Section 1: Declare the Modules

```
import os
from collections import defaultdict
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, precision_score, recall_score, classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import time


import warnings
warnings.filterwarnings('ignore')
```

Section 2: Data Import and Preprocess

```
!pip install wget
import wget

link_to_data = 'https://raw.githubusercontent.com/SIT719/2020-S2/master/data/Week_5_NSL-KDD-Dataset/training_attack_types.txt?raw=true'
DataSet = wget.download(link_to_data)
```

```
    Requirement already satisfied: wget in /usr/local/lib/python3.10/dist-packages (3.2)
```

```
DataSet
```

```
    'training_attack_types (4).txt'
```

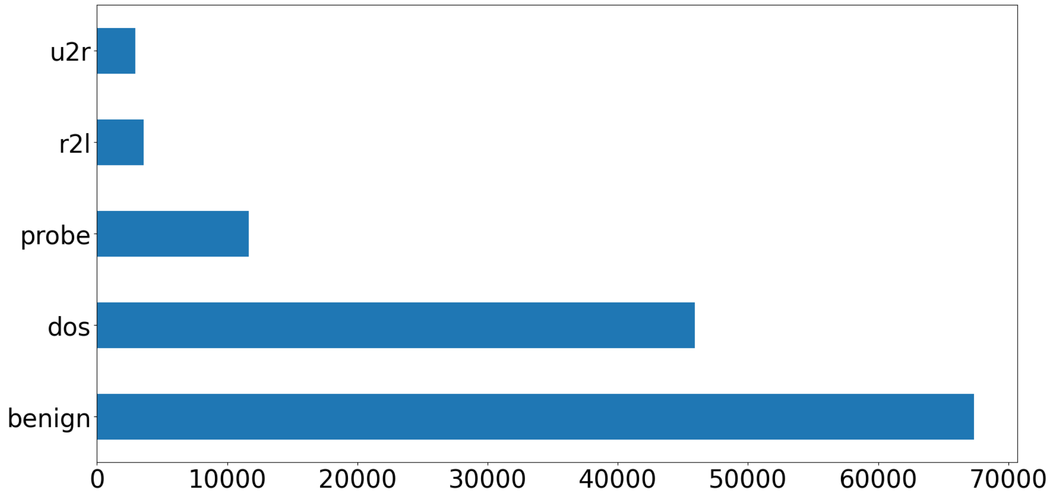
```
header_names = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot', 'num


# Differentiating between nominal, binary, and numeric features

# root_shell is marked as a continuous feature in the kddcup.names
# file, but it is supposed to be a binary feature according to the
# dataset documentation

# training_attack_types.txt maps each of the 22 different attacks to 1 of 4 categories
# file obtained from http://kdd.ics.uci.edu/databases/kddcup99/training_attack_types

col_names = np.array(header_names)

nominal_idx = [1, 2, 3]
binary_idx = [6, 11, 13, 14, 20, 21]
numeric_idx = list(set(range(41)).difference(nominal_idx).difference(binary_idx))

nominal_cols = col_names[nominal_idx].tolist()
binary_cols = col_names[binary_idx].tolist()
numeric_cols = col_names[numeric_idx].tolist()


# training_attack_types.txt maps each of the 22 different attacks to 1 of 4 categories
# file obtained from http://kdd.ics.uci.edu/databases/kddcup99/training_attack_types

category = defaultdict(list)
category['benign'].append('normal')

with open(DataSet, 'r') as f:
    for line in f.readlines():
        attack, cat = line.strip().split(' ')
        category[cat].append(attack)

attack_mapping = dict((v,k) for k in category for v in category[k])
```

```
attack_mapping
```

```
{'normal': 'benign',
 'apache2': 'dos',
 'back': 'dos',
 'mailbomb': 'dos',
 'processtable': 'dos',
 'snmpgetattack': 'dos',
 'teardrop': 'dos',
 'smurf': 'dos',
 'land': 'dos',
 'neptune': 'dos',
 'pod': 'dos',
 'udpstorm': 'dos',
 'ps': 'u2r',
 'buffer_overflow': 'u2r',
 'perl': 'u2r',
 'rootkit': 'u2r',
 'loadmodule': 'u2r',
 'xterm': 'u2r',
 'sqlattack': 'u2r',
 'httptunnel': 'u2r',
 'ftp_write': 'r2l',
 'guess_passwd': 'r2l',
 'snmpguess': 'r2l',
 'imap': 'r2l',
 'spy': 'r2l',
 'warezclient': 'r2l',
 'warezmaster': 'r2l',
 'multihop': 'r2l',
