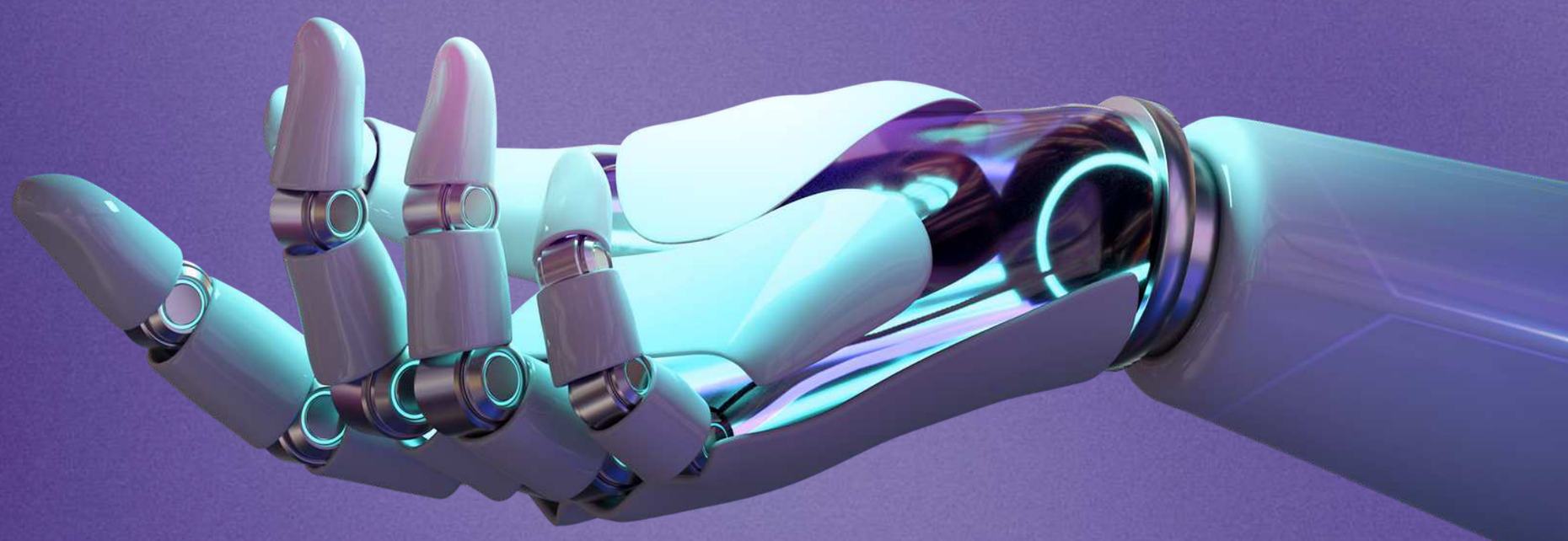
FINAL PROJECT
MACHINE LEARNING

IS A FILE MALICIOUS?



BY

DANA BRAYNIN (207106865)
RAZ GRAIDER (315150953)

Report

Part 1- Exploring the data

In this part, through the exploration, we want to understand the general nature of the data and for this purpose we will use plots.

Duplicates

We decided to check if there is any duplicate data because when duplicate data is present in the training set, the model might overfit to these duplicated instances. We received data with 60,000 samples and after removing duplicates we were left with 59,947 samples.

Glimpse and information about the data

We received 23 features. Few of them appear as float, but they represent classification to 2 groups and do not have a numeric meaning ('has_debug', 'has_relocations', 'has_resources', 'has_signature' and 'has_tls'). There are a few columns with missing values (NaN). Features 'A', 'B' and 'C' are anonymous features. Feature 'B' has very little variance compared to feature 'A' that has very wide variance. Therefore, feature 'A' has negative values, but feature 'B' does not. The features 'sha256', 'file_type_trid' and 'C' are categorical features. In feature 'sha256' each file has its own name. 'file_type_trid' has 89 categories and 12 of them appear only once. In 'C' feature, there are only 7 categories.

Visualizations

Feature distribution: This step will help us to examine if each feature distribution is close to normal distribution (1.1, 1.2). From the plots we learned a few things: The binary features are also shown like this and are distributed around "0" and "1". There are some features that have very extreme examples that disturb the density plot ('file_size', 'vsize', 'imports', 'exports', 'symbols', 'numstrings', 'avlength'). Feature 'A' is the only one that is close to the gaussian distribution, therefore we will assume that this feature is normally distributed. We also showed it using 'boxplot' (1.3) and it confirmed this claim. It seems that there are a lot of outliers, but they seem relatively equal between the two kinds of the label.

There are 60 categories in 'file_type_trid' that appear less than 100 times (1.4). We decided to remove these catagories because it is negligble in relation to the number of the given samples. We saw that the category "VR" in 'C' feature (1.5) appears significantly less than the others. 'C' is unknown, so we cannot predict if this category is important.

Feature correlation: There are not many strong correlations between the numeric features and the label (1.6), but there are significant correlations. The correlation between 'size' and 'numstrings' is 0.9 (1.7). It makes sense that the more strings in the file, the bigger the file. There are radical outliers. The correlation between 'size' and 'MZ' is 0.72 (1.8). We learned that a large file is more likely to tend to be malicious. We can see that there are a few outliers that may disrupt this correlation. In contrust of what we assumed; it seems that most of the malicious files are grouped in an area where the size is small. The correlation between 'numstrings' and 'MZ' is 0.65 (1.9). We assume that the more strings in a file, the bigger the possibility that "MZ" will appear. The plot is like the correlation plot between "size" and "MZ", due to the strong correlation between "size" and "numstrings". The last strong correlation is 0.64 between "avlength" and "printables" (1.10). We saw that the correlations with the label are not very high, which can be a problem for the model. A solution is to unite between highly correlated features.

Outliers: All the numeric features have outliers (1.11). This can be an issue because the model can learn from them and its gerenalizaion will not be as good. Later, we will deal with these outliers to improve the correlation with the label.
Missing values: For most of the features, there are thousands of missing values. The 3 anonymous features have a high percentage of missing values, especially "A" and "B" (1.12). This strengthens the uncertainty regarding these features. We also learned that 'file_type_trid', 'file_type_prob_trid' and 'size' do not have missing values.
Correlation with the label: We saw that there is no feature with high correlation with the label (1.6) . A reason for that may be that as we saw, most of the features have missing values.

Part 2- Pre-Processing

We split up the train to train and validation. The common choice is to split to 80% train and 20% validation. We took into considiration that we have many samples (59,947), therefore we will give the validation 25% allowing for a more robust evaluation of the model.
Dealing with outliers: We used the Winsorization technique that addresses outliers in non-normal features (we assumed that only 'A' is normally distributed) by capping or truncating extreme values at a predefined threshold. Winsorization replaces outliers with less extreme values. This approach retains the information provided by the outliers while reducing their impact on the analysis. We decided that the threshold will be 5%, meaning that all the samples in each feature that lie at the 5% (top and bottom) will be replaced by maximum and minimum values within the specified limits. After this, we noticed (2.1, 2.2) that for most of the features, the distibution of the samples has drasticlly changed for the better.
Are the features normally distributed? The only feature that is close to normal distribution is feature 'A'. We will normalize the other features because deviations from normality may affect some model's performance and the validity of the results. Normally distributed features tend to have well-defined and intuitive interpretations.
Dealing with missing values: After dealing with outliers, we have fewer missing values. There are no samples with over 50% of missing values or features with many missing values.
Filling missing values in numeric binary features- We used the most frequent approach that can help maintain the distribution of the data by imputing missing values with the dominant category. By comparing the distribution of each binary feature before and after this step, we noticed that the dominante category did not change (2.3), so the distribution maintained.
Filling missing values in numeric non-binary features- After dealing with outliers, we were left with 4 numeric non-binary features with missing values ('paths', 'MZ', 'A', 'B'). Except for 'B', the rest contain outliers and are not roughly symmetric (2.2). When a numeric feature has a roughly symmetric distribution without outliers, using the mean for imputation may be appropriate, so we used it for 'B'. For the rest, we used the median technique.
We saw that overall, the density of each numeric feature did not change drastically (2.4).
Filling missing values in categorical features- We filled in the missing values of 'C' with the Forward-fill method that carries forward the last observed category to fill in the missing values. It preserves the distribution of the observed values in the column and does not introduce any new categories. This method is straightforward to implement and computationally efficient, particularly for large datasets like ours. The dominant category and the order of the frequent

categories remained the same (2.5).

Dealing with categorical features:

'file_type_trid'- There are 60 categories that appear less then 100 times. We removed all the samples that have these categories in the feature before encoding because if a category occurs very rarely, it may not provide enough information for the model. It helps prevent overfitting and improves model generalization. We used Binary Encoding that encodes each category as an integer, converts the integer to binary code, and creates binary features based on the binary representation. The feature has 28 categories, and we would like to avoid a situtaion of large dimesionality. Binary encoding is useful for categorical features with many unique categories as it can reduce the dimensionality.

'C'- We used frequency encoding method that encodes categorical variables based on the frequency of each one. It replaces each category with the percentage of times it appears. This method does not add any new features. We added 5 new features in the encoding of 'file_type_trid', so we wanted a method that does not add any new features.

How do the categorical features distribute? Neither 'file_type_trid' features (binary) nor 'C_freq_encode' (based on certain frequencies) distribute normally (2.6, 2.7, 2.8). There are no outliers in 'C_freq_encode' (2.9). No new strong correlation was added, and the new features do not have a strong correlation with the label (2.10). The strong correlations we had were reduced and there are no correlations that are above 0.53.

Dimension reduction:

Is the dimensionality of the train data is too large? If the number of features is significantly larger than the number of observations, it indicates a high-dimensional dataset. There are 26 features and 59,058 observations, so we do not have high-dimensional train data. High-dimensional train data may create a problem because it can increase the model's variance, overfitting, difficulties in understanding the model, irrelevant features and the model may require computational resources (extended explanation in part 2 "Dimension reduction").

PCA- We used the cross-validation technique to find the best number for 'n_components'. We wanted to explain at least 99% of train data. We were left with the same number of features (26). This does not help us to reduce dimensionality, so we will not use PCA.

Feature selection- We considered backward/forward selection. Both gave the same features, test and train MSE. We decided to use backward selection and were left with 20 features, while the deleted ones have low correlation with the label (2.10). This can reduce overfitting and improve computational efficiency, but it can lead to potential loss of information.

Feature engineering:

'has_reliability'- A file that has '0' in 'has_debug' and 'has_signature' will probably be a malicious file. This feature will be '1' only if a file has a debug section and a signature.

'numstrings_percentage'- The feature will be the percentage of 'numstrings' in 'printables'. (Extended explanation for both in part 2 "Feature engineering").

After performing the models, we noticed that when applying the two feature engineering the validation AUC is lower than when not applying them. We decided not to use this step.

Data transformation: We transformed the data using feature-wise transformations because we have a large variety of feature types.

Min-Max scaling- We tried this method because it preserves the relative relationships between

data points while ensuring that the scaled values fall within the desired range.
Standardizaion (Z-score normalization)- This normalization process is useful when working with features that have different scales or units like in our project. After performing the models, we decided to use min-max scaling because it gave a better validation AUC.

Part 3- Modeling

We used 'GridSearchCV' to find the best values for some of the hyper-parameters (reasoning for the values we checked in each model and their effect on the variance and bias are in part 3 in the notebook). The rest are default.
Logistic Regression (3.1)**-** The hyper-parameters we checked, and their values are: 'C'=100, 'solver'=sag, 'max_iter'=200. The train AUC is 0.812 and the validation AUC is 0.805. The gap between the two is relatively small, something that we want to achieve.
Naïve Bayes classifier (3.2)**-** We decided to leave the two hyper-parameters as their default value (reasoning in the notebook). The train AUC is 0.765 and the validation AUC is 0.760. Here too the gap is relatively small, but the values are lower than in the previous model.
Random Forest (3.3)**-** The hyper-parameters we checked, and their values are: 'n_estimators'=500, 'max_depth'=None, 'max_features'=log2. The train AUC is 0.999 and the validation AUC is 0.959 (the model is random; this AUC might slightly change in each run). Although the AUCs are high, the train AUC is very close to 1 so there is a risk of overfitting.
Multi-Layer Perceptron (ANN) (3.4)**-** The hyper-parameters we checked and their values are: 'alpha'=0.0001, 'hidden_layer_sizes'=(100,). The train AUC is 0.947 and the validation AUC is 0.928. The gap between the AUCs is relatively large, therefore there is a risk of overfitting.
XGBoost (Extreme Gradient Boosting) classifier (3.5)**-** This is a new model, the extended explanation of how it works is in the appendices (6.1). The hyper-parameters we checked, and their values are: 'n_estimators'=200, 'max_depth'=10, 'alpha'=0.1, 'learning_rate'=0.1, 'random_state'=42. The train AUC is 0.999 and the validation AUC is 0.962. This is the highest validation AUC we received but the train AUC is close to 1, risk of overfitting.
The model we would like to continue with for the rest of the project is XGBoost Classifier because it gave us the highest validation AUC.
Feature importance: We analyzed the 9 features that have the highest feature importance and explain together more than 70% of the model (3.6). The detailed explanation of each feature is in part 3 under the title "Feature importance".

Part 4- Model evaluation

Confusion matrix: From analyzing the confusion matrix (4.1), we learned that our model was correct significantly more than when it was incorrect (true positive and false negative are much higher than false positive and true negative), meaning that overall, our model predicted well the validation set. It is better to classify a not malicious file as a malicious file than the other way around and be exposed to danger. We noticed that true negative has more samples than false positive. This means that we have a relatively significant number of malicious files that were classified as not malicious. Although we see that our model predicted, for the most part, the samples correctly, this is its Achilles' heel.
K-fold cross validation: The common choice for k is 5\10. For large datasets like ours, it is acceptable to use a smaller number of folds, so we decided to perform 5-fold cross validation.

We preformed it on each model we used (4.2, 4.3, 4.4, 4.5, 4.6). From the plot that compares the mean AUC between all the models, after 5-fold cross validation (4.7), we noticed that Random Forest has a greater mean AUC in the 5-fold cross validation than XGBoost Classifier, that had the highest validation AUC in part 3. We still chose our main model to be XGBoost Classifier because it is recommended to choose the model with a higher validation AUC when we prioritize performance on unseen data, like in this project.

Is our model overfitted? We used a plot to show the train & validation AUC (4.8). We do not have a concrete answer to this question, but we will show both sides of the coin:

On the one hand, the validation AUC is close to the train AUC, and both are higher than 0.9. It can indicate that the model has learned patterns from the train data that can be applied to unseen data. This suggests that the model has good generalization capabilities. Also, it suggests that the model is not overfitting or underfitting. We noticed that both curves are far away from the diagonal line, indicating good model performance.

On the other hand, we suspect an overfitting in our model. We noticed that the training AUC is very close to 1. While it indicates good performance on the train data, it is important to be cautious about potential overfitting which occurs when the model becomes too specialized. Also, we noticed that the training ROC curve is close to the top-left corner. While this may seem impressive, it raises concerns about the model's ability to generalize to new data.

What did we do/can do to increase the generalizability of the model? After the pre processing, we were left with 44,960 samples. This large train data can help the model to generalize better. We put a lot of emphasis on pre processing to improve generalization by handling missing values, removing outliers, feature selection and standardizing the data. We increased the number of trees in our model so it can improve the model's generalization capabilities by reducing the impact of individual tree biases. XGBoost Classifier uses gradient boosting algorithms that optimize an objective function by iteratively adding new trees to the ensemble, which can help to capture diverse perspectives. We chose the values of the hyperparameters in the model by using grid search to enhance the generalization. We observed a large gap between train and validation performance, indicating overfitting. Regularly diagnosing will allow us to take appropriate actions to handle it.

Part 5- Prediction

We created a pipeline function that includes pre processing on all the train data and on the test data. It fits XGBoost classifier model on the train and predicts the probabilities of the samples in the test data to be malicious. We exported a csv file with the results to the samples in the test data. The classification of the samples in the test data is close to a uniform distribution (5.1), meaning that about half of the files were classified as malicious.

Part 6- New tools we used

During our project we used 3 new tools, Winsorization, Binary encoding and XGBoost classifier, and explained them during this report and in the notebook.

