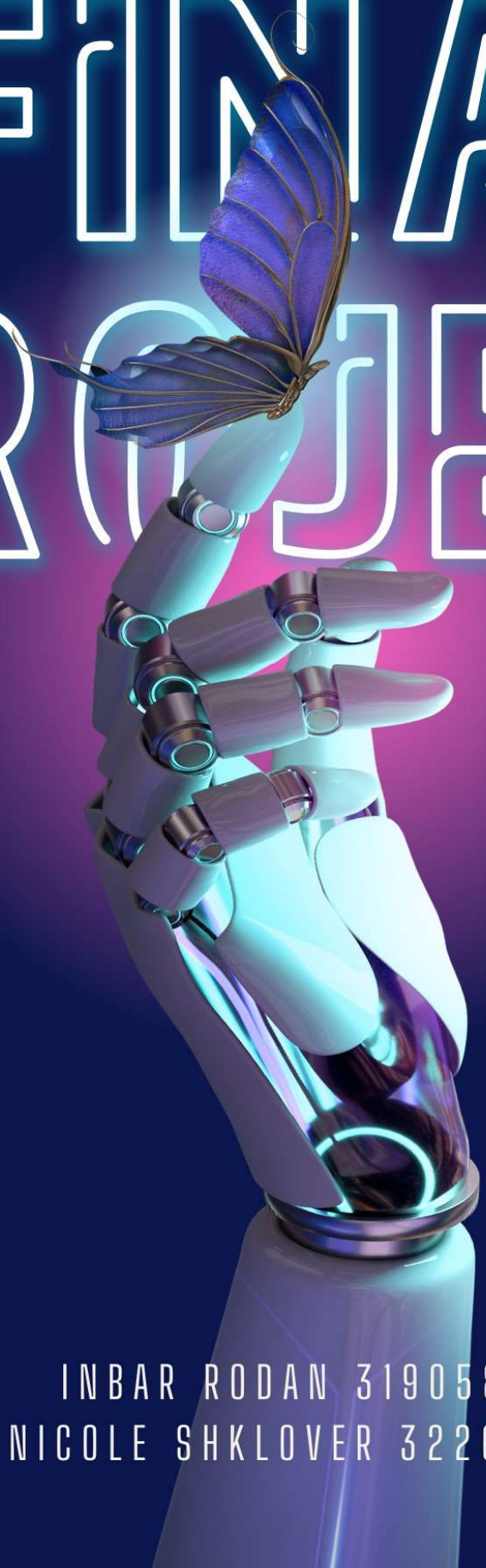
MACHINE LEARNING

FINAL PROJECT



INBAR RODAN 319058251
NICOLE SHKLOVER 322605601

Part 1 - Exploring the data

(All the following appears in appendix 1)

At first, we wanted to look at the characteristics of the features in the train dataset (mean, std, min/max values, number of missing values etc.). Then we wanted to see the distribution of each feature using plots. Some things we noticed are:

A is approximately normally distributed and size, vsize, numstrings and printables are approximately normally distributed with log scale applied to them.

One more thing we wanted to check is the number of samples at the edges of the plots.

Then, we checked the correlation between the features.

We saw that the correlation between numstrings and size is very high (0.9). later we'll disduss removing it.

We checked how many missing values there are in each:

Number of missing values in size feature: 0, Number of missing values in numstrings feature: 2718. We deleted numstrings because size has zero missing values and numstrings has a lot more missing values.

Part 2 - Pre-processing

we created two functions called preprocessing and preprocessing2, that receives train and test/validation datasets and returns the preprocessed datasets. The functions contain the following:

Outliers – only on train data:

Looking at the visualization of the data, for each feature we looked at the samples that their values are rare, these are samples that most likely do not represent the distribution of the data and we don't want our models to learn from them.

percentage of label 1 in the suspected outliers in:

imports: 1.6666666666666667, paths: 14.583333333333334, MZ: 12.76595744680851, exports: 1.8867924528301887, avlength: 98.59154929577466, urls: 13.793103448275861, B: 84.4155844155844

We can see that we should not delete these outliers in avlength and B, because most of their labels are 1 what can imply a connection between the rare values in those features and the labels. (Appendix 2)

In addition, we saw in the exploration that some of the features are normally distributed when we applied log scale on them or as is.

We used IQR to find the outliers in those features (assumed normal distribution). (Explanation about IQR is in appendix 2)

These outliers can be removed from the dataset to obtain a more accurate representation of the data. However, we know that when removing outliers, it can be problematic because they may contain valuable information. We chose to remove them because they represent a small percentage of the samples from the dataset. Hence there is a lower chance that they all are valuable.

Missing values – on train and validation (using train's mean and most frequent value):

We used the isnull function to detect and count missing values for each sample and for each feature.

We chose to fill the missing values in the numerical features with the column's mean because this way helps to maintain the overall integrity and statistical properties of the data.

In the categorical features, we chose to fill the missing values with the most frequent value because it helps to maintain the majority class distribution and preserves the integrity of the categorical data.

Normalization - on train and validation (using train's minmax scaler):

The data is not normalized because there are values that are not in the range of 0 to 1.

We think that the data should be normalized because normalization is important for several reasons:

1. Normalizing the data brings all the features and samples to a common scale, ensuring that they are comparable and can be analyzed together.

2. Values with larger size can dominate the analysis and overshadow values with smaller size. Normalizing prevents this overshadowing.

Overall, normalizing the data leads to more accurate insights from the data.

That's why we normalized them by using MinMaxScaler for numerical data. (Explanation about MinMaxScaler in appendix 3).

Dealing with categorical data - on train and validation:

We used OneHotEncoder to deal with categorical data. (Explanation about OneHotEncoder in appendix 4)

Because we chose this type of dealing that increases the number of features in the data (with dummy features), we decided to remove the features that add "too much" dimensions (sha256 and file_type_trid).

Dimensionality - on train and validation:
As we mentioned earlier, because we used the OneHotEncoder method, the number of columns increased.
This can be a problem for several reasons:
1. More columns means more dimensions --> more complex model --> the variance increases and the bias decreases (bias variance tradeoff) --> can lead to overfitting.
2. When we increase the number of features in a dataset, we also increase the potential for noise that weakens the meaningful signal in the data.
3. Curse of dimensionality - increasing the number of features means that the data might become sparser, and data points become more distant. This makes it harder to recognize patterns or relations.

There are several ways for recognizing when the dimensionality is too big:
1. Using dimensionality reduction techniques. In PCA for example, we can see the amount of variance in the data explained by each principal component. If a few principal components capture most of the variance, it suggests that the remaining dimensions might not contribute significantly and could potentially be dropped.
2. Techniques like feature importance or feature selection can help identify the most relevant features. If many features have low importance to the model's performance, it suggests that the dimensionality may be too high.
It is important to say that there is no specific threshold that tells when dimensionality is getting too big.

Dimensionality reduction - on train and validation:
We tried two methods of reduction (manual feature selection and PCA):
We performed manual feature selection for features we thought could be dropped:
1. There were features that most of their values were zeros (symbols, registry) meaning there is a low chance they help to predict the labels. Symbols contains 94% zeros and registry contains 91%. We chose to remove only symbols after checking the AUC score with and without it, and when checking registry in a similar way we saw it is better to keep it.
2. We know that the sha256 feature represents a unique value for each sample, meaning there's a very low chance it helps to predict the labels. Moreover, when using OneHotEncoder with this type of categorical feature it increases dramatically the dimensionality. We removed it.
3. We saw that there is high correlation between size and numstrings and thought that if we'll delete one of them it will make the model's prediction better because then the model will learn less noise. Size doesn't have missing values at all but numstrings does. So, we decided to remove numstrings.
4. In file_type_trid there are 89 categories (as you can see in the exploration part). Turning each category into a dummy feature is problematic because it increases the dimensions dramatically. So, we decided to remove it.
5. After all the preprocessing, we checked the correlation between the "new" features. We noticed that the correlation between size and vsize is 0.81 (high) and the correlation between size and printables is 0.84 (high). The correlation between vsize and printables is 0.66(not that high). We think that the correlation between size and vsize is not surprising because of the meaning of them. After checking the AUC score with and without vsize we decided to remove vsize to improve the score.
After checking the AUC score with and without printables we decided not to remove it because removing it decreases the score. It doesn't surprise us because the connection between them is not clear to us so we can't assume what exactly made this change in correlations, unlike size and vsize.
We performed PCA method on the data preprocessed data to check if the dimensionality is too big even after the manual feature reduction. We wanted the reduced data to keep 99% of the variance. After checking the AUC score with and without the PCA reduction, we decided to not use this method because it lowers the score (ROC curve and AUC score in appendix 6).

