Configuration Manual for Intrusion Detection Using Machine learning With Real-Time Dashboard

MSc Research Project
Data Analytics

Sangeetha Mora
Student ID: 23219602

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sangeetha Mora		
Student ID:	23219602		
Programme:	MSc Data Analytics	**Year:**	2024 - 25
Module:	MSc Research Project		
Lecturer:	Noel Cosgrave		
Submission Due Date:	12/12/2024		
Project Title:	Intrusion Detection Using Machine learning With Real-Time Dashboard		
Word Count:	724	**Page Count:**	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sangeetha mora

Date: 12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□NA
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□NA
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□NA

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Intrusion Detection Using Machine learning With Real-Time Dashboard

Sangeetha Mora

Student ID: 23219602

1. Introduction

This manual contains the step-by-step replication of "**Intrusion Detection Using Machine learning With Real-Time Dashboard**". The study has several models including Decision Tree, Anomaly detection, Gaussian naïve bayes, Random Forest & Random Forest with recursive feature elimination.

2. Deployment Environment

The environment used for this study is a ubuntu virtual machine running on google gcp however the system requirements are listed as below.

2.1 Hardware Specification

- **Processor:** Intel Core i5 or equivalent
- **RAM:** 8 GB or higher

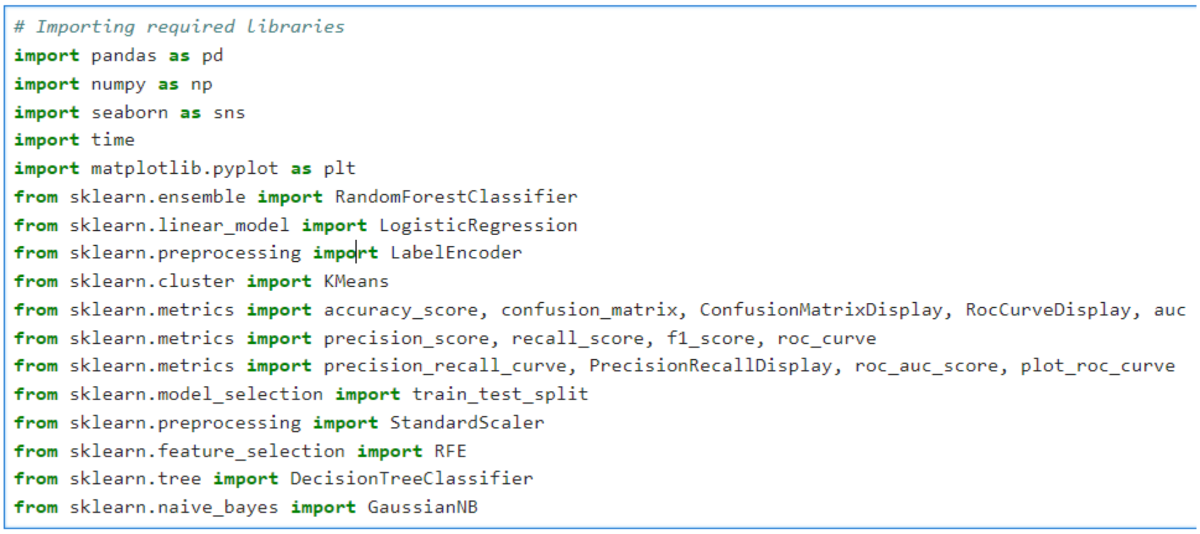
2.2 Software Specification

- **Operating System:** Ubuntu 20.0.4
- **Programming Language:** Python 3.11
- **IDE:** Jupyter Notebook