 'phf': 'r2l',
 'named': 'r2l',
 'sendmail': 'r2l',
 'xlock': 'r2l',
 'xsnoop': 'r2l',
 'worm': 'probe',
 'nmap': 'probe',
 'ipsweep': 'probe',
 'portsweep': 'probe',
 'satan': 'probe',
 'mscan': 'probe',
 'saint': 'probe'}
```

```
#Processing Training Data

train_file='https://raw.githubusercontent.com/SIT719/2020-S2/master/data/Week_5_NSL-KDD-Dataset/KDDTrain%2B.txt'



train_df = pd.read_csv(train_file, names=header_names)

train_df['attack_category'] = train_df['attack_type'].map(lambda x: attack_mapping[x])

train_df.drop(['success_pred'], axis=1, inplace=True)


#Processing test Data
test_file='https://raw.githubusercontent.com/SIT719/2020-S2/master/data/Week_5_NSL-KDD-Dataset/KDDTest%2B.txt'

test_df = pd.read_csv(test_file, names=header_names)
test_df['attack_category'] = test_df['attack_type'].map(lambda x: attack_mapping[x])
test_df.drop(['success_pred'], axis=1, inplace=True)


train_attack_types = train_df['attack_type'].value_counts()
train_attack_cats = train_df['attack_category'].value_counts()

test_attack_types = test_df['attack_type'].value_counts()
test_attack_cats = test_df['attack_category'].value_counts()

train_attack_types.plot(kind='barh', figsize=(20,10), fontsize=20)

train_attack_cats.plot(kind='barh', figsize=(20,10), fontsize=30)

train_df[binary_cols].describe().transpose()
train_df.groupby(['su_attempted']).size()
train_df['su_attempted'].replace(2, 0, inplace=True)
test_df['su_attempted'].replace(2, 0, inplace=True)
train_df.groupby(['su_attempted']).size()
train_df.groupby(['num_outbound_cmds']).size()

#Now, that's not a very useful feature - let's drop it from the dataset

train_df.drop('num_outbound_cmds', axis = 1, inplace=True)
test_df.drop('num_outbound_cmds', axis = 1, inplace=True)
numeric_cols.remove('num_outbound_cmds')
```

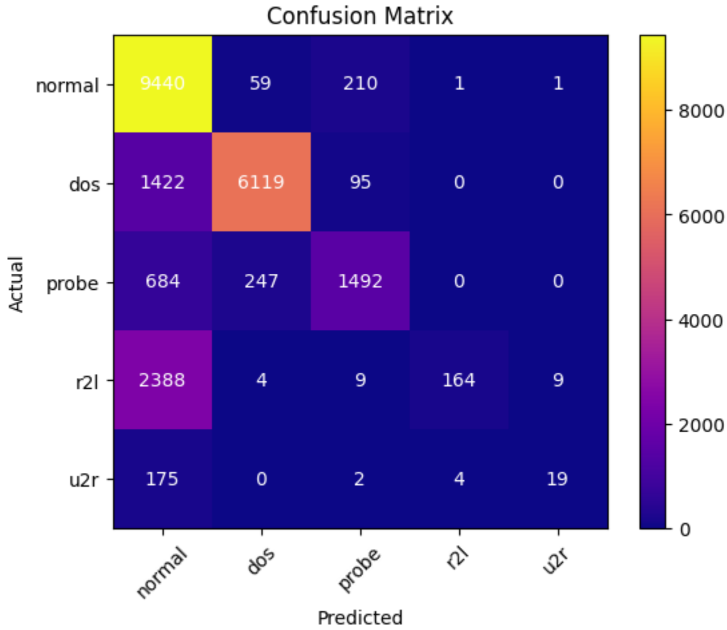
```
#Data Preparation

train_Y = train_df['attack_category']
train_x_raw = train_df.drop(['attack_category','attack_type'], axis=1)
test_Y = test_df['attack_category']
test_x_raw = test_df.drop(['attack_category','attack_type'], axis=1)


combined_df_raw = pd.concat([train_x_raw, test_x_raw])
combined_df = pd.get_dummies(combined_df_raw, columns=nominal_cols, drop_first=True)

train_x = combined_df[:len(train_x_raw)]
test_x = combined_df[len(train_x_raw):]

# Store dummy variable feature names
dummy_variables = list(set(train_x)-set(combined_df_raw))

#execute the commands in console
train_x.describe()
train_x['duration'].describe()
# Experimenting with StandardScaler on the single 'duration' feature
from sklearn.preprocessing import StandardScaler