Mathematical transformation on features - on train and validation:
As we mentioned earlier, we applied log scale on some features when we used IQR method to detect outliers. We applied log scale to the features permanently in the train and test/validation data (removed the outliers only on the train data).

Part 3 - Modeling

In this section we will discuss the hyperparameters we chose to change (We used grid search to search for the best hyper parameters that will give us the best model performance – appears as a comment in the modeling part of each model in the notebook. We didn't run the code with grid search to save time…).

The models we chose are:
KNN Model: In this model the n_neighbors hyperparameter represents the number of nearest neighbors considered by the KNN algorithm. By increasing the number of neighbors, the model's complexity decreases, potentially reducing variance and increasing bias. By decreasing the number of neighbors, the model's complexity increases, potentially reducing bias and increasing variance making it more likely to overfit.

Logistic Regression Model: In this model the hyperparameters are C (determines the amount of regularization applied to the model) and penalty (type of regularization: l1 or l2). Higher values of C mean less regularization, so the model fits the training data more closely. This can lead to low bias and high variance, making the model more likely to overfit. Lower values of C mean more regularization, so the model is simpler with higher bias and lower variance, which can help prevent overfitting.
L1 regularization (Lasso) tends to produce sparse models by making some coefficients to zero, which can reduce variance but potentially increase the bias. L2 regularization (Ridge) balances between bias and variance by finding the best coefficients.

Adaptive Boosting (Ada Boost) Model: In this model the hyperparameters are n_estimators and learning_rate. n_estimators represents the number of weak learners (base estimators) to be used in the AdaBoost ensemble. By increasing the number of estimators, the model becomes more complex, decreasing the bias. If the model becomes too complex, it may start to memorize noise or outliers in the train dataset and increase the variance. This can lead to overfitting. The learning rate affects how much each weak learner contributes to the final prediction. A smaller learning rate decreases the weight of each weak learner, making the model less complex. This can increase bias and decrease variance as the model relies less on individual weak learners and may struggle to capture complex patterns in the data. A larger learning rate assigns more weight to each weak learner, making the model more sensitive to individual predictions, increase the variance and decrease the bias. This can lead to overfitting.

Multi-Layer Perceptron (MLP): In this model the hyperparameters are hidden_layer_sizes, max_iter and alpha. hidden_layer_sizes determines the number of neurons in each hidden layer of the MLP. Increasing the number of neurons per layer increases the model's complexity, allowing it to learn more complex patterns in the data. This can lead to lower bias because the model becomes more capable of fitting more relationships in the train data. However, increasing the number of neurons also increases the variance of the predictions what can result the model becoming too specific to the training data increasing the risk of overfitting. max_iter determines the maximum number of iterations for training the MLP. It controls the amount of time the model learns from the data. Increasing max_iter increases the model's training time, so it has more opportunities to capture complex relationships in the data what leads to reducing bias and increasing variance. If max_iter is set too high, the model may continue to fit the training data too closely, leading to too high variance and overfitting. Alpha controls the amount of regularization applied to the weights of the MLP. Regularization helps prevent overfitting by penalizing large weights. Increasing alpha increases the amount of the regularization, which can lead to higher bias due to reducing the weights making it more resistant to noise in the data. However, if alpha is set too high, the model may become too constrained and have difficulty capturing complex patterns, decreasing the variance too much, resulting in underfitting and higher bias. Reducing alpha decreases the amount of the regularization, so the model becomes sensitive to noise and outliers in the data, increasing the variance.

Part 4 – Evaluating the models
Kfold cross validation: We created a function called kfold_cross_validation that splits the original (not pre-processed) data to train and validation 5 times (5 folds) and each time performs the preprocessing functions on the current train and validation.
For each model of the above models, it fits the model in every fold, predicts the validation labels and calculates the AUC score. For each model the function also calculates the mean of the AUC scores (of each fold) and plots the AUC scores of each fold. The best AUC score was for the AdaBoost model (average AUC: 0.92). (The plots and scores are in appendix 6). **We chose Ada-Boost as the prime model for the next steps.**

Confusion matrix: We created a function called confusion_matrix_plot that plots the confusion matrix for the Ada Boost model. Then we split the original train data to train and validation datasets, preprocessed them, fitted the model on the (new) train, predicted on the validation and called our function to plot the confusion matrix for it.
In addition, we calculated the accuracy, precision and recall scores to get a better perspective of the model's performance (The plot, scores and explanation of confusion matrix are in appendix 7).

In the confusion matrix, the cells represent classifications made by the Ada-Boost model based on the true and predicted labels of the validation dataset.
In our case, the number of files that are malicious, as the prediction of the model, is 5272. The number of files that are not malicious, but the prediction of the model is that they are malicious, is 1124. The number of files that are malicious, but the prediction of the model is that they are not malicious is 713. The number of files that are not malicious, as the prediction of the model, is 4891.
From the above numbers we can understand that our model correctly classifies most of the data, but still has mistakes.

Performances gap: We created a function called train_vs_validation that predicts and calculates the AUC score for the train and validation. (Performance gap and scores are in appendix 8).

Overfitting: Overfitting can be detected when there is a significant difference between the AUC scores of the training and validation\test datasets (especially when the train is much higher). Another way to identify overfitting is by training the model on different data each time and observing the range of performances (as we did in K-fold Cross Validation), when a wide range suggests overfitting. By customizing the K-Fold Cross Validation process to the specific data, we can estimate if the model is truly overfitted. When we did K-fold cross validation, the range of performances was narrow (similar for each fold), indicating no overfitting. But because the train AUC score is higher than the validation's this might imply a bit of overfitting. Although the performance gap is not that high, so we think the overfitting is not dramatic.
To simulate real-world scenarios, the preprocessing steps are performed on the training data, and the validation data's preprocessing aligns with the values derived from the training data, without incorporating any learning from the validation set. This approach ensures that decision making is based solely on the training data, considering the potential issues of overfitting. Additionally, to try preventing overfitting and reducing the learning of noise, we used measures such as dimensionality reduction, handling categorical features and removing suspected outliers in the preprocessing of the training data.

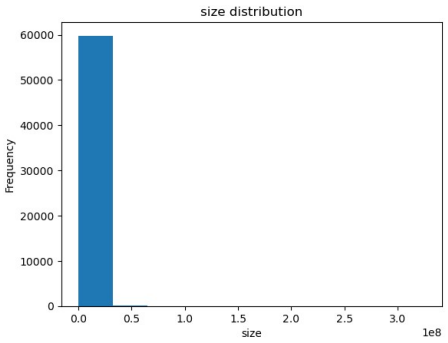
Feature importance: We created the feature_importance function, that plots every feature and its contribution to the model's performance (plot in appendix 9).
The top 3 important features are: file_type_prob_trid, size and imports.
file_type_prob_trid (1st place): The probability of the file being a certain file type is highly relevant in determining whether a file is malicious. It makes sense that the type of the file is connected to the indication of a malicious file. For example, malicious files might appear more as a certain file type.
size (2nd place): The size of the file on disk can provide valuable insights into its potential being malicious. Large files may contain malicious code, while very small files may indicate compressed malware. For example, we can try to identify files that are outside of the expected size range for a given file type.
imports (3rd place):
The number of imported functions. We assume that malicious files can contain some functions not found in not malicious files or contain more/less functions.

