Data Collection, Data Cleansing and EDA

From the dataset given, there is no null data; therefore, handling of missing values is not required. However, when looking into the number of data in the “Action” column which will be used for training and testing in the later parts, **the data of each attribute is an imbalance which could cause bias towards the majority class**, which in this case is “allow”.
Also, **the model will not predict well when implemented on unseen data**.
In this project, the dataset is preprocessed via the following steps.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65532 entries, 0 to 65531
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Source Port           65532 non-null  int64
 1   Destination Port      65532 non-null  int64
 2   NAT Source Port       65532 non-null  int64
 3   NAT Destination Port  65532 non-null  int64
 4   Action                65532 non-null  object
 5   Bytes                 65532 non-null  int64
 6   Bytes Sent            65532 non-null  int64
 7   Bytes Received        65532 non-null  int64
 8   Packets               65532 non-null  int64
 9   Elapsed Time (sec)    65532 non-null  int64
 10  pkts_sent             65532 non-null  int64
 11  pkts_received         65532 non-null  int64
dtypes: int64(11), object(1)
memory usage: 6.0+ MB
```

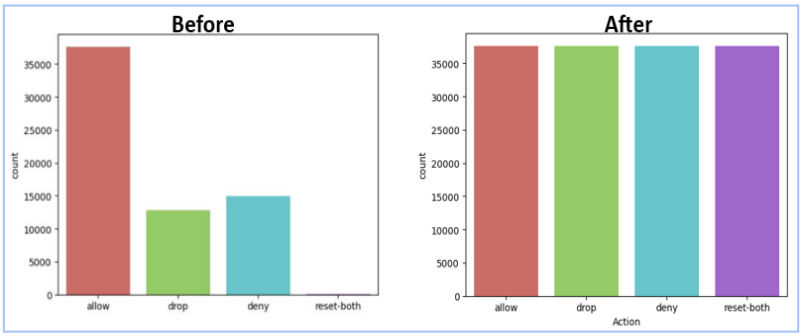
```
allow          37640
deny           14987
drop           12851
reset-both        54
Name: Action, dtype: int64
```

1) Import data, split and apply SMOTE (for resampling)

```
# Apply SMOTE (regular, svm)
sm = SMOTE(sampling_strategy='auto')
X_resampled, y_resampled = sm.fit_resample(X, y)
```

```
value_counts = pd.Series(y_resampled).value_counts()
print(value_counts)
```
✓ 0.0s

```
allow          37640
drop           37640
deny           37640
reset-both     37640
Name: Action, dtype: int64
```



2) Feature Scaling

2.1) Check if the data is normal distributed or not by using Shapiro Test - The shapiro test resulted in p-value = 0.0 that suggests the data is not normal distributed (not normal distribution)

```
#Check Shapiro test (Normal distribution)
scipy.stats.shapiro(X_resampled)
```

```
/usr/local/lib/python3.9/dist-packages/scipy/stats/_morestats.py:1816: UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
ShapiroResult(statistic=0.0008412003517150879, pvalue=0.0)
```

2.2) After check that data is not normal distributed, then apply Normalization to the data - Resulting in 4 features obtained from normalization including Bytes, Bytes Sent, Packets and Packets sent

```
#Normalization
x_normal= (X_resampled - np.min(X_resampled)) / (np.max(X_resampled) / np.min(X_resampled)).values
x_normal
```

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received
0	0.0	0.0	0.0	0.0	0.000006	0.000002	0.0	9.651429e-07	0.0	0.000000	0.0
1	0.0	0.0	0.0	0.0	0.000223	0.000097	0.0	1.737257e-05	0.0	0.000012	0.0
2	0.0	0.0	0.0	0.0	0.000008	0.000004	0.0	9.651429e-07	0.0	0.000000	0.0
3	0.0	0.0	0.0	0.0	0.000154	0.000087	0.0	1.351200e-05	0.0	0.000009	0.0
4	0.0	0.0	0.0	0.0	0.001196	0.000425	0.0	2.895429e-05	0.0	0.000016	0.0
...	...	...	...	...	...	...	...	...	...	...	...
150555	0.0	0.0	0.0	0.0	0.000011	0.000010	0.0	2.895429e-06	0.0	0.000003	0.0
150556	0.0	0.0	0.0	0.0	0.000004	0.000005	0.0	0.000000e+00	0.0	0.000000	0.0
150557	0.0	0.0	0.0	0.0	0.000003	0.000005	0.0	0.000000e+00	0.0	0.000000	0.0
150558	0.0	0.0	0.0	0.0	0.000004	0.000005	0.0	0.000000e+00	0.0	0.000000	0.0
150559	0.0	0.0	0.0	0.0	0.000003	0.000004	0.0	0.000000e+00	0.0	0.000000	0.0

150560 rows × 11 columns

3) Features Selection

There are 4 features are selected for training and testing including Destination port, Bytes, Bytes Sent and Elapsed Time (sec)

