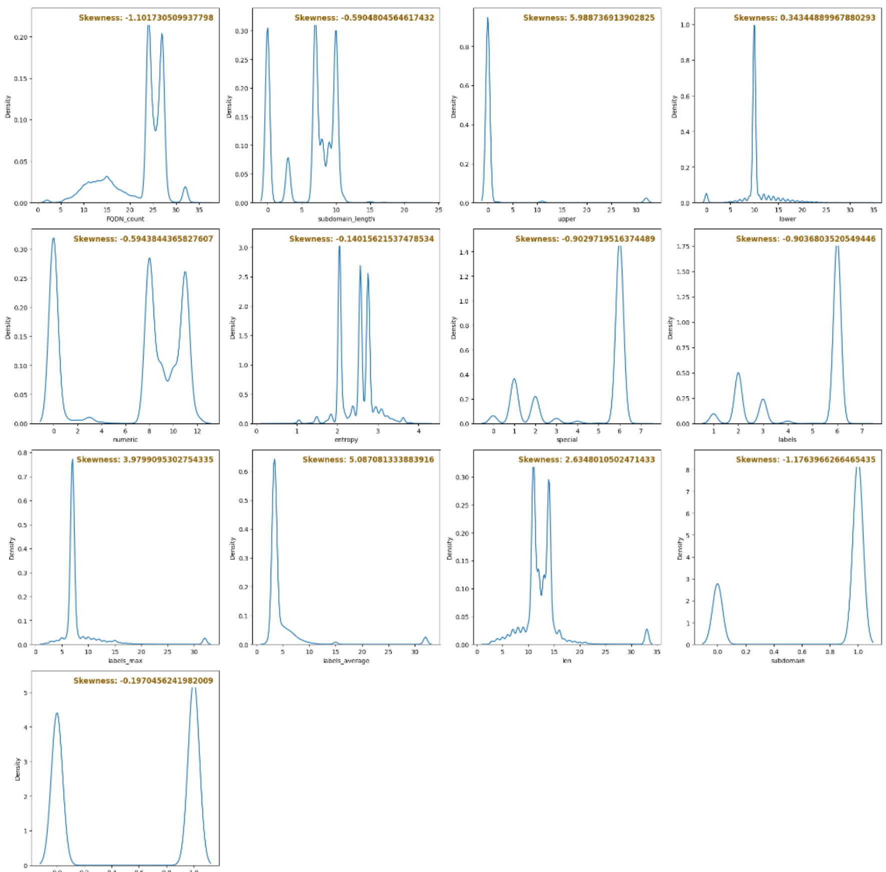
AI for Cybersecurity Applications 2023 – Assignment 3

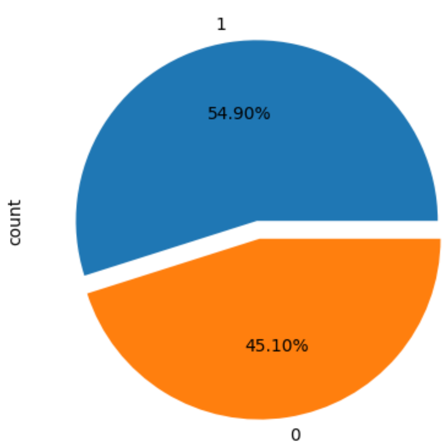
Hussien Tarek Ismail Abdelrazik - 300389897

Code Modularity: I have divided the implementation into functions & utilized, comments, naming conventions, pipelines & abstractions to make code readable, interpretable & clean.