Summary

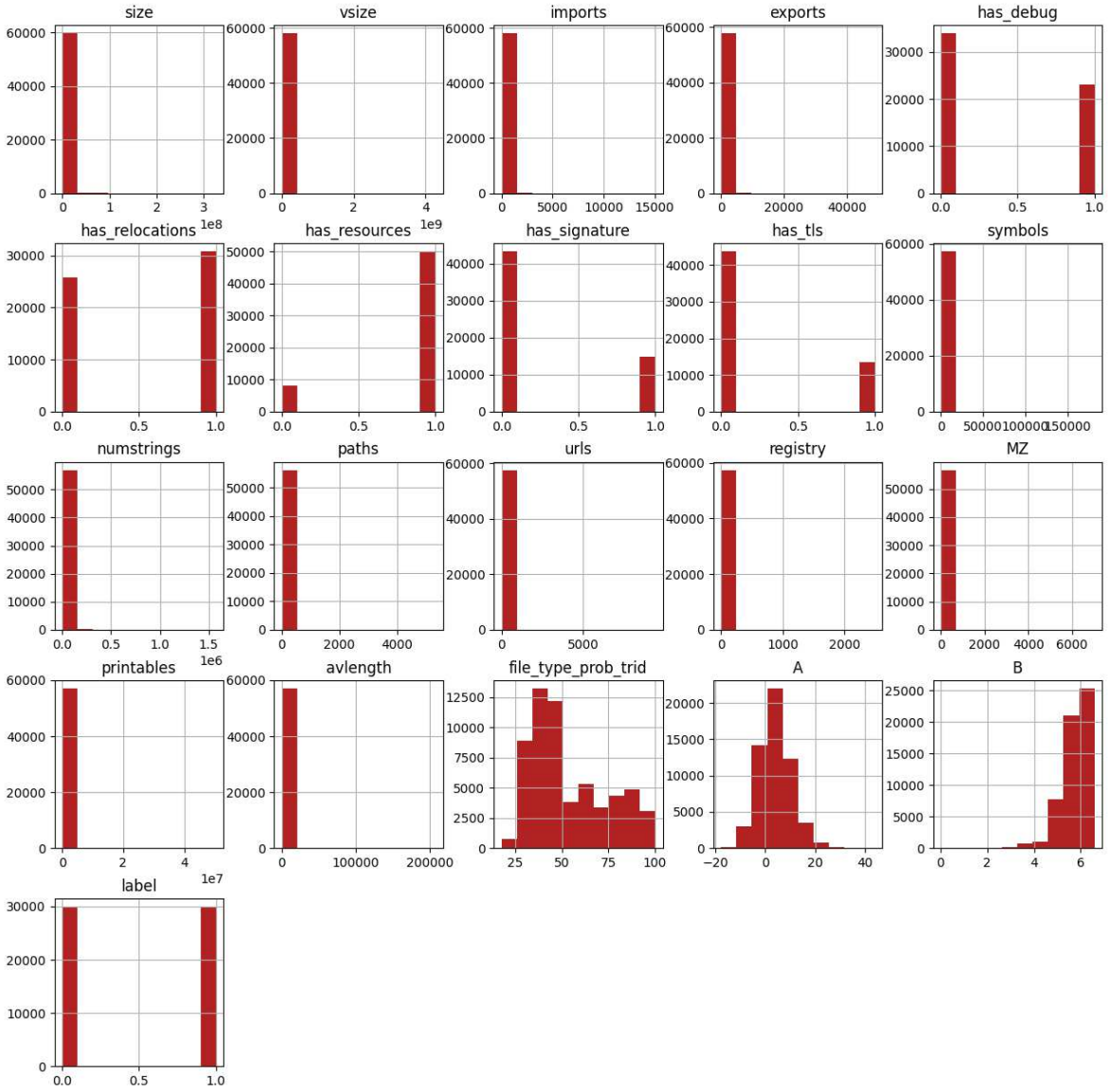
We developed a machine learning model that uses conclusions from the train data to predict the probability of each file in the test data being malicious.

Appendices

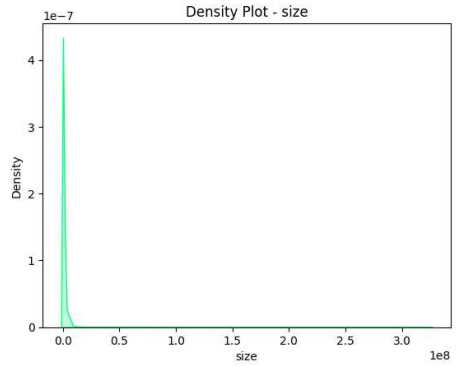
The responsibilities of each team member

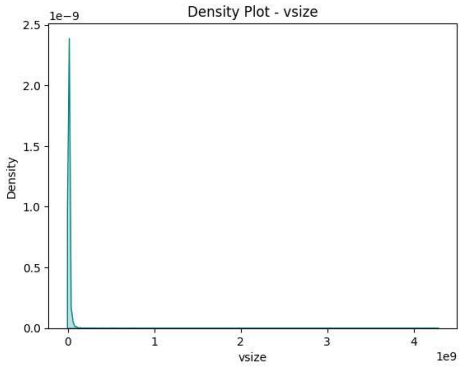
We saw fit to work on all parts of the project collaboratively and together through meetings or conversations on Google Meet.
The reason for this decision is that, in our opinion, this project consists of many small and important parts, each of which builds on the rest. In this situation, we feared that if we worked separately there would be things that would not be understood by both of us and that things would fall through the cracks. Additionally, working together allowed us to be in sync and share ideas with each other.
During our meetings, there was a clear division of roles, but we switched between the roles from time to time. Our meetings proceeded as follows: While one team member was responsible for most of the theoretical and practical information searching, the other was responsible for the programming side and building the notebook in Visual Studio Code. We decided to switch roles frequently so that each team member would fully experience both sides of the project.

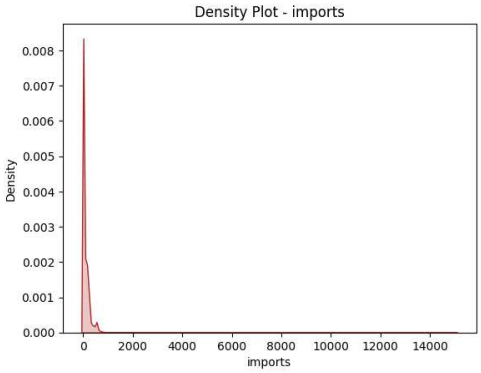
1.1

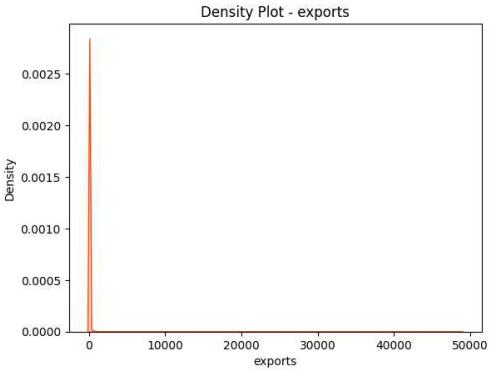

size
vsize
imports
exports
has_debug
has_relocations
1e8
has_resources
1e9
has_signature
has_tls
symbols
numstrings
paths
urls
registry
MZ
printables
1e6
avlength
file_type_prob_trid
A
B
label
1e7

