



Google Developer Student Clubs  
Vellore Institute of Technology, Chennai

# HOUSE OF DEVELOPERS

TENSOR GUYS TEAM NO: 37

## PROBLEM STATEMENT:

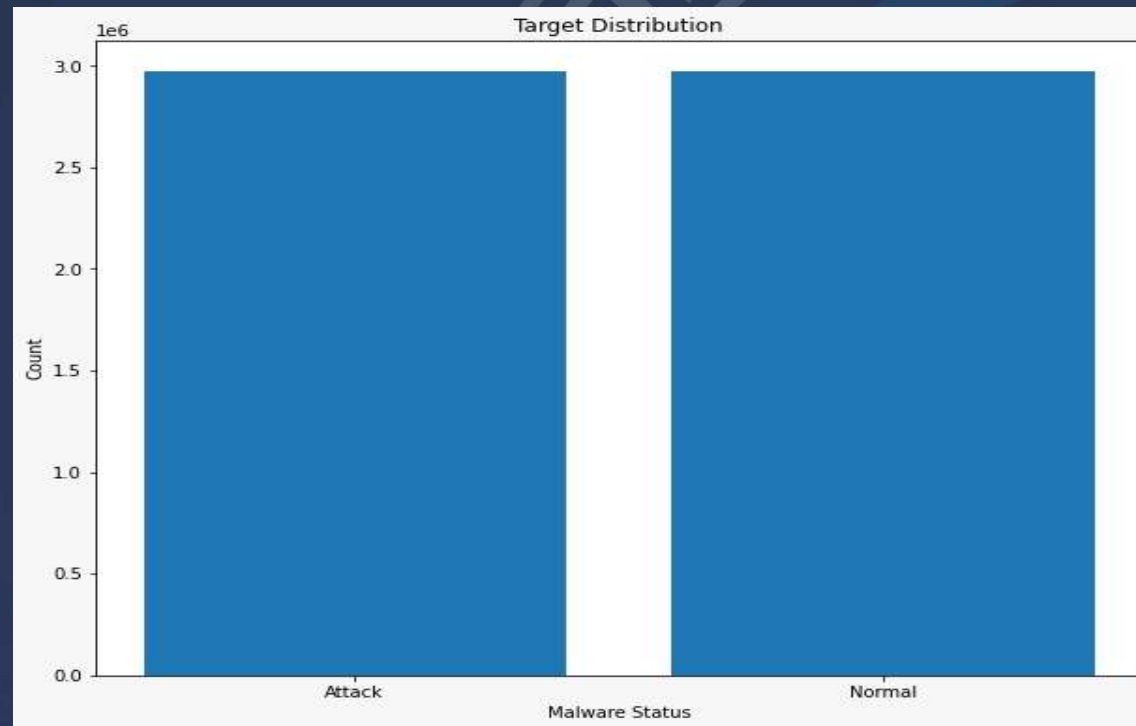
- DETECTION OF MALWARE INFECTION USING ETHICALLY COLLECTED DATA.
- A CLASSIFICATION PROBLEM BETWEEN MALWARE 'ATTACK' AND 'NO ATTACK'
- THE TARGET VARIABLE IS 'MALWARE\_STATUS'
- WHILE THE REMAINING 82 ARE CONSIDERED AS FEATURES.

## PROJECT:

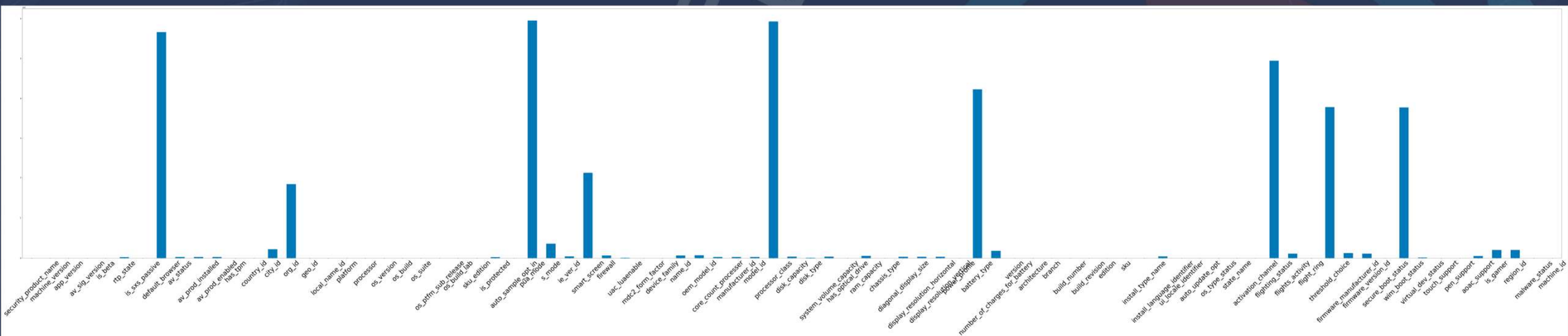
- ANALYSE, VISUALIZE AND PROCESS THE COLLECTED DATA
- TRAIN MACHINE LEARNING AND DEEP LEARNING MODELS ON THE COLLECTED DATA
- DETERMINE THE BEST MODELS USING VARIOUS PERFORMANCE METRICS.
- DEPLOY THE FINAL MODEL TO INFER ON REAL WORLD DATA.

