is having successful high predictions of malware traffic.

4. Literature Review

Android is a platform that has progressively grown to be the largest smartphone operating system by the number of devices. These many devices are becoming a target of attackers seeking to obtain the personal information in them. Detecting Android malware is still in its infancy stages but is picking up quickly to find better ways. For instance, (Zhu, Jin, Yang, Wu, & Chen, 2017) outlines a method of detecting malware by deep learning methods. Machine learning algorithms, in this case, look for differences between the flow of data. The resulting tool is called Deep Flow, which analyzes the information by extracting it from running applications (Zhu, Jin, Yang, Wu, & Chen, 2017). The analytics can predict whether the application is a possible malware since malicious cases can be looking for information in the wrong places. The researcher builds the model with thousands of applications, allowing the algorithm to understand what to look for in malware. The results indicated that Deep Flow is much better at classifying the malware from normal applications, with a chance to improve and perform much better. The research concludes that the typical way of looking for malware through their signatures is currently ineffective, as the dynamic nature of the tools sails through the filters. The best approach is to apply dynamically changing predictors and detectors to adapt to the changes in attacks and flag down potential threats before they spread. A combination of the traditional method and a dynamic one can also yield more efficient results if applied correctly.

Clustering and classification form a large part of predicting algorithms in statistics. These techniques allow the grouping of items with similar traits. The working of Android means that malware target specific areas for certain information. (Chakraborty, Pierazzi, & Subrahmanian, 2017) Seeks to combine these classification methods in predicting malicious software in Android. The approach seeks to target malware families rather than individual ones. Malware families are sets of malicious apps that share the same origin. By bundling up these characteristics of the families and feeding them to a classifier, (Chakraborty, Pierazzi, & Subrahmanian, 2017) indicates that it is possible to flag down larger numbers of malware than individual inspections. The approach shows great potential for expansion and adaptation to a large-scale detector. Grouping based on similar features such as author, the activity of app, characteristics, required permissions, and dynamic features, among others, builds an excellent overview for a prediction model—the approach results in a better prediction process with high accuracy. Further, the classification also applies smoothly to malware families with smaller numbers. The researchers also indicate that the prediction models allow easy detection of newer malware with modifications but originating from previous ones. Such results indicate that the mass detection of malware can get rid of a large percentage of dangerous applications and leave enough room for the experts to keep building on upcoming malware. Automation is possible in building wider classification features and makes it tougher for the attackers to avoid detection through complex hiding techniques.

The dynamic detection mentioned above is not a new concept but rather an ongoing improvement of malware predictors. One of the key reasons malware is hard to handle is its constantly changing nature. The dynamic characteristic allows attackers to keep infecting more devices by altering their working. (Singh & Hofmann, 217) Analyzes this dynamic nature to detect malware in Android.

The process focuses on the behavior of the application and malware to uncover which ones are suspicious or not. The system calls of Android applications reveal what goes on as it processes data on different levels. Building a classifier on different approaches such as k-NN, Neural Networks, SVM, and random forest and deep learning provides a unique look into the differences. The approach indicates an accuracy rate of 97 percent when using the Support Vector Machine is achieved when using the approach (Singh & Hofmann, 217). The rest of the methods of classification shows much lower accuracy rates. Adding predictors to such classification methods shows a great promise of detecting malware in Android. The percentages of false identification of clean applications are also quite low when investigating the behavioral characteristics. The researchers also revealed that the method could be expanded to other detection and predicting methods, as it is flexible enough to accommodate several dimensions. An expansion of the target applications and data is also a good case for machine learning models, with more promises of better detection. The process, however, depends on previous identification of system calls that can be targeted, new techniques from the attackers can at times pass through undetected.

Most Android detection studies are seeking ways of looking for malware. An excellent approach would be to combine different models of detection and build a process that performs in-depth detection. (Kim, Kang, Rho, Sezer, & Im, 2018) Proposes such a model, which combines similarities of features and existing features to feed a deep learning model as a detector. Such an approach is still not applied widely, indicating much promise in the approach security experts use to weed out malware. Analysis of static features forms a good start of predicting Android malware, as the characteristics are already identifiable. However, the dynamic nature of the apps also needs some consideration. The model calculates the weight of each feature and process and formulates a combined rating of the possibility of the app being malware (Kim, Kang, Rho, Sezer, & Im, 2018). The study additionally allows dynamic inclusion of features not previously identified, with the model adding it to its detection process. The result is an effective process that can adapt to advantageous changes it meets on the way. The neural network built on several vectors progressively learns on the data and improves its classification processes. The results show a high possibility of increasing the detection rate of the model. The accuracy of the multimodal model also outshines other approaches that focus on individual features. The room for comparison of different detection methods also allows the neural network to build a much stronger detection technique than typical approaches. However, the dataset for high accuracy achievement needs is large, as the progressive learning of neural networks depends on as much data as possible.

(Li, et al., 2018) This reveals that most malware detection methods shine on a particular set of problems but lack enough room for expansion. This claim is evident when looking at the increasing rate of malware introduced to Android users. The researchers propose a new approach to detecting the harmful cases that can effectively keep pace with the increase in malware in the Android platform. The process approaches by identifying the permissions of the applications and classifying which ones are more prone to be utilized by malware. Machine learning prediction and classification split the clean from the malicious cases and a comparison of effectiveness and performance recorded. The model the study uses is called Significant Permission Identification and gets the permissions from the listed permissions of the application instead of dynamically requested ones. Such a path means that early detected permissions possibly misused build a strong case of prediction and eventual detection. Tree-based machine algorithms result in better detectors than the rest (Li, et al., 2018). The approach also results in more efficiency in that the valid permissions can be reduced before analysis, which fastens the detection. By pruning the permissions and features, the study results in twenty-two permissions, increasing detection performance by more than eighty-five percent. The detection accuracy increases to ninety percent compared to typical detectors that analyze requested permissions. The early nature of the approach shows that highly effective predictors are possible soon, especially with possible levels of detection and prediction that are possibly built on the approach.

A good area to look for malware is in analyzing the network flow. The information shared through the network can show if sensitive data is leaking through. An approach by (Wang, et al., 2017) indicates a possible method of detecting Android malware by analyzing the semantics of network flows. The study proposes a model that studies the flow of HTTP information that applications generate and applies a natural language process. The textual information of the network information provides a good overview of the mobile apps. The malware detection occurs upon extraction and processing of the textual information. Getting rid of words that do not help the process is also crucial in avoiding errors in accuracy through false classifications. The approach also removes meaningless symbols such as punctuation. High-frequency words also hold no need in the study, as (Wang, et al., 2017) indicates, leaving the set with crucial words that can provide good results after natural analysis. Generated N-gram sets additionally segment the data sets into sections for training the detection model. Early results indicate that malicious applications do not always generate malicious texts. Encrypted texts also present a hard time for the model, as they contain gibberish texts. However, the approach is quite good at detecting malicious applications and shows much potential in the application on malware detection. The case of encryption is slowly growing as more applications encrypt their network traffic. A combination with other effective approaches is possible to improve the overall accuracy rates. A possible combination would be to provide an analysis of the source and target IP addresses. The study shows the advances Android Malware detection has taken, alongside the increasing complexity of the malware.

5. Methodology Used

Detecting android malware using predictive analytics requires using certain techniques to achieve the expected results. In this capstone project, two types of network traffic are labeled in our dataset as either ‘Malware’ or ‘Benign’. For this, we need to predict, depending on several features whether network traffic is malware or not using Google Colab.

Data cleaning is the first important step to use to prepare a cleaned dataset for training and testing with fewer errors. In this part, we loaded the data and checked the head values to identify the number of entries, columns, and drop the null values.

In Exploratory Data Analysis, we aimed to use a five-point summary to detect data type (continuous and discrete features). Data visualization was implemented in EDA to check head values and understand the variables that we are going to deal with during the analysis. Moreover, in the visualization, we aimed to use charts to understand the overall distributions of continuous and discrete columns that were detected. Histogram was used for continuous variables, bar chart was used for discrete variables to check the skew, and boxplot was used to detect outliers and treat them.

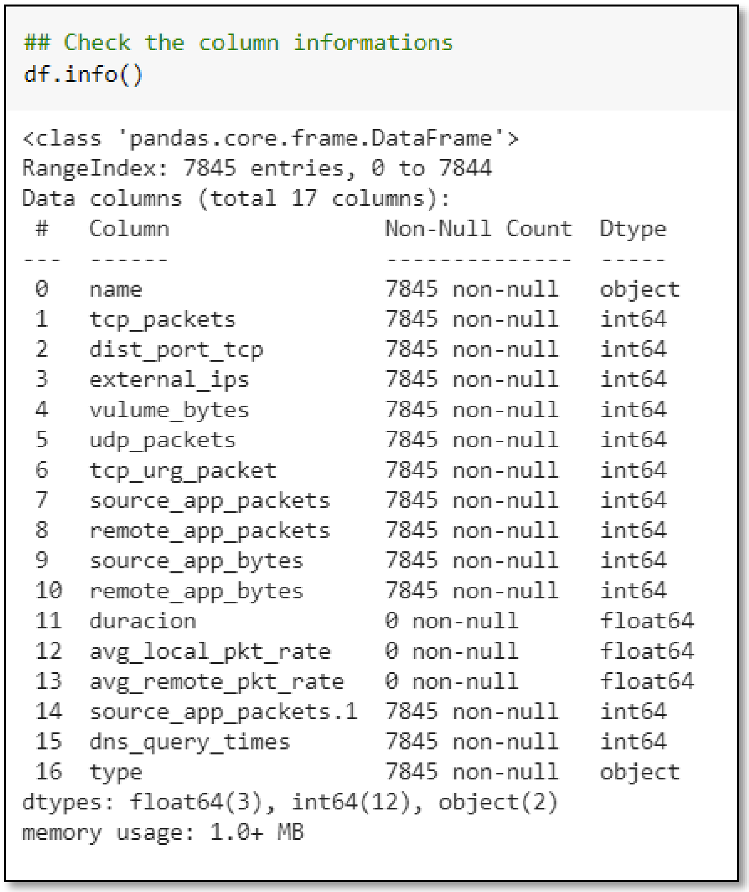
The feature selection method was used to select the final set of features where the statistical testing hypothesis was applied. The aim of this is to check whether a particular feature is correlated with the target variable or not. Since the target variable is categorical, the ANOVA technique was used to test and check the correlation between the continuous variables. Since the target variable is categorical and the predictor is also categorical, we explored the correlation between them visually using a bar plot and applying the Chi-Square test.