durations = train_x['duration'].values.reshape(-1, 1)
standard_scaler = StandardScaler().fit(durations)
scaled_durations = standard_scaler.transform(durations)
pd.Series(scaled_durations.flatten()).describe()

# Experimenting with MinMaxScaler on the single 'duration' feature
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler().fit(durations)
min_max_scaled_durations = min_max_scaler.transform(durations)
pd.Series(min_max_scaled_durations.flatten()).describe()

# Experimenting with RobustScaler on the single 'duration' feature
from sklearn.preprocessing import RobustScaler

min_max_scaler = RobustScaler().fit(durations)
robust_scaled_durations = min_max_scaler.transform(durations)
pd.Series(robust_scaled_durations.flatten()).describe()

# Experimenting with MaxAbsScaler on the single 'duration' feature
from sklearn.preprocessing import MaxAbsScaler

max_Abs_scaler = MaxAbsScaler().fit(durations)
robust_scaled_durations = max_Abs_scaler.transform(durations)
pd.Series(robust_scaled_durations.flatten()).describe()

# Let's proceed with StandardScaler- Apply to all the numeric columns

standard_scaler = StandardScaler().fit(train_x[numeric_cols])

train_x[numeric_cols] = \
    standard_scaler.transform(train_x[numeric_cols])

test_x[numeric_cols] = \
    standard_scaler.transform(test_x[numeric_cols])

train_x.describe()



train_Y_bin = train_Y.apply(lambda x: 0 if x is 'benign' else 1)
test_Y_bin = test_Y.apply(lambda x: 0 if x is 'benign' else 1)
```




```
def evaluate_classifier(class_type, train_x, train_Y, test_x, test_Y, label_names=None, **kwargs):
    classifier = class_type(**kwargs)
    classifier.fit(train_x, train_Y)

   # Record start time for training
    start_train_time = time.time()
    classifier.fit(train_x, train_Y)
    end_train_time = time.time()
    train_time = end_train_time - start_train_time  # Calculate training time

    # Make predictions on the test data
    start_test_time = time.time()
    y_pred = classifier.predict(test_x)
    end_test_time = time.time()
    test_time = end_test_time - start_test_time  # Calculate test time

    y_pred = classifier.predict(test_x)
    class_name = classifier.__class__.__name__
    print(f"Classifier: {class_name}")
    print(f"Evaluation Report for {class_name}:")
    show_evaluation_results(test_Y, y_pred, label_names, train_time, test_time)
    print("=" * 40)


performance_metrics = []


def show_evaluation_results(test_Y, y_pred, label_names, train_time, test_time):
    accuracy = accuracy_score(test_Y, y_pred)
    conf_matx = confusion_matrix(test_Y, y_pred)
