Microsoft Malware Prediction Project Report

# Objective

The main objective in this capstone project is to predict a Windows machine’s probability of getting infected by various families of malware, based on different properties of that machine. The effects of several properties were investigated. Some of the properties included ram,os version, windows defender version, empty disk space, processor version, default browser, Geolocation, Battery type like this there are 83 properties taken.

# Data Wrangling

## Data collection

Data was collected from the kaggle website sourced from by [Microsoft](http://www.microsoft.com/), [Windows Defender ATP Research](https://www.microsoft.com/en-us/WindowsForBusiness/windows-atp), [Northeastern University College of Computer and Information Science](https://www.ccis.northeastern.edu/), and [Georgia Tech Institute for Information Security & Privacy](https://cyber.gatech.edu/).

## Getting to know about data set:

Dataset size is 4.6gb with 83 columns, to load the load the dataset into jupyter notebook quickly used dask dataframe with predefined data types

The goal of this competition 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.

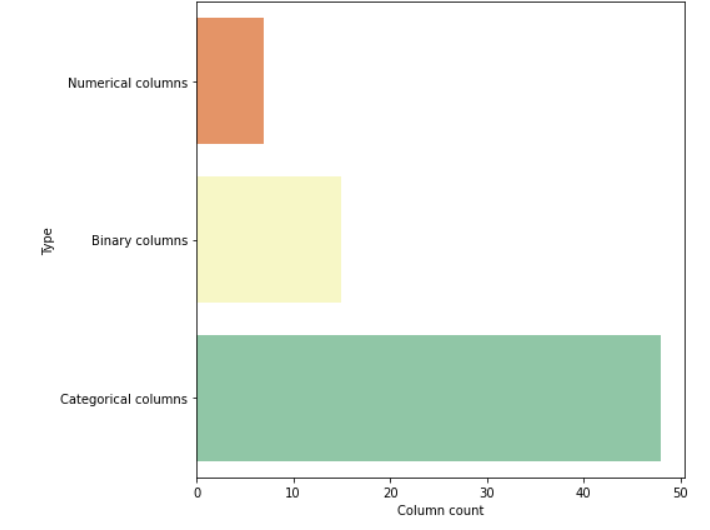
Each row in this dataset corresponds to a machine, uniquely identified by a MachineIdentifier. HasDetections is the ground truth and indicates that Malware was detected on the machine. Using the information and labels in train.csv, you must predict the value for HasDetections for each machine in test.csv.

## Dealing with missing values and skewed data.

Properties with more than 50% missing values are removed resulting in 74 properties, filling these missing values using less than 50% data wont provide statistically correct values and also highly skewed properties are not considered for model development. For the properties which have missing values less than 50% were looked individually and filled with statistically appropriate values.