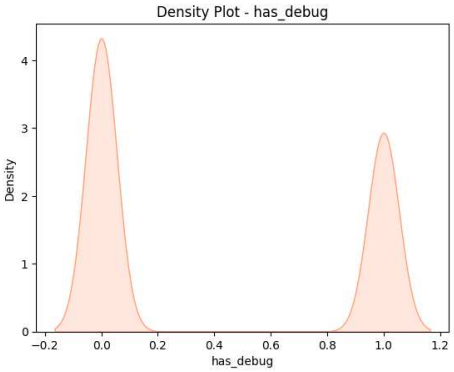
1.2

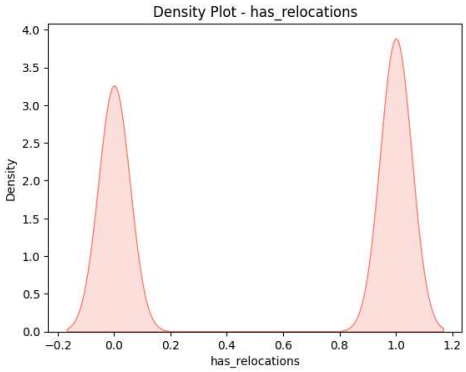

Density Plot - size
1e−7
Density
size
1e8

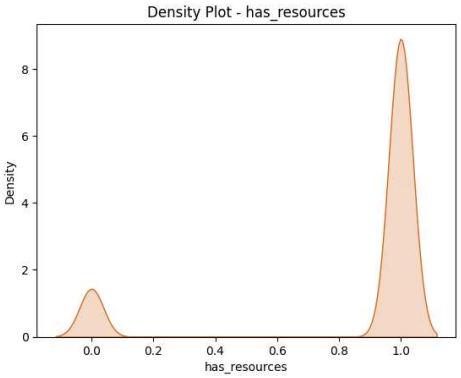

Density Plot - vsize
1e−9
Density
vsize
1e9

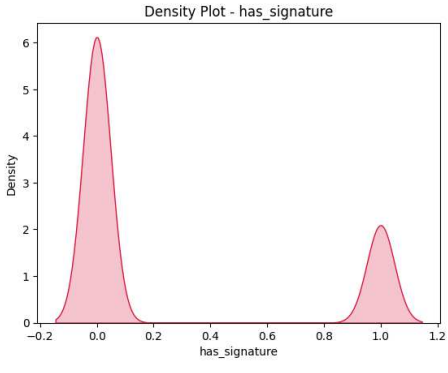

Density Plot - imports
Density
imports

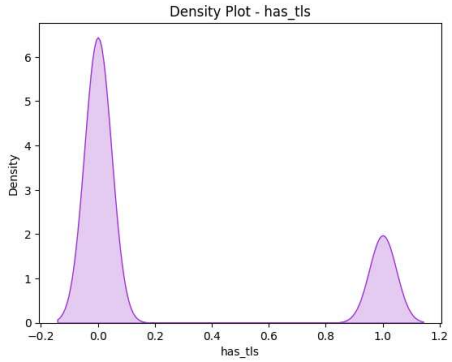

Density Plot - exports
Density
exports

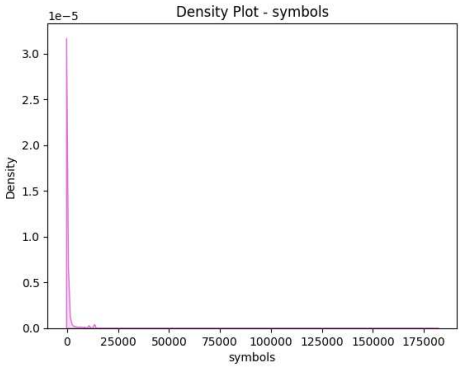

Density Plot - has_debug
Density
has_debug

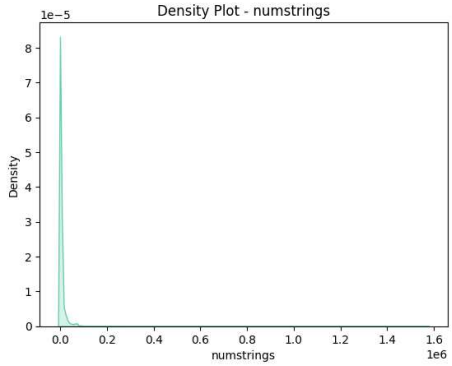

Density Plot - has_relocations
Density
has_relocations

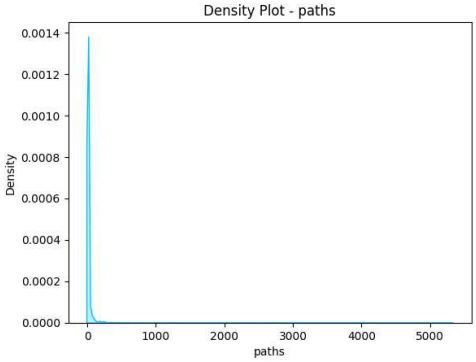

Density Plot - has_resources
Density
has_resources

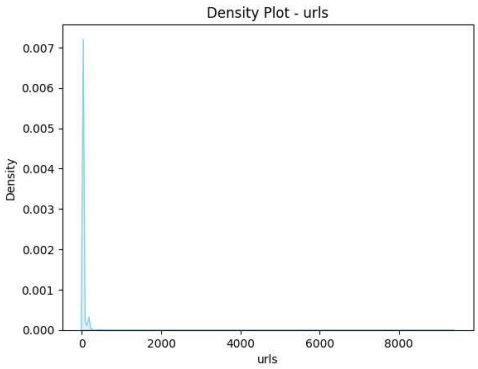

Density Plot - has_signature
Density
has_signature

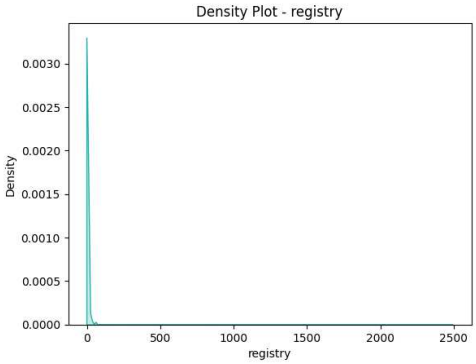

Density Plot - has_tls
Density
has_tls

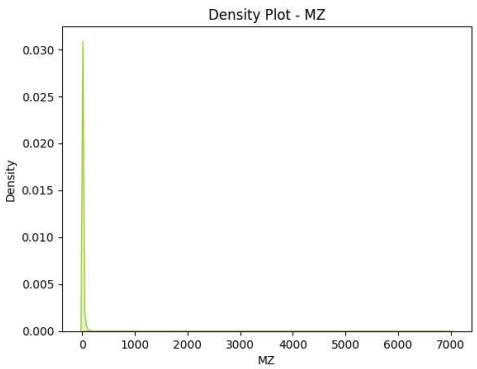

Density Plot - symbols
1e−5
Density
symbols

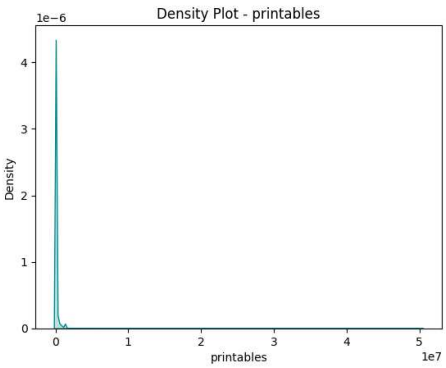

Density Plot - numstrings
1e−5
Density
numstrings
1e6

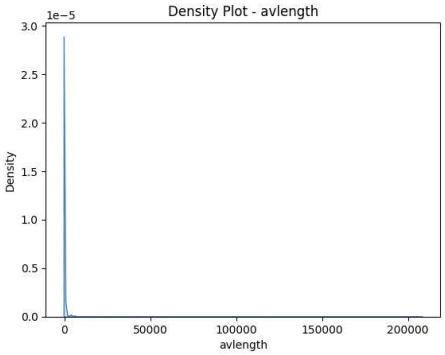

Density Plot - paths
Density
paths

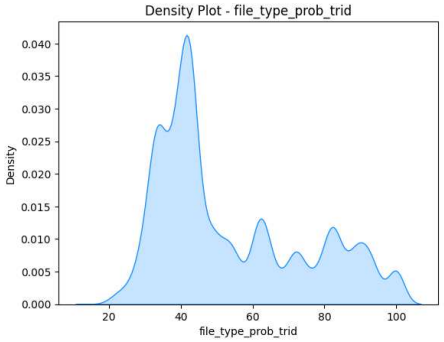

Density Plot - urls
Density
urls

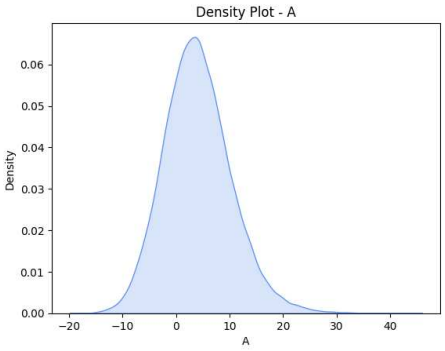

Density Plot - registry
Density
registry

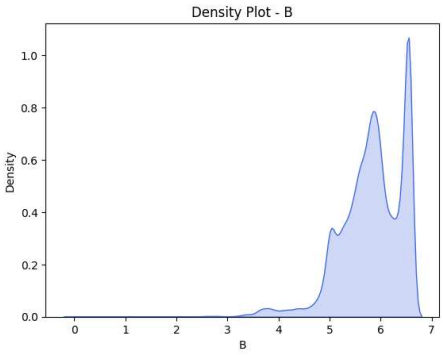

Density Plot - MZ
Density
MZ

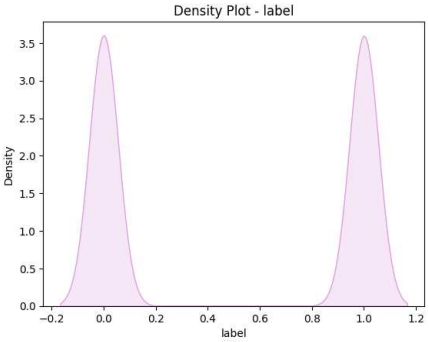

Density Plot - printables
1e−6
Density
printables
1e7


Density Plot - avlength
1e−5
Density
avlength

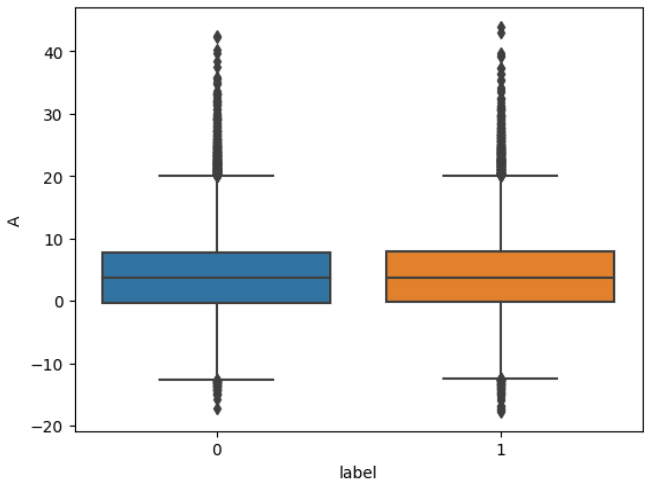

Density Plot - file_type_prob_trid
Density
file_type_prob_trid


Density Plot - A
Density
A

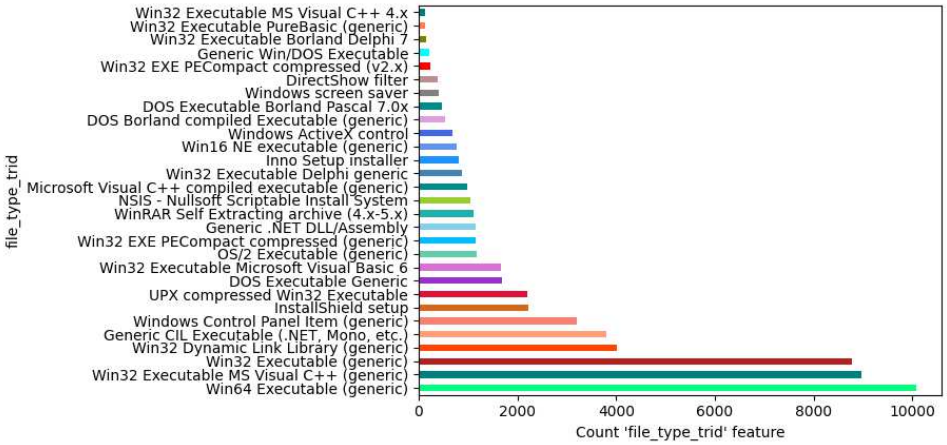

Density Plot - B
Density
B


Density Plot - label
Density
label

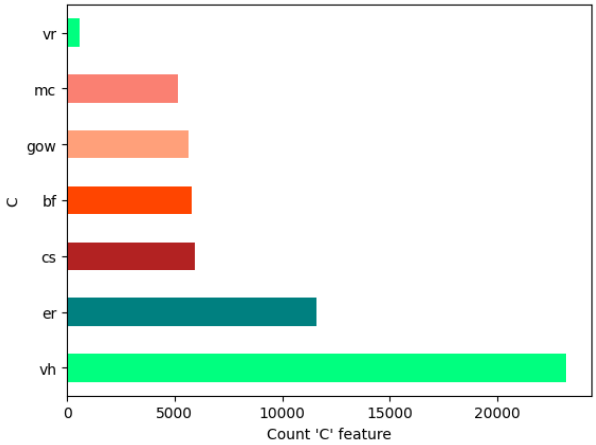
1.3


A
label

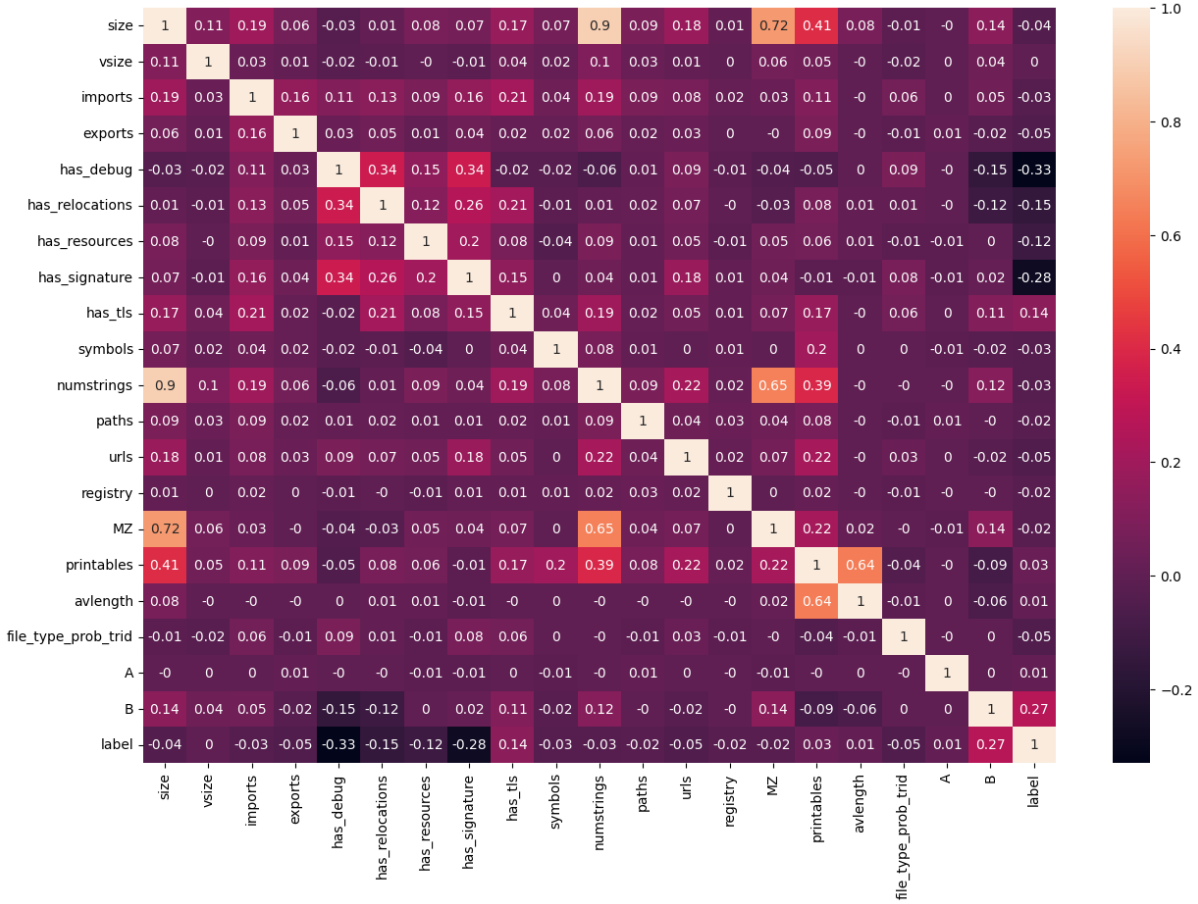
1.4


Win32 Executable MS Visual C++ 4.x
Win32 Executable PureBasic (generic)
Win32 Executable Borland Delphi 7
Generic Win/DOS Executable
Win32 EXE PECompact compressed (v2.x)
DirectShow filter
Windows screen saver
DOS Executable Borland Pascal 7.0x
DOS Borland compiled Executable (generic)
Windows ActiveX control
Win16 NE executable (generic)
Inno Setup installer
Win32 Executable Delphi generic
Microsoft Visual C++ compiled executable (generic)
NSIS - Nullsoft Scriptable Install System
WinRAR Self Extracting archive (4.x-5.x)
Generic .NET DLL/Assembly
Win32 EXE PECompact compressed (generic)
OS/2 Executable (generic)
Win32 Executable Microsoft Visual Basic 6
DOS Executable Generic
UPX compressed Win32 Executable
InstallShield setup
Windows Control Panel Item (generic)
Generic CIL Executable (.NET, Mono, etc.)
Win32 Dynamic Link Library (generic)
Win32 Executable (generic)
Win32 Executable MS Visual C++ (generic)
Win64 Executable (generic)
file_type_trid
0
2000
4000
6000
8000
10000
Count 'file_type_trid' feature

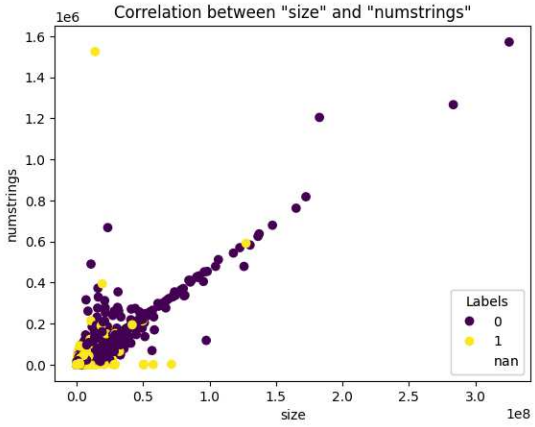
1.5


vr
mc
gow
bf
cs
er
vh
C
0
5000
10000
15000
20000
Count 'C' feature

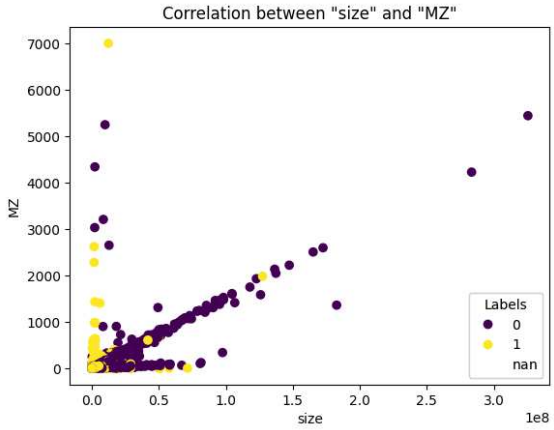
1.6


size 1 0.11 0.19 0.06 -0.03 0.01 0.08 0.07 0.17 0.07 0.9 0.09 0.18 0.01 0.72 0.41 0.08 -0.01 -0 0.14 -0.04
vsize 0.11 1 0.03 0.01 -0.02 -0.01 -0 -0.01 0.04 0.02 0.1 0.03 0.01 0 0.06 0.05 -0 -0.02 0 0.04 0
imports 0.19 0.03 1 0.16 0.11 0.13 0.09 0.16 0.21 0.04 0.19 0.09 0.08 0.02 0.03 0.11 -0 0.06 0 0.05 -0.03
exports 0.06 0.01 0.16 1 0.03 0.05 0.01 0.04 0.02 0.02 0.06 0.02 0.03 0 -0 0.09 -0 -0.01 0.01 -0.02 -0.05
has_debug -0.03 -0.02 0.11 0.03 1 0.34 0.15 0.34 -0.02 -0.02 -0.06 0.01 0.09 -0.01 -0.04 -0.05 0 0.09 -0 -0.15 -0.33
has_relocations 0.01 -0.01 0.13 0.05 0.34 1 0.12 0.26 0.21 -0.01 0.01 0.02 0.07 -0 -0.03 0.08 0.01 0.01 -0 -0.12 -0.15
has_resources 0.08 -0 0.09 0.01 0.15 0.12 1 0.2 0.08 -0.04 0.09 0.01 0.05 -0.01 0.05 0.06 0.01 -0.01 -0.01 0 -0.12
has_signature 0.07 -0.01 0.16 0.04 0.34 0.26 0.2 1 0.15 0 0.04 0.01 0.18 0.01 0.04 -0.01 -0.01 0.08 -0.01 0.02 -0.28
has_tls 0.17 0.04 0.21 0.02 -0.02 0.21 0.08 0.15 1 0.04 0.19 0.02 0.05 0.01 0.07 0.17 -0 0.06 0 0.11 0.14
symbols 0.07 0.02 0.04 0.02 -0.02 -0.01 -0.04 0 0.04 1 0.08 0.01 0 0.01 0 0.2 0 0 -0.01 -0.02 -0.03
numstrings 0.9 0.1 0.19 0.06 -0.06 0.01 0.09 0.04 0.19 0.08 1 0.09 0.22 0.02 0.65 0.39 -0 -0 -0 0.12 -0.03
paths 0.09 0.03 0.09 0.02 0.01 0.02 0.01 0.01 0.02 0.01 0.09 1 0.04 0.03 0.04 0.08 -0 -0.01 0.01 -0 -0.02
urls 0.18 0.01 0.08 0.03 0.09 0.07 0.05 0.18 0.05 0 0.22 0.04 1 0.02 0.07 0.22 -0 0.03 0 -0.02 -0.05
registry 0.01 0 0.02 0 -0.01 -0 -0.01 0.01 0.01 0.01 0.02 0.03 0.02 1 0 0.02 -0 -0.01 -0 -0 -0.02
MZ 0.72 0.06 0.03 -0 -0.04 -0.03 0.05 0.04 0.07 0 0.65 0.04 0.07 0 1 0.22 0.02 -0 -0.01 0.14 -0.02
printables 0.41 0.05 0.11 0.09 -0.05 0.08 0.06 -0.01 0.17 0.2 0.39 0.08 0.22 0.02 0.22 1 0.64 -0.04 -0 -0.09 0.03
avlength 0.08 -0 -0 -0 0 0.01 0.01 -0.01 -0 0 -0 -0 -0 -0 0.02 0.64 1 -0.01 0 -0.06 0.01
file_type_prob_trid -0.01 -0.02 0.06 -0.01 0.09 0.01 -0.01 0.08 0.06 0 -0 -0.01 0.03 -0.01 -0 -0.04 -0.01 1 -0 0 -0.05
A -0 0 0 0.01 -0 -0 -0.01 -0.01 0 -0.01 -0 0.01 0 -0 -0.01 -0 0 -0 1 0 0.01
B 0.14 0.04 0.05 -0.02 -0.15 -0.12 0 0.02 0.11 -0.02 0.12 -0 -0.02 -0 0.14 -0.09 -0.06 0 0 1 0.27
label -0.04 0 -0.03 -0.05 -0.33 -0.15 -0.12 -0.28 0.14 -0.03 -0.03 -0.02 -0.05 -0.02 -0.02 0.03 0.01 -0.05 0.01 0.27 1
size vsize imports exports has_debug has_relocations has_resources has_signature has_tls symbols numstrings paths urls registry MZ printables avlength file_type_prob_trid A B label
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

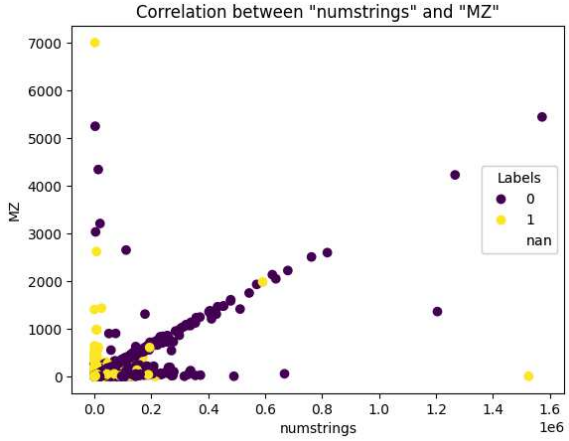
1.7


Correlation between "size" and "numstrings"
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
numstrings
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e8
size
Labels
0
1
nan