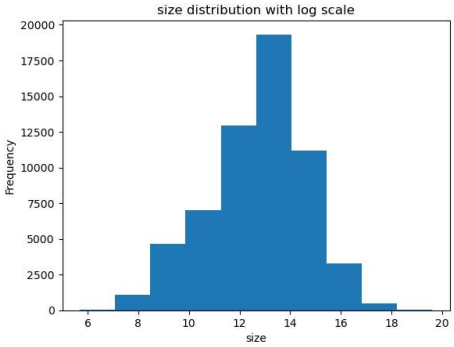
Part 5 – Prediction
We created the prediction function, that fits the model on the preprocessed train data and predicts the probability of the test data being malicious. In addition, it exports the predictions to a csv file called "results_30.csv" where each row (file) has its name (sha256) and the probability.
When looking at the file, we noticed that all the predictions are close to 0.5. Unfortunately, we don't know why it's happening. We think it might be because the model's bias is too high what makes the predictions close to each other.
We wanted to check if the train's probabilities are close and looks similar to the test's, so we checked the train's prediction (appears as a comment in the notebook after the test's prediction) and it is the same. Considering the bias variance tradeoff, we tried to change the hyper parameters of the AdaBoost model, but this is the best validation AUC score we achieved, so we decided to leave it as is because our main purpose is high AUC score on the test.

At the end of the notebook, we created a pipeline function that contains all the above, only with the AdaBoost model.

Appendices (נספחים):

1)



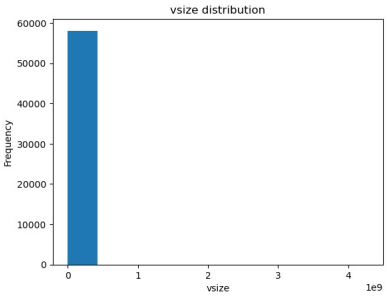


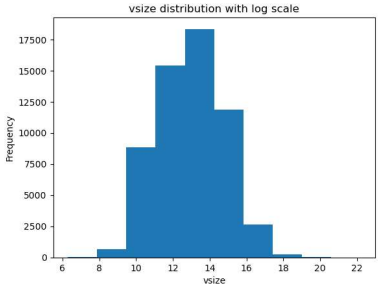

```
#Find out about missing values
train_data_copy.isnull().sum()
```

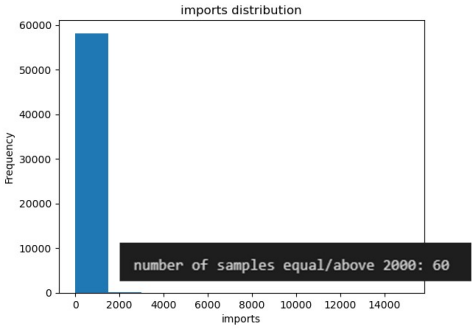
✓ 0.0s

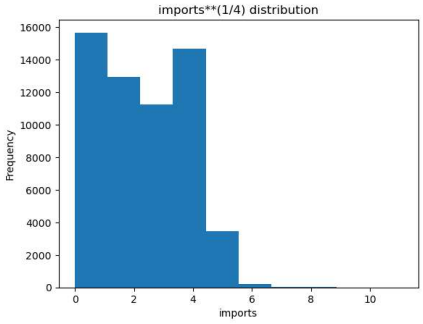
sha256	0
size	0
vsize	1935
imports	1739
exports	2093
has_debug	2927
has_relocations	3324
has_resources	1961
has_signature	1937
has_tls	2898
symbols	2656
numstrings	2718
paths	3660
urls	2349
registry	2525
MZ	3089
printables	2739
avlength	2757
file_type_trid	0
file_type_prob_trid	0
A	3704
B	3751
C	2051
label	0

