Table of Contents

1.Introduction

1.1 Project details

We designed a Machine Learning based malware predictor that contains some cyber security aspects analysis for Microsoft windows OS using kaggle for dataset. 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, Microsoft takes this problem very seriously and is deeply invested in improving security.

As one part of their overall strategy for doing so, Microsoft is challenging the data science community to develop techniques to predict if a machine will soon be hit with malware. As with their previous, Malware Challenge (2015), Microsoft is providing Kagglers with an unprecedented malware dataset to encourage open-source progress on effective techniques for predicting malware occurrences. The challenge faced:

1. Large Dataset
2. Missing Values
3. Categorical Feature Encoding
4. Feature Engineering
5. Non-stationary Features — Adversarial Validation
6. Testing Metric: Area Under Curve (AUC)
7. Primary ML Model - Explore different gradient boosting framework Eg. XGboost,LightGBM etc. among chose model which gives best performance,to train the model

1.1.1. Large Dataset

The size of the training and testing data is 9 million and 8 million rows, respectively. There are 81 features in total, with 52 being categorical, 23 of which are encoded numerically to protect the privacy of the information. The train.csv and test.csv are two main files with following information:

Train.csv :- This file contains 8921483 entries where each entry corresponds to a machine which is uniquely identified by a MachineIdentifier and HasDetections is the ground truth and indicates that Malware was detected on the machine. And contains 83 columns including MachineIdentifier and HasDetections using which model is to be trained. Here features, columns and variables are analogous with each other

Test.csv :- This file contains 7853253 entries. And contains 82 columns including MachineIdentifier except HasDetections

Each malware file has an identifier, a 20 character hash value uniquely identifying the file, and a class label, which is an integer representing one of the 9 family names to which the malware may belong (Table 1.). Taking into consideration the large dataset, several factors helped reduce the burden of working with large datasets. To start, the small amount of data was considered using sample() function. Since the original dataset took time to load we considered working on 1% of the original dataset (i.e. around 89,000 rows) for two models. While the whole dataset was considered for Light Gradient Boosting Machine. The sample() helps to choose particular fraction of data randomly, following snapshots of code demonstrate the same:

Family Name	# Train Samples	Type
Ramnit	1541	Worm
Lollipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownloader
Kelihos_ver1	398	Backdoor
Obfuscator.ACY	1228	Any kind of obfuscated malware
Gatak	1013	Backdoor

Table 1: Malware families in the dataset

```
[3]  data.shape
     ✓ 0.1s
...  (8921483, 83)
```

Randomly selecting 1% of original train dataset

```
[4]  df= data.sample(frac =.01)
     ✓ 55.3s
```

```
[5]  df.shape
     ✓ 0.8s
...  (89215, 83)
```

Secondly, we explicitly mentioned the data types for example switching from float64 to float32. A function was called under name to reduce memory usage. The following snapshots of code demonstrate how to quickly optimize a Pandas dataframe to data types that reduce memory footprint

```
dtypes = {
        'MachineIdentifier':                                    'category',
        'ProductName':                                          'category',
        'EngineVersion':                                        'category',
        'AppVersion':                                           'category',
        'AvSigVersion':                                         'category',
        'IsBeta':                                               'int8',
        'RtpStateBitfield':                                     'float16',
        'IsSxsPassiveMode':                                     'int8',
        'DefaultBrowsersIdentifier':                            'float16',
        'AVProductStatesIdentifier':                            'float32',
        'AVProductsInstalled':                                  'float16',
        'AVProductsEnabled':                                    'float16',
        'HasTpm':                                               'int8',
        'CountryIdentifier':                                    'int16',
        'CityIdentifier':                                       'float32',
        'OrganizationIdentifier':                               'float16',
        'GeoNameIdentifier':                                    'float16',
        'LocaleEnglishNameIdentifier':                          'int8',
        'Platform':                                             'category',
        'Processor':                                            'category',
        'OsVer':                                                'category',
        'OsBuild':                                              'int16',
        'OsSuite':                                              'int16',
        'OsPlatformSubRelease':                                 'category',
        'OsBuildLab':                                           'category',
        'SkuEdition':                                           'category',
        'IsProtected':                                          'float16',
        'AutoSampleOptIn':                                      'int8',
        'PuaMode':                                              'category',
        'SMode':                                                'float16',
        'IeVerIdentifier':                                      'float16',
        'SmartScreen':                                          'category',
        'Firewall':                                             'float16'
```

```
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage(deep=True).sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage(deep=True).sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(end_mem, 100 * (start_mem - end_mem) / start_mem))
    return df
```

[6]

```
train = reduce_mem_usage(train)
```

```
Mem. usage decreased to 22.57 Mb (1.5% reduction)
```

1.1.2. Missing values

Oftentimes datasets contain missing values due to a variety of reasons. An example would be — a company was not actively collecting information until a certain business unit pointed out that it may be a good piece of information to collect, study and perhaps use it to improve or build a ML model. Or it could be — information just does not exist for that particular data point. An example may be not filling in optional information in a fill-able form. First to figure out which rows are missing, a simple way to summarize all missing values is using .isnull() within the Pandas library.

```
train.isnull().sum()
```

[13]

```
MachineIdentifier                          0
ProductName                                0
EngineVersion                              0
AppVersion                                 0
AvSigVersion                               0
                                        ...
Census_IsPenCapable                        0
Census_IsAlwaysOnAlwaysConnectedCapable  723
Wdft_IsGamer                            3125
Wdft_RegionIdentifier                   3125
HasDetections                              0
Length: 83, dtype: int64
```

```
test.isnull().sum()
```

[14]

```
MachineIdentifier                          0
ProductName                                0
EngineVersion                              0
AppVersion                                 0
AvSigVersion                               0
                                        ...
Census_IsTouchEnabled                      0
Census_IsPenCapable                        0
Census_IsAlwaysOnAlwaysConnectedCapable  468
Wdft_IsGamer                            1590
Wdft_RegionIdentifier                   1590
Length: 82, dtype: int64
```

Since the dataset consisted of many NA values one way was to ditch the values since they do not provide any information that may be useful for the analysis. In order to understand the percentage of missing values of column as well as one category value a function was written. All the columns with high NA rate (70%) threshold and high one category (90%) values were removed. In below snapshots good_cols is dataframe which consists of filtered columns.

	Feature	type	Unique_values	Percentage of missing values	Percentage of values in the biggest category
28	PuaMode	category	1	99.980945	99.980945
41	Census_ProcessorClass	category	3	99.602085	99.602085
8	DefaultBrowsersIdentifier	float16	242	95.155523	95.155523
68	Census_IsFlightingInternal	float16	1	83.090288	83.090288
52	Census_InternalBatteryType	category	20	70.907359	70.907359
...	...	...	...	...	...
1	ProductName	category	3	0.000000	98.919464
45	Census_HasOpticalDiskDrive	int8	2	0.000000	92.279325
51	Census_PowerPlatformRoleName	category	8	0.000000	69.172224
54	Census_OSVersion	category	263	0.000000	15.878496
82	HasDetections	int8	2	0.000000	50.381662

83 rows × 5 columns

```
good_cols = list(train.columns)

for col in train.columns:

    # remove columns with high NA rate
    na_rate = train[col].isnull().sum() / train.shape[0]

    # remove columns with high Unbalanced values rate
    unbalanced_rate = train[col].value_counts(normalize=True, dropna=False).values[0]

    if na_rate > na_rate_threshold:
        good_cols.remove(col)
    elif unbalanced_rate > unbalanced_feature_rate_threshold:
        good_cols.remove(col)
```

Other challenges are discussed in following sections

1.2 Purpose

Malware is intrusive-software that is designed to damage and destroy computers and computer systems and Malware is a contraction for "malicious software." Examples of common malware include viruses, worms, Trojan viruses, spyware, adware, and ransomware. Malware prediction is one of the important steps in the security of computer systems.Along with advancement of technology anti-malware Software Industries receives a massive number of malware pirated files to be examined. However, currently used signature-based methods are unable to provide accurate prediction of zero day attacks. The dark world hackers are using them to lure into systems through the points mentioned in the vulnerability databases. Hence, it is highly necessary to predict the malware at an early stage to avoid further loss. That's why Machine Learning based malware prediction arises.The objective of the project work predicts a computer driven system's chances of getting attacked by various malwares in the base level in the time of manufacturing of the System based on Windows Operating System and the device. That helps billions of machines from damage before it happens.The objective of the project is prediction of a computer driven system's chances of getting attacked by various malwares in the base level in the time of manufacturing of the System based on different specification of the Operating System and the software along with hardware component of machine.That helps billions of machines from damage before it happens.

1.3 Scope

In the past few years, the malware industry has grown so rapidly that the syndicates invest heavily in technologies to evade traditional protection, forcing the anti-malware groups/communities to build more robust software to detect and terminate these attacks. The major part of protecting a computer system from a malware attack is to identify whether a given piece of file/software is malware.

1.4 Objective

The goal is to predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine. The telemetry data containing these properties and the machine infections was generated by combining heartbeat and threat reports collected by Microsoft's endpoint protection solution, Windows Defender. Each row in this dataset corresponds to a machine, uniquely identified by a MachineIdentifier. HasDetections is the ground truth and indicates that Malware was detected on the machine. Using the information and labels in train.csv, you must predict the value for HasDetections for each machine in test.csv.

1.5 Technology & Literature Review

1.5.1. Software used:

1. Python
2. Jupyter NoteBook
3. Numpy
4. Pandas
5. Matplotlib
6. Seaborn
7. Time
8. Visual studio

1.5.2. Hardware used (Laptop features are mentioned):

1. GPU (Nvidia RTX)
2. Intel i5 7th Gen processor
3. RAM 128GB

1.5.3. Literature Review:

Since the end of the competition in April 2015, more than 50 research papers and thesis works have cited the competition and the dataset. Among the citations, several papers are not in English, which we are unable to read [1, 2, 3, 4]. The remaining articles can be divided into two principal classes. Few of the papers referenced the challenge to either perform an abstract comparison or highlight the importance of machine learning for malware classification in industry, where the size of data is huge [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]. Whereas few papers partial or complete evaluation on the dataset to verify the effectiveness and/or efficiency of their proposed approach for various tasks. The diversity of the contributions has made the dataset a benchmark for various tasks, helping researchers provide a standard for evaluation and comparison.

2. Project Management

2.1 Feasibility Study

2.1.1 Technical Feasibility:

As the project title suggests the target market for our implemented Machine Learning model are the computers operating over Windows OS. There are some antivirus softwares that actually detect malwares but there is no device in the market that warns the client about potential hazards in their computers. This helps the client to increase their security by removing all the potential hazards.

The technical resources actually required for our malware prediction are: a perfectly working laptop installed with python, installed all python dependencies, dataset, and different Machine Learning Models. We had all the available technologies required for the project and also we were skilled enough to implement the project title using the technologies.

As the market for the WIndows OS is very large and majority of the organizations depend on Windows OS this project has a large scope for early profitability. Some expansion ideas include the implementation of this product on Mac OS and Linux. Also the budget required for the project is very minimal as we are using the open source dataset available over the web.

2.1.2 Time Schedule Feasibility:

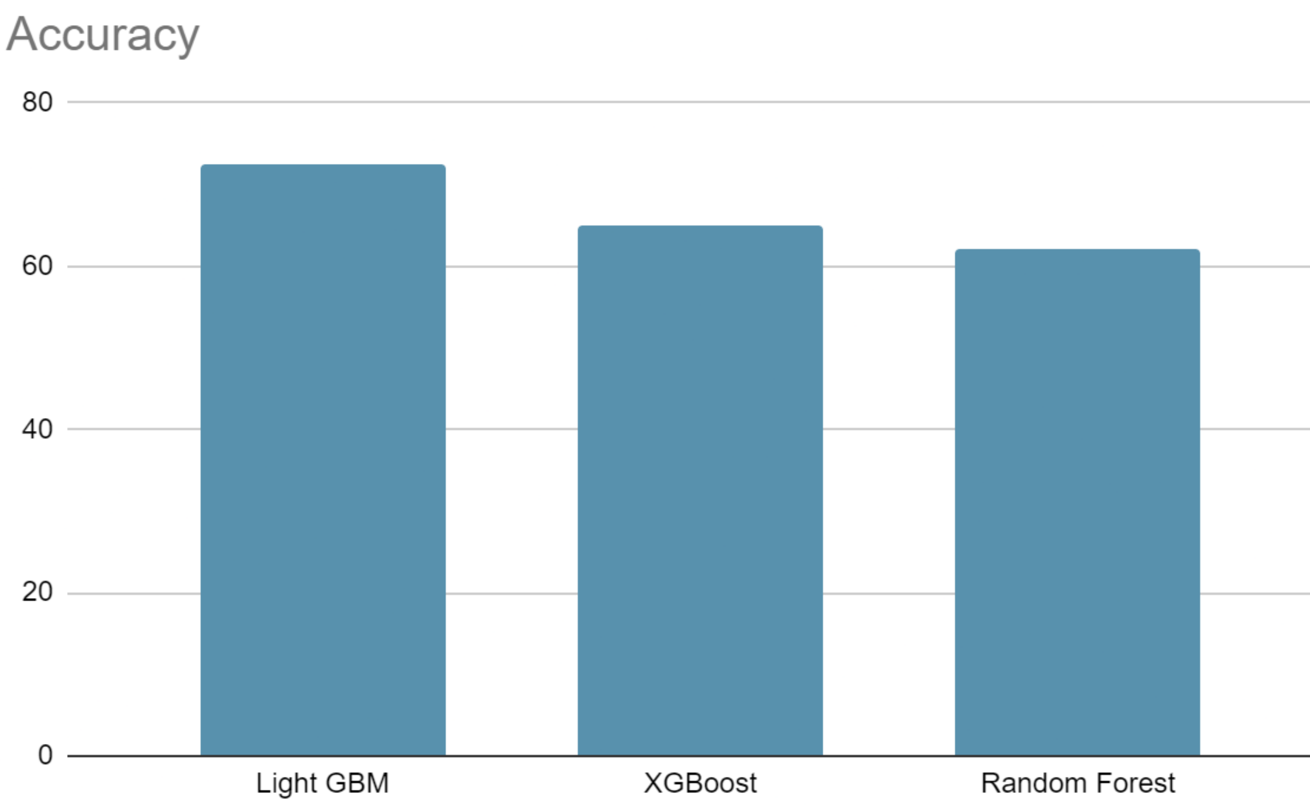
With a highly skilled team of people, the target of achieving good accuracy for the project is very easy and efficient. The major timeline of this project is as follows:

Goal	Deadline
Start of the Project	30/05/2022
Gathering Data	03/06/2022
Analyzing the Dataset	07/06/2022
Preprocessing the Dataset	15/06/2022
Analyzing the preprocessed Data	20/06/2022
Implementing ML models	30/06/2022
Increasing accuracy of the obtained results	05/07/2022

Final touch up to the model	08/07/2022
End of the Project	11/07/2022

2.1.3 Operational Feasibility:

Accuracy and precision are the most important factors in any project. So we have implemented 3 different models and finally used the model with the highest accuracy. The respective accuracies of the three models are:



2.1.4 Implementation Feasibility:

As mentioned earlier in this report we have the required technology and have apt skills for the implementation of this project. The implementation of this project can solve one of the major problems of detecting vulnerabilities and thus taking actions according to those vulnerabilities using the available resources.

The designing of the antivirus softwares would be easier with the implementation of this project as then the organization would only have to look into the vulnerabilities of the Windows OS. The revenue generation of this model would also be very easy.

Thus, concluding the feasibility study of the project we would like to mention that the model built by us would be an innovation in the anti-malware industry.

2.2 Project Planning

2.2.1 Project Development Approach and Justification:

The approach selected by us is a very simple and conventional approach used for any Machine Learning project. According to our approach we have first searched for the open source dataset which we were going to use for training the Machine Learning models. After fetching the dataset we conducted the Exploratory Data Analysis technique for getting apt knowledge about the dependencies in the dataset.

After analyzing the dataset we processed the data according to the needs of the models to be implemented and removed all the null values in the dataset. We then conducted the analysis of the processed data again using the Exploratory data analysis method to know the dataset well enough and the dependencies of the attributes in the dataset. Before training the model we also performed the feature engineering on the processed dataset and finally encoded with label encoding and frequency encoding and then proceeded further.

Then we finally trained the models one after the other to check the maximum accuracy. The accuracies obtained from each model was as follows:

Model	Accuracy
Light GBM	73%
XGBoost	65%
Random Forest	62%

Thus, the final model selected by us is Light GBM as it had the highest accuracy which would be able to predict 73% of all the vulnerabilities.

The justification for the above approach is that the use of EDA helped us to analyze the data in depth. Also performing feature engineering led to an increase in accuracy of the Machine Learning models. Thus higher the accuracy the better is the product.

2.2.2 Milestone & Deliverables:

The major milestones of the project include the completion of analysis of the data, preprocessing the data, analyzing the processed data, applying feature engineering, encoding the dataset and finally implementing the models and then attempting to increase the accuracy obtained from the implemented models. The deadlines of each and every milestone is mentioned earlier in the report.

The major deliverables of the project include a fully functional and accurate Machine learning model used to predict the vulnerabilities in a Windows OS and thus making the process of building antivirus software easier according to the system vulnerabilities.

2.2.3 Roles & Responsibilities:

The major roles and responsibilities in the team were that each and everyone on the team would contribute equally to the project with the skill they are proficient of. The brainstorming part was covered by each and everyone on the team including Charmi, Shubham and Vedant. Then the various aspects of EDA were covered by all three of us. All three of us distributed the model as:

- Charmi: XGBoost
- Shubham: Light GBM
- Vedant: Random Forest

Thus, the preprocessing of data was conducted individually according to the assigned Machine Learning model leading to the EDA of the processed data. Thus, finally we mutually decided the final model to be used with highest accuracy.

The major responsibility of the project was the documenting process which included the weekly reports and the final thesis reports which were divided amongst us equally.Thus each and every individual on the team contributed equally as per their proficiency. None of the team members was burdened with a lot to do.

2.2.3 Group Dependencies:

As mentioned in the roles and responsibilities section each and every team member has contributed equally to the project. The major dependency was at the time of evaluating the accuracies of different models and deciding the final model to be used. Also the thesis report had a dependency on each and every person as everyone had implemented different models for the project.The testing was conducted by the people except who had actually implemented the Machine Learning model. Thus, testing of model was conducted as follows:

- Light GBM: Charmi, Vedant
- XGBoost: Shubham, Vedant
- Random Forest: Charmi, Shubham

2.3 Project Scheduling:

This project took exactly 6 weeks to complete. Final output was to be submitted to BISAG-N on 11th July 2022 as decided by the Faculty Coordinators.The timely submission of weekly reports and all other documentation was made sure by us. All the milestones were completed within the time limit given by the faculty coordinator.

Start of the project	**30/05/2022**
Milestone 1	**04/06/2022**
Brainstorming	02/06/2022
Finalizing the technologies to be used	03/06/2022
Fulfilling all the requirements & dependencies	04/06/2022
Milestone 2	**20/06/2022**
Data Analysis	07/06/2022
Processing data and its analysis	17/06/2022
Feature engineering	20/06/2022
Milestone 3	**05/07/2022**
Training Models	30/06/2022
Increasing Accuracy	05/07/2022
Milestone 4	**10/07/2022**
Deciding final Model	10/07/2022
End of the Project	11/07/2022

3. System Requirement Study

3.1 Study of Current System

In the present scenario, the market does not have any softwares to predict the potential vulnerabilities of a Windows OS. The system we are thinking of designing gives a list of potential hazards which can cause the malware to intervene with the computers working on Windows OS. Thus, the current system is a normal antivirus system which can conduct a scan over the machine and find the malwares which are present in the systems.