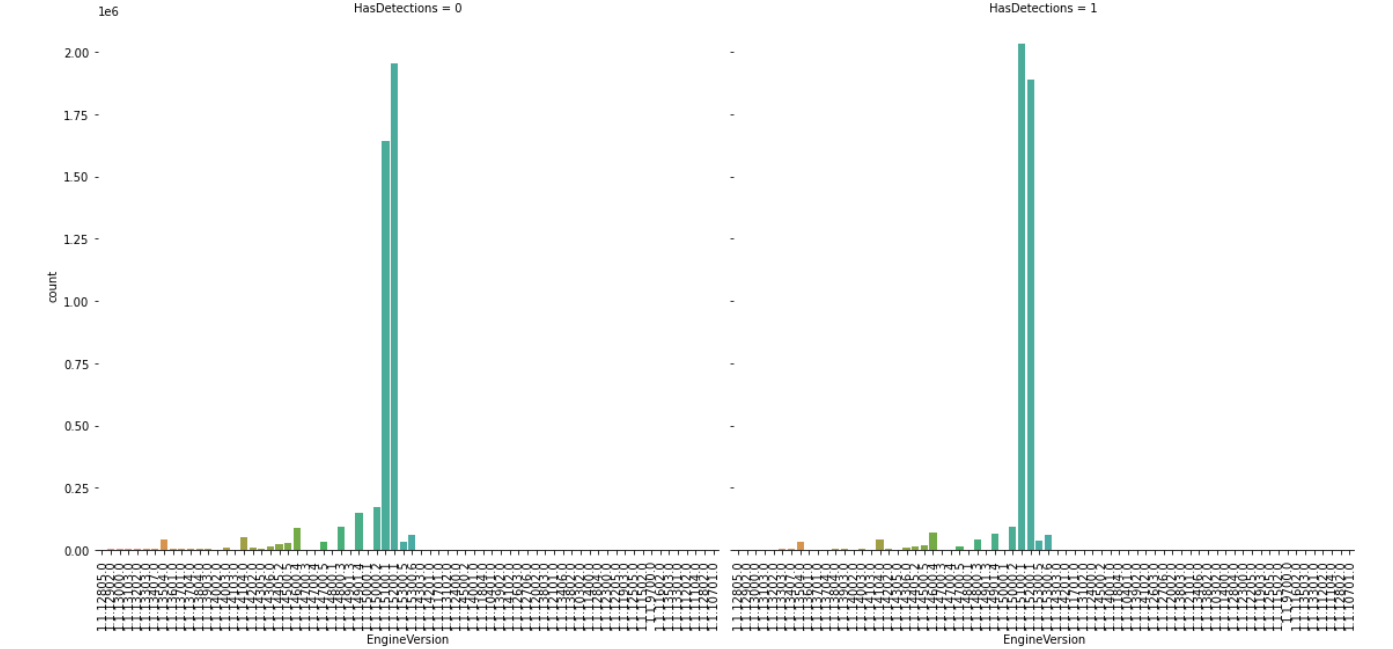
# Exploratory data analysis

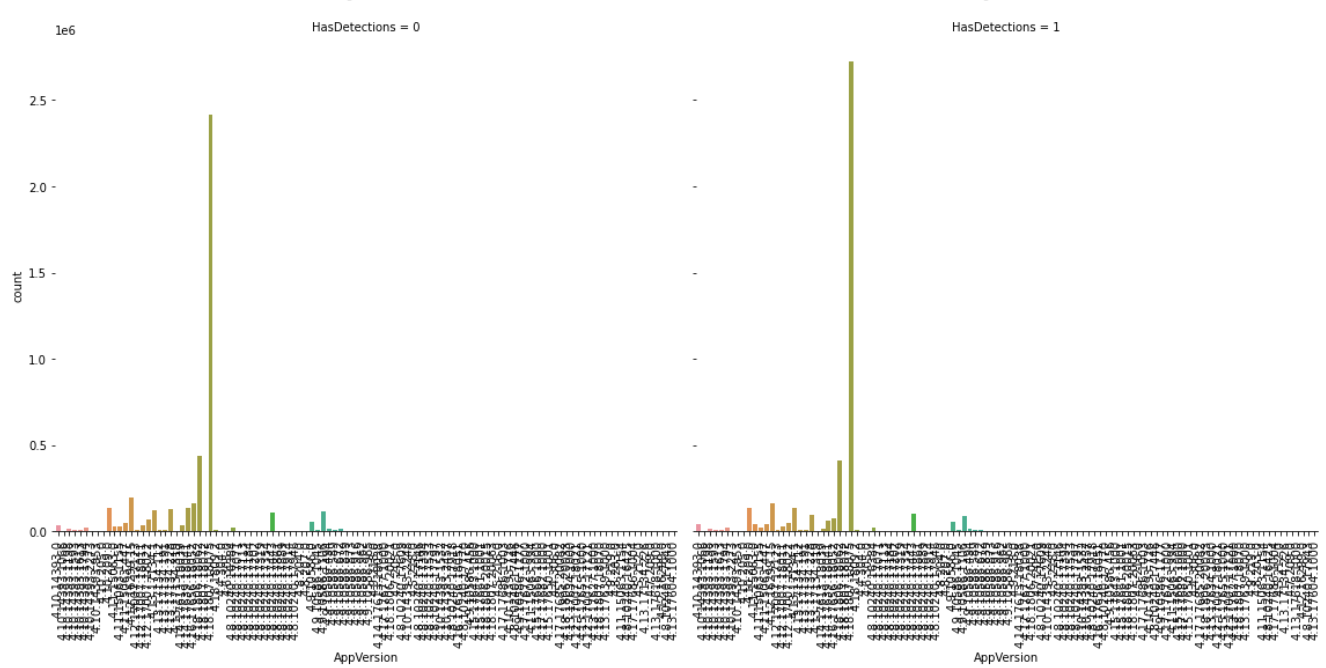
## Column type distribution:



The above graph shows that data is most occupied by categorical columns, so we will take a deep dive into categorical columns how each category effect ‘HasDetection’ value