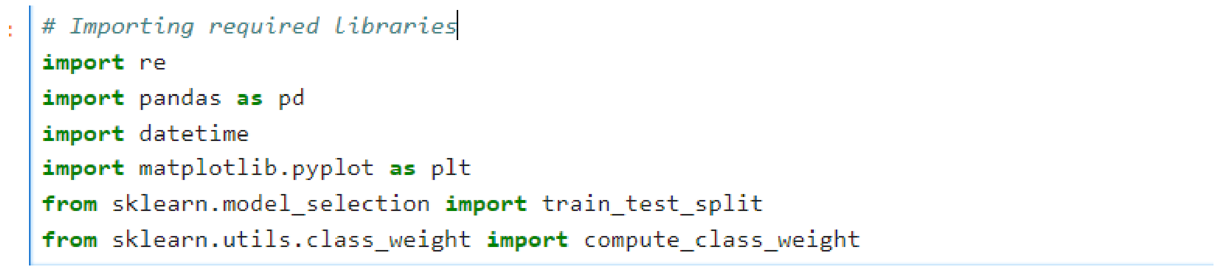
2.1 Python Libraries Required

- **pandas:** For data analysis and manipulation
- **NumPy:** For numerical operations
- **matplotlib.pyplot:** For creating visualizations.
- **Seaborn:** For statistical data visualization
- **scikit-learn:** A comprehensive machine learning library for tasks like classification, regression, clustering, and model selection.
- **Model Selection:** train_test_split for splitting data into training and testing sets.
- **Classification:** RandomForestClassifier, LogisticRegression, DecisionTreeClassifier, GaussianNB for various classification algorithms.
- **Clustering:** KMeans for clustering data.

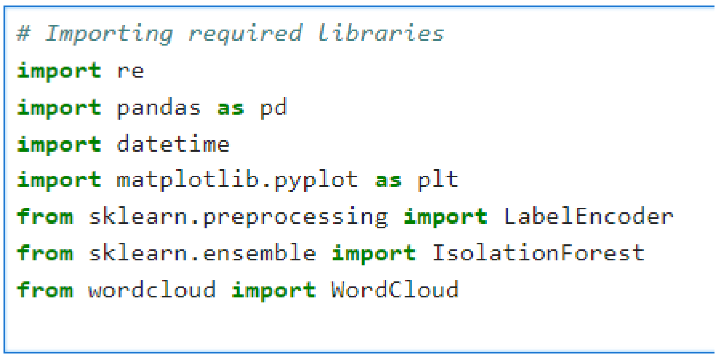
- **Preprocessing**: LabelEncoder, StandardScaler for data preprocessing.
- **Metrics:** accuracy_score, confusion_matrix, precision_score, recall_score, f1_score, roc_curve, roc_auc_score, PrecisionRecallDisplay, RocCurveDisplay for evaluating model performance.
- **re:** For regular expressions, used for pattern matching and text manipulation.
- **datetime:** For working with dates and times.
- **wordcloud:** For generating word clouds from text data.
- **sklearn.utils.class_weight:** For computing class weights, useful for imbalanced datasets.
- **sklearn.ensemble.IsolationForest:** For anomaly detection.



```python
# Importing required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import time
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay, auc
from sklearn.metrics import precision_score, recall_score, f1_score, roc_curve
from sklearn.metrics import precision_recall_curve, PrecisionRecallDisplay, roc_auc_score, plot_roc_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
```



```python
# Importing required libraries
import re
import pandas as pd
import datetime
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
```



```python
# Importing required libraries
import re
import pandas as pd
import datetime
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import IsolationForest
from wordcloud import WordCloud
```

3. Data sources

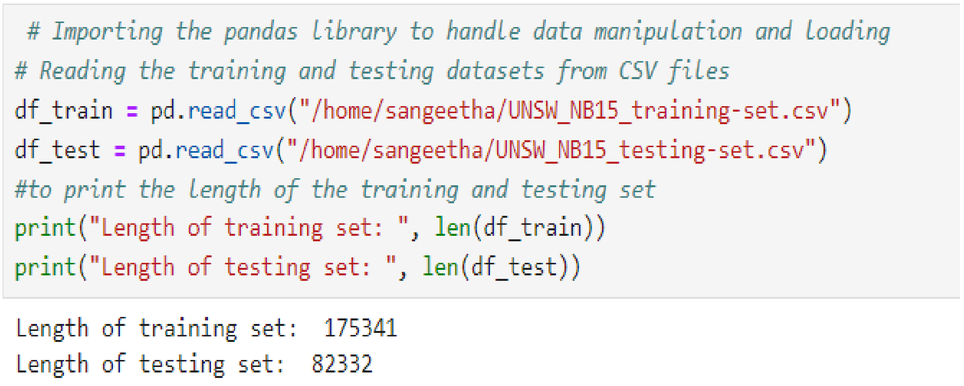
The total of three datasets were used in this study.