Some examples of current antivirus softwares that are present in the market are: QuickHeal Antivirus, Avast, McAfee, etc.

3.2 Problems and Weaknesses of current system

As we all know that with emerging technology, hackers are also evolving their attacks and optimizing the malwares which are sometimes unable to detect using the antivirus softwares that are currently available in the market. Sometimes these antivirus softwares show false positives which can remove some very important files from our device. The system we have designed just informs the user about the potential vulnerabilities in their device which makes our system much more useful when it comes to losing files.

Our system can also be used in a way where an antivirus is designed according to the potential vulnerabilities of a device and thus can revolutionize the anti-malware market.

3.3 User Characteristics

The potential users which we have aimed for this project can be an organization, a single user or anyone who uses a Windows OS based computer. It can also be used to find vulnerabilities in a network of computers based on Windows OS. According to our analysis the market is very large and has a lot of scope with a minimum number of competitors. The majority of users would be some organizations who devise their own anti malware softwares as they could get a hint of what all they need to keep in mind while building their own software.

3.4 Hardware and Software Requirements:

Hardware Requirements: Majorly, there is no requirement for any special hardware as it is a completely software based project.

Software Requirements:

- Minimum RAM: 4GB (Due to large dataset)
- No Graphic card needed
- Python and required libraries
- Editor like VS Code

Thus, this project can be achieved using minimum requirements

3.5 Constraints

3.5.1 User Interface:

As of now, we are not making any user interface for the project. We have created the model and thus if the user wants to check for his vulnerabilities then he has to put all his in a CSV file and then predict using the model built by us.

3.5.2 Communications Interface:

There are no communications interfaces required as we are not hosting our project over the web and the only communication required is internal for calling the libraries, dataset, etc. Thus, there is no need for a communications interface.

3.5.3 Hardware Interface:

There is no hardware interface required as it is completely a software based solution to the problem statement.

3.5.4 Criticality of the Application:

The application is very rare as up to our knowledge no antivirus software detects the potential vulnerabilities.

3.5.5 Safety and Security Considerations:

The software built by us is completely safe as it does not involve any communication over the internet which can cause any kind of damage.

3.6 Assumptions & Dependencies

3.6.1 Assumptions:

- The open source data is correct
- The computer to be tested is being operated on Windows OS
- The computer is to be tested is installed with python and required libraries

3.6.2 Dependencies:

- Python
- Libraries like Pandas, Numpy, Matplotlib,sklearn
- Visual Studio Code or any other editor
- Jupyter Notebook
- Various Machine Learning Models

4. Requirement of Proposed System

4.1 Main Module of the System

1. EDA - Exploratory Data Analysis
2. Feature Engineering
3. Frequency and Label encoding in LGBM
4. Model Implementation

4.2 Module Descriptions

1) EDA - Exploratory Data Analysis :-

➔ Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

➔ Our EDA is categorized in following Parts:-

I. EDA of Raw Train and Test Data
II. Feature Type Analysis
III. EDA of Pre-processed Data
   A. Univarient Graphical EDA
      1. Univariate Graphical EDA of Categorical Type Data
      2. Univariate Graphical EDA of Numerical Type Data
   B. Corelation Analysis
      1. For Numerical Features
      2. For Whole Dataset
   C. Multivariate Graphical EDA of Most Co-related Feature

2) Feature Engineering :-

➔ Feature engineering is a machine learning technique that leverages data to create new variables that aren't in the training set. It can produce new features for both supervised and unsupervised learning, with the goal of simplifying and speeding up data transformations while also enhancing model accuracy.

➔ Feature Engineering is implemented in Light GBM and XG-boost Model.

➔ Feature Engineering is done by the Good columns(features) those are the filtered by pre-processing.

➔ One by one features are taken and according to its feature type transformation is applied.

➔ Ex:- `df['AppVersion_3'] = df['AppVersion'].apply(lambda x: x.split('.')[3]).astype('category')`

3) Frequency and Label encoding in LGBM:-

- ➔ Count or frequency encoding: Replace the categories by the count of the observations that show that category in the dataset. Similarly, we can replace the category by the frequency -or percentage- of observations in the dataset.
- ➔ We usually deal with datasets that contain multiple labels in one or more than one columns. These labels can be in the form of words or numbers. To make the data understandable or in human-readable form, the training data is often labelled in words. So for this reason we applied Label Encoding
- ➔ Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form.
- ➔ Light GBM algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

4) Model Implementation:-

- ➔ We implemented Three Models Light GBM, XG-Boost and Random Forest.
- ➔ These are covered in detail in chapter no 7 Testing

4.3 Features of New System

- ❖ Features of System as per below:

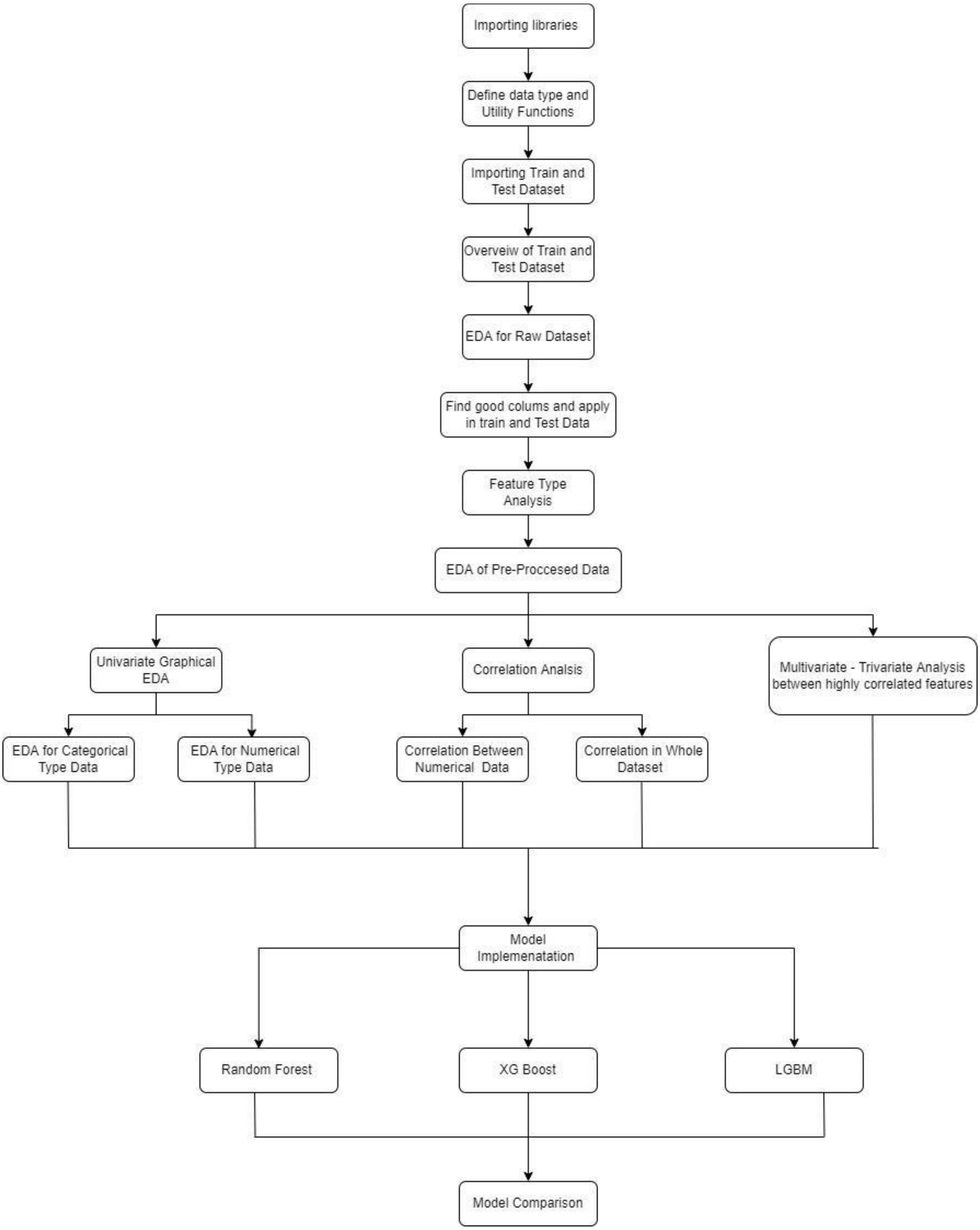
1) Identify the missing value and percentage of packet missing
2) Identify the Uniques value according to the dataset variable/features
3) Identify the Skewness of the features
4) Identify the Null value count for all features of dataset
5) Identify the good feature which are use in various module
6) Distinguish feature by their types and give statistics about the dataset features
7) Give a proper analysis of Categorical Type Data in followings ways:
    a) Top 10 most occurred categories for the categorical feature
    b) Accuracy and F1 score
    c) Univarient Plot
    d) Bivarient Plot against the Target variable/feature
    e) Confussion Matrix
    f) Probability of Detecting a Malware vs Paticular variable/feature
    g) Pie chart for categories distribution

   h) Bar Graph for fraction of infected machines in each category
   i) Frequency Graph of malware for that particular feature

8) Give a proper analysis of Numerical Type Data in followings ways:
   a) Top 10 most occurred categories for the categorical feature
   b) Min value, Max value, NaN values, Number of unique values, Mean value, Variance value
   c) Accuracy and F1 score
   d) Univarient Plot
   e) Bivarient Plot against the Target variable/feature
   f) Confussion Matrix
   g) Probability of Detecting a Malware vs Paticular variable /feature
9) Corelation Analysis for whole dataset
10) Identify the most positive and negative correlated feature among the whole dataset
11) Multivariate - Trivariate Analysis between highly correlated features and against the target variable
   a) Accuracy Score and F1 score between those variable/ feature
   b) Logistic Predictions between those variable/ feature
   c) Scatterplot between those variable/ feature
   d) Confusion matrix for two features and against the target variable after fitting a logistic regression model
12) Give the Prediction of input dataset according to the Various model like Light GBM, XG-boost, Random Forest
13) Give the Histogram graph of predicted results

5. System Design

5.1 System Architecture Design



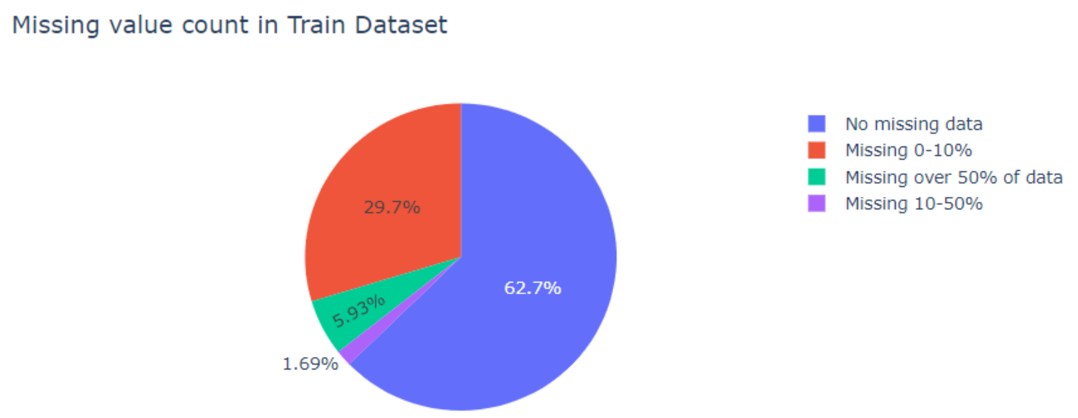
5.2 Database Design

5.2.1 Train Dataset Design:-

- ➔ Train Dataset number of records: 8921483
- ➔ Train Dataset number of columns: 83

	count	mean	std	min	25%	50%	75%	max
IsBeta	8921483.0	7.509962e-06	2.740421e-03	0.000000	0.000000	0.0	0.000000e+00	1.00000
RtpStateBitfield	8889165.0	NaN	0.000000e+00	0.000000	7.000000	7.0	7.000000e+00	3.50000
IsSxsPassiveMode	8921483.0	1.733378e-02	1.305118e-01	0.000000	0.000000	0.0	0.000000e+00	1.00000
DefaultBrowsersIdentifier	433438.0	NaN	NaN	1.000000	788.000000	1632.0	2.372000e+03	3.21200
AVProductStatesIdentifier	8885262.0	4.784002e+04	1.403237e+04	3.000000	49480.000000	53447.0	5.344700e+04	7.05070
AVProductsInstalled	8885262.0	NaN	0.000000e+00	0.000000	1.000000	1.0	2.000000e+00	7.00000
AVProductsEnabled	8885262.0	NaN	0.000000e+00	0.000000	1.000000	1.0	1.000000e+00	5.00000
HasTpm	8921483.0	9.879711e-01	1.090149e-01	0.000000	1.000000	1.0	1.000000e+00	1.00000
CountryIdentifier	8921483.0	1.080490e+02	6.304706e+01	1.000000	51.000000	97.0	1.620000e+02	2.22000
CityIdentifier	8596074.0	8.126650e+04	4.892339e+04	5.000000	36825.000000	82373.0	1.237000e+05	1.67962
OrganizationIdentifier	6169965.0	NaN	0.000000e+00	1.000000	18.000000	27.0	2.700000e+01	5.20000
GeoNameIdentifier	8921270.0	NaN	NaN	1.000000	89.000000	181.0	2.670000e+02	2.96000
LocaleEnglishNameIdentifier	8921483.0	2.790453e+01	6.560791e+01	-128.000000	-29.000000	58.0	7.500000e+01	1.27000
OsBuild	8921483.0	1.571997e+04	2.190685e+03	7600.000000	15063.000000	16299.0	1.713400e+04	1.82440
OsSuite	8921483.0	5.751534e+02	2.480847e+02	16.000000	256.000000	768.0	7.680000e+02	7.84000
IsProtected	8885439.0	NaN	0.000000e+00	0.000000	1.000000	1.0	1.000000e+00	1.00000
AutoSampleOptIn	8921483.0	2.891896e-05	5.377558e-03	0.000000	0.000000	0.0	0.000000e+00	1.00000
SMode	8383724.0	0.000000e+00	0.000000e+00	0.000000	0.000000	0.0	0.000000e+00	1.00000
IeVerIdentifier	8862589.0	NaN	NaN	1.000000	111.000000	117.0	1.370000e+02	4.29000
Firewall	8830133.0	NaN	0.000000e+00	0.000000	1.000000	1.0	1.000000e+00	1.00000
UacLuaenable	8910645.0	1.302773e+01	9.867771e+03	0.000000	1.000000	1.0	1.000000e+00	1.67772
Census_OEMNameIdentifier	8826005.0	NaN	NaN	1.000000	1443.000000	2102.0	2.668000e+03	6.14400

```
There are:
74 columns without missing values
35 columns with less than 10% of missing values
 2 columns withmissing values between 10% and 50%
 7 columns with more than 50% of missing values
```

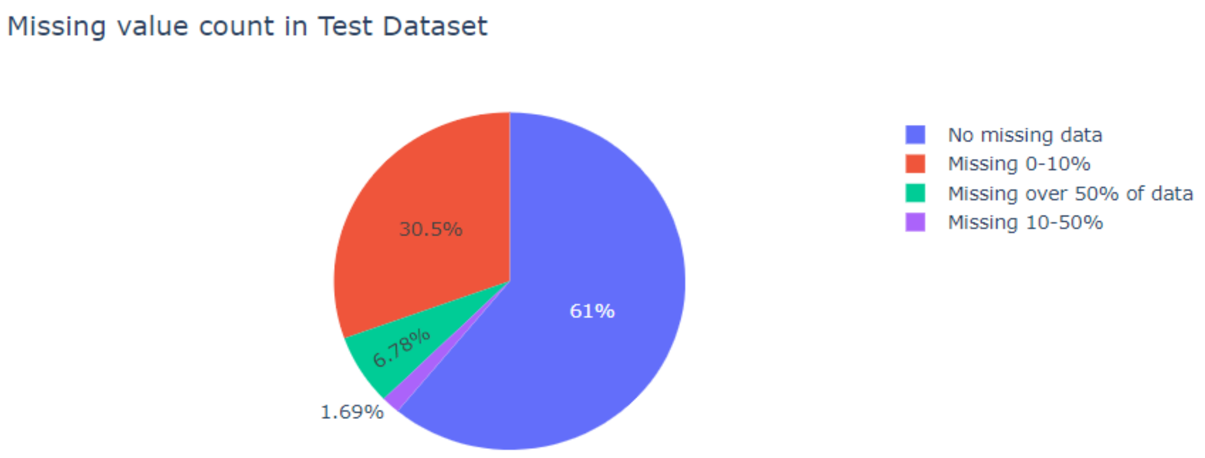


5.2.2 Test Dataset Design:-

- ➔ Test Dataset number of records: 7853253
- ➔ Test Dataset number of columns: 82

	count	mean	std	min	25%	50%	75%	max
IsBeta	7853253.0	5.857445e-06	2.420209e-03	0.000000	0.000000	0.0	0.000000e+00	1.00000
RtpStateBitfield	7821031.0	NaN	0.000000e+00	0.000000	7.000000	7.0	7.000000e+00	4.00000
IsSxsPassiveMode	7853253.0	1.586807e-02	1.249651e-01	0.000000	0.000000	0.0	0.000000e+00	1.00000
DefaultBrowsersIdentifier	307119.0	NaN	NaN	1.000000	508.000000	1632.0	2.376000e+03	3.21400
AVProductStatesIdentifier	7829486.0	4.944971e+04	1.226556e+04	2.000000	53447.000000	53447.0	5.344700e+04	7.05020
AVProductsInstalled	7829486.0	NaN	0.000000e+00	1.000000	1.000000	1.0	2.000000e+00	6.00000
AVProductsEnabled	7829486.0	NaN	0.000000e+00	0.000000	1.000000	1.0	1.000000e+00	5.00000
HasTpm	7853253.0	9.917166e-01	9.063571e-02	0.000000	1.000000	1.0	1.000000e+00	1.00000
CountryIdentifier	7853253.0	1.094486e+02	6.318849e+01	1.000000	51.000000	97.0	1.640000e+02	2.22000
CityIdentifier	7661291.0	8.121283e+04	4.903231e+04	1.000000	36829.000000	82373.0	1.232060e+05	1.67962
OrganizationIdentifier	5371124.0	NaN	0.000000e+00	1.000000	18.000000	27.0	2.700000e+01	5.20000
GeoNameIdentifier	7853106.0	NaN	NaN	1.000000	89.000000	193.0	2.670000e+02	2.96000
LocaleEnglishNameIdentifier	7853253.0	2.681400e+01	6.578649e+01	-128.000000	-29.000000	56.0	7.500000e+01	1.27000
OsBuild	7853253.0	1.591664e+04	2.127204e+03	7600.000000	15063.000000	17134.0	1.713400e+04	1.82890
OsSuite	7853253.0	5.519506e+02	2.528661e+02	16.000000	256.000000	768.0	7.680000e+02	7.84000
IsProtected	7829604.0	NaN	0.000000e+00	0.000000	1.000000	1.0	1.000000e+00	1.00000
AutoSampleOptIn	7853253.0	2.062839e-05	4.541803e-03	0.000000	0.000000	0.0	0.000000e+00	1.00000
SMode	2021981.0	0.000000e+00	0.000000e+00	0.000000	0.000000	0.0	0.000000e+00	1.00000
IeVerIdentifier	7803457.0	NaN	NaN	1.000000	111.000000	137.0	1.370000e+02	4.29000
Firewall	7794781.0	NaN	0.000000e+00	0.000000	1.000000	1.0	1.000000e+00	1.00000
UacLuaenable	7845388.0	1.822908e+02	3.467205e+05	0.000000	1.000000	1.0	1.000000e+00	8.08482
Census_OEMNameIdentifier	7763707.0	NaN	NaN	1.000000	1443.000000	2102.0	2.668000e+03	6.14400
Census_OEMModelIdentifier	7757318.0	2.406951e+05	7.159188e+04	1.000000	190007.000000	248411.0	3.086900e+05	3.45499
Census_ProcessorCoreCount	7791976.0	NaN	0.000000e+00	1.000000	2.000000	4.0	4.000000e+00	2.24000
Census_ProcessorManufacturerIdentifier	7791972.0	NaN	0.000000e+00	1.000000	5.000000	5.0	5.000000e+00	1.00000