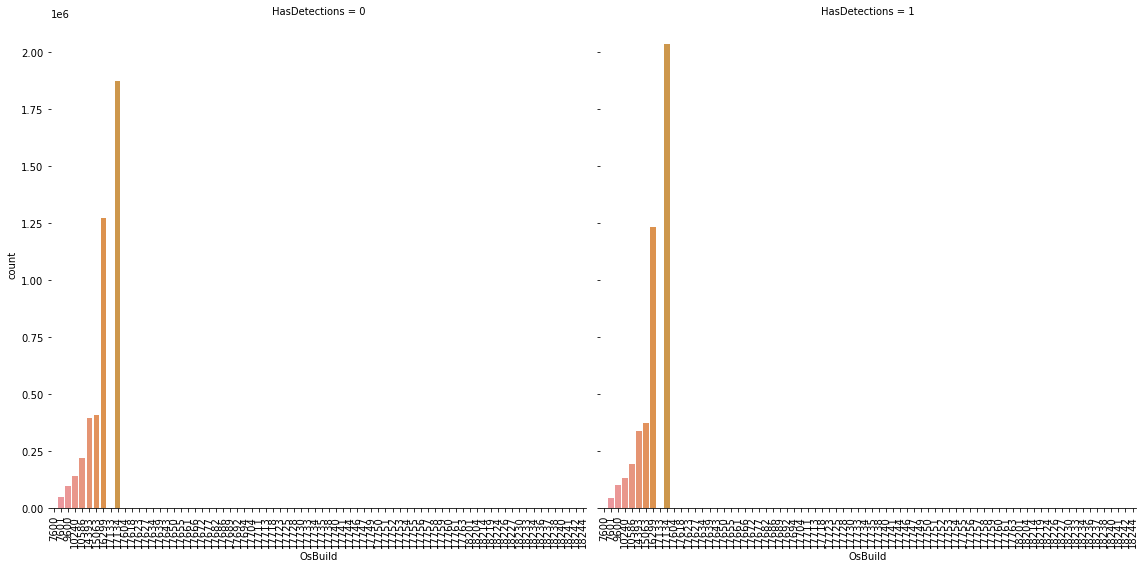
## Features related to version

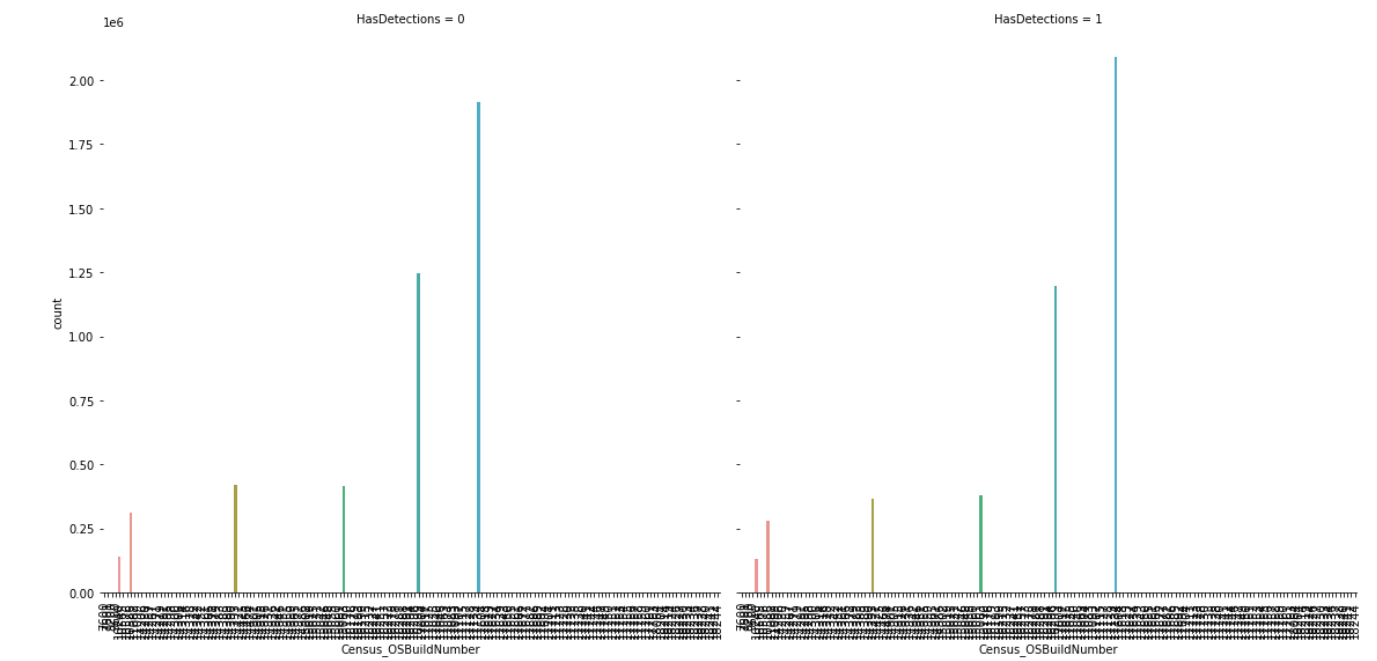


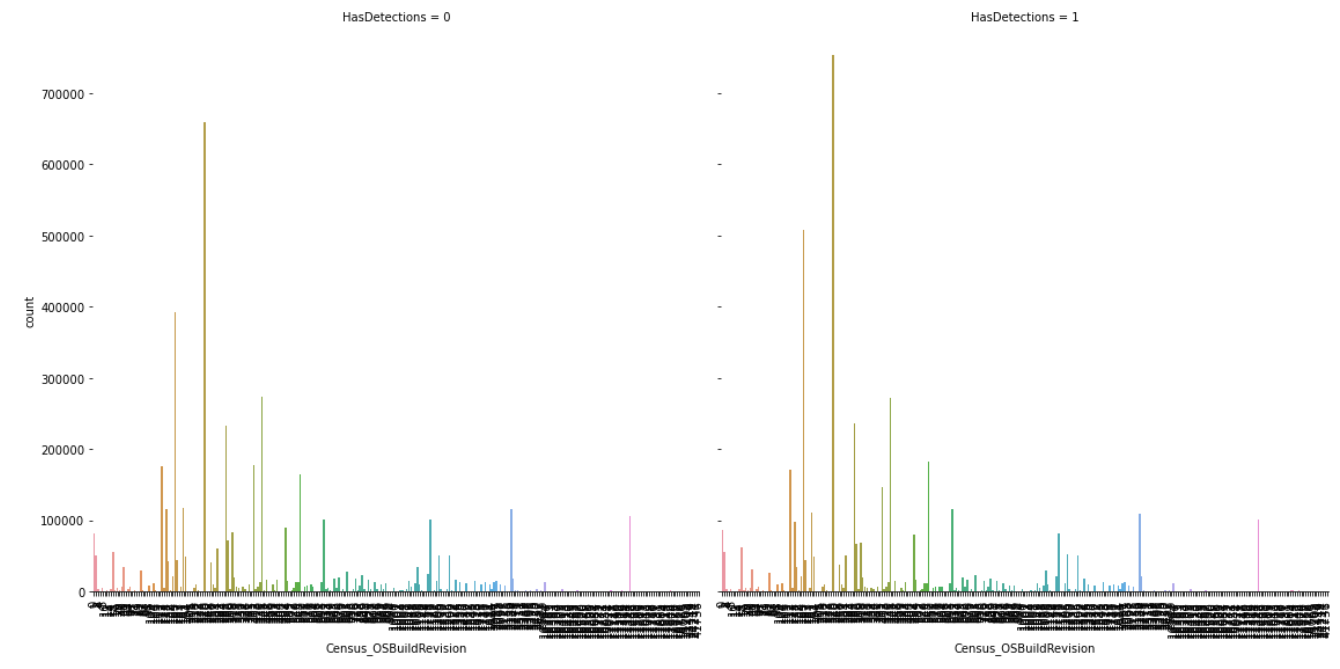


* EngineVersion and AppVersion: Both distributions seems to be almost the same

## Features related to build

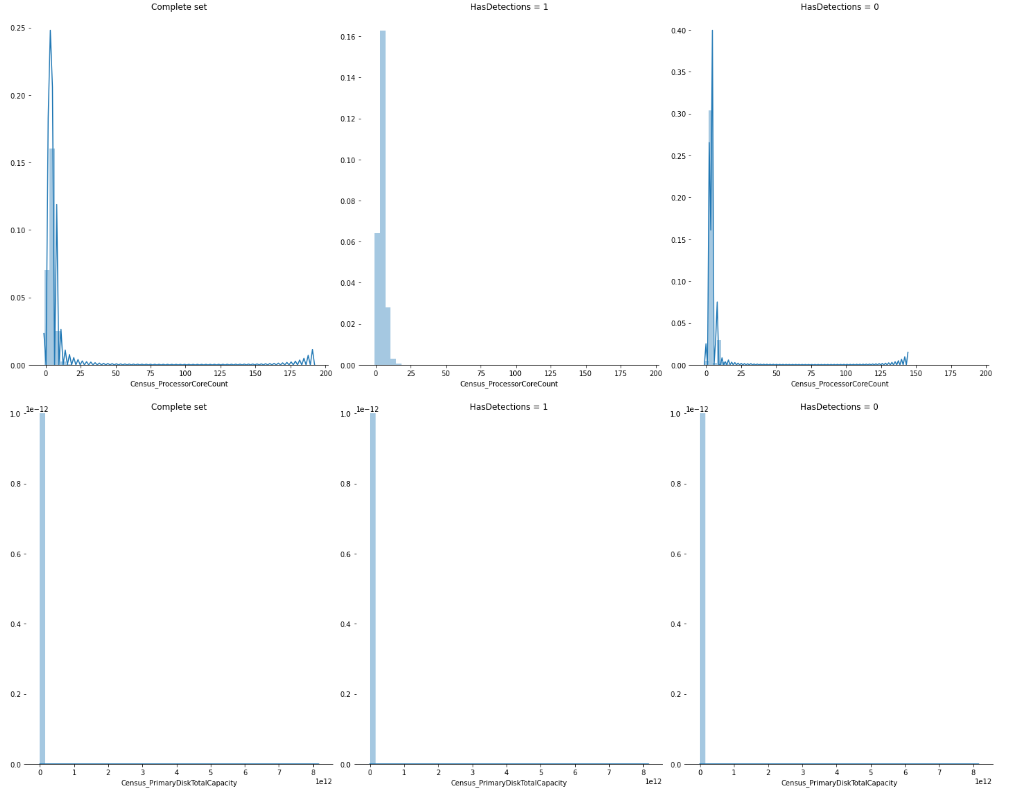




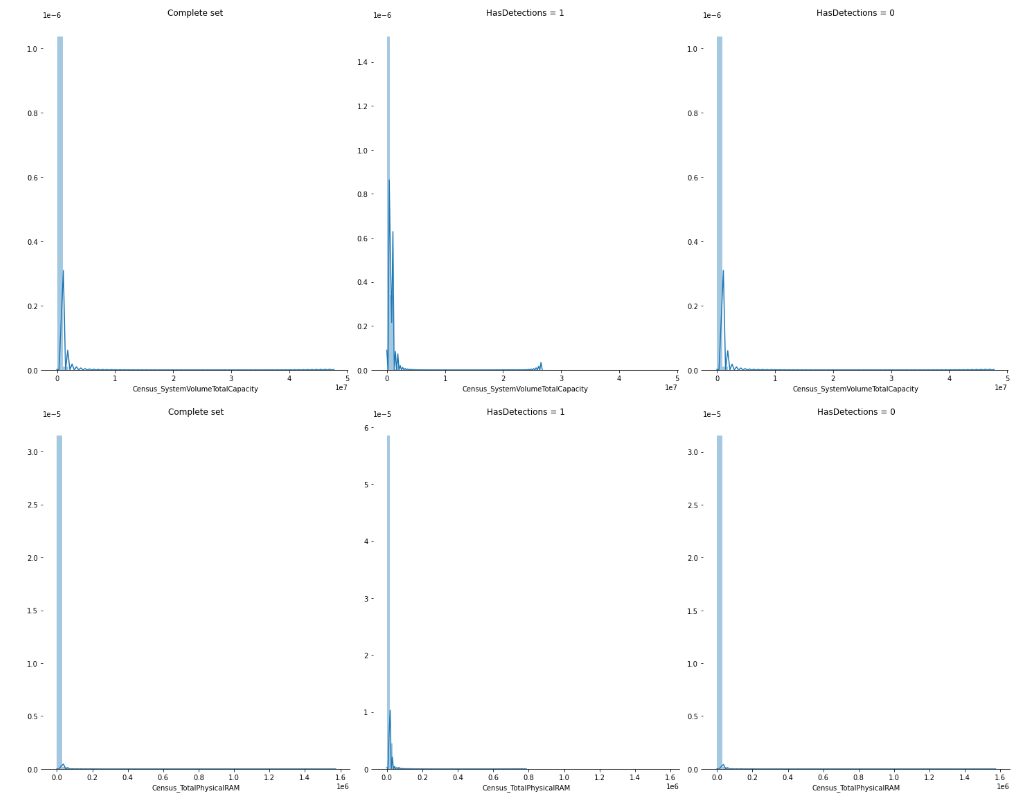


