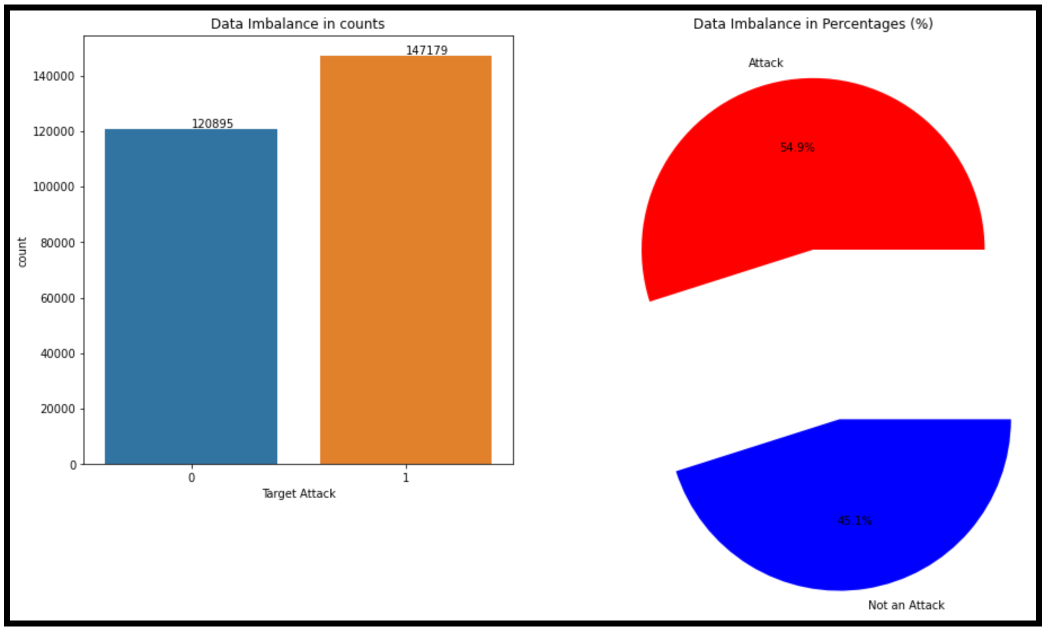
CSI - 5388[F] - AI for Cybersecurity applications
Assignment - 3

Submitted by - Deepank Kartikey (300247255)

<u>TASK - 1</u>

1. <u>Data Analysis</u>

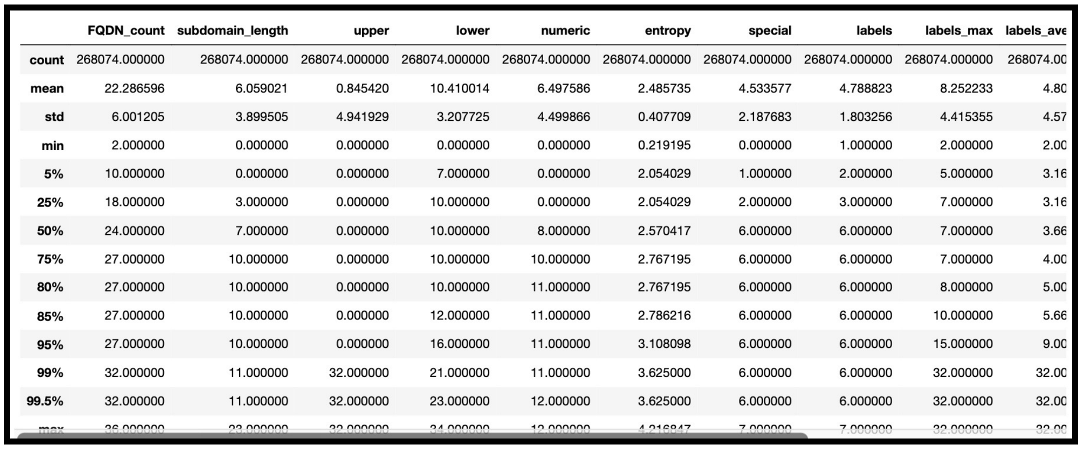
- Being a cybersecurity dataset, the original nature of features may be hidden due to confidentiality (*categorical features may be encoded as integer*). Although it is evident while checking the dtypes of features that only 3 features are categorical but when unique value counts are checked for them, it helped in understanding if a feature were categorical or not. By calculating the **unique category counts** it was concluded that multiple features are categorical variables but encoded as numerical. This can be found in Static_model.ipynb notebook as well.
- Secondly, **box plots** were plotted to check the distribution of each feature with 'Target Attack' (Refer to **appendix - 1**). It is evident from the box plots that FQDN_count, subdomain_length, lower, labels_max, labels_average, len have **outliers** in the dataset and those need to be removed.
- To validate if the dataset is **imbalanced or not**, the distribution of classes of 'Target Attack' has been plotted using bar graph and pie chart. It is evident that there is a slight imbalance in the data, which is further handled using undersampling towards the end.



