```
### Internet Firewall Data
```

Additional Information

There are 12 features in total. Action feature is used as a class. There are 4 classes in total. These are allow, action, drop and reset-both classes.

```
###!pip install keras-tuner
```

```
###! unzip /content/internet+firewall+data.zip
```

```
###! pip install tensorflow
###! pip install bayesian-optimization
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn import preprocessing
import matplotlib.pyplot as plt
tf.random.set_seed(123)
np.random.seed(123)

# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam, Ftrl
from keras.callbacks import EarlyStopping, ModelCheckpoint
###from keras.wrappers.scikit_learn import KerasClassifier
from math import floor
from sklearn.metrics import make_scorer, accuracy_score
from bayes_opt import BayesianOptimization
from sklearn.model_selection import StratifiedKFold
from keras.layers import LeakyReLU
LeakyReLU = LeakyReLU(alpha=0.1)
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
```

```
log_data = pd.read_csv("/content/log2.csv")
```

```
log_data.columns
```

```
Index(['Source Port', 'Destination Port', 'NAT Source Port',
       'NAT Destination Port', 'Action', 'Bytes', 'Bytes Sent',
       'Bytes Received', 'Packets', 'Elapsed Time (sec)', 'pkts_sent',
       'pkts_received'],
      dtype='object')
```

```
log_data.info()
```

```
<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
```

```
from sklearn.model_selection import train_test_split
```

```
training_data, testing_data = train_test_split(log_data, test_size = 0.3, random_state = 0)
```

```
print(training_data.shape, testing_data.shape)
```

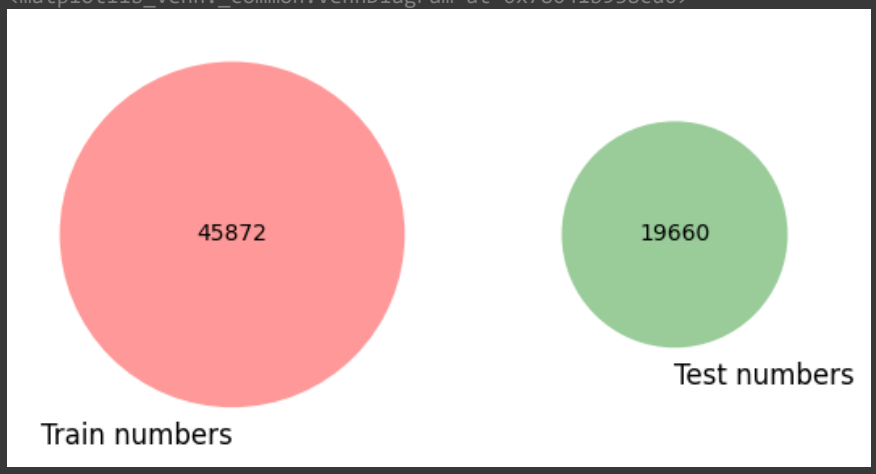
```
(45872, 12) (19660, 12)
```

```
training_data['ID'] = training_data.index
testing_data['ID'] = testing_data.index
```

```
###! pip install matplotlib_venn
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
from matplotlib_venn import venn3, venn3_circles
```

```
set_numbers_train = set(training_data[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
set_numbers_test = set(testing_data[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
venn2((set_numbers_train, set_numbers_test), set_labels = ('Train numbers', 'Test numbers'))
```

```
<matplotlib_venn._common.VennDiagram at 0x78041b938cd0>
```




```
training_data.columns
```

```
Index(['Source Port', 'Destination Port', 'NAT Source Port',
       'NAT Destination Port', 'Action', 'Bytes', 'Bytes Sent',
       'Bytes Received', 'Packets', 'Elapsed Time (sec)', 'pkts_sent',
       'pkts_received', 'ID'],
      dtype='object')
```

```
num_var = [feature for feature in training_data.columns if training_data[feature].dtypes != 'O']
discrete_var = [feature for feature in num_var if len(training_data[feature].unique()) <= 25]
cont_var = [feature for feature in num_var if feature not in discrete_var]
categ_var = [feature for feature in training_data.columns if feature not in num_var]
```

