Evaluation of Models in Windows Malware Threat Detection

Mohit Battu

Sai Keerthi Appili

Department of Computer Science Engineering

Blekinge Institute of Technology

Karlskrona

Sweden

2015

Abstract

In this paper, we propose a Machine Learning Model for malware detection. There has been a huge increase in the volume of malware, which poses a serious security threat. Few Malware types are viruses, worms, or other malicious programs. If a computer is infected by malware, hackers can harm consumers and companies in numerous ways which results in slowing down the performance of the system.

In this study, we investigate which strategies are essential to predict if a machine will soon be hit with malware. Under Machine learning, Decision Tree Classifier and Extreme Gradient Boosting are popular tools to build prediction models. Windows Malware threat detection can be solved by using a semi-supervised learning approach which consists of the models like Sequential Neural-Network, Decision Tree Classifier and Light Gradient Boosting Model facilitates us to detect Malware Threats.

For this Malware Detection project, we implemented 5 different types of Models namely Extreme Gradient Boosting, Decision Tree Classifier, Sequential Neural-Network, Random Forest Classifier, and Light Gradient Boosting Model. These above-referred models had achieved 65.79%, 58.8%, 51.75%, 50%, and 77.4% accuracies respectively for detecting a Malware Threat. Thus, we propose a Light Gradient Boosting Model, a Machine Learning approach to detect malware samples.

The proposed model achieved 77.8% accuracy in detecting Windows Malware Threats. The proposed model has got a 68% precision and 71% recall rate on the Train dataset whereas on Test Dataset proposed model has achieved 66% precision and 69% recall rate.

We conclude that on Windows challenging malware detection dataset demonstrates that our proposed Model can handle large-scale data, takes lower memory to run, and achieves better performance in comparison with the other 4 models. Thus, it is the best prediction model for Windows Malware Threat Detection.

**Keywords:**

Decision Tree Classifier, Extreme Gradient Boosting, Light Gradient Boosting Model, Malware Detection, Random Forest Classifier, and Sequential Neural-Network.

Acknowledgements

We would like to express our deep and sincere gratitude to Prof. Wlodek Kulesza, for his valuable guidance and support. He taught us the methodology to present the research works as clearly as possible and convincingly conveyed the concept of critical thinking. It was a great privilege and honour for us to study under his guidance. Our completion would not have been accomplished without his timely advice despite his busy schedule. Finally, we would like to thank our friends and family for their motivation and encouragement throughout the learning period.

Sai Keerthi Appili

Battu Mohit

Contents

Abstract 3

Acknowledgements 5

Contents 7

List of figures 8

List of tables 9

List of symbols 10

List of acronyms 11

1 Chapter: Introduction 13

2 Chapter: Survey of related work 14

2.1 DataSet: 14

2.2 Decision Tree Classifier: 15

2.3 Random Forest Classifier: 15

2.4 XGBoost: 15

2.5 LightGBM: 16

2.6 Sequential Neural Network: 16

3 Chapter: Problem statement, objectives and main contribution 17

4 Chapter: Solution 18

4.1 Modeling 18

4.2 Implementation or Application: 19

4.3 Validation or Verification 32

5 Chapter: Conclusion and future work 34

5.1 Conclusion: 34

5.2 Future Works: 34

6 Reference 35

List of figures

[Figure 4‑1: Importing Python Libraries 21](#_Toc54719126)

[Figure 4‑2: Loading Train DataSet 22](#_Toc54719127)

[Figure 4‑3: Column Types 22](#_Toc54719128)

[Figure 4‑4: Binary Label Count 23](#_Toc54719129)

[Figure 4‑5: Cardinality Graph 24](#_Toc54719130)

[Figure 4‑6: Cardinality Code 25](#_Toc54719131)

[Figure 4‑7: Missing Values Count 26](#_Toc54719132)

[Figure 4‑8: Dropping Columns Code 27](#_Toc54719133)

[Figure 4‑9: Dropping Rows with Null 27](#_Toc54719134)

[Figure 4‑10: Hyperparameter Code 28](#_Toc54719135)

[Figure 4‑11: LGB Accuracy Score 29](#_Toc54719136)

[Figure 4‑12: DTC Accuracy Score 30](#_Toc54719137)

[Figure 4‑13: RFC Accuracy Score 30](#_Toc54719138)

[Figure 4‑14: XGB Accuracy Score 31](#_Toc54719139)

[Figure 4‑15: SNN Accuracy Score 31](#_Toc54719140)

[Figure 4‑16: Confusion Matrix for LGBM 32](#_Toc54719141)

[Figure 4‑17: Model Metrics 33](#_Toc54719142)

[Figure 6‑1. Description of the figure. **Error! Bookmark not defined.**](#_Toc54719143)

List of tables

[Table 2‑1: DataSet Description 14](#_Toc54719097)

[Table 4‑1: DataSet Name and Type 19](#_Toc54719098)

[Table 4‑2: Model Metrics for Training 32](#_Toc54719099)

[Table 4‑3: Model Metrics for Testing 32](#_Toc54719100)

[Table 6‑1. Description of the table placed above the table. **Error! Bookmark not defined.**](#_Toc54719101)

List of symbols

List of acronyms

|  |  |
| --- | --- |
| **Acronym** | **Unfolding** |
| DTC | Decision Tree Classifier |
| RFC | Random Forest Classifier |
| LGB | Light Gradient Boosting |
| XGB | Extreme Gradient Boosting |
| SNN | Sequential Neural Network |
| CART | Classification and Regression Tree |

# Chapter: Introduction

Malware which is a hazard to the computer enters the user’s system without permission. Malware is also known as “Malicious Software”. It is a great threat in the computing world.

The performance of the system also gets degraded when the malicious software enters the system, thereby it scans for the vulnerabilities present in the operating system and performs unauthorized actions on the system. The volume of malware is evolving rapidly and is getting out of control. Malware is abundant over the internet and can easily infect various hosts simultaneously. Once the Malware enters the user’s system, then the system results in unresponsive, and the application halts from opening.

Malware is user-friendly for hackers. Nowadays, it can earn more income for attackers by providing them to steal user’s login credentials, credit card information, browser auto-fill data, and cryptocurrency wallets.

More and more companies are trying different strategies to solve this problem of Malware, but due to the large volume of Malware, it can still escape even complex forms of authentications.

A machine learning-based model “Light Gradient Boosting Model” is used to predict if the machine gets hit by the malware using certain features that are dependent on the identification of Malware.

# Chapter: Survey of related work

In this section, there is a brief description of the Malware Dataset and the model's definition.

## DataSet:

Malware Threat Data Set consists of 83 features and 8.92 million number of records. The total size of the Data Set is approximately 7.89GB. This Data Set was mainly divided into three main parts consisting of Numerical, Binary, and Categorical Columns. There are 7 numerical, 21 binary, and 54 categorical columns which sum up to the 83 features in our Dataset. Each row in this dataset corresponds to a windows machine, which is uniquely identified by a MachineIdentifier. *MachineIdentifier* is the first column in our dataset and is called an independent variable. *HasDetections* is the target variable and it is present in the last column of the Malware dataset. *HasDetections* is a binary column which indicates that Malware was Detected on the machine in the form of “0” and “1”. Here “0” indicates that Malware is not found and “1” represents Malware is found, see Table 2-1 [1].

Table 2‑1: DataSet Description

|  |  |  |
| --- | --- | --- |
| **Serial#** | **MachineIdentifier** | **Detected#** |
| 1 | 0000028988387b115f69f31a3bf04f09 | 0 |
| 2 | 000007535c3f730efa9ea0b7ef1bd645 | 0 |
| 3 | 000007905a28d863f6d0d597892cd692 | 0 |
| 4 | 00000b11598a75ea8ba1beea8459149f | 1 |
| 5 | 000014a5f00daa18e76b81417eeb99fc | 1 |

This Data Set was downloaded from the Kaggle repository under the name of Microsoft Malware Prediction.

## Decision Tree Classifier:

A Decision Tree is a simple representation of classifying examples. The Decision Tree has two major classes namely classification and regression tree. These two terms together are called CART. This CART was first invented in 1984 by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone [2]. We can address the amount of uncertainty in the data for set S as

|  |  |
| --- | --- |
|  | (‑) |

Here *p(x)* is the proportion of the number of elements in class *x* to the number of elements in set *S* and *x* is the set of classes in *S*.

## Random Forest Classifier:

Tin Kam Ho had created the first algorithm for the random forests in the year 1995. Random forests are often used as *BlackBox* models in industries since they produce sufficient estimates across a wide range of data [3]. We can address the estimated probability for predicting class k for a sample as

|  |  |
| --- | --- |
|  | (‑) |

where *p(k|x)* is the estimated density of class labels of the leaf of the *tth* tree.

## XGBoost:

Tianqi Chen had created the XGBoost algorithm. This algorithm was released on 27th March 2014. XGBoost algorithm was developed with the help of C++, Java, Python, R, Julia, and Scala. It provides a parallel tree boosting that solve many data science problems in a fast and accurate way. We can address that the objective function is a sum of the specific loss function and regularization terms for all predictions. Mathematically, it can be represented as:

|  |  |
| --- | --- |
|  | (‑) |

where *yi* is original values and is predicted values. Here is a sum of the regularization term.