- To check for **skewness** in data, displot or **distribution plots** were created for each of the numerical features. From all the plots it is evident that variables are categorical (encoded as numerical) in nature and hence no skewness treatment is required. (Refer to the **appendix - 2** for the distribution plots)

2. **<u>Feature Engineering and Data Cleaning</u>**

- The data was cleaned for **outliers detected** using box plots. After careful consideration and exploring data at different percentile levels like 0.05 (5 percentile), 0.25 (quartile), 0.5 (50 percentile), 0.75, 0.8, 0.85, 0.95, 0.99, 0.995; it was decided to remove rows where values of each column are greater than their respective 95 percentile, as 95 -percentile and 100-percentile distribution had visible differences.

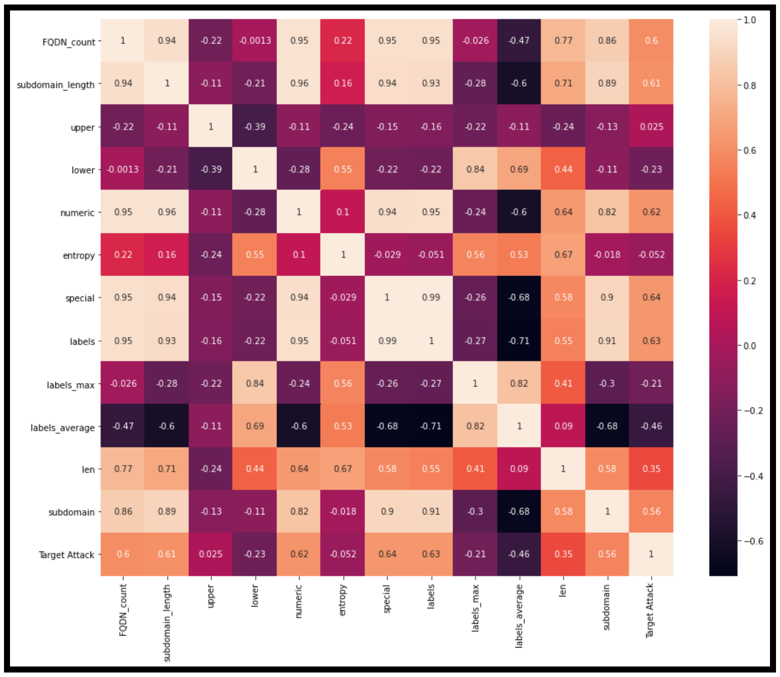


	FQDN_count	subdomain_length	upper	lower	numeric	entropy	special	labels	labels_max	labels_ave
count	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.00
mean	22.286596	6.059021	0.845420	10.410014	6.497586	2.485735	4.533577	4.788823	8.252233	4.80
std	6.001205	3.899505	4.941929	3.207725	4.499866	0.407709	2.187683	1.803256	4.415355	4.57
min	2.000000	0.000000	0.000000	0.000000	0.000000	0.219195	0.000000	1.000000	2.000000	2.00
5%	10.000000	0.000000	0.000000	7.000000	0.000000	2.054029	1.000000	2.000000	5.000000	3.16
25%	18.000000	3.000000	0.000000	10.000000	0.000000	2.054029	2.000000	3.000000	7.000000	3.16
50%	24.000000	7.000000	0.000000	10.000000	8.000000	2.570417	6.000000	6.000000	7.000000	3.66
75%	27.000000	10.000000	0.000000	10.000000	10.000000	2.767195	6.000000	6.000000	7.000000	4.00
80%	27.000000	10.000000	0.000000	10.000000	11.000000	2.767195	6.000000	6.000000	8.000000	5.00
85%	27.000000	10.000000	0.000000	12.000000	11.000000	2.786216	6.000000	6.000000	10.000000	5.66
95%	27.000000	10.000000	0.000000	16.000000	11.000000	3.108098	6.000000	6.000000	15.000000	9.00
99%	32.000000	11.000000	32.000000	21.000000	11.000000	3.625000	6.000000	6.000000	32.000000	32.00
99.5%	32.000000	11.000000	32.000000	23.000000	12.000000	3.625000	6.000000	6.000000	32.000000	32.00
max	36.000000	23.000000	32.000000	34.000000	12.000000	4.216847	7.000000	7.000000	32.000000	32.00