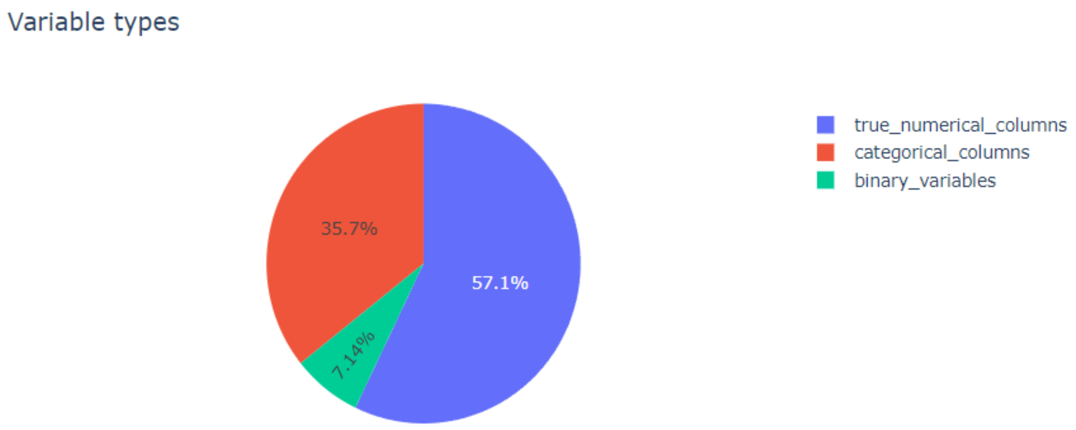
```
There are:
72 columns without missing values
36 columns with less than 10% of missing values
 2 columns withmissing values between 10% and 50%
 8 columns with more than 50% of missing values
```



5.2.3 Logical Description of Data

➢ There are 3 Type of data in our dataset

1. Categorical Columns (Count = 4)
2. Binary Variables (Count = 32)
3. True Numerical Columns (Count = 20)



➢ Highly negative correlated features:

dtype: float64

Feature 1	Feature 2	Value of Correlation
AVProductStatesIdentifier	AVProductsInstalled	-0.6329
Census_OSBuildNumber	Census_OSBuildRevision	-0.5642
OsBuild	Census_OSBuildRevision	-0.4932

➢ Highly positive correlated features:

dtype: float64

Feature 1	Feature 2	Value of Correlation
Census_OSInstallLanguageIdentifier	Census_OSUILocaleIdentifier	0.9885
OsBuild	Census_OSBuildNumber	0.9379
Census_InternalPrimaryDisplayResolutionHorizontal	Census_InternalPrimaryDisplayResolutionVertical	0.9015
Census_ProcessorManufacturerIdentifier	Census_ProcessorModelIdentifier	0.7984
CountryIdentifier	GeoNameIdentifier	0.5985
Census_ProcessorCoreCount	Census_TotalPhysicalRAM	0.5979
Census_InternalPrimaryDiagonalDisplaySizeInInches	Census_InternalBatteryNumberOfCharges	0.5297

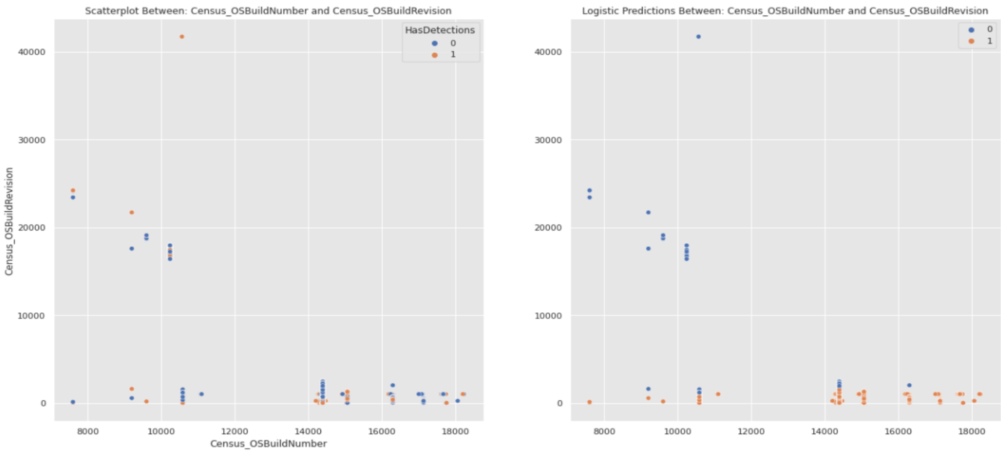
Observations

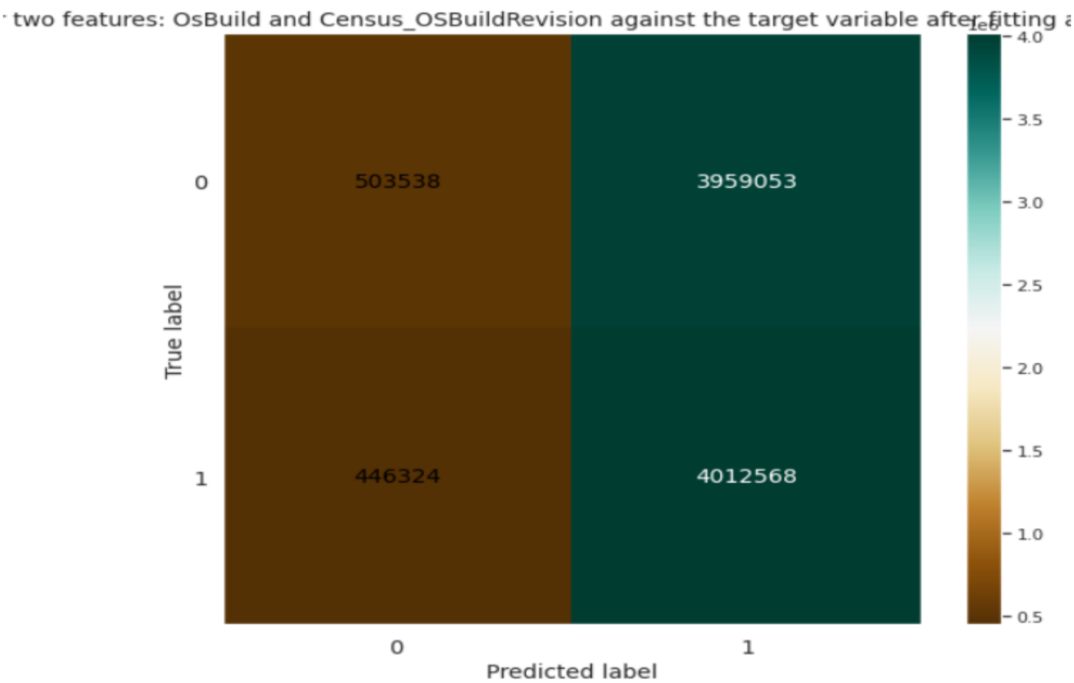
- Census_ProcessorCoreCount: Malware detection is right-skewed.
- Census_PrimaryDiskTotalCapacity: Almost symmetric.
- Census_SystemVolumeTotalCapacity, Census_TotalPhysicalRAM: Malware non-detection is right-skewed.
- Census_InternalPrimaryDiagonalDisplaySizeInInches: Malware non-detection has a long right-tail.
- Census_InternalPrimaryDisplayResolutionHorizontal: Almost symmetric.
- Census_InternalPrimaryDisplayResolutionVertical: Malware non-detection has a long right-tail.
- Census_InternalBatteryNumberOfCharges: Almost symmetric.

5.2.4 Relationship between Features

```
%%time
multivariate_plot("OsBuild", "Census_OSBuildRevision")
gc.collect()
```

```
Fitting a logistic regression model for the features OsBuild and Census_OSBuildRevision against the target variable
              precision    recall  f1-score   support

           0       0.53      0.11      0.19   4462591
           1       0.50      0.90      0.65   4458892

    accuracy                           0.51   8921483
   macro avg       0.52      0.51      0.42   8921483
weighted avg       0.52      0.51      0.42   8921483


accuracy score: 0.5062057507703596
F1 score: 0.5062057507703596
```





5.2.5 Snapshots

★ **Train and Test data null Value Count**

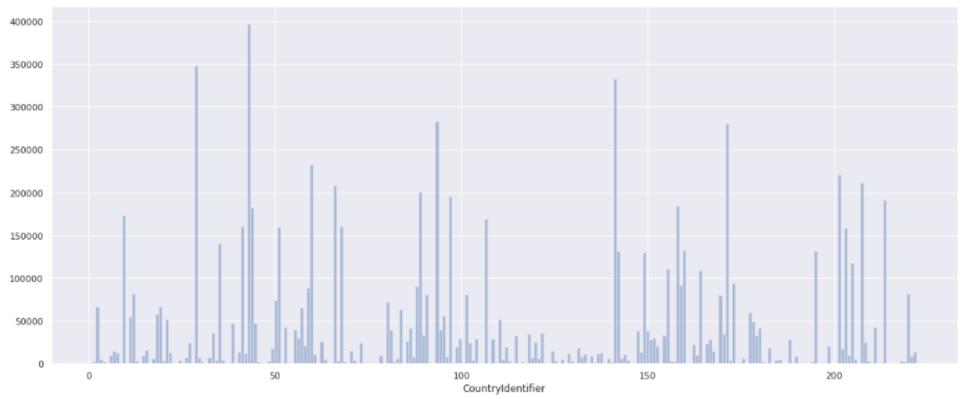
```
In [ ]: train.isnull().sum()

Out[ ]: MachineIdentifier                            0
        ProductName                                  0
        EngineVersion                                0
        AppVersion                                   0
        AvSigVersion                                 0
                                                   ...
        Census_IsPenCapable                          0
        Census_IsAlwaysOnAlwaysConnectedCapable    723
        Wdft_IsGamer                              3125
        Wdft_RegionIdentifier                     3125
        HasDetections                                0
        Length: 83, dtype: int64
```

```
In [ ]: test.isnull().sum()

Out[ ]: MachineIdentifier                            0
        ProductName                                  0
        EngineVersion                                0
        AppVersion                                   0
        AvSigVersion                                 0
                                                   ...
        Census_IsTouchEnabled                        0
        Census_IsPenCapable                          0
        Census_IsAlwaysOnAlwaysConnectedCapable    468
        Wdft_IsGamer                              1590
        Wdft_RegionIdentifier                     1590
        Length: 82, dtype: int64
```

★ **Country Identifier Frequency**

```
In [65]:
fig, ax = plt.subplots(ncols=1, nrows=1,figsize=(20,8))
ax = sns.distplot(train["CountryIdentifier"], kde=False, bins=250)

print("Number of country identifiers: " + str(train["CountryIdentifier"].nunique()))
print("The most frequent country identifier: " + str(train["CountryIdentifier"].mode()[0]))
```

```
Number of country identifiers: 222
The most frequent country identifier: 43
```



★ Feature Analaysis and Stats

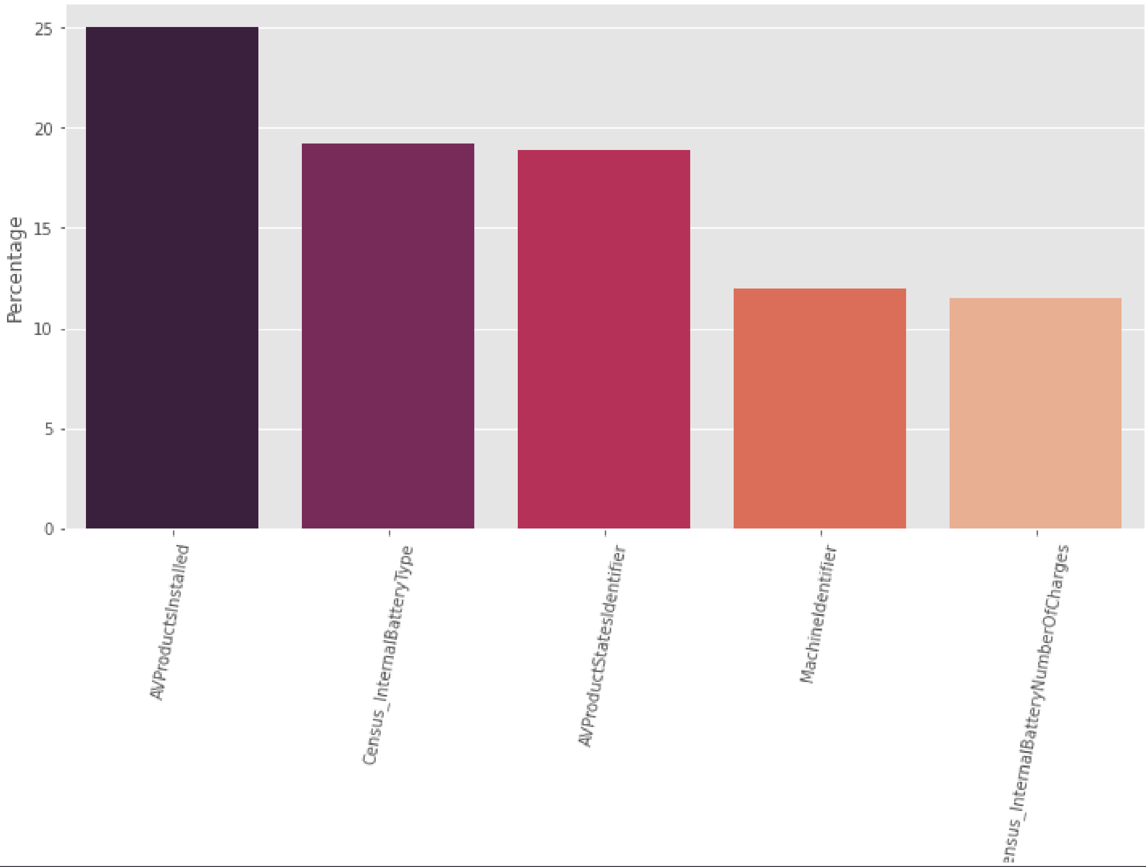
In [42]:

```
join_stats = train_stats.copy()
join_stats.columns = ['Feature', 'Train Unique values', 'Train Type']
join_stats['Test Unique values'] = 0
join_stats['Test Type'] = '???'

for index, row in join_stats.iterrows():
    for test_index, test_row in test_stats.iterrows():
        if row['Feature'] == test_row['Feature']:
            join_stats.loc[index, 'Test Unique values'] = test_row['Unique values']
            join_stats.loc[index, 'Test Type'] = test_row['Type']

join_stats['% changed'] = (1 - (join_stats['Test Unique values'] / join_stats['Train Unique values'])) * 100
join_stats = join_stats[['Feature', 'Train Unique values', 'Test Unique values', '% changed', 'Train Type', 'Test Type']]
join_stats
```

	Feature	Train Unique values	Test Unique values	% changed	Train Type	Test Type
0	MachineIdentifier	8921483	7853253	11.9737	Categorical	Categorical
26	Census_SystemVolumeTotalCapacity	536848	509175	5.1547	Numerical	Numerical
20	Census_OEMModelIdentifier	175365	167776	4.3275	Numerical	Numerical
7	CityIdentifier	107366	105817	1.4427	Numerical	Numerical
50	Census_FirmwareVersionIdentifier	50494	49811	1.3526	Numerical	Numerical
34	Census_InternalBatteryNumberOfCharges	41087	36359	11.5073	Numerical	Numerical
4	AVProductStatesIdentifier	28970	23492	18.9092	Numerical	Numerical
3	AvSigVersion	8531	9357	-9.6823	Categorical	Categorical
24	Census_PrimaryDiskTotalCapacity	5735	5797	-1.0811	Numerical	Numerical
27	Census_TotalPhysicalRAM	3446	3700	-7.3709	Numerical	Numerical
23	Census_ProcessorModelIdentifier	2583	2591	-0.3097	Numerical	Numerical
19	Census_OEMNameIdentifier	2564	2500	2.4961	Numerical	Numerical
30	Census_InternalPrimaryDisplayResolutionHorizontal	2050	2118	-3.3171	Numerical	Numerical
31	Census_InternalPrimaryDisplayResolutionVertical	1552	1570	-1.1598	Numerical	Numerical
29	Census_InternalPrimaryDiagonalDisplaySizeInInches	785	803	-2.2930	Numerical	Numerical
49	Census_FirmwareManufacturerIdentifier	712	722	-1.4045	Numerical	Numerical
14	OsBuildLab	663	672	-1.3575	Categorical	Categorical
35	Census_OSVersion	469	475	-1.2793	Categorical	Categorical
16	IeVerIdentifier	303	294	2.9703	Numerical	Numerical
9	GeoNameIdentifier	292	289	1.0274	Numerical	Numerical
38	Census_OSBuildRevision	285	294	-3.1579	Numerical	Numerical
10	LocaleEnglishNameIdentifier	252	253	-0.3968	Numerical	Numerical
6	CountryIdentifier	222	222	0.0000	Numerical	Numerical
37	Census_OSBuildNumber	165	156	5.4545	Numerical	Numerical
43	Census_OSUILocaleIdentifier	147	139	5.4422	Numerical	Numerical
2	AppVersion	110	120	-9.0909	Categorical	Categorical

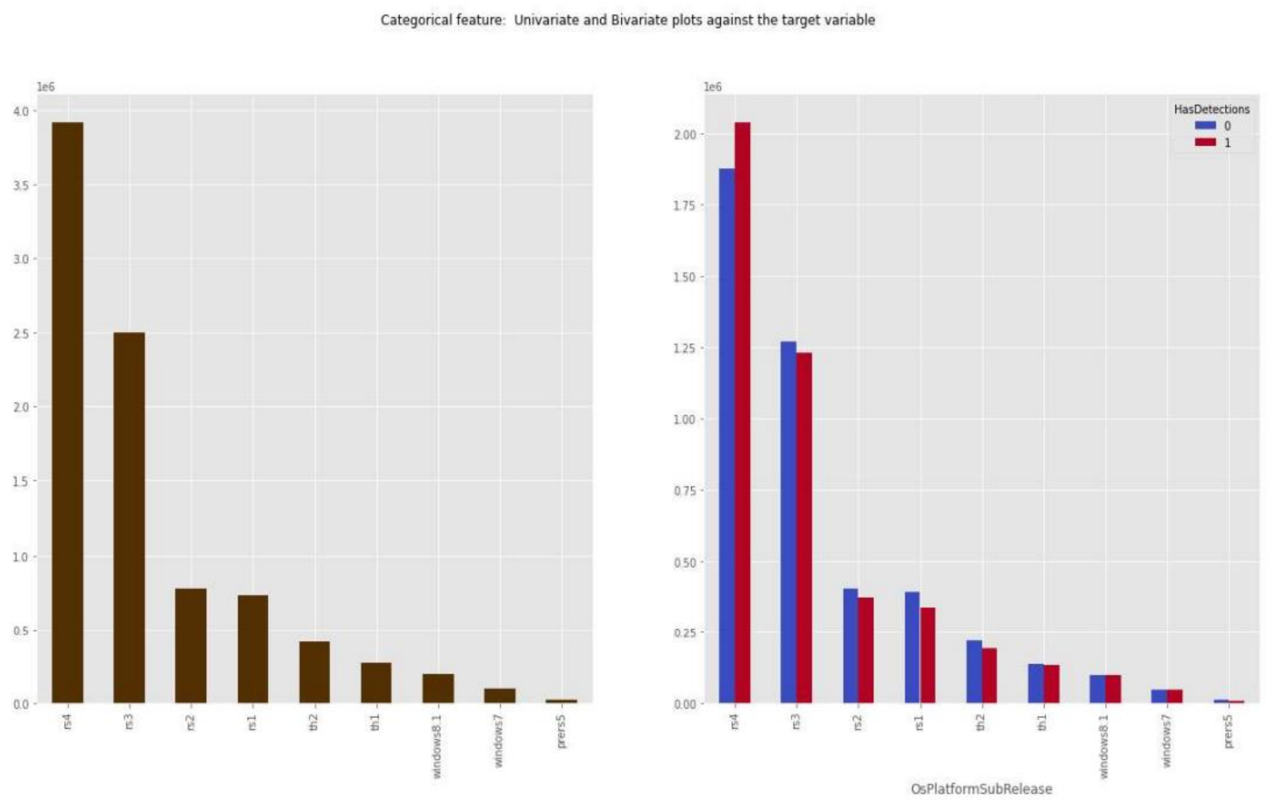
★ **Features that have at least 10% more or less categories on the train set**