## LightGBM:

Guolin Ke had created the Light GBM algorithm. It was first released in the year 2016. This algorithm was developed with the help of C++, Python, R, and C. It provides a faster training speed and higher efficiency. It is [histogram-based](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html) and places continuous values into discrete bins, which leads to faster training and more efficient memory usage [4].

## Sequential Neural Network:

Warren McCulloch and Walter Pitts had created a computation model for neural networks in the year 1943. These algorithms were developed with the help of threshold logic [5]. We can address the cost function of a sequential neural network as

|  |  |
| --- | --- |
|  | (‑) |

Here cost function sums over all samples from *i* to *n*, where *y* is a true value and is a predicted value.

# Chapter: Problem statement, objectives and main contribution

Malware is present in huge quantities and can easily affect as many hosts as possible. Malware avoids most of the security measures undertaken by the user thereby forcing the anti-malware groups to build more robust software to detect these malware attacks. A foremost part of protecting a computer system from a malware attack is to identify whether a given piece of file is malware or not.

**Research Problem**: How to develop a Machine learning Model for detecting the Malware threats on Windows Operating System?

1. For detecting malware threats in a windows system, “Microsoft Malware Prediction” has provided a train and test data set consisting of more than a billion observations which are made available in the Kaggle Dataset Repository.
2. Observed values present in the training dataset of Microsoft Malware Prediction are first refined by discarding missing values then feature engineered with the help of data science tools resulting in improved efficiency on unseen data.
3. Malware threat detection can be solved by using a semi-supervised learning approach which consists of the models like Sequential Neural-Network, Decision Tree Classifier, Random Forest Classifier, Extreme Gradient Boosting, and Light Gradient Boosting Model enables us to detect whether a windows machine will be soon hit with malware or not.
4. The above-referred models are different kinds of algorithmic approaches where only one of the best performing models shall be considered for predicting Windows Malware Threat Detection.

The main contributions of the project are as follows:

* Implementation of the model in Python 3 environments on Jupyter Notebook (Anaconda 3).
* The implementation of python libraries in the project are Pandas, NumPy, Matplotlib, Seaborn, Scikit-Learn, Keras, and Light Gradient Boosting [6].
* Validation of the model is done by using the Confusion matrix and Model Metrics.

# Chapter: Solution

We anticipate that the Light GBM model can give better results and accuracy than Decision Tree Classifier, Random Forest, XGBoost, Sequential Neural Networks. The dataset we gathered is of 8.9 million records with many unknown values for certain features. To tackle the large dataset Light GBM seems more appropriate which takes lower memory to run and achieves better performance.

## Modeling

Import

libraries

Evaluation of model

Training the model

Label encoding

Testing the model

Column cardinality

Null values count

Label Count

Load Malware Dataset

## Implementation or Application:

Implementation of algorithm is done in Jupyter notebook using a python programming language. Malware Dataset downloaded from the Kaggle Repository [1].

Table 4‑1: DataSet Name and Type

|  |  |
| --- | --- |
| **Column Name** | **Type** |
| 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 |
| **Column Name** | **Type** |
| SMode | Float16 |
| IeVerIdentifier | Float16 |
| SmartScreen | Category |
| Firewall | Float16 |
| UacLuaenable | Float32 |
| Census\_MDC2FormFactor | Category |
| Census\_OSVersion | Category |
| Census\_OSArchitecture | Category |
| Census\_OSBranch | Category |
| Census\_OSBuildNumber | Int16 |
| Census\_OSBuildRevision | Int32 |
| Census\_OSEdition | Category |
| Census\_OSSkuName | Category |
| Census\_OSInstallTypeName | Category |
| Census\_OSInstallLanguageIdentifier | Float16 |
| Census\_OSUILocaleIdentifier | Int16 |
| Census\_OSWUAutoUpdateOptionsName | Category |
| Census\_IsPortableOperatingSystem | Int8 |
| Census\_GenuineStateName | Category |
| Census\_ActivationChannel | Category |
| Census\_IsFlightingInternal | Float16 |
| Census\_IsFlightsDisabled | Float16 |
| Census\_FlightRing | Category |
| Census\_ThresholdOptIn | Float16 |
| Census\_FirmwareManufacturerIdentifier | Float16 |
| Census\_FirmwareVersionIdentifier | Float32 |
| Census\_IsSecureBootEnabled | Int8 |
| Census\_IsWIMBootEnabled | Float16 |
| Census\_IsVirtualDevice | Float16 |
| Census\_IsTouchEnabled | Int8 |
| Census\_IsPenCapable | Int8 |
| Census\_IsAlwaysOnAlwaysConnectedCapable | Float16 |
| Wdft\_IsGamer | Float16 |
| Wdft\_RegionIdentifier | Float16 |
| Census\_OEMNameIdentifier | Float16 |
| Census\_OEMModelIdentifier | Float32 |
| Census\_ProcessorCoreCount | Float16 |
| Census\_ProcessorManufacturerIdentifier | Float16 |
| **Column Name** | **Type** |
| Census\_ProcessorModelIdentifier | Float16 |
| Census\_ProcessorClass | Category |
| Census\_PrimaryDiskTotalCapacity | Float32 |
| Census\_PrimaryDiskTypeName | Category |
| Census\_SystemVolumeTotalCapacity | Float32 |
| Census\_HasOpticalDiskDrive | Int8 |
| Census\_TotalPhysicalRAM | Float32 |
| Census\_ChassisTypeName | Category |
| Census\_InternalPrimaryDiagonalDisplaySizeInInches | Float16 |
| Census\_InternalPrimaryDisplayResolutionHorizontal | Float16 |
| Census\_InternalPrimaryDisplayResolutionVertical | Float16 |
| Census\_PowerPlatformRoleName | Category |
| Census\_InternalBatteryType | Category |
| Census\_InternalBatteryNumberOfCharges | Float32 |
| Census\_DeviceFamily | Category |
| HasDetections | Int8 |

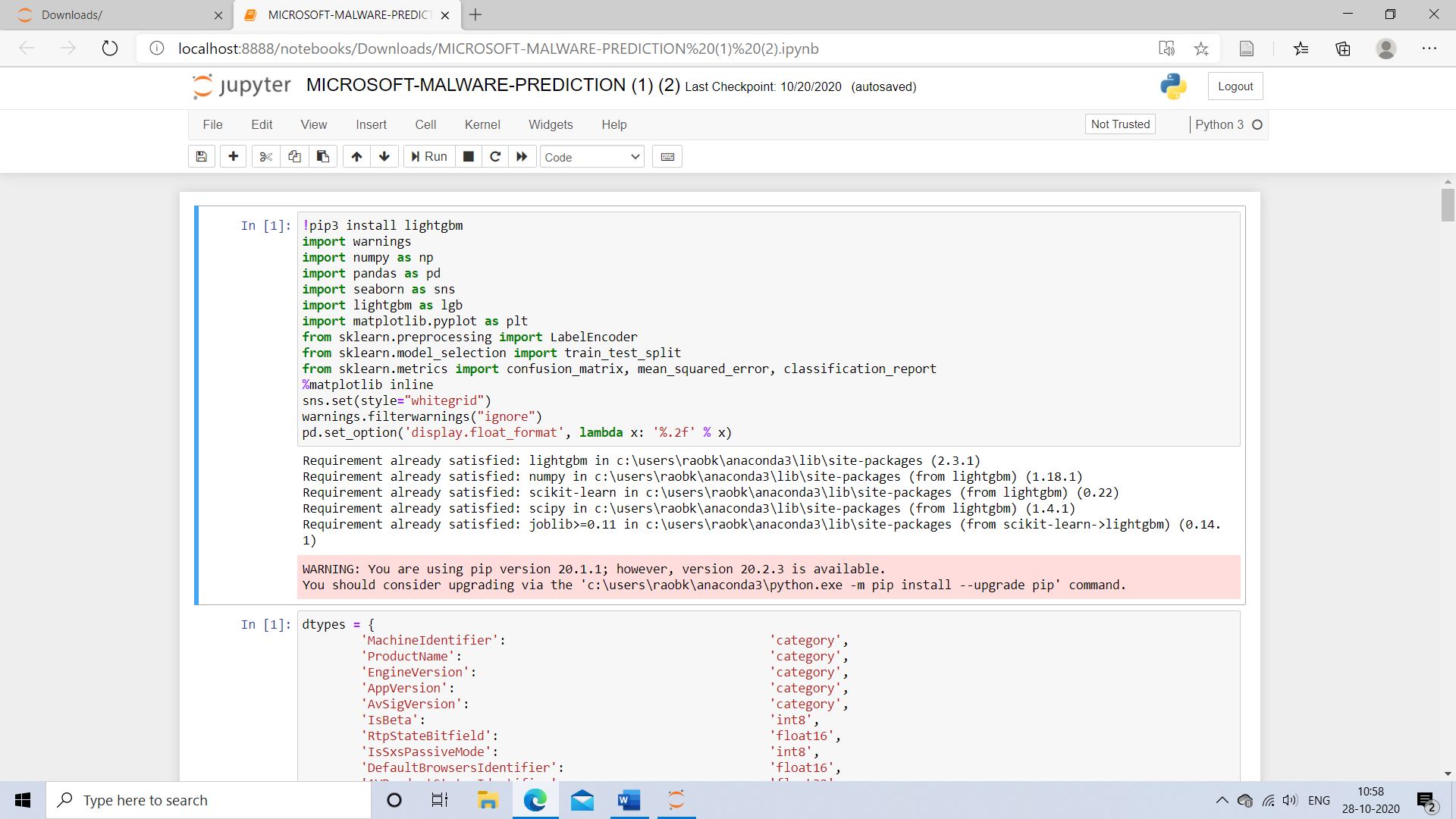
The “HasDetections” column is the target variable used to predict if malware is hit. Required libraries are to be installed before the implementation of the model. 