- Only one feature was found to have **null values** (i.e. *longest_word*) and only 8 such records existed in the dataset. Since the amount of null records is very low in comparison to the whole dataset hence the corresponding rows were removed.
- For converting features with **string values to numerical**, Python's hash() was used and applied to *timestamp, longest_word,* and *sld* features. Other methods like Frequency encoding was also tried on the data but there was a huge disadvantage of these methods: If categories have the same count, some information may be lost as different categories will be encoded with the same value(s).

3. **<u>Feature Filtering/Selection</u>**

- **Correlation-**based feature selection and Chi-square Significance test are the methods used for feature selection. By calculating the correlation score between the target and all the features, it was clear that features: upper, and entropy have very less correlation values when compared to other features, hence they were dropped. (Refer to Matrix)
- **Chi-square** significance values were also used to select the top 10 features to further build the model.
- Following **features** were **finalized** after applying both the above-mentioned methods: ['FQDN_count', 'subdomain_length', 'numeric', 'special', 'labels', 'labels_average', 'longest_word', 'sld', 'len', 'subdomain']

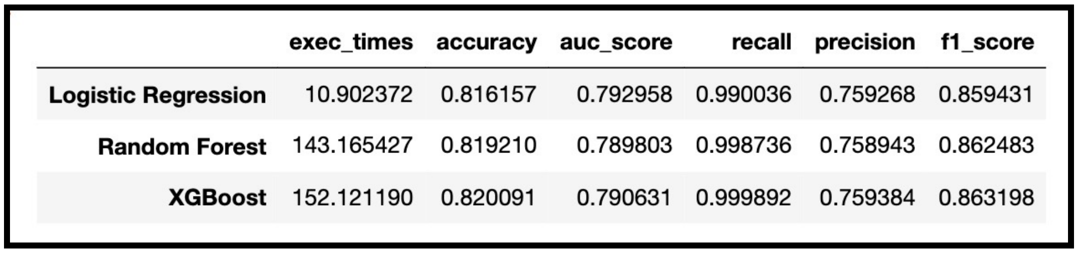


4. **Model training**

- After pre-processing the data, **train_test_split** was used to split the data with 80% training and 20% testing splits. The property **stratify = y** was used to preserve the ratio of both classes after splitting as well.
- Once the splits were done, data was normalized using **Min-Max scaling**. The reason for using Min-Max being features did not follow a normal distribution which is evident from distribution plots in the appendix section (Refer to the appendix - 2). The scaler model was exported using the **Pickle** library for usage while creating the dynamic model.
- To train and tune the models, Stratified K fold was used along with Randomized Search CV. I preferred Randomized Search CV to get the best hyperparameters in quicker times.