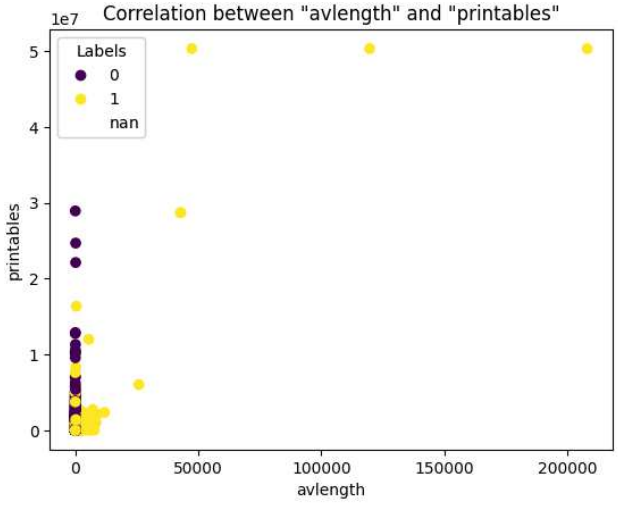
1.8


Correlation between "size" and "MZ"
MZ
size
1e8
Labels
0
1
nan

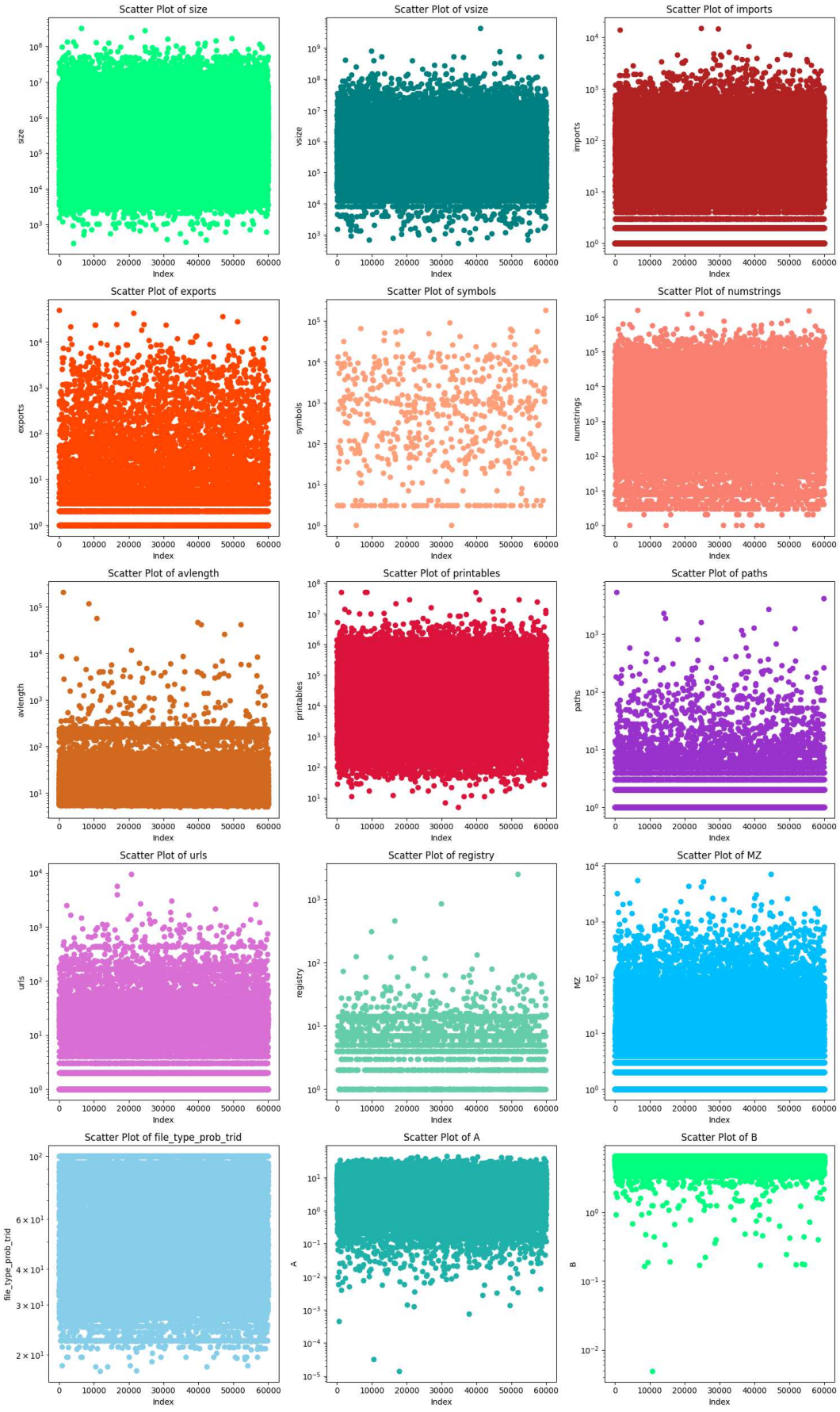
1.9


Correlation between "numstrings" and "MZ"
MZ
numstrings
1e6
Labels
0
1
nan

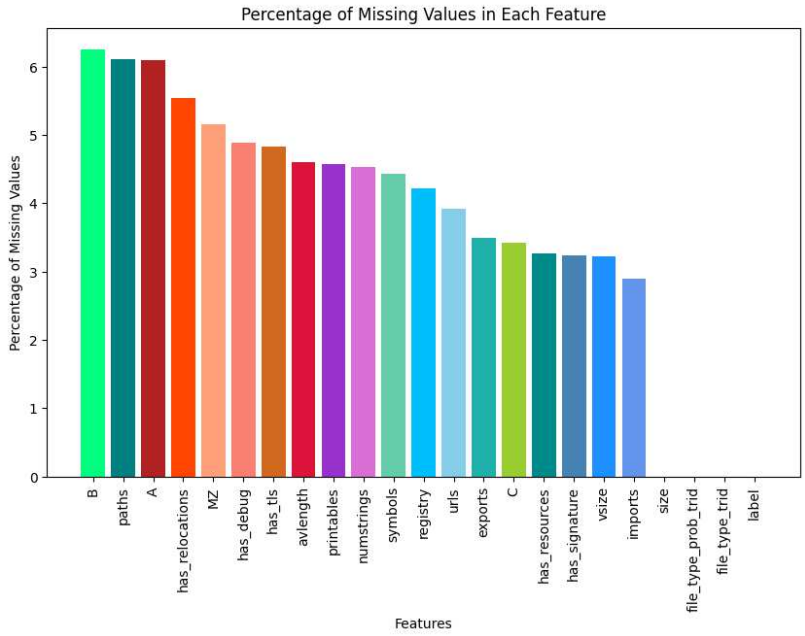
1.10


Correlation between "avlength" and "printables"
1e7
printables
avlength
Labels
0
1
nan

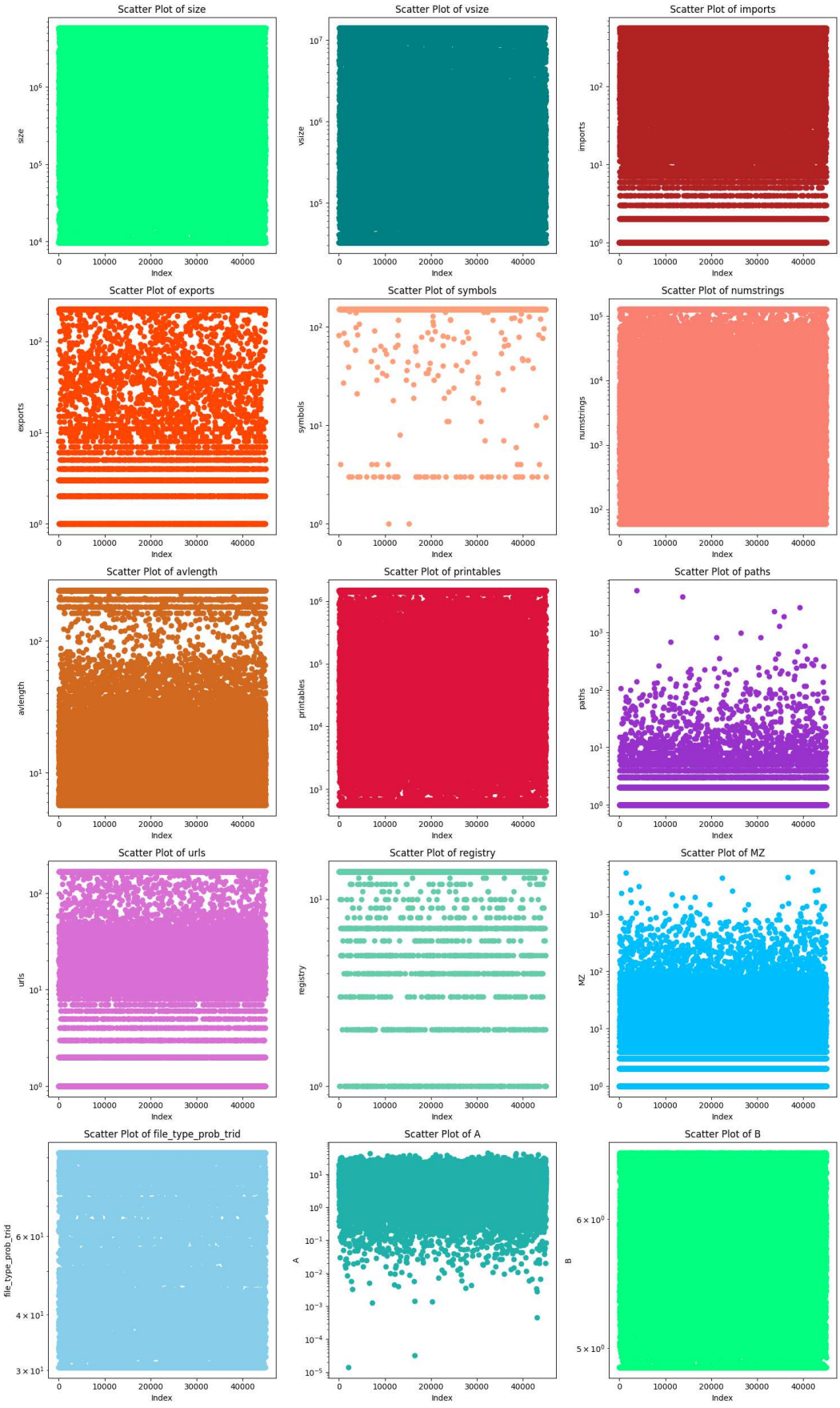
1.11


Scatter Plot of size
Scatter Plot of vsize
Scatter Plot of imports
Scatter Plot of exports
Scatter Plot of symbols
Scatter Plot of numstrings
Scatter Plot of avlength
Scatter Plot of printables
Scatter Plot of paths
Scatter Plot of urls
Scatter Plot of registry
Scatter Plot of MZ
Scatter Plot of file_type_prob_trid
Scatter Plot of A
Scatter Plot of B
Index

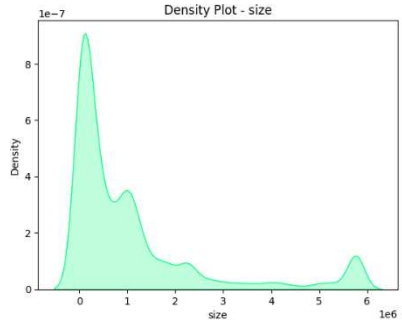
1.12

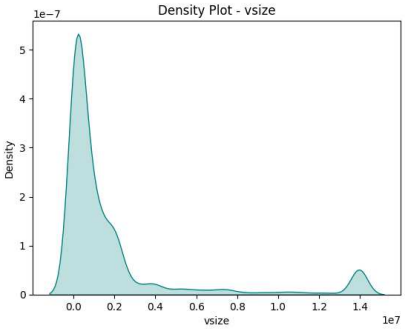

Percentage of Missing Values in Each Feature
Percentage of Missing Values
0
1
2
3
4
5
6
B
paths
A
has_relocations
MZ
has_debug
has_tls
avlength
printables
numstrings
symbols
registry
urls
exports
C
has_resources
has_signature
vsize
imports
size
file_type_prob_trid
file_type_trid
label
Features

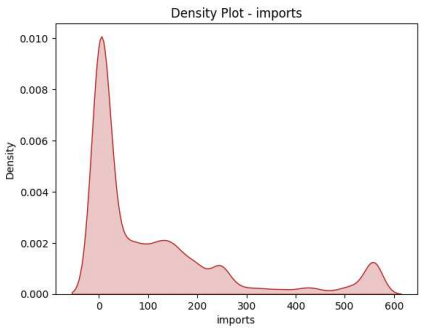
2.1

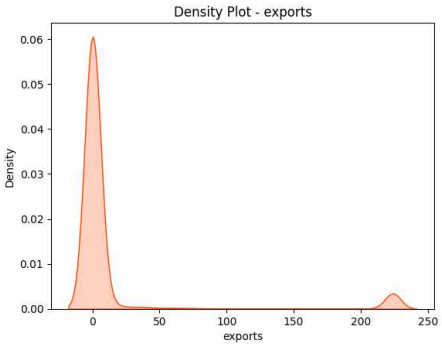

Scatter Plot of size
size
Index
Scatter Plot of vsize
vsize
Index
Scatter Plot of imports
imports
Index
Scatter Plot of exports
exports
Index
Scatter Plot of symbols
symbols
Index
Scatter Plot of numstrings
numstrings
Index
Scatter Plot of avlength
avlength
Index
Scatter Plot of printables
printables
Index
Scatter Plot of paths
paths
Index
Scatter Plot of urls
urls
Index
Scatter Plot of registry
registry
Index
Scatter Plot of MZ
MZ
Index
Scatter Plot of file_type_prob_trid
file_type_prob_trid
Index
Scatter Plot of A
A
Index
Scatter Plot of B
B
Index