    f1score = f1_score(test_Y, y_pred, average="macro")
    precision = precision_score(test_Y, y_pred, average="macro")
    recall = recall_score(test_Y, y_pred, average="macro")

    print(f"F-Score: {f1score}")
    print(f"Precision: {precision}")
    print(f"Re-call: {recall}")
    print(f"Accuracy: {accuracy}")
    print(f"Confusion Matrix:\n{conf_matx}")

    clrp = classification_report(test_Y, y_pred, target_names=label_names)
    print(clrp)

    class_far = calculate_false_alarm_rate(conf_matx, label_names)
    for label_name, false_alarm in class_far.items():
        print(f"False Alarm of {label_name}: {false_alarm:.4f} ({false_alarm * 100:.2f}%)")

    overall_far = sum(class_far.values()) / len(class_far)
    print(f"Overall False Alarm Rate: {overall_far:.4f} ({overall_far * 100:.2f}%)")

    print(f"Training Time: {train_time:.4f} seconds")  # Print training time
    print(f"Test Time: {test_time:.4f} seconds")  # Print test time
    total_time = train_time + test_time
    print(f"Total Time: {total_time:.4f} seconds")

    # Calculate the error rate
    error_rate = 1 - accuracy
    print(f"Error Rate: {error_rate:.4f} ({error_rate * 100:.2f}%)")

    # Create the heatmap using Matplotlib
    plt.imshow(conf_matx, interpolation='nearest', cmap='plasma')
```

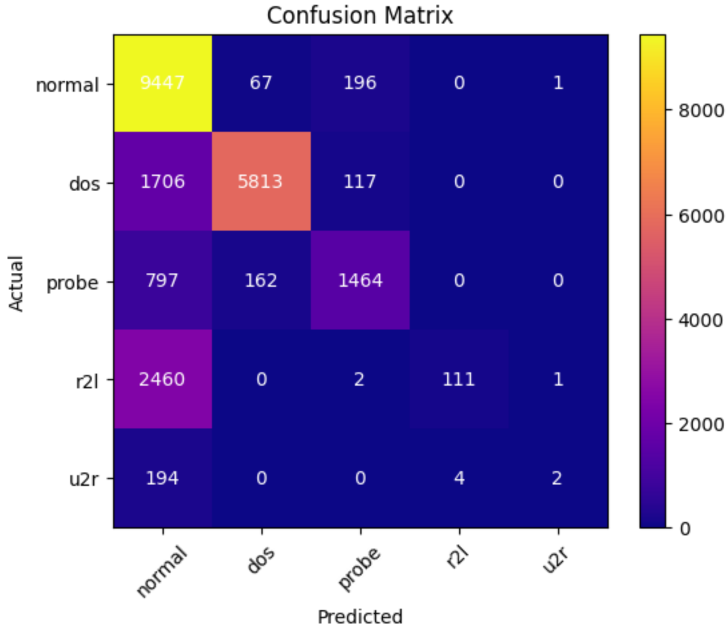
```
        plt.title("Confusion Matrix")
        plt.colorbar()

        # Label the axes with numbers
        for i in range(len(label_names)):
            for j in range(len(label_names)):
                plt.text(j, i, str(conf_matx[i, j]), ha='center', va='center', color='white')

        # Label the axes
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.xticks(np.arange(len(label_names)), label_names, rotation=45)
        plt.yticks(np.arange(len(label_names)), label_names)

        # Append performance metrics to the list
        performance_metrics.append({
            'F-Score': f1score,
            'Precision': precision,
            'Recall': recall,
            'Accuracy': accuracy,
            'Overall FAR': overall_far,
            'Training Time': train_time,
            'Test Time': test_time,
            'Total Time': total_time,
            'Error Rate': error_rate
        })

        plt.show()

    def calculate_false_alarm_rate(confusion_matrix, label_names):
        class_far = {}
        for i in range(len(label_names)):
            false_alarms = sum(confusion_matrix[j][i] for j in range(len(label_names)) if j != i)
            class_far[label_names[i]] = false_alarms / sum(confusion_matrix[i])
        return class_far