Before building the model, we pre-processed our target variable which is the ‘type’ ad replaced benign network traffic with 0 and malware network traffic with 1. Then, we went into splitting the data into training and testing. Next, the minmax transformation was implemented to make the data more standard which resulted in having a divided data ratio of 70:30 randomly. After splitting, 4144 records in the training set and 1776 records for the testing set. In this method, different models were implemented on the training data such as logistic regression, decision tree, random forest, and KNN. For each of these models, the validation has been made using the testing set.

6. Sources of Data

The dataset was found on the Kaggle website. The data is helpful to conduct the analysis and deliver the project goals since it was collected based on real network traffic. The dataset describes what a network packet consists of such as source, destination, DNS query time, and many others as described in the table below. It was found that data is quantitative. Moreover, it was observed that the dataset consisted of a total of 17 columns and 7845 rows. We noticed that some columns have NULL values, so cleaning data must be considered before splitting the data into training and testing to reduce the chance of errors.

No.	Column Name	Description
1	name	Name of Application
2	tcp_packets	transmission control protocol
3	dist_port_tcp	distributed port transmission control protocol
4	external_ips	External Internet Protocol
5	vulume_bytes	Volume of application in bytes
6	udp_packets	User Datagram Protocol
7	tcp_urg_packet	Urg flag of tcp
8	source_app_packets	Source application packet
9	remote_app_packets	Remote application packet
10	source_app_bytes	Source application size in bytes
11	remote_app_bytes	Remote application size in bytes
12	dns_query_times	Domain Name System query in times
13	type	Type of application, malware or benign (Target variable)



```
## Check the column informations
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7845 entries, 0 to 7844
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   name                 7845 non-null   object 
 1   tcp_packets          7845 non-null   int64  
 2   dist_port_tcp        7845 non-null   int64  
 3   external_ips         7845 non-null   int64  
 4   vulume_bytes         7845 non-null   int64  
 5   udp_packets          7845 non-null   int64  
 6   tcp_urg_packet       7845 non-null   int64  
 7   source_app_packets   7845 non-null   int64  
 8   remote_app_packets   7845 non-null   int64  
 9   source_app_bytes     7845 non-null   int64  
 10  remote_app_bytes     7845 non-null   int64  
 11  duracion             0 non-null      float64
 12  avg_local_pkt_rate   0 non-null      float64
 13  avg_remote_pkt_rate  0 non-null      float64
 14  source_app_packets.1 7845 non-null   int64  
 15  dns_query_times      7845 non-null   int64  
 16  type                 7845 non-null   object 
dtypes: float64(3), int64(12), object(2)
memory usage: 1.0+ MB
```

Figure 1 – checking the column information and finding NULL valuse

7. Analysis

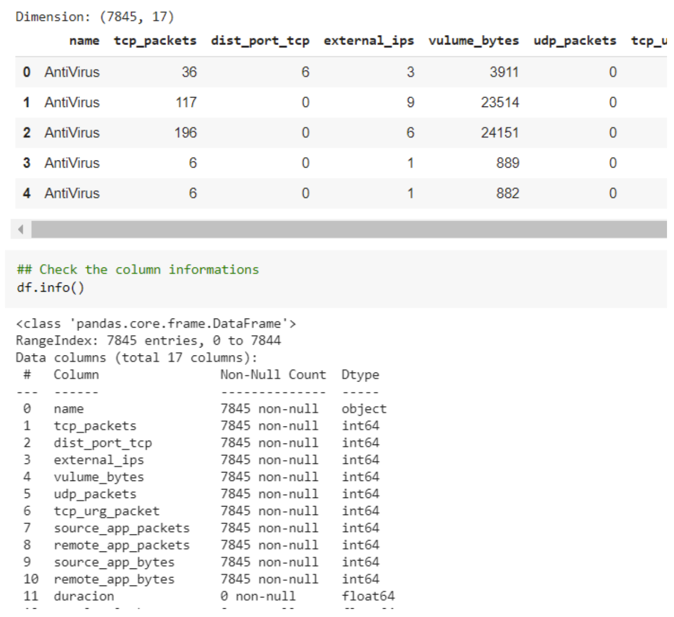
Importing required Libraries

In the beginning we will import the data science libraries as mentioned below:

- NumPy is used for calculating single summary measures, handling missing values, etc.
- Pandas is used for reading data from CSV files, data manipulation, Five-point summary, cross-tabulation, etc.
- MatplotLib is used for data visualization with Pandas visualization methods.
- SciPy is also used for performing statistical tests like ANOVA, Chi-Square tests, etc.

Load the data and check head values

Checking head values allows viewing the first few rows to understand the data and how data is imported. Moreover, checking the column information identifies the columns that needs to be dropped to work with clean data. For example, dropping the columns that contain NULL Values.



Dimension: (7845, 17)

	name	tcp_packets	dist_port_tcp	external_ips	vulume_bytes	udp_packets	tcp_u
0	AntiVirus	36	6	3	3911	0	
1	AntiVirus	117	0	9	23514	0	
2	AntiVirus	196	0	6	24151	0	
3	AntiVirus	6	0	1	889	0	
4	AntiVirus	6	0	1	882	0	

```
## Check the column informations
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7845 entries, 0 to 7844
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   name                7845 non-null   object
 1   tcp_packets         7845 non-null   int64
 2   dist_port_tcp       7845 non-null   int64
 3   external_ips        7845 non-null   int64
 4   vulume_bytes        7845 non-null   int64
 5   udp_packets         7845 non-null   int64
 6   tcp_urg_packet      7845 non-null   int64
 7   source_app_packets  7845 non-null   int64
 8   remote_app_packets  7845 non-null   int64
 9   source_app_bytes    7845 non-null   int64
 10  remote_app_bytes    7845 non-null   int64
 11  duracion            0 non-null      float64
```

Figure 2 - checking head values and column information.

Data cleaning

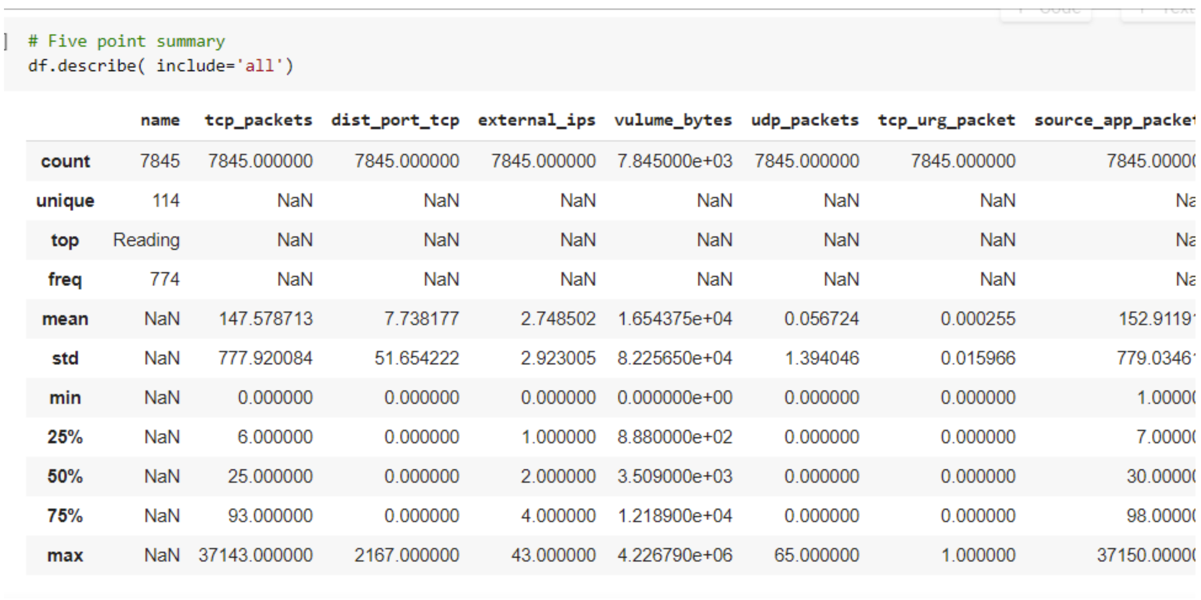
After checking the column information, it has been identified that there are some columns having no values (NULL) which needs to be dropped:

1. duracion
2. avg_local_pkt_rate
3. avg_remote_pkt_rate
4. source_app_packets.1

After dropping the NULL columns, we check the unique values in each column to help us identify discrete, categorical, and continuous columns.

Exploratory data analytics

We explored the data by applying a five-point summary to give us a brief on each variable as showing in figure 3 below.



```
# Five point summary
df.describe( include='all')
```

	name	tcp_packets	dist_port_tcp	external_ips	vulume_bytes	udp_packets	tcp_urg_packet	source_app_packe
count	7845	7845.000000	7845.000000	7845.000000	7.845000e+03	7845.000000	7845.000000	7845.0000
unique	114	NaN	NaN	NaN	NaN	NaN	NaN	Na
top	Reading	NaN	NaN	NaN	NaN	NaN	NaN	Na
freq	774	NaN	NaN	NaN	NaN	NaN	NaN	Na
mean	NaN	147.578713	7.738177	2.748502	1.654375e+04	0.056724	0.000255	152.9119
std	NaN	777.920084	51.654222	2.923005	8.225650e+04	1.394046	0.015966	779.0346
min	NaN	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	1.0000
25%	NaN	6.000000	0.000000	1.000000	8.880000e+02	0.000000	0.000000	7.0000
50%	NaN	25.000000	0.000000	2.000000	3.509000e+03	0.000000	0.000000	30.0000
75%	NaN	93.000000	0.000000	4.000000	1.218900e+04	0.000000	0.000000	98.0000
max	NaN	37143.000000	2167.000000	43.000000	4.226790e+06	65.000000	1.000000	37150.0000

Figure 3 – Five-point summary results

We explore the categorical variables starting with the column (type) by a bar plot which resulted in having 4704 benign network traffic and 3141 malicious network traffic.

```
type
benign       4704
malicious    3141
dtype: int64
```

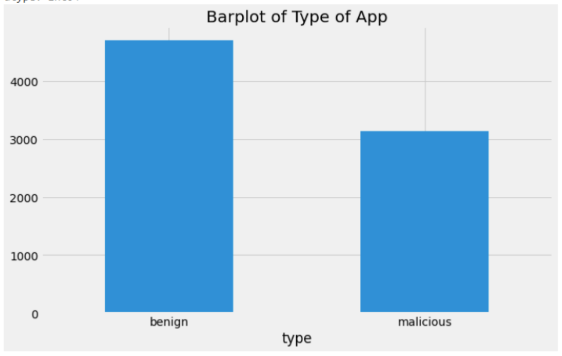


Figure 4 – bar plot of ‘type’ variable

Another categorical column (Name) has been visualized using a bar plot to see the different names that are most repeated and used in the network traffic. As it is showing in figure 5 that the most used name is (Reading) but we are still not sure whether it is malware or benign.