# EXPLORATORY AND STATISTICAL DATA ANALYSIS:

## - CHECK FOR CLASS IMBALANCE:

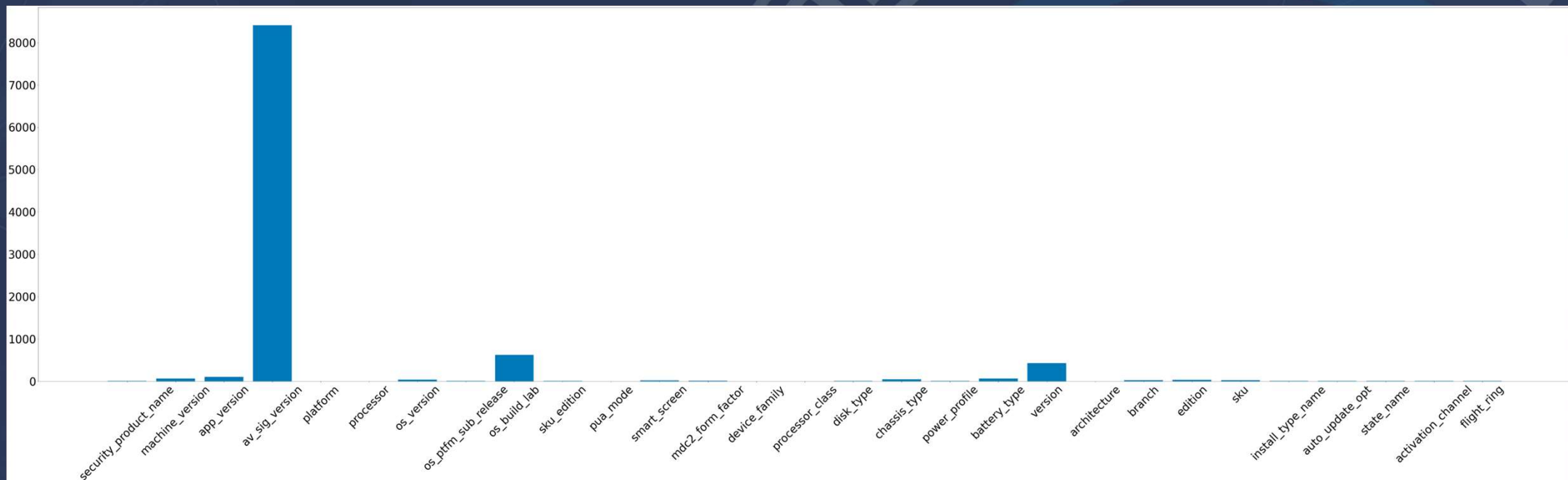


- NULL VALUES IN EVERY FEATURE COLUMN:
- EVERY ROW HAS A NULL VALUE
- EVERY COLUMN THAT HAS MORE THAT 50% OF NULL VALUES IS DROPPED
- SINCE THE DATA IS CATEGORICAL DATA THE EXISTENCE OF OUTLIER IS RARE  
HOWEVER ,TO REMOVE OUTLIER IF ANY, WE SHALL USE KNN TECHNIQUE,  
BUT AGAIN SINCE THE COMPUTATION POWER IS NOT THAT STRONG WE  
WILL GO WITH MODE.