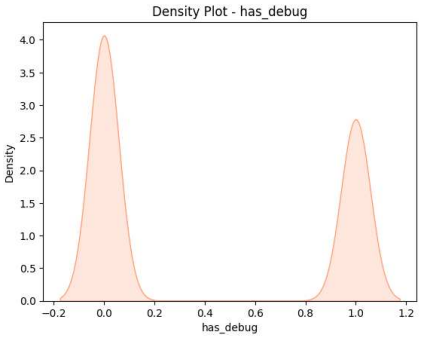
2.2

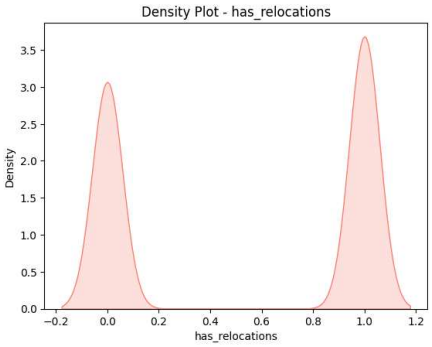

Density Plot - size
1e−7
Density
size
1e6

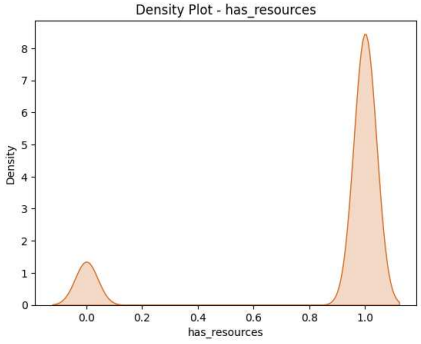

Density Plot - vsize
1e−7
Density
vsize
1e7

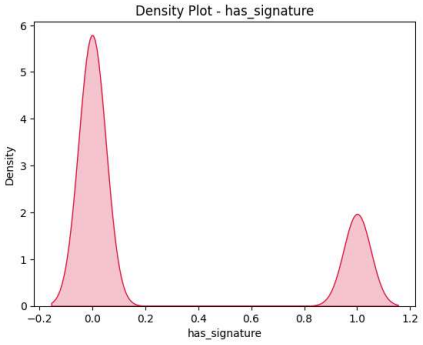

Density Plot - imports
Density
imports

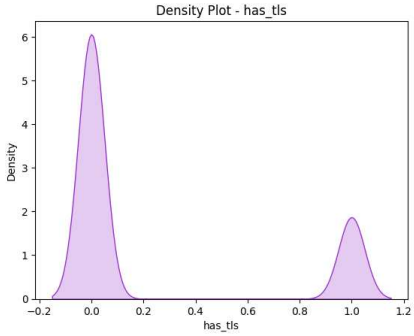

Density Plot - exports
Density
exports

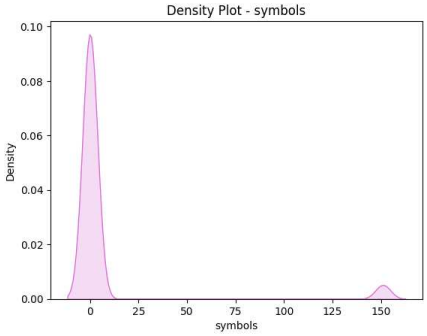

Density Plot - has_debug
Density
has_debug

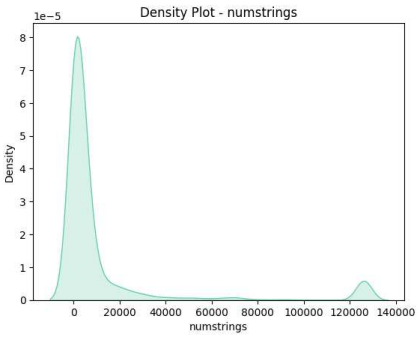

Density Plot - has_relocations
Density
has_relocations

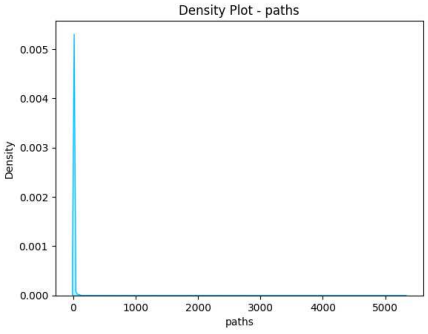

Density Plot - has_resources
Density
has_resources

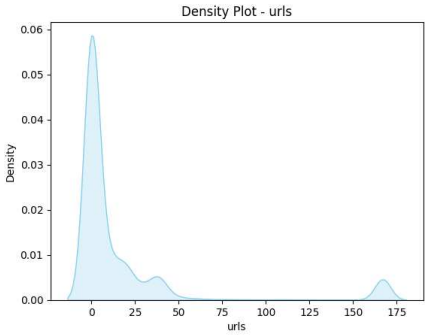

Density Plot - has_signature
Density
has_signature

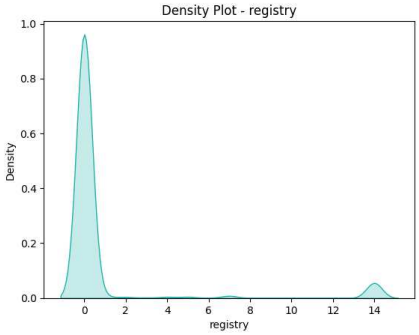

Density Plot - has_tls
Density
has_tls

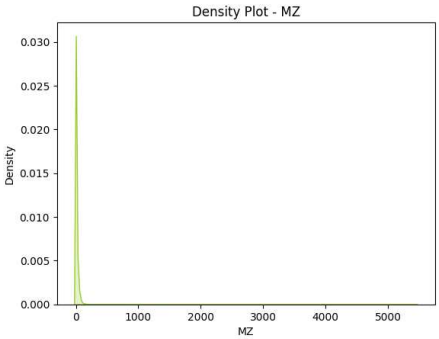

Density Plot - symbols
Density
symbols

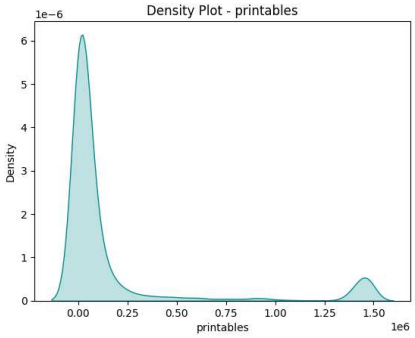

Density Plot - numstrings
1e−5
Density
numstrings

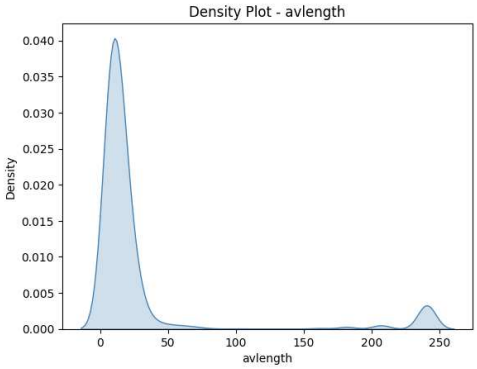

Density Plot - paths
Density
paths

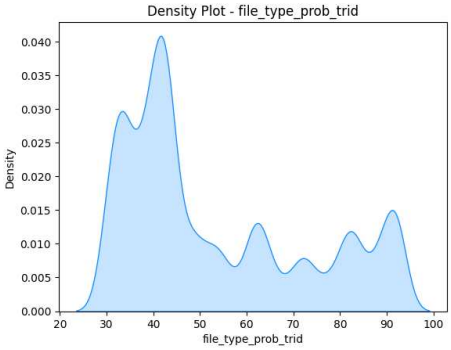

Density Plot - urls
Density
urls

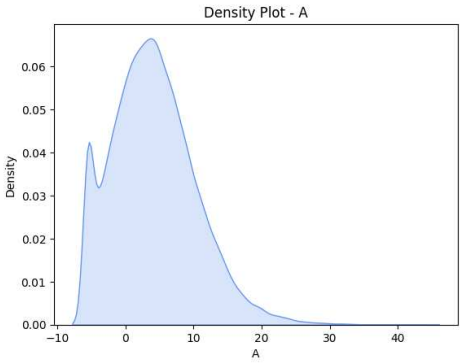

Density Plot - registry
Density
registry

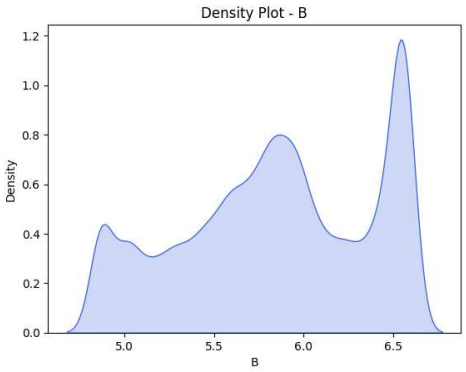

Density Plot - MZ
Density
MZ


Density Plot - printables
1e−6
Density
printables
1e6

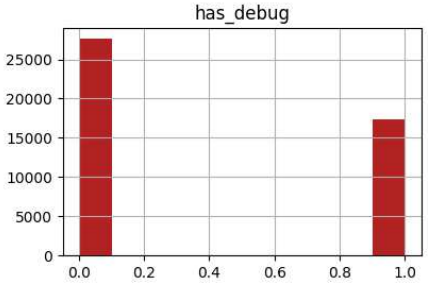

Density Plot - avlength
Density
avlength

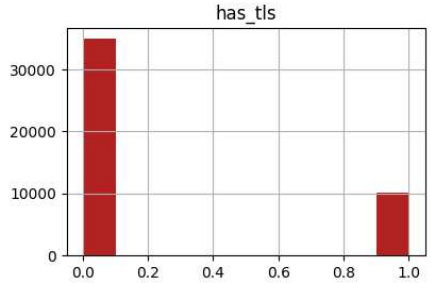

Density Plot - file_type_prob_trid
Density
file_type_prob_trid

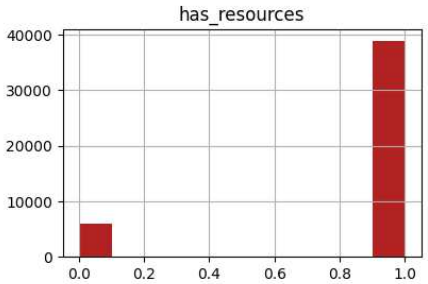

Density Plot - A
Density
A

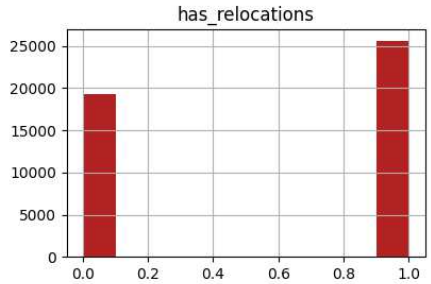

Density Plot - B
Density
B

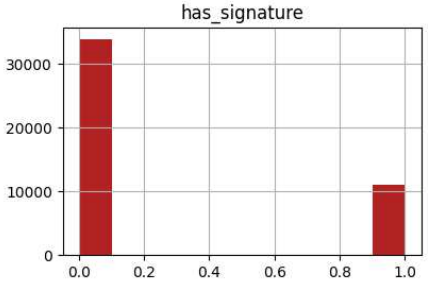
2.3


has_debug

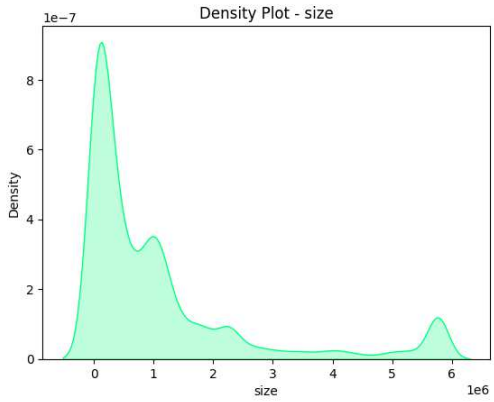

has_tls

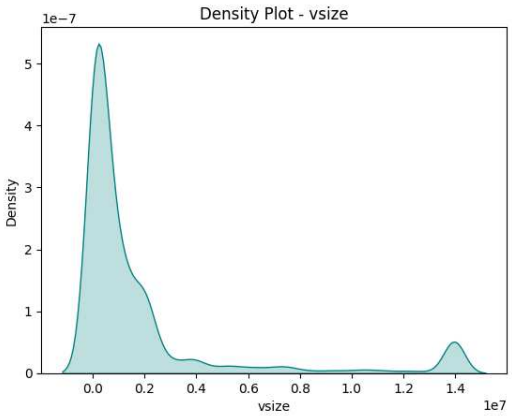

has_resources

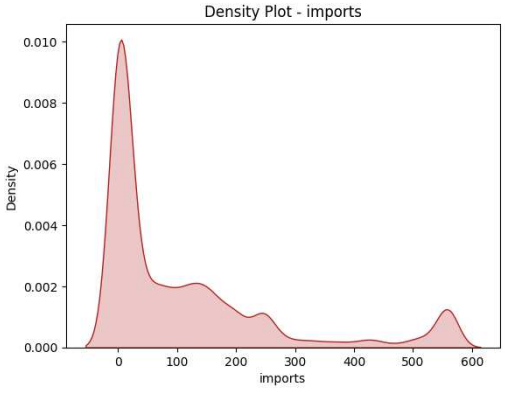

has_relocations

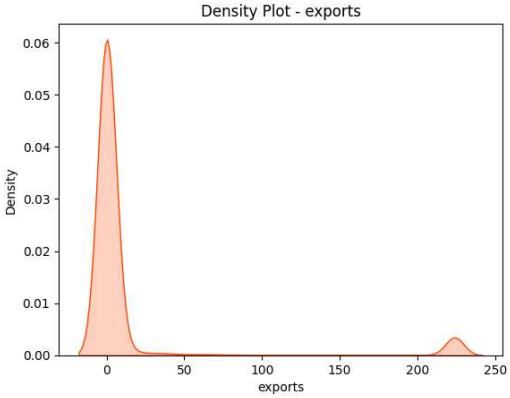

has_signature

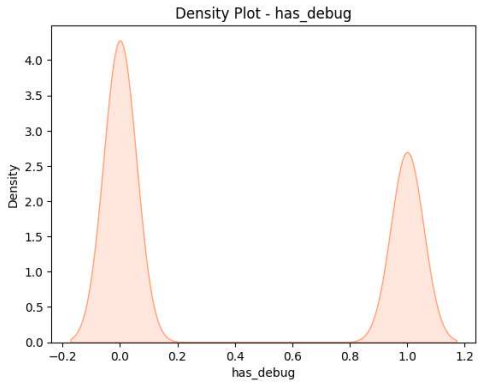
2.4

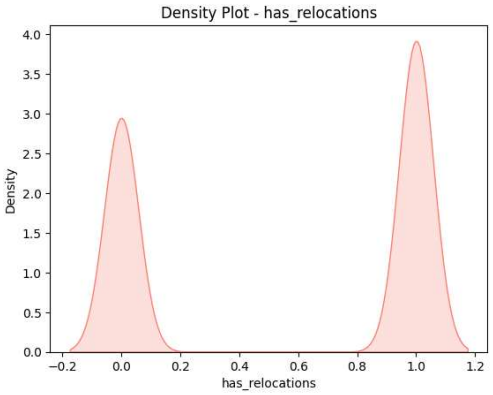

Density Plot - size
1e−7
Density
size
1e6

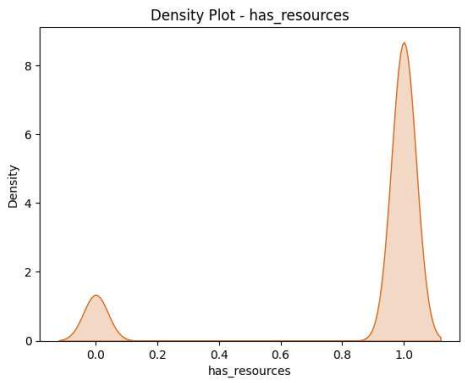

Density Plot - vsize
1e−7
Density
vsize
1e7

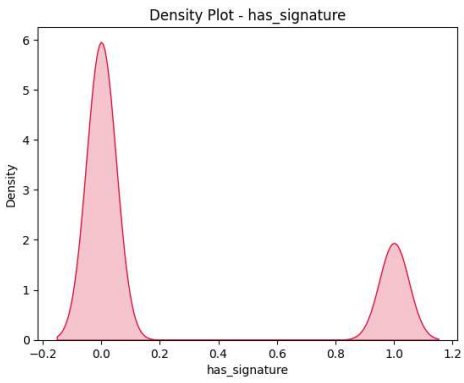