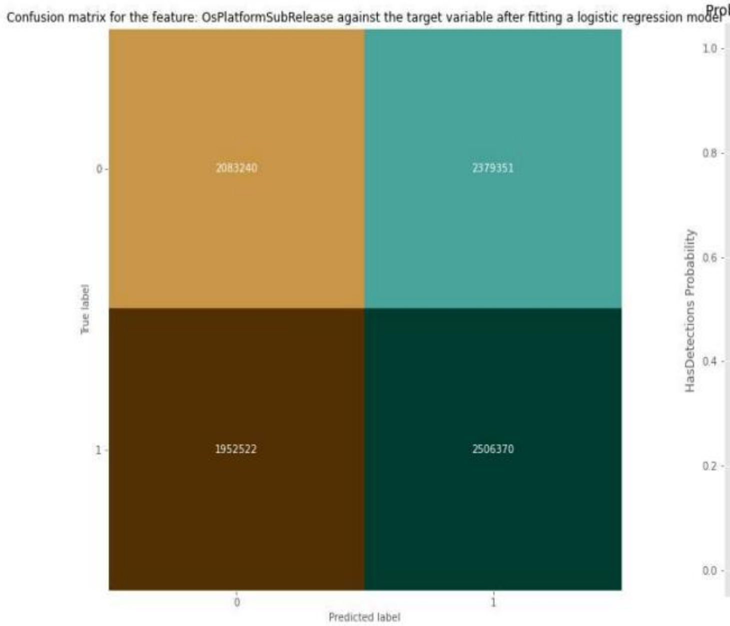
	Feature	Train Unique values	Test Unique values	% changed	Train Type	Test Type
5	AVProductsInstalled	8	6	25.0000	Numerical	Numerical
33	Census_InternalBatteryType	78	63	19.2308	Categorical	Categorical
4	AVProductStatesIdentifier	28970	23492	18.9092	Numerical	Numerical
0	MachineIdentifier	8921483	7853253	11.9737	Categorical	Categorical
34	Census_InternalBatteryNumberOfCharges	41087	36359	11.5073	Numerical	Numerical

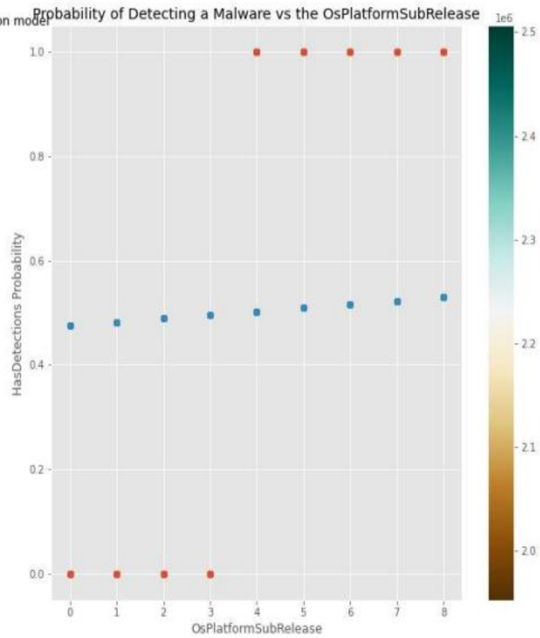


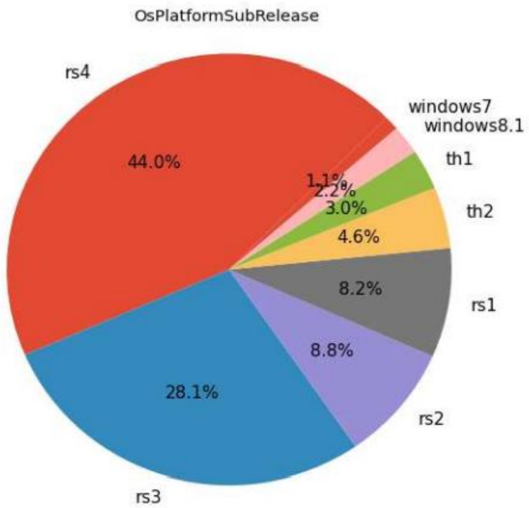
★ Univariate Graphical EDA of Categorical Data (Feature = OsPlatformSubRelease)

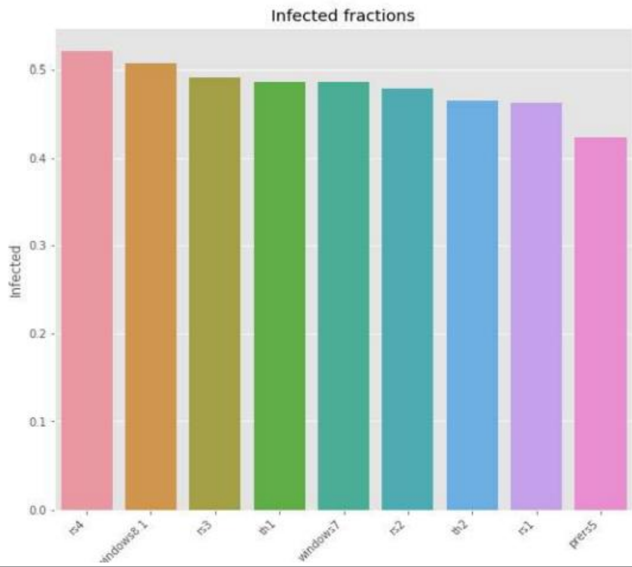
```
Top 10 most occurred categories for the categorical feature OsPlatformSubRelease
rs4            3915526
rs3            2503681
rs2             780270
rs1             730819
th2             411606
th1             270192
windows8.1      194508
windows7         93889
prers5           20992
Name: OsPlatformSubRelease, dtype: int64

Fitting a logistic regression model for the feature OsPlatformSubRelease against the target variabl
e
              precision    recall  f1-score   support

           0       0.52      0.47      0.49   4462591
           1       0.51      0.56      0.54   4458892

    accuracy                           0.51   8921483
   macro avg       0.51      0.51      0.51   8921483
weighted avg       0.51      0.51      0.51   8921483


accuracy score: 0.5144447397366559
F1 score: 0.5144447397366559
```

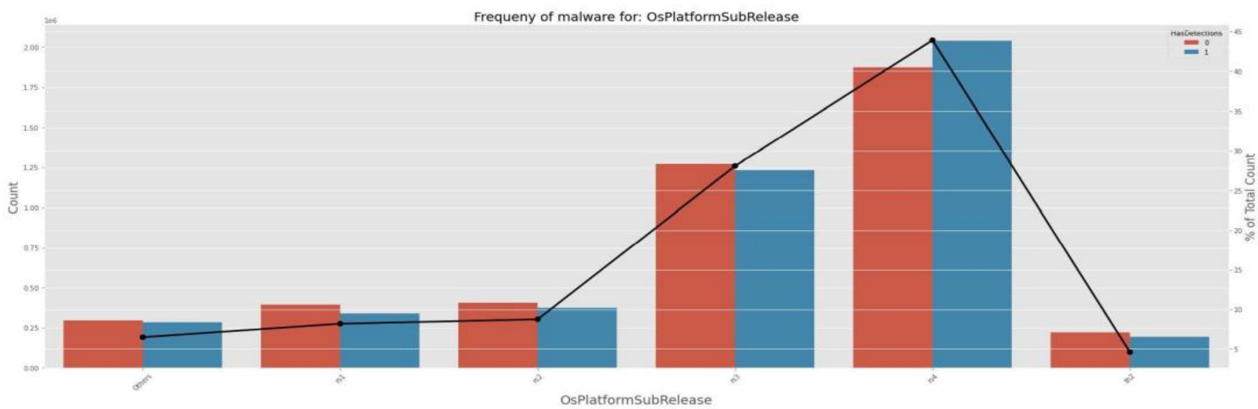



Confusion matrix for the feature: OsPlatformSubRelease against the target variable after fitting a logistic regression model
2083240
2379351
1952522
2506370
True label
Predicted label


Probability of Detecting a Malware vs the OsPlatformSubRelease
HasDetections Probability
OsPlatformSubRelease
1e6

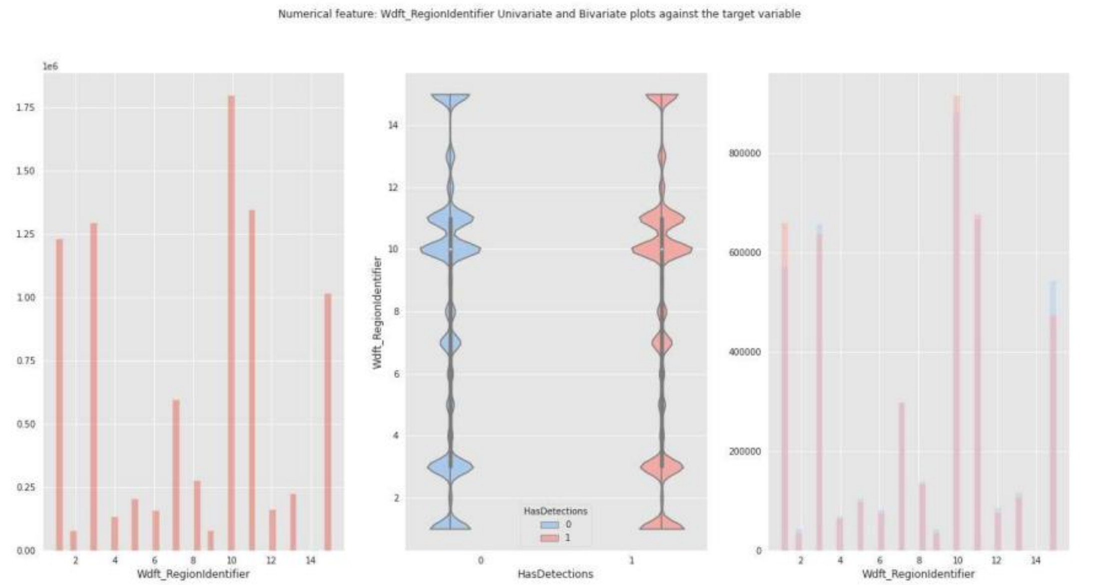

OsPlatformSubRelease
rs4
windows7
windows8.1
th1
th2
rs1
rs2
rs3
44.0%
1.1%
2.2%
3.0%
4.6%
8.2%
8.8%
28.1%

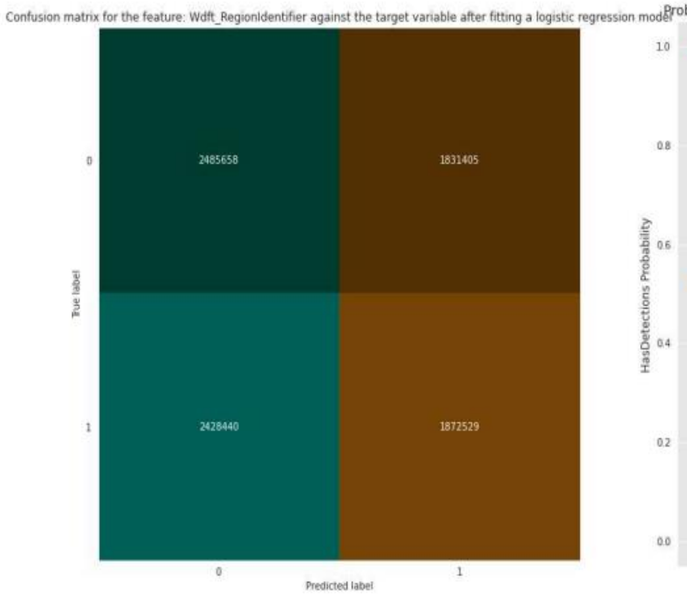

Infected fractions
Infected
rs4
windows8.1
rs3
th1
windows7
rs2
th2
rs1
prers5

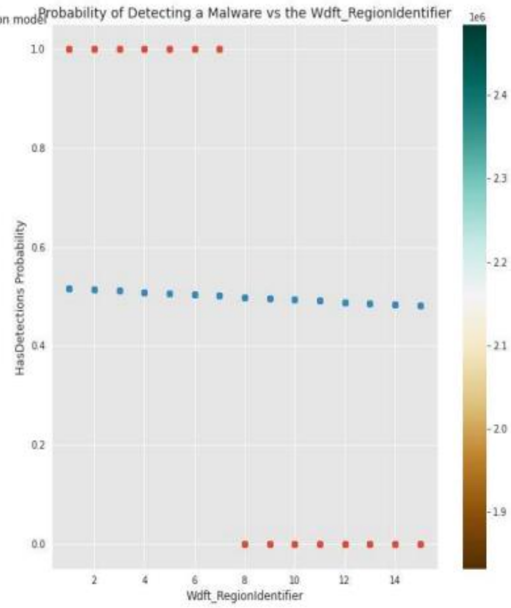

Frequeny of malware for: OsPlatformSubRelease
Count
% of Total Count
HasDetections
OsPlatformSubRelease

★ Univariate Graphical EDA of Numerical Data (Feature = Wdft_RegionIdentifier)

```
Top 10 Values counts for the numerical feature Wdft_RegionIdentifier
10.0000    1800105
11.0000    1347828
3.0000     1295892
1.0000     1232258
15.0000    1017591
7.0000      597297
8.0000      276029
13.0000     225130
5.0000      205372
12.0000     163711
Name: Wdft_RegionIdentifier, dtype: int64
Min value 1.0
Max value 15.0
NaN values 303451
Number of unique values 15
Mean value nan
Variance value 0.0

Fitting a logistic regression model for the feature Wdft_RegionIdentifier against the target variable
              precision    recall  f1-score   support

           0       0.51      0.58      0.54   4317063
           1       0.51      0.44      0.47   4300969

    accuracy                           0.51   8618032
   macro avg       0.51      0.51      0.50   8618032
weighted avg       0.51      0.51      0.50   8618032


accuracy score: 0.5057055949664611
F1 score: 0.5057055949664611
```