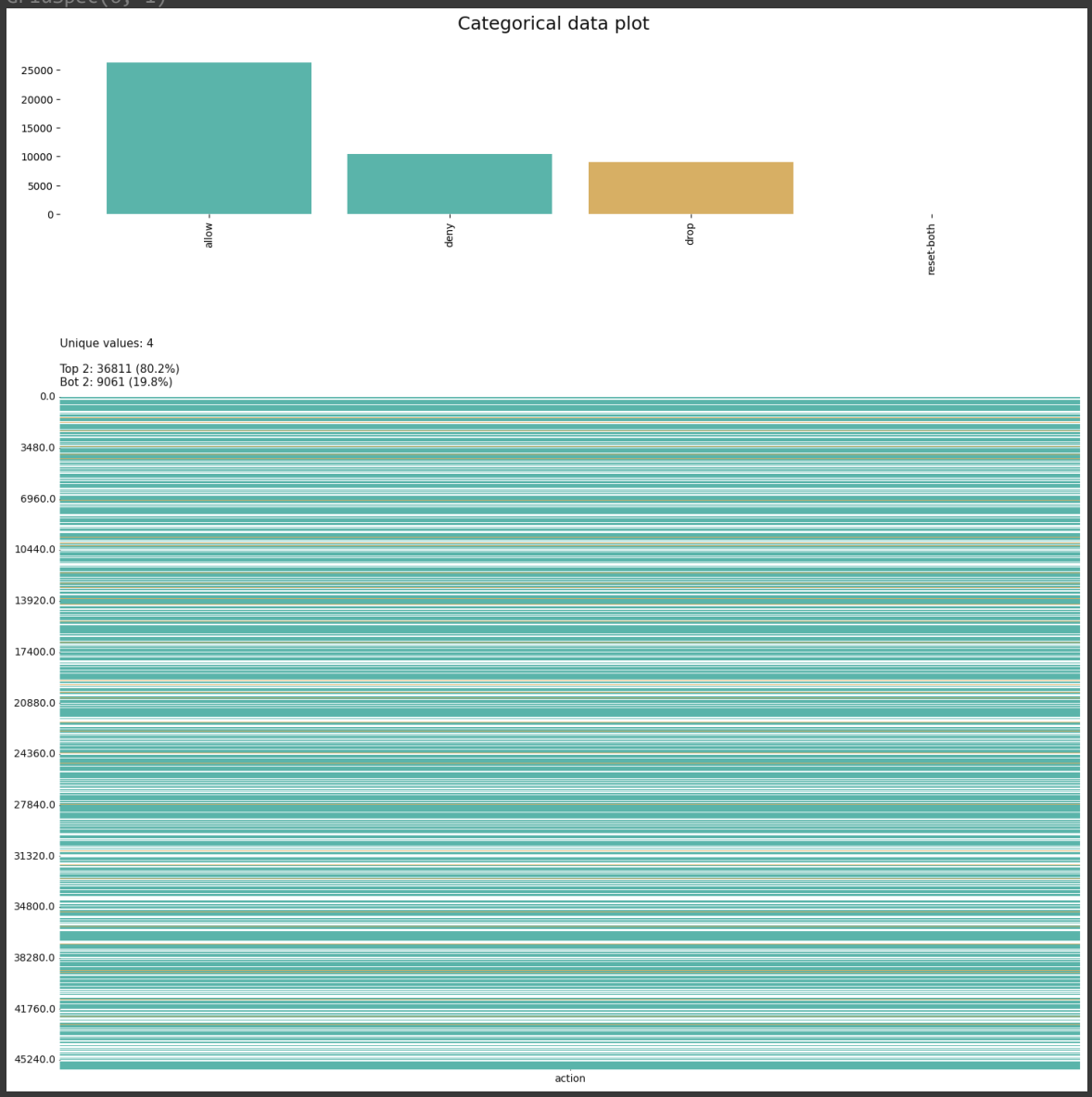
```
###! pip install klib
###!pip install keras-tuner
```

```
import klib
```