```
sel = SelectFromModel(RandomForestClassifier(n_estimators = 100), max_features=4)
sel.fit(X_resampled, y_resampled)
sel.get_support()
selected_feat= X_resampled.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)
```

```
4
Index(['Destination Port', 'Bytes', 'Bytes Sent', 'Elapsed Time (sec)'], dtype='object')
```

Decision Tree Classification

Data Pipeline for Decision Tree Classification
In the previous section, separated groups of training and testing data are obtained. They are given dataset undergoes preprocessing process through data pipeline including data sampling, feature selection, grid search, classification training model and classification report

```
## create a pipeline
pipeline = Pipeline([
    ('smote', SMOTE(sampling_strategy='auto')),
    ('feature_selection', SelectFromModel(RandomForestClassifier(n_estimators=100), max_features=4)),
    ('classifier', DecisionTreeClassifier())
])

params = {
    'classifier__max_depth': [2, 3, 4, 5, 10, 15, 20],
    'classifier__min_samples_leaf': [5, 10, 20, 50, 100],
    'classifier__criterion': ["gini", "entropy", "log_loss"],
    'classifier__splitter' : ["best", "random"]
}
grid_search = GridSearchCV(estimator=dt,
                           param_grid=params,
                           cv=5, n_jobs=-1, verbose=1, scoring = "accuracy")
grid_search.fit(X_train, y_train)

## fit the pipeline with training data
pipeline.fit(X_train, y_train)

## predict the labels of new data
y_predicted = pipeline.predict(X_test)

## print classification report
print(classification_report(y_test, y_predicted))
```

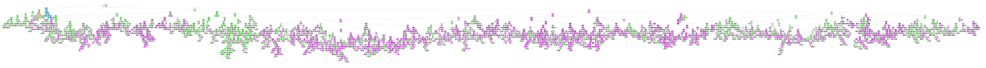
Apply Decision Tree Model for training and testing and compare the results from 2 different sets of parameters including:
1) use all default parameters and
2) use parameters filtered by grid search (best estimators) and apply pruning parameters with cross-validation.

1) Use default parameters The result shows accuracy of 0.99 and weighted average of 0.99 for precision, recall and F1-score which suggest that **the model is overfit**.

```
# 1) Decision Tree Classification using all 'Default' parameters (gini, best splitter, 2 min sample split, 1 min sample leaf)

from sklearn.tree import DecisionTreeClassifier
classifier_initial = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
                                            max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)
classifier_initial.fit(X_trainTree, y_trainTree)
```

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	7526
deny	0.98	0.97	0.98	7503
drop	1.00	1.00	1.00	7528
reset-both	0.98	0.98	0.98	7555
accuracy			0.99	30112
macro avg	0.99	0.99	0.99	30112
weighted avg	0.99	0.99	0.99	30112



This results in a **very complex tree that leads to overfit**. So, we should **adjust the parameters, apply pruning parameters and cross-validation** to help solving overfit.

Apply Grid Search to estimate best parameters for training and testing. The grid search suggests to use 'entropy' criterion, maximum tree depth of 15 and minimum sample leaf of 5. Moreover, we decided to apply ccp_alpha (ccp_alpha = 0.0175 give best-fit for training and testing for this database) to help pruning and reducing complexity of the tree and avoid overfitting.

