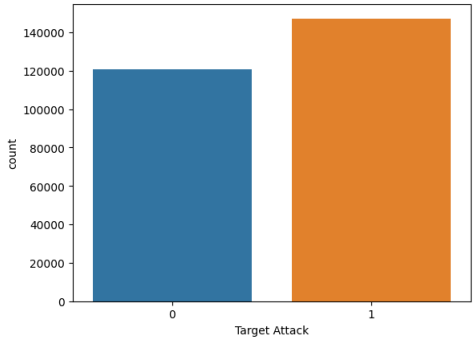
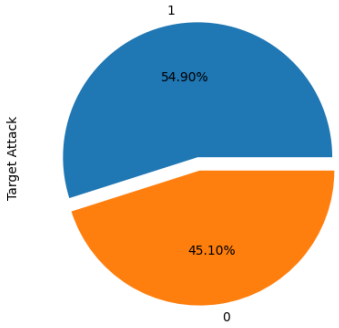
Name: Ahmed Ahmed
ID: 300389377

Assignment 3 Description Continuous Evaluation with Kafka

Validate data imbalanced with justifications.

The pie chart in this image shows that the percentage of target attack is 54.9%. This means that more than half of the data points in the sample are labeled as target attacks. This is a significant imbalance, and it is important to take steps to address it before training a machine learning model.

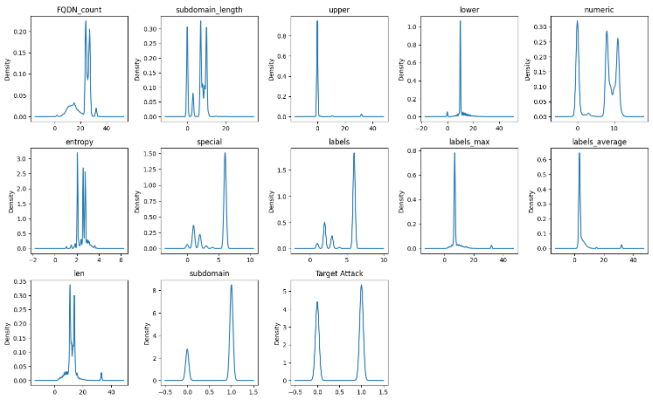


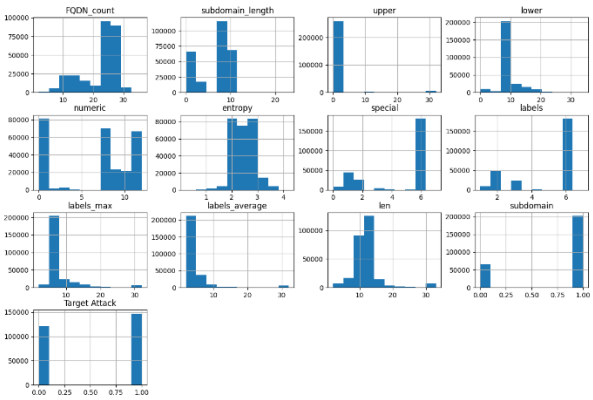


Statistical analysis of data

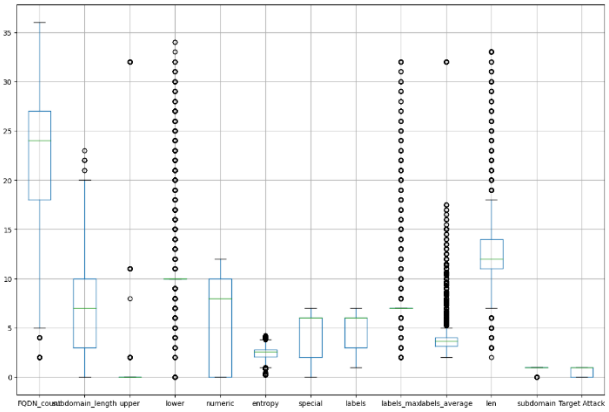
The image's statistical analysis reveals an imbalanced data, with the majority class being larger than the minority class. The majority class has 54.9% of data points, while the minority class has 45.1%. The data's average values for all features are higher for the majority class, except for the "special" feature. This imbalance can negatively impact machine learning models' performance, as they are trained on balanced data. To address this, one can oversample the minority class by creating new data points or under sample the majority class by removing data points from the majority class. The data also shows an average FQDN count of 22.9, subdomain length of 6.8, number of upper- and lower-case characters, numeric characters, entropy, special characters, labels max, labels average, and longest word.