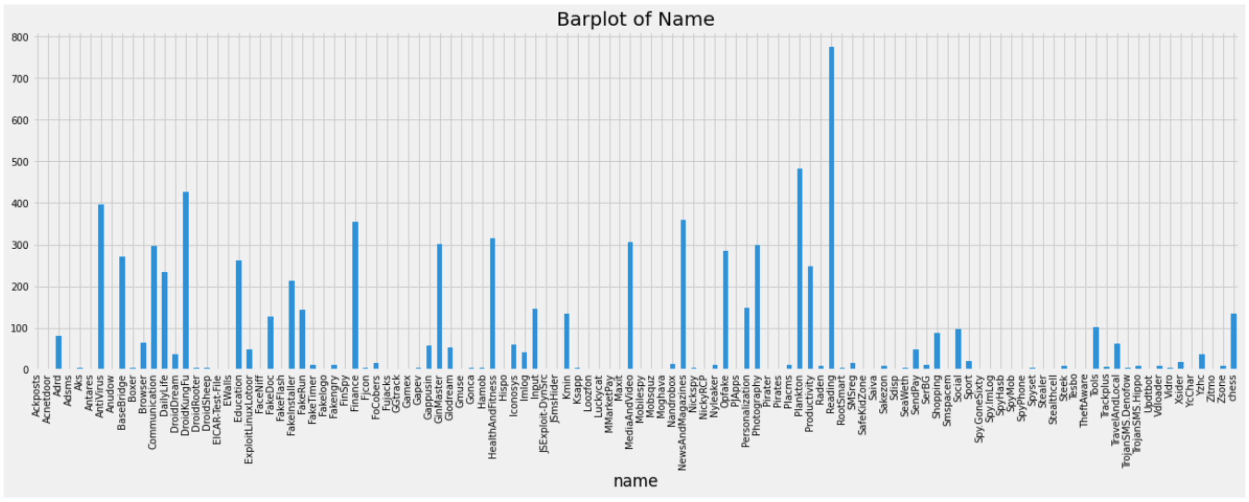


Figure 5 – bar plot of ‘name’ variable

There is a list of flag types in the TCP header, but when it comes to the (tcp_urg_packet) it is mainly being used to inform the receiving station that this packet needs to be prioritized. Visualization has been done also by a bar plot for the column (tcp_urg_packet). Which shows whether any packet was flagged as urgent by flagging it as = 1 as it is showing in the figure below, all traffics were not flagged as urgent since all of them showed as =0. As a result, we understand that neither benign nor malware traffic is flagged as urgent packets.

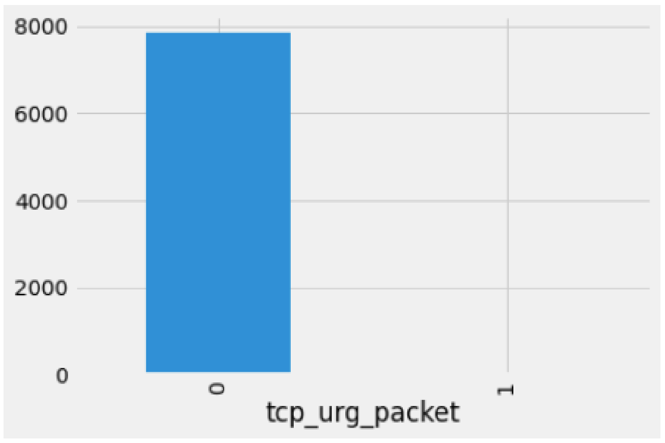


Figure 6 – bar plot of urgent flagging in TCP header

<u>Continuous variable visualization</u>

we visualized the following continuous variables using histogram plot.

1. tcp_packets,
2. dist_port_tcp,
3. external_ips,
4. vulume_bytes,
5. udp_packets,
6. source_app_packets,
7. remote_app_packets, s
8. ource_app_bytes,
9. remote_app_bytes,
10. dns_query_times.

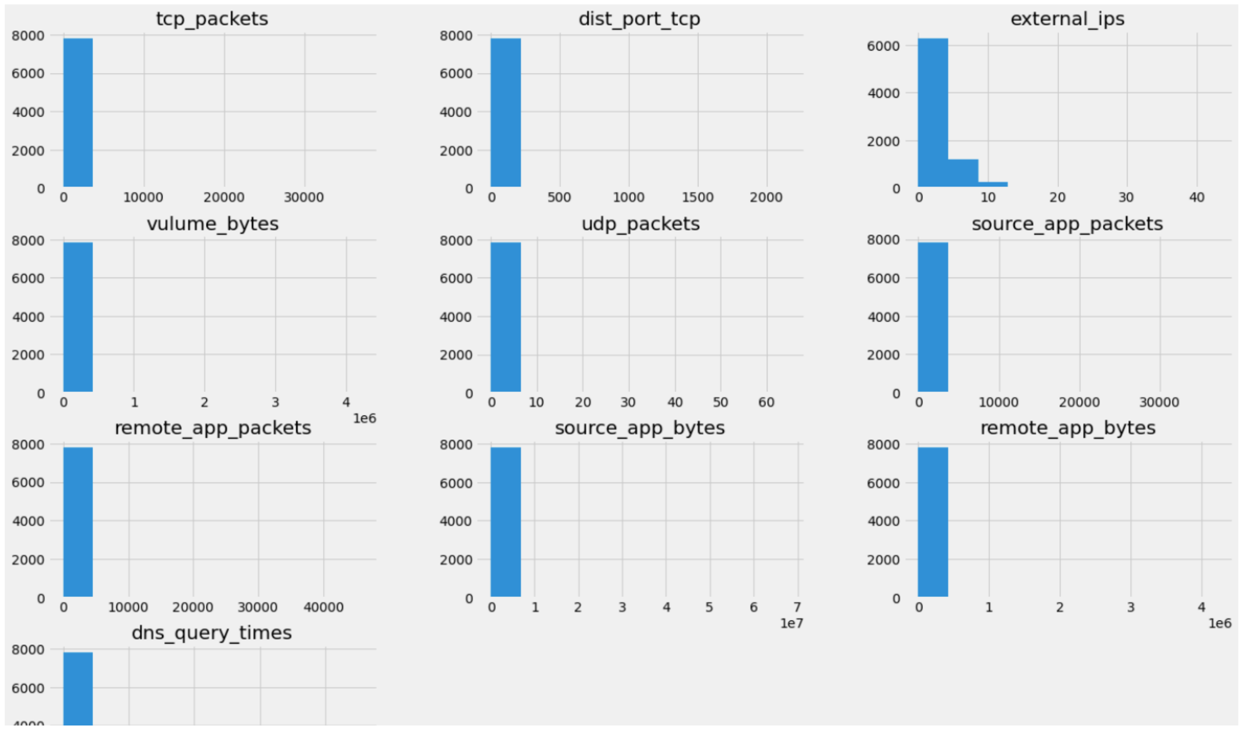


Figure 7 – continuous variables visualization using histogram

As a result, we observed that data has a good number of outliers which needed to be detected, and the best way to detect an outlier is using a boxplot.

Anomaly Detection & Treatment

In this section, a boxplot will be used for each column to detect outliers and treat them. We will use a boxplot depending on the two levels of the target variable. That is, there will be two boxplots in the same window, one is for type 'malicious' and another is for type 'benign'. Ideally, we will remove the outliers that are mainly belonging to benign class. Since we are focusing more on detecting malicious applications. The figure below shows an example of how the outliers did were removed from each column.

For example, the tcp_packets outliers were detected after the number of 200. For this, by several trials, we get the observation with the tcp_packet >200 should be removed as those are high outliers. This process is applied to all the columns with the continuous variables to clean them from their outliers.

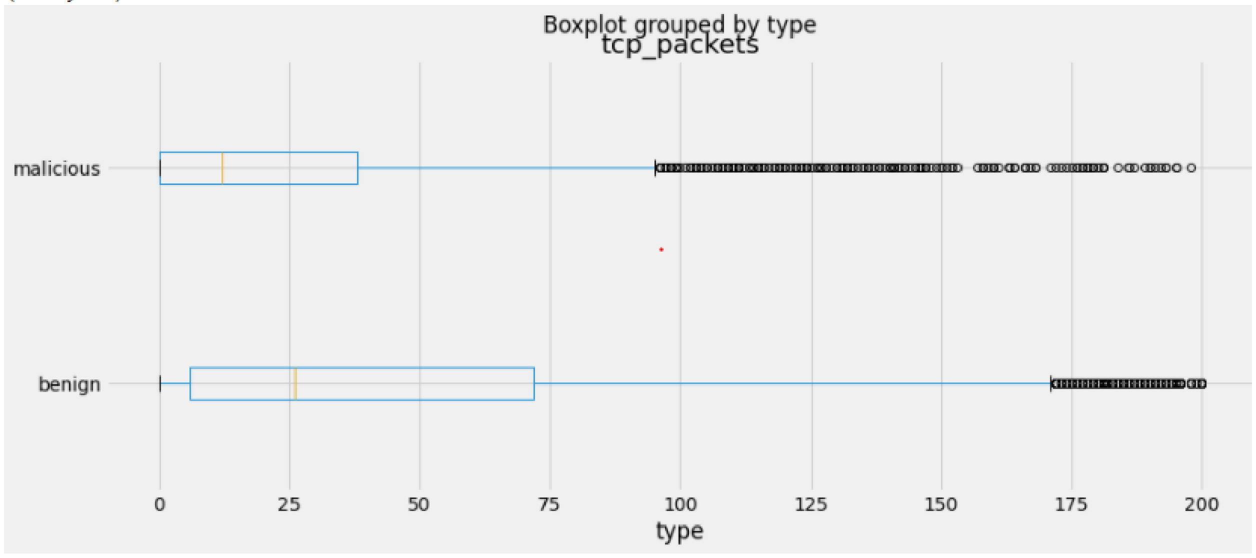


Figure 8 – tcp_packets outlier detection

After detecting and treating the outliers, we visualize the columns again by a histogram plot except for dns_query_times and external_ips columns since we noticed that they are discrete variables and visualizing them will be using bar plot instead of the histogram.