- **For part 1, the Static Model**
  - **Data Imbalance:**

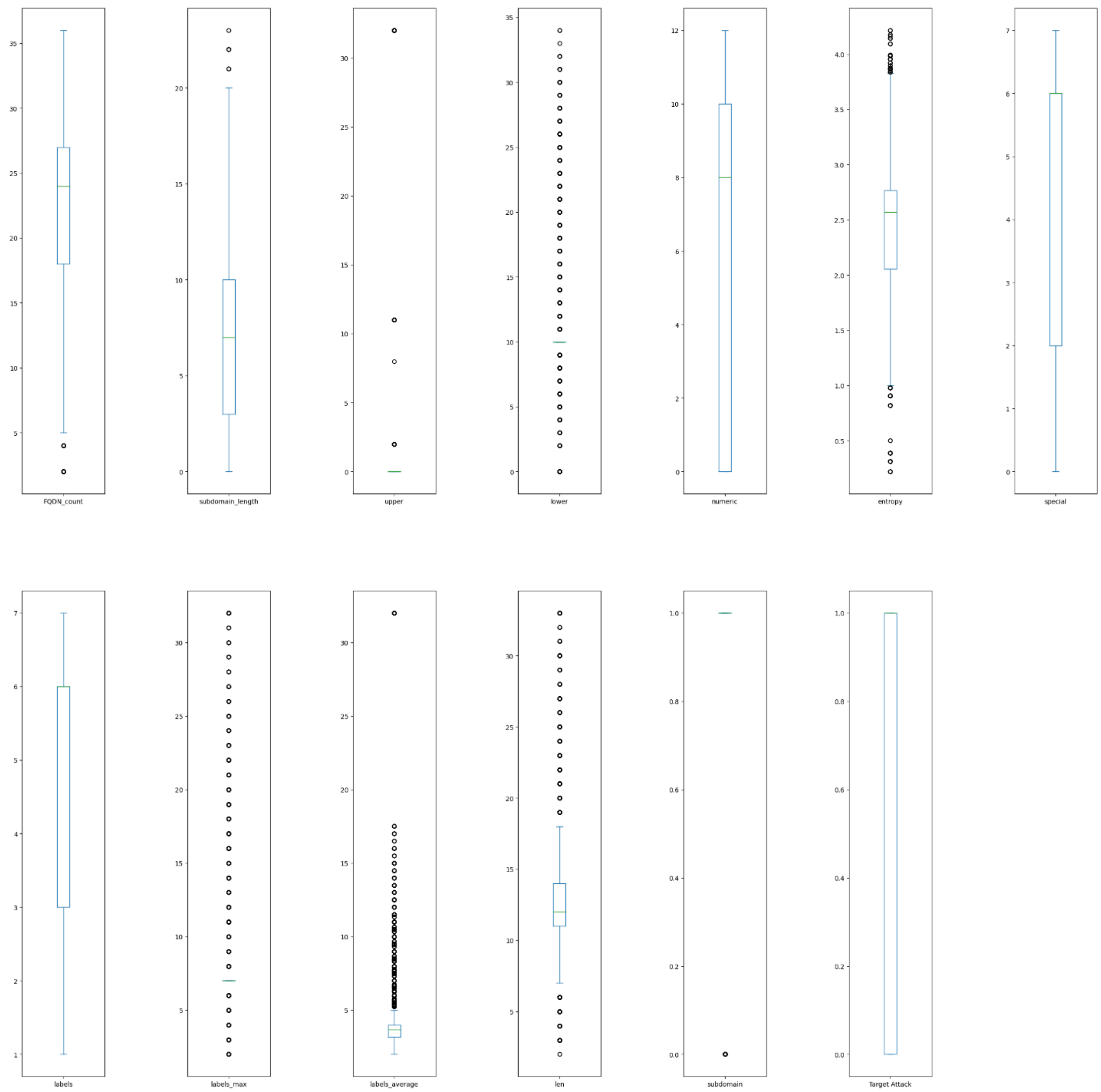


As shown in features distribution & skewness, we can see that most of features are skewed for example: the features (FQDN_count, special, labels) are negatively skewed & the features (upper, labels_max, labels_average) are positively skewed.