- UNSW-NB15
- Authentication Data (Generated in my ubuntu VM)
- Network Data (Generated in my ubuntu VM)

I will zip all the three datasets if not running from ubuntu use my .log file for testing

4. Implementation of UNSW-NB15

Importing data



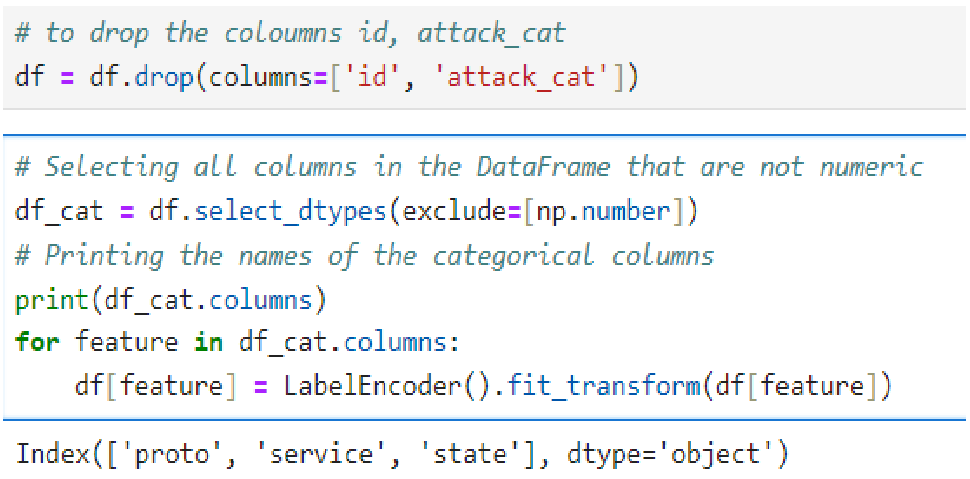
```
 # Importing the pandas library to handle data manipulation and loading
# Reading the training and testing datasets from CSV files
df_train = pd.read_csv("/home/sangeetha/UNSW_NB15_training-set.csv")
df_test = pd.read_csv("/home/sangeetha/UNSW_NB15_testing-set.csv")
#to print the length of the training and testing set
print("Length of training set: ", len(df_train))
print("Length of testing set: ", len(df_test))
```

```
Length of training set:  175341
Length of testing set:  82332
```

Data preprocessing :

Selecting the non-numeric columns in the data frame that are not numeric and

Converting each non-numeric column into numeric representation



```
# to drop the coloumns id, attack_cat
df = df.drop(columns=['id', 'attack_cat'])
```

```
# Selecting all columns in the DataFrame that are not numeric
df_cat = df.select_dtypes(exclude=[np.number])
# Printing the names of the categorical columns
print(df_cat.columns)
for feature in df_cat.columns:
    df[feature] = LabelEncoder().fit_transform(df[feature])
```

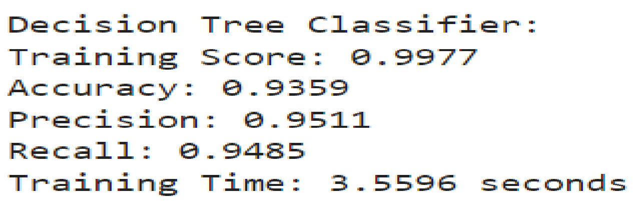
```
Index(['proto', 'service', 'state'], dtype='object')
```

Evaluation

Decision Tree Classifier

Decision Trees are prone to overfitting with complex datasets, and easy to

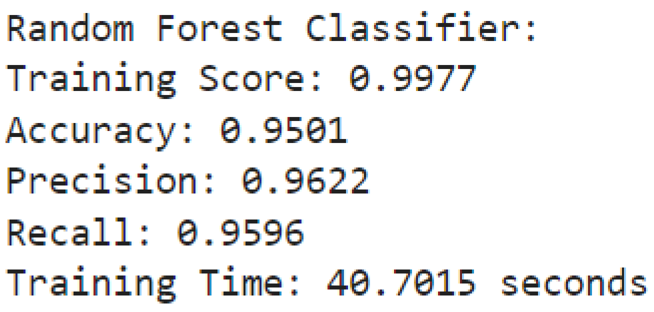
initialize but for large dataset overfitting may happen.



```
Decision Tree Classifier:
Training Score: 0.9977
Accuracy: 0.9359
Precision: 0.9511
Recall: 0.9485
Training Time: 3.5596 seconds
```