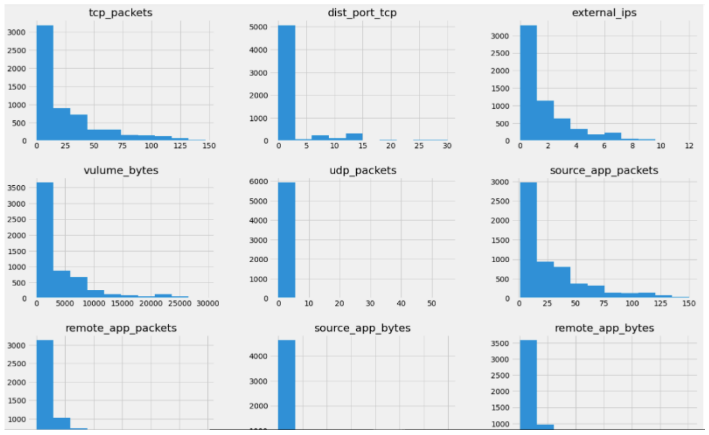


Figure 9 – Histogram of all continuous variables after detecting outliers.

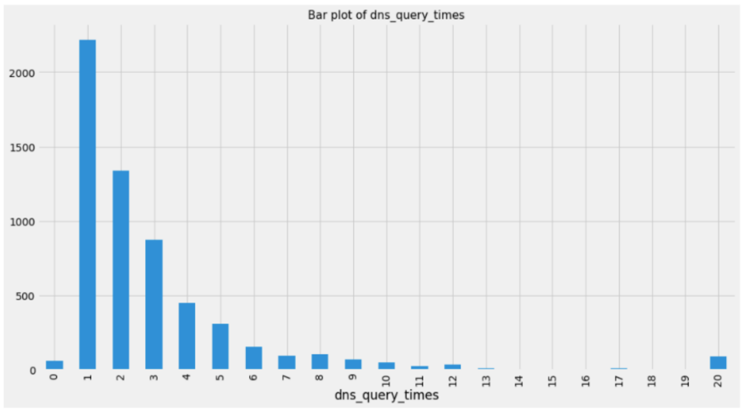


Figure 10 – bar plot of dns_query_times

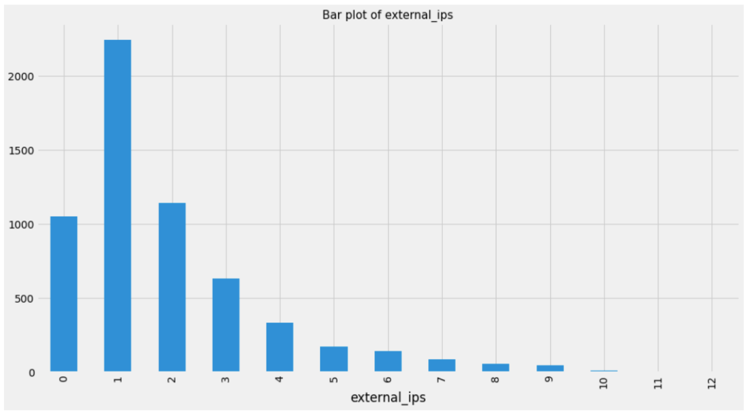


Figure 11 – bar plot of external_ips

In figure 12 below, we visualize all bar plots and check their skewness. We notice that udp_packets and tcp_udp_packets are having an extremely skewed distribution, so we discard them from further analysis.

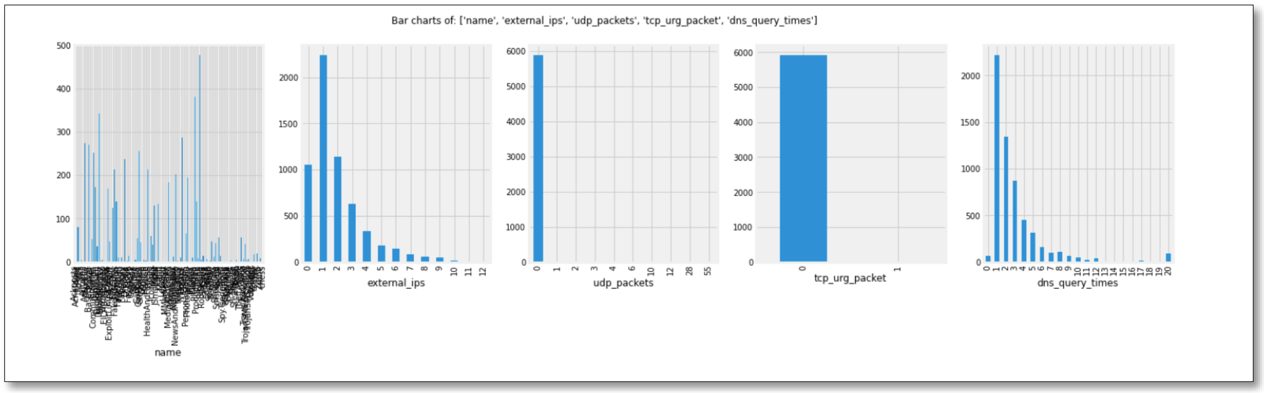


Figure 12 – visualizing all bar plots together and check their skewness

After treating all outliers, data has become more balanced in terms of classes of target variables.

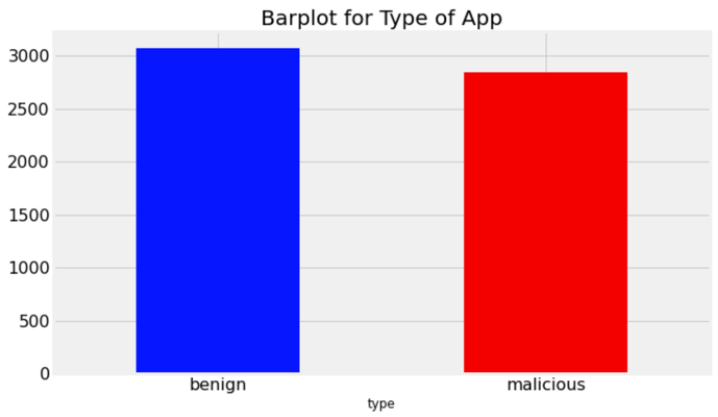


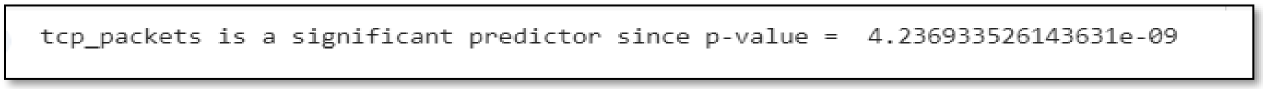
Figure 13 – bar plot of 'type' variable after treating outliers

Feature Selection

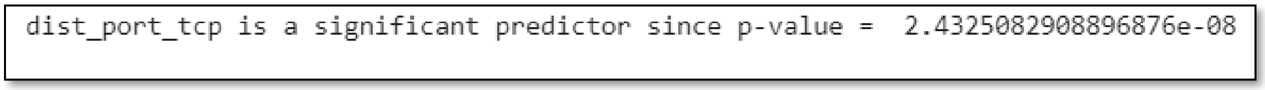
Statistical feature selection using ANOVA Test

As our target variable is categorical, we will use the ANOVA technique for checking the correlation with continuous variables. Analysis of variance (ANOVA) is performed to check if there is any relationship between the given continuous and categorical variables.

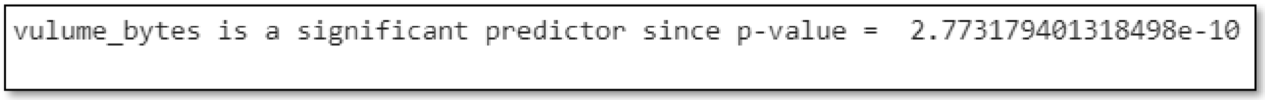
- Assumption(H0): There is NO relation between the given variables (i.e. The average(mean) values of the numeric Predictor variable is the same for all the groups in the categorical Target variable)
- ANOVA Test result: Probability of H0 being true



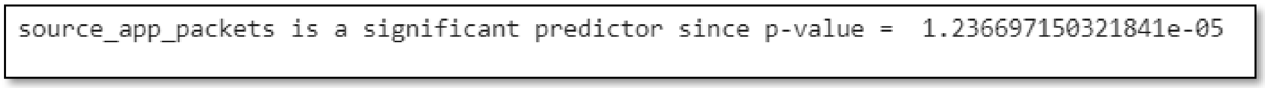
```
tcp_packets is a significant predictor since p-value =  4.236933526143631e-09
```



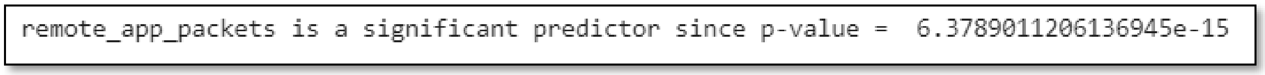
```
dist_port_tcp is a significant predictor since p-value =  2.4325082908896876e-08
```



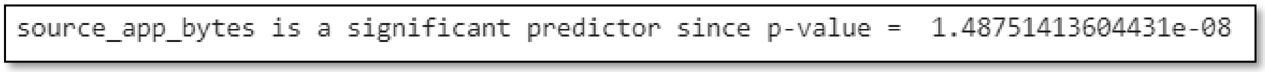
```
vulume_bytes is a significant predictor since p-value =  2.773179401318498e-10
```



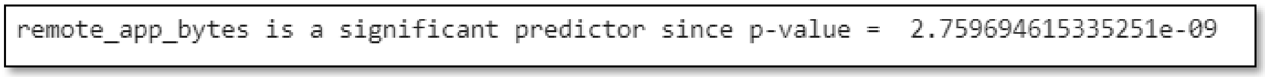
```
source_app_packets is a significant predictor since p-value =  1.236697150321841e-05
```



```
remote_app_packets is a significant predictor since p-value =  6.3789011206136945e-15
```



```
source_app_bytes is a significant predictor since p-value =  1.48751413604431e-08
```



```
remote_app_bytes is a significant predictor since p-value =  2.759694615335251e-09
```

Relationship exploration: categorical vs. categorical

In figure14 below, we visualized the variable ‘name’ with our target variable (type) to check the most repeated names in malware network traffic. As it is showing that the name (plankton) and (DroidKungFu) are the most names used in malware traffic.