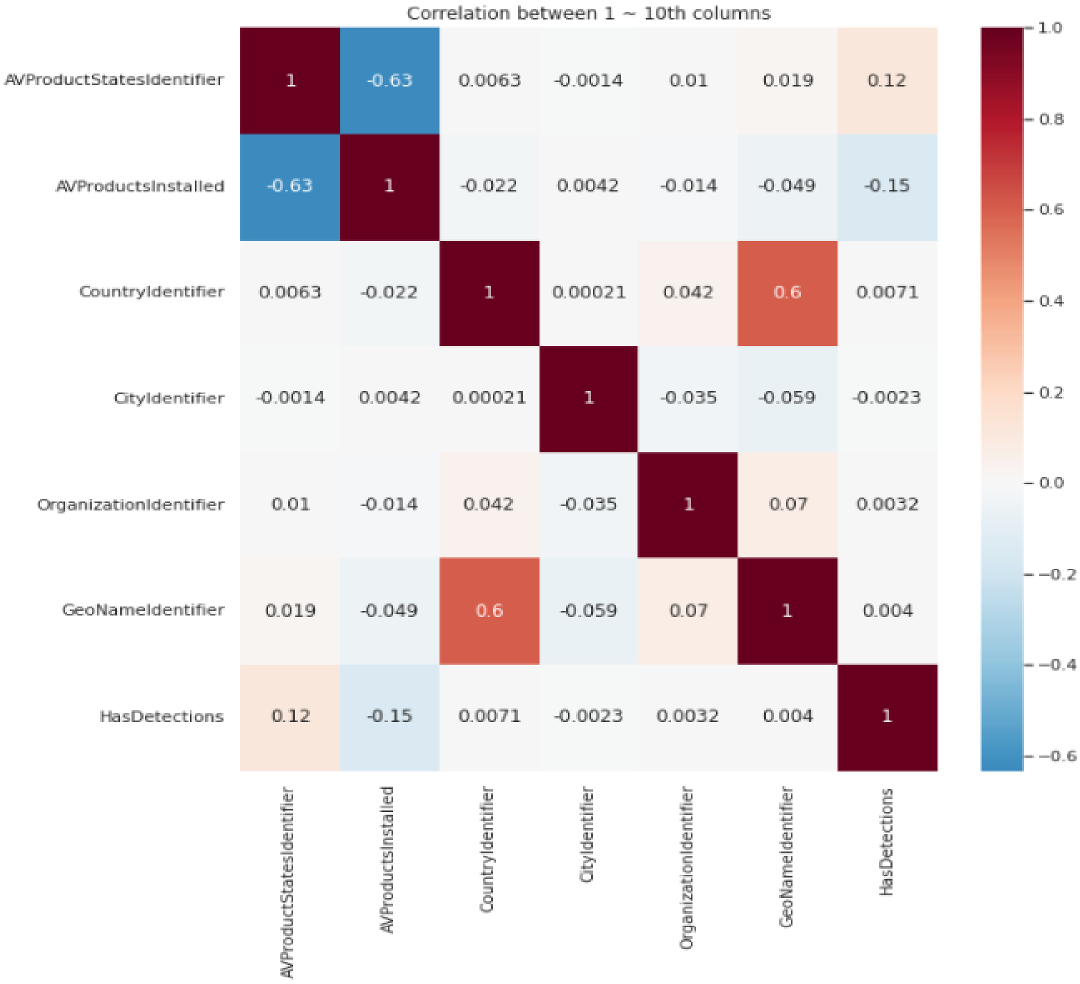




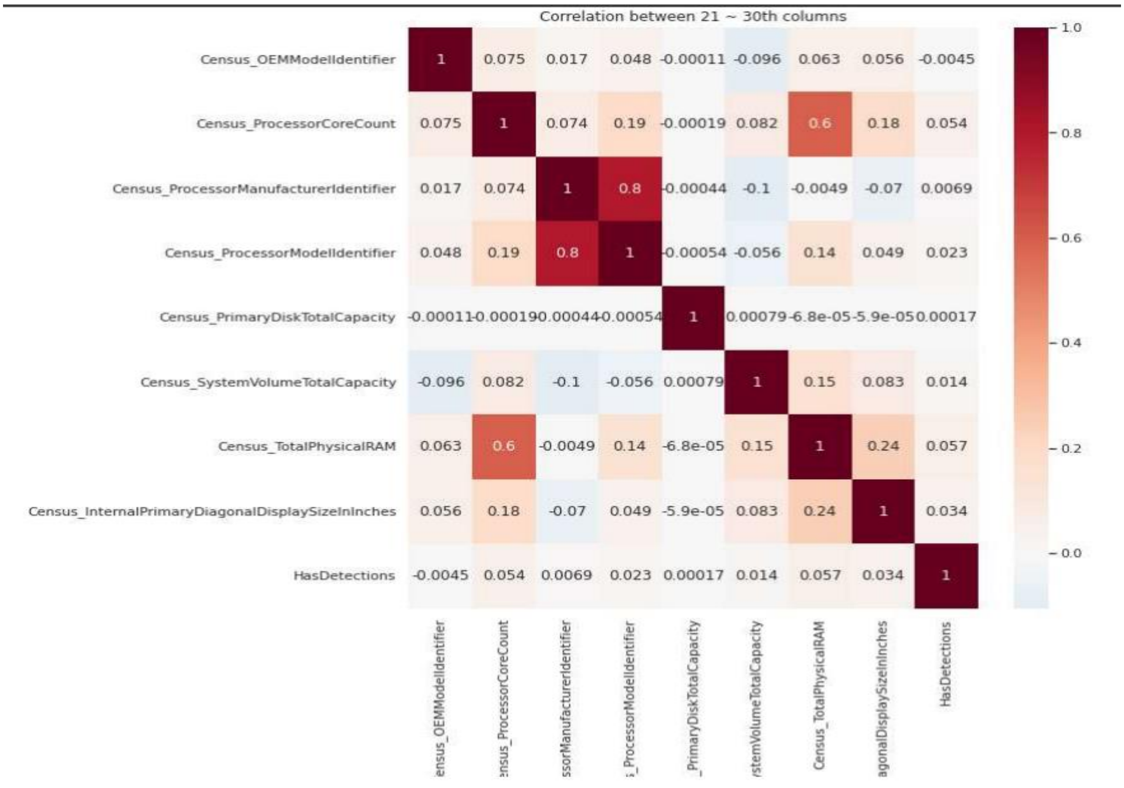


★ **<u>Corelation Analysis</u>**

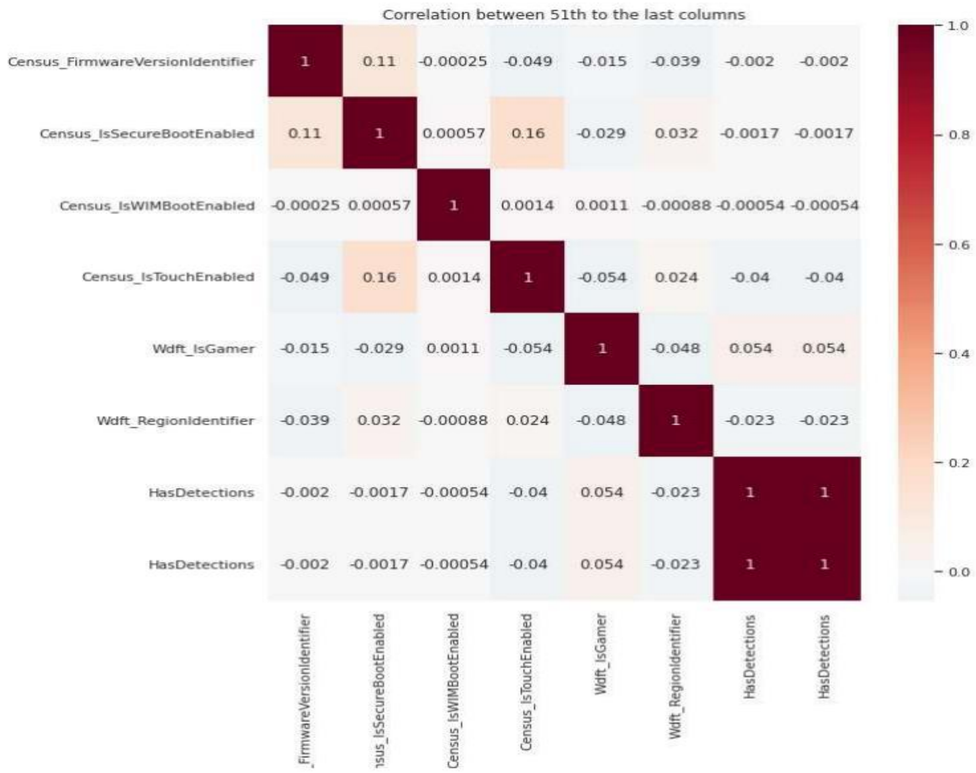
○ **<u>Between first 10 column</u>**



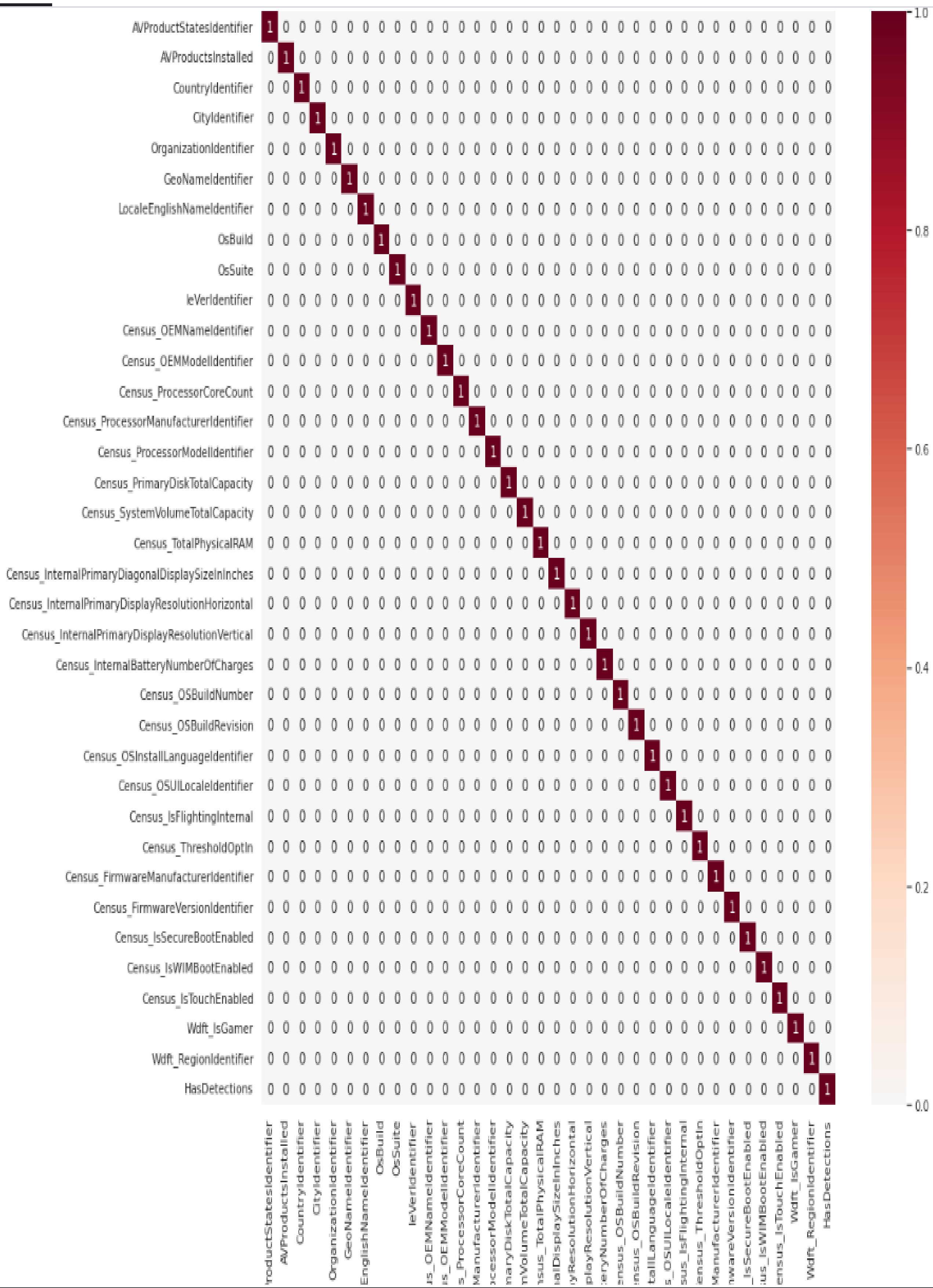
- **Between 21th to 30th column**



- **Between 51th column to last column**



★ Correlation Analysis for whole Dataset and Resultant Correlated Feature



In [129]:
```
s = corr.unstack().drop_duplicates()
so = s.sort_values(kind="quicksort")

print("Top most highly negative correlated features:")
print(so[(so<-0.4)])
print()

print("Top most highly positive correlated features:")
print(so[(so<1) & (so>0.5)].sort_values(ascending=False))
```

```
Top most highly negative correlated features:
AVProductStatesIdentifier  AVProductsInstalled      -0.6329
Census_OSBuildNumber       Census_OSBuildRevision   -0.5642
OsBuild                    Census_OSBuildRevision   -0.4932
dtype: float64

Top most highly positive correlated features:
Census_OSInstallLanguageIdentifier                Census_OSUILocaleIdentifier                     0.9885
OsBuild                                           Census_OSBuildNumber                            0.9379
Census_InternalPrimaryDisplayResolutionHorizontal  Census_InternalPrimaryDisplayResolutionVertical  0.9015
Census_ProcessorManufacturerIdentifier            Census_ProcessorModelIdentifier                 0.7984
CountryIdentifier                                 GeoNameIdentifier                               0.5985
Census_ProcessorCoreCount                         Census_TotalPhysicalRAM                         0.5979
Census_InternalPrimaryDiagonalDisplaySizeInInches  Census_InternalBatteryNumberOfCharges           0.5297
dtype: float64
```

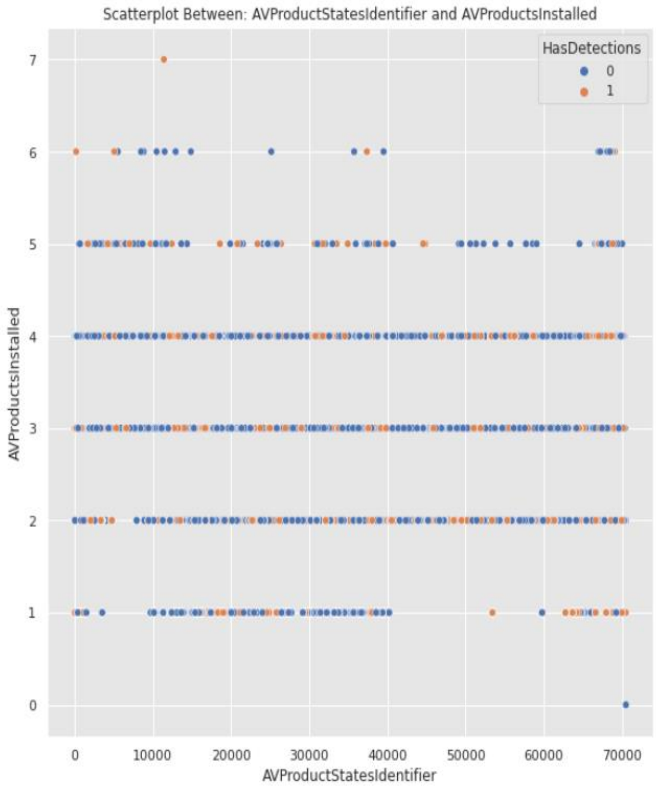
- ★ **Multivariate - Trivariate Analysis between highly correlated features and against the target variable; "HasDetections".**

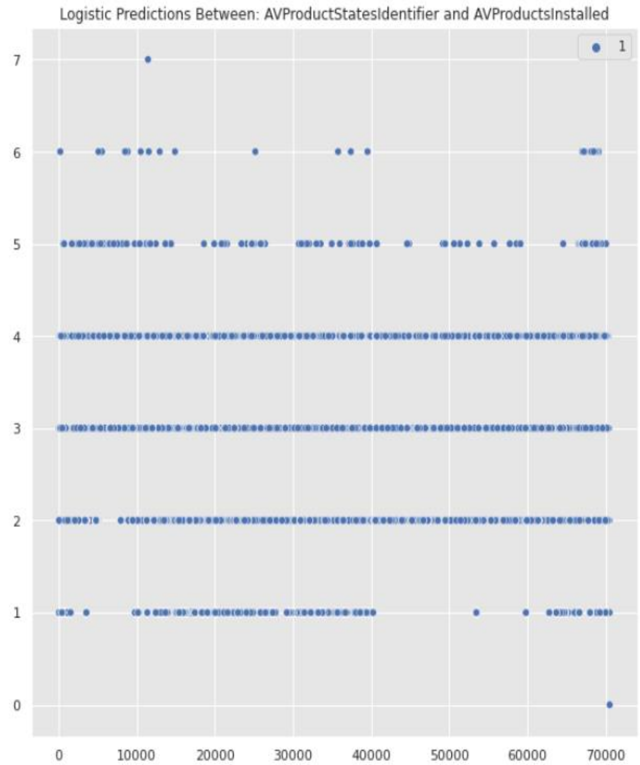
- ○ **Highly Negatie Related Features (AVProductStatesIdentifier & AVProductsInstalled [-0.6329])**

In [135]:
```
%%time
multivariate_plot("AVProductStatesIdentifier", "AVProductsInstalled")
gc.collect()
```

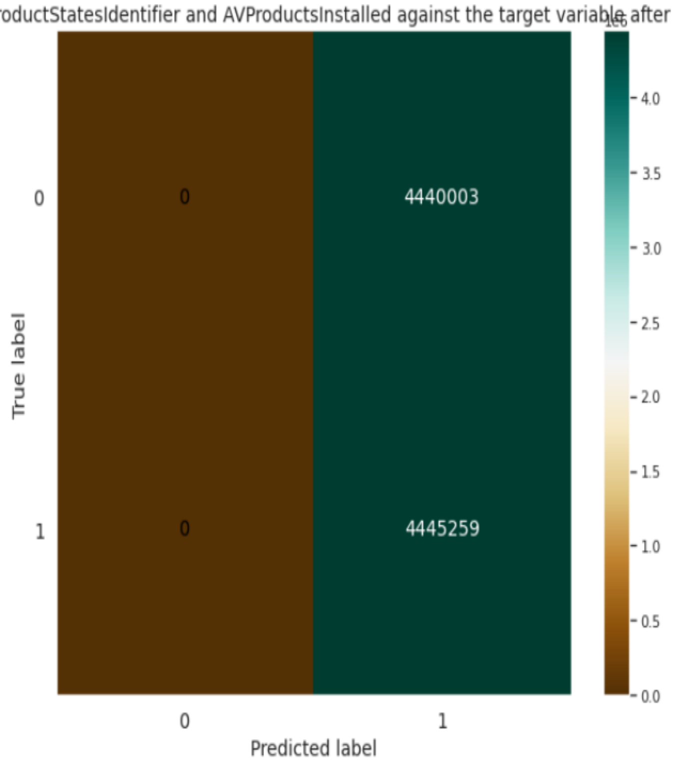
```
Fitting a logistic regression model for the features AVProductStatesIdentifier and AVProductsInstalled against the target variable
              precision    recall  f1-score   support

           0       0.00      0.00      0.00   4440003
           1       0.50      1.00      0.67   4445259

    accuracy                           0.50   8885262
   macro avg       0.25      0.50      0.33   8885262
weighted avg       0.25      0.50      0.33   8885262


accuracy score: 0.5002957706818325
F1 score: 0.5002957706818325
CPU times: user 1min 51s, sys: 7.65 s, total: 1min 58s
Wall time: 1min 52s
```

Out[135]:
```
146
```





Confusion matrix for two features: AVProductStatesIdentifier and AVProductsInstalled against the target variable after fitting a logistic regression model



- **Highly Positive Related Features (OsBuild & Census_OSBuildNumber [0.9379])**

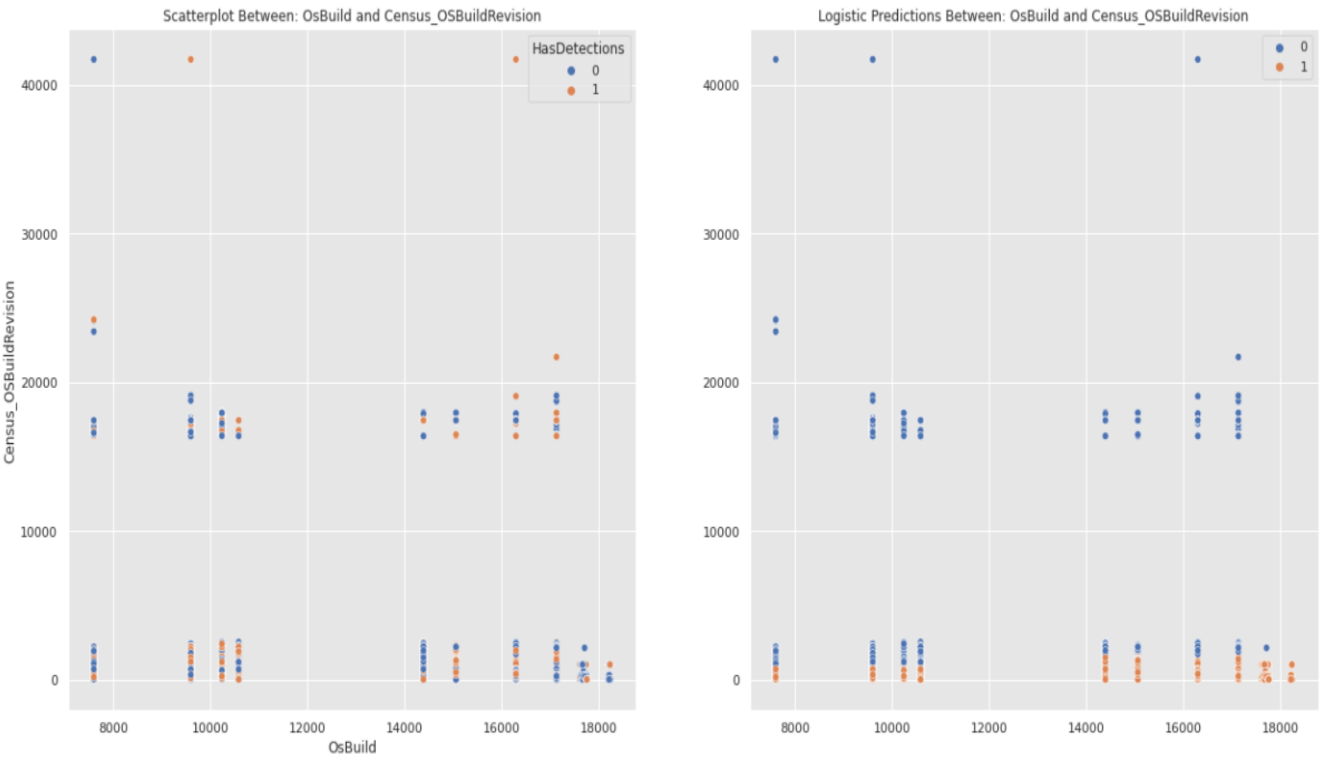
In [137]:

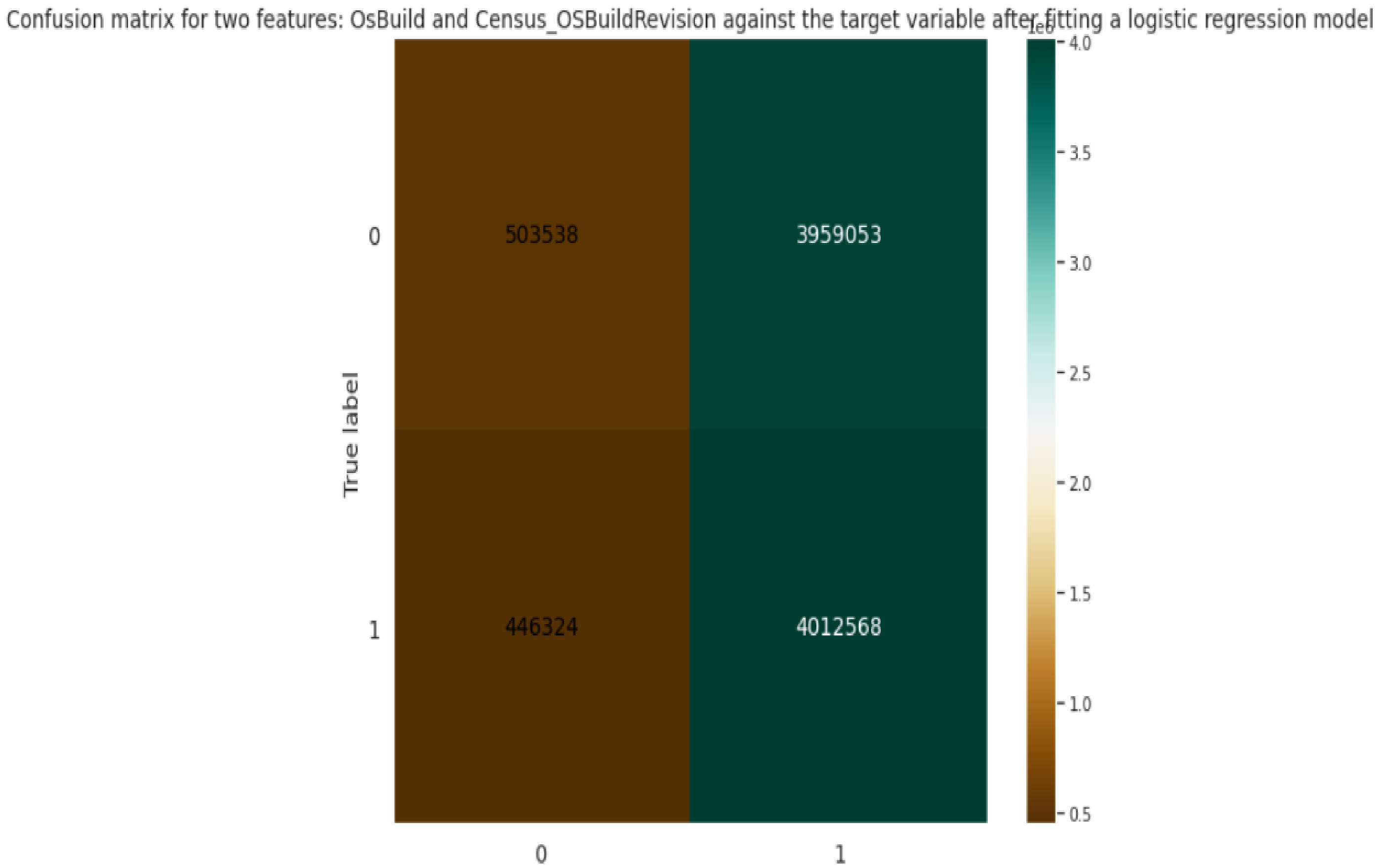
```
%%time
multivariate_plot("OsBuild", "Census_OSBuildRevision")
gc.collect()
```

```
Fitting a logistic regression model for the features OsBuild and Census_OSBuildRevision against the target variable
              precision    recall  f1-score   support

           0       0.53      0.11      0.19   4462591
           1       0.50      0.90      0.65   4458892

    accuracy                           0.51   8921483
   macro avg       0.52      0.51      0.42   8921483
weighted avg       0.52      0.51      0.42   8921483


accuracy score: 0.5062057507703596
F1 score: 0.5062057507703596
CPU times: user 1min 50s, sys: 5.37 s, total: 1min 56s
Wall time: 1min 50s
```

Out[137]:

```
11536
```





★ **LGBM Model**

○ **Label Encoding**

In [80]:

```
indexer = {}
for col in cat_cols:
    # print(col)
    _, indexer[col] = pd.factorize(train[col].astype(str), sort=True)

for col in tqdm_notebook(cat_cols):
    # print(col)
    train[col] = indexer[col].get_indexer(train[col].astype(str))
    test[col] = indexer[col].get_indexer(test[col].astype(str))

    train = reduce_mem_usage(train, verbose=False)
    test = reduce_mem_usage(test, verbose=False)
```

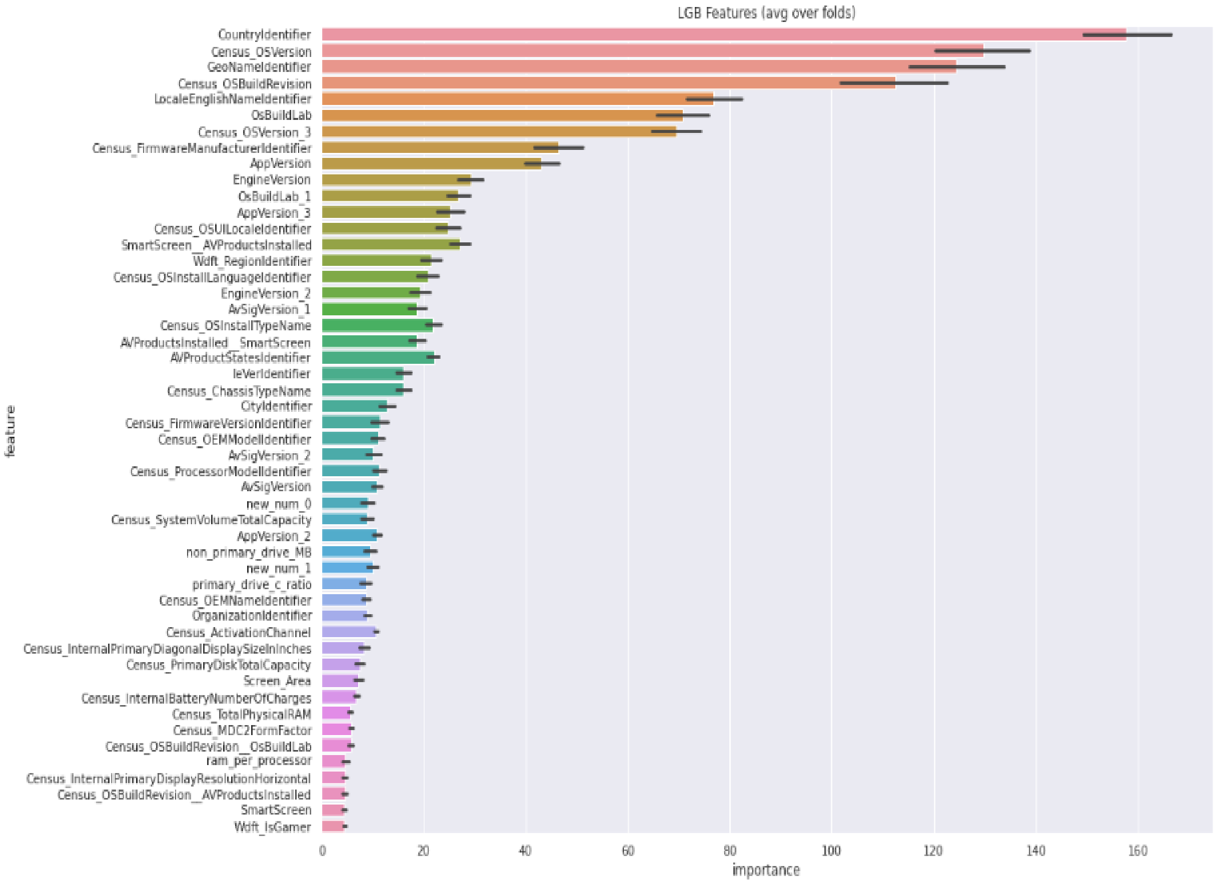
```
0%|          | 0/49 [00:00<?, ?it/s]
```

○ **Finding the Features for Frequency Encoding**

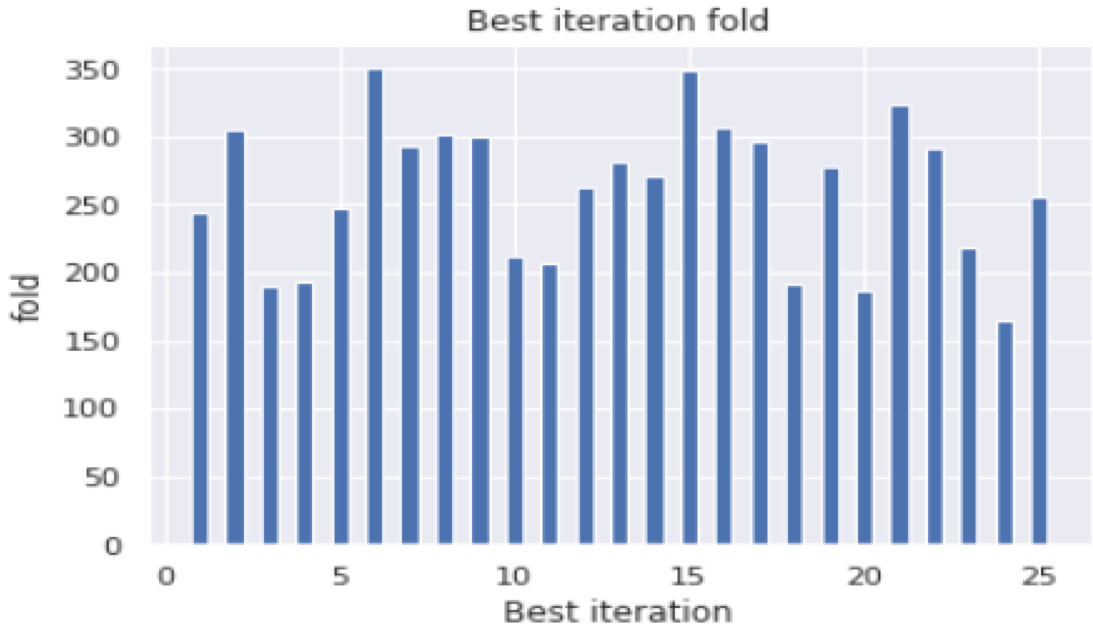
In [76]:
```python
to_encode = []
for col in cat_cols:
    if train[col].nunique() > 1000:
        print(col, train[col].nunique())
        to_encode.append(col)
```

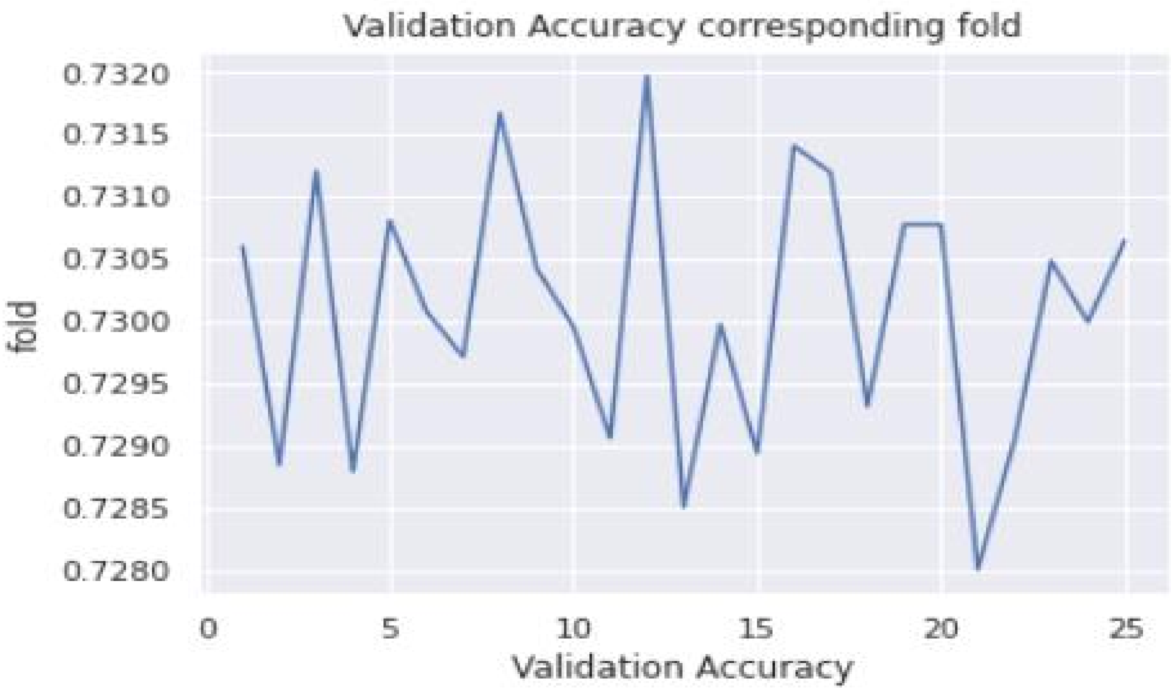
```
AvSigVersion 8531
AVProductStatesIdentifier 28970
CityIdentifier 107366
Census_OEMNameIdentifier 2564
Census_OEMModelIdentifier 175365
Census_ProcessorModelIdentifier 2583
Census_InternalBatteryNumberOfCharges 41087
Census_FirmwareVersionIdentifier 50494
AvSigVersion_2 2766
monitor_dims 10061
Census_OSBuildRevision__OsBuildLab 15367
Census_OSBuildRevision__SmartScreen 1763
Census_OSBuildRevision__AVProductsInstalled 1169
OsBuildLab__Census_OSBuildRevision 15367
OsBuildLab__SmartScreen 2307
OsBuildLab__AVProductsInstalled 1951
SmartScreen__Census_OSBuildRevision 1763
SmartScreen__OsBuildLab 2307
AVProductsInstalled__Census_OSBuildRevision 1169
AVProductsInstalled__OsBuildLab 1951
```

- **Feature Importance according to LGBM**



- **Graph of Best iteration aganist fold and Validation Accuracy of best iteration aganist fold**





★ **XG-Boost Model**

○ **Concatenate both train_sample and test sets before label encoding**

```
In [ ]: train_shape = train_sample.shape
        test_shape = test.shape

        train_and_test = pd.concat([train_sample,test], axis="rows", sort=False)

        del train_sample
        del test
        gc.collect()

Out[ ]: 22
```

```
In [ ]: train_and_test.head()
```

Out[ ]:

	EngineVersion	AppVersion	AvSigVersion	AVProductStatesIdentifier	AVProductsInstalled	Cou
0	1.1.15200.1	4.12.17007.18022	1.275.1311.0	53447.0	1.0	
1	1.1.15100.1	4.14.17613.18039	1.273.1179.0	53447.0	1.0	
2	1.1.14800.3	4.14.17639.18041	1.267.1740.0	53447.0	1.0	
3	1.1.15100.1	4.12.17007.17123	1.273.1545.0	22728.0	2.0	
4	1.1.14901.4	4.8.10240.16384	1.269.1329.0	53447.0	1.0	

5 rows × 57 columns

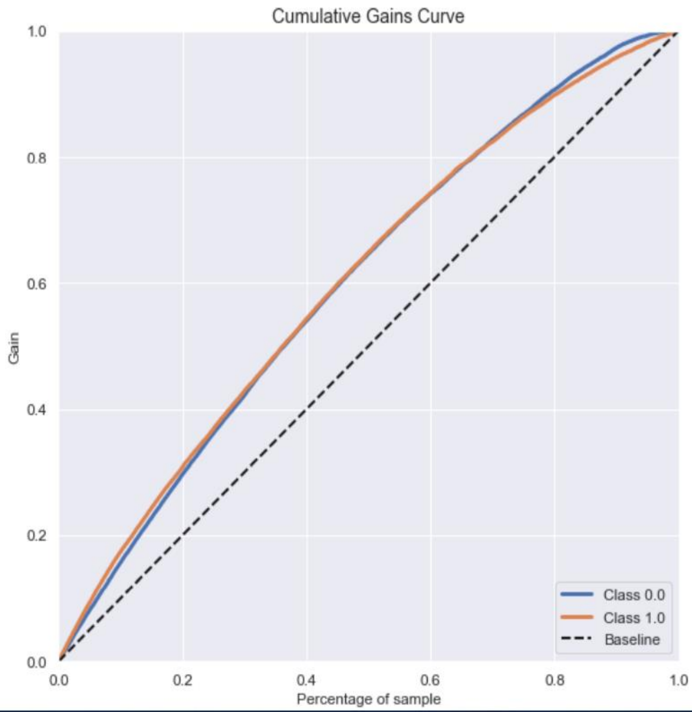
```
In [ ]: train_and_test.tail()
```

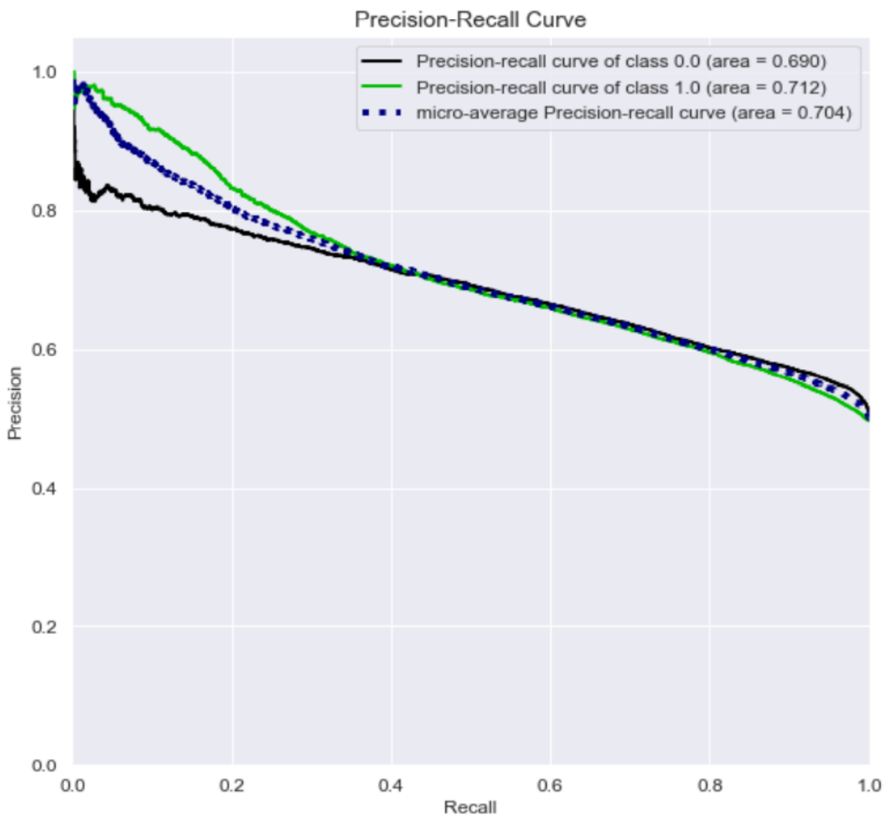
Out[ ]:

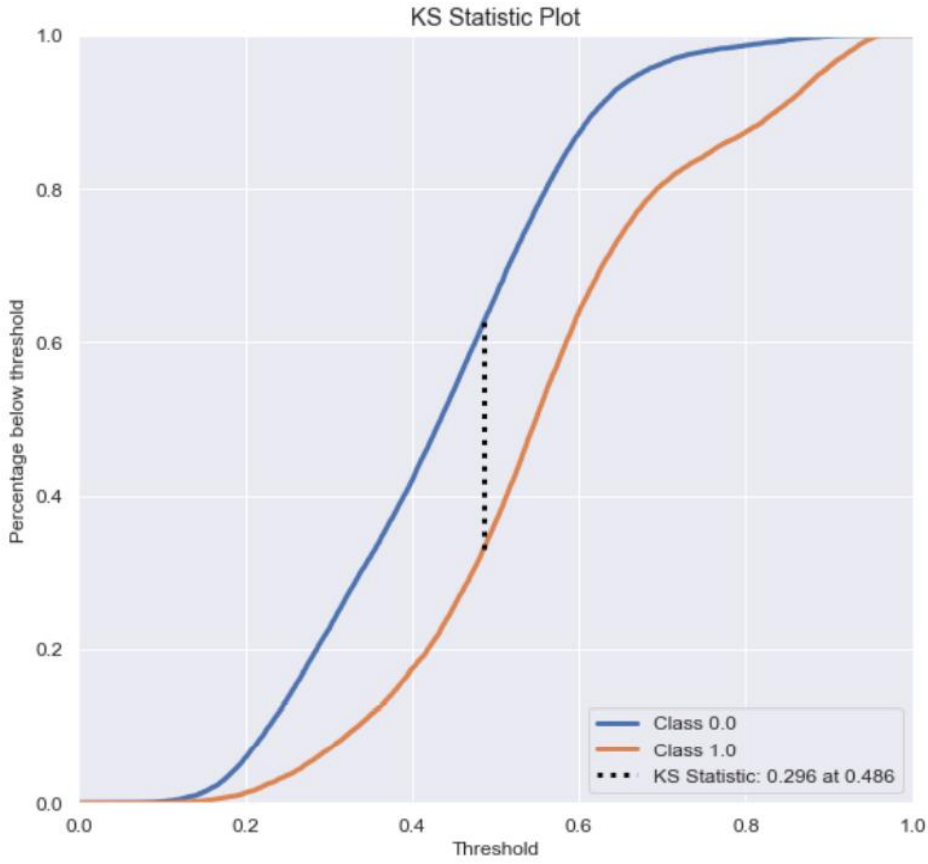
	EngineVersion	AppVersion	AvSigVersion	AVProductStatesIdentifier	AVProductsInstalled
39261	1.1.15300.6	4.18.1809.2	1.277.571.0	53447.0	1.0
39262	1.1.14104.0	4.12.16299.15	1.251.42.0	53447.0	1.0
39263	1.1.15300.5	4.18.1807.18075	1.275.1669.0	53447.0	1.0
39264	1.1.15400.5	4.13.17134.320	1.281.675.0	45615.0	2.0
39265	1.1.15300.6	4.18.1809.2	1.277.1047.0	62773.0	1.0

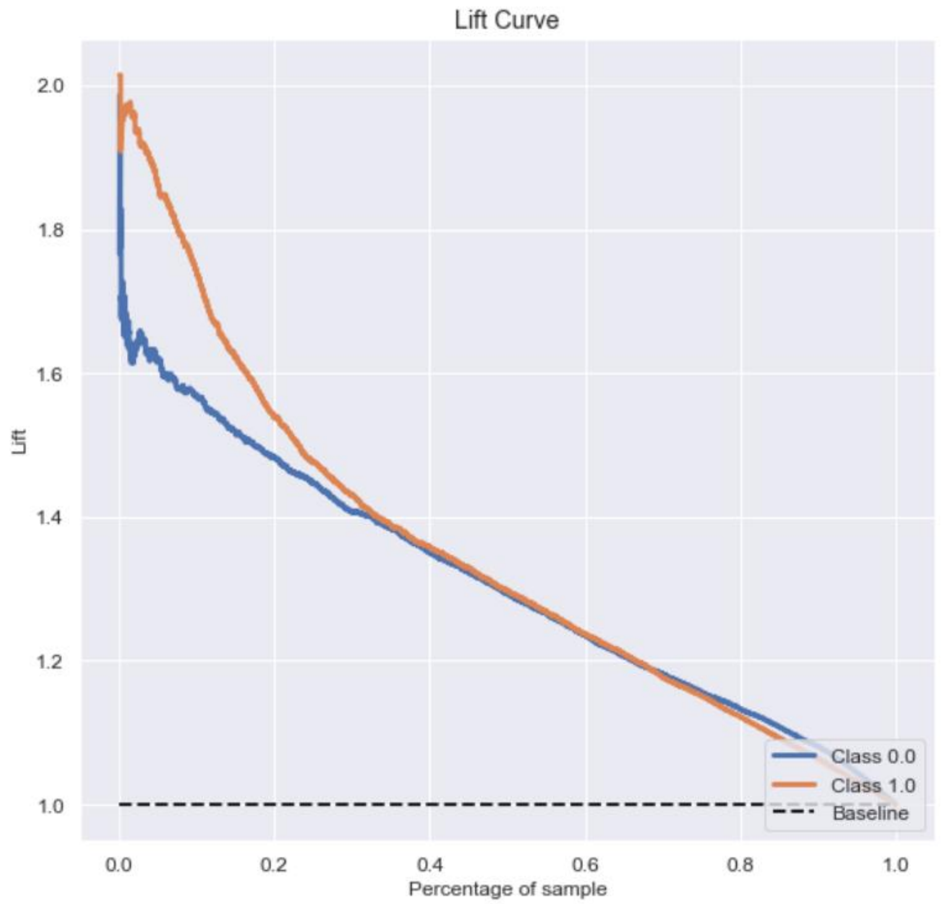
5 rows × 57 columns

- **Cumulative Gains curve, Precison Recall Curve, KS Statistic Plot, Lift Curve, Confussion Matrix and accuracy Results**

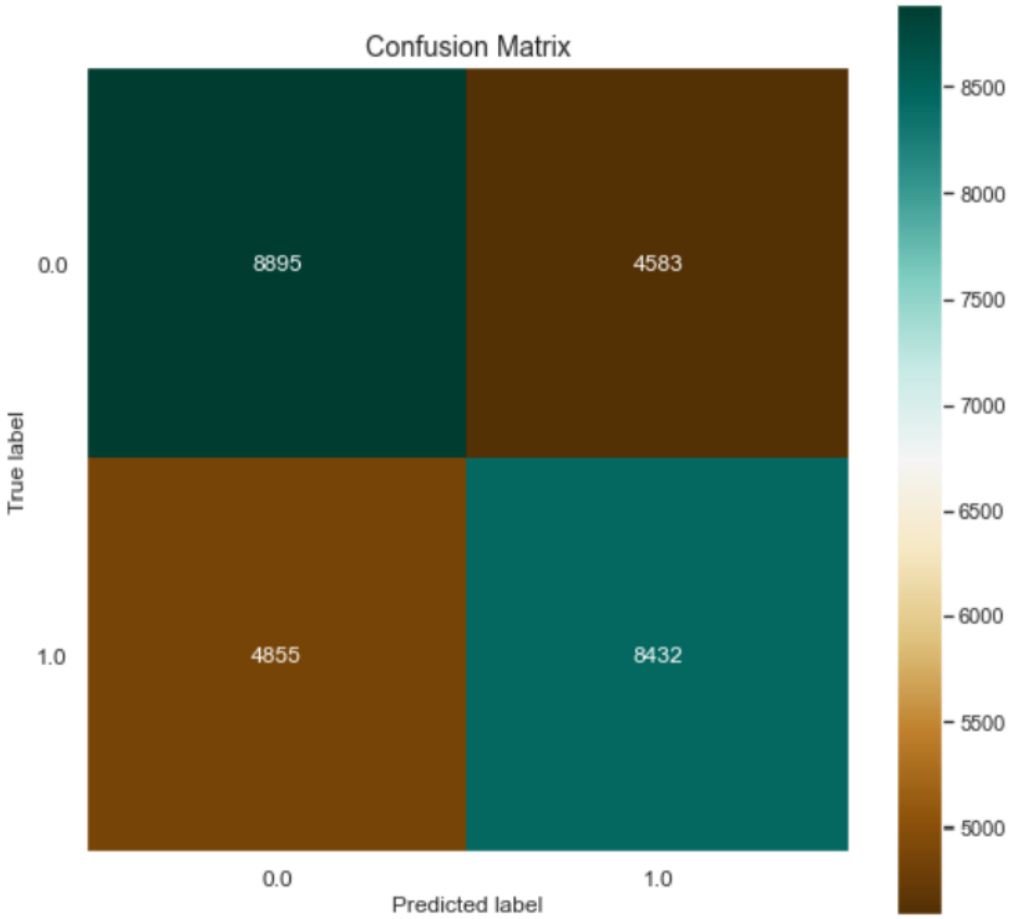




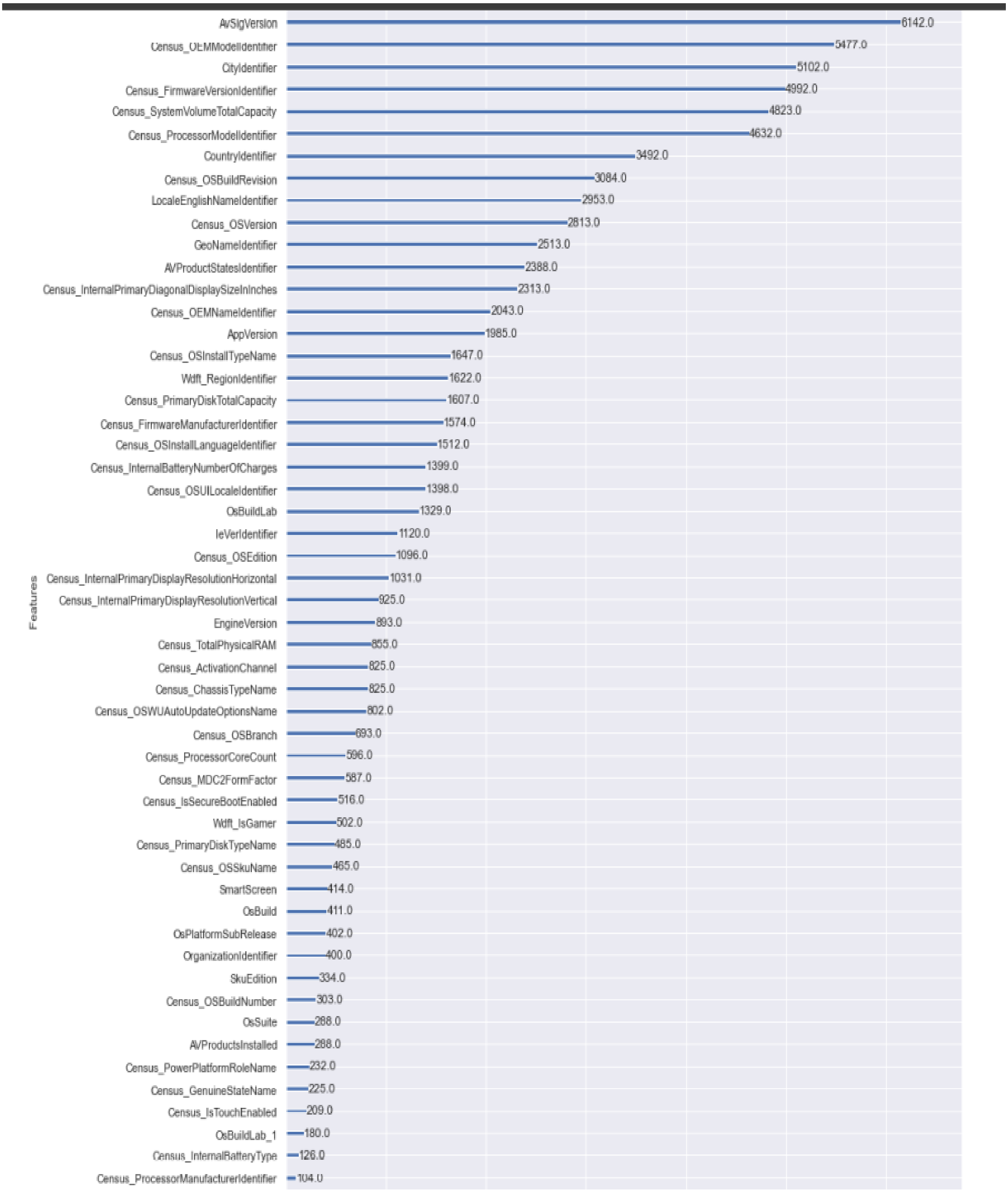

KS Statistic Plot
Percentage below threshold
Threshold
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Class 0.0
Class 1.0
KS Statistic: 0.296 at 0.486


Lift Curve
Lift
Percentage of sample
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Class 0.0
Class 1.0
Baseline

```
[0]	validation_0-auc:0.65489	validation_1-auc:0.58117
[100]	validation_0-auc:0.82409	validation_1-auc:0.70326
[200]	validation_0-auc:0.86452	validation_1-auc:0.70580
[300]	validation_0-auc:0.88780	validation_1-auc:0.70591
[338]	validation_0-auc:0.89600	validation_1-auc:0.70594
              precision    recall  f1-score   support

         0.0       0.65      0.66      0.65     13478
         1.0       0.65      0.63      0.64     13287

    accuracy                           0.65     26765
   macro avg       0.65      0.65      0.65     26765
weighted avg       0.65      0.65      0.65     26765


accuracy_score 0.6473753035680927

roc-auc score for the class 1, from target 'HasDetections'  0.7061433486187174

elapsed time in seconds:  48.19315528869629
```



- **Feature Impotance Graph according to F1-score bt XG-Boost**



★ Random Forest Model

- ### Pre-Processing

```python
def castNum(train):
    col = ['EngineVersion', 'AppVersion', 'AvSigVersion', 'OsBuildLab', 'Census_OSVersion']
    for c in col:
        for i in range(6):
            train[c + str(i)] = train[c].map(lambda x: re.split('\.|-', str(x))[i] if len(re.split('\.|-', str(x))) > i else -1)
            try:
                train[c + str(i)] = pd.to_numeric(train[c + str(i)])
            except:
                print(f'{c + str(i)} cannot be casted to number')
[5]

castNum(test)
castNum(train)
[6]

OsBuildLab2 cannot be casted to number
OsBuildLab3 cannot be casted to number
OsBuildLab2 cannot be casted to number
OsBuildLab3 cannot be casted to number
```

```python
train['HasExistsNotSet'] = train['SmartScreen'] == 'ExistsNotSet'
test['HasExistsNotSet'] = test['SmartScreen'] == 'ExistsNotSet'
[7]

def na_values(train):
    for col, val in train.items():
        if pd.api.types.is_string_dtype(val):
            train[col] = val.astype('category').cat.as_ordered()
            train[col] = train[col].cat.codes
        elif pd.api.types.is_numeric_dtype(val) and val.isnull().sum() > 0:
            train[col] = val.fillna(val.median())

na_values(train)
na_values(test)
```

- ### Accuracy Results

```python
m = RandomForestClassifier(n_estimators=100, min_samples_leaf=200, max_features=0.5, n_jobs=-1, oob_score=False)
m.fit(x_train, y_train)

print_score(m)
[11]

[0.6502281086708829, 0.6218395774578824, 0.6481336173074768, 0.6202580516601489]

pred = m.predict(x_test)
print(f'Accuracy Score = {accuracy_score(y_test, pred)}')
print('Classification Report:')
print(classification_report(y_test, pred))
[12]

Accuracy Score = 0.6202580516601489
Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.68      0.64     40189
           1       0.64      0.56      0.60     40105

    accuracy                           0.62     80294
   macro avg       0.62      0.62      0.62     80294
weighted avg       0.62      0.62      0.62     80294
```

6. Testing

6.1 Model:1 XGBoost Tuning

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now.

6.1.1 Strategies

1. Reading the test and train data
2. Applying Feature Engineering
3. Filling NA values with statistical mode
4. Encode the categorical features before machine learning modeling

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

def MultiLabelEncoder(columnlist,dataframe):
    for i in columnlist:
        #print(i)
        labelencoder_X=LabelEncoder()
        dataframe[i]=labelencoder_X.fit_transform(dataframe[i])

MultiLabelEncoder(categorical_columns, train_and_test)
```

5. Back to train and test dataset after label encoding

```
train_sample = train_and_test[0:train_shape[0]]
test = train_and_test[(train_shape[0]):(train_and_test.shape[0]+1)]
```