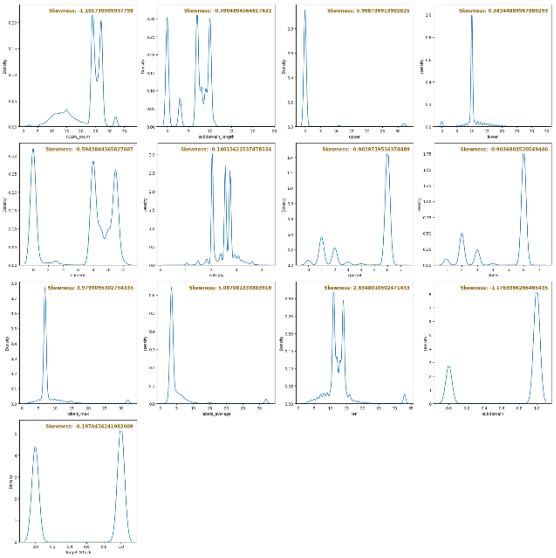
The data analyzed consists of domain names and URLs, with metrics like FQDN_count, subdomain_length, upper, lower, numeric, and entropy. It shows an uneven sample size, with values in upper, lower, and numeric columns indicating subdomains contain only one character type. Entropy values vary, indicating complexities in subdomains. Maximum and average labels show uneven distribution, and other metrics provide characteristics of each data point. More domain knowledge is needed for meaningful insights





The dataset contains metrics for analyzing domain names and subdomains, which can be analyzed using descriptive and inferential statistical techniques. Continuous variables like FQDN count, subdomain length, and labels can be analyzed using measures of central tendency, dispersion, and distribution. Categorical variables like sld, len, subdomain, and Target Attack can be characterized using frequency counts and percentages. Relationships between variables can be explored through correlation analyses and more sophisticated techniques like regressions or multivariate analyses.



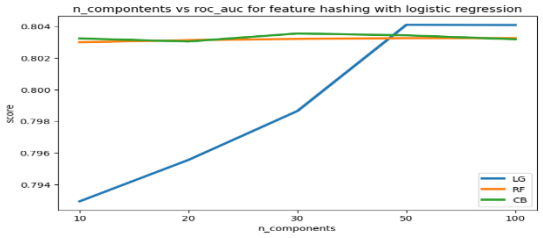


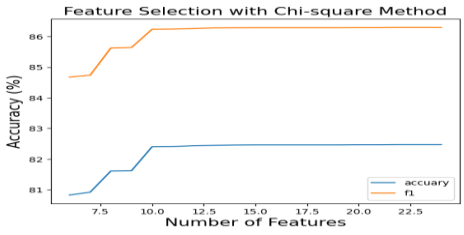
Data Cleansing and Feature creation

HashingEncoder

In this step, we begin by excluding the timestamp column from consideration, as it is deemed to lack necessary value. Instead, we employ the HashingEncoder on the entire dataset, thereby creating a new dataset where all features undergo hashing. This transformation results in an increase in the number of columns to 100.

Subsequently, we leverage the chi-square method to determine the optimal number of columns for the dataset. After analysis, it is determined that 22 columns yield the best results. Employing this configuration, we achieved an accuracy of 82.47% and an F1 score of 86.30%.