As shown in following pie chart of target column the Number of Positive samples is more than negative samples, by a little portion of the total samples that is not enough to result data imbalance also it focuses on the targeted class (positive samples)

- **Statistical Data Analysis:**



Summary Statistics:

	FQDN_count	subdomain_length	upper	lower	numeric	entropy	special	labels	labels_max	labels_average	len	subdomain	Target Attack
count	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000	268074.000000
mean	22.286596	6.059021	0.845420	10.410014	6.497586	2.485735	4.533577	4.788823	8.252233	4.802239	12.576714	0.753497	0.549024
std	6.001205	3.899505	4.941929	3.207725	4.499866	0.407709	2.187683	1.803256	4.415355	4.573066	4.177828	0.430975	0.497592
min	2.000000	0.000000	0.000000	0.000000	0.000000	0.219195	0.000000	1.000000	2.000000	2.000000	2.000000	0.000000	0.000000
25%	18.000000	3.000000	0.000000	10.000000	0.000000	2.054029	2.000000	3.000000	7.000000	3.166667	11.000000	1.000000	0.000000
50%	24.000000	7.000000	0.000000	10.000000	8.000000	2.570417	6.000000	6.000000	7.000000	3.666667	12.000000	1.000000	1.000000
75%	27.000000	10.000000	0.000000	10.000000	10.000000	2.767195	6.000000	6.000000	7.000000	4.000000	14.000000	1.000000	1.000000
max	36.000000	23.000000	32.000000	34.000000	12.000000	4.216847	7.000000	7.000000	32.000000	32.000000	33.000000	1.000000	1.000000

From the kernel distribution, boxplot & Statistics Summary it seems that the feature (subdomain) is binary categorical feature. we shall confirm that by checking the unique values of this feature.

	timestamp
0	56:19.8
1	07:23.9
2	23:15.1
3	04:51.9
4	12:44.0
...	...
268061	33:51.5
268062	36:02.5
268063	37:21.5
268064	24:25.1
268065	20:56.1

268066 rows × 1 columns

It seems like timestamp is in time format of (mm:ss.S) where mm is minutes, ss is seconds & S fraction of second. So we shall confirm by checking max & min value of each section of this format

	min	sec	sec_frac
0	56	19	8
1	07	23	9
2	23	15	1
3	04	51	9
4	12	44	0
...	...	...	...
268061	33	51	5
268062	36	02	5
268063	37	21	5
268064	24	25	1
268065	20	56	1

268066 rows × 3 columns

```
Max values:
min         59
sec         59
sec_frac     9
dtype: object

Min values:
min         00
sec         00
sec_frac     0
dtype: object
```

As shown the max value of minutes is 59 & max value of seconds is 59 & max value of fraction of second is 9. So we can confirm that the format of timestamp is (mm:ss.S)

- **Data Cleansing & Feature Creation**:

	timestamp
0	33798
1	4439
2	13951
3	2919
4	7640
...	...
268061	20315
268062	21625
268063	22415
268064	14651
268065	12561

Transform timestamp into Aggregated form of it's Components (minutes, seconds, fraction of second) converted into the total number of fraction of seconds to standardize all the values into same.

```
report_unique_values_count(X, categorical_columns)
```

✓ 0.0s

```
sld sld
192                               109517
224                                70188
FHEPFCELEHFCEPFFFACACACACACACABN    4498
DESKTOP-3JF04TC                     1961
239                                 1906
                                   ...
bukkit                                 1
pc-builds                              1
yaarlnk                                1
onenote                                1
queue-it                               1
Name: count, Length: 11110, dtype: int64 , Unique Values Count: 11110
longest_word longest_word
2         109981
4          70188
N           4498
C           2969
9           1906
           ...
yaa            1
queue          1
kit            1
airdrop        1
mal            1
Name: count, Length: 6224, dtype: int64 , Unique Values Count: 6224
```

As shown in analysis of the unique values of each categorical feature the number of unique values for each categorical feature is very large (6224 for longest_word & 11110 for sld) So transforming it by one hot encoding will result in very large

Hash Encoding Categorical Features a good rule of thumb is to use multiple of 2 number of components for hashing trick to avoid collisions. So we shall use 2^4 = 16.