Random Forest Classifier

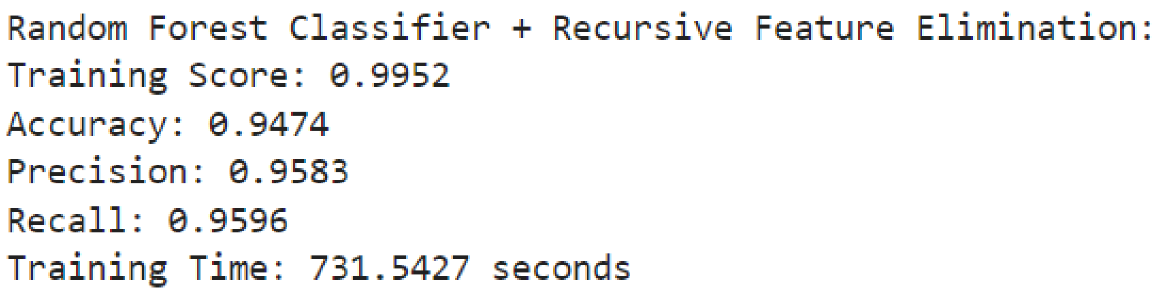
Random Forest was one of the techniques used to generalize well over the dataset to improve intrusion detection accuracy in this project. This quality allowed us to predict the attack accurately by feeding noisy or imbalanced data into the test



```
Random Forest Classifier:
Training Score: 0.9977
Accuracy: 0.9501
Precision: 0.9622
Recall: 0.9596
Training Time: 40.7015 seconds
```

Random Forest Classifier with Recursive Feature Elimination

RFE was employed to narrow down and select the critical features when applied with Random Forest on UNSW-NB15dataset, entailing high dimensionality &noisy features.



```
Random Forest Classifier + Recursive Feature Elimination:
Training Score: 0.9952
Accuracy: 0.9474
Precision: 0.9583
Recall: 0.9596
Training Time: 731.5427 seconds
```

5. **Implementation of Authentication data**

Using Exception handling importing the data and creating the data frame

```python
try:
    # Read the authentication data file into a DataFrame
    with open('/etc/data/auth_stream.log', 'r') as file:
        log_lines = file.readlines()

    # Createing a DataFrame from the authentication data
    auth_df = pd.DataFrame(log_lines, columns=['log'])
    print("Log file loaded successfully!")

    # Displaying the first few rows to check the data
    print(auth_df.head())
except FileNotFoundError:
    print("Error: The log file was not found. Please check the file path.")
except PermissionError:
    print("Error: Permission denied while trying to read the log file.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Log file loaded successfully!
                                                 log
0  Dec  2 15:31:24 hadoop sshd[23593]: pam_unix(s...
1  Dec  2 15:31:24 hadoop systemd-logind[877]: Ne...
2  Dec  2 15:31:24 hadoop sshd[23667]: error: Fai...
3  Dec  2 15:31:32 hadoop sudo: sangeetha : TTY=p...
4  Dec  2 15:31:32 hadoop sudo: pam_unix(sudo:ses...
```