Density Plot - imports
Density
imports

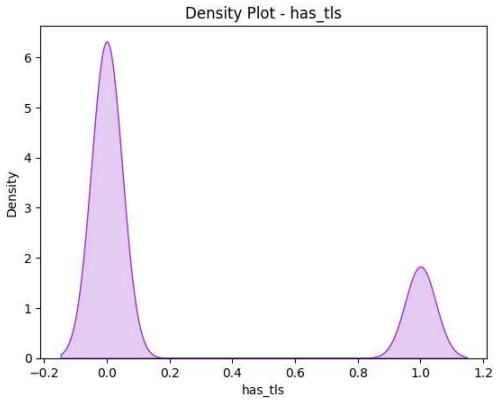

Density Plot - exports
Density
exports

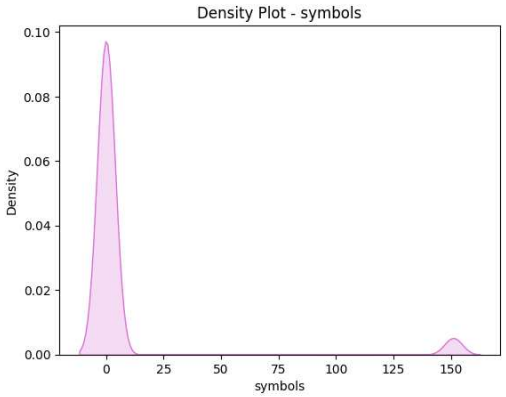

Density Plot - has_debug
Density
has_debug

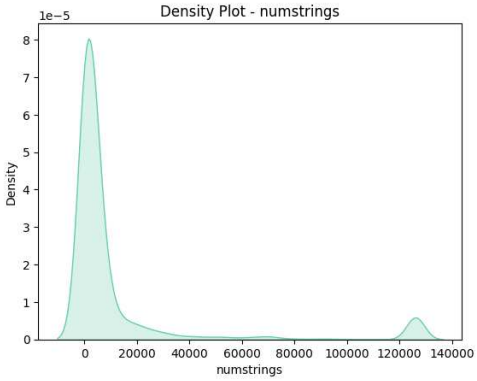

Density Plot - has_relocations
Density
has_relocations

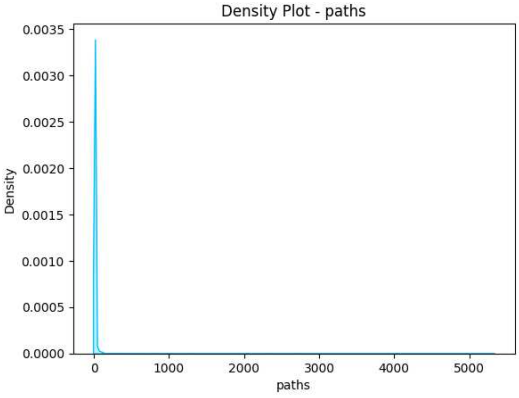

Density Plot - has_resources
Density
has_resources

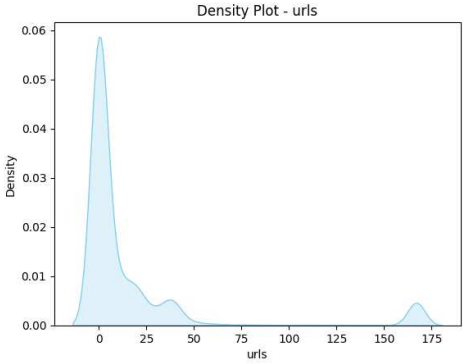

Density Plot - has_signature
Density
has_signature

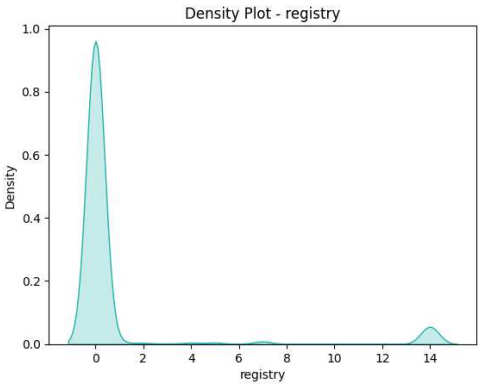

Density Plot - has_tls
Density
has_tls

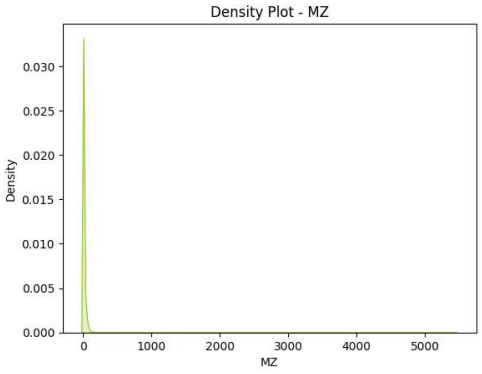

Density Plot - symbols
Density
symbols

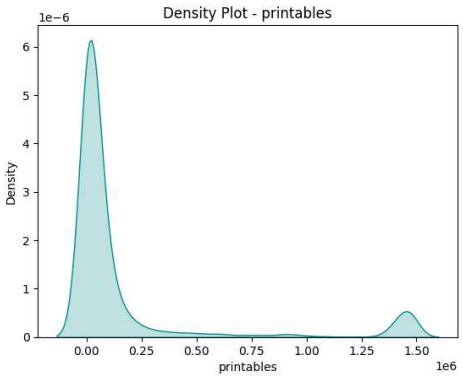

Density Plot - numstrings
1e−5
Density
numstrings

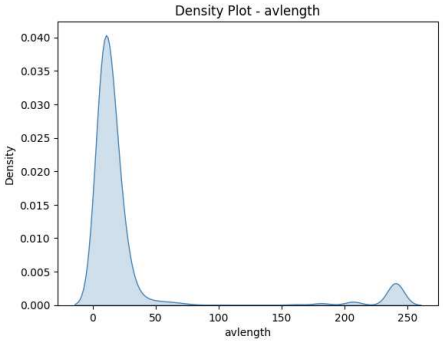

Density Plot - paths
Density
paths

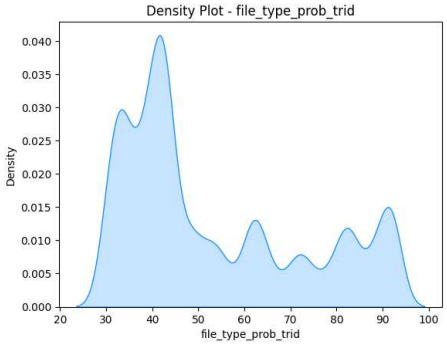

Density Plot - urls
Density
urls

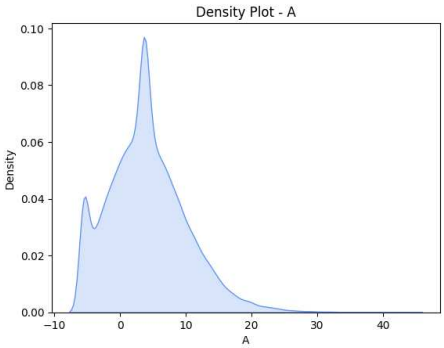

Density Plot - registry
Density
registry

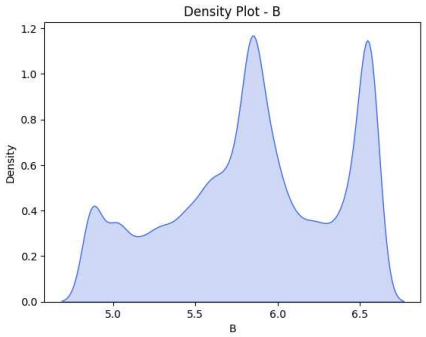

Density Plot - MZ
Density
MZ


Density Plot - printables
1e−6
Density
printables
1e6

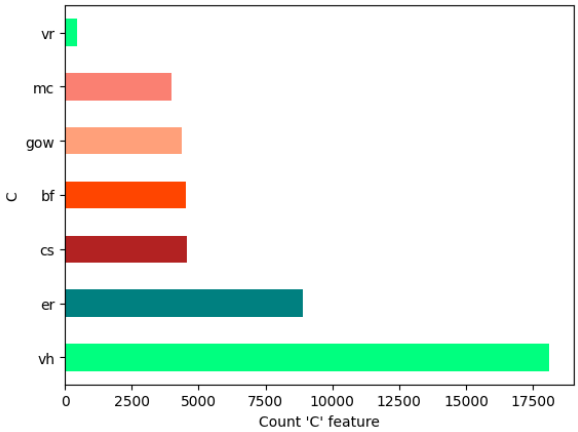

Density Plot - avlength
Density
avlength


Density Plot - file_type_prob_trid
Density
file_type_prob_trid

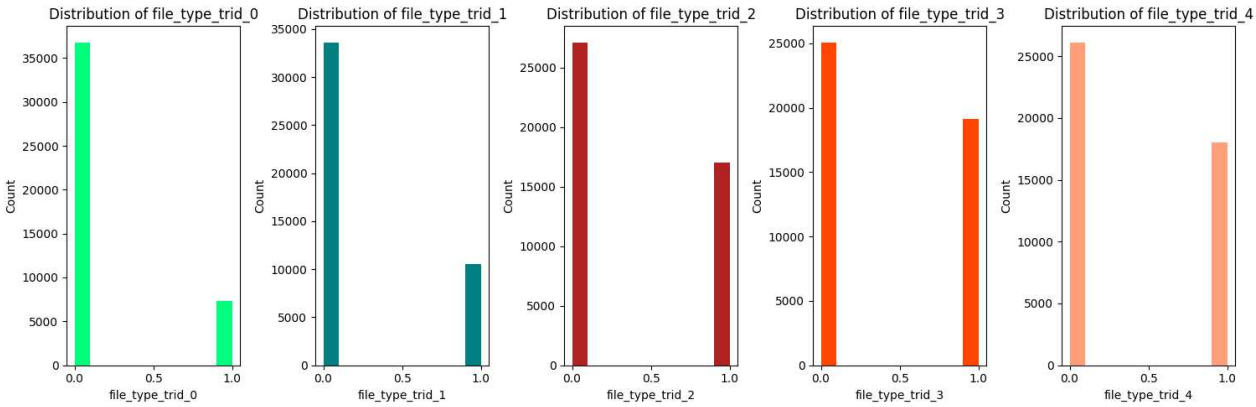

Density Plot - A
Density
A


Density Plot - B
Density
B

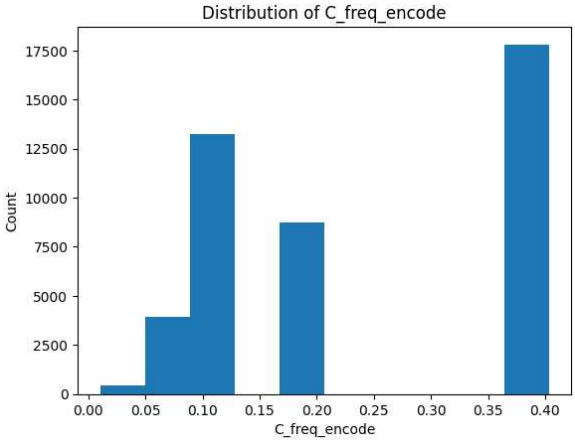
2.5


vr
mc
gow
bf
cs
er
vh
C
Count 'C' feature

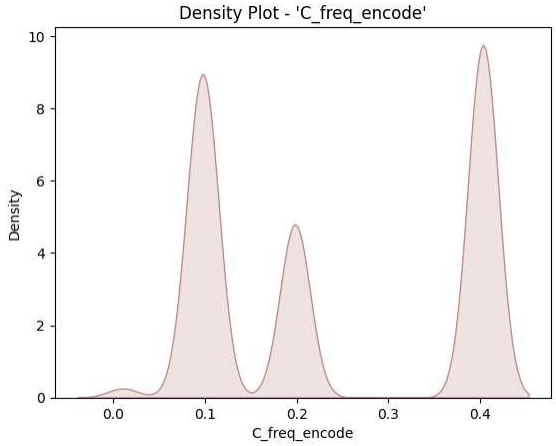
2.6


Distribution of file_type_trid_0
Distribution of file_type_trid_1
Distribution of file_type_trid_2
Distribution of file_type_trid_3
Distribution of file_type_trid_4
Count
file_type_trid_0
file_type_trid_1
file_type_trid_2
file_type_trid_3
file_type_trid_4

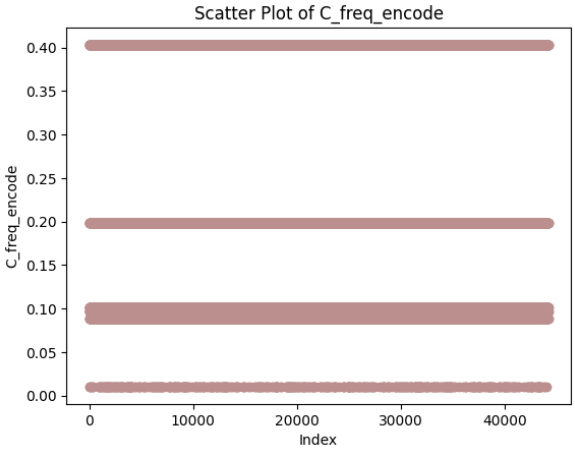
2.7


Distribution of C_freq_encode
Count
C_freq_encode
0
2500
5000
7500
10000
12500
15000
17500
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40

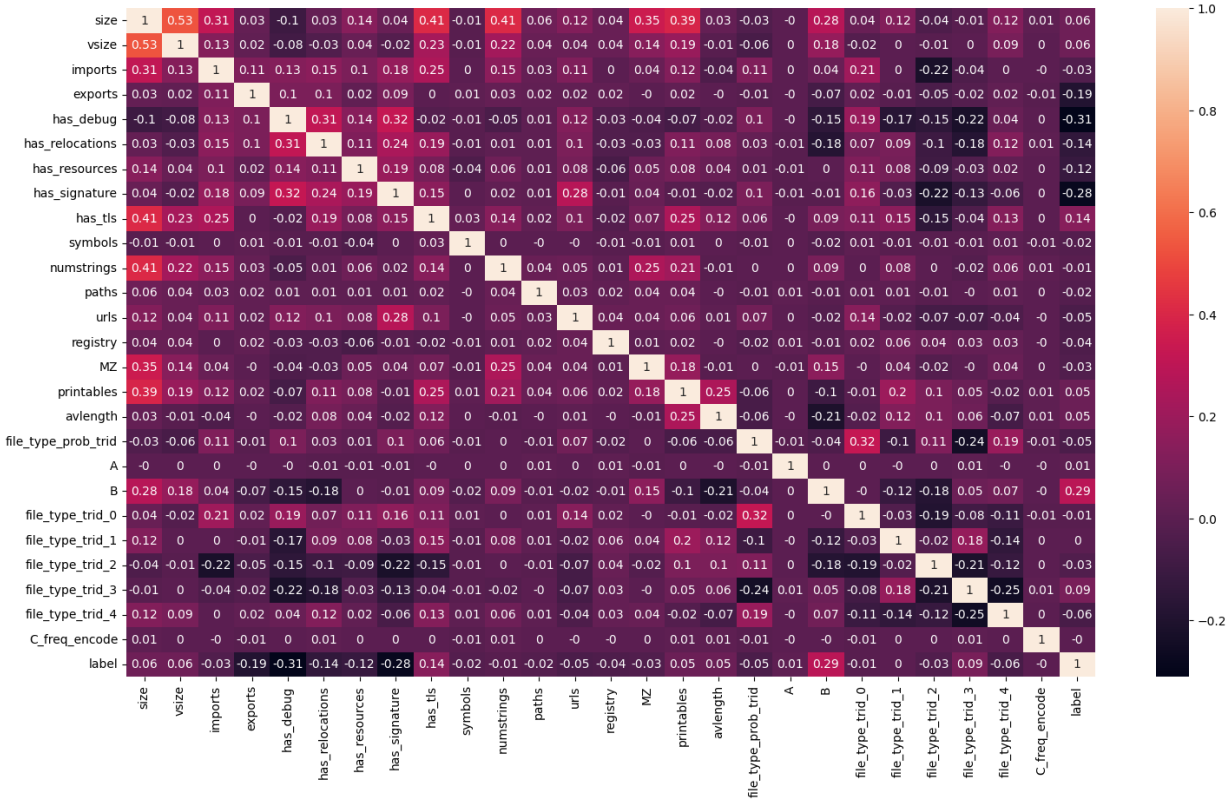
2.8


Density Plot - 'C_freq_encode'
Density
C_freq_encode
0
2
4
6
8
10
0.0
0.1
0.2
0.3
0.4

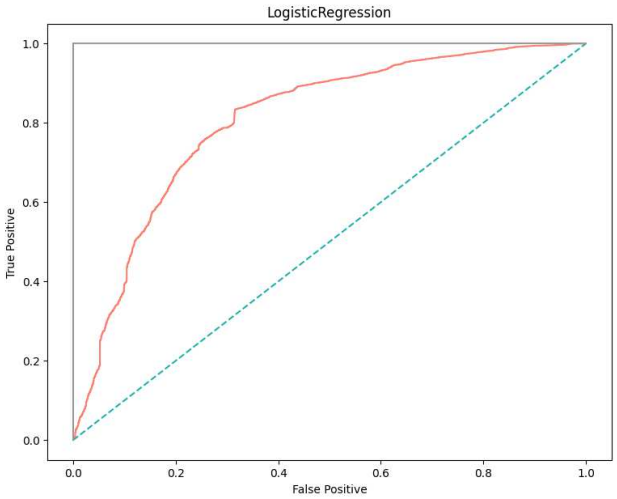
2.9


Scatter Plot of C_freq_encode
C_freq_encode
Index
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0
10000
20000
30000
40000