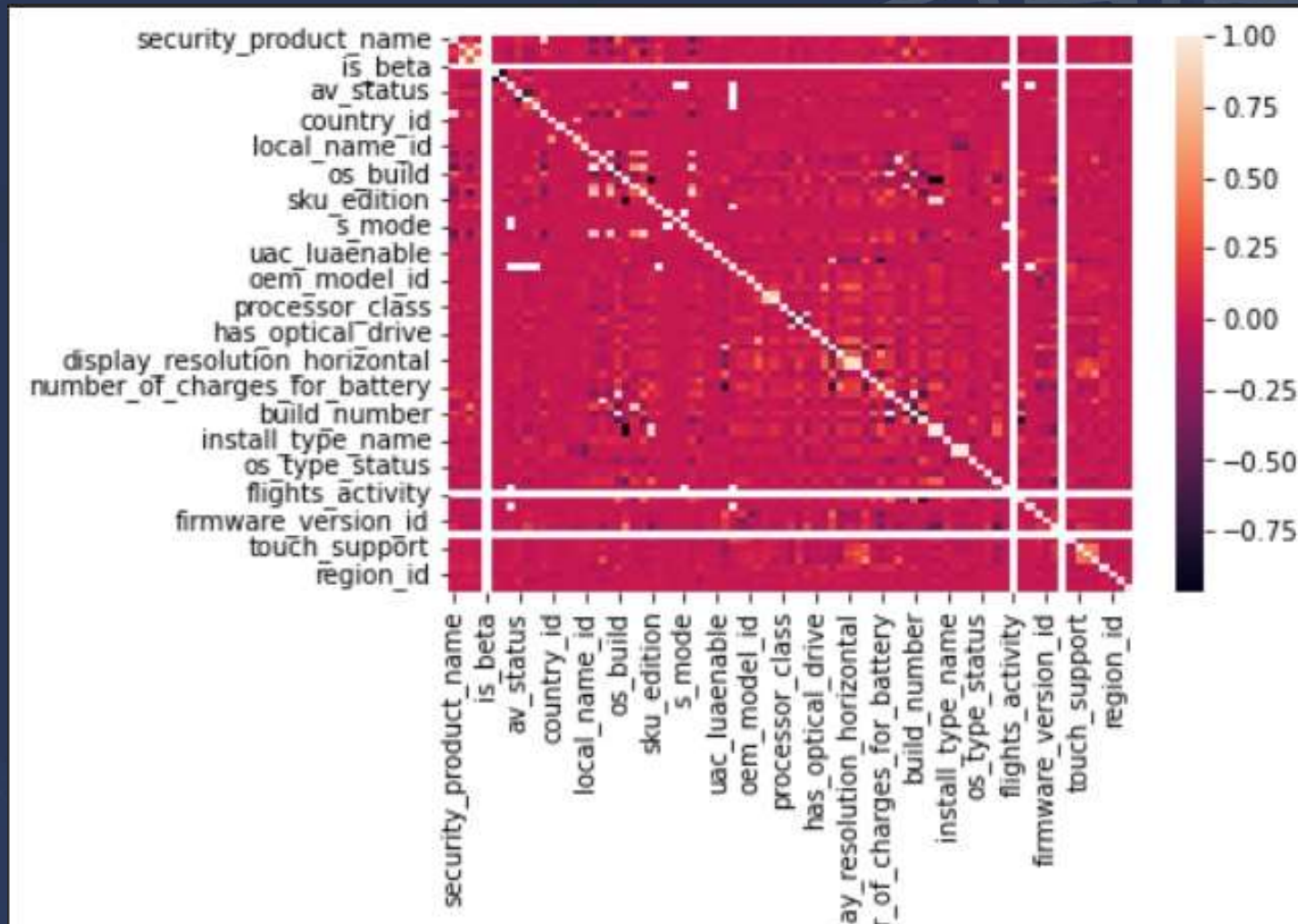




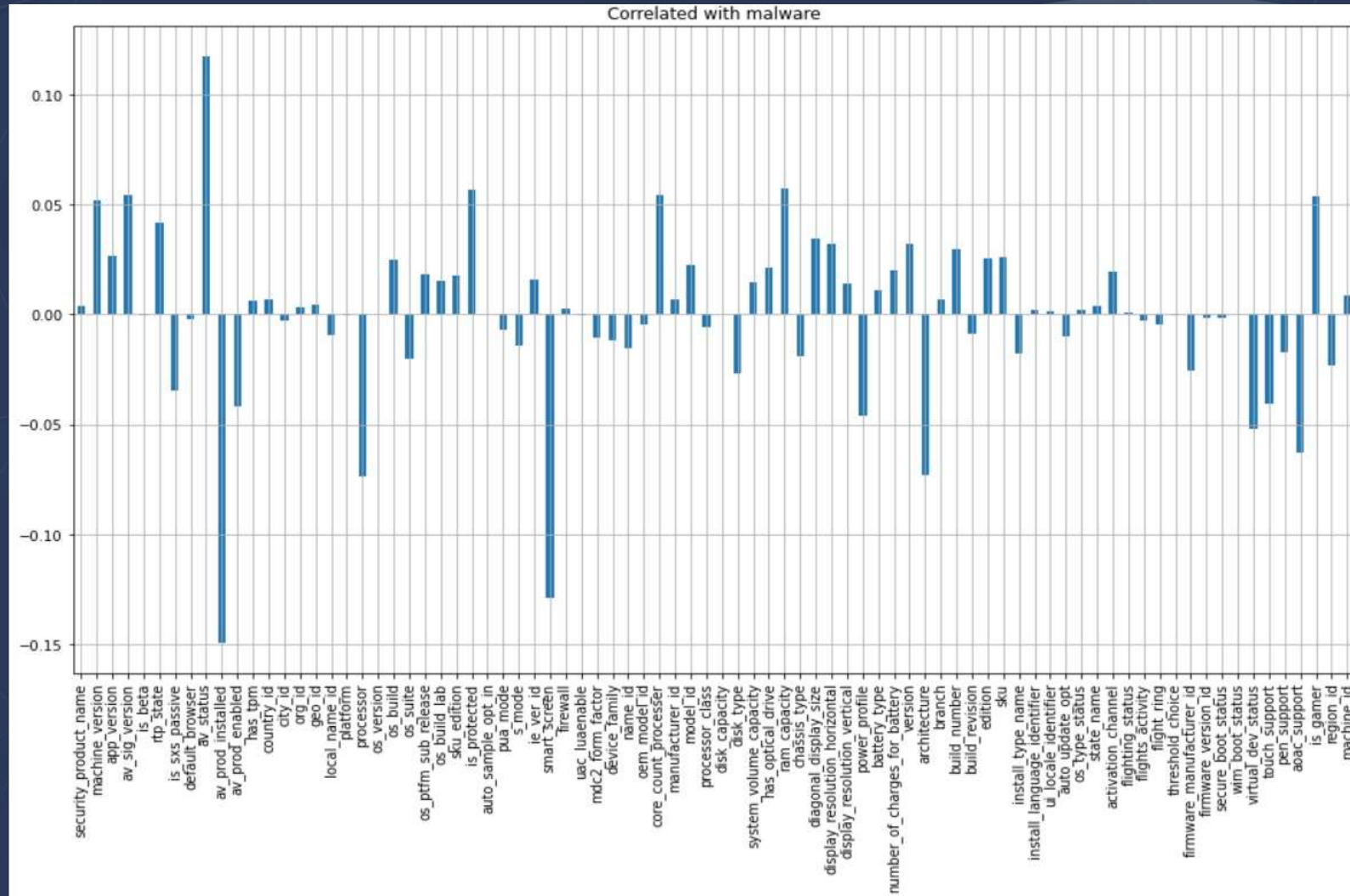
- UNDERSTANDING THE FEATURE VARIABLES:
  - The no.of unique values for each feature is extremely low compared to the sample size of 5950285 which suggests that the all features are categorical in nature
- Plot no.of unique values for each feature.



## -CORRELATION HEATMAP B/W ALL FEATURE VARIABLES:



## - CORRELATION PLOT BETWEEN TARGET AND FEATURES:





LITERATURE SURVEY:

POSSIBLE SOLUTIONS:

METHOD 1: CONDITIONAL STATEMENTS

-HARDCODE CONDITIONAL STATEMENTS TO LOOK FOR PATTERNS IN DATA TO DETECT THE MALWARE INFECTIONS.

CONS: CANNOT RECOGNIZE ALL PATTERNS IN THE DATA BY HARDCODING

METHOD 2: SIGNATURE - DETECTION:

- UTILIZES A LARGE DATABASE OF MALWARE SIGNATURES AND DETECTS MALWARE BY MATCHING THE CURRENT SIGNATURE TO THE SIGNATURES IN THE DATABASE:

CONS: REQUIRES LARGE AMOUNTS OF STORAGE

- PROJECT WORKFLOW:
- Proposed ML Models: logisticRegression, DecisionTrees,
- Proposed DL Models: ANNs and CNNs 1D





# DATA PREPROCESSING

# Data Pre-processing

1. To pre-process the data with already existing millions of Null values, firstly we reduced the features with more than 50% null values.