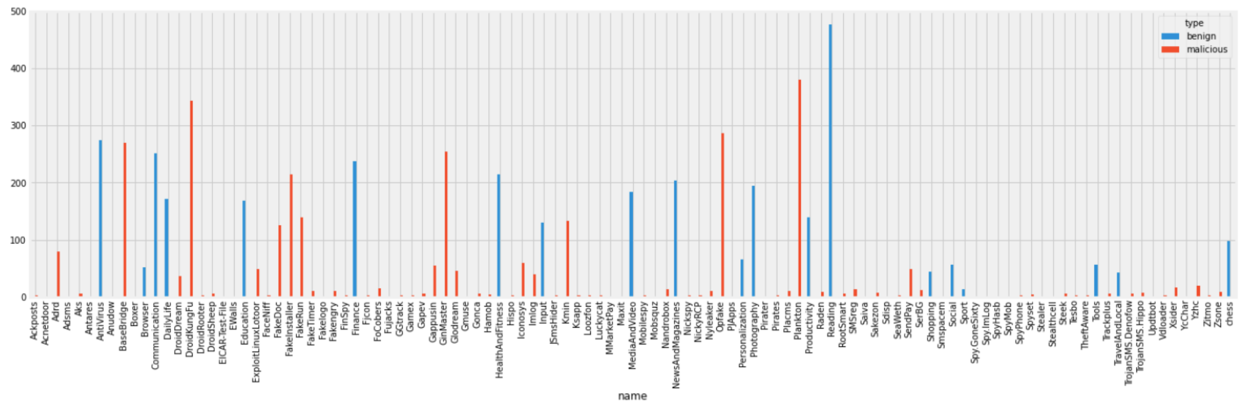


Figure 14 – visualizing (name) and ‘type’ variables using bar plot

In figure15 below, we visualized the categorical variable (external_ips) with our target variable (type) to find the most repeated IP Addresses the network traffic is coming from. As it is showing that there is more than one external IP used repeatedly in the network traffic, and most of the network traffic is coming from malicious external IP Addresses

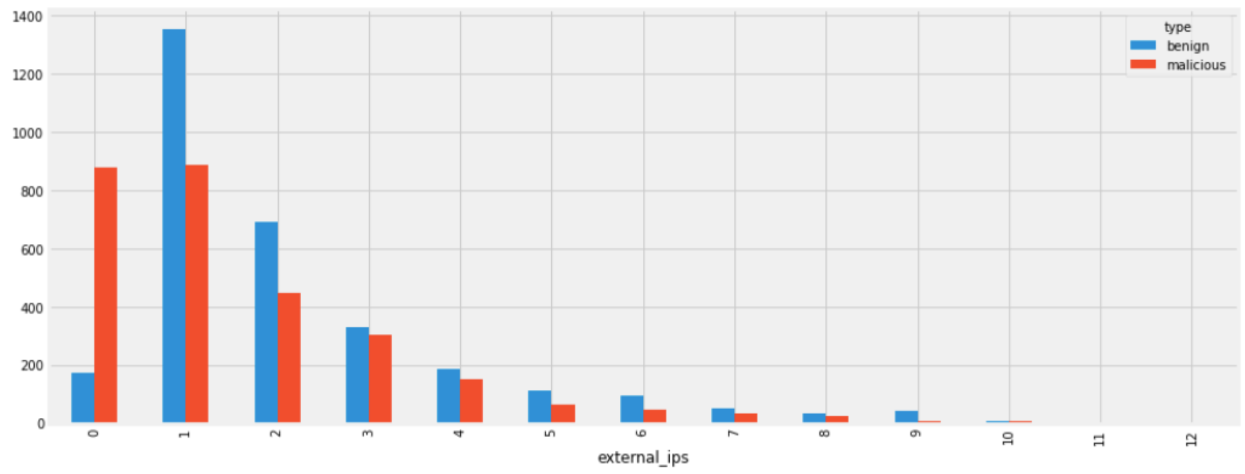


Figure 15 visualizing (external_ips) and ‘type’ variables using bar plot

In figure16 below, we visualized the categorical variable (dns_query_times) with our target variable (type) to find DNS query times for both benign and malware traffic. As it is showing that the number of the query times initiated by malware network traffic is more than the benign network traffic.

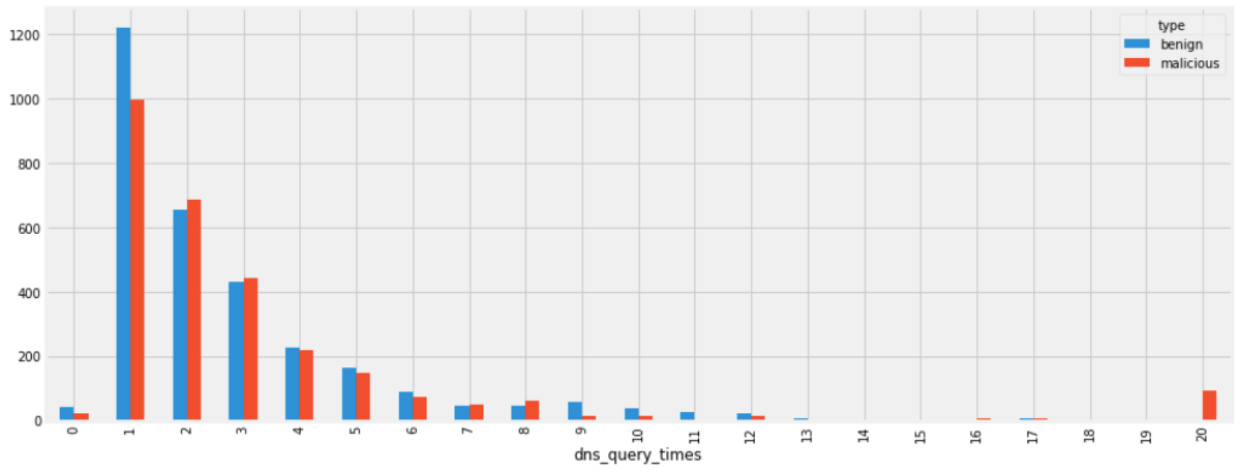


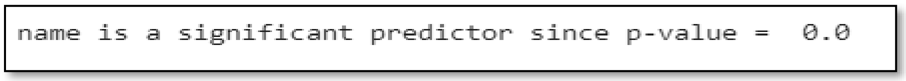
Figure 16 - visualizing (dns_query_times) and (type) variables using bar plot

Statistical feature selection (Categorical vs. Categorical) using Chi-Square Test

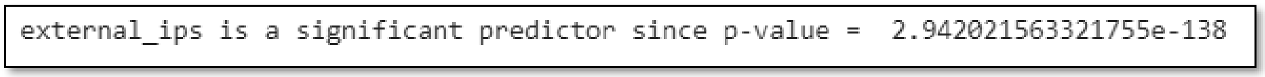
When the target variable is Categorical, and the predictor is also categorical then we explore the correlation between them visually using bar plots and statistically using the Chi-square test.

Chi-Square test is conducted to check the correlation between two categorical variables

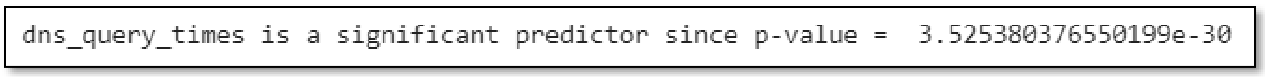
- Assumption(H0): The two columns are NOT related to each other
- Result of Chi-Sq Test: The Probability of H0 being True



```
name is a significant predictor since p-value =  0.0
```



```
external_ips is a significant predictor since p-value =  2.942021563321755e-138
```

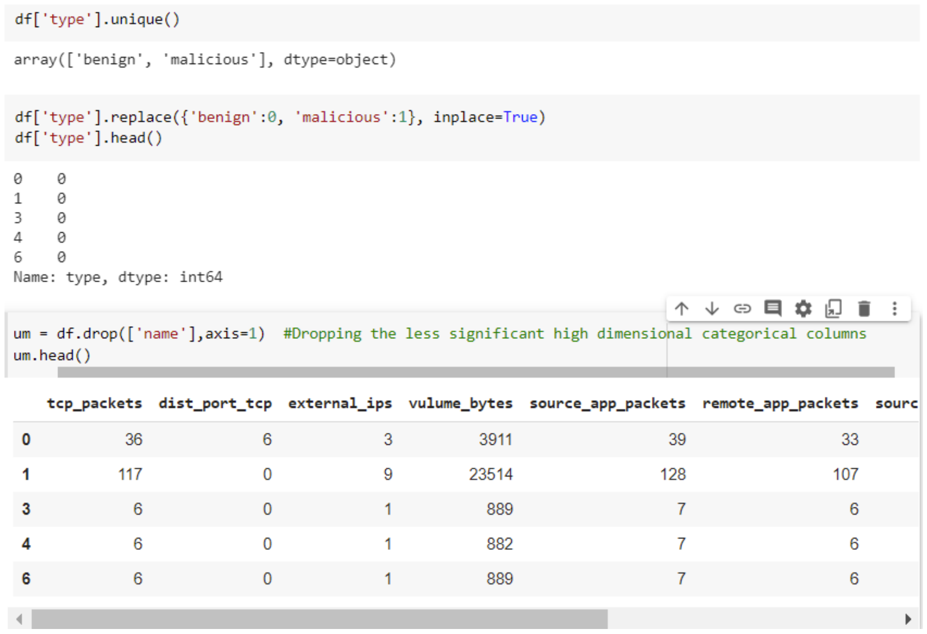


```
dns_query_times is a significant predictor since p-value =  3.525380376550199e-30
```

From the above analysis, we get that almost all predictors are statistically significant at 5% level except the first column **(name)**, which is very high dimensional and not so significant. So, we decide to drop that column and continue with the rest of the feature.