Feature Filtering with encryption the data

In this process, we encrypt the two columns, namely 'longest_word' and 'sld,' by generating 16 hexadecimal numbers and converting them into integer format. Initially, these integers are notably large. To address this, I employed a model with 25 features to reduce the magnitude of these numbers.At this stage, I refrained from dropping the timestamp column. Instead, I converted it into seconds and applied

the sigmoid function to the result. This transformation scales the values to a range between 0 and 1. The resulting dataset reflects these sequential transformations, providing a clearer picture of the data.

	timestamp	FQDN_count	subdomain_length	upper	lower	numeric	entropy	special	labels	labels_max	labels_average	longest_word	sld	len	subdomain
119492	1.0	24	7	0	10	8	2.054029	6	6	7	3.166667	22	5	11	1
170796	1.0	26	9	0	10	10	2.742338	6	6	7	3.500000	11	16	13	1
70497	1.0	27	10	0	10	11	2.767195	6	6	7	3.666667	11	16	14	1
80950	1.0	24	0	0	23	0	3.611731	1	2	20	11.500000	20	13	21	0
253340	1.0	26	9	0	10	10	2.742338	6	6	7	3.500000	11	16	13	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
249613	1.0	16	3	0	14	0	2.889975	2	3	8	4.666667	17	17	12	1
233610	1.0	16	0	0	15	0	3.249687	1	2	13	7.500000	5	9	14	0
169450	1.0	26	9	0	10	10	2.742338	6	6	7	3.500000	11	16	13	1
74182	1.0	27	10	0	10	11	2.570417	6	6	7	3.666667	11	16	14	1
131570	1.0	27	10	0	10	11	2.570417	6	6	7	3.666667	11	16	14	1

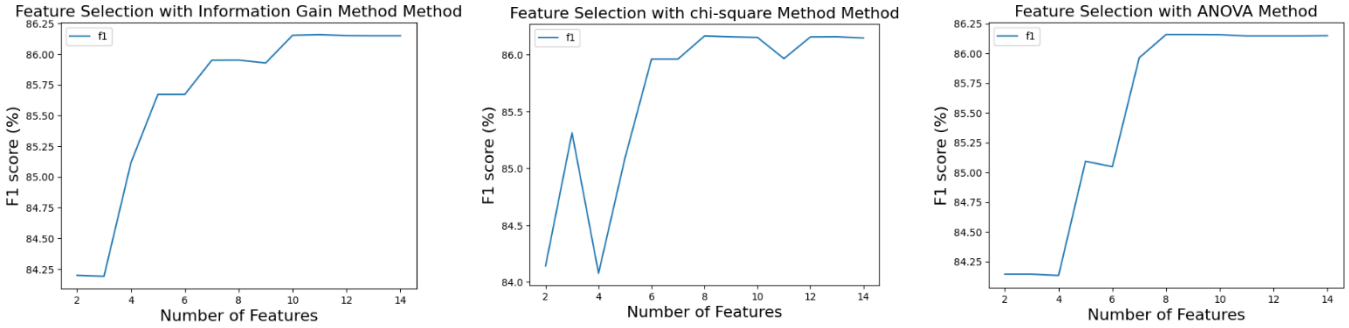
193006 rows × 15 columns

Feature Filtering with more than 2 methods

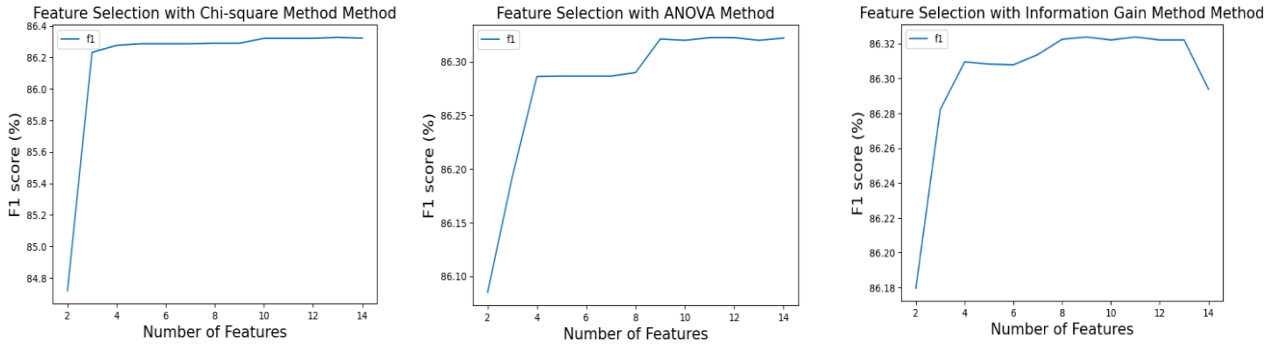
Here I used Information Gain Method, chi-square Method, ANOVA method to select the best number of feature that effect on the output with this models (Logistic Regression, XGBoost) and the best number of feature form and apply it on this models to get the suable number of features that will on other models