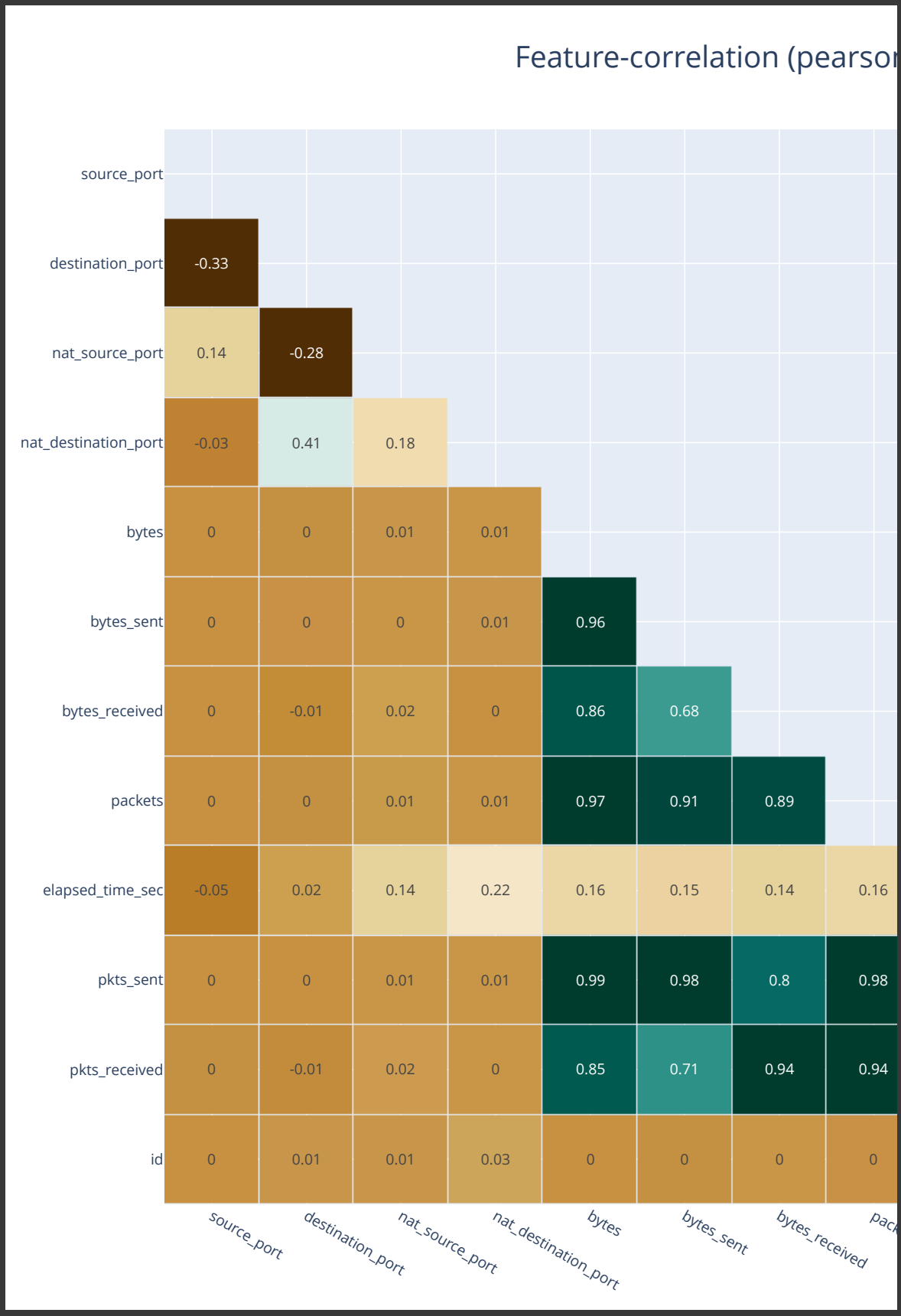
```
training_data = klib.clean_column_names(training_data)
testing_data = klib.clean_column_names(testing_data)
```

```
klib.cat_plot(training_data)
```

```
GridSpec(6, 1)
```

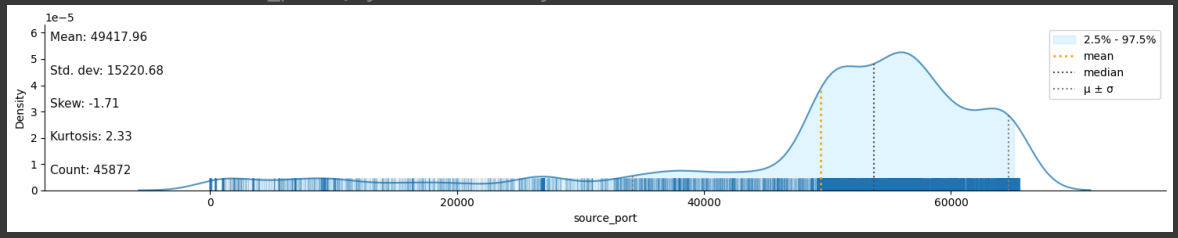



```
klib.corr_interactive_plot(training_data)
```




```
klib.dist_plot(training_data)
```

```
Large dataset detected, using 10000 random samples for the plots. Summary statistics are still
<Axes: xlabel='source_port', ylabel='Density'>
```



```
klib.missingval_plot(training_data)
```

```
No missing values found in the dataset.
```

```
klib.corr_mat(training_data)
```