Figure 4‑1: Importing Python Libraries

**Loading DataSet**:

The dataset is imported to Jupyter notebook using panda’s module.

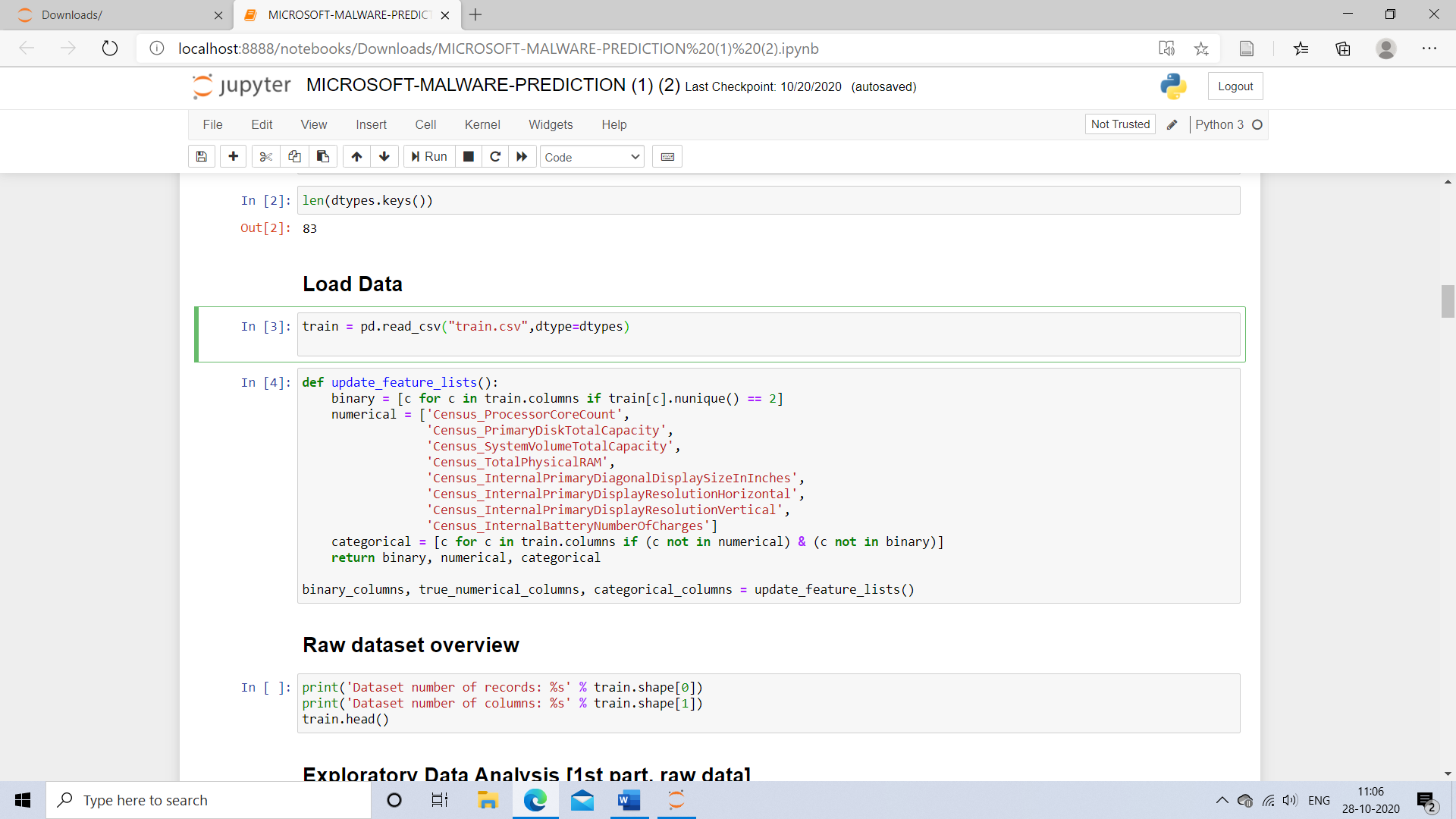


Figure 4‑2: Loading Train DataSet

**Exploratory Data Analysis:**

The data comprises more categorical columns than numerical columns and binary columns.

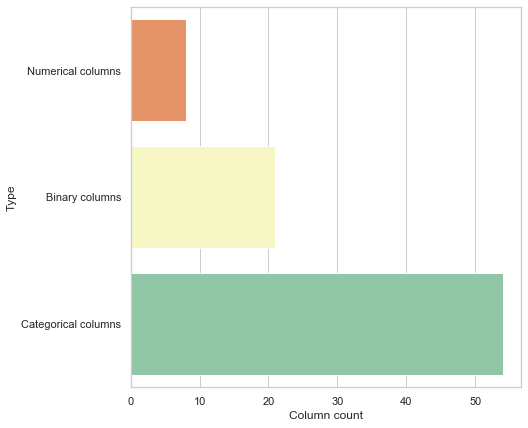


Figure 4‑3: Column Types

**Label Count**:

The target variable “HasDetections” binary value counts are inspected to find if any bias exists or an imbalance of data. There is balanced data for “HasDetections” without any problem of misclassification in our dataset. Binary count of 0’s and 1’s in the “HasDetections” column is in the same proportion.

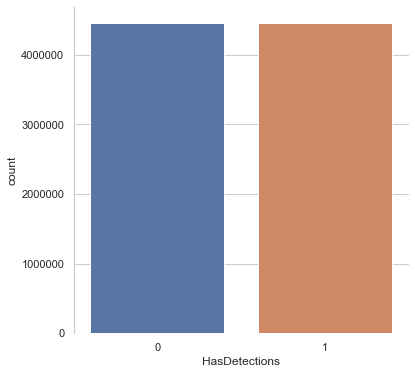


Figure 4‑4: Binary Label Count

**Column Cardinality**:

Each category cardinality count is noticed and analyzed to remove columns with high cardinality.