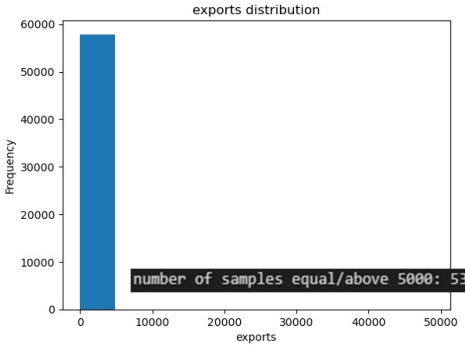
dtype: int64

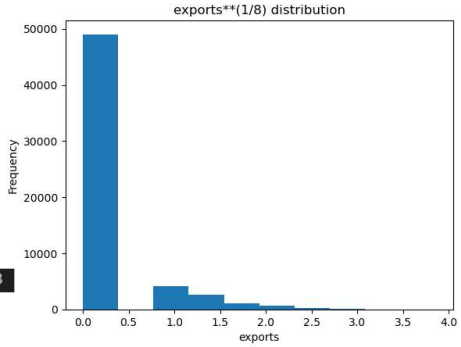


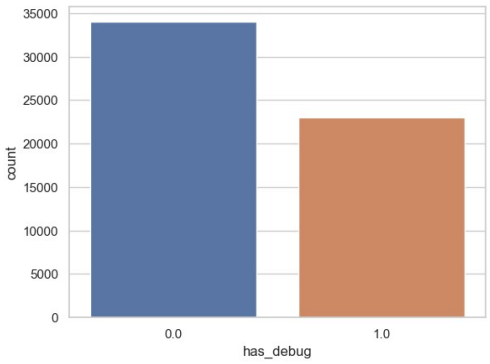


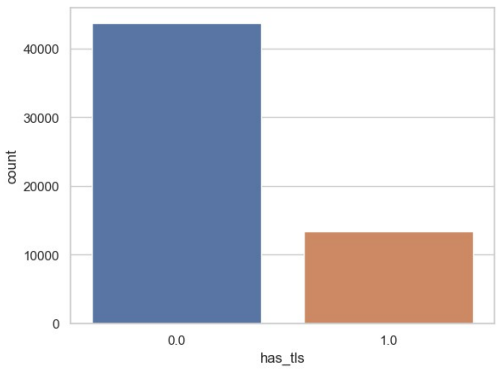


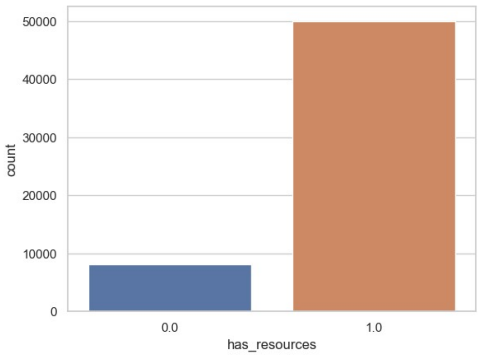


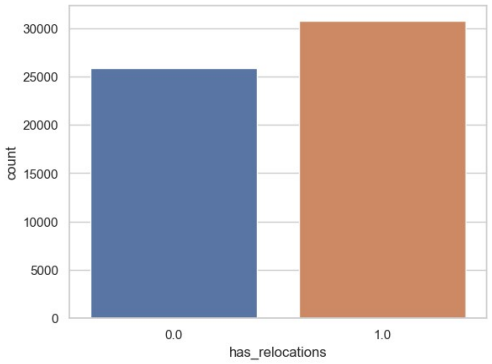


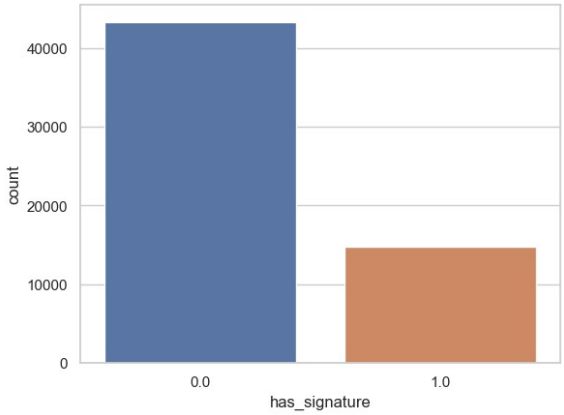


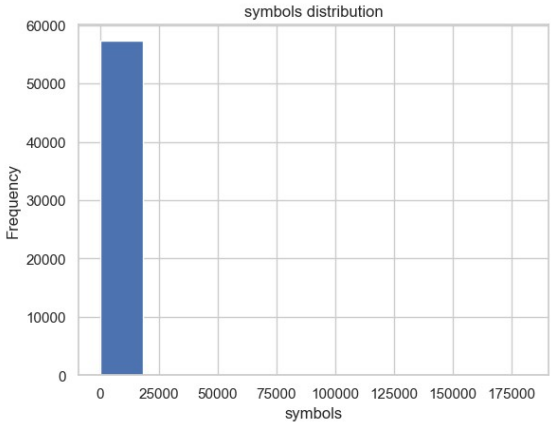




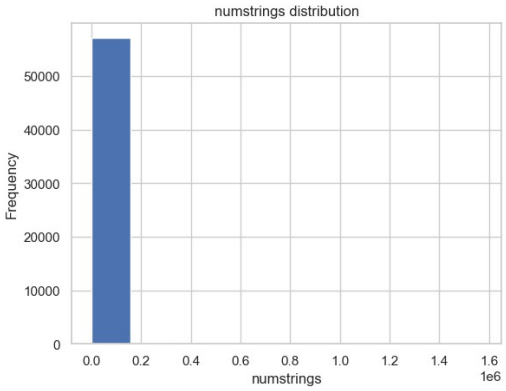


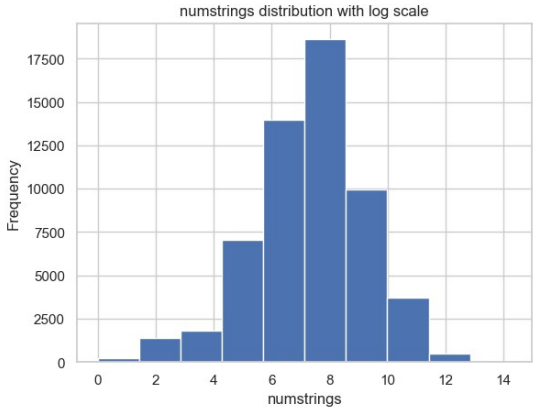


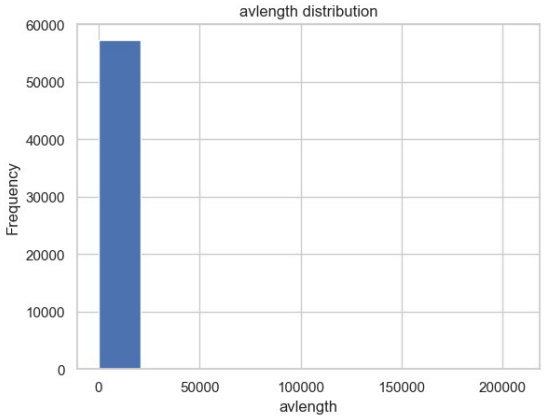




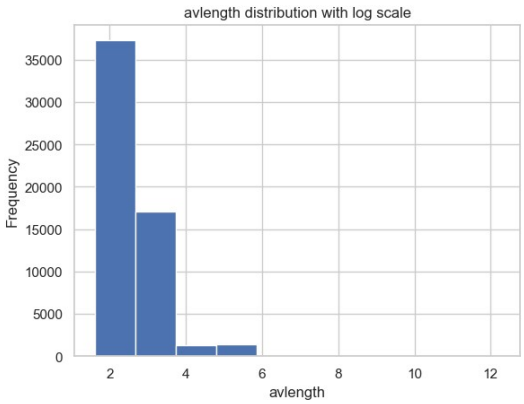

```
Value count for the symbols:

0.0         56767
3.0            63
4.0             9
3644.0          6
1080.0          4
             ...
109.0           1
13118.0         1
191.0           1
26.0            1
181660.0        1
Name: symbols, Length: 415, dtype: int64
```

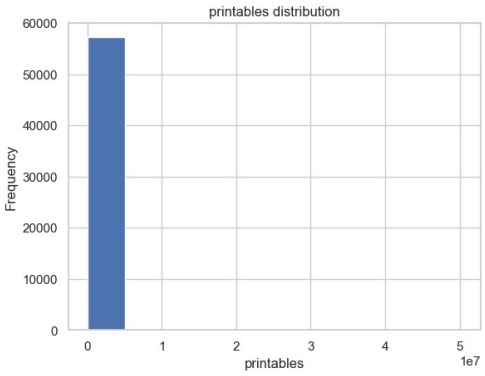


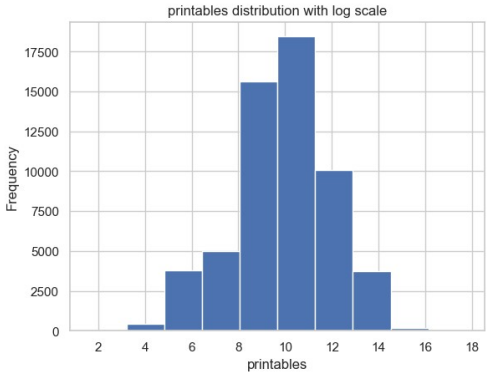


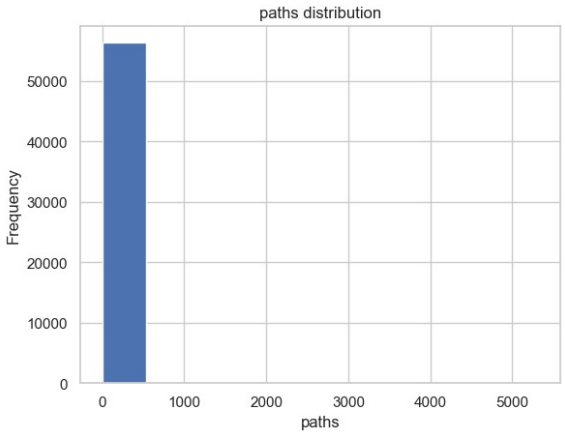



```
Value count for the avlength:

7.447368      1447
11.714286      340
240.616879     338
14.227368      283
206.887923     159
              ...
14.708924        1
21.977233        1
5.715961         1
12.929775        1
14.714286        1
Name: avlength, Length: 44462, dtype: int64
number of samples equal/above 1000: 71
```