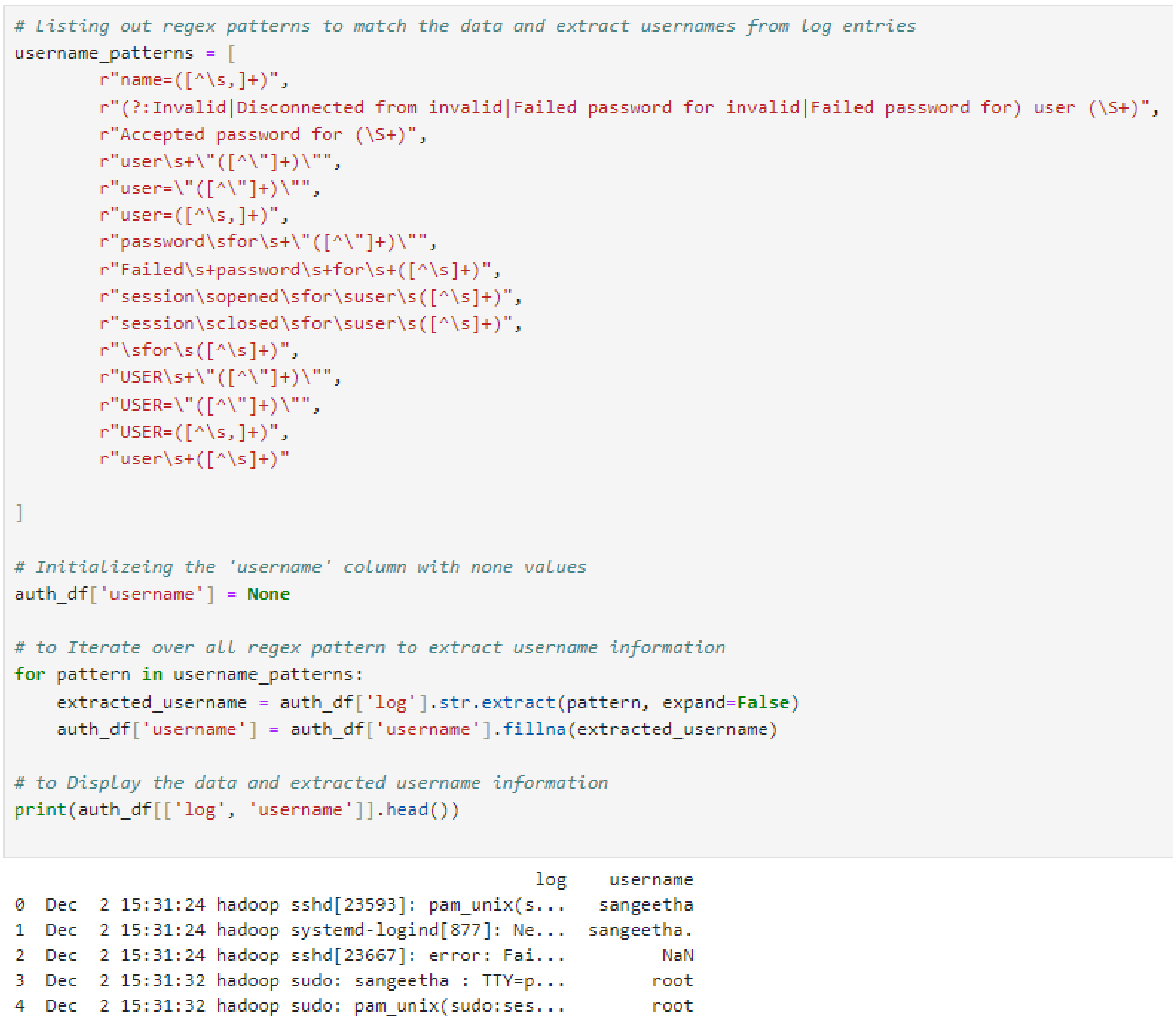
Data preprocessing

Using Regex converting unstructured data into structured and extracting timestamp from the the authentication data and converting it into datetime format

```
# To get the current year to the timestamp
current_year = datetime.datetime.now().year

# Extracting timestamp from the data and converting it to the datetime format
auth_df['timestamp'] = pd.to_datetime(
    auth_df['log'].str.extract(r"(\w{3}\s+\d{1,2}\s\d{2}:\d{2}:\d{2})", expand=False)
    .apply(lambda x: f"{current_year} {x}" if pd.notnull(x) else None),
    format='%Y %b %d %H:%M:%S',
    errors='coerce'
)

# Formating the timestamp to a consistent string format(YYYY-MM-DD HH:MM:SS)
auth_df['timestamp'] = auth_df['timestamp'].dt.strftime('%Y-%m-%d %H:%M:%S')

# to Display the extracted data and formatted timestamps
print(auth_df[['log', 'timestamp']].head())
```

```
                                          log            timestamp
0  Dec  2 15:31:24 hadoop sshd[23593]: pam_unix(s...  2024-12-02 15:31:24
1  Dec  2 15:31:24 hadoop systemd-logind[877]: Ne...  2024-12-02 15:31:24
2  Dec  2 15:31:24 hadoop sshd[23667]: error: Fai...  2024-12-02 15:31:24
3  Dec  2 15:31:32 hadoop sudo: sangeetha : TTY=p...  2024-12-02 15:31:32
4  Dec  2 15:31:32 hadoop sudo: pam_unix(sudo:ses...  2024-12-02 15:31:32
```