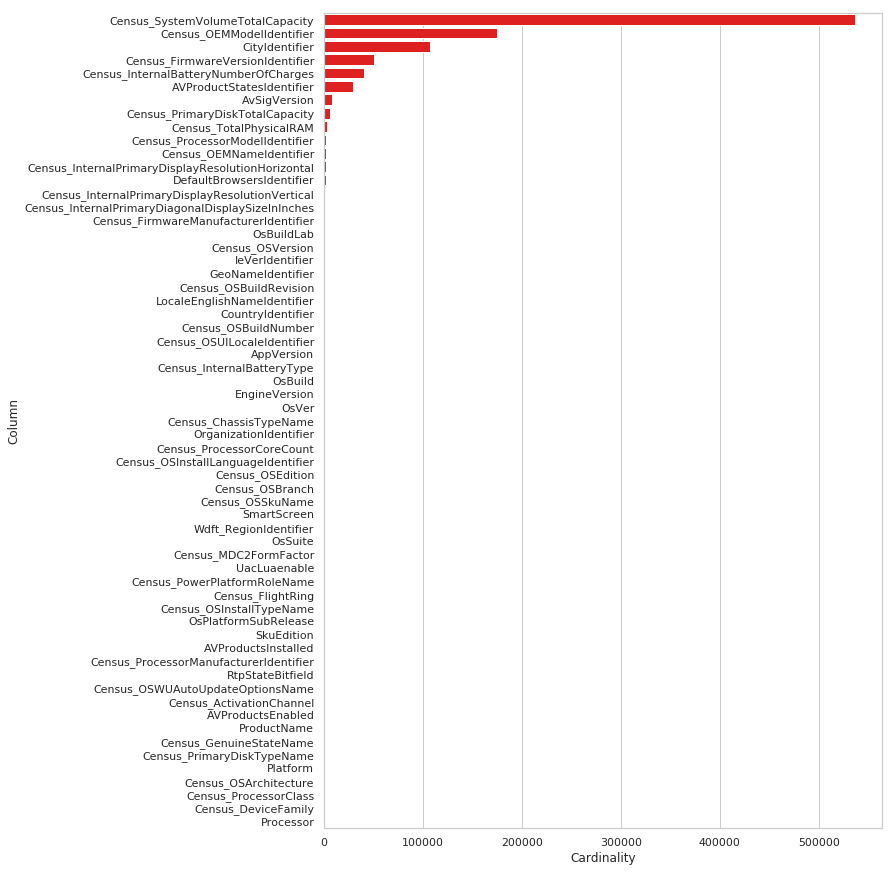


Figure 4‑5: Cardinality Graph

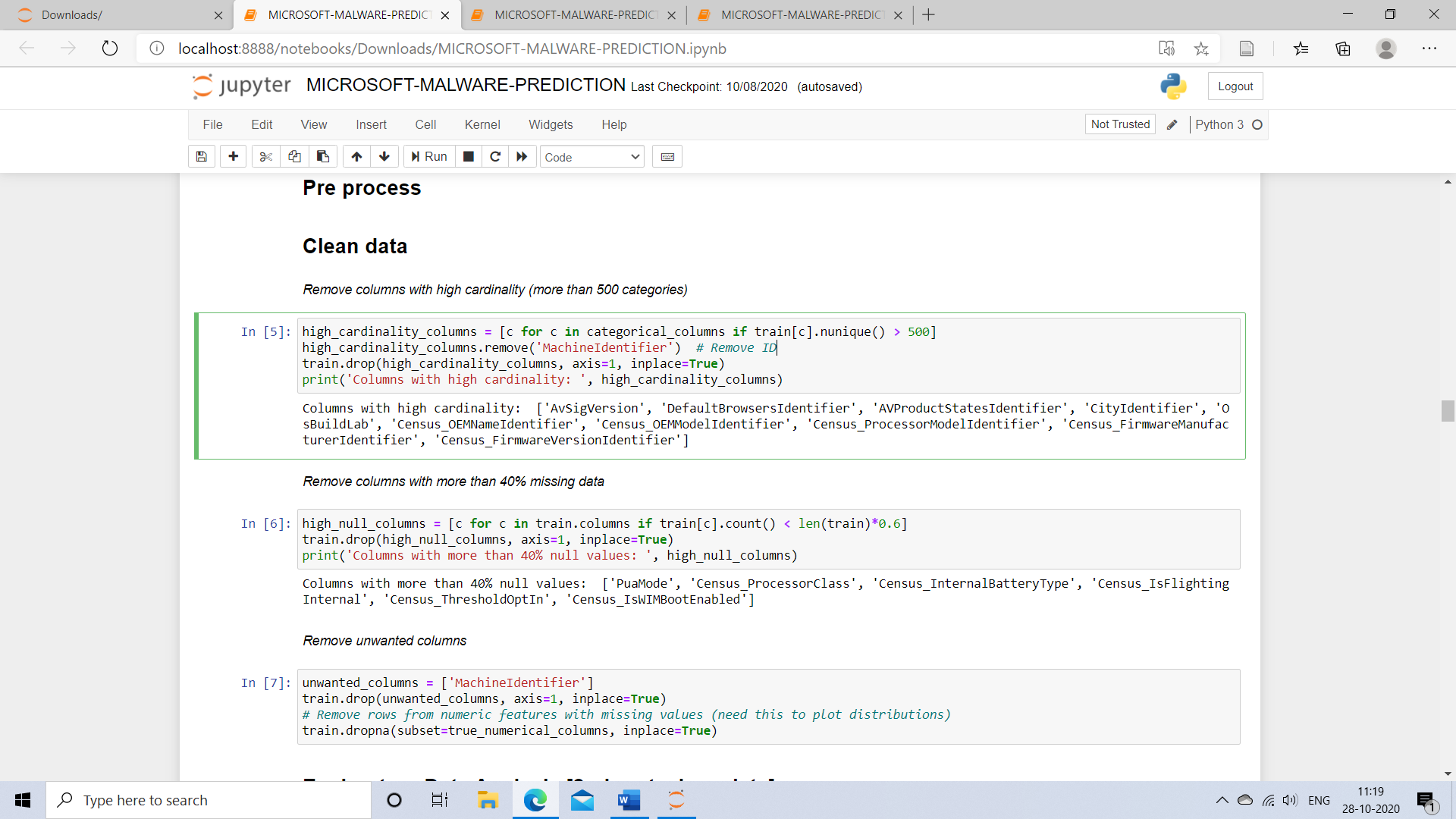


Figure 4‑6: Cardinality Code

Removing columns with cardinality number greater than 500 which we assumed as high cardinality.

**Counting the number of null values:**

The number of null values is analyzed for each feature to drop the columns with more percentage of missing values.

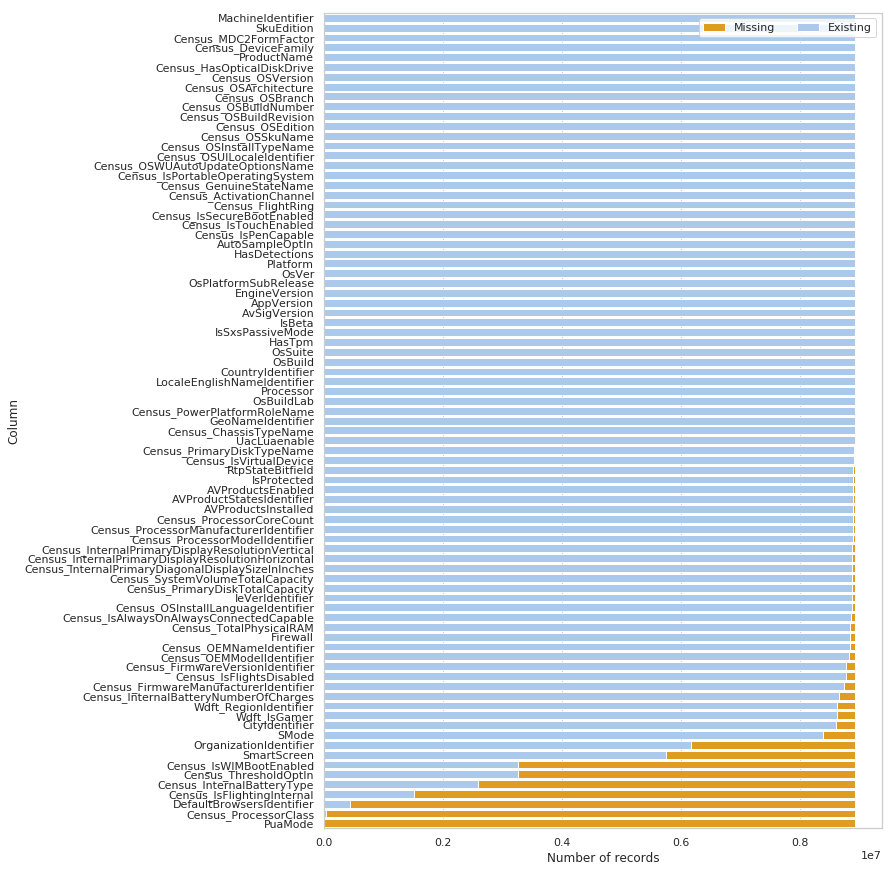


Figure 4‑7: Missing Values Count

**Removing rows with a null value:**

All the rows where the null values exist are removed without leading to any problem while training the model.

Dropping the columns with more than 40% missing data.