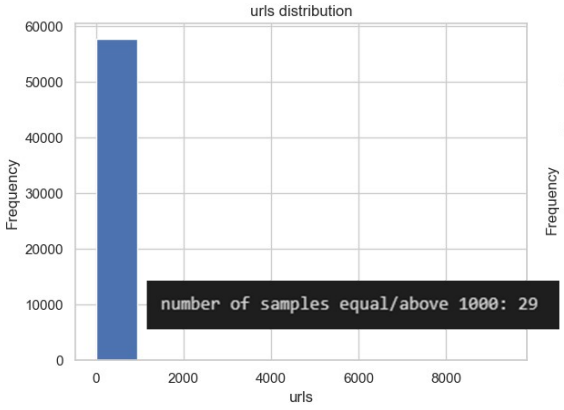


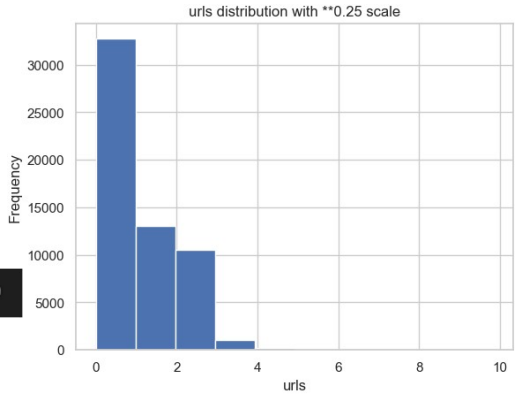


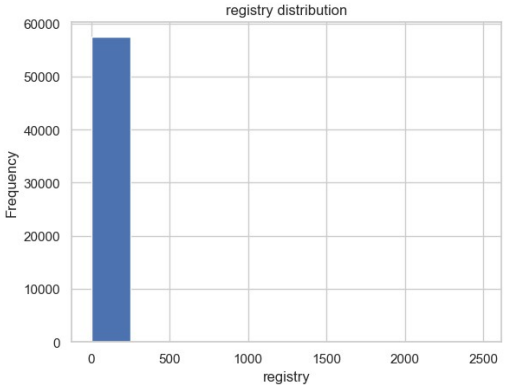


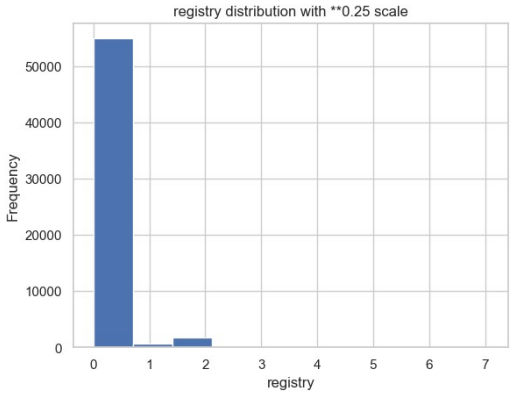



```
Value count for the paths:

0.0       40792
1.0        7653
2.0        4130
3.0        1089
4.0         705
           ...
266.0         1
185.0         1
164.0         1
815.0         1
4146.0        1
Name: paths, Length: 168, dtype: int64
number of samples equal/above 200: 48
```

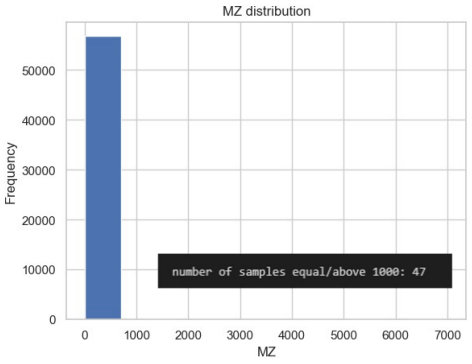


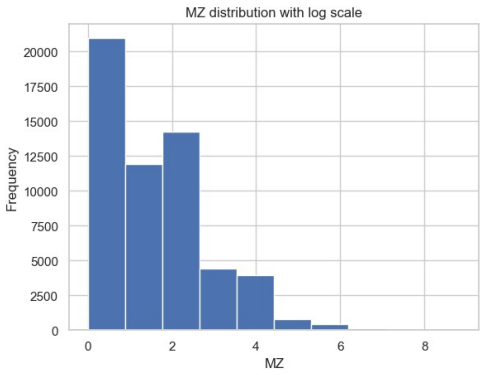


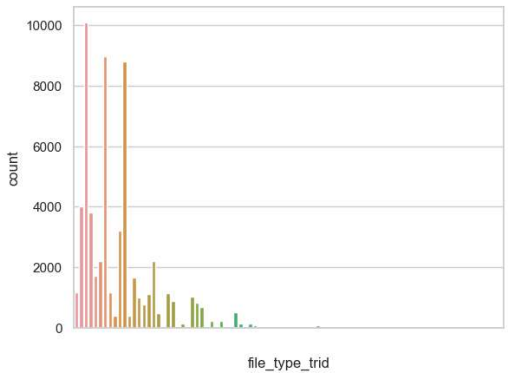




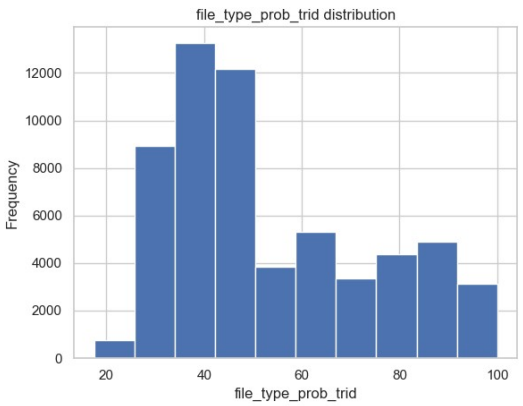

```
Value count for the registry:

0.0       54948
14.0        483
7.0         445
1.0         326
4.0         255
5.0         254
2.0         207
3.0         108
6.0          63
8.0          52
9.0          52
12.0         47
10.0         46
16.0         19
13.0         18
11.0         17
15.0         14
21.0         13
20.0         12
27.0         10
25.0          6
18.0          6
63.0          5
...
41.0          1
60.0          1
46.0          1
Name: registry, dtype: int64
```