```
grid_search.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=15, min_samples_leaf=5,
                       random_state=42)
```

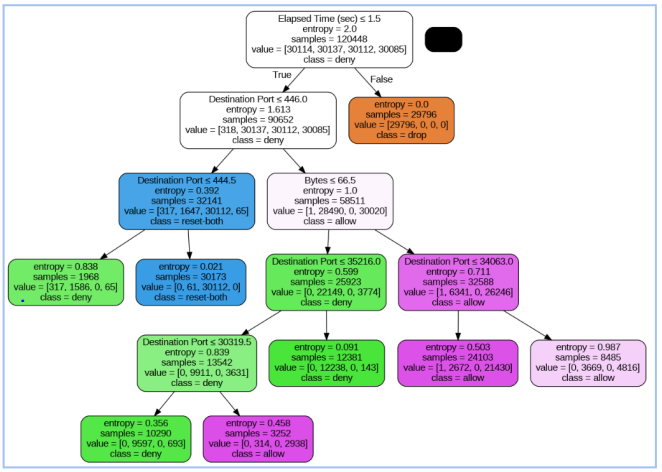
2) Adjust parameters, apply pruning parameters and do cross-validation: After applying best estimated parameters from grid search together with pruning parameter (ccp_alpha), the result shows accuracy of 0.93 and weighted average of 0.94 for precision, 0.93 for recall and F1-score which suggests that the model is not much overfit.

```
from sklearn.tree import DecisionTreeClassifier
classifier_final = DecisionTreeClassifier(criterion='entropy', splitter='best', max_depth = 10, min_samples_split=2, min_samples_leaf= 5, ccp_alpha=0.0175)
classifier_final.fit(X_trainTree, y_trainTree)
```

	precision	recall	f1-score	support
allow	1.00	0.99	0.99	7526
deny	0.95	0.77	0.85	7503
drop	1.00	1.00	1.00	7528
reset-both	0.81	0.97	0.89	7555
accuracy			0.93	30112
macro avg	0.94	0.93	0.93	30112
weighted avg	0.94	0.93	0.93	30112

```
Cross validation scores: [0.93329182 0.93212951 0.93212951 0.93328905 0.93316451]
Average cross-validation score: 0.9328008837988213
```

The cross-validation also suggests an average cross-validation score of approx. 0.93 which is not much overfit. **The model after adjust parameters, apply pruning and cross-validation is getting better fit (best-fit) and the overfitting problem is solved**.



The tree after adjusting and applying pruning parameters is a lot less complex and is better used for training and testing. This suggests that **pruning parameters (e.g. ccp_alpha) is very important for decision tree classification as it can easily overfit the data, particularly when the tree is complex, deep and has too many leaves**.

Random Forest

The steps are as following:

1. Import necessary libraries such as RandomForestClassifier, make_classification, train_test_split, GridSearchCV, accuracy_score, precision_recall_fscore_support, classification_report, and confusion_matrix.
2. A dictionary **param_grid** is defined, which contains a range of values for the hyperparameters of the Random Forest Classifier.
3. An instance of the Random Forest Classifier is created with **random_state** is 42.
4. Create a GridSearchCV object.
5. Train the model on the training data using GridSearchCV to optimize the model's hyperparameters..
6. Define the predicted data using the best estimator by GridSearchCV.
7. Calculate the accuracy of the model.
8. Show the best hyperparameters, accuracy score,and the classification report of the model.

```
# Define the parameter grid for Grid Search
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10],
    'max_features': ['auto', 'sqrt']
}

# Create a Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Create a Grid Search object
grid_search = GridSearchCV(rf, param_grid=param_grid, cv=5)

# Train the model on the training data
grid_search.fit(X_trainBasic, y_trainBasic)

# Predict the classes of the test data using the best estimator found by Grid Search
y_pred = grid_search.best_estimator_.predict(X_testBasic)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_testBasic, y_pred)

# Print the best hyperparameters and accuracy score
print("Best hyperparameters:", grid_search.best_params_)
print("Accuracy:", accuracy)
print(classification_report(y_testBasic, y_pred))
```

Let's see the result of each method

1) Use Random Forest model without EDA and Gridsearch:

- We use a dataset with no EDA (Exploratory Data Analysis). The GridSearchCV function was used to optimize the model's hyperparameters.
- The result shows accuracy of 0.999 ~ 1 and weighted average of 1 for precision, recall and F1-score which suggest that **the model is overfit**.