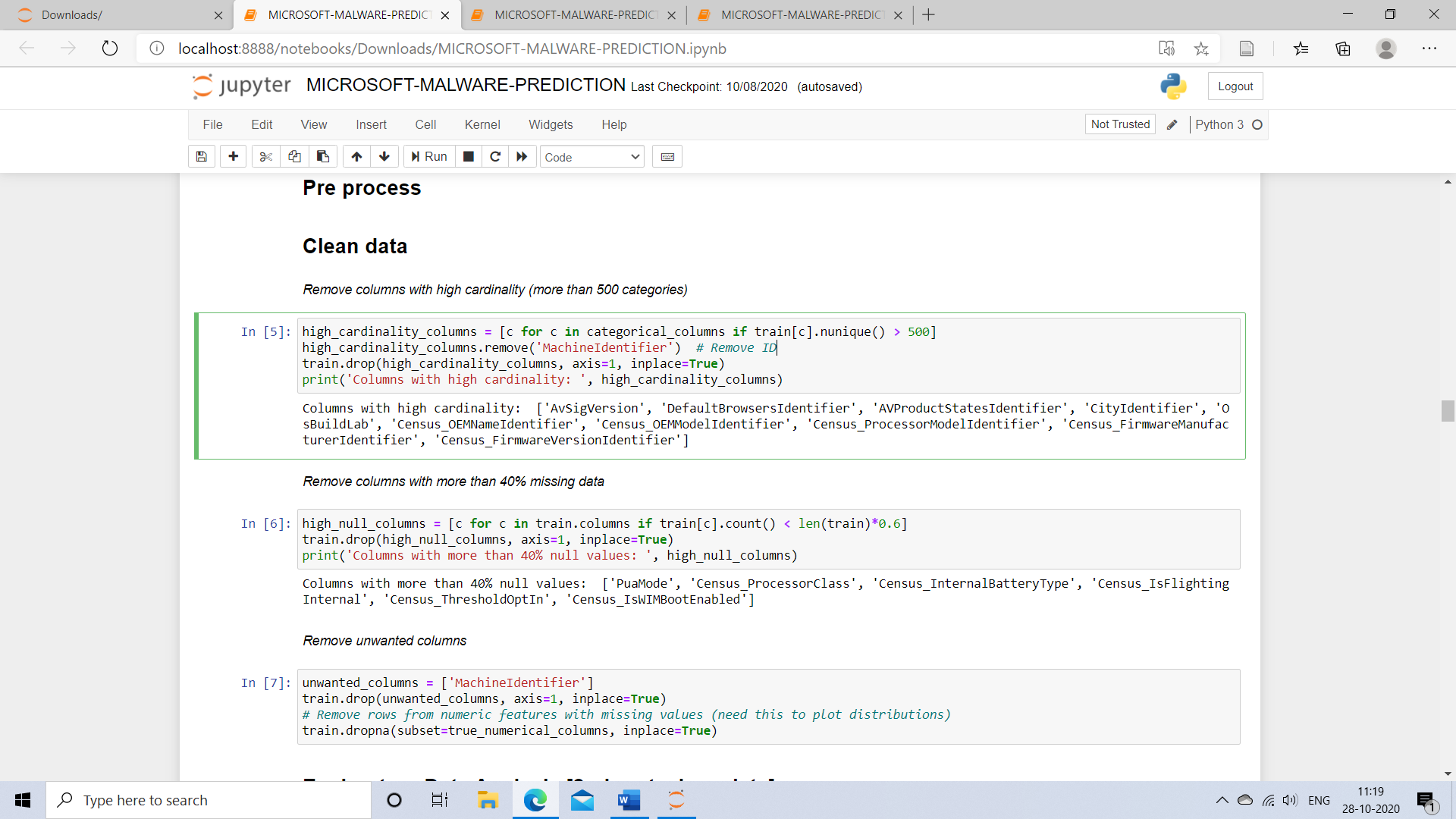


Figure 4‑8: Dropping Columns Code

Dropping the rows with null values

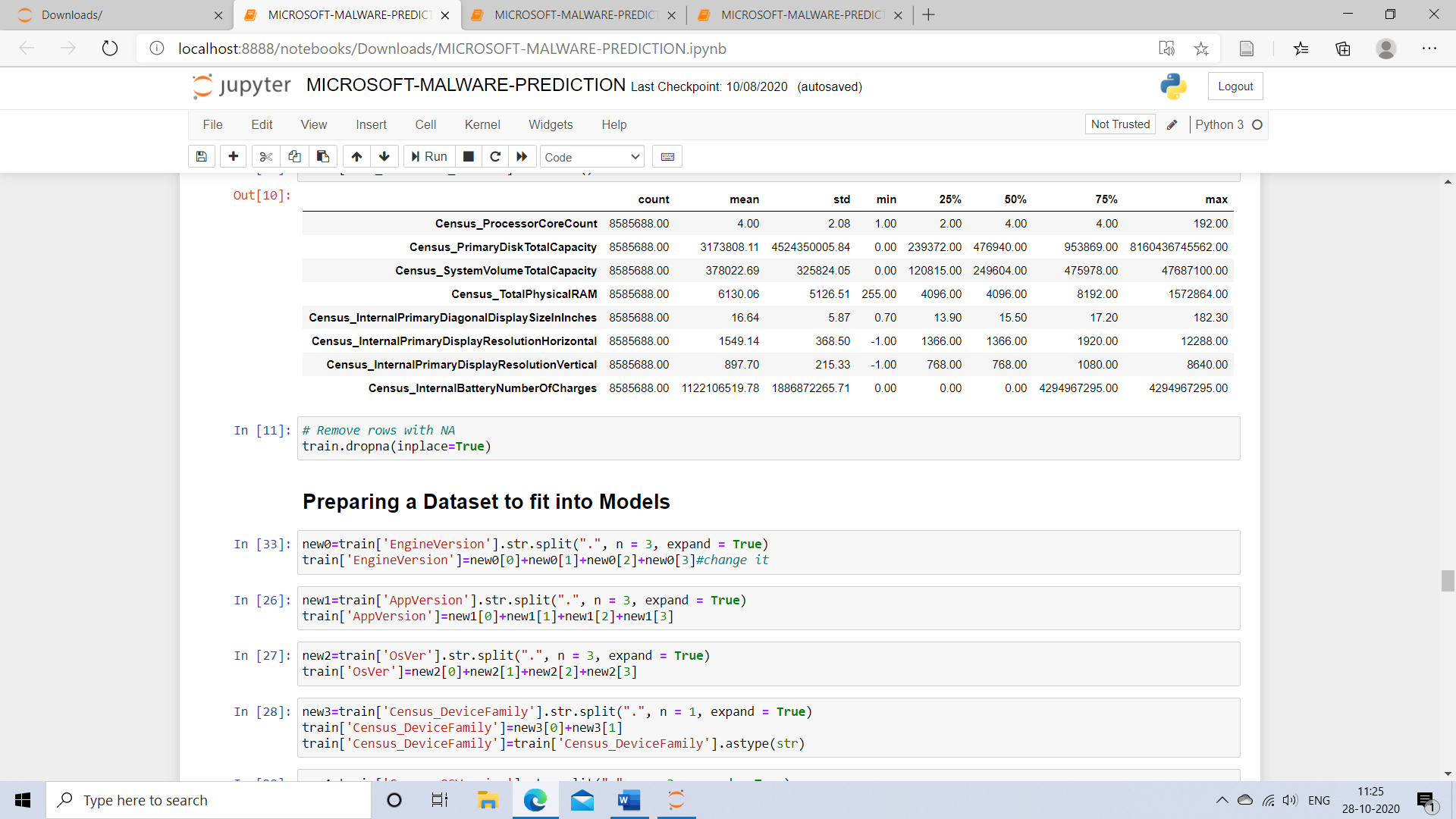


Figure 4‑9: Dropping Rows with Null

**Model Implementation**:

1. **Light GBM**:

Training data is fitted into the LightGBM Model. The hyperparameters are given to fine-tune the model while training the data to give much better accuracy.

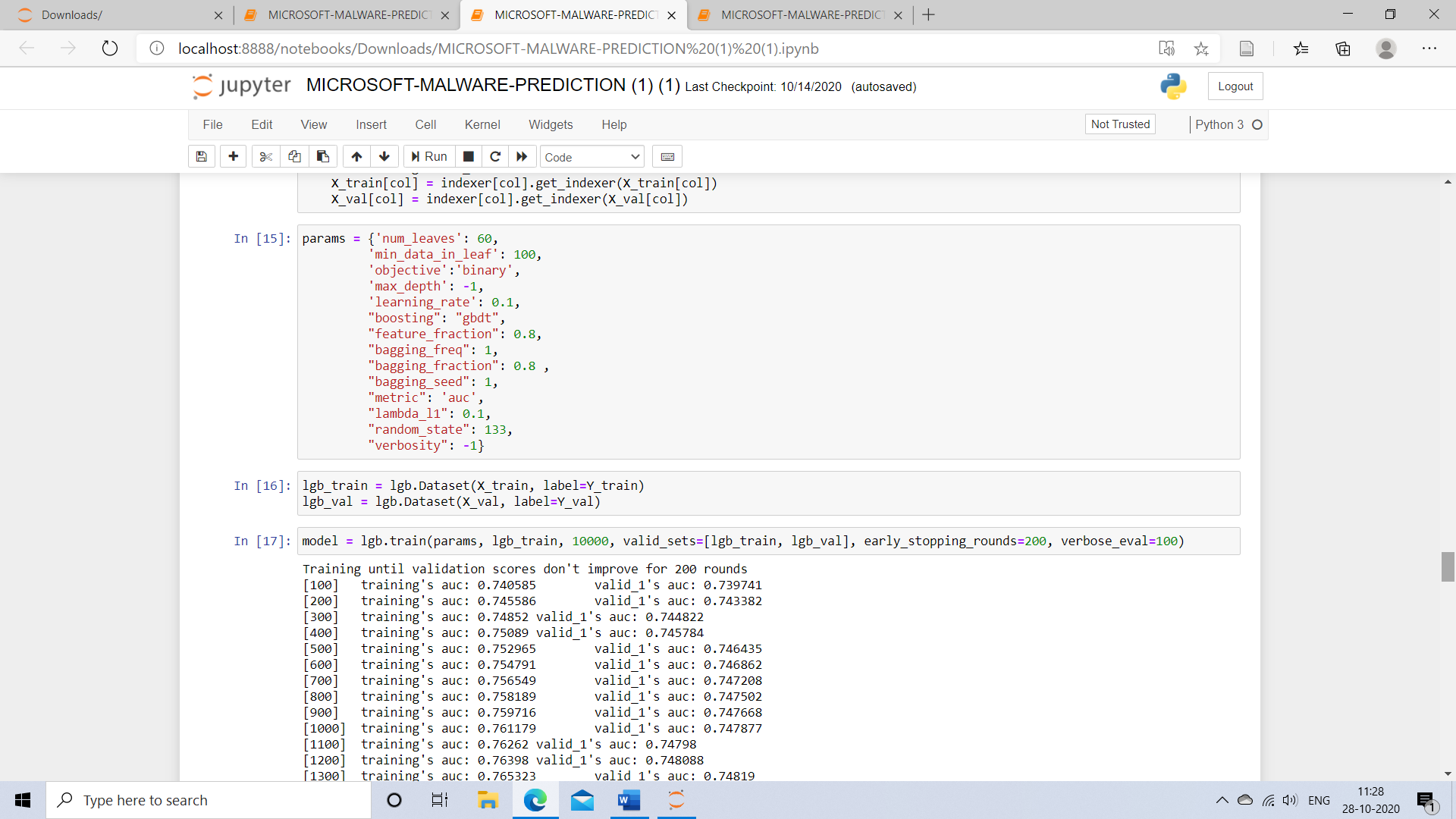


Figure 4‑10: Hyperparameter Code

After training the data the Light GBM accuracy is approximately 74%.