Extracting user names and making it structured by using regex

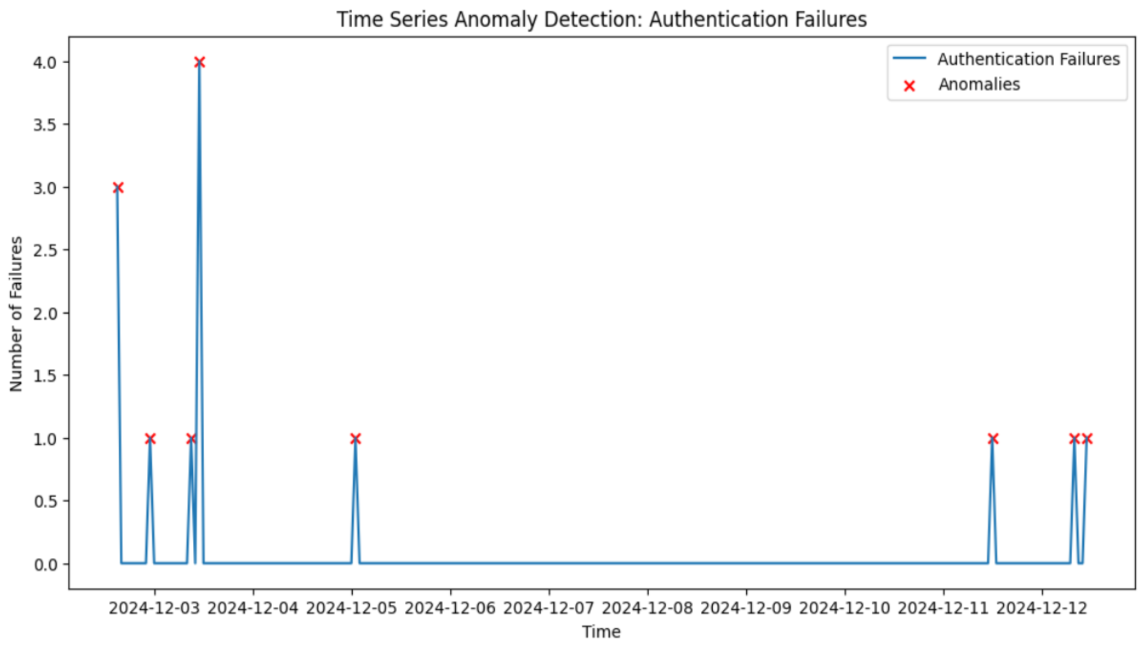


```python
# Listing out regex patterns to match the data and extract usernames from log entries
username_patterns = [
        r"name=([^\s,]+)",
        r"(?:Invalid|Disconnected from invalid|Failed password for invalid|Failed password for) user (\S+)",
        r"Accepted password for (\S+)",
        r"user\s+\"([^\"]+)\"",
        r"user=\"([^\"]+)\"",
        r"user=([^\s,]+)",
        r"password\sfor\s+\"([^\"]+)\"",
        r"Failed\s+password\s+for\s+([^\s]+)",
        r"session\sopened\sfor\suser\s([^\s]+)",
        r"session\sclosed\sfor\suser\s([^\s]+)",
        r"\sfor\s([^\s]+)",
        r"USER\s+\"([^\"]+)\"",
        r"USER=\"([^\"]+)\"",
        r"USER=([^\s,]+)",
        r"user\s+([^\s]+)"

]

# Initializeing the 'username' column with none values
auth_df['username'] = None

# to Iterate over all regex pattern to extract username information
for pattern in username_patterns:
    extracted_username = auth_df['log'].str.extract(pattern, expand=False)
    auth_df['username'] = auth_df['username'].fillna(extracted_username)

# to Display the data and extracted username information
print(auth_df[['log', 'username']].head())
```

```
                                               log    username
0  Dec  2 15:31:24 hadoop sshd[23593]: pam_unix(s...   sangeetha
1  Dec  2 15:31:24 hadoop systemd-logind[877]: Ne...  sangeetha.
2  Dec  2 15:31:24 hadoop sshd[23667]: error: Fai...         NaN
3  Dec  2 15:31:32 hadoop sudo: sangeetha : TTY=p...        root
4  Dec  2 15:31:32 hadoop sudo: pam_unix(sudo:ses...        root
```