	source_port	destination_port	nat_source_port	nat_destination_port	bytes
source_port	1.00	-0.33	0.14	-0.03	-0.00
destination_port	-0.33	1.00	-0.28	0.41	-0.00
nat_source_port	0.14	-0.28	1.00	0.18	0.01
nat_destination_port	-0.03	0.41	0.18	1.00	0.01
bytes	-0.00	-0.00	0.01	0.01	1.00
bytes_sent	-0.00	0.00	0.00	0.01	0.96
bytes_received	0.00	-0.01	0.02	-0.00	0.86
packets	-0.00	-0.00	0.01	0.01	0.97
elapsed_time_sec	-0.05	0.02	0.14	0.22	0.16
pkts_sent	-0.00	-0.00	0.01	0.01	0.99
pkts_received	-0.00	-0.01	0.02	0.00	0.85
id	-0.00	0.01	0.01	0.03	-0.00

```
training_data.columns
```

```
Index(['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'action', 'bytes', 'bytes_sent',
       'bytes_received', 'packets', 'elapsed_time_sec', 'pkts_sent',
       'pkts_received', 'id'],
      dtype='object')
```

```
# Checking for outliers in the continuous variables
num_train_dataset = training_data[['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'action', 'bytes', 'bytes_sent',
       'bytes_received', 'packets', 'elapsed_time_sec', 'pkts_sent',
       'pkts_received', 'id']]
```

```
training_data['action'].value_counts()
```

```
action
allow         26311
deny          10500
drop           9021
reset-both       40
Name: count, dtype: int64
```

```
training_data['action'] = training_data['action'].astype('category').cat.codes
testing_data['action'] = testing_data['action'].astype('category').cat.codes
```

```
training_data['action'].value_counts()
```

```
action
0    26311
1    10500
2     9021
3       40
Name: count, dtype: int64
```

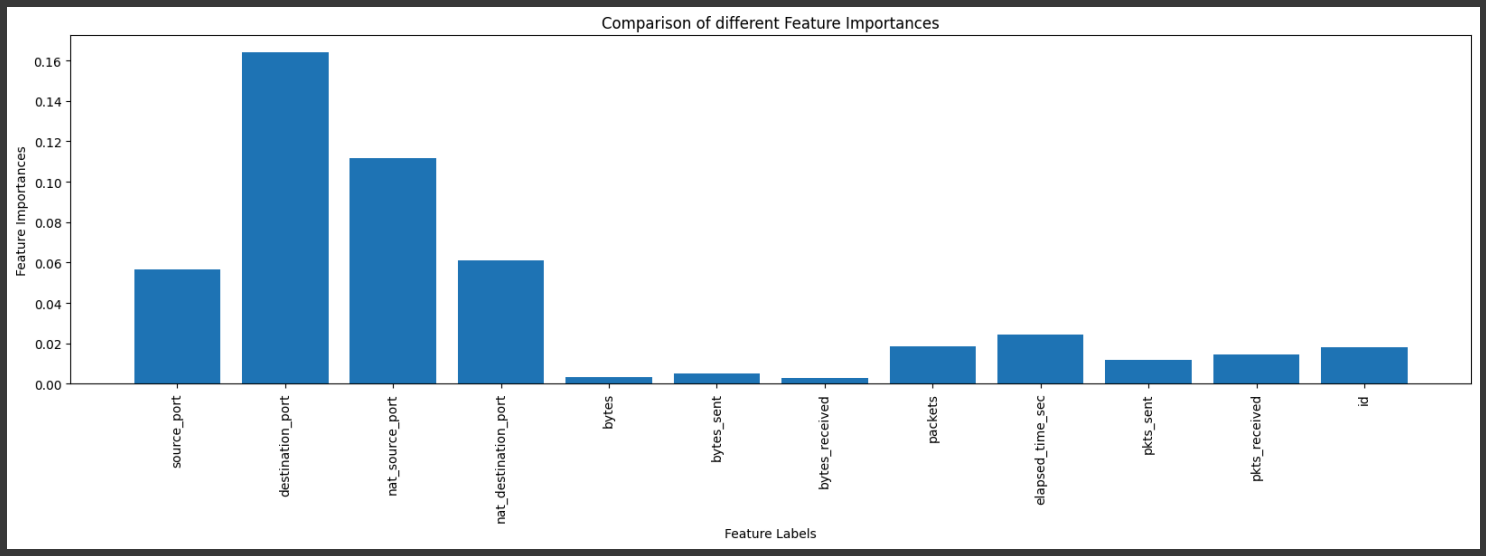
```
y_train = training_data['action']
x_train = training_data.drop('action', axis = 1)
y_test = testing_data['action']
x_test = testing_data.drop('action', axis = 1)
```

```
from sklearn.ensemble import ExtraTreesClassifier
extra_tree_forest = ExtraTreesClassifier(n_estimators = 5,
                                         criterion ='entropy', max_features = 2)
extra_tree_forest.fit(x_train, y_train)
feature_importance = extra_tree_forest.feature_importances_
feature_importance_normalized = np.std([tree.feature_importances_ for tree in
                                         extra_tree_forest.estimators_],
                                         axis = 0)
```

```
feature_importance_normalized
```

```
array([0.05675952, 0.16426633, 0.11188091, 0.06129195, 0.00335718,
       0.00493371, 0.00305258, 0.0187216 , 0.0243437 , 0.01163417,
       0.01434409, 0.01811124])
```

```
plt.figure(figsize = [20,5])
plt.bar(x_train.columns, feature_importance_normalized)
plt.xlabel('Feature Labels')
plt.ylabel('Feature Importances')
plt.xticks(rotation = 90)
plt.title('Comparison of different Feature Importances')
plt.show()
```




```
print("The columns of the x_train :", x_train.columns)
print("The columns of the x_test :", x_test.columns)
```

```
The columns of the x_train : Index(['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'bytes', 'bytes_sent', 'bytes_received',
       'packets', 'elapsed_time_sec', 'pkts_sent', 'pkts_received', 'id'],
      dtype='object')
The columns of the x_test : Index(['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'bytes', 'bytes_sent', 'bytes_received',
       'packets', 'elapsed_time_sec', 'pkts_sent', 'pkts_received', 'id'],
      dtype='object')
```

```
x_train2 = x_train[['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'bytes', 'bytes_sent', 'bytes_received',
       'packets', 'elapsed_time_sec', 'pkts_sent', 'pkts_received']]
```

```
x_test2 = x_test[['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'bytes', 'bytes_sent', 'bytes_received',
       'packets', 'elapsed_time_sec', 'pkts_sent', 'pkts_received']]
```

```
print(x_train2.shape, x_test2.shape)
```

```
(45872, 11) (19660, 11)
```

```
y_train.value_counts()
```

```
action
0    26311
1    10500
2     9021
3       40
Name: count, dtype: int64
```

```
x_train2 = pd.DataFrame(x_train2)
x_test2 = pd.DataFrame(x_test2)
```