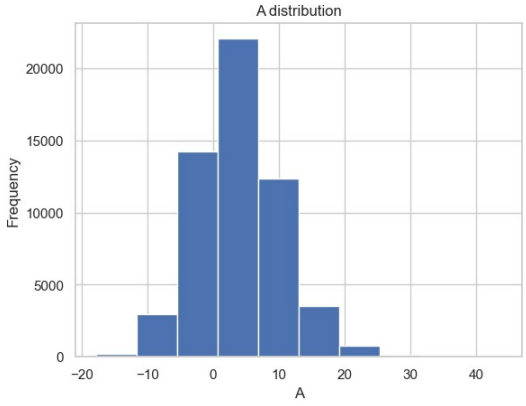


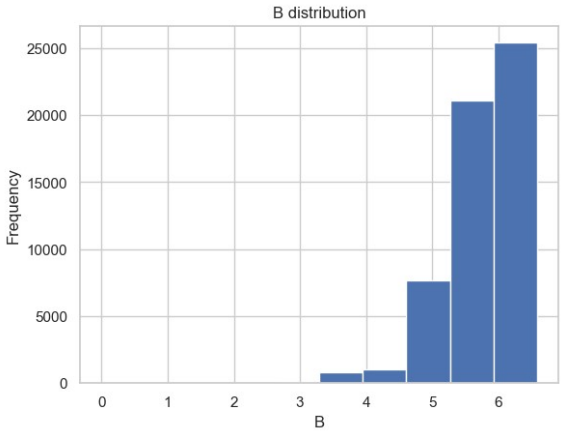


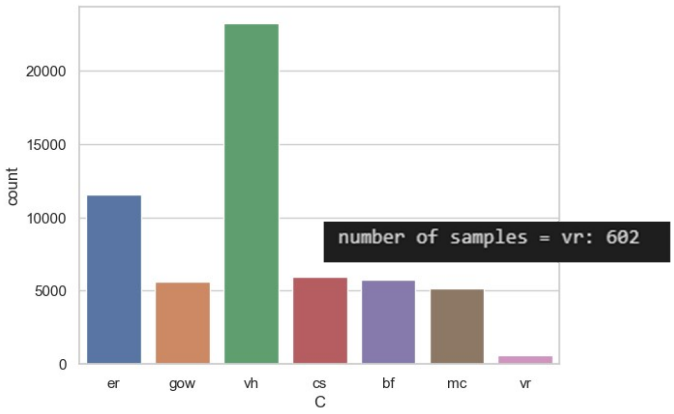


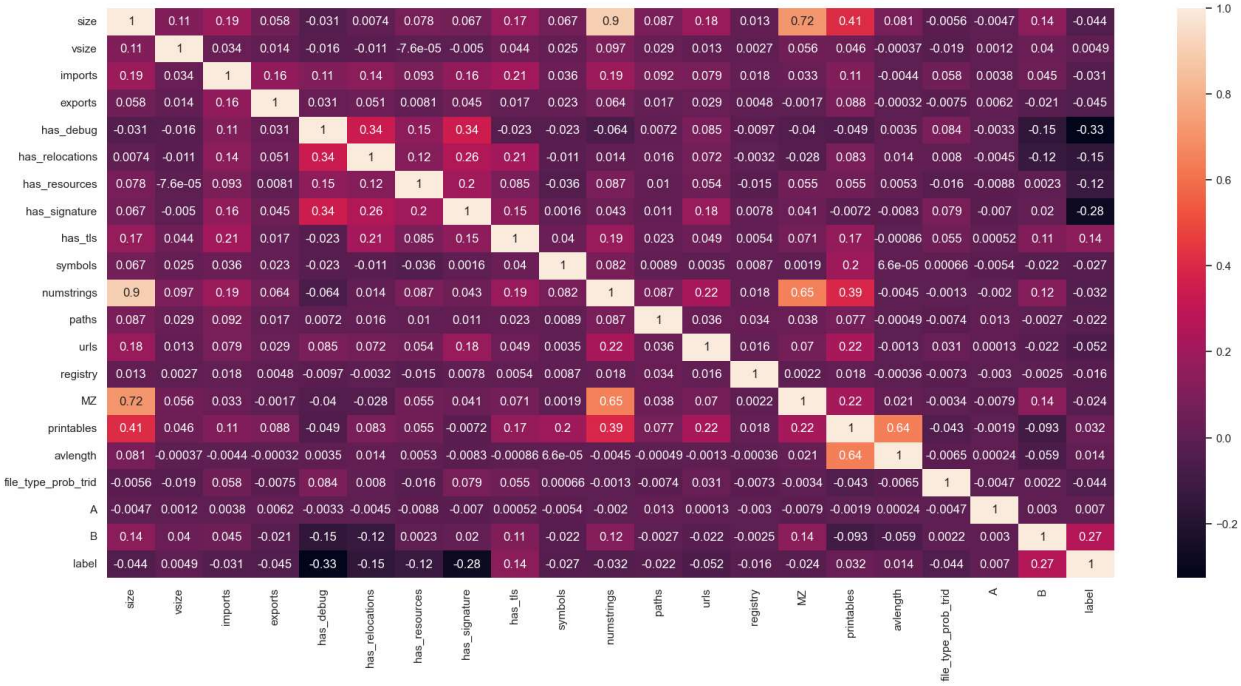

```
Value count for the file_type_trid:

Win64 Executable (generic)                  10085
Win32 Executable MS Visual C++ (generic)     8967
Win32 Executable (generic)                   8781
Win32 Dynamic Link Library (generic)         4010
Generic CIL Executable (.NET, Mono, etc.)    3804
                                             ...
MS Flight Simulator Gauge                       1
Photoshop filter plug-in                        1
VirtualDub Filter Plug-in                       1
GIMP Plugin (Win)                               1
WinArchiver Mountable compressed Archive        1
Name: file_type_trid, Length: 89, dtype: int64

number of unique samples:  12

number of categories:  (89,)
```


file_type_prob_trid distribution
Frequency
file_type_prob_trid


A distribution
Frequency
A


B distribution
Frequency
B


count
number of samples = vr: 602
er
gow
vh
cs
bf
mc
vr
C


size 1 0.11 0.19 0.058 -0.031 0.0074 0.078 0.067 0.17 0.067 0.9 0.087 0.18 0.013 0.72 0.41 0.081 -0.0056 -0.0047 0.14 -0.044
vsize 0.11 1 0.034 0.014 -0.016 -0.011 -7.6e-05 -0.005 0.044 0.025 0.097 0.029 0.013 0.0027 0.056 0.046 -0.00037 -0.019 0.0012 0.04 0.0049
imports 0.19 0.034 1 0.16 0.11 0.14 0.093 0.16 0.21 0.036 0.19 0.092 0.079 0.018 0.033 0.11 -0.0044 0.058 0.0038 0.045 -0.031
exports 0.058 0.014 0.16 1 0.031 0.051 0.0081 0.045 0.017 0.023 0.064 0.017 0.029 0.0048 -0.0017 0.088 -0.00032 -0.0075 0.0062 -0.021 -0.045
has_debug -0.031 -0.016 0.11 0.031 1 0.34 0.15 0.34 -0.023 -0.023 -0.064 0.0072 0.085 -0.0097 -0.04 -0.049 0.0035 0.084 -0.0033 -0.15 -0.33
has_relocations 0.0074 -0.011 0.14 0.051 0.34 1 0.12 0.26 0.21 -0.011 0.014 0.016 0.072 -0.0032 -0.028 0.083 0.014 0.008 -0.0045 -0.12 -0.15
has_resources 0.078 -7.6e-05 0.093 0.0081 0.15 0.12 1 0.2 0.085 -0.036 0.087 0.01 0.054 -0.015 0.055 0.055 0.0053 -0.016 -0.0088 0.0023 -0.12
has_signature 0.067 -0.005 0.16 0.045 0.34 0.26 0.2 1 0.15 0.0016 0.043 0.011 0.18 0.0078 0.041 -0.0072 -0.0083 0.079 -0.007 0.02 -0.28
has_tls 0.17 0.044 0.21 0.017 -0.023 0.21 0.085 0.15 1 0.04 0.19 0.023 0.049 0.0054 0.071 0.17 -0.00086 0.055 0.00052 0.11 0.14
symbols 0.067 0.025 0.036 0.023 -0.023 -0.011 -0.036 0.0016 0.04 1 0.082 0.0089 0.0035 0.0087 0.0019 0.2 6.6e-05 0.00066 -0.0054 -0.022 -0.027
numstrings 0.9 0.097 0.19 0.064 -0.064 0.014 0.087 0.043 0.19 0.082 1 0.087 0.22 0.018 0.65 0.39 -0.0045 -0.0013 -0.002 0.12 -0.032
paths 0.087 0.029 0.092 0.017 0.0072 0.016 0.01 0.011 0.023 0.0089 0.087 1 0.036 0.034 0.038 0.077 -0.00049 -0.0074 0.013 -0.0027 -0.022
urls 0.18 0.013 0.079 0.029 0.085 0.072 0.054 0.18 0.049 0.0035 0.22 0.036 1 0.016 0.07 0.22 -0.0013 0.031 0.00013 -0.022 -0.052
registry 0.013 0.0027 0.018 0.0048 -0.0097 -0.0032 -0.015 0.0078 0.0054 0.0087 0.018 0.034 0.016 1 0.0022 0.018 -0.00036 -0.0073 -0.003 -0.0025 -0.016
MZ 0.72 0.056 0.033 -0.0017 -0.04 -0.028 0.055 0.041 0.071 0.0019 0.65 0.038 0.07 0.0022 1 0.22 0.021 -0.0034 -0.0079 0.14 -0.024
printables 0.41 0.046 0.11 0.088 -0.049 0.083 0.055 -0.0072 0.17 0.2 0.39 0.077 0.22 0.018 0.22 1 0.64 -0.043 -0.0019 -0.093 0.032
avlength 0.081 -0.00037 -0.0044 -0.00032 0.0035 0.014 0.0053 -0.0083 -0.00086 6.6e-05 -0.0045 -0.00049 -0.0013 -0.00036 0.021 0.64 1 -0.0065 0.00024 -0.059 0.014
file_type_prob_trid -0.0056 -0.019 0.058 -0.0075 0.084 0.008 -0.016 0.079 0.055 0.00066 -0.0013 -0.0074 0.031 -0.0073 -0.0034 -0.043 -0.0065 1 -0.0047 0.0022 -0.044
A -0.0047 0.0012 0.0038 0.0062 -0.0033 -0.0045 -0.0088 -0.007 0.00052 -0.0054 -0.002 0.013 0.00013 -0.003 -0.0079 -0.0019 0.00024 -0.0047 1 0.003 0.007
B 0.14 0.04 0.045 -0.021 -0.15 -0.12 0.0023 0.02 0.11 -0.022 0.12 -0.0027 -0.022 -0.0025 0.14 -0.093 -0.059 0.0022 0.003 1 0.27
label -0.044 0.0049 -0.031 -0.045 -0.33 -0.15 -0.12 -0.28 0.14 -0.027 -0.032 -0.022 -0.052 -0.016 -0.024 0.032 0.014 -0.044 0.007 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