5. **Model Evaluation**

- I used Logistic Regression, Random Forest, and XGBoost **algorithms** for training and predictions on the static dataset. Following is the summary of all metrics for the models:



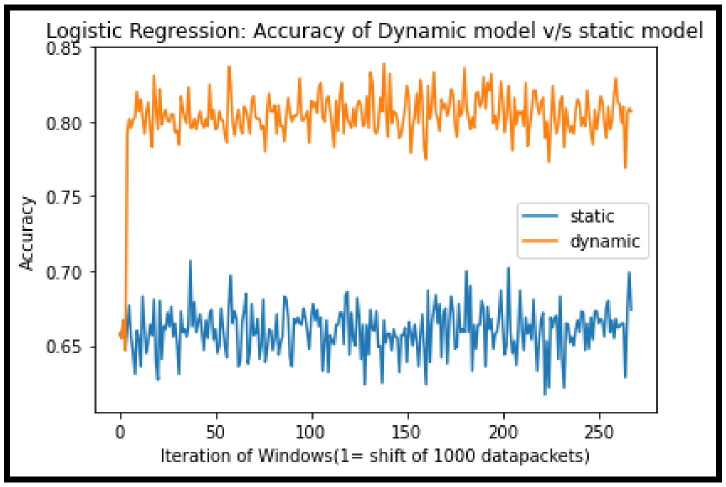
	exec_times	accuracy	auc_score	recall	precision	f1_score
Logistic Regression	10.902372	0.816157	0.792958	0.990036	0.759268	0.859431
Random Forest	143.165427	0.819210	0.789803	0.998736	0.758943	0.862483
XGBoost	152.121190	0.820091	0.790631	0.999892	0.759384	0.863198

The 2 major metrics that have been considered to choose the model for the dynamic part are accuracy and execution times. It can be seen that although Tree models have very slightly better metrics but are very slow on execution times. I have defined the execution time as the time taken to train the model, hyperparameter tune it, and make predictions.

- In **online learning**, the re-training and prediction times play an important role and hence I chose a fine-tuned Logistic regression model and exported it using the Pickle library for use in the dynamic model.

TASK - 2

- For the **dynamic model**, the data produced is being consumed by a KafkaConsumer object. The incoming bytes are passed through eval() to be converted to a string and then split by the separator (,). Further X (features) and y (Target) are created by appending the parsed strings. Once the record count in a current window reaches 1000 or multiple of 1000, preprocessing (feature selection & transformation) and evaluation of the current window data is done followed by model prediction and re-training (if needed).
- Once all the data has been consumed, a performance comparison graph is also created between the static model and the dynamic model showing the change in accuracy.



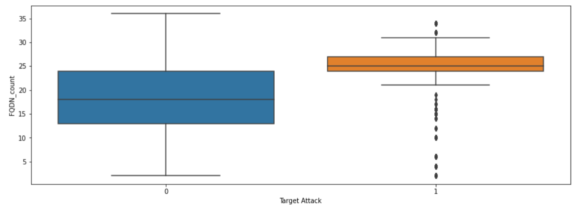
- It is evident from the scores and the comparison graph above that the static model's accuracy on the Kafka data revolved around 66% whereas the dynamic model's accuracy numbers were around 80% on average. The maximum accuracy score achieved by the dynamic model is 83%.
- The **re-training criteria** that has been chosen is when the accuracy score of the current window drops below 65% and in this case, the dynamic model is re-trained on the current window records. The reason behind the threshold being 65% is the average score of the static model i.e. 66%.
- While **analyzing the performance of the static model with the dynamic model** it is clear that the latter adapts to the incoming stream of data, calculates the accuracy score and

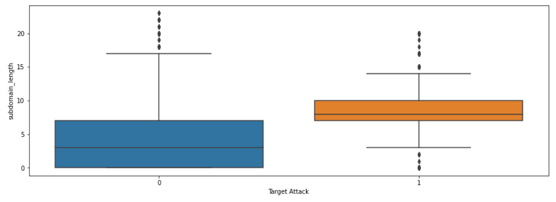
if it goes below the threshold, triggers the re-training logic. Once the retraining is complete and the new batch of data is incorporated, the dynamic or adaptive model gets improved and performs better.