Pearson Correlation

```
x_train2.astype(float).corr()
```

	source_port	destination_port	nat_source_port	nat_destination_port	bytes	bytes_sent	bytes_re
source_port	1.000000	-0.332822	0.144624	-0.026874	-0.000384	-0.001174	0.
destination_port	-0.332822	1.000000	-0.281346	0.410776	-0.003970	0.002086	-0.
nat_source_port	0.144624	-0.281346	1.000000	0.176844	0.010345	0.002506	0.
nat_destination_port	-0.026874	0.410776	0.176844	1.000000	0.005874	0.009378	-0.
bytes	-0.000384	-0.003970	0.010345	0.005874	1.000000	0.961022	0.
bytes_sent	-0.001174	0.002086	0.002506	0.009378	0.961022	1.000000	0.
bytes_received	0.001163	-0.014347	0.022666	-0.001877	0.858692	0.683532	1.
packets	-0.002848	-0.004812	0.012151	0.006277	0.974205	0.907204	0.
elapsed_time_sec	-0.045490	0.023451	0.141599	0.219658	0.159429	0.148848	0.
pkts_sent	-0.002032	-0.001526	0.007036	0.007417	0.989239	0.975093	0.
pkts_received	-0.003874	-0.009608	0.019156	0.003822	0.853142	0.707939	0.

```
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

```
print(x_test2.columns, x_train2.columns)
```

```
Index(['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'bytes', 'bytes_sent', 'bytes_received',
       'packets', 'elapsed_time_sec', 'pkts_sent', 'pkts_received'],
      dtype='object') Index(['source_port', 'destination_port', 'nat_source_port',
       'nat_destination_port', 'bytes', 'bytes_sent', 'bytes_received',
       'packets', 'elapsed_time_sec', 'pkts_sent', 'pkts_received'],
      dtype='object')
```

EVAL ML

```
###!pip install evalml
```

```
import evalml
```

```
from sklearn.preprocessing import LabelEncoder
```

```
lbl= LabelEncoder()
```

```
y_test= lbl.fit_transform(y_test)
y_test[:12]
```

```
array([0, 2, 0, 1, 2, 0, 1, 0, 2, 1, 0, 0])
```

```
y_train= lbl.fit_transform(y_train)
y_train[:12]
```

```
array([0, 0, 2, 0, 0, 0, 2, 0, 1, 0, 1, 0])
```

```
evalml.problem_types.ProblemTypes.all_problem_types
```

```
[<ProblemTypes.BINARY: 'binary'>,
 <ProblemTypes.MULTICLASS: 'multiclass'>,
 <ProblemTypes.REGRESSION: 'regression'>,
 <ProblemTypes.TIME_SERIES_REGRESSION: 'time series regression'>,
 <ProblemTypes.TIME_SERIES_BINARY: 'time series binary'>,
 <ProblemTypes.TIME_SERIES_MULTICLASS: 'time series multiclass'>,
 <ProblemTypes.MULTISERIES_TIME_SERIES_REGRESSION: 'multiseries time series regression'>]
```