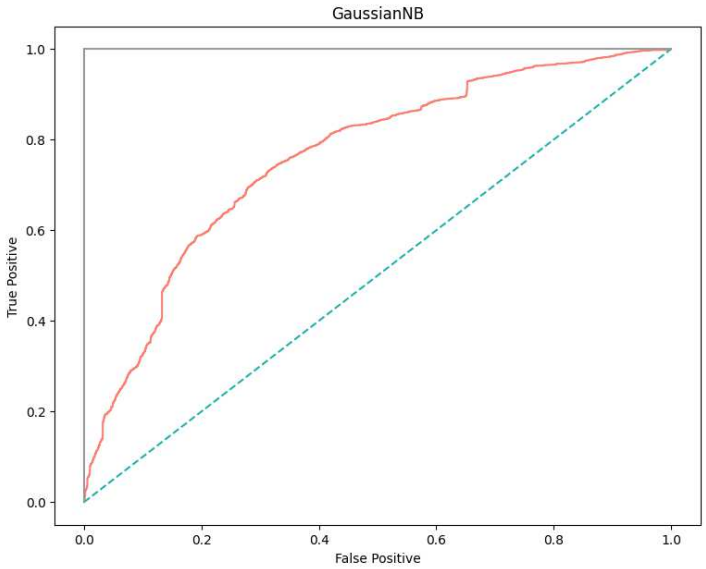
2.10


size 1 0.53 0.31 0.03 -0.1 0.03 0.14 0.04 0.41 -0.01 0.41 0.06 0.12 0.04 0.35 0.39 0.03 -0.03 -0 0.28 0.04 0.12 -0.04 -0.01 0.12 0.01 0.06
vsize 0.53 1 0.13 0.02 -0.08 -0.03 0.04 -0.02 0.23 -0.01 0.22 0.04 0.04 0.04 0.14 0.19 -0.01 -0.06 0 0.18 -0.02 0 -0.01 0 0.09 0 0.06
imports 0.31 0.13 1 0.11 0.13 0.15 0.1 0.18 0.25 0 0.15 0.03 0.11 0 0.04 0.12 -0.04 0.11 0 0.04 0.21 0 -0.22 -0.04 0 -0 -0.03
exports 0.03 0.02 0.11 1 0.1 0.1 0.02 0.09 0 0.01 0.03 0.02 0.02 0.02 -0 0.02 -0 -0.01 -0 -0.07 0.02 -0.01 -0.05 -0.02 0.02 -0.01 -0.19
has_debug -0.1 -0.08 0.13 0.1 1 0.31 0.14 0.32 -0.02 -0.01 -0.05 0.01 0.12 -0.03 -0.04 -0.07 -0.02 0.1 -0 -0.15 0.19 -0.17 -0.15 -0.22 0.04 0 -0.31
has_relocations 0.03 -0.03 0.15 0.1 0.31 1 0.11 0.24 0.19 -0.01 0.01 0.01 0.1 -0.03 -0.03 0.11 0.08 0.03 -0.01 -0.18 0.07 0.09 -0.1 -0.18 0.12 0.01 -0.14
has_resources 0.14 0.04 0.1 0.02 0.14 0.11 1 0.19 0.08 -0.04 0.06 0.01 0.08 -0.06 0.05 0.08 0.04 0.01 -0.01 0 0.11 0.08 -0.09 -0.03 0.02 0 -0.12
has_signature 0.04 -0.02 0.18 0.09 0.32 0.24 0.19 1 0.15 0 0.02 0.01 0.28 -0.01 0.04 -0.01 -0.02 0.1 -0.01 -0.01 0.16 -0.03 -0.22 -0.13 -0.06 0 -0.28
has_tls 0.41 0.23 0.25 0 -0.02 0.19 0.08 0.15 1 0.03 0.14 0.02 0.1 -0.02 0.07 0.25 0.12 0.06 -0 0.09 0.11 0.15 -0.15 -0.04 0.13 0 0.14
symbols -0.01 -0.01 0 0.01 -0.01 -0.01 -0.04 0 0.03 1 0 -0 -0 -0.01 -0.01 0.01 0 -0.01 0 -0.02 0.01 -0.01 -0.01 -0.01 0.01 -0.01 -0.02
numstrings 0.41 0.22 0.15 0.03 -0.05 0.01 0.06 0.02 0.14 0 1 0.04 0.05 0.01 0.25 0.21 -0.01 0 0 0.09 0 0.08 0 -0.02 0.06 0.01 -0.01
paths 0.06 0.04 0.03 0.02 0.01 0.01 0.01 0.01 0.02 -0 0.04 1 0.03 0.02 0.04 0.04 -0 -0.01 0.01 -0.01 0.01 0.01 -0.01 -0 0.01 0 -0.02
urls 0.12 0.04 0.11 0.02 0.12 0.1 0.08 0.28 0.1 -0 0.05 0.03 1 0.04 0.04 0.06 0.01 0.07 0 -0.02 0.14 -0.02 -0.07 -0.07 -0.04 -0 -0.05
registry 0.04 0.04 0 0.02 -0.03 -0.03 -0.06 -0.01 -0.02 -0.01 0.01 0.02 0.04 1 0.01 0.02 -0 -0.02 0.01 -0.01 0.02 0.06 0.04 0.03 0.03 -0 -0.04
MZ 0.35 0.14 0.04 -0 -0.04 -0.03 0.05 0.04 0.07 -0.01 0.25 0.04 0.04 0.01 1 0.18 -0.01 0 -0.01 0.15 -0 0.04 -0.02 -0 0.04 0 -0.03
printables 0.39 0.19 0.12 0.02 -0.07 0.11 0.08 -0.01 0.25 0.01 0.21 0.04 0.06 0.02 0.18 1 0.25 -0.06 0 -0.1 -0.01 0.2 0.1 0.05 -0.02 0.01 0.05
avlength 0.03 -0.01 -0.04 -0 -0.02 0.08 0.04 -0.02 0.12 0 -0.01 -0 0.01 -0 -0.01 0.25 1 -0.06 -0 -0.21 -0.02 0.12 0.1 0.06 -0.07 0.01 0.05
file_type_prob_trid -0.03 -0.06 0.11 -0.01 0.1 0.03 0.01 0.1 0.06 -0.01 0 -0.01 0.07 -0.02 0 -0.06 -0.06 1 -0.01 -0.04 0.32 -0.1 0.11 -0.24 0.19 -0.01 -0.05
A -0 0 0 -0 -0 -0.01 -0.01 -0.01 -0 0 0 0.01 0 0.01 -0.01 0 -0 -0.01 1 0 0 -0 0 0.01 -0 -0 0.01
B 0.28 0.18 0.04 -0.07 -0.15 -0.18 0 -0.01 0.09 -0.02 0.09 -0.01 -0.02 -0.01 0.15 -0.1 -0.21 -0.04 0 1 -0 -0.12 -0.18 0.05 0.07 -0 0.29
file_type_trid_0 0.04 -0.02 0.21 0.02 0.19 0.07 0.11 0.16 0.11 0.01 0 0.01 0.14 0.02 -0 -0.01 -0.02 0.32 0 -0 1 -0.03 -0.19 -0.08 -0.11 -0.01 -0.01
file_type_trid_1 0.12 0 0 -0.01 -0.17 0.09 0.08 -0.03 0.15 -0.01 0.08 0.01 -0.02 0.06 0.04 0.2 0.12 -0.1 -0 -0.12 -0.03 1 -0.02 0.18 -0.14 0 0
file_type_trid_2 -0.04 -0.01 -0.22 -0.05 -0.15 -0.1 -0.09 -0.22 -0.15 -0.01 0 -0.01 -0.07 0.04 -0.02 0.1 0.1 0.11 0 -0.18 -0.19 -0.02 1 -0.21 -0.12 0 -0.03
file_type_trid_3 -0.01 0 -0.04 -0.02 -0.22 -0.18 -0.03 -0.13 -0.04 -0.01 -0.02 -0 -0.07 0.03 -0 0.05 0.06 -0.24 0.01 0.05 -0.08 0.18 -0.21 1 -0.25 0.01 0.09
file_type_trid_4 0.12 0.09 0 0.02 0.04 0.12 0.02 -0.06 0.13 0.01 0.06 0.01 -0.04 0.03 0.04 -0.02 -0.07 0.19 -0 0.07 -0.11 -0.14 -0.12 -0.25 1 0 -0.06
C_freq_encode 0.01 0 -0 -0.01 0 0.01 0 0 0 -0.01 0.01 0 -0 -0 0 0.01 0.01 -0.01 -0 -0 -0.01 0 0 0.01 0 1 -0
label 0.06 0.06 -0.03 -0.19 -0.31 -0.14 -0.12 -0.28 0.14 -0.02 -0.01 -0.02 -0.05 -0.04 -0.03 0.05 0.05 -0.05 0.01 0.29 -0.01 0 -0.03 0.09 -0.06 -0 1
size vsize imports exports has_debug has_relocations has_resources has_signature has_tls symbols numstrings paths urls registry MZ printables avlength file_type_prob_trid A B file_type_trid_0 file_type_trid_1 file_type_trid_2 file_type_trid_3 file_type_trid_4 C_freq_encode label
1.0 0.8 0.6 0.4 0.2 0.0 −0.2

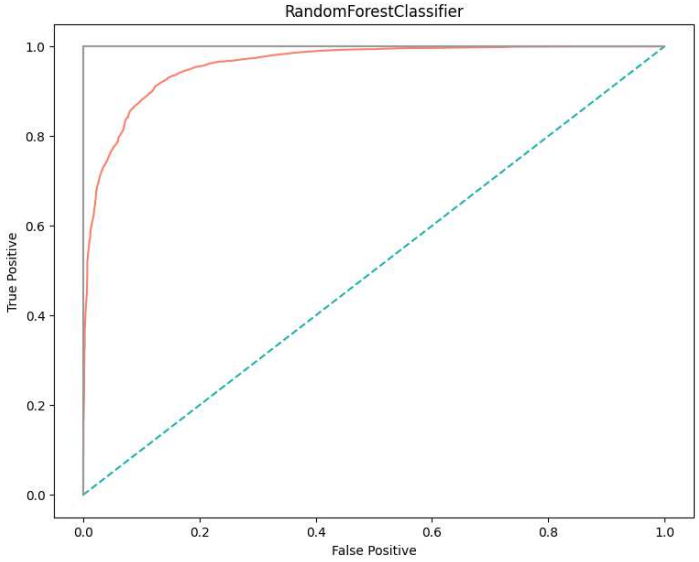
3.1


LogisticRegression
True Positive
False Positive
0.0 0.2 0.4 0.6 0.8 1.0

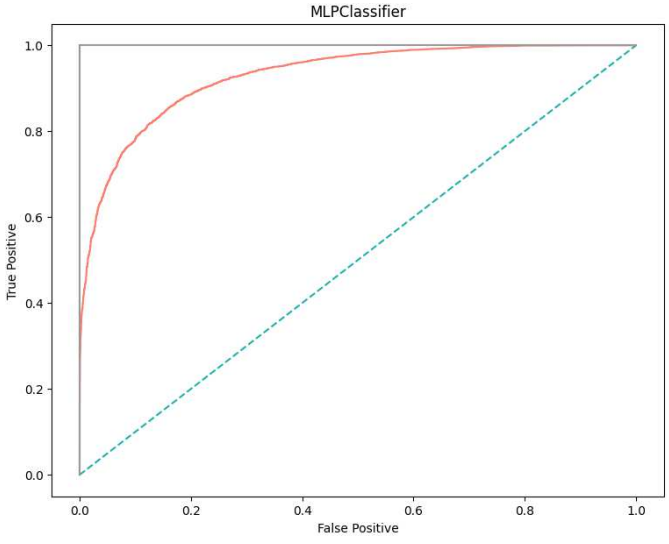
3.2


GaussianNB
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

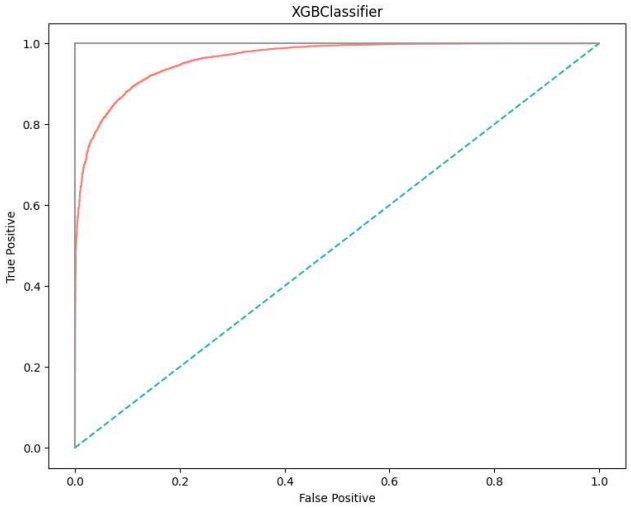
3.3


RandomForestClassifier
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

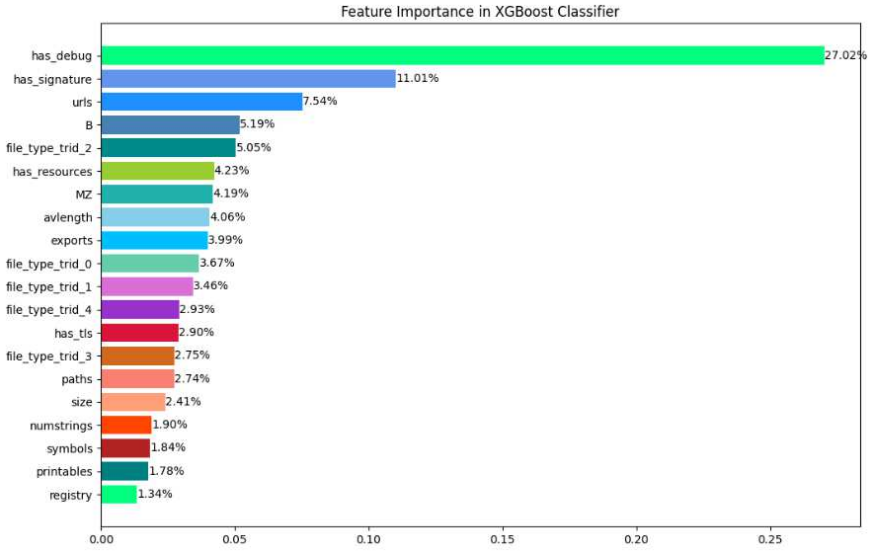
3.4


MLPClassifier
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

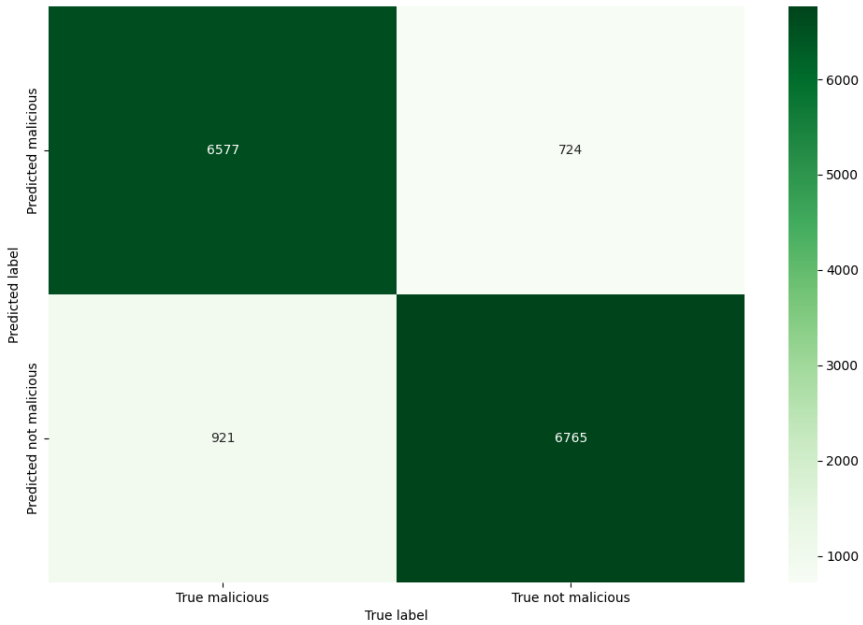
3.5


XGBClassifier
True Positive
False Positive
0.0
0.2
0.4
0.6
0.8
1.0

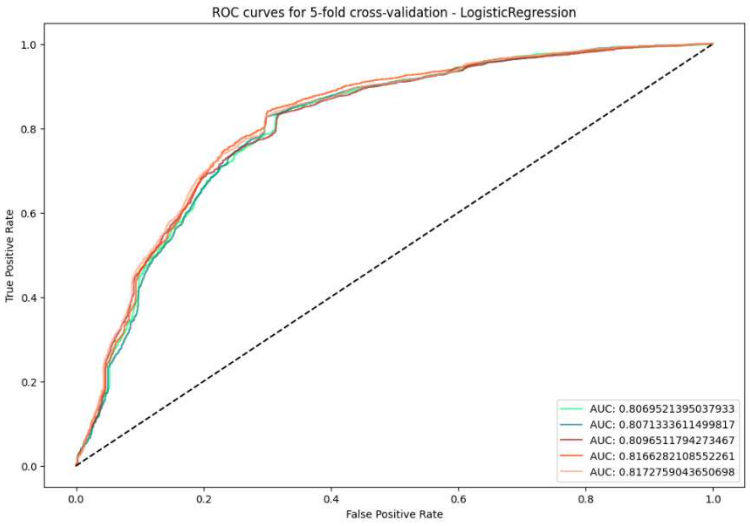
3.6


Feature Importance in XGBoost Classifier
has_debug 27.02%
has_signature 11.01%
urls 7.54%
B 5.19%
file_type_trid_2 5.05%
has_resources 4.23%
MZ 4.19%
avlength 4.06%
exports 3.99%
file_type_trid_0 3.67%
file_type_trid_1 3.46%
file_type_trid_4 2.93%
has_tls 2.90%
file_type_trid_3 2.75%
paths 2.74%
size 2.41%
numstrings 1.90%
symbols 1.84%
printables 1.78%
registry 1.34%
0.00
0.05
0.10
0.15
0.20
0.25