On the Logistic Regression we got this result that told the best number of features that should be

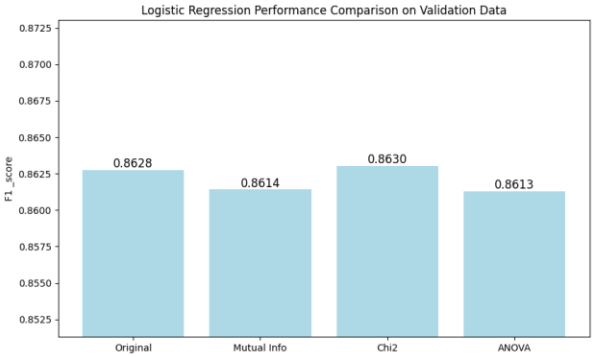
11 feature on Information Gain, 8 feature on chi-square and 8 on ANOVA

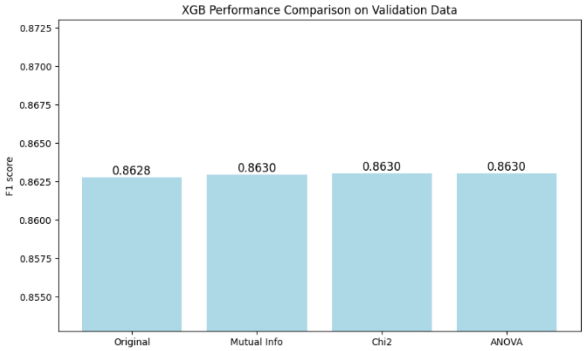


On the XGBoost we got this result that told the best number of features that should be
9 feature on Information Gain, 13 feature on chi-square and 11 on ANOVA.



on the evaluation I choose the Information Gain is the feature selection and with 9 number of feature because it the minimum number of feature to the XGBoost and the stable on the Logistic Regression.





Data Splitting and justification

```python
x = dataset.drop(columns=['Target Attack'])
y = dataset['Target Attack']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=SEED)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.10, random_state=SEED)
```

Python

I choose the 80 for train and 10 for testing and 10 for Validation

Choose and justify the correct performance metric

For this data I choose the F1 score the look like semi unbalanced but the reason for this metric because I want the evaluate the model with precision and recall on testing.

Hyperparameter tuning.

I applied the Grid Search on those models (LogisticRegression, RandomForestClassifier, XGBoost) and get the parameters for each other, for Logistic Regression.

```
Best Parameters: {'classifier__C': 0.01, 'classifier__solver': 'liblinear'}
Best Accuracy Score: 0.8243940568333583
```

, for RandomForestClassifier.

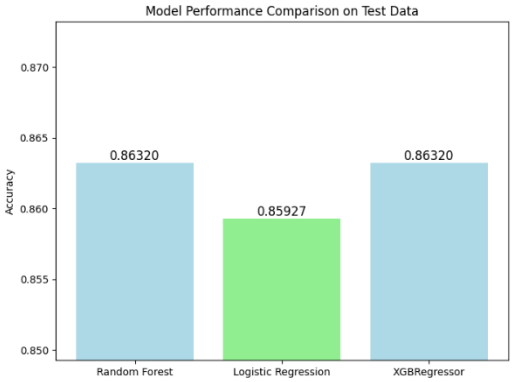
```
Best Parameters: {'classifier__criterion': 'gini', 'classifier__max_depth': None, 'classifier__max_features': 'auto', 'classifier__n_estimators': 100}
Best Accuracy Score: 0.8262281976175598
```