* OsBuild and Census\_OSBuildNumber: Not much going on here, seems to be something close to 50% malware detection and 50% with no malware detection.
* Census\_OSBuildRevision: Here things are more interesting, some categories have a similar label count, but some of them have a lot more of one label than the other.

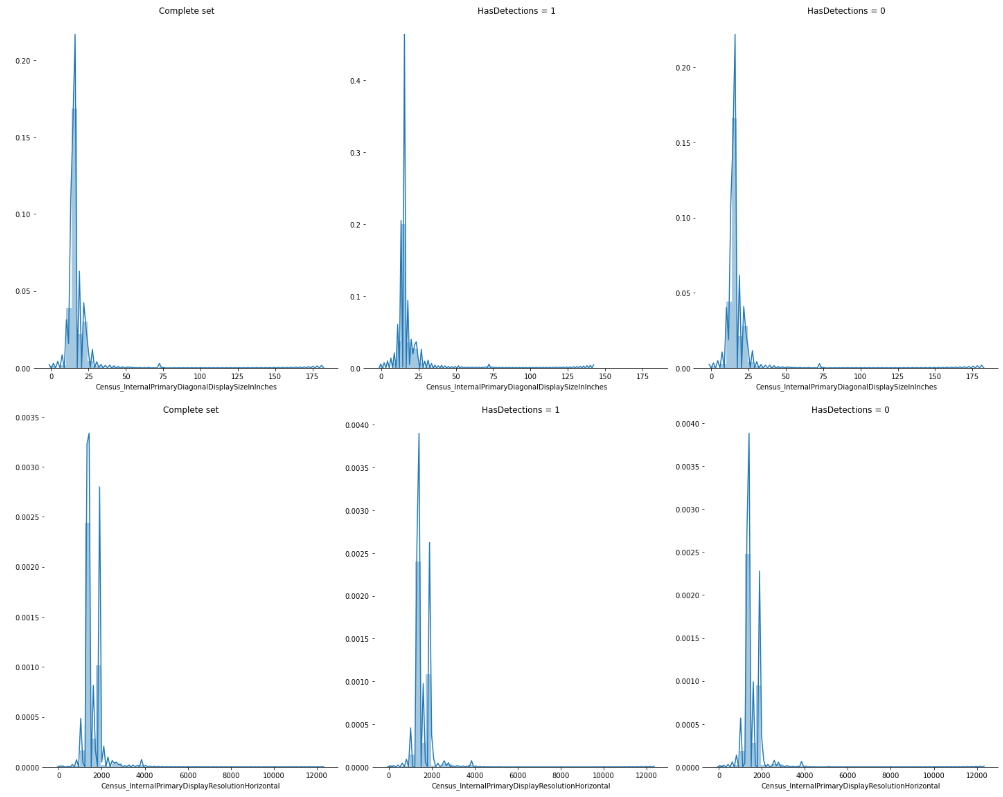
## Numerical Features:

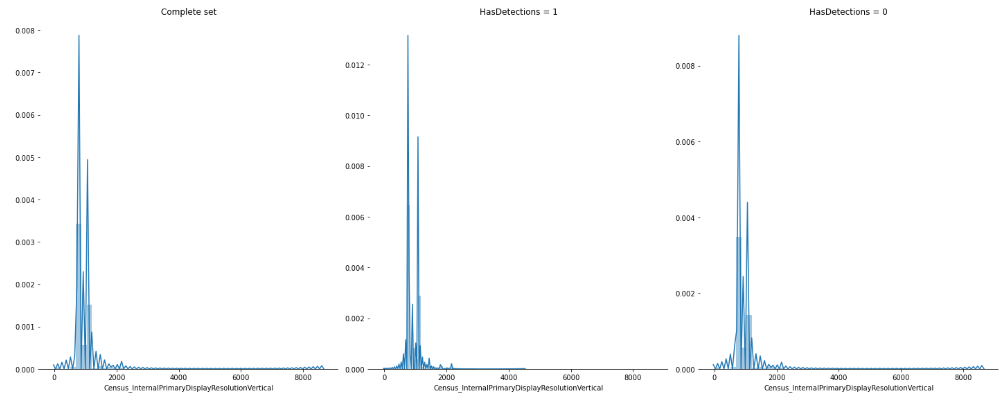


* Census\_ProcessorCoreCount: Malware detection seems to more skewed towards right and non-detection are more concentrated on the first 10 values.
* Census\_PrimaryDiskTotalCapacity: The distributions seems to be the same.