Data Pre-processing

In this part, we pre-processed the column (type) to reflect benign as 0 and malicious as 1. We also dropped the column (name) since it is the less significant high-dimensional categorical column.



```
df['type'].unique()
```

```
array(['benign', 'malicious'], dtype=object)
```

```
df['type'].replace({'benign':0, 'malicious':1}, inplace=True)
df['type'].head()
```

```
0    0
1    0
3    0
4    0
6    0
Name: type, dtype: int64
```

```
um = df.drop(['name'],axis=1)  #Dropping the less significant high dimensional categorical columns
um.head()
```

	tcp_packets	dist_port_tcp	external_ips	vulume_bytes	source_app_packets	remote_app_packets	sourc
0	36	6	3	3911	39	33	
1	117	0	9	23514	128	107	
3	6	0	1	889	7	6	
4	6	0	1	882	7	6	
6	6	0	1	889	7	6	

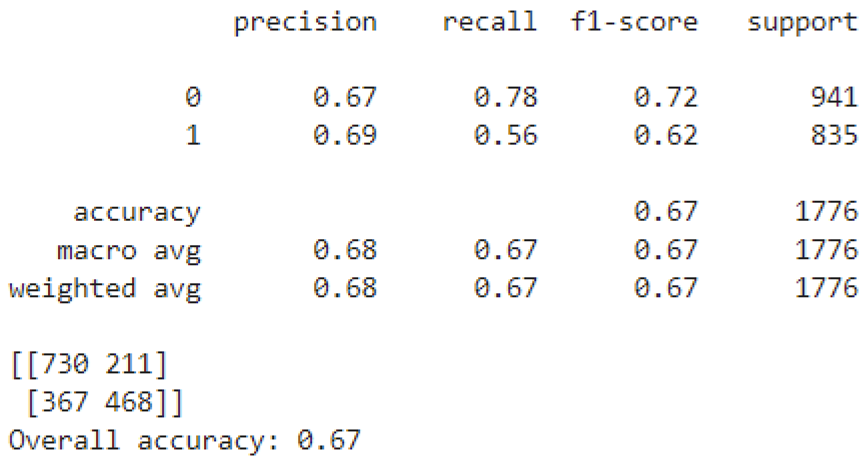
Figure 17- preprocessing variable (type)

Machine Learning: Splitting the data into Training and Testing sample

In splitting the dataset, we do not use the full data for creating the model. Some data is randomly selected and kept aside for checking how good the model is. This is known as Testing Data and the remaining data is called Training data on which the model is built. Typically, 70% of data is used as training data and the rest 30% is used as testing data.

Logistic Regression

Logistic regression is used to train or build an algorithm that will predict the target value, and it showed an accuracy of 0.67 of its ability to predict whether network traffic is malicious or not.

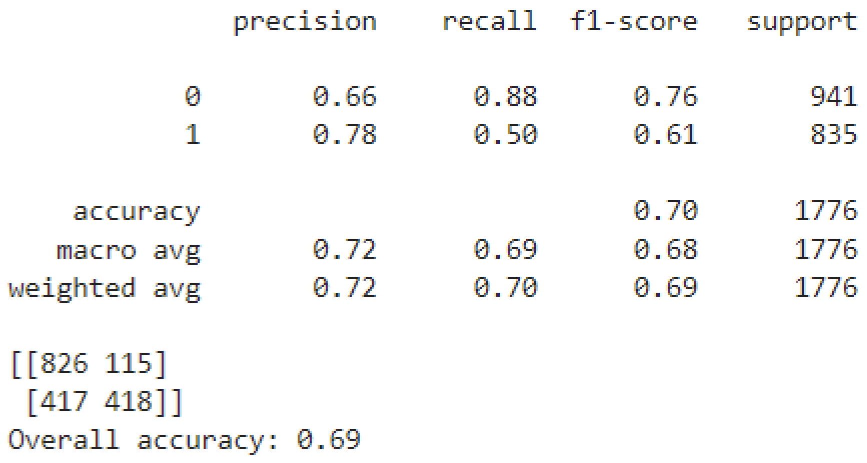


```
              precision    recall  f1-score   support

           0       0.67      0.78      0.72       941
           1       0.69      0.56      0.62       835

    accuracy                           0.67      1776
   macro avg       0.68      0.67      0.67      1776
weighted avg       0.68      0.67      0.67      1776

[[730 211]
 [367 468]]
Overall accuracy: 0.67
```

Figure 18 – logistic Regression accuracy

Decision Tree

The decision tree is an additional model used to check its ability to predict the target value, and it showed an accuracy of 0.69 of its ability to predict whether network traffic is malicious or not.



```
              precision    recall  f1-score   support

           0       0.66      0.88      0.76       941
           1       0.78      0.50      0.61       835

    accuracy                           0.70      1776
   macro avg       0.72      0.69      0.68      1776
weighted avg       0.72      0.70      0.69      1776

[[826 115]
 [417 418]]
Overall accuracy: 0.69
```

Figure 19 – Decision Tree accuracy

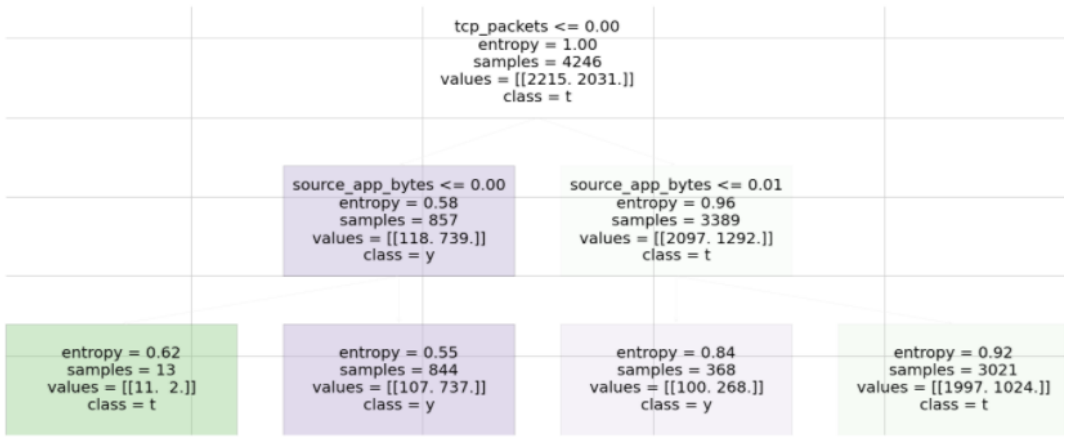
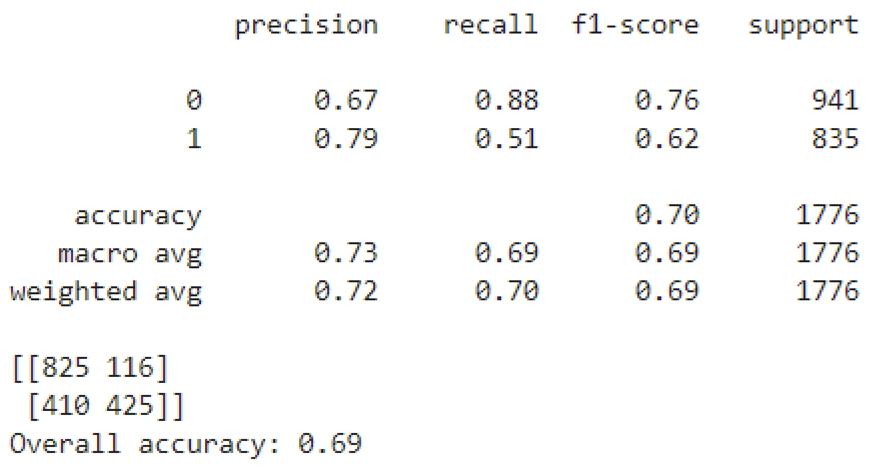


Figure 20 – Decicsion tree plot

Random Forest

Random forest is another model used to check its ability to predict the target value, and it showed an accuracy of 0.69 of its ability to predict whether network traffic is malicious or not.

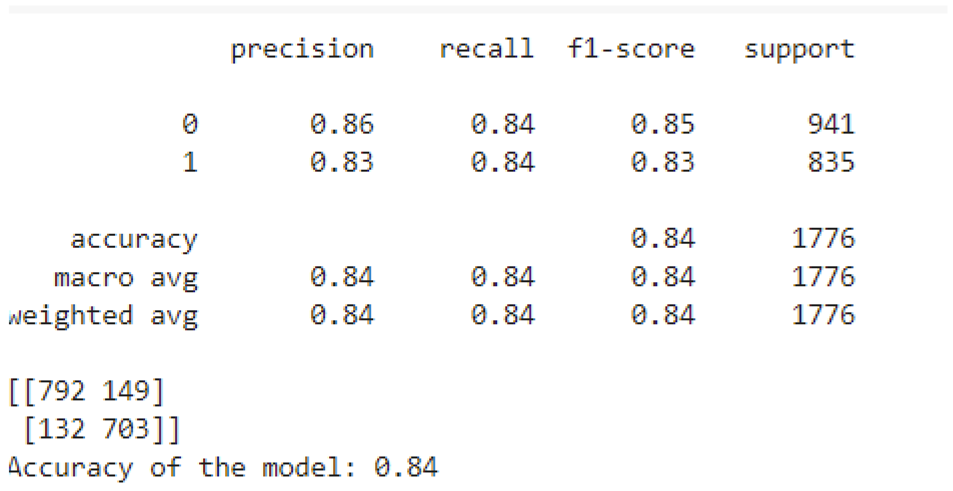


```
              precision    recall  f1-score   support

           0       0.67      0.88      0.76       941
           1       0.79      0.51      0.62       835

    accuracy                           0.70      1776
   macro avg       0.73      0.69      0.69      1776
weighted avg       0.72      0.70      0.69      1776

[[825 116]
 [410 425]]
Overall accuracy: 0.69
```

Figure 21 – Random Forest Accuracy

AdaBoost

AdaBoost model showed an accuracy of 0.84 of its ability to predict whether a network traffic is malicious or not.

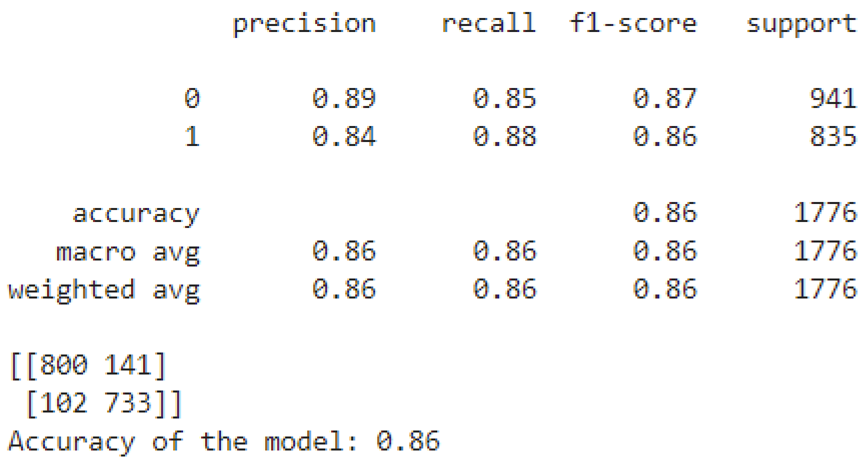


```
              precision    recall  f1-score   support

           0       0.86      0.84      0.85       941
           1       0.83      0.84      0.83       835

    accuracy                           0.84      1776
   macro avg       0.84      0.84      0.84      1776
weighted avg       0.84      0.84      0.84      1776

[[792 149]
 [132 703]]
Accuracy of the model: 0.84
```

Figure 22 – AdaBoost Accuracy

KNN

KNN model showed an accuracy of 0.86 of its ability to predict whether a network traffic is malicious or not.



```
              precision    recall  f1-score   support

           0       0.89      0.85      0.87       941
           1       0.84      0.88      0.86       835

    accuracy                           0.86      1776
   macro avg       0.86      0.86      0.86      1776
weighted avg       0.86      0.86      0.86      1776

[[800 141]
 [102 733]]
Accuracy of the model: 0.86
```

Figure 23 – KNN Accuracy

8. Results

From the below table, we can see that AdaBoost algorithm and KNN are giving higher accuracy between all algorithms used. As a result, we can rely on KNN as the best model to predict malware applications in Android operation system.