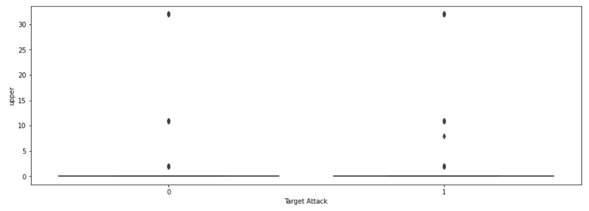
- It can be concluded that for a static model to predict accurately over time, the data it is operating on must have a similar distribution as its training data. However, in data streams like Kafka, the data keeps changing and distributions can drift over time which is referred to as **Concept drift**. It has also been detected in our comparison plot of the static and dynamic models.
- From this assignment, the following **advantages of online learning** were found that can be broadly described with 3 categories:
    a. <u>Continuously upgrade models:</u> It has the ability to continually update the model and quickly learn new distributions based on the latest data points.
    b. <u>Adapts to Change</u>: These adaptive models forget the old data and learn the new patterns based on the changes in data or the importance of the features.
    c. <u>Lesser memory</u>: Online ML has the capability to analyze large quantities of data as it divides the data into chunks or windows of smaller sizes and hence does not require a considerable amount of memory to store big datasets.
- Finally, there are some **limitations of online learning** as well:
    a. <u>No convergence</u>: In this type of learning, **each data point may only be seen** by the model **once** and hence there is no convergence which is fundamental of traditional machine learning.
    b. <u>Challenge with model evaluation</u>: Since the goal of online learning is to adapt the model to new data points, the performance of the new model cannot be evaluated using the past test data. Hence it cannot be decided if a model trained 10 minutes ago performs better or worse than a model trained 20 minutes ago.

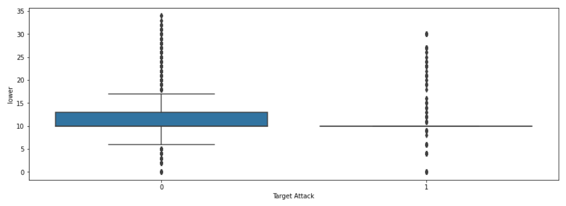
APPENDIX

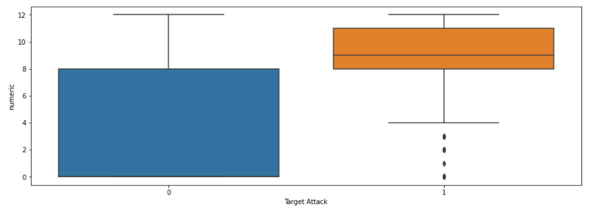
1. **Box Plots -** to check the distribution of features with target variable