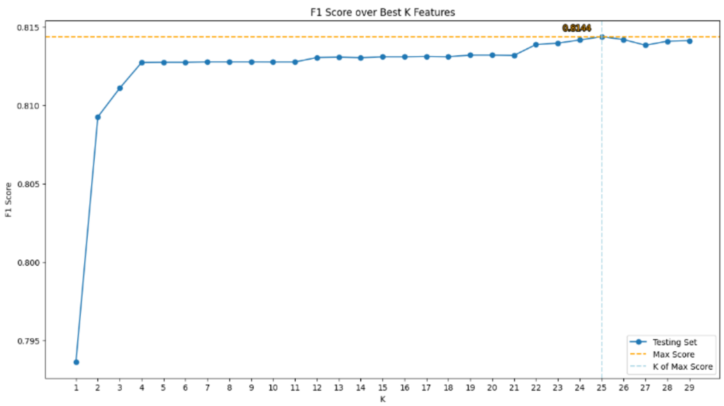
```
n_components = 2**4 #16
hash_columns=[f'hash_comp_{i}' for i in range(n_components)]
hasher = FeatureHasher(n_features=n_components, input_type='string')

# create column combined_cat that combine both values of the two categorical values of same record in single list to prepare it for hashing
X['combined_cat'] = X[categorical_columns].astype(str).values.tolist()

hashed_features = hasher.transform(X['combined_cat'])
hash_df = pd.DataFrame(hashed_features.toarray(), columns=hash_columns)
hash_result = pd.concat([X[categorical_columns], hash_df], axis=1)
hash_result
```

✓ 0.5s

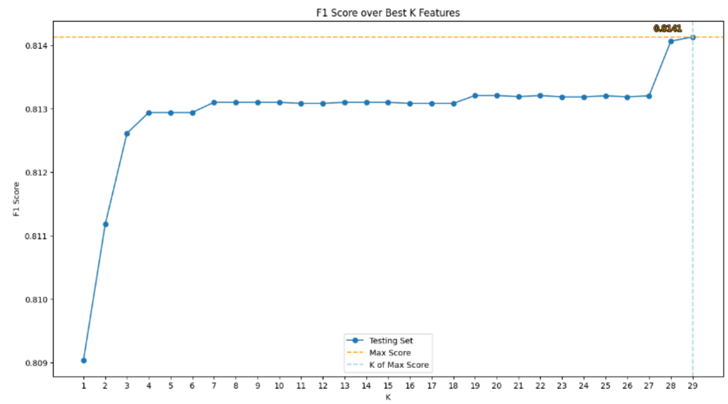
	sld	longest_word	hash_comp_0	hash_comp_1	hash_comp_2	hash_comp_3	hash_comp_4	hash_comp_5	hash_comp_6	hash_comp_7	hash_comp_8	hash_comp_9	hash_comp_10	hash_comp_11
0	192	2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
1	192	2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
2	192	2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
3	192	2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
4	local	local	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
268061	almaalomah	alma	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
268062	192	2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
268063	192	2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0
268064	radio-vintage	radio	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
268065	192	2	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0

268066 rows × 18 columns

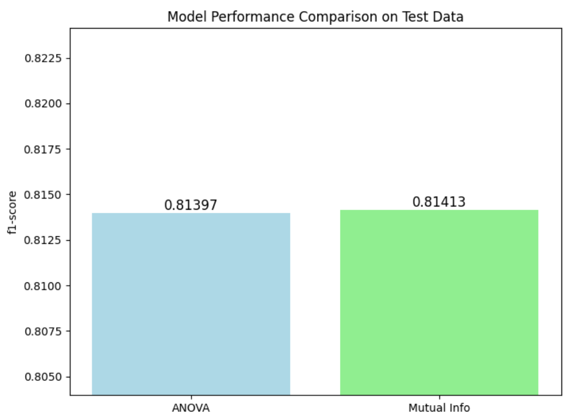
- **Feature Filtering:**



ANOVA Test Top K value for Best Number of Features is 23 as it results in the best baseline model performance (f1-score=0.8146)