* Census\_SystemVolumeTotalCapacity and Census\_TotalPhysicalRAM: Malware detection seems to more concentrated on the beginning and non-detection seems to more skewed towards right.
* Census\_InternalPrimaryDiagonalDisplaySizeInInches: The distributions are very similar but non-detection have a longer right tail.





* Census\_InternalPrimaryDisplayResolutionHorizontal: The distributions seems to be the same.
* Census\_InternalPrimaryDisplayResolutionVertical: The distributions are very similar but non-detection have a longer right tail.
* Census\_InternalBatteryNumberOfCharges: The distributions seems to be the same.

# Data Preprocessing

## Feature creation

Some new features are created like aspect ratio, screen area, ram per processor etc.

## Label encoding

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated

## One-hot encoding:

It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature.

## Sampling

Since the data is huge we don't need to take rows for developing the model, we take a sample of 100k rows with ‘HasDetections’ as 0 and ‘HasDetections’ as 1, ie. equal distribution

## Dimensionality reduction

The dimension of the design matrix is huge such that training our model becomes computationally expensive. We use only a set of selected features for training our model. In stage 1 we use data\_vars function to give us an initial estimate of the importance of features. After that we threshold the importance and remove the features which lie below a certain threshold. In second step we use iterate\_vif function to remove highly correlated independent variables

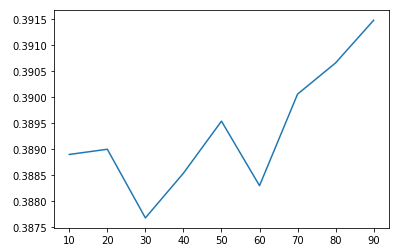
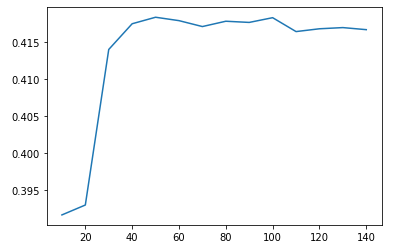
# Machine Learning Modelling

For classifying whether the machine can be infected with malware or not, three classification models were selected: Random Forest Model, Light Gradient Boost and Extreme Gradient Boost. Then the prediction of these models were evaluated with two metrics: AUC score and mean absolute error. Using GridSearchCV in cross validation was helpful for tuning hyperparameters in the models..

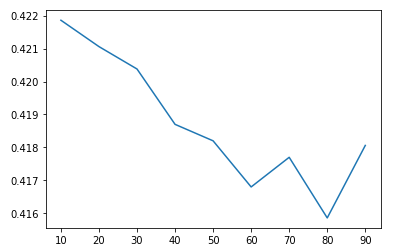
## Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

The hyperparameters for this model are max\_deapth, min\_sample\_leaf, n\_estimators. The range is large and this might take a lot of time to test out hyperparameters. So we will try to reduce the search space for the hyperparameters by individually conducting a validation testing and picking up a range for which we see the least error. This will narrow our search space.



max\_depth min\_sample\_leaf



n\_estimators

The Y-axis represents the error with respect to the values on the X-axis. Now to narrow down the values we will consider values surrounding them to the values which least error.

Redefining hyparameters:

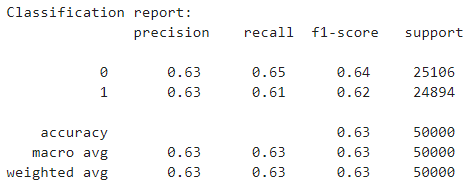
max\_depth = np.arange(10,50,10)

min\_samples\_leaf=np.arange(10,40,10)

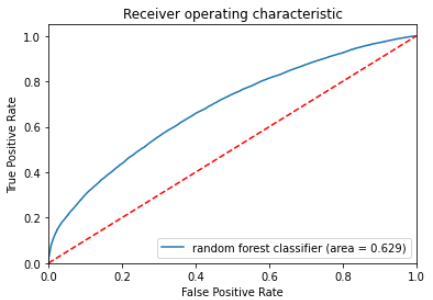
n\_estimators = np.array([60,70,80])

With the GridSearchCV a hyperparameter tuning is done and a model is developed.

The classification report for the random forest classifier



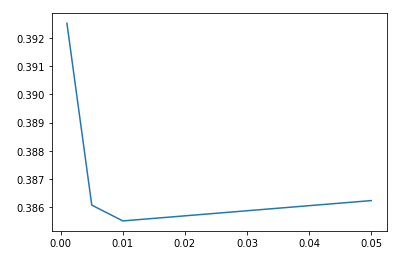
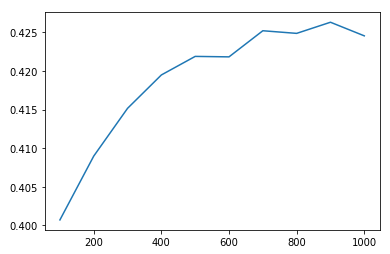
AUC is 0.629



## LightGBM Classifier

Decision trees, apply level-wise tree growth whereas LightGBM applies leaf-wise tree growth. This makes LightGBM faster.