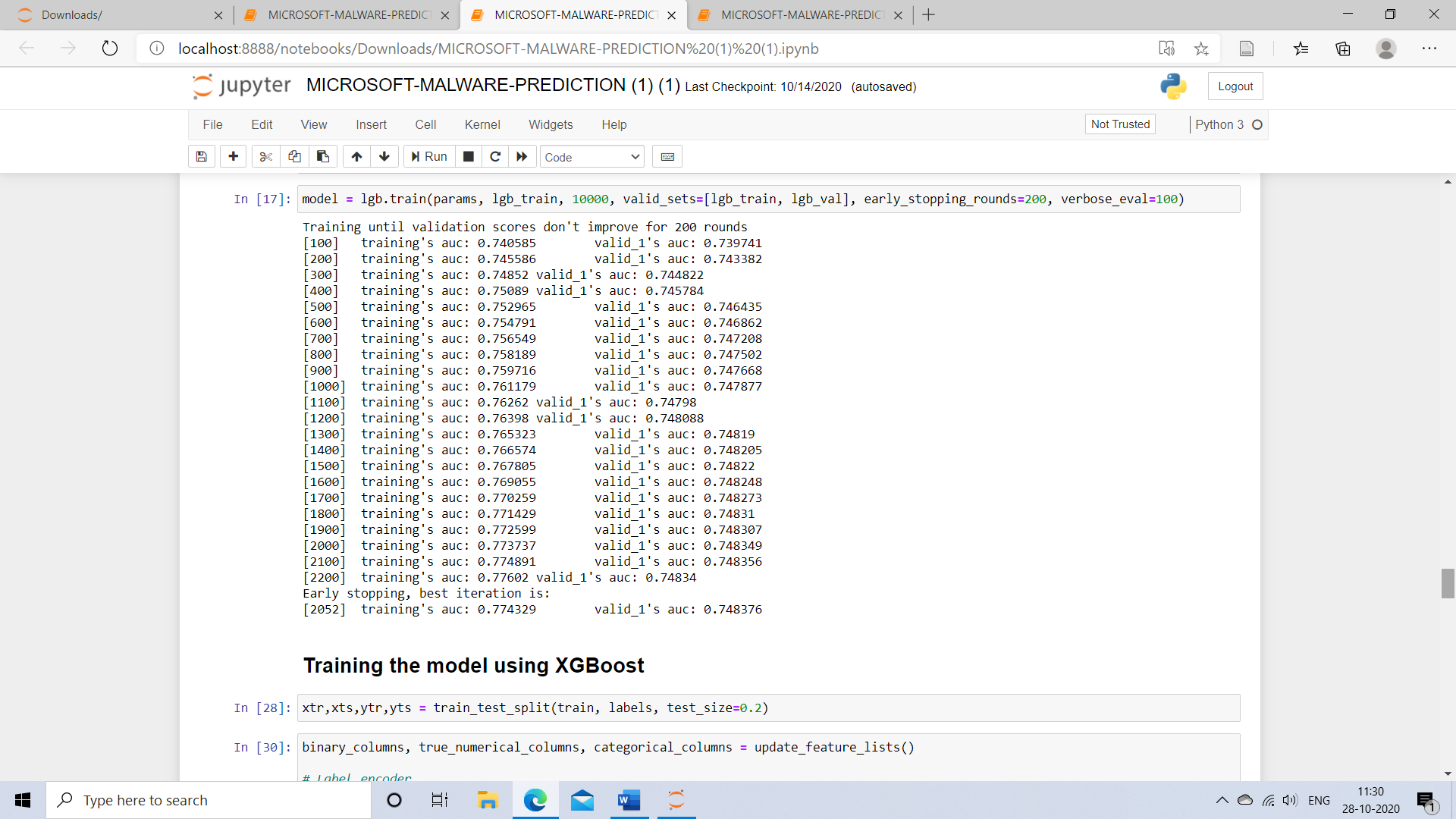


Figure 4‑11: LGB Accuracy Score

1. **Decision Tree Classifier**:

Label encoding of categorical variables is done to fit in the Decision Tree Classifier. Training data is fitted into the Decision Tree classifier. The model showed an accuracy of approximately 58%.

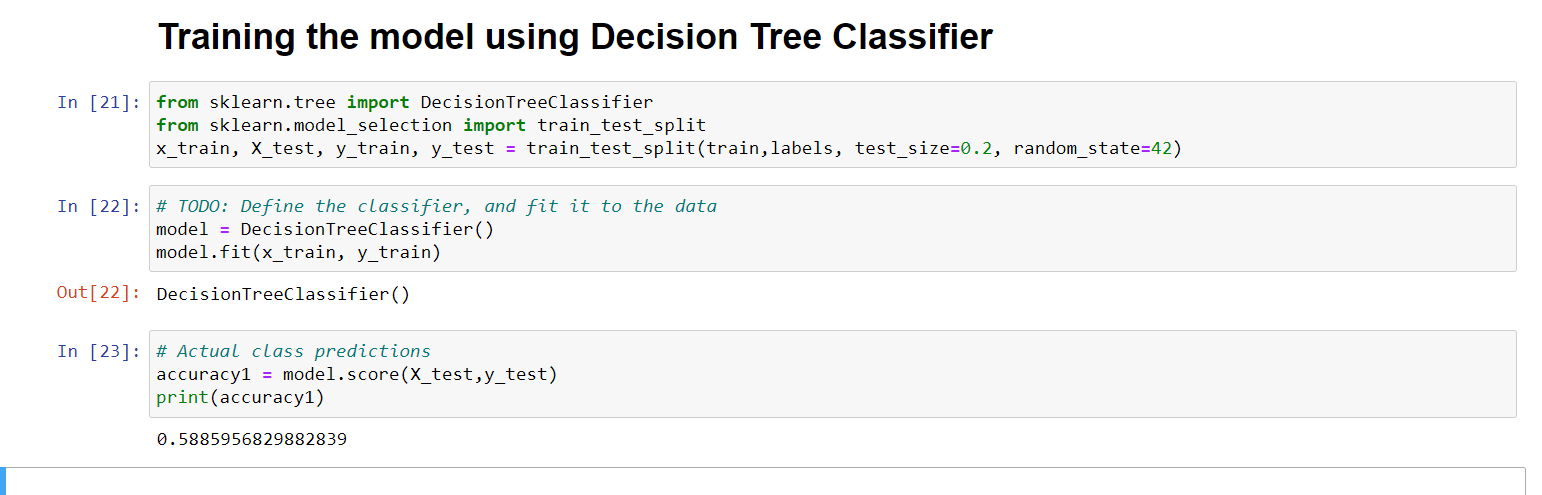


Figure 4‑12: DTC Accuracy Score

1. **Random Forest Classifier**:

Converting all categorical columns to numerical type with label encoders. The training data after fitting into the model shown an accuracy of approximately 50%.