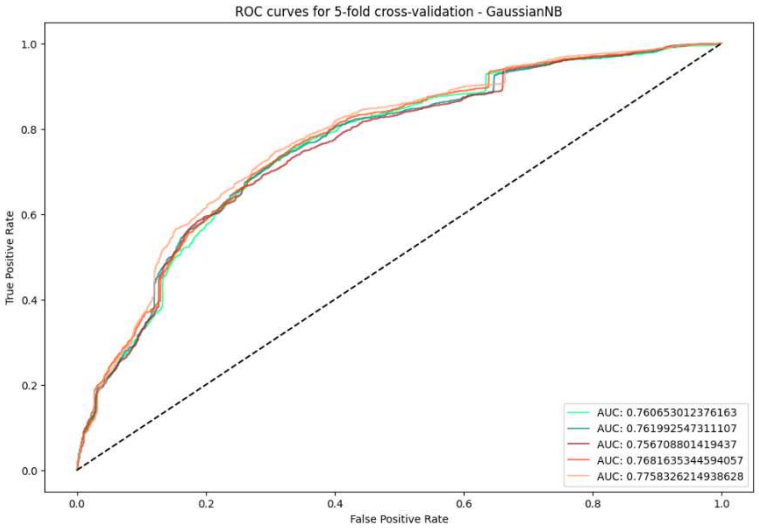
4.1


Predicted malicious
Predicted not malicious
Predicted label
6577
724
921
6765
True malicious
True not malicious
True label
6000
5000
4000
3000
2000
1000

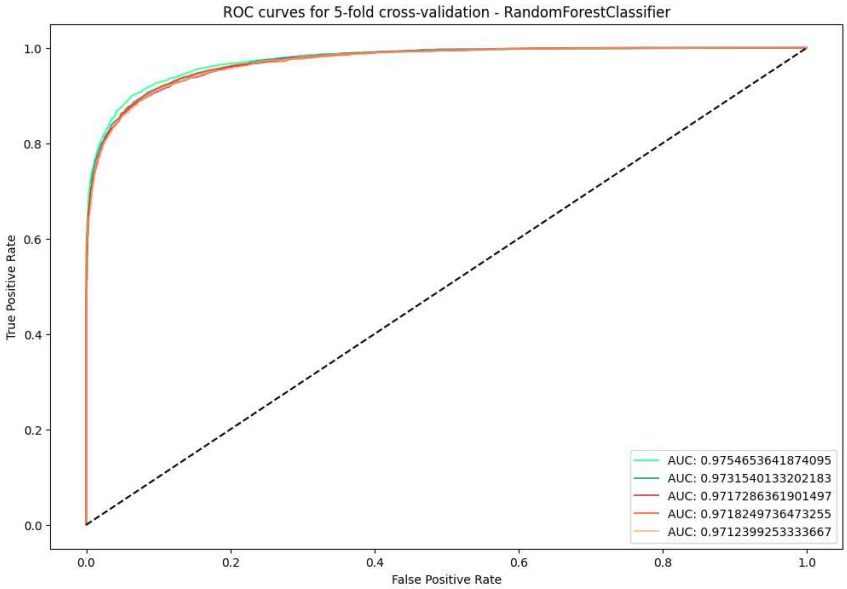
4.2


ROC curves for 5-fold cross-validation - LogisticRegression
True Positive Rate
False Positive Rate
AUC: 0.8069521395037933
AUC: 0.8071333611499817
AUC: 0.8096511794273467
AUC: 0.8166282108552261
AUC: 0.8172759043650698

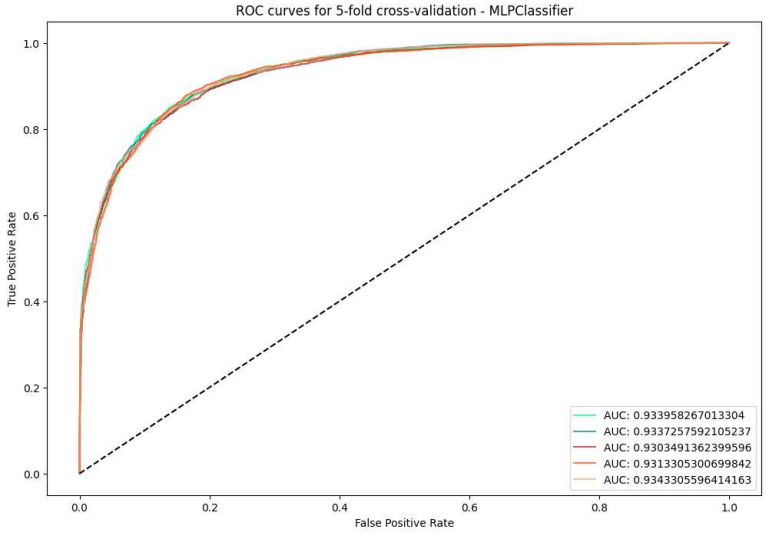
4.3


ROC curves for 5-fold cross-validation - GaussianNB
True Positive Rate
False Positive Rate
AUC: 0.760653012376163
AUC: 0.761992547311107
AUC: 0.756708801419437
AUC: 0.7681635344594057
AUC: 0.7758326214938628

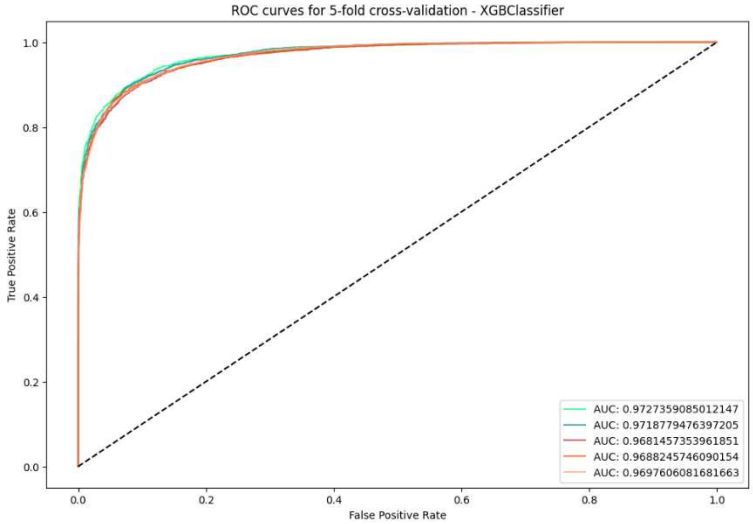
4.4


ROC curves for 5-fold cross-validation - RandomForestClassifier
True Positive Rate
False Positive Rate
AUC: 0.9754653641874095
AUC: 0.9731540133202183
AUC: 0.9717286361901497
AUC: 0.9718249736473255
AUC: 0.9712399253333667

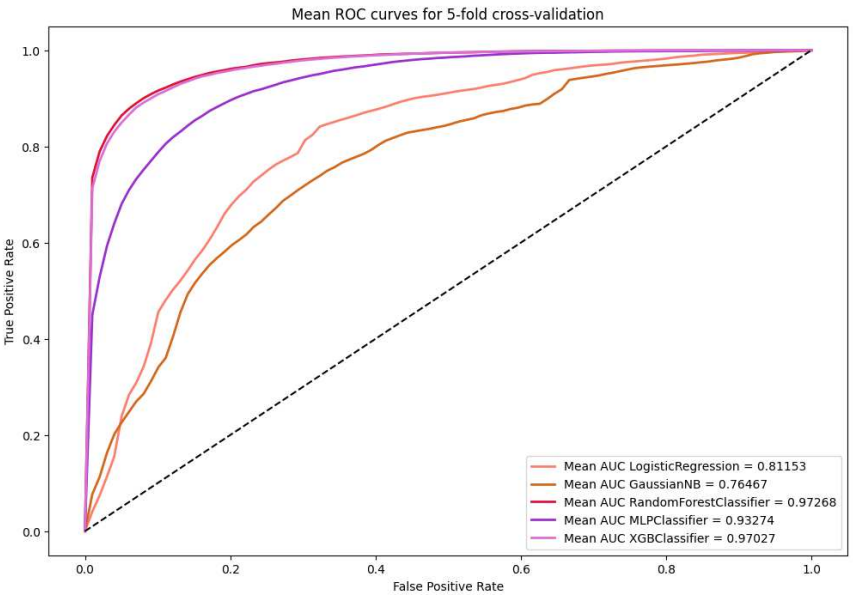
4.5


ROC curves for 5-fold cross-validation - MLPClassifier
True Positive Rate
False Positive Rate
AUC: 0.933958267013304
AUC: 0.9337257592105237
AUC: 0.9303491362399596
AUC: 0.9313305300699842
AUC: 0.9343305596414163

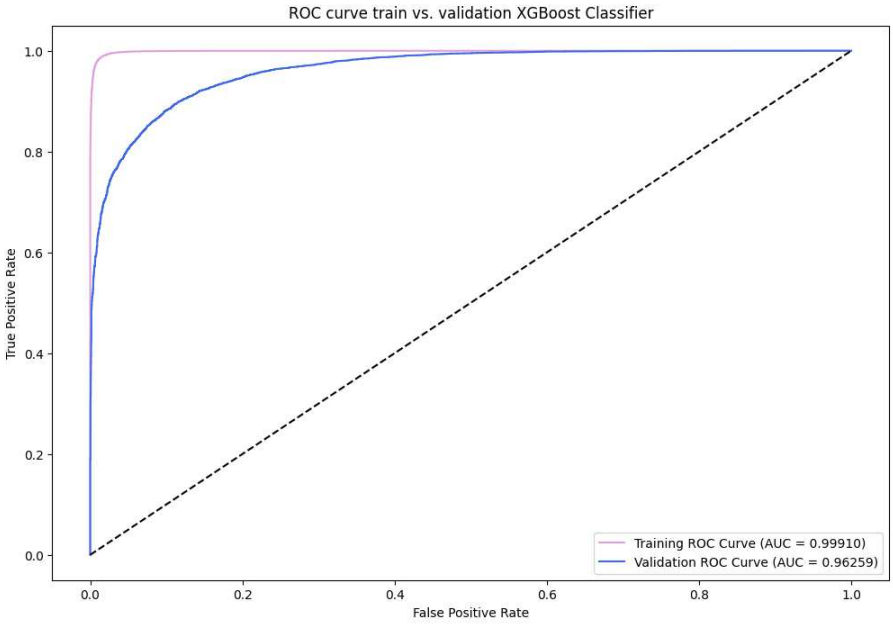
4.6


ROC curves for 5-fold cross-validation - XGBClassifier
True Positive Rate
False Positive Rate
AUC: 0.9727359085012147
AUC: 0.9718779476397205
AUC: 0.9681457353961851
AUC: 0.9688245746090154
AUC: 0.9697606081681663

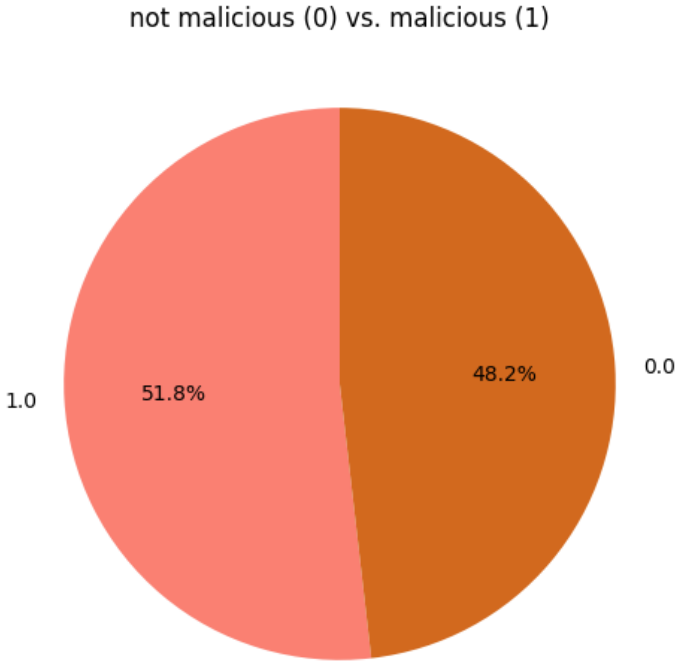
4.7


Mean ROC curves for 5-fold cross-validation
True Positive Rate
False Positive Rate
Mean AUC LogisticRegression = 0.81153
Mean AUC GaussianNB = 0.76467
Mean AUC RandomForestClassifier = 0.97268
Mean AUC MLPClassifier = 0.93274
Mean AUC XGBClassifier = 0.97027

4.8


ROC curve train vs. validation XGBoost Classifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Training ROC Curve (AUC = 0.99910)
Validation ROC Curve (AUC = 0.96259)

5.1


not malicious (0) vs. malicious (1)
1.0
51.8%
48.2%
0.0

6.1

XGBoost (Extreme Gradient Boosting) Classifier

Boosting is an ensemble method, meaning it's a way of combining predictions from several models into one.
Gradient boosting is a specific type of boosting, called like that because it minimizes the loss function using a gradient descent algorithm.

XGBoost is a gradient boosting algorithm that uses decision trees as its "weak" predictors. Beyond that, its implementation was specifically engineered for optimal performance and speed.

To understand how XGBoost Classifier works, let's break down its key components and the underlying principles:

- Loss Function: XGBoost Classifier uses a loss function to quantify the difference between the predicted and actual labels. The goal is to minimize the loss function by iteratively improving the model.
- Gradient Boosting: XGBoost Classifier utilizes the gradient boosting framework. It builds an ensemble of weak models in a sequential manner, where each subsequent model corrects the mistakes made by the previous models. This iterative process improves the ensemble's predictive power.
- Decision trees as base learners: XGBoost Classifier employs decision trees as the base learners. These trees are shallow, usually with a small depth, to prevent overfitting. Each decision tree predicts the class labels based on a subset of features. The construction of decision trees is done using gradient boosting, which involves fitting the tree to the negative gradient of the loss function.
- Gradient Calculation: During the training process, XGBoost Classifier calculates the gradient (partial derivative) of the loss function with respect to the predictions made by the ensemble so far. The gradient reflects the direction and magnitude of the improvement needed to minimize the loss function. This calculation is performed for each training instance.
- Weighted Residuals: After calculating the gradient, XGBoost Classifier converts it into weighted residuals. The weights are determined based on the loss function and the predicted probabilities or scores from the previous iteration. These weighted residuals represent the "errors" made by the ensemble, and the subsequent weak model (decision tree) aims to correct these errors.
- Tree Construction: XGBoost Classifier builds decision trees sequentially, with each tree constructed to minimize the weighted residuals. It uses a technique called "regularized learning" to control the complexity of the trees and prevent overfitting. The regularization terms, such as L1 and L2 regularization, are added to the loss function during tree construction.
- Ensemble Combination: The individual decision trees are added to the ensemble one by one, with each tree contributing to the final prediction. The predictions from all the trees

are combined through weighted averaging, where the weights are determined based on the performance of the trees on the training data.

The values are typically calculated using the log of odds and probabilities. The output of the tree becomes the new residual for the dataset, which is used to construct another tree. This process is repeated until the residuals stop reducing or for a specified number of times. Each subsequent tree learns from the previous trees and is not assigned equal weight, unlike how Random Forest works.

Historically, XGBoost has performed quite well for structured tabular data.