## Checking and Dropping the Columns with more than 50% of the Null Values

```
null_percent=df.isnull().sum()/df.shape[0]*100
null_percent
col_drop=null_percent>null_percent>50].keys()
col_drop
```

4]

Python

```
Index(['default_browser', 'pua_mode', 'processor_class', 'battery_type',
      'flighting_status', 'threshold_choice', 'wim_boot_status'],
      dtype='object')
```

```
df.drop(col_drop, axis=1, inplace=True)
```

5]

Python



2. We converted the columns with 'Object' data types to the Label Encoded Values, by using sklearn Label Encoder.

After Dropping we need to Encode the String Values

```
[7]: object_col_list=[]
      for i in df.columns:
      ... if df[i].dtype==object:
      ...     object_col_list.append(i)
      len(object_col_list)

.. 27

[8]: l_encoder = preprocessing.LabelEncoder()
      for element in object_col_list:
      ... df[element] = l_encoder.fit_transform(df[element])
```

### 3. Whether the data is Categorical or Numerical ?

To find this out we tried to check the unique values of each column, we figured out that the max unique values are 461196 which far very less from the number of rows i.e. more than 59 lacs, this implies that all the columns are Categorical.

```
print(num_unique[34])
print(num_unique[40])
```

[23]

...	141844
	461196

Now, comes the portion of imputation :

We tried to impute the Data by various Algorithms like KNN and Random Forest, since the dataset is so Huge normal methods are not suitable but since our PCs are not that powerful we are going with normal imputation technique, here Mode.

```
df.set_index(['machine_id'], inplace=True)

for i in df.columns:
    if df[i].isna().sum() > 0:
        df[i].fillna(df[i].mode()[0], inplace=True)
df.head()
```

machine_id	security_product_name	machine_version	app_version	av_sig_version	is_beta	rtp_state	is_sxs_passive	av_status	av_prod_installed	av_prod_enabled	...	secure_boot_status	virtua
4300000	4	65	77	8109	0	7.0	0	53447.0	1.0	1.0	...	0	
4300001	4	64	51	7767	0	7.0	0	53447.0	1.0	1.0	...	1	
4300002	4	65	57	7919	0	7.0	0	53447.0	1.0	1.0	...	1	
4300003	4	65	57	7985	0	7.0	0	53447.0	1.0	1.0	...	0	
4300004	4	64	51	7720	0	7.0	0	53447.0	1.0	1.0	...	1	

5 rows × 77 columns



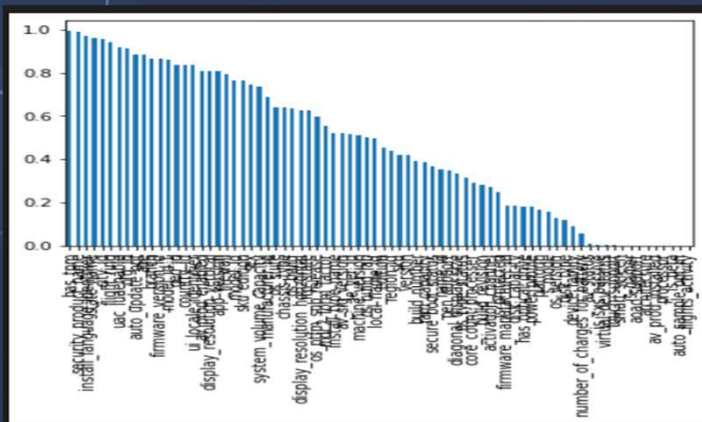
# FEATURE REDUCATION



## Feature Reduction :

- For the feature reduction we used chi-square test
- We used this here because the data needs to be reduced with important features
- The columns with high p-values are removed.
- There we have reduced our features to 40, after removing the high p-value columns.

Look at the columns to be dropped :



```
print(final_drop)

Index(['security_product_name', 'install_language_identifier', 'state_name',
      'city_id', 'flight_ring', 'uac_luaenable', 'org_id', 'auto_update_opt',
      'age', 'branch', 'firmware_version_id', 'model_id.1', 'geo_id',
      'country_id', 'ui_locale_identifier', 'av_prod_enabled',
      'display_resolution_vertical', 'app_version', 'firewall', 'model_id',
      'sku_edition', 'sex', 'system_volume_capacity', 'manufacturer_id',
      'os_suite', 'chassis_type', 'os_build', 'display_resolution_horizontal',
      'rtp_state', 'os_ptfm_sub_release', 'mdc2_form_factor',
      'install_type_name', 'av_sig_version', 'ie_ver_id', 'machine_version',
      'os_build_lab'],
      dtype='object')
```



# MODEL TRAINING

## For Selecting the appropriate model

For the model selection we used Lazy Classifiers, which do train 29 models to check which of the models work better.