Mutual Info Top K value for Best Number of Features is 29 as it results in the best baseline model performance (f1-score = 0.8142)



Since Top K performance of ANOVA baseline is better than Top K performance of Mutual Info baseline, Also with less number of Features we shall use ANOVA for feature filtering.

- **Data Splitting and justification:**

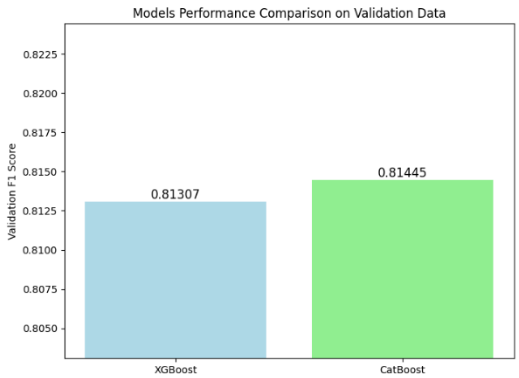
Using train_test_split to shuffle the data & split it into Train, Validation & Test Sets. Also we shall use stratify to keep the same ratio of positive & negative samples in all sets. The ratio of Train, Validation & Test Sets is 70%, 15% & 15% respectively So that we can have enough data to train the model & also enough data to validate & test the model.

- **Performance Metric Choice & Justification:**

Performance evaluation metric would be f1-score metric although data is balanced to balance the model performance between precision & recall.

- **Train & Compare Two Models:**

we shall experiment XGBoost & CatBoost learning algorithms for building Models as they are ensemble models that are effective in Anomaly Detection & Classification problems.



As shown the performance of both models are very close to each other, but Catboost model is slightly better than XGBoost model. Even though CatBoost Classifier is slower especially in training than XGBoost so we shall use XGBoost Classifier

- **Best Model Hyperparameter Tuning:**

```
# Define the XGBoost model
xgb_model = xgb.XGBClassifier(random_state=777)

# Define the hyperparameters grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.03, 0.1, 0.5],
}
```

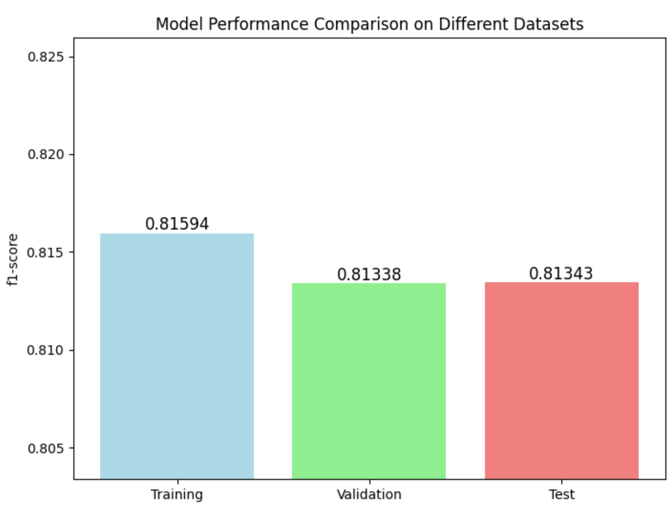
Hyperparameter Tuning Grid Search Parameters

```
Best Model Parameters: {'learning_rate': 0.5, 'max_depth': 7, 'n_estimators': 100}
Best Model F1-Score On Train Data Cross-Validation: 0.8132905747044733
```

```
XGBoost Optimized Validation F1 Score: 0.8134
```

Hyperparameter tuning resulted slight improvement in the model performance from 0.81307 to 0.8134 f1-score.