```
Best hyperparameters: {'max_depth': 10, 'max_features': 'auto', 'n_estimators': 50}
Accuracy: 0.9990844586861982
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00      7545
        deny       1.00      1.00      1.00      2994
        drop       1.00      1.00      1.00      2562
  reset-both       1.00      0.50      0.67         6

    accuracy                           1.00     13107
   macro avg       1.00      0.87      0.92     13107
weighted avg       1.00      1.00      1.00     13107
```

```
Accuracy: 0.9885427736450585
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00      7526
        deny       0.98      0.97      0.98      7503
        drop       1.00      1.00      1.00      7528
  reset-both       0.98      0.98      0.98      7555

    accuracy                           0.99     30112
   macro avg       0.99      0.99      0.99     30112
weighted avg       0.99      0.99      0.99     30112
```

2) Use Random Forest model over EDA data without Gridsearch:

- Using EDA data, so that the model has less bias towards the majority data. The learning has less overfit as accuracy reduces from 1 to 0.9885

2) Use Random Forest model with the data that already EDA and Gridsearch:

- We use a dataset with EDA that are **X_trainTree** and **y_trainTree** (from the EDA with decision tree).
- The result shows accuracy of 0.988 and weighted average of 0.99 for precision, recall and F1-score which suggest that **the model is overfit**.

2.1) define gridsearch by n_estimators, max_depth, max_features

2.2) define gridsearch by n_estimators, max_depth, min_samples_split, max_samples_leaf

```
pipe = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('rf', RandomForestClassifier(random_state=42))
])
# Define the parameter grid for Grid Search
param_grid = {
    'classifier__n_estimators': [50, 100, 150],
    'classifier__max_depth': [None, 5, 10],
    'classifier__max_features': ['auto', 'sqrt']
}

# Create a Grid Search object with the pipeline
grid_search = GridSearchCV(pipe, param_grid=param_grid, cv=5)

# Train the model on the training data
grid_search.fit(X_trainTree, y_trainTree)

# Predict the classes of the test data using the best estimator found by Grid Search
y_pred = grid_search.best_estimator_.predict(X_testTree)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_testTree, y_pred)
# Print the best hyperparameters found by Grid Search, as well as the accuracy of the model
print("Best hyperparameters:", grid_search.best_params_)
print("Accuracy:", accuracy)
print(classification_report(y_testTree, y_pred))
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid=param_grid, cv=5)
grid_search.fit(X_trainTree, y_trainTree)

print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)
```

```
Best hyperparameters: {'classifier__max_depth': None, 'classifier__max_features': 'auto', 'classifier__n_estimators': 50}
Accuracy: 0.9889744952178533
              precision    recall  f1-score   support

       allow       1.00      1.00      1.00      7526
        deny       0.98      0.97      0.98      7503
        drop       1.00      1.00      1.00      7528
  reset-both       0.98      0.98      0.98      7555

    accuracy                           0.99     30112
   macro avg       0.99      0.99      0.99     30112
weighted avg       0.99      0.99      0.99     30112
```

```
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best score: 0.9886507115468666
```

After We see the 1st and 2nd result, the results are too **overfitting**. So, we would like to try to adjust **the parameters by applying pruning parameters and cross-validation** to help solving overfit.

Apply Model for training and testing and compare the results from 3 different sets of parameters including:

3) Use Random Forest model with the data that already EDA + apply pruning parameters to reduce overfitting:

- We use a dataset with EDA that are **X_trainTree** and **y_trainTree** (from the EDA decision tree).
- We apply pruning parameter (ccp_alpha) = 0.0175
- The result shows accuracy of 0.91 and weighted average of 0.91 for precision, recall and F1-score which suggest that **the model is overfit**.

```
# Define the parameter grid for Grid Search
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10],
    'max_features': ['auto', 'sqrt'],
    'ccp_alpha': [0.0175]
}

# Create a Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Create a Grid Search object
grid_search = GridSearchCV(rf, param_grid=param_grid, cv=5)

# Train the model on the training data
grid_search.fit(X_trainTree, y_trainTree)

# Predict the classes of the test data using the best estimator found by Grid Search
y_pred = grid_search.best_estimator_.predict(X_testTree)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_testTree, y_pred)

# Print the best hyperparameters and accuracy score
print("Best hyperparameters:", grid_search.best_params_)
print("Accuracy:", accuracy)
print(classification_report(y_testTree, y_pred))
```