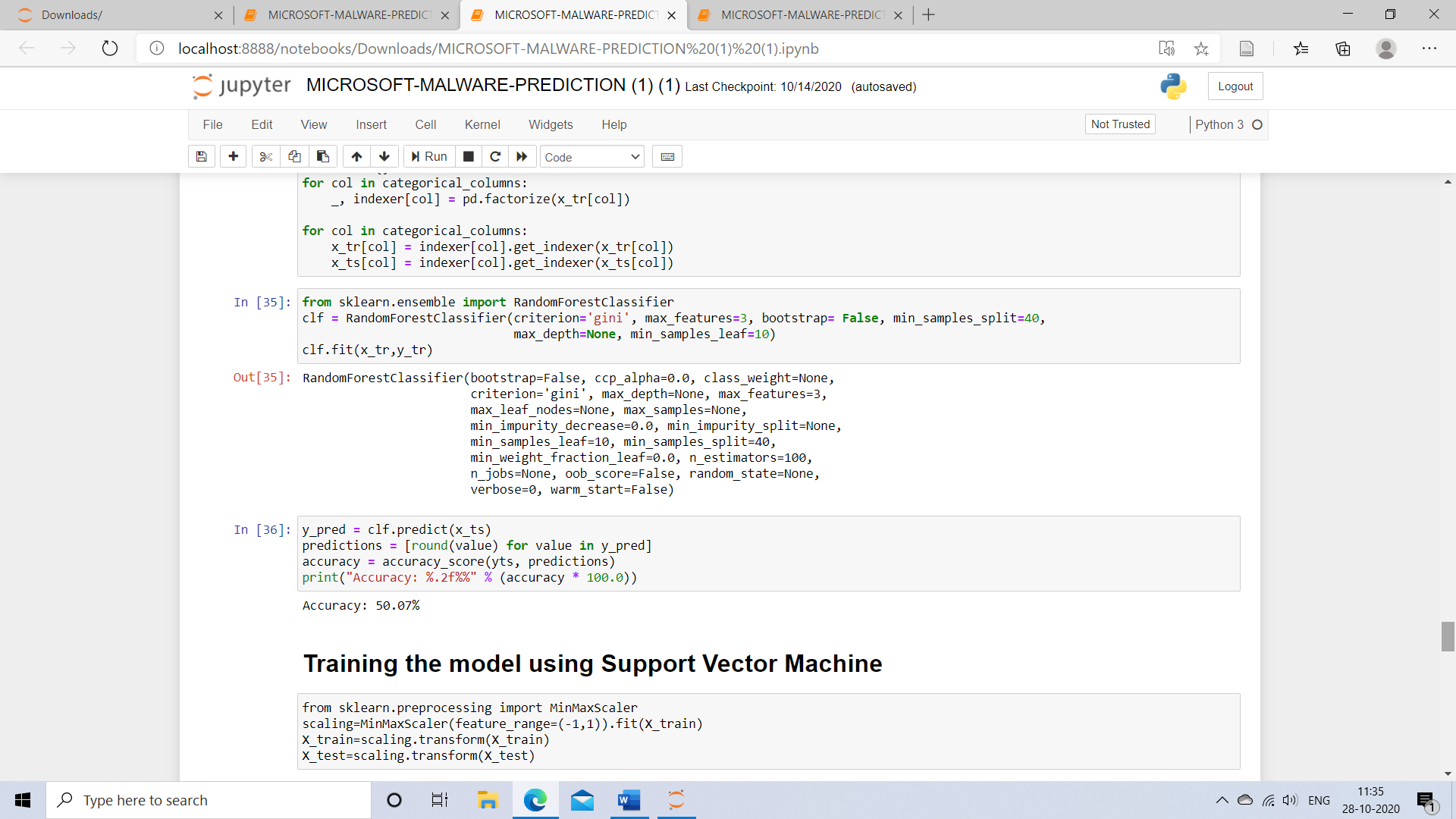


Figure 4‑13: RFC Accuracy Score

1. **XGBoost**:

The model after fitting the data shown an accuracy of about 65%.

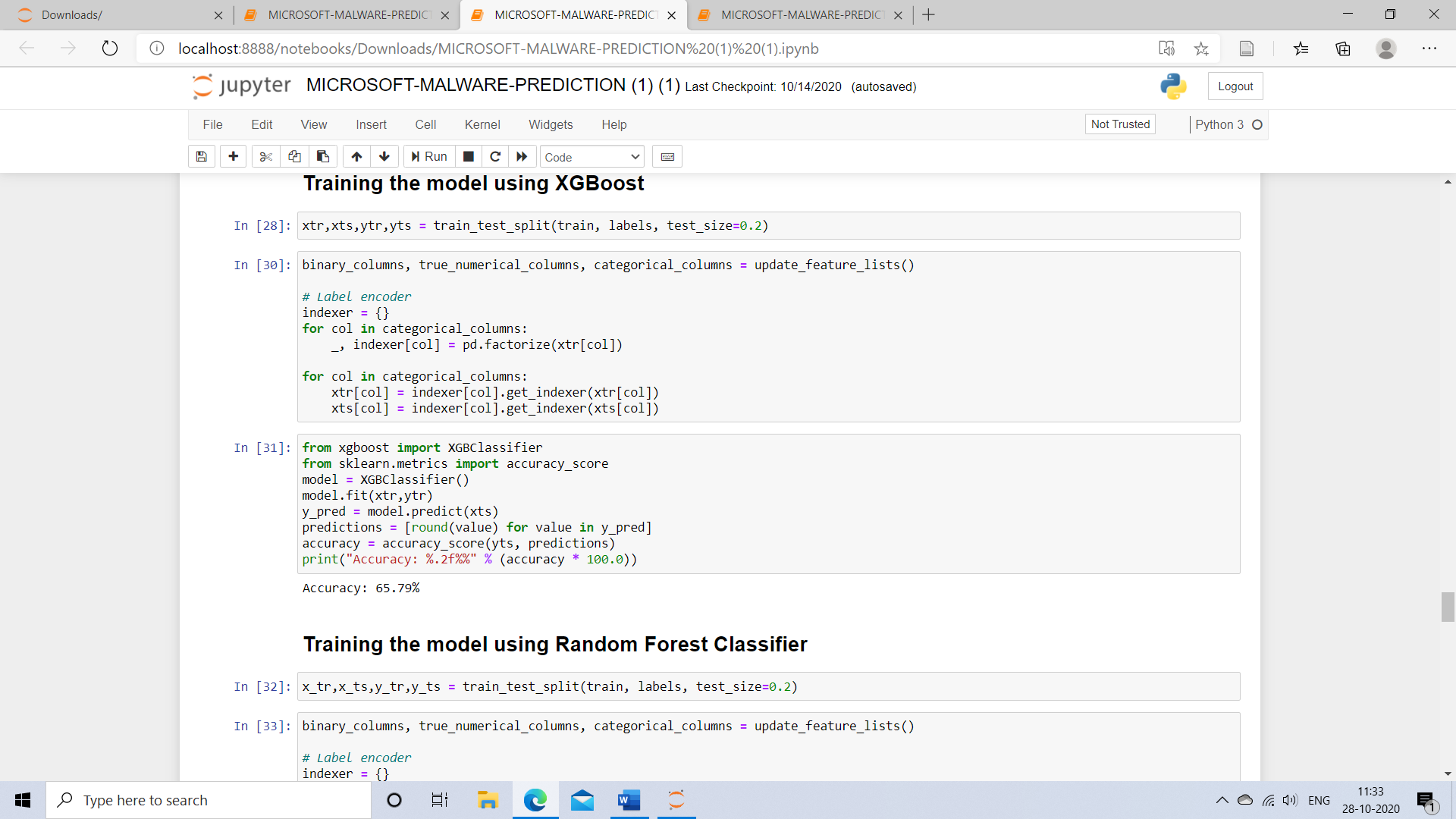


Figure 4‑14: XGB Accuracy Score

1. **Sequential Neural Network:**

The layers are added sequentially with activation function as relu, loss as binary entropy, optimizer as adam. Forward Propagation and Backward propagation occur for every iteration and to minimize the loss adjusts the weights accordingly.

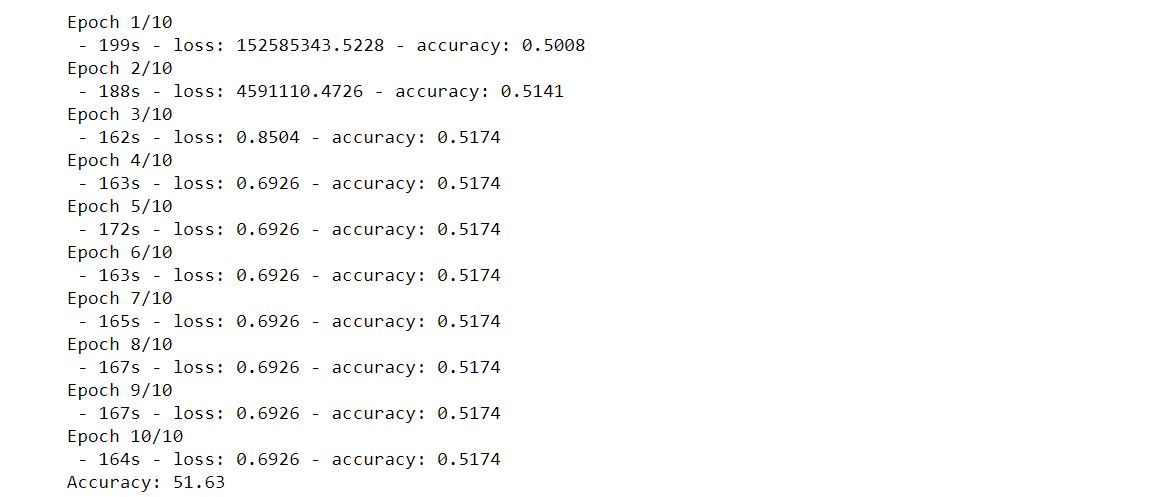


Figure 4‑15: SNN Accuracy Score

The accuracy after 100 iterations in training the data is about 51%.

## Validation or Verification

The validation of the model is performed, and the results are interpreted to know the misclassification count and accurate prediction count. The confusion matrix is used to represent true positives, true negatives, false positives, and false negatives.