- **Best Model Optimized Results:**



Model Performance on Train Set is very close to the performance on Validation & Test Sets which is a Good Indication that the Model is less prune to overfitting.

- **For part 2, the Dynamic Model**
  - **1K Record Data Streaming Window:**

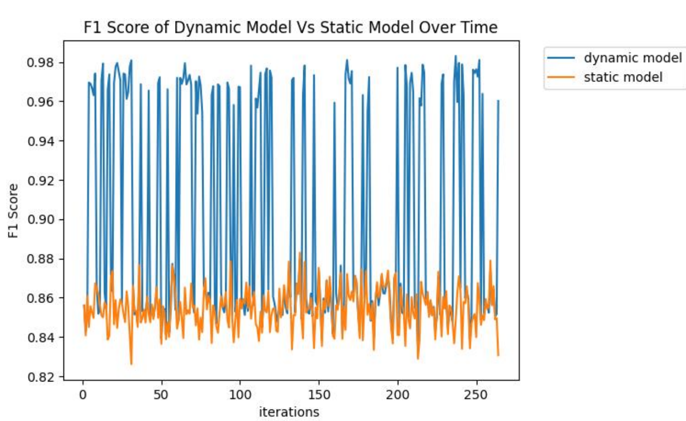
```python
def stream_batch(itr, consumer, num_of_rec=1e3):
    rec_counts, rec_batch_list = 0, []
    for c in consumer:
        if rec_counts < num_of_rec:
            rec_batch_list.append(c.value)
            rec_counts = rec_counts + 1
        else:   break
    print(f"Window {itr}")

    return rec_batch_list
```

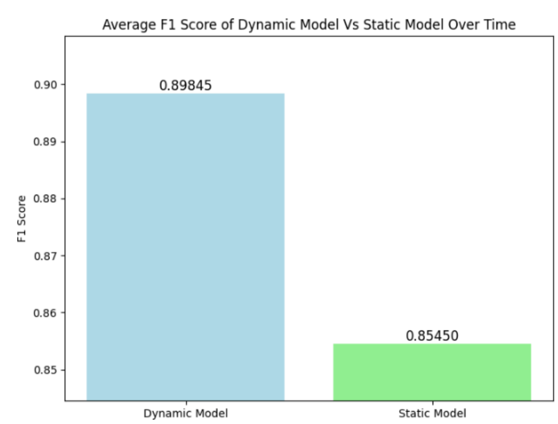
Function that Retrieves a Data Batch or Number of Records (default: 1000) from the Data Streamed by Through the Passed Consumer

  - **Training Re-evaluation Process:**

Load the Previously Fitted Best Model in Static Model & in Dynamic Model As Initial Model & Set Dynamic Model Retraining Threshold to 0.85 f1-socre this because at first we experimented a threshold of 0.8 & since actual static model performance on static test data was 0.81 so we thought it wouldn't tolerate 0.01 lower performance than model performance on static data but threshold was too low & the model was not able to detect any drifts in the data as static model performance on streaming data was higher than 0.81 which resulted the dynamic model to remain static. So we increased the threshold to 0.85. we made the num_of_batches to consume to 265 as total number or records in streaming data is approximately 265000 & we sated the batch_size to 1000 this require consuming the data in 265 batches. We consume streamed data in batches & record static & dynamic models' performance on each batch. The strategy used to re-train the dynamic model is that each time the model performance on a certain batch is below defined threshold this batch is accumulated over the previously batches that also resulted in a performance drop & the model is retrained on those accumulated performance drop batches in addition to static data. This Strategy is used to improve model performance on mistake & avoid forgetting.

  - **Static & Dynamic Models Results on Streaming Data:**





The average performance of dynamic model on all streamed batches is significantly higher than the static model performance on static test data