```
from evalml.automl import AutoMLSearch
automl = AutoMLSearch(X_train=x_train2, y_train=y_train, problem_type='MULTICLASS')
automl.search()
```

```
{1: {'Random Forest Classifier w/ Label Encoder + Imputer + Undersampler + RF Classifier Select From Model':
7.521424770355225,
  'Total time of batch': 7.649293899536133},
 2: {'LightGBM Classifier w/ Label Encoder + Imputer + Undersampler + Select Columns Transformer': 4.330684185028076,
  'Extra Trees Classifier w/ Label Encoder + Imputer + Undersampler + Select Columns Transformer': 2.6364517211914062,
  'Elastic Net Classifier w/ Label Encoder + Imputer + Undersampler + Standard Scaler + Select Columns Transformer':
16.121755599975586,
  'XGBoost Classifier w/ Label Encoder + Imputer + Undersampler + Select Columns Transformer': 9.915470600128174,
  'Logistic Regression Classifier w/ Label Encoder + Imputer + Undersampler + Standard Scaler + Select Columns
Transformer': 5.948143005371094,
  'Total time of batch': 39.58880019187927}}
```

```
automl.rankings
```

	id	pipeline_name	search_order	ranking_score	mean_cv_score	standard_deviation_cv_score	percent_better_than_base
0	5	XGBoost Classifier w/ Label Encoder + Imputer ...	5	0.008207	0.008207	0.000291	99.94
1	2	LightGBM Classifier w/ Label Encoder + Imputer...	2	0.010703	0.010703	0.001802	99.93
2	1	Random Forest Classifier w/ Label Encoder + Im...	1	0.017463	0.017463	0.002832	99.88
3	6	Logistic Regression Classifier w/ Label Encode...	6	0.104168	0.104168	0.000655	99.32
4	4	Elastic Net Classifier w/ Label Encoder + Impu...	4	0.278498	0.278498	0.007485	98.18
5	3	Extra Trees Classifier w/ Label Encoder + Impu...	3	0.319827	0.319827	0.001308	97.91
6	0	Mode Baseline Multiclass Classification Pipeline	0	15.369940	15.369940	0.001183	0.00

```
automl.best_pipeline
```

```
pipeline = MulticlassClassificationPipeline(component_graph={'Label Encoder': ['Label Encoder', 'X', 'y'], 'Imputer':
['Imputer', 'X', 'Label Encoder.y'], 'Undersampler': ['Undersampler', 'Imputer.x', 'Label Encoder.y'], 'Select Columns
Transformer': ['Select Columns Transformer', 'Undersampler.x', 'Undersampler.y'], 'XGBoost Classifier': ['XGBoost
Classifier', 'Select Columns Transformer.x', 'Undersampler.y']}, parameters={'Label Encoder':{'positive_label': None},
'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean',
'boolean_impute_strategy': 'most_frequent', 'categorical_fill_value': None, 'numeric_fill_value': None,
'boolean_fill_value': None}, 'Undersampler':{'sampling_ratio': 0.25, 'min_samples': 100, 'min_percentage': 0.1,
'sampling_ratio_dict': None}, 'Select Columns Transformer':{'columns': ['destination_port', 'nat_source_port',
'bytes', 'bytes_sent', 'packets', 'elapsed_time_sec']}, 'XGBoost Classifier':{'eta': 0.1, 'max_depth': 6,
'min_child_weight': 1, 'n_estimators': 100, 'n_jobs': -1, 'eval_metric': 'logloss'}}, random_seed=0)
```