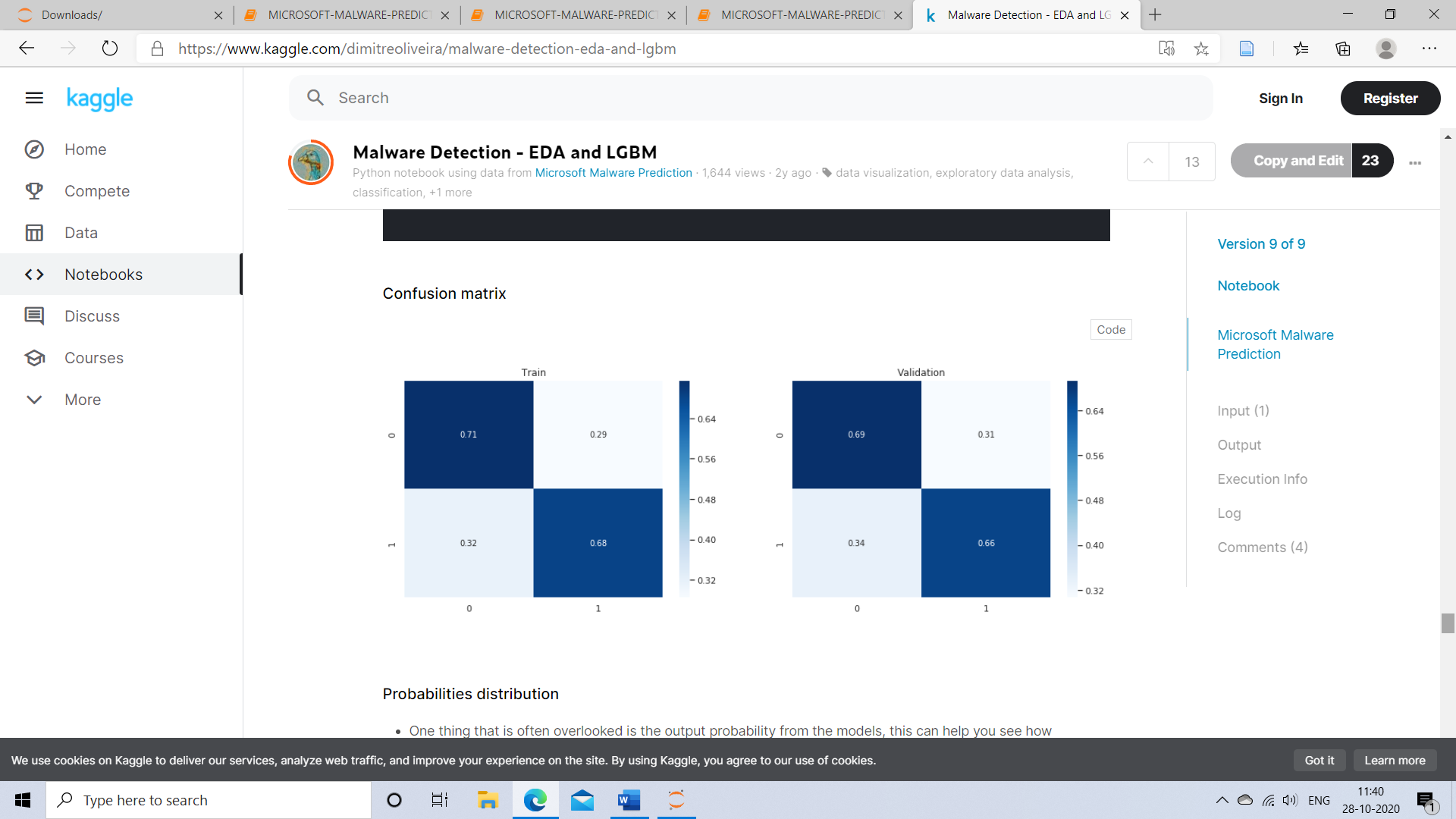


Figure 4‑16: Confusion Matrix for LGBM

**Model Metrics**:

The metrics are analysed to know the correctness of the LightGBM Model thereby, evaluated precision, recall, and f1score.

Table 4‑2: Model Metrics for Training

|  |  |  |  |
| --- | --- | --- | --- |
| Training | Precision | Recall | F1score |
| HasDetections = 0 | 0.68 | 0.71 | 0.70 |
| HasDetections = 1 | 0.72 | 0.68 | 0.70 |

Table 4‑3: Model Metrics for Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Validation | Precision | Recall | F1score |
| HasDetections = 0 | 0.66 | 0.69 | 0.68 |
| HasDetections = 1 | 0.70 | 0.66 | 0.68 |

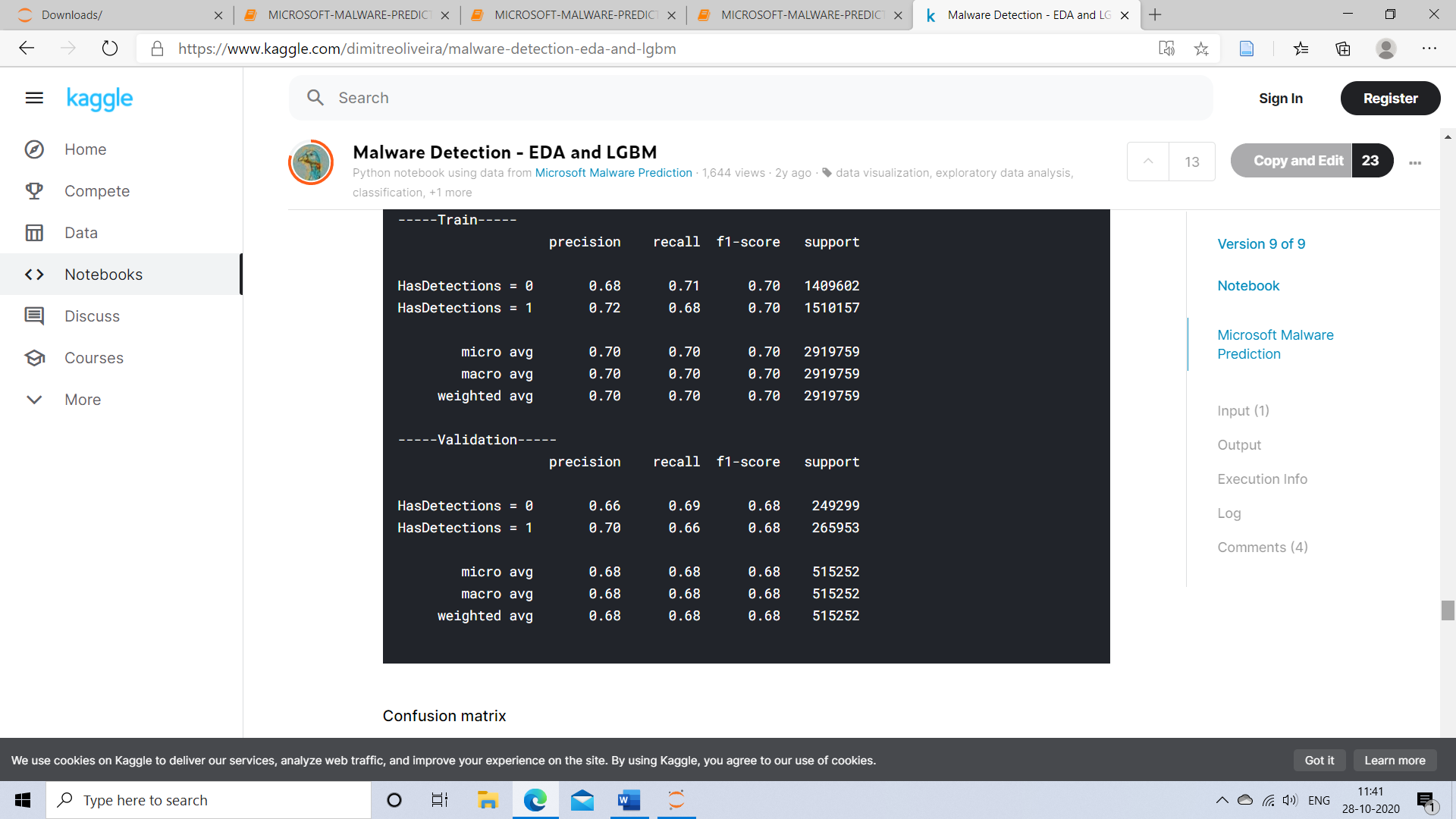


Figure 4‑17: Model Metrics

# Chapter: Conclusion and future work

## Conclusion:

We can summarize that Light Gradient Boosting will produce better results and generates more accurate results than the Random Forest, Decision Tree Classifier, XG Boost and Sequential Neural Networks. The Light GBM will produce the 77.4% accuracy that is 11.61% more accurate than XGBoost,18.6% more accurate than Decision Tree Classifier, 25.7% more accurate than Sequential Neural Network, and 27.4% more accurate than Random Forest Classifier.

## Future Works:

Try Dimensionality reduction on high cardinality features, techniques like Principal Component Analysis, t-distributed stochastic neighbor embedding or auto encoders may help. Other approaches to parameter tunning like Bayesian optimization may help as well. Impute the data using multivariate imputation by chained equations. Use other models which can use categorical data without encoding.

# Reference

[1] “Microsoft Malware Prediction.” https://kaggle.com/c/microsoft-malware-prediction (accessed Sep. 17, 2020).

[2] P. H. Swain and H. Hauska, “The decision tree classifier: Design and potential,” *IEEE Transactions on Geoscience Electronics*, vol. 15, no. 3, pp. 142–147, Jul. 1977, doi: 10.1109/TGE.1977.6498972.

[3] V. Y. Kulkarni and P. K. Sinha, “Pruning of Random Forest classifiers: A survey and future directions,” in *2012 International Conference on Data Science Engineering (ICDSE)*, Jul. 2012, pp. 64–68, doi: 10.1109/ICDSE.2012.6282329.

[4] G. Ke *et al.*, “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3146–3154.

[5] L. Denoyer and P. Gallinari, “Deep Sequential Neural Network,” *arXiv:1410.0510 [cs]*, Oct. 2014, Accessed: Sep. 17, 2020. [Online]. Available: http://arxiv.org/abs/1410.0510.

[6] P. Lemenkova, “Python libraries matplotlib, seaborn and pandas for visualization geospatial datasets generated by QGIS,” *Analele stiintifice ale Universitatii ”Alexandru Ioan Cuza” din Iasi - seria Geografie*, vol. 1, pp. 13–32, Sep. 2020, doi: 10.6084/m9.figshare.13010069.