Model Comparison based on overall accuracy:

Algorithms	**Precision**	**Recall**	**Accuracy**
Logistic regression	0.68	0.67	67%
Decision tree	0.72	0.70	69%
Random forest	0.72	0.70	69%
AdaBoost	**0.84**	**0.84**	**84%**
KNN	**0.86**	**0.86**	**86%**

In networking, it is very important to know each traffic's general name, from where it its coming, and where it is going. For this, among all 9 features, we found the top 3 important features in the network traffic are as listed:

- tcp_packets
- dist_port_tcp
- source_app_packets

The feature importance chart is as below:

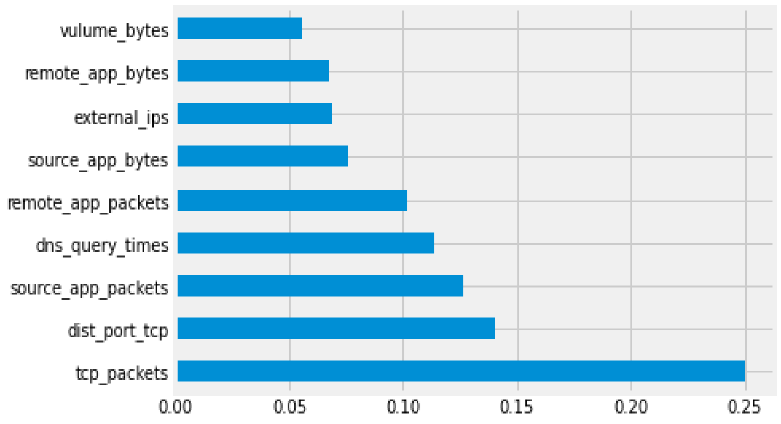


Figure 24 – feature importance chart

Overall, the results met our expectation by proving the ability of different machine learning algorithms to predict malware in the network traffic. There were some differences in the percentage of the prediction for each algorithm, but it was successfully proved to us that machine learning algorithms can extract meaningful information from the amount of data it receives to predict malware occurrence. Since the use of Android devices keeps increasing worldwide, it is the time to limit the chances of failed malware detection by adopting machine learning algorithms into the operating system. Having a strong operating system in the device will ensure high performance and security. Building machine learning algorithms in Android operating system will also ensure that all packets from different sources and stores are being captured, analyzed, and treated based on the previous and continuous learning of the model. Android users should always be protected by applying the latest and best technologies on their devices, we assume that build-in machine learning algorithms in the operating system will help toward fast detection of malware, taking the required action, and assure high level of security.

9. Conclusions/Future Work

In conclusion, we understand how important it is to embed new technology to the traditional techniques which is still used nowadays to defend against malware threats. We never underestimate the power of any technology, but we believe that integrating those technologies together will build a strong network and security infrastructure to defend against daily threats and malware attacks on the Android operating system.

While working on this project, we saw that there where limited work done on analyzing network traffic using predictive analytics techniques. In addition, machine learning algorithms were applied wonderfully which showed us how powerful and important it is today to move forward towards adopting this kind of technology into day-to-day businesses to save time, effort, and provide a high amount of security, and take better decisions.

Using predictive analytics can help a lot in the information security field including securing the Android operating system from malware by using machine learning algorithms such as logistic regression, Decision tree, Random Forest, AdaBoost, and KNN. Although a small dataset of network traffic was used to apply those models, the accuracy percentage of each algorithm gave an indication this is possible to be integrated into the Android operating system instead of relying on third-party applications to scan the device from malware and depending on the end-user to be aware and careful which will end up one day with malware in the device by a random click.

Huge companies like Google and many more, are working hard to apply those technologies to their systems. We believe that they are doing a great job in securing their operating system. As technology gets updated every day, and attackers get a new path to sneak into the Android operating system, we need to be in the front security layer defending against expected and non-expected malware which is targeting the Android operating system by predicting it and dropping it before it reaches the user's device.

10. Bibliography

Chakraborty, T., Pierazzi, F., & Subrahmanian, V. S. (2017). Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Transactions on Dependable and Secure Computing, 17*(2), 262-277.

Kim, T., Kang, B., Rho, M., Sezer, S., & Im, E. G. (2018). A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security, 14*(3), 773-788.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics, 14*(7), 3216-3225.

Singh, L., & Hofmann, M. (217). Dynamic behavior analysis of android applications for malware detection. *In 2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)* (pp. 1-7). IEEE.

Wang, S., Yan, Q., Chen, Z., Yang, B., Zhao, C., & Conti, M. (2017). Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security, 13*(5), 1096-1109.

Zhu, D., Jin, H., Yang, Y., Wu, D., & Chen, W. (2017). DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data. *In 2017 IEEE symposium on computers and communications (ISCC)* (pp. 438-443). IEEE.

Appendix

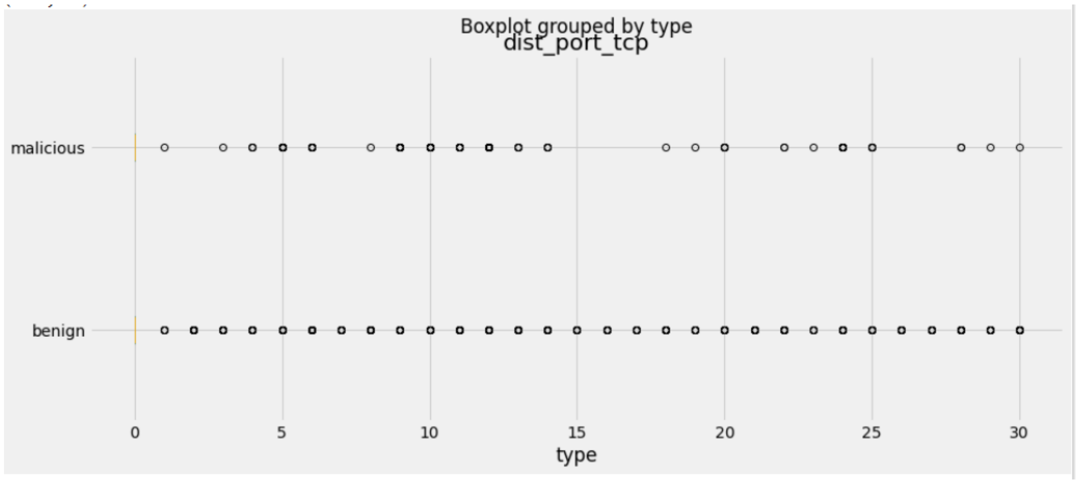


Figure 25 -box plot of observations with dist_port_tcp >30 should be dropped because of high outliers

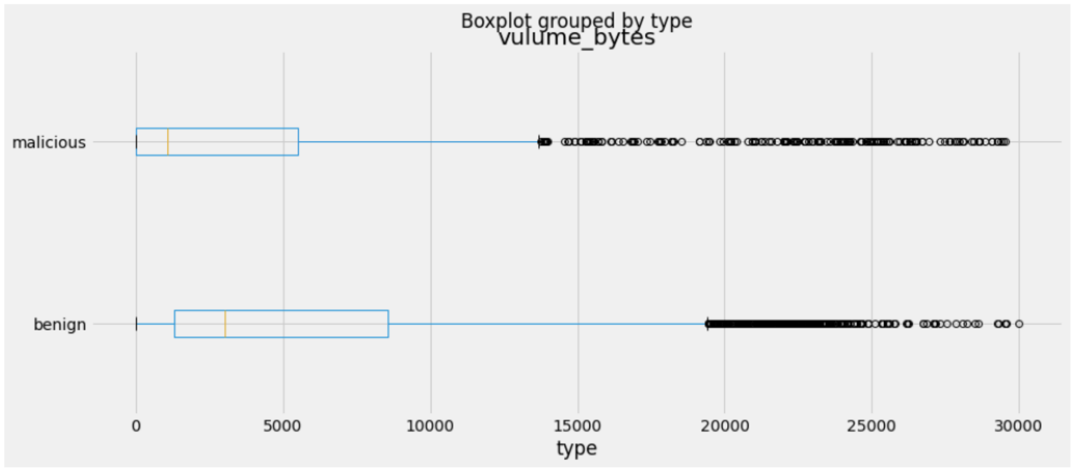


Figure 26 - box plot of observations with vulume_bytes>30000 should be dropped because of high outliers

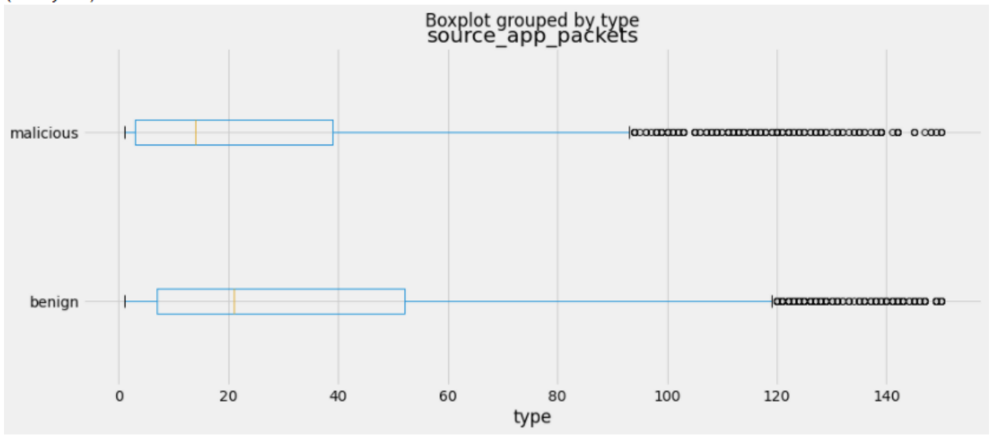


Figure 27 - box plot of observations with source_app_packets>150 should be dropped because of high outliers