```
Best hyperparameters: {'ccp_alpha': 0.0175, 'max_depth': None, 'max_features': 'auto', 'n_estimators': 50}
Accuracy: 0.9117295430393199
              precision    recall  f1-score   support

       allow       0.99      1.00      0.99      7526
        deny       0.84      0.80      0.82      7503
        drop       1.00      1.00      1.00      7528
  reset-both       0.82      0.85      0.83      7555

    accuracy                           0.91     30112
   macro avg       0.91      0.91      0.91     30112
weighted avg       0.91      0.91      0.91     30112
```

Data pipeline for Random Forest

Data pipeline grouping preprocessing steps together. In this case, we group SMOTE and feature selection steps. Using GridSearch using the command below.

```
pipe = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('rf', RandomForestClassifier(random_state=42))
])
```

```
# Create a Grid Search object with the pipeline
grid_search = GridSearchCV(pipe, param_grid=param_grid, cv=5)
```

XG boost

	precision	recall	f1-score	support
allow	1.00	1.00	1.00	7526
deny	0.98	0.97	0.98	7503
drop	1.00	1.00	1.00	7528
reset-both	0.98	0.98	0.98	7555
accuracy			0.99	30112
macro avg	0.99	0.99	0.99	30112
weighted avg	0.99	0.99	0.99	30112

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

parameters = {
    "loss":["deviance"],
    "learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    "min_samples_split": np.linspace(0.1, 0.5, 12),
    "min_samples_leaf": np.linspace(0.1, 0.5, 12),
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["friedman_mse",  "mae"],
    "subsample":[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],
    "n_estimators":[10]
    }

clf = GridSearchCV(GradientBoostingClassifier(), parameters, cv=10, n_jobs=-1)

clf.fit(X_trainTree, y_trainTree)
print(clf.score(X_trainTree, y_trainTree))
print(clf.best_params_)
```

Apply model for training and testing, and result in only EDA data sets of parameters.

1) Used GridSearchCV to perform hyperparameter tuning for a machine learning model by a dataset with EDA that are **X_trainTree** and **y_trainTree**

- We selected parameters, defined various hyperparameters and their possible values. These hyperparameters include:
- Set GridSearchCV object is created with the GradientBoosting
- And then The result shows best parameters:

```
{'criterion': 'friedman_mse', 'learning_rate': 0.2, 'loss': 'deviance', 'max_depth': 5, 'max_features': 'sqrt', 'min_samples_leaf': 0.1, 'min_samples_split': 0.13636363636363638, 'n_estimators': 10, 'subsample': 1.0}
```

2) Use the result parameters by GridsearchCV in XGboost model.

- We set the parameters according to GridsearchCV's suggestion
- We build a pipeline This pipeline consists of three steps: 1) SMOTE 2) Feature Selection 3) Apply XGboost Classifier.

```
xgboost = GradientBoostingClassifier(loss = 'deviance', learning_rate = 0.2, min_samples_split = 0.13636363636363638,
 min_samples_leaf = 0.1, max_depth = 5
 ,max_features = 'sqrt', criterion = 'friedman_mse', subsample = 1.0, n_estimators = 10)


## create a pipeline
pipeline = Pipeline([
 ('smote', SMOTE(sampling_strategy='auto')),
 ('feature_selection', SelectFromModel(RandomForestClassifier(n_estimators=100), max_features=4)),
 ('classifier', xgboost)
])

## fit the pipeline with training data
pipeline.fit(X_train, y_train)

## predict the labels of new data
y_pred = pipeline.predict(X_test)

## print classification report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
allow	0.99	0.98	0.98	7526
deny	0.85	0.76	0.80	7503
drop	0.96	1.00	0.98	7528
reset-both	0.82	0.87	0.85	7555
accuracy			0.90	30112
macro avg	0.90	0.90	0.90	30112
weighted avg	0.90	0.90	0.90	30112

After applying best estimated parameters from grid search, the result shows accuracy of 0.90 and weighted average of 0.90 for precision, 0.90 for recall and F1-score which suggests that the model is **not much overfit**.

KNN

Apply model for training and testing and compare the results from 2 different sets of parameters including: 1) use all default parameters and 2) use parameters filtered by grid search (best estimators)