    # Define label_names based on your dataset
    label_names = ['normal', 'dos', 'probe', 'r2l', 'u2r']

    classifiers = [
        DecisionTreeClassifier,
        RandomForestClassifier,
        SVC,
        KNeighborsClassifier,
        LogisticRegression,
    ]

    # Then, pass label_names as an argument when calling the analysis function
    for classifier in classifiers:
        evaluate_classifier(classifier, train_x, train_Y, test_x, test_Y, label_names=label_names)
```

```
Classifier: DecisionTreeClassifier
Evaluation Report for DecisionTreeClassifier:
F-Score: 0.5307001870846837
Precision: 0.8143330644602192
Re-call: 0.5095817847363824
Accuracy: 0.7644606103619588
Confusion Matrix:
[[9440   59  210    1    1]
 [1422 6119   95    0    0]
 [ 684  247 1492    0    0]
 [2388    4    9  164    9]
 [ 175    0    2    4   19]]
              precision    recall  f1-score   support

      normal       0.67      0.97      0.79      9711
         dos       0.95      0.80      0.87      7636
       probe       0.83      0.62      0.71      2423
         r2l       0.97      0.06      0.12      2574
         u2r       0.66      0.10      0.17       200

    accuracy                           0.76     22544
   macro avg       0.81      0.51      0.53     22544
weighted avg       0.82      0.76      0.73     22544

False Alarm of normal: 0.4808 (48.08%)
False Alarm of dos: 0.0406 (4.06%)
False Alarm of probe: 0.1304 (13.04%)
False Alarm of r2l: 0.0019 (0.19%)
False Alarm of u2r: 0.0500 (5.00%)
Overall False Alarm Rate: 0.1408 (14.08%)
Training Time: 6.0219 seconds
Test Time: 0.0220 seconds
Total Time: 6.0440 seconds
Error Rate: 0.2355 (23.55%)
```

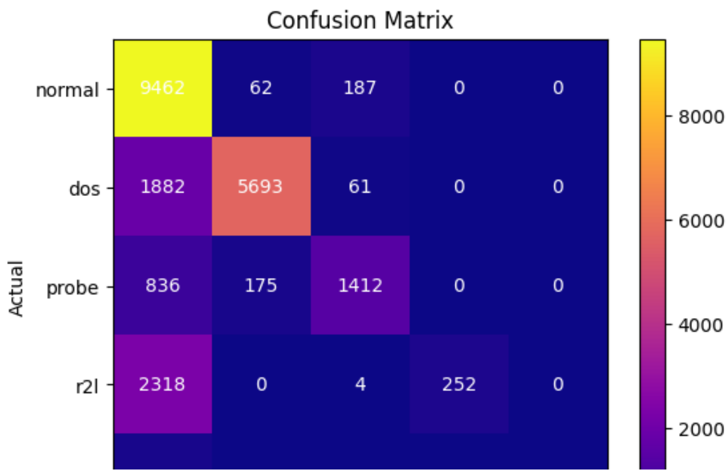


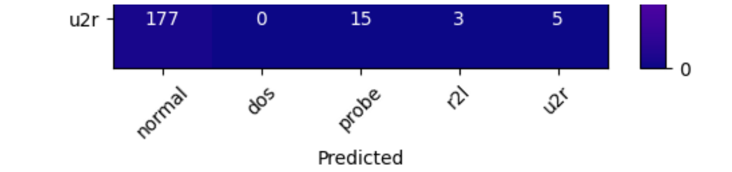

```
========================================
Classifier: RandomForestClassifier
Evaluation Report for RandomForestClassifier:
F-Score: 0.48520134547757243
Precision: 0.7794255669285004
Re-call: 0.4782819951803451
Accuracy: 0.7468506032647267
Confusion Matrix:
[[9447   67  196    0    1]
 [1706 5813  117    0    0]
 [ 797  162 1464    0    0]
 [2460    0    2  111    1]
 [ 194    0    0    4    2]]
              precision    recall  f1-score   support

      normal       0.65      0.97      0.78      9711
         dos       0.96      0.76      0.85      7636
       probe       0.82      0.60      0.70      2423
         r2l       0.97      0.04      0.08      2574
         u2r       0.50      0.01      0.02       200

    accuracy                           0.75     22544
   macro avg       0.78      0.48      0.49     22544