```
best_pipeline=automl.best_pipeline
```

```
automl.describe_pipeline(automl.rankings.iloc[0]["id"])
```

```
INFO:evalml.pipelines.component_graph.describe:3. Undersampler
        * sampling_ratio : 0.25
INFO:evalml.pipelines.components.component_base.describe:        * sampling_ratio : 0.25
        * min_samples : 100
INFO:evalml.pipelines.components.component_base.describe:        * min_samples : 100
        * min_percentage : 0.1
INFO:evalml.pipelines.components.component_base.describe:        * min_percentage : 0.1
        * sampling_ratio_dict : None
INFO:evalml.pipelines.components.component_base.describe:        * sampling_ratio_dict : None
4. Select Columns Transformer
INFO:evalml.pipelines.component_graph.describe:4. Select Columns Transformer
        * columns : ['destination_port', 'nat_source_port', 'bytes', 'bytes_sent', 'packets', 'elapsed_time_sec']
INFO:evalml.pipelines.components.component_base.describe:        * columns : ['destination_port', 'nat_source_port',
5. XGBoost Classifier
INFO:evalml.pipelines.component_graph.describe:5. XGBoost Classifier
        * eta : 0.1
INFO:evalml.pipelines.components.component_base.describe:        * eta : 0.1
        * max_depth : 6
INFO:evalml.pipelines.components.component_base.describe:        * max_depth : 6
        * min_child_weight : 1
INFO:evalml.pipelines.components.component_base.describe:        * min_child_weight : 1
        * n_estimators : 100
INFO:evalml.pipelines.components.component_base.describe:        * n_estimators : 100
        * n_jobs : -1
INFO:evalml.pipelines.components.component_base.describe:        * n_jobs : -1
        * eval_metric : logloss
INFO:evalml.pipelines.components.component_base.describe:        * eval_metric : logloss

INFO:evalml.automl.automl_search.describe_pipeline:
Training
INFO:evalml.automl.automl_search.describe_pipeline:Training
========
INFO:evalml.automl.automl_search.describe_pipeline:========
Training for multiclass problems.
INFO:evalml.automl.automl_search.describe_pipeline:Training for multiclass problems.
Total training time (including CV): 9.9 seconds
INFO:evalml.automl.automl_search.describe_pipeline:Total training time (including CV): 9.9 seconds

INFO:evalml.automl.automl_search.describe_pipeline:
Cross Validation
INFO:evalml.automl.automl_search.describe_pipeline:Cross Validation
----------------
INFO:evalml.automl.automl_search.describe_pipeline:----------------
             Log Loss Multiclass  MCC Multiclass  AUC Weighted  AUC Macro  AUC Micro  Precision Weighted  Precision M
0                          0.008           0.998         1.000      0.990      1.000               0.999            0
1                          0.008           0.997         1.000      0.997      1.000               0.998            0
2                          0.008           0.998         1.000      0.996      1.000               0.999            0
mean                       0.008           0.998         1.000      0.994      1.000               0.999            0
std                        0.000           0.000         0.000      0.003      0.000               0.000            0
coef of var                0.035           0.000         0.000      0.003      0.000               0.000            0
INFO:evalml.automl.automl_search.describe_pipeline:             Log Loss Multiclass  MCC Multiclass  AUC Weighted  AU
0                          0.008           0.998         1.000      0.990      1.000               0.999            0
1                          0.008           0.997         1.000      0.997      1.000               0.998            0
2                          0.008           0.998         1.000      0.996      1.000               0.999            0
mean                       0.008           0.998         1.000      0.994      1.000               0.999            0
std                        0.000           0.000         0.000      0.003      0.000               0.000            0
coef of var                0.035           0.000         0.000      0.003      0.000               0.000            0
```

```
evalml.problem_types.ProblemTypes.all_problem_types
```

```
[<ProblemTypes.BINARY: 'binary'>,
 <ProblemTypes.MULTICLASS: 'multiclass'>,
 <ProblemTypes.REGRESSION: 'regression'>,
 <ProblemTypes.TIME_SERIES_REGRESSION: 'time series regression'>,
 <ProblemTypes.TIME_SERIES_BINARY: 'time series binary'>,
 <ProblemTypes.TIME_SERIES_MULTICLASS: 'time series multiclass'>,
 <ProblemTypes.MULTISERIES_TIME_SERIES_REGRESSION: 'multiseries time series regression'>]
```

```
from evalml.objectives import get_optimization_objectives
```

```
from evalml.problem_types import ProblemTypes

for objective in get_optimization_objectives(ProblemTypes.MULTICLASS):
    print(objective.name)
```

```
MCC Multiclass
Log Loss Multiclass
AUC Weighted
AUC Macro
AUC Micro
Precision Weighted
Precision Macro
Precision Micro
F1 Weighted
F1 Macro
F1 Micro
Balanced Accuracy Multiclass
Accuracy Multiclass
```

```
automl_auc = AutoMLSearch(X_train=x_train2, y_train=y_train,
                          problem_type='multiclass',
                          objective='F1 Weighted',
                          additional_objectives=['Balanced Accuracy Multiclass', 'Accuracy Multiclass'],
                          max_batches=1,
                          optimize_thresholds=True)

automl_auc.search()
```

```
{1: {'Random Forest Classifier w/ Label Encoder + Imputer + Undersampler + RF Classifier Select From Model':
6.218914270401001,
  'Total time of batch': 6.40790581703186}}
```

```
automl_auc.rankings
```

	id	pipeline_name	search_order	ranking_score	mean_cv_score	standard_deviation_cv_score	percent_better_than_base
0	1	Random Forest Classifier w/ Label Encoder + Im...	1	0.998302	0.998302	0.000325	58.01
1	0	Mode Baseline Multiclass Classification Pipeline	0	0.418140	0.418140	0.000039	0.00