```
del train_and_test
```

6. Remove the HasDetections columns from test set, it has been added during dataframe concatenation.

7. Tuning XGBoost model by splitting the dataset into 70/30 train-valid

8. Reasonable values for key inputs:

learning_rate=0.03,

n_estimators=3000

max_depth=11,

min_child_weight=9,

gamma=0.2,

subsample=1,

colsample_bytree=0.4,

objective= 'binary:logistic',

nthread=-1,

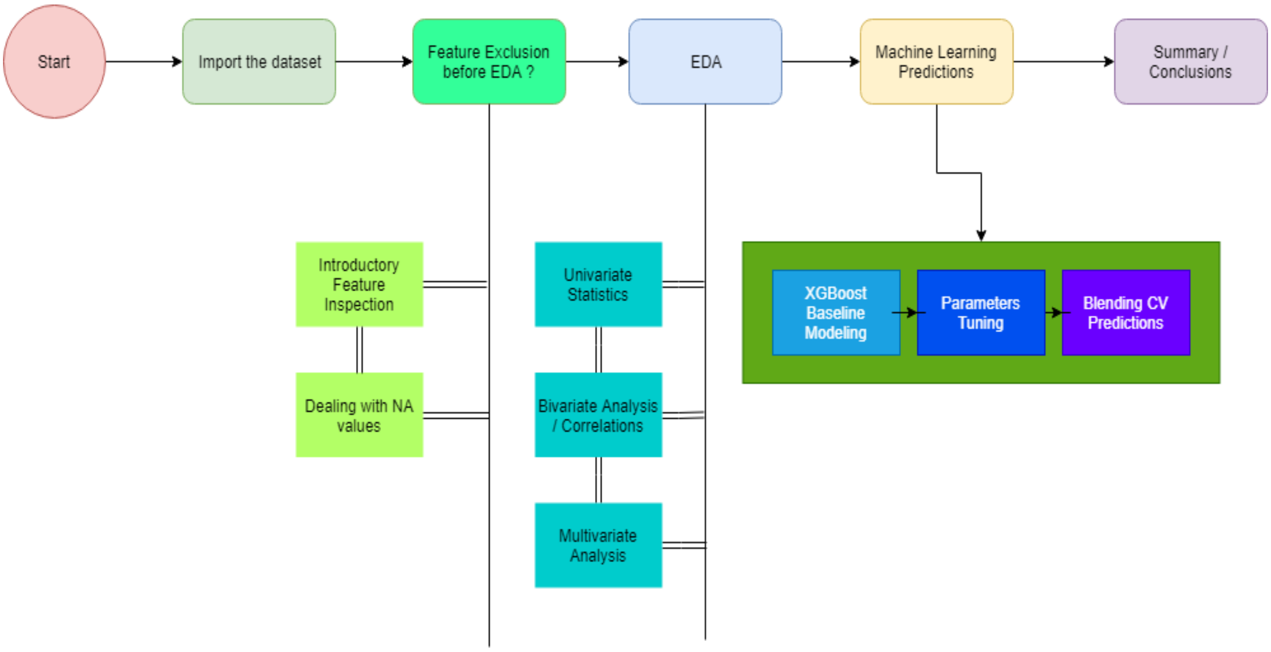
scale_pos_weight=1,

reg_alpha = 0.6,

reg_lambda = 3,

seed=42

6.1.2. Architecture



6.1.3. Predictions

Results:

	precision	recall	f1-score	support
0.0	0.65	0.66	0.65	13478
1.0	0.65	0.63	0.64	13287
accuracy			0.65	26765
macro avg	0.65	0.65	0.65	26765
weighted avg	0.65	0.65	0.65	26765

The HasDetection depicts the predicted values:

	A	B
1	MachineIdentifier	HasDetections
2	c033cd83be992462cb7ad6f867c6d321	0.43279508
3	3fbcfb93505b6d85eba681e3c134c0b8	0.52639157
4	2b5bd3f450bd523bdcdc45ef91f14015	0.19845395
5	37c04b343360ac324656704322c55833	0.540673
6	f453734e77be6080374873bdf92b4f39	0.48713914
7	827abe88dc25ca2b86751a998d81ad03	0.6390072
8	cc79f0b0f56b190256f9a3c501c25399	0.3992669
9	c12342a71438e810102dd11b6ed9e9ea	0.43560076
10	892432f7a4d0730987db2210da5b1815	0.80492014
11	3aa280cf1927829a9840afdf228c69d3	0.41930452
12	126e688ab1872fb5eeb792c33b1bb2cb	0.85391384
13	adecabda2b2a09f553d05640c3a6cf87	0.60197705
14	41bfd39c08228096c970de59e1fa2478	0.41574556
15	10f8686906400ee3ac57ccca0cb01fd4	0.3479435
16	ecd55ae344f8b28c365b4eecf138785a	0.39531365
17	cf40e67f6b3e15b1bccf7eb48f12ebfc	0.32424307
18	9579527501b833eedbdb8f41030d4f60	0.41432106
19	148bb0c46c308ce50fad0e063eb42918	0.46129423
20	4d1793f4292e2e3132b5553e4444db48	0.5102178
21	c4c374b0f0a6a1f5527e93dbd8f408de	0.58170617
22	cf07878a9851436380d5f080fca270da	0.4710803
23	cc82f9b286ca402875f7fc603212237e	0.4889827
24	20a35dcfcbdf4481df86bbed84b2b21c	0.4619006
25	57bbed4f10edf4d860d8f6dbe22bde8e	0.47443753
26	a346109df70b3afc22736fb9f89cc581	0.34957823

6.2 Model 2: Random Forest Classification

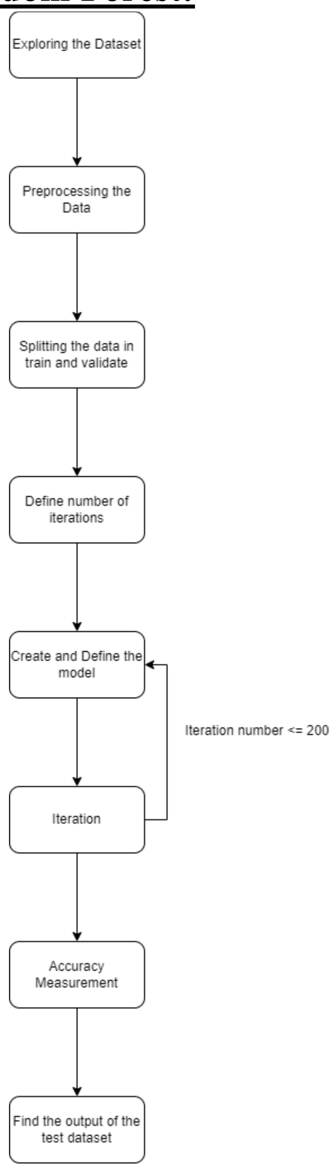
➔ Random Forest Classification is a classification algorithm which consists of a number of iterations each consisting of different decision trees. Following are the advantages of random forest classification.

- Superior method for missing values
- Balances dataset if it is uneven
- Highest accuracy in classification methods
- Handles variables fast
- Used for large datasets

6.2.1 Strategies:

1. Exploring and knowing the dataset.
2. Preprocessing the dataset to remove all the null values and converting all categorical variables into numeric variables
3. Splitting the dataset in train and validation sets
4. Define the number of iterations as 200
5. Train the model using the Random Forest Classification for each iteration to increase the accuracy
6. Find the correlation matrix and accuracy and precision matrix of the trained model
7. Finally predict the test data using the trained model
8. Analyze the final output obtained

6.1.2 Architecture of Random Forest:



6.1.3 Predictions:

Result:

```
Accuracy Score = 0.6205195905048945
Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.68      0.64     40189
           1       0.64      0.56      0.60     40105

    accuracy                           0.62     80294
   macro avg       0.62      0.62      0.62     80294
weighted avg       0.62      0.62      0.62     80294
```

Predicted Values:

```
y_pred = m.predict(test)
op=pd.DataFrame(y_pred)
op.head(15)
```

[13] ✓ 1.1s

	0
0	0
1	0
2	1
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	1
11	0
12	0
13	1
14	0

6.3 Model 3:- LGBM - Light Gradient Boosted Machine

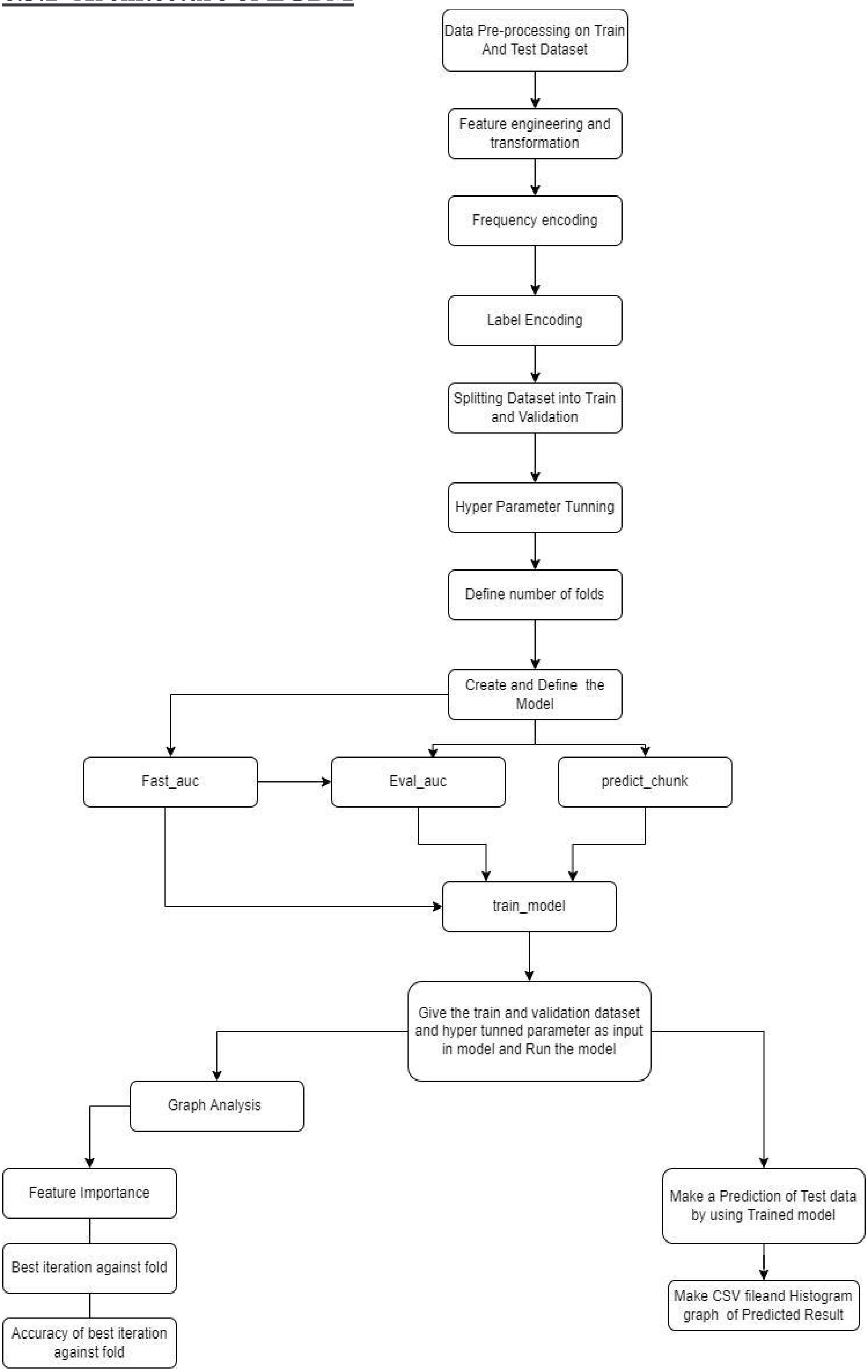
➔ LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel, distributed, and GPU learning.
- Capable of handling large-scale data.

6.3.1 Strategy :-

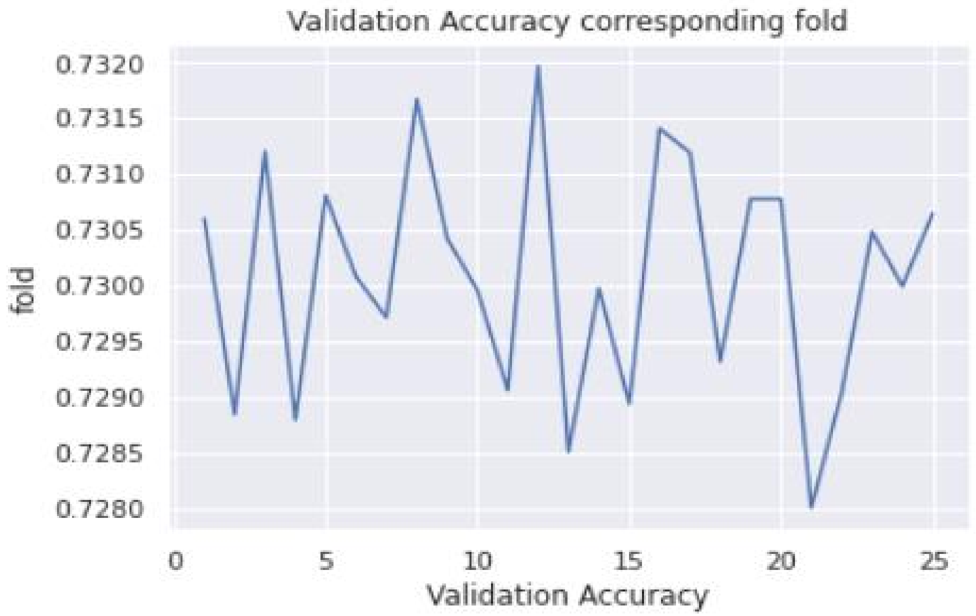
1. Pre-proccessed the data According to the good columns take one by one column and as per it's data type use 'astype()' function.
2. Feature Engineering, Frequency and label encoding
3. Spliting the Train and Validation dataset from train data
4. Perform Hyper parameter tunning using Bayesian Optimization Algorithm (we perform till 15th iteration to get Hyper tunned parameter.
5. Define fold=25 and Parameter
6. Developing model
    a. Define Utility function like Fast_auc for Eval_auc for Train and validation Accuracy and predict_chunks for predicting the test datapoints
    b. Define the the main Train_model function by using utilities and pre-define functions and splitted dataset.
7. Run the model till 25th Fold
8. Feature importance and accuracy graphs
9. Predict the test data by trained model
10. Analysis the Output results

6.3.2 Architecture of LGBM



6.3.3 Results:-

- **Accurcy Vs. Best Itreation**



- **Predicted Results**

In [108]:

```
Predicted_result.head(15)
```

Out[108]:

	MachineIdentifier	HasDetections
0	0000010489e3af074adeac69c53e555e	0.527677
1	00000176ac758d54827acd545b6315a5	0.413577
2	0000019dcefc128c2d4387c1273dae1d	0.430654
3	0000055553dc51b1295785415f1a224d	0.156977
4	00000574cefffeca83ec8adf9285b2bf	0.261852
5	000007ffedd31948f08e6c16da31f6d1	0.641214
6	000008f31610018d898e5f315cdf1bd1	0.150142
7	00000a3c447250626dbcc628c9cbc460	0.062730
8	00000b6bf217ec9aef0f68d5c6705897	0.478442
9	00000b8d3776b13e93ad83676a28e4aa	0.093597
10	00000dec341e29f26b92c3be03640bdc	0.483305
11	00000e658ce75c1e2a3bb47bcc3b08f3	0.722523
12	0000102ff65968bbdc04b69073434b05	0.278627
13	000011236a5dc4ff119541c42bb4287e	0.685291
14	0000124d8811c1a5b5848c4d730cfbf8	0.415547

7. Limitation and future extension

Ultimately, the use of ensembles of complex classifiers has barely scratched the surface of what is possible. Feature gathering took up a large portion of research time, and there is plenty more to be done on the classifiers. Manipulating the structure and number of layers in both the convolutional neural network and feed-forward neural network could drastically affect the performance of the classifiers. Adding new classifiers also has the potential to add significant improvements to the malware classification.

The machine learning features used here rely on the correct disassembly of the malware binary. The use of more advanced binary packers by malware authors can make disassembling difficult. One way to handle this is to use dynamic analysis to run the packed program and dump process memory image once the malware has unpacked itself. This is then given to the disassembler.

8. Conclusion and Reference

8.1 Conclusion

The main contribution of our thesis is the malware data sets that we have found through kaggle. As we have mentioned, these malware data sets have been hosted as a part of competition by Microsoft. We also discussed malware analysis frameworks and libraries used on Jupyter. From our initial investigation, we believe that Light LGM is best fit compared to XGBoost and Random forest because of two reasons, ability to yield highest accuracy of 73% and to give results in the least time. For the future, we plan to explore additional datasets for hosting. In addition, we also plan to carry out a more in-depth investigation of malware analysis using a few more algorithms like Logistic Regression as well as enhance the frameworks that we have designed.

8.2 References

1. Evgeny Burnayev and Dmitry Smolyakov. One-class machine of reference vectors using privileged information. In Information technologies and systems (ITaS), 2016.
2. Nak-Hyun Kim, Byung ik Kim, and Tae jin Lee. Performance analysis of the malware classification method in accordance with the changes in assembly code. 2016.
3. Philippe Biondi, Xavier Mehrenberger, and Sarah Zennou. Rebus : un bus de communication facilitant la coopération entre outils d'analyse de sécurité. In Symposium on Information and Communications Security, 2015.
4. Hyun-Jong Lee, Heo-Jae Hyeok, and Doosung Hwang. Performance Comparison of Machine Learning Algorithms for Malware Detection. volume 26, pages 143–146. The Korean Society Of Computer And Information, 2018. URL: http://www.dbpia.co.kr/Article/NODE07303202
5. J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In 2015 10th International Conference on

Malicious and Unwanted Software (MALWARE), pages 11–20, Oct 2015. doi:10.1109/MALWARE.2015.7413680
6. Felan Carlo C. Garcia and Felix P. Muga II. Random forest for malware classification. CoRR, abs/1609.07770, 2016. URL:https://arxiv.org/abs/1609.07770
7. Mohammad Imran. Evlauation of Hidden Markov Model for Malware Behavioural Classification. PhD thesis, 2016.
8. Beilun Wang, Ji Gao, and Yanjun Qi. A theoretical framework for robustness of (deep) classifiers under adversarial noise. CoRR, abs/1612.00334v8, Feb 2017. URL: http://arxiv.org/abs/1612.00334v8.
9. Ji Gao, Beilun Wang, Zeming Lin, and Weilin Xu Yanjun Qi. Deepcloak: Masking deep neural network models for robustness against adversarial samples. CoRR, abs/1702.06763, 2017. URL:https://arxiv.org/abs/1702.06763
10. A. P. Norton and Y. Qi. Adversarial-playground: A visualization suite showing how adversarial examples fool deep learning. In 2017 IEEE Symposium on Visualization for Cyber Security (VizSec), pages 1–4, Oct 2017.7. doi:10.1109/VIZSEC.2017.8062202
11. Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In NDSS, 2018.
12. Daniel Scofield, Craig Miles, and Stephen Kuhn. Fast model learning for the detection of malicious digital documents. In Proceedings of the 7th Software Security, Protection, and Reverse Engineering / Software Security and Protection Workshop, SSPREW-7, pages 3:1–3:8, New York, NY, USA, 2017. ACM. URL:https://dl.acm.org/doi/10.1145/3151137.3151142, https://doi.org/10.1145/3151137.3151142
13. Tingting He, Jingfeng Xue, Jianwen Fu, Yong Wang, and Chun Shan. Research on malicious code analysis method based on semi-supervised learning. In Ming Xu, Zheng Qin, Fei Yan, and Shaojing Fu, editors, Trusted Computing and Information Security, pages 227–241, Singapore, 2017. Springer Singapore.
14. Songqing Yue. Imbalanced malware images classification: a CNN based approach. CoRR, abs/1708.08042, 2017. URL: https://arxiv.org/abs/1708.08042
15. Johann Vierthaler, Roman Kruszelnicki, and Julian Schütte. Webeye automated collection of malicious http traffic. Technical report, Fraunhofer Research Institute for Applied and Integrated Security, November 2017.
16. Joachim Hansen. The study of keyword search in open source search engines and digital forensics tools with respect to the needs of cyber crime investigations, 2017.
17. Thomas Barabosch, Niklas Bergmann, Adrian Dombeck, and Elmar Padilla. Quincy: Detecting host-based code injection attacks in memory dumps. In Michalis Polychronakis and Michael Meier, editors, Detection of Intrusions and Malware, and Vulnerability Assessment, pages 209–229, Cham, 2017. Springer International Publishing.
18. Morten Oscar Østbye. Multinomial malware classification based on call graphs, May 2017.

19. Tonya Fields and Jonathan Graham. Classifying network attack data using random forest. In CATA, Dec 2016.
20. Charles A. Fowler. A HYBRID INTELLIGENCE/MULTI-AGENT SYSTEM FOR MINING INFORMATION ASSURANCE DATA. PhD thesis, May 2015.
21. R. Paranthaman and B. Thuraisingham. Malware collection and analysis. In 2017 IEEE International Conference on Information Reuse and Integration (IRI), pages 26–31, Aug 2017.https://ieeexplore.ieee.org/document/8102915
22. Zhiwu Xu, Cheng Wen, Shengchao Qin, and Zhong Ming. Effective malware detection based on behaviour and data features. In Meikang Qiu, editor, Smart Computing and Communication, pages 53–66, Cham, 2018. Springer International Publishing.
23. https://in.indeed.com/career-advice/career-development/what-is-technical-feasibility
24. https://www.igi-global.com/dictionary/implementation-feasibility/48210
25. https://en.wikipedia.org/wiki/Feasibility_study#:~:text=of%20the%20land.-,Operational%20feasibility%20study,analysis%20phase%20of%20system%20development.
26. https://www.villanovau.com/resources/project-management/project-team-roles-and-responsibilities/
27. https://www.uscybersecurity.net/anti-virus-software/
28. https://www.lawinsider.com/dictionary/communications-interfaces#:~:text=Remove%20Advertising-,Communications%20Interfaces%20means%20the%20interfaces%20and%20protocols%20that%20enable%20software,Interoperate%20with%20the%20Microsoft%20Platform
29. https://towardsdatascience.com/understanding-random-forest-58381e0602d2
30. https://corporatefinanceinstitute.com/resources/knowledge/other/random-forest/

Bhaskaracharya National Institute for Space Applications and Geo-informatics
MeitY, Government of India
Phone: 079 - 23213081 Fax: 079 - 23213091
E-mail: info@bisag.gujarat.gov.in, website: https://bisag-n.in/

ISO 9001:2008
ISO 27001:2013
CMMI LEVEL-5

Report Verification Procedure

Date:

Project Name: Malware Prediction System

Student Name & ID: 1. Charmi Shah
2. Shubham Kathiriya
3. Vedant Doshi

	Soft Copy	Hard Copy
Report Format:	☐	☐
Project Index:	☐	☐

Sign by Training Coordinator **Sign by Project Guide**