2)

```
check_labels(train_data_reduced,"imports",'>=', 2000)
check_labels(train_data_reduced,"paths",'>=', 200)
check_labels(train_data_reduced,"MZ",'>=', 1000)
check_labels(train_data_reduced,"exports",'>=', 5000)
check_labels(train_data_reduced,"avlength",'>=', 1000)
check_labels(train_data_reduced,"urls",'>=', 1000)
check_labels(train_data_reduced,"B",'<', 2)
```

✓ 0.0s

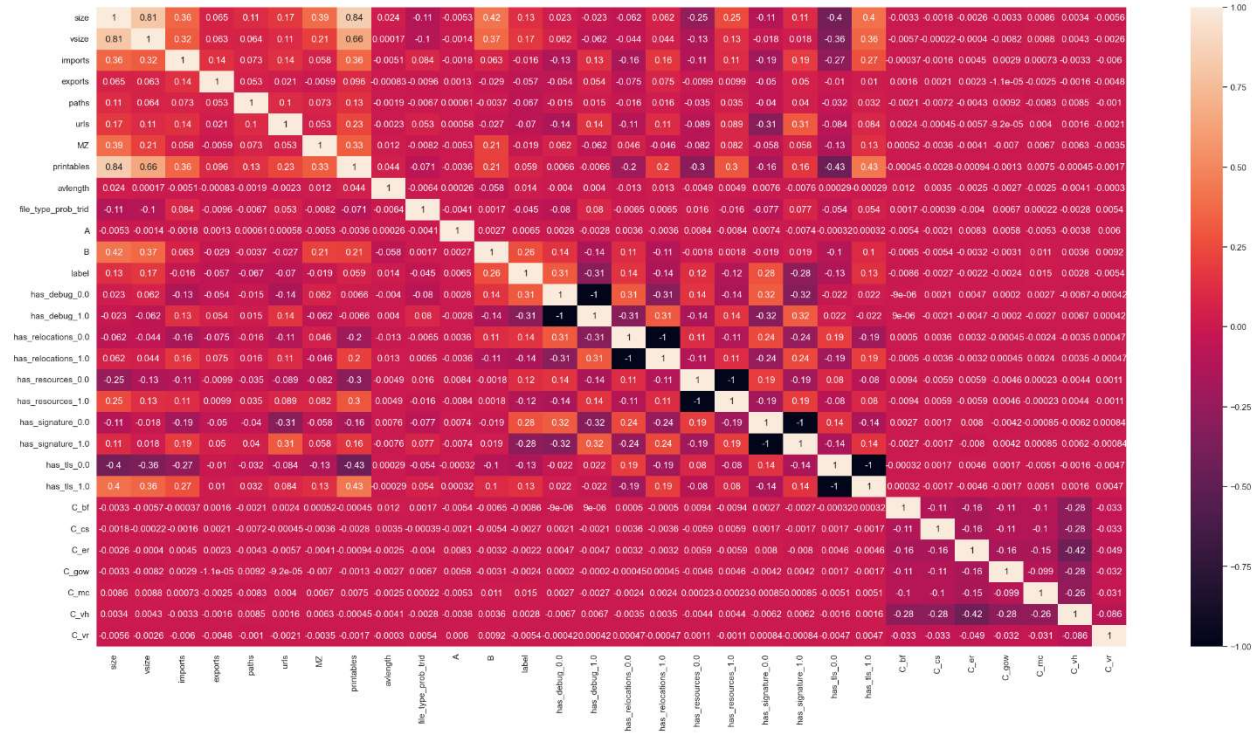
```
percentage of label 1 in the suspected outliers in imports is: 1.6666666666666667
percentage of label 1 in the suspected outliers in paths is: 14.583333333333334
percentage of label 1 in the suspected outliers in MZ is: 12.76595744680851
percentage of label 1 in the suspected outliers in exports is: 1.8867924528301887
percentage of label 1 in the suspected outliers in avlength is: 98.59154929577466
percentage of label 1 in the suspected outliers in urls is: 13.793103448275861
percentage of label 1 in the suspected outliers in B is: 84.4155844155844
```

The IQR (Interquartile Range) method is used to identify outliers from a dataset. It involves calculating the IQR, which is the difference between the third quartile (Q3) and the first quartile (Q1). The lower and upper bounds are determined by multiplying the IQR by 1.5. Data points outside these bounds are considered outliers.

3) MinMaxScaler rescales the data by subtracting the minimum value and dividing by the range (maximum value minus minimum value) of each feature. All features will have values between 0 and 1, where 0 represents the minimum value and 1 represents the maximum value in the original dataset. This normalization preserves the relative relationships and distributions among the data points while putting them on a consistent scale.

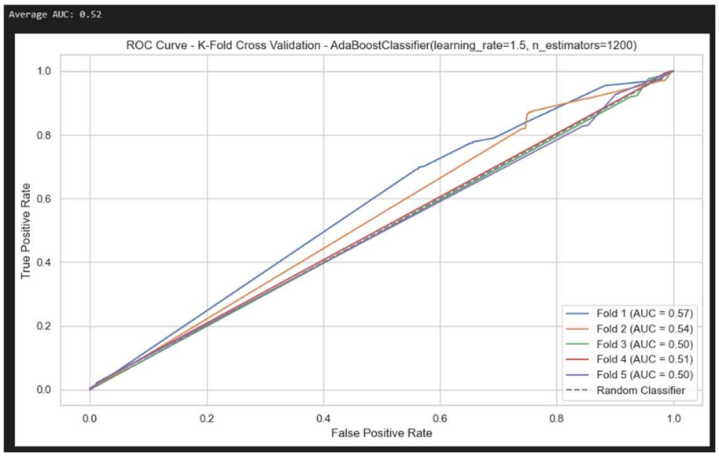
4) OneHotEncoder is used to convert categorical features into a numerical representation.
It transforms categorical features into binary columns and adds new ones (known as “dummy features”) for each unique category in the original feature.
One of the benefits of this method is that it avoids any ordinality assumptions and it helps prevent the algorithm from assigning unintended numerical relations to different categories.