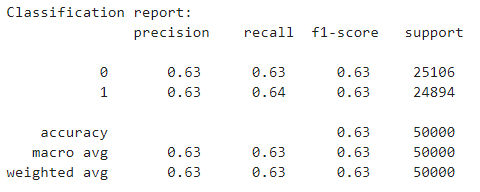
The parameters I consider for model development are num\_leaves, learning\_rate.



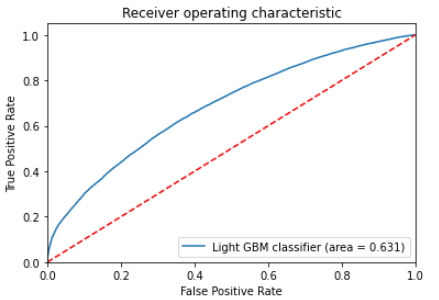
num\_leaves learning\_rate

According to the graphs above, hyperparameters range are narrowed down to make GridSearchCv work fast.

Classification report for LGBM Classifier:



AUC for LGBM classifier is 0.631



## XGBoost Classifier

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

Defining hyperparameters

min\_child\_weight=np.arange(1,11,1)

max\_depth=np.arange(1,11,1)

gamma=np.arange(0,6,0.5)

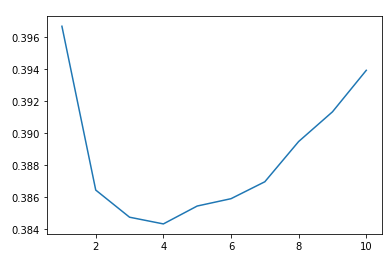
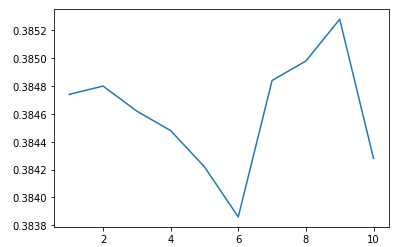
subsample=np.arange(0,1,0.2)

colsample\_bytree=np.arange(0,1.2,0.2)

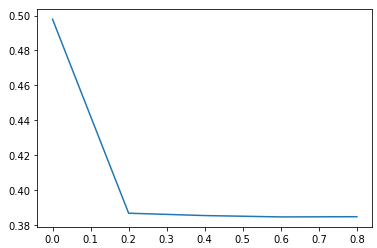
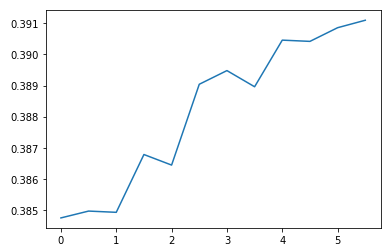
reg\_alpha=[0.01,0.05 ,0.1, 0.15,0.2,0.25,0.3,0.35,0.4]

learning\_rate=[0.01,0.05,0.1,0.15,0.2,0.25,0.3]

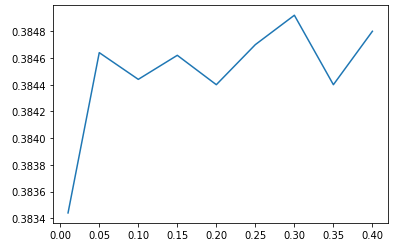
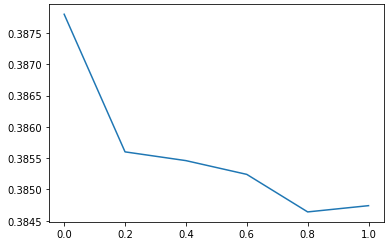
Plotting the graph for each hyperparameter again the error.



min\_child\_weight max\_depth



gamma subsample



colsample\_bytree reg\_alpha

After watching these plots, hyperparameter range is narrowed down.

Redefining hyperparameters

min\_child\_weight=[5,6,7]

max\_depth=[3, 4, 5]

gamma=[0.1,0.5, 1]

subsample=[0.6, 0.8, 1.0]

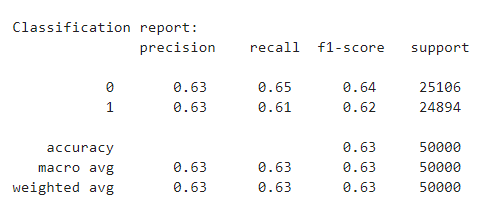
colsample\_bytree=[0.6, 0.8, 1.0]

reg\_alpha=[0.01,0.02,0.03]

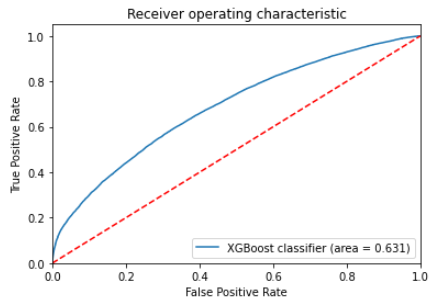
learning\_rate=[0.05,0.10,0.15]

Now with GridSearchCV, hyperparameters are tuned to develop the xgboost model with the largest AUC score.