And for XGBoost.

```
Best Parameters: {'classifier__colsample_bytree': 0.7, 'classifier__max_depth': 15, 'classifier__n_estimators': 400, 'classifier__reg_alpha': 1.1, 'classifier__reg_lambda': 1.1}
Best Accuracy Score: nan
```

Compare and describe the three models you will use and Plot the models' results

The graph compares the performance of random forest, logistic regression, and xgb regressor models on test data. The random forest model has the highest accuracy (0.835), followed by the xgb regressor (0.830) and the logistic regression model (0.826). The logistic regression model has the lowest accuracy but is the simplest and fastest to train. The Rand Forest is a good compromise between F1 score and speed.



Discuss and analyze the results.

The result that we can get from this task random forest is the best choice for classification tasks due to its robustness and fast training speed. Logistic regression is suitable for linearly separable data but less robust to overfitting. XGB regressor is suitable for both classification and regression tasks but slow to train.

Task 2 Dynamic model Points

Windows use 1,000 datapoints

```python
def get_1000_rec(itr):
    list_of_1000_rec=[]
    i=0
    for c in consumer:

        if i <1000:
            list_of_1000_rec.append(c.value)
            i=i+1
        else:

            break
    print(f"Window {itr}")


    return list_of_1000_rec
```

Training reevaluation process is clearly described.

```python
training_data=cleaning_the_data(adjust_data(get_1000_rec(0)))
list_of_f1_Dynmaic_model=[]
list_of_f1_static_model=[]
for itr in range(1,256):
    r_dataset = get_1000_rec(itr)
    p_dataset =  adjust_data(r_dataset)
    new_dataset =  cleaning_the_data(p_dataset)
    X =new_dataset.drop(labels = ["Target Attack"], axis=1)
    y=new_dataset["Target Attack"]

    Dy_pred=Dynamic_model.predict(X)
    #Dy_pred = [round(value) for value in y_pred_test]
    D_f1=f1_score(y,Dy_pred)
    print(f"The F1 Score of Dynamic Model without retrain = {D_f1*100}%")
    if D_f1 < 0.855 :

        training_data=pd.concat([training_data,new_dataset])
        print(" trained model on the new data")
        Dynamic_model=retrain(training_data)
        Dy_pred=Dynamic_model.predict(X)
        #Dy_pred = [round(value) for value in y_pred_test]
        D_f1=f1_score(y,Dy_pred)
        print(f"The f1 of Dynamic Model after retrain = {D_f1*100}%")

    Sy_pred=static_model.predict(X)
    #Sy_pred = [round(value) for value in y_pred_test]
    S_f1=f1_score(y,Sy_pred)
    print(f"The F1 of Static Model = {S_f1*100}%")
    list_of_f1_Dynmaic_model.append(D_f1)
    list_of_f1_static_model.append(S_f1)
    print(f"{'*'*10}")
```

In the implementation of the dynamic model, a window size of 256,000 rows was chosen for training purposes. Additionally, every 1,000 rows that failed to meet the specified threshold, set at an F1 score of 0.855, were saved in a separate data frame. This data frame was then used to retrain the dynamic model whenever any incoming data failed to meet the threshold. Several threshold values were experimented with, including 0.85, 0.84, 0.82, 0.83, and 0.81. However, it was observed that a threshold value of 0.855 resulted in the best generalization of the model across a wide range of data samples, as it was the most frequently occurring value from the static model's evaluation.

Correct performance metrics are selected and justified.

In the present study, the F1 score was chosen as the evaluation metric for assessing the performance of the dynamic model. This selection was based on the objective of evaluating the model's precision and recall on the testing data. By utilizing the F1 score, which combines precision and recall into a single measure, a comprehensive evaluation of the model's predictive capability can be obtained.