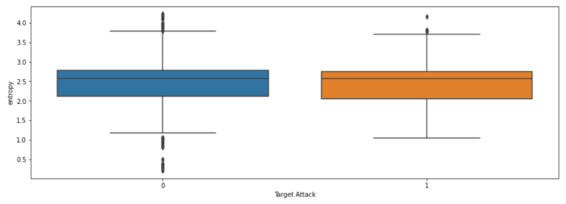


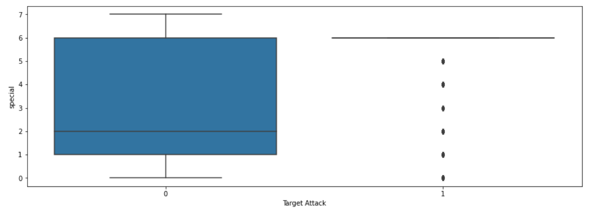


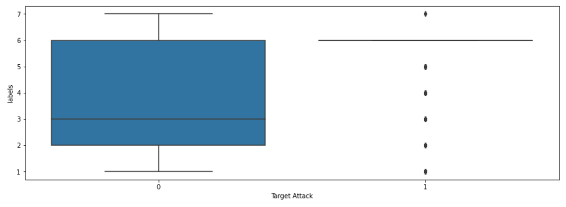


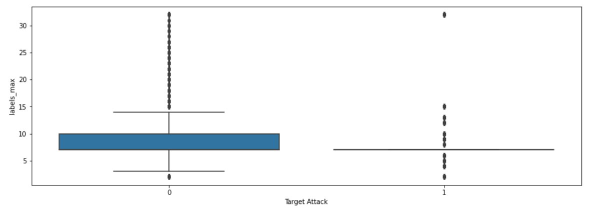


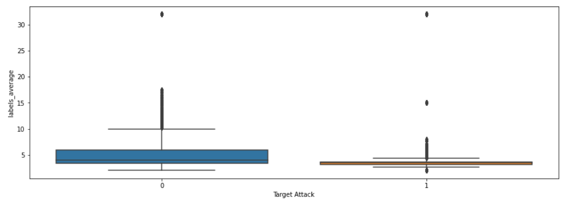


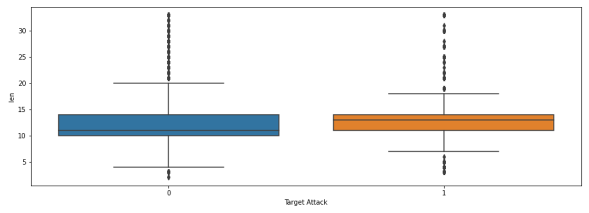


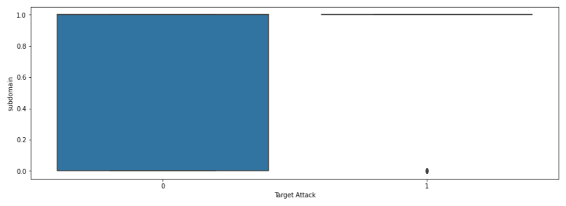




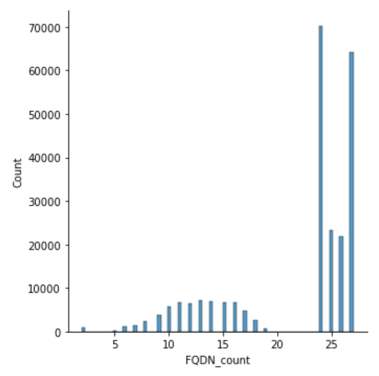


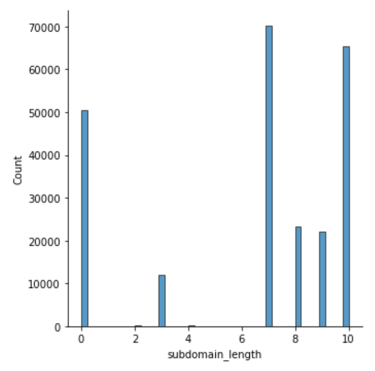


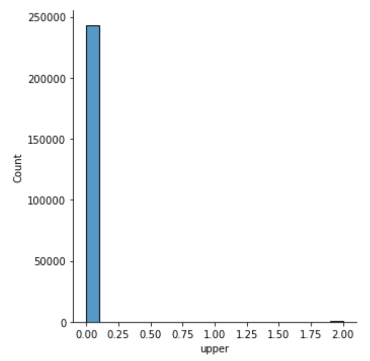


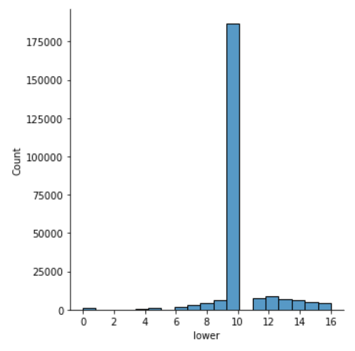


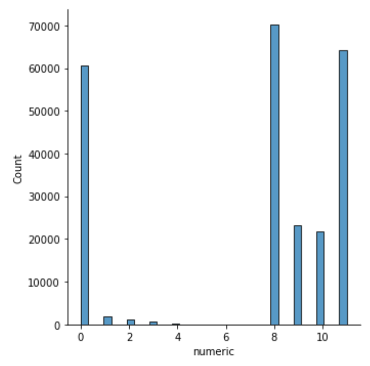
2. **distribution plots**- for checking data skewness in the numerical features