5) Second correlation matrix:

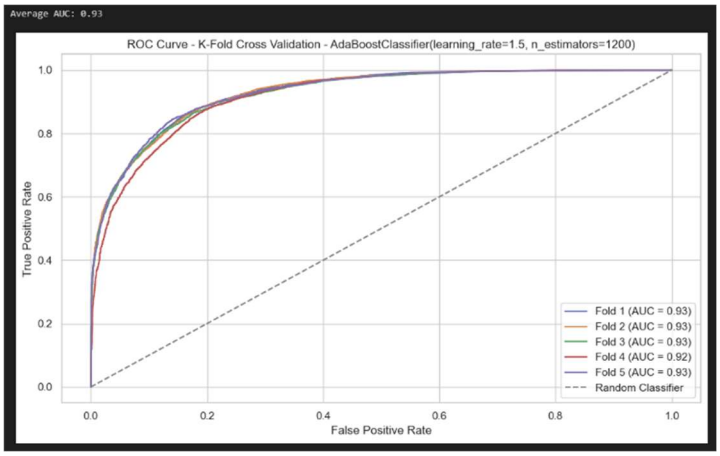


6) The ROC curves for each model:

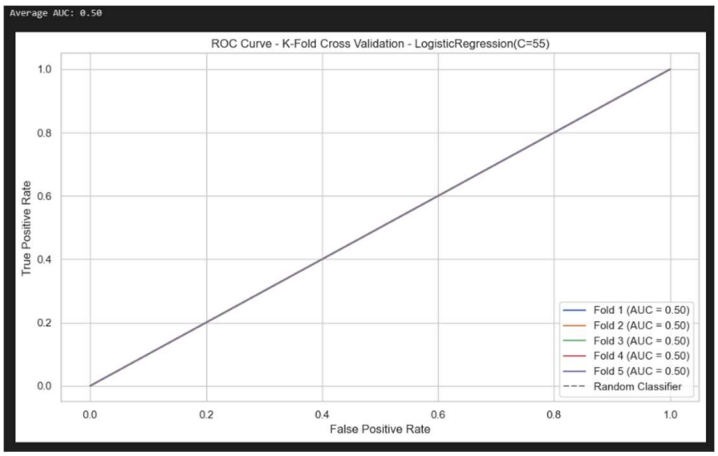
Ada Boost with PCA:



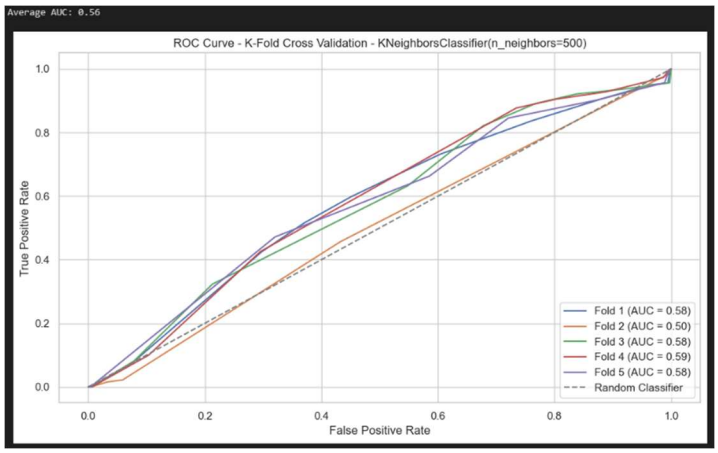
Ada Boost without PCA:



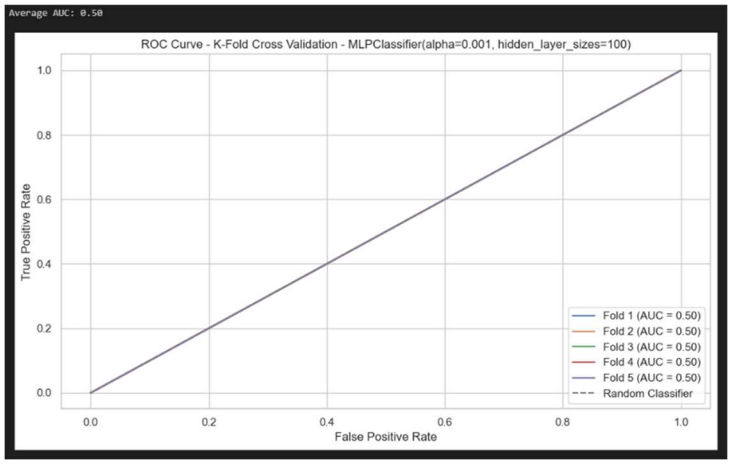
Logistic regression:



KNN:

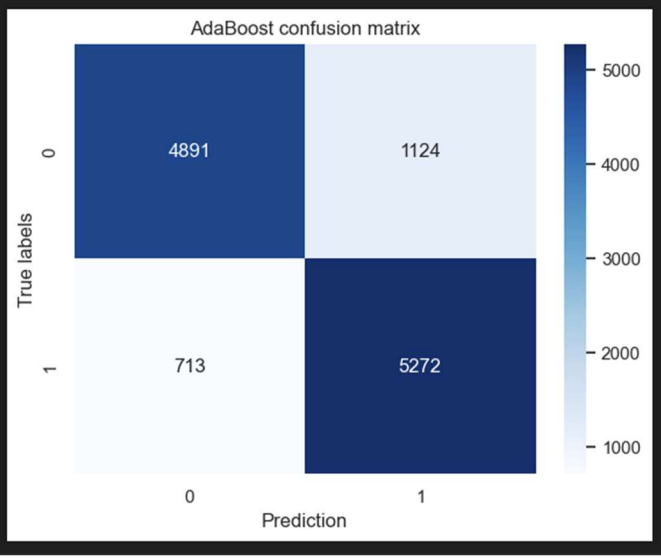


MLP:

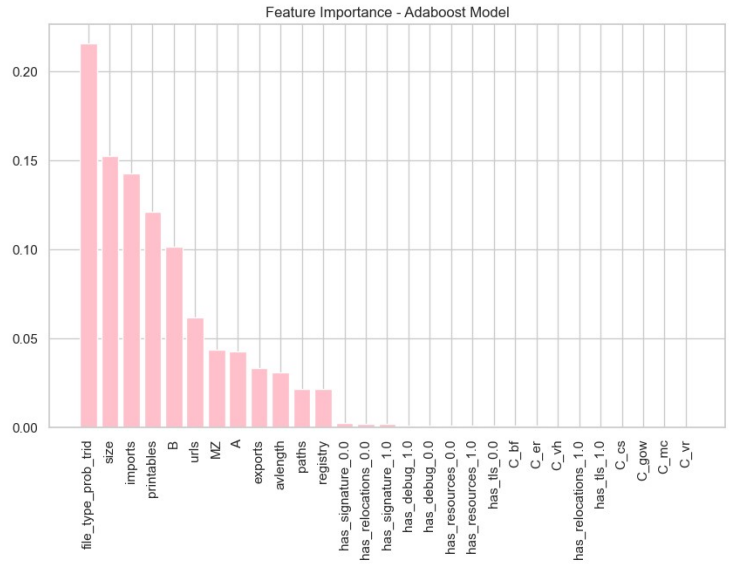


7) In the context of a Confusion Matrix with the Ada-Boost model, the cells represent classifications made by the model based on the true and predicted labels of the validation dataset.
- True Positive (TP): the number of files that are malicious as the prediction of the model. (Bottom right)
- False Positive (FP): the number of files that are not malicious, but the prediction of the model is that they are malicious. (Top right)
- False Negative (FN): the number of files that are malicious, but the prediction of the model is that they are not malicious. (Bottom left)
- True Negative (TN): the number of files that are not malicious as the prediction of the model. (Top left)
In this case, the model has an accuracy of 84.69%, indicating that it classified 84.69% of the samples correctly.
In addition, the model has a precision of 82.42%, suggesting that when it predicts a sample as malicious, we can be 82.42% sure that it is correct.
Moreover, the model has a recall of 88.08%, indicating that it correctly identifies 88.08% of the actual malicious samples.

```
x_train.shape: (48000, 23), y_train.shape: (48000,), x_validation.shape: (12000, 23), y_validation.shape: (12000,)
accuracy = 0.8469166666666667
precision = 0.8242651657285803
recall = 0.8808688387635756
```



8)
```
Train AUC score: 0.8728015036112402
Validation AUC: 0.847001335425013
Performances gap: Train AUC score - Validation AUC =  0.025800168186227213
```

9)


Appendix - Contribution of teammates:

We did everything together. We met almost every day at Inbar's house near the university, and worked together from the beginning till the end :)