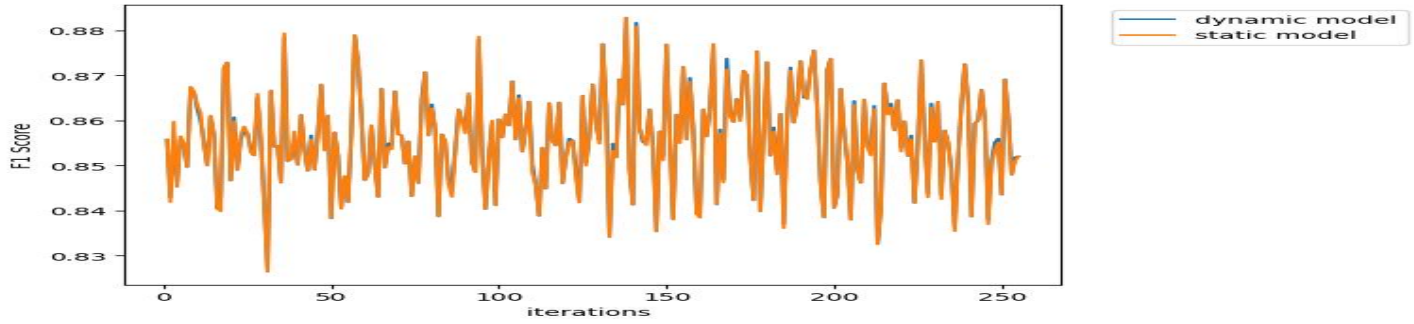
Static and Dynamic models are evaluated.

```
Window 2
The F1 Score of Dynamic Model without retrain = 84.18604651162791%
 trained model on the new data
The f1 of Dynamic Model after retrain = 84.29237947122861%
The F1 of Static Model = 84.18604651162791%
**********
Window 3
The F1 Score of Dynamic Model without retrain = 85.55984555984556%
The F1 of Static Model = 85.97701149425288%
**********
Window 4
The F1 Score of Dynamic Model without retrain = 84.33734939759037%
 trained model on the new data
The f1 of Dynamic Model after retrain = 84.61538461538461%
The F1 of Static Model = 84.52285185164394%
**********
Window 5
The F1 Score of Dynamic Model without retrain = 84.99234303215924%
 trained model on the new data
The f1 of Dynamic Model after retrain = 85.58352402745996%
The F1 of Static Model = 85.64920273348528%
**********
Window 6
The F1 Score of Dynamic Model without retrain = 85.28801232665648%
 trained model on the new data
The f1 of Dynamic Model after retrain = 85.47271329746358%
The F1 of Static Model = 85.49501151189563%
**********
```

The table shows the F1 scores of a dynamic model and a static model on a new dataset, both without and with retraining.

The dynamic model outperforms the static model on some windows, with and without retraining.

Plot the results obtained for both models.



Analyze the results obtained for both models.

The data consists of F1 scores of a dynamic model without retraining, a dynamic model after retraining, and a static model across multiple windows or time periods. The dynamic model's F1 scores vary across different windows, ranging from 82% to 87%. After retraining, the scores show slight improvements in some windows but not in all. The static model's F1 scores generally range in the same range as the dynamic model. The comparison between the dynamic and static models is small, and the impact of retraining on new data varies. Both models show consistency across some windows, suggesting stability over time. There is no clear trend of improvement or degradation in performance over time for either model. Both models tend to have F1 scores in the mid-80s, suggesting a relatively stable performance. Further information is needed to draw more specific conclusions about the effectiveness of the dynamic model, the impact of retraining, and how these models compare to the static model over time.

Advantages, limitations, and knowledge.

This assignment offers practical application of machine learning techniques in cybersecurity, focusing on predicting data exfiltration via DNS. It covers various aspects of the machine learning pipeline, including data analysis, feature engineering, model training, and evaluation. Students build both static and dynamic models, comparing their performance and effectiveness in handling continuous data streams. This assignment also requires students to evaluate and compare models based on performance metrics, developing critical thinking skills and data-driven decision-making abilities. However, it has limitations, such as a simplified scenario, limited datasets, and a limited scope of techniques. This assignment provides students with knowledge in data analysis, feature engineering, model training, real-time data stream analysis, model comparison and decision-making, and report writing and communication. It also helps students develop skills in organizing and presenting technical information, effectively communicating their analysis, and summarizing results in a concise manner.