weighted avg       0.81      0.75      0.71     22544

False Alarm of normal: 0.5310 (53.10%)
False Alarm of dos: 0.0300 (3.00%)
False Alarm of probe: 0.1300 (13.00%)
```

```
False Alarm of r2l: 0.0016 (0.16%)
False Alarm of u2r: 0.0100 (1.00%)
Overall False Alarm Rate: 0.1405 (14.05%)
Training Time: 27.4886 seconds
Test Time: 0.5240 seconds
Total Time: 28.0126 seconds
Error Rate: 0.2531 (25.31%)
```

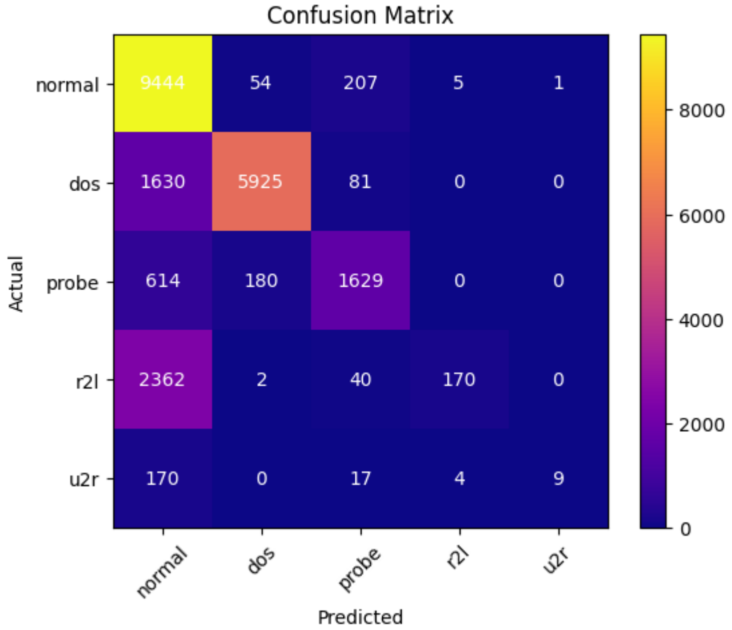



```
========================================
Classifier: SVC
Evaluation Report for SVC:
F-Score: 0.5061406287985528
Precision: 0.8868031619708534
Re-call: 0.48511142759360626
Accuracy: 0.7462739531582683
Confusion Matrix:
[[9462   62  187    0    0]
 [1882 5693   61    0    0]
 [ 836  175 1412    0    0]
 [2318    0    4  252    0]
 [ 177    0   15    3    5]]
              precision    recall  f1-score   support

      normal       0.64      0.97      0.78      9711
         dos       0.96      0.75      0.84      7636
       probe       0.84      0.58      0.69      2423
         r2l       0.99      0.10      0.18      2574
         u2r       1.00      0.03      0.05       200

    accuracy                           0.75     22544
   macro avg       0.89      0.49      0.51     22544
weighted avg       0.82      0.75      0.71     22544

False Alarm of normal: 0.5368 (53.68%)
False Alarm of dos: 0.0310 (3.10%)
False Alarm of probe: 0.1102 (11.02%)
False Alarm of r2l: 0.0012 (0.12%)
False Alarm of u2r: 0.0000 (0.00%)
Overall False Alarm Rate: 0.1358 (13.58%)
Training Time: 141.4420 seconds
Test Time: 14.0296 seconds
Total Time: 155.4716 seconds
Error Rate: 0.2537 (25.37%)
```





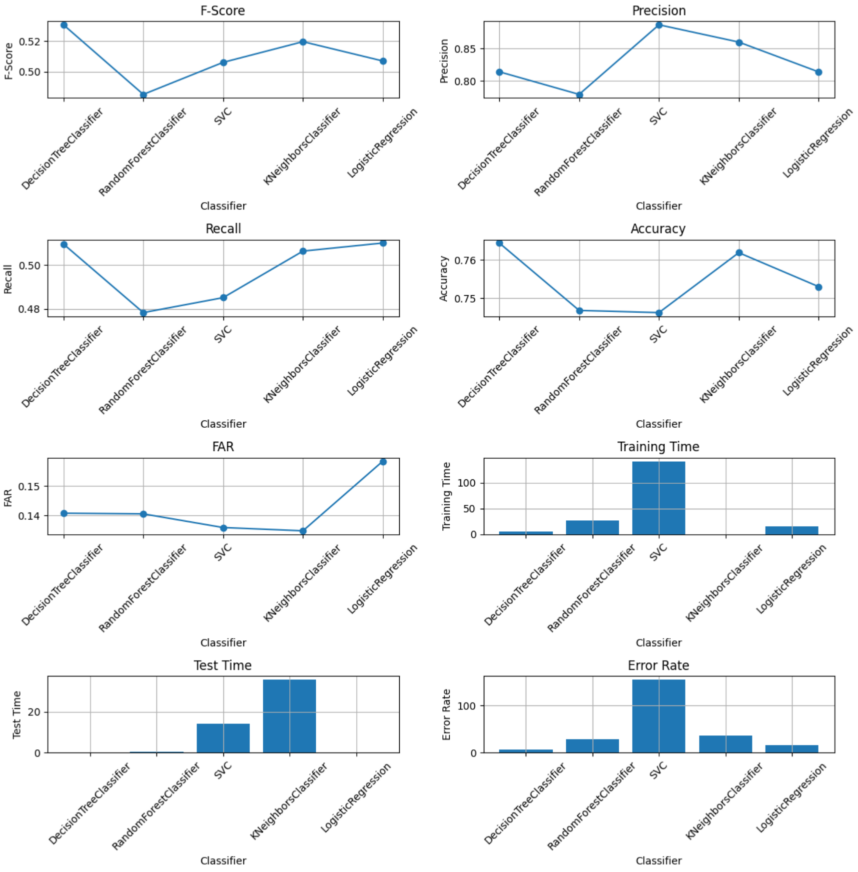

```
========================================
Classifier: KNeighborsClassifier
Evaluation Report for KNeighborsClassifier:
F-Score: 0.5196654360220211
Precision: 0.8601556370245385
Re-call: 0.5063574671667119
Accuracy: 0.7619322214336409
Confusion Matrix:
[[9444   54  207    5    1]
 [1630 5925   81    0    0]
 [ 614  180 1629    0    0]
 [2362    2   40  170    0]
 [ 170    0   17    4    9]]
              precision    recall  f1-score   support

      normal       0.66      0.97      0.79      9711
         dos       0.96      0.78      0.86      7636
       probe       0.83      0.67      0.74      2423
         r2l       0.95      0.07      0.12      2574
         u2r       0.90      0.04      0.09       200

    accuracy                           0.76     22544
   macro avg       0.86      0.51      0.52     22544
weighted avg       0.82      0.76      0.73     22544