```
automl_auc.describe_pipeline(automl_auc.rankings.iloc[0]["id"])
```

```
INFO:evalml.pipelines.components.component_base.describe:        * max_depth : None
        * percent_features : 0.5
INFO:evalml.pipelines.components.component_base.describe:        * percent_features : 0.5
        * threshold : median
INFO:evalml.pipelines.components.component_base.describe:        * threshold : median
        * n_jobs : -1
INFO:evalml.pipelines.components.component_base.describe:        * n_jobs : -1
5. Random Forest Classifier
INFO:evalml.pipelines.component_graph.describe:5. Random Forest Classifier
        * n_estimators : 100
INFO:evalml.pipelines.components.component_base.describe:        * n_estimators : 100
        * max_depth : 6
INFO:evalml.pipelines.components.component_base.describe:        * max_depth : 6
        * n_jobs : -1
INFO:evalml.pipelines.components.component_base.describe:        * n_jobs : -1

INFO:evalml.automl.automl_search.describe_pipeline:
Training
INFO:evalml.automl.automl_search.describe_pipeline:Training
========
INFO:evalml.automl.automl_search.describe_pipeline:========
Training for multiclass problems.
INFO:evalml.automl.automl_search.describe_pipeline:Training for multiclass problems.
Total training time (including CV): 6.2 seconds
INFO:evalml.automl.automl_search.describe_pipeline:Total training time (including CV): 6.2 seconds

INFO:evalml.automl.automl_search.describe_pipeline:
Cross Validation
INFO:evalml.automl.automl_search.describe_pipeline:Cross Validation
---------------
INFO:evalml.automl.automl_search.describe_pipeline:---------------
             F1 Weighted  Balanced Accuracy Multiclass  Accuracy Multiclass # Training # Validation
0                  0.998                         0.769                0.998     30,581       15,291
1                  0.998                         0.785                0.998     30,581       15,291
2                  0.999                         0.865                0.999     30,582       15,290
mean               0.998                         0.806                0.999          -            -
std                0.000                         0.051                0.000          -            -
coef of var        0.000                         0.064                0.000          -            -
INFO:evalml.automl.automl_search.describe_pipeline:             F1 Weighted  Balanced Accuracy Multiclass  Accuracy M
0                  0.998                         0.769                0.998     30,581       15,291
1                  0.998                         0.785                0.998     30,581       15,291
2                  0.999                         0.865                0.999     30,582       15,290
mean               0.998                         0.806                0.999          -            -
std                0.000                         0.051                0.000          -            -
coef of var        0.000                         0.064                0.000          -            -
```

```
best_pipeline_auc = automl_auc.best_pipeline
```

```
# get the score on holdout data
best_pipeline_auc.score(x_test2, y_test,  objectives=["F1 Weighted"])
```

```
OrderedDict([('F1 Weighted', 0.9979879263564811)])
```

```
best_pipeline.save("internet_firewall_data_pipelines.pkl")
```

```
final_model=best_pipeline_auc.load('internet_firewall_data_pipelines.pkl')
```

```
final_model.predict_proba(x_test2)
```

	0	1	2	3
13378	0.999932	0.000024	0.000023	0.000021
26886	0.000141	0.002669	0.997051	0.000140
5340	0.999917	0.000032	0.000023	0.000028
20300	0.001252	0.998110	0.000216	0.000421
57848	0.000141	0.002669	0.997051	0.000140

```
y_pred = final_model.predict(x_test2)
```

```
y_pred = pd.DataFrame(y_pred)
```

```
y_pred.value_counts()
```

```
0    11327
1     4485
2     3845
3        3
Name: count, dtype: int64
```

```
y_train = pd.DataFrame(y_train)
```

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred, labels=[0, 1, 2, 3]))
```

```
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00     11329
               1       1.00      1.00      1.00      4487
               2       1.00      1.00      1.00      3830
               3       1.00      0.21      0.35        14

        accuracy                           1.00     19660
       macro avg       1.00      0.80      0.84     19660
    weighted avg       1.00      1.00      1.00     19660
```

```
output = {
 0: 'allow', 1: 'deny', 2: 'drop', 3: 'reset-both'
 }
```

```
y_pred = pd.DataFrame(y_pred)
```

```
y_pred.rename(columns = {0:"Label"}, inplace=True)
```

```
y_pred = y_pred["Label"].map(output)
```

```
y_pred.value_counts()
```

```
Label
allow         11327
deny           4485
drop           3845
reset-both        3
Name: count, dtype: int64
```