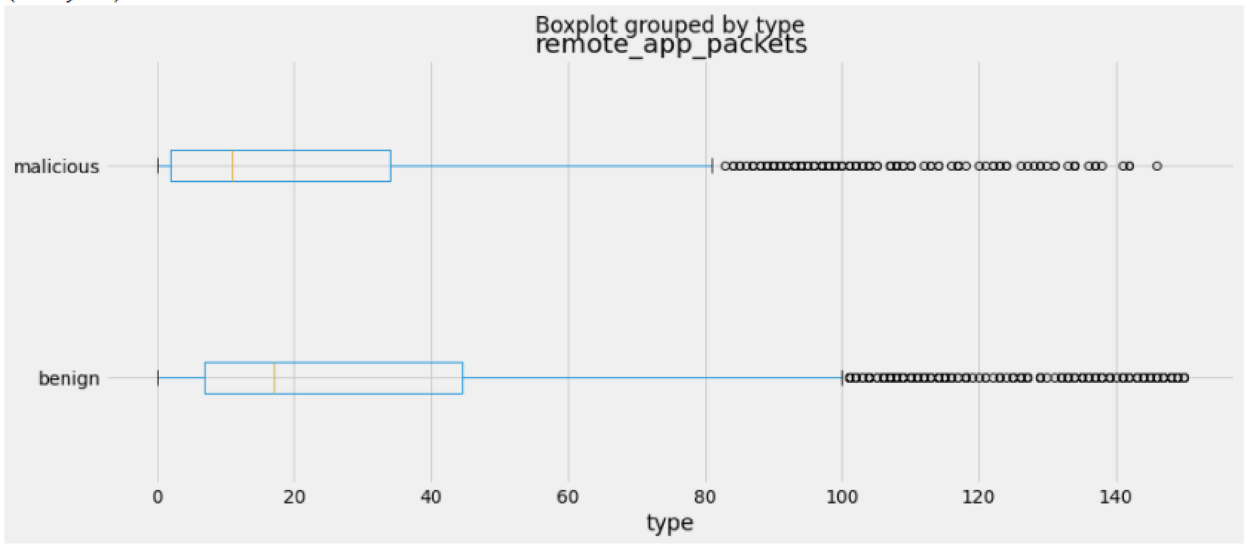


Figure 28 - box plot of observations with remote_app_packets>150 should be dropped because of high outliers

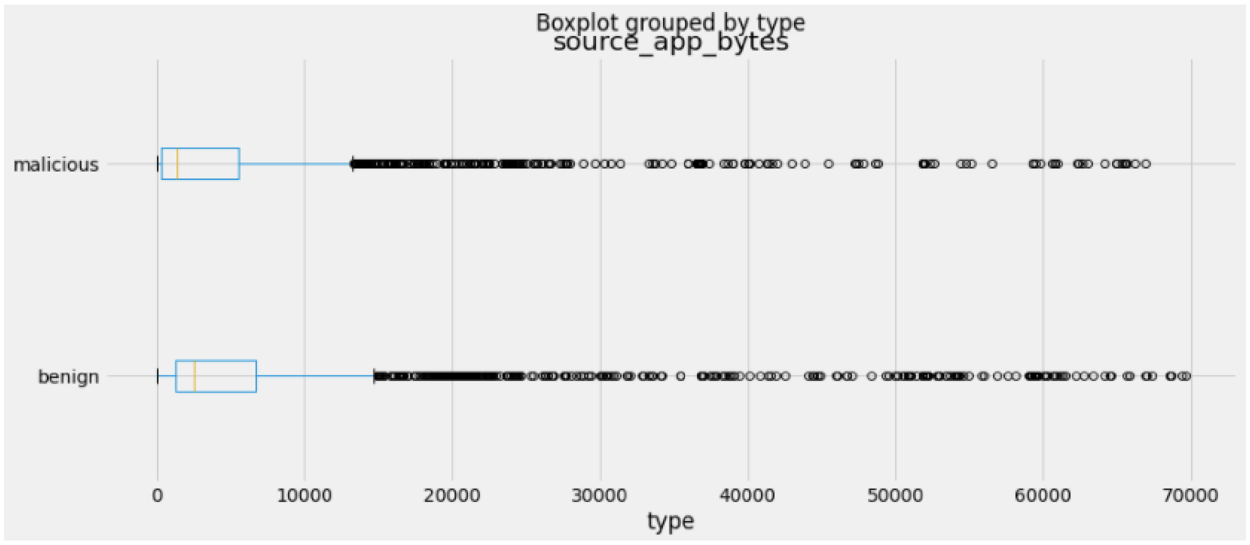


Figure 29 - box plot of observations with source_app_packets>70000 should be dropped because of high outliers

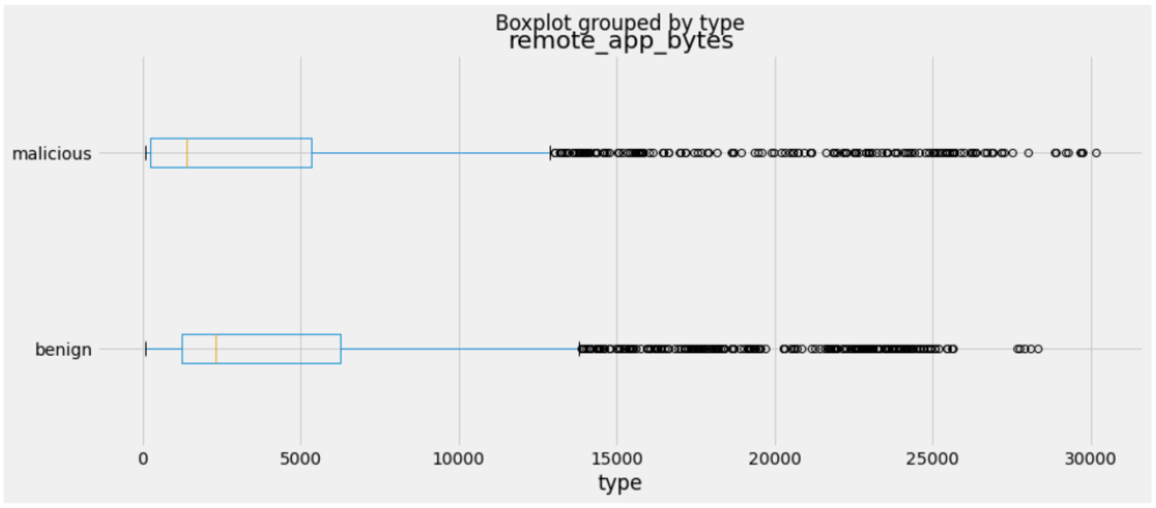


Figure 30 - box plot of observations with remote_app_packets>30000 should be dropped because of high outliers

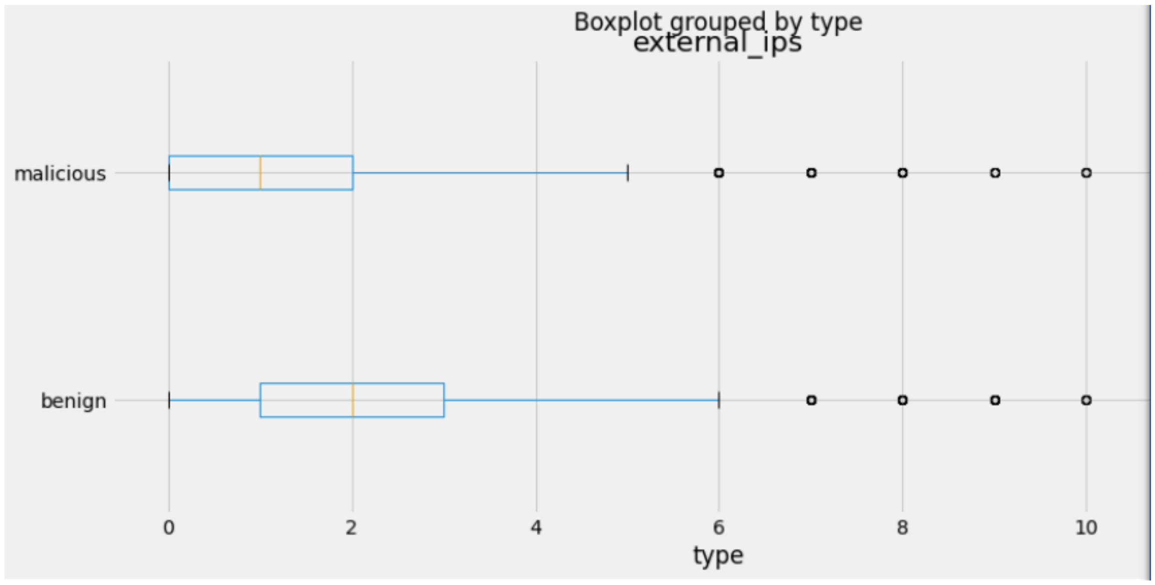


Figure 31 - box plot of observations with external_ips>14 should be dropped because of high outliers

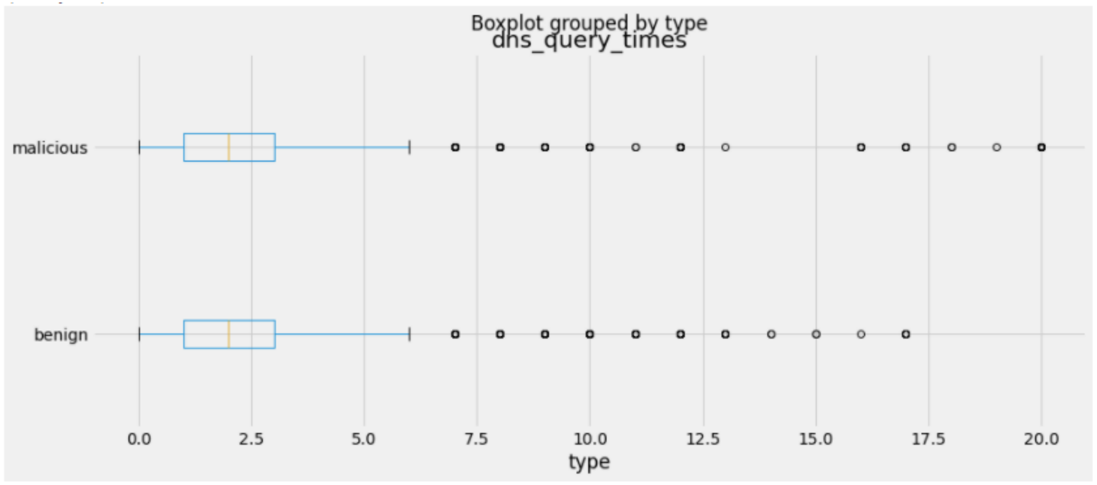


Figure 32 - box plot of observations with dns_query_times>20 should be dropped because of high outliers