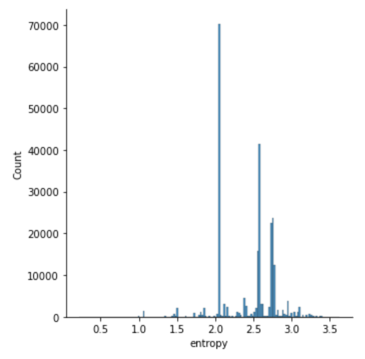

Count
FQDN_count

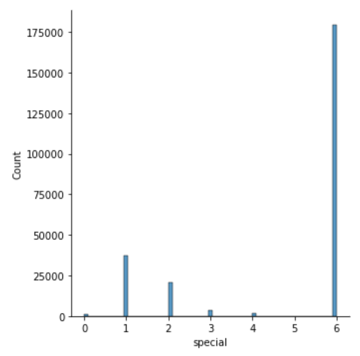

Count
subdomain_length

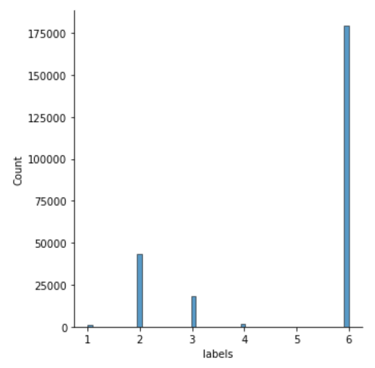

Count
upper

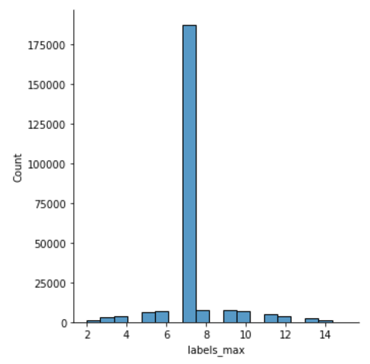

Count
lower

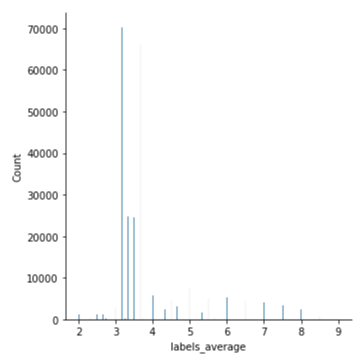

Count
numeric

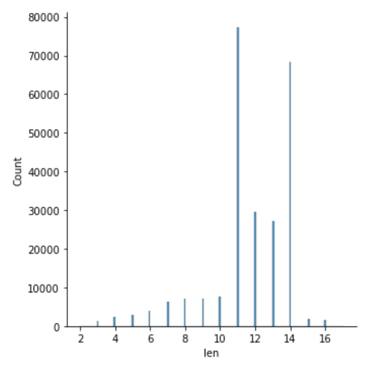

Count
entropy

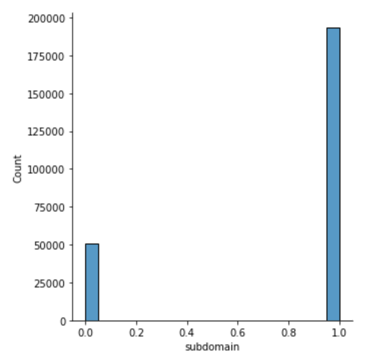

Count
special


Count
labels


Count
labels_max


Count
labels_average




Count
subdomain