False Alarm of normal: 0.4918 (49.18%)
False Alarm of dos: 0.0309 (3.09%)
False Alarm of probe: 0.1424 (14.24%)
False Alarm of r2l: 0.0035 (0.35%)
False Alarm of u2r: 0.0050 (0.50%)
Overall False Alarm Rate: 0.1347 (13.47%)
Training Time: 0.3045 seconds
Test Time: 35.9046 seconds
Total Time: 36.2091 seconds
Error Rate: 0.2381 (23.81%)
```




```
========================================
Classifier: LogisticRegression
Evaluation Report for LogisticRegression:
F-Score: 0.5070502281065372
Precision: 0.8141604993873506
Re-call: 0.5100569661036316
Accuracy: 0.7530606813342796
Confusion Matrix:
[[8993   90  624    2    2]
 [1560 6052   24    0    0]
 [ 496   99 1825    3    0]
 [2471    2    2   99    0]
 [ 184    3    0    5    8]]
              precision    recall  f1-score   support

      normal       0.66      0.93      0.77      9711
         dos       0.97      0.79      0.87      7636
       probe       0.74      0.75      0.75      2423
         r2l       0.91      0.04      0.07      2574
         u2r       0.80      0.04      0.08       200
```

```
    accuracy                           0.75     22544
   macro avg       0.81      0.51      0.51     22544
weighted avg       0.80      0.75      0.72     22544

False Alarm of normal: 0.4851 (48.51%)
False Alarm of dos: 0.0254 (2.54%)
False Alarm of probe: 0.2683 (26.83%)
False Alarm of r2l: 0.0039 (0.39%)
False Alarm of u2r: 0.0100 (1.00%)
Overall False Alarm Rate: 0.1585 (15.85%)
```

```
metrics_names = ['F-Score', 'Precision', 'Recall', 'Accuracy', 'FAR', 'Training Time', 'Test Time', 'Error Rate']
metrics_values = [list(metric.values()) for metric in performance_metrics]

plt.figure(figsize=(12, 12))  # Increase the figure size to accommodate additional charts

for i, metric_name in enumerate(metrics_names):
    plt.subplot(4, 2, i + 1)

    # Special handling for Time and Error Rate metrics
    if metric_name in ['Training Time', 'Test Time', 'Error Rate']:
        plt.bar([classifier.__name__ for classifier in classifiers], [values[i] for values in metrics_values])
    else:
        plt.plot([classifier.__name__ for classifier in classifiers], [values[i] for values in metrics_values], marker='o')

    plt.title(metric_name)
    plt.xlabel('Classifier')
    plt.xticks(rotation=45)
    plt.ylabel(metric_name)
    plt.grid(True)

plt.tight_layout()
plt.show()
```


F-Score
Precision
Recall
Accuracy
FAR
Training Time
Test Time
Error Rate
Classifier
DecisionTreeClassifier
RandomForestClassifier
SVC
KNeighborsClassifier
LogisticRegression
0.52
0.50
0.85
0.80
0.48
0.76
0.75
0.15
0.14
100
50
20
0