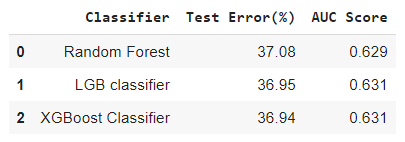
Classification report



AUC for XGBoost Classifier is 0.631



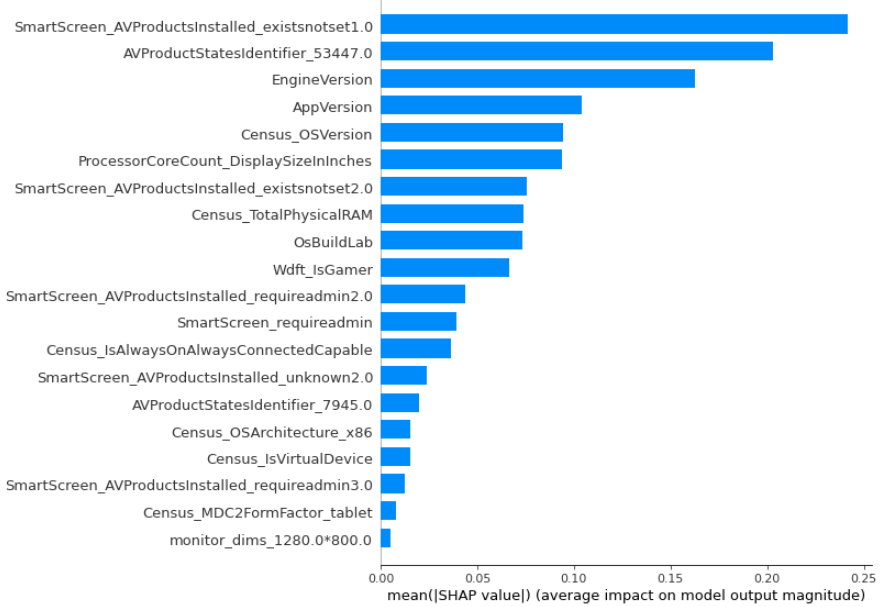
## Best Model

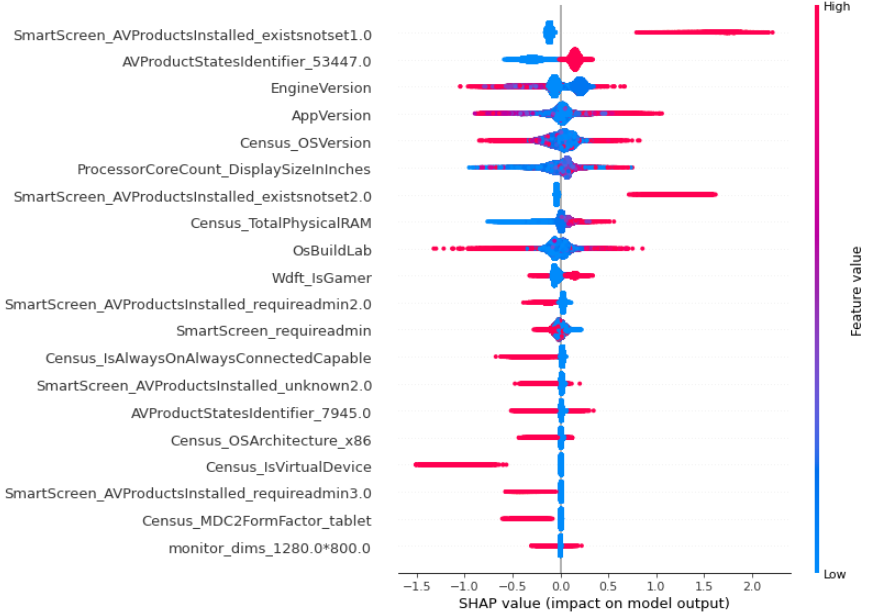


The above table shows three models with their metrics test error and AUC score when the three models are compared, best is the XGBoost model with highest AUC score and the least test error.

# Conclusion

## Feature Importance





The top three variables the model is dependent on mostly are smartscreen\_AVProdctsInstalled\_existsnotset1.0, AVProductStatesIdentifier\_53477.0 and Engine version

**smartscreen\_AVProdctsInstalled\_existsnotset1.0** is a new feature created by combining smartscreen and AVProductsinstalled, this feature tells one antivirus product is installed in a system and smartscreen is enabled or not. The SmartScreen filter built into Windows automatically scans applications, files, downloads, and websites, blocking known-dangerous content and warning you before you run unknown applications. This feature is positive correlation with the target variable.

**EngineVersion** is negatively correlated with target, as expected if the engine is old version that means there are no latest software updates to protect the system from the latest malware, so there is a high risk of being attacked by malwares if it is a low engine version.

**AVProductStatesIdentifier\_53477.0** is the configuration ID of the anti virus product. This specific configuration is to scan for the file before downloading it. This column tells whether it is enabled or disable. This feature is negatively correlated with the target variable.

## Recommendation for the user to avoid malware:

* The machine with one antivirus software it is better to enable smart screen feature
* In the antivirus software, set the configuration to enable which scans of the file before downloading it.
* Keep the microsoft defender to the latest version

## Future work

Here the model is developed only based on the properties of the windows machine, there is a chance that some malware attacks seasonally like for christmas there will be more ads on the browser and people tend to spend more time on the systems and may click on these vulnerable ads containing malwares. So, in the future, time series analysis is done and accordingly a model is developed.