From there we concluded that the Boosting ML models were giving a local test accuracy of about 65%, the tree models whereas gave about 61%, the SVM was not that efficient.

We tried many neural network architectures :

Those with Conv1D gave about 50% of the local test accuracy


The Dense neural ones gave it about 67% of the local test accuracy

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
XGBClassifier	0.61	0.61	0.61	0.61	0.84
AdaBoostClassifier	0.61	0.61	0.61	0.61	0.67
RandomForestClassifier	0.60	0.60	0.60	0.60	1.24
LGBMClassifier	0.60	0.60	0.60	0.60	0.47
RidgeClassifier	0.60	0.60	0.60	0.60	0.04
LogisticRegression	0.60	0.60	0.60	0.60	0.13
LinearDiscriminantAnalysis	0.60	0.60	0.60	0.59	0.11
LinearSVC	0.59	0.60	0.60	0.59	0.99
CalibratedClassifierCV	0.59	0.60	0.60	0.59	3.63
RidgeClassifierCV	0.59	0.60	0.60	0.59	0.06
ExtraTreesClassifier	0.59	0.59	0.59	0.59	0.90
NearestCentroid	0.59	0.59	0.59	0.59	0.04
SVC	0.59	0.59	0.59	0.59	3.59
BaggingClassifier	0.58	0.58	0.58	0.58	0.85
BernoulliNB	0.57	0.57	0.57	0.57	0.04
SGDClassifier	0.56	0.56	0.56	0.56	0.19
NuSVC	0.56	0.56	0.56	0.56	4.01
ExtraTreeClassifier	0.55	0.55	0.55	0.55	0.03
KNeighborsClassifier	0.54	0.54	0.54	0.54	0.58
DecisionTreeClassifier	0.54	0.54	0.54	0.54	0.16
Perceptron	0.52	0.52	0.52	0.52	0.04
PassiveAggressiveClassifier	0.52	0.52	0.52	0.52	0.04
QuadraticDiscriminantAnalysis	0.51	0.52	0.52	0.47	0.06
LabelSpreading	0.52	0.52	0.52	0.52	1.67
LabelPropagation	0.52	0.52	0.52	0.52	1.44
GaussianNB	0.51	0.51	0.51	0.38	0.03
DummyClassifier	0.49	0.50	0.50	0.33	0.03

## These are our results of the lazy classifier

(Rest of the model results are available in the notebooks themselves)





As a Conclusion for the model we decided to  
work with Light Gradient Bosst Model  
(lightGBM)



# HYPER PARAMETER TUNING :

For hyperparameter tuning we used a manual way of tackling with them :

- The results for lightGBM were :

max\_depth=8, num\_leaves=180, min\_child\_samples=200

```
model = lgb.LGBMClassifier(max_depth=8, num_leaves=180, min_child_samples=200)
model.fit(xtrain, y_train)
acc = model.score(xtest, y_test)
```

*Below given are some results of Hyperparameter Tuning :*

Num leaves : 70

min\_child samples : 100

Accuracy : 0.6536616

Num leaves : 70

min\_child samples : 200

Accuracy : 0.6540208

Num leaves : 70

min\_child samples : 300

Accuracy : 0.6536232

Num leaves : 70

min\_child samples : 400

Accuracy : 0.6534576



*After all this processing we got the following results on the Kaggle :*

✓	<b>submission_m3.csv</b> Complete · Ashish Udainiya · 1s ago	<b>0.63199</b>	<input type="checkbox"/>
✓	<b>Submission.csv</b> Complete · Ashish Udainiya · 6h ago	<b>0.63564</b>	<input type="checkbox"/>
✓	<b>Submission.csv</b> Complete · Ashish Udainiya · 11h ago	<b>0.63267</b>	<input type="checkbox"/>
✓	<b>Submission.csv</b> Complete · Ashish Udainiya · 12h ago	<b>0.63932</b>	<input type="checkbox"/>
✓	<b>submission.csv</b> Complete · Jeshur Joshua · 14h ago	<b>0.58451</b>	<input type="checkbox"/>
⚠	<b>submission.csv</b>		



THANK YOU