```
for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_trainBasic, y_trainBasic)
    y_pred = knn.predict(X_testBasic)
    f1 = classification_report(y_testBasic, y_pred, output_dict=True)['weighted avg']['f1-score']
    print(f"KNN with k={k}: F1-score = {f1:.4f}")
```

1) Use default parameters by a dataset without EDA that X_trainBasic and y_trainBasic**:**

- **We set the parameter only K parameter**

	precision	recall	f1-score	support
allow	1.00	0.99	1.00	7545
deny	0.98	0.99	0.99	2994
drop	1.00	1.00	1.00	2562
reset-both	0.00	0.00	0.00	6
accuracy			0.99	13107
macro avg	0.74	0.75	0.74	13107
weighted avg	0.99	0.99	0.99	13107

After running the code, the result show KNN with k= 5: F1-score = 0.9934 accuracy of 0.99 and weighted average of 0.74 for precision which suggest that **the model is overfit**.

```
KNN with k=5: F1-score = 0.9934
```

```
## create a pipeline
pipeline = Pipeline([
    ('smote', SMOTE(sampling_strategy='auto')),
    ('feature_selection', SelectFromModel(RandomForestClassifier(n_estimators=100), max_features=4)),
    ('classifier', KNeighborsClassifier())
])

#GridSearch Parameter
parameters = {
    "classifier__n_neighbors":[3,5,7,9,11,13,15,17,21],
    "classifier__metric": ["minkowski", "manhattan", "infinity"],
    "classifier__weights": ["uniform","distance"]
    }

grid = GridSearchCV(pipeline, parameters, cv=5, n_jobs=-1, scoring='f1_macro')

## fit the pipeline with training data
grid.fit(X_train, y_train)

## predict the labels of new data
y_pred = grid.predict(X_test)

# print classification report
print(classification_report(y_test, y_pred))
```

2) **Use GridSearchCV to perform hyperparameter tuning for learning model by a dataset with EDA**

- We set the parameters according to GridsearchCV's suggestion
- We build a pipeline This pipeline consists of three steps: 1) SMOTE 2) Feature Selection 3) Apply KNNClassifier.

	precision	recall	f1-score	support
allow	1.00	0.98	0.99	7545
deny	0.98	0.98	0.98	2994
drop	1.00	1.00	1.00	2562
reset-both	0.05	0.83	0.09	6
accuracy			0.99	13107
macro avg	0.76	0.95	0.77	13107
weighted avg	0.99	0.99	0.99	13107

After applying best estimated parameters from grid search, the result shows accuracy of 0.99 and weighted average of 0.77 for precision, 0.99 for recall and F1-score which suggests that the model is still **overfit**.

Summary

1) Summarize the Models

Analyzing the definition of Firewall, the main objection of it is to prevent inappropriate access to the source behind the firewall. Since we need to distinguish the access, we focus on “Recall” in our model evaluation.

Recall of each optimized model

	Decision Tree	Random Forest	XGBoost	KNN
allow	0.99	1.00	0.98	0.98
deny	0.77	0.80	0.76	0.98
drop	1.00	1.00	1.00	1.00
reset-both	0.97	0.85	0.87	0.83
macro avg	**0.93**	**0.91**	**0.90**	**0.95**
weighted avg	**0.93**	**0.91**	**0.90**	**0.99**

From the results above, we can conclude that “Random Forest” is the most suitable model for this case for the following reasons.

- **KNN:** Highest weighted average but the lowest score on reset-both. The score can be overfitting.
- **Decision Tree:** High weighted average but quite low score on “deny” with very high score on “reset-both” which may lead to overfitting.
- **XGBoost:** Similar to Random Forest but weighted average is lower.

2) Compare best model with the results in the reference paper

TABLE III. EVALUATION RESULTS

Method	F_1 Score	Precision	Recall
SVM Linear	75.4	67.5	85.3
SVM Polynomial	53.6	61.8	47.4
SVM RBF	76.4	63.0	97.1
SVM Sigmoid	74.8	60.3	98.5
Random Forest	91.0	91.0	91.0

From the table above, we can conclude that the optimized Random Forest model has more accuracy on Precision and less likely to be overfitted by the score of Recall with high F1 Score.

However, using specific dataset and specific machine learning algorithm in this study can lead to possible limitations including the results may not be generalized for different dataset or log files, the SVM approach might not be the best approach for other classification problems and the classification performance can be depended on other unselected features.