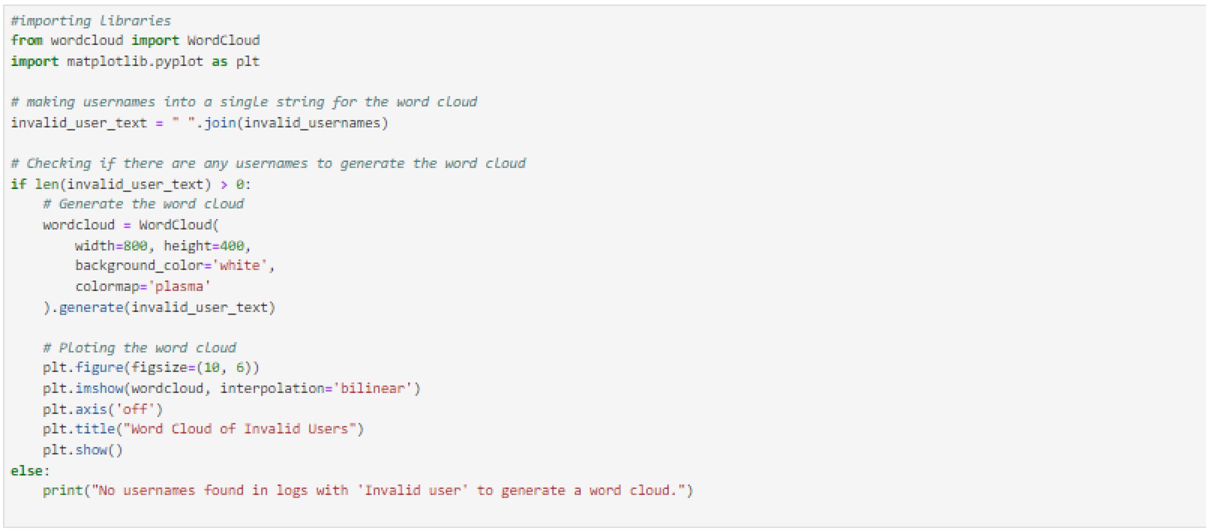
Evaluation

Time series Anomaly detection

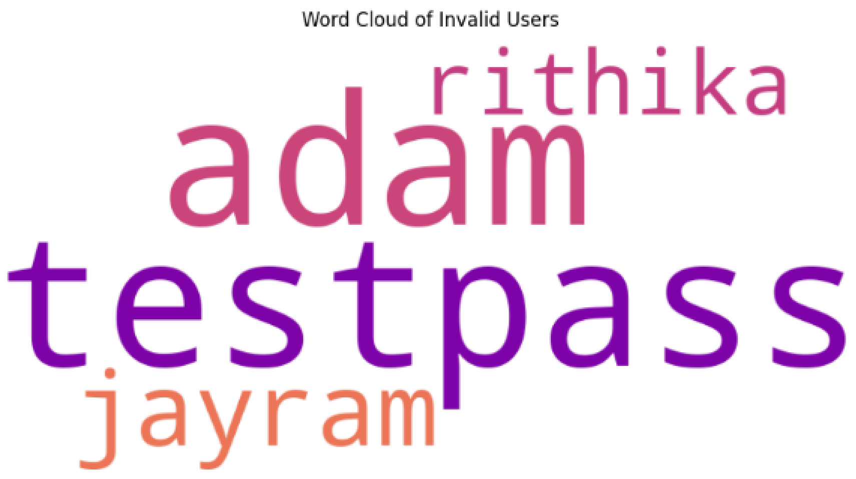


Real – time dashboard visualization of authentication data

To check the attackers run the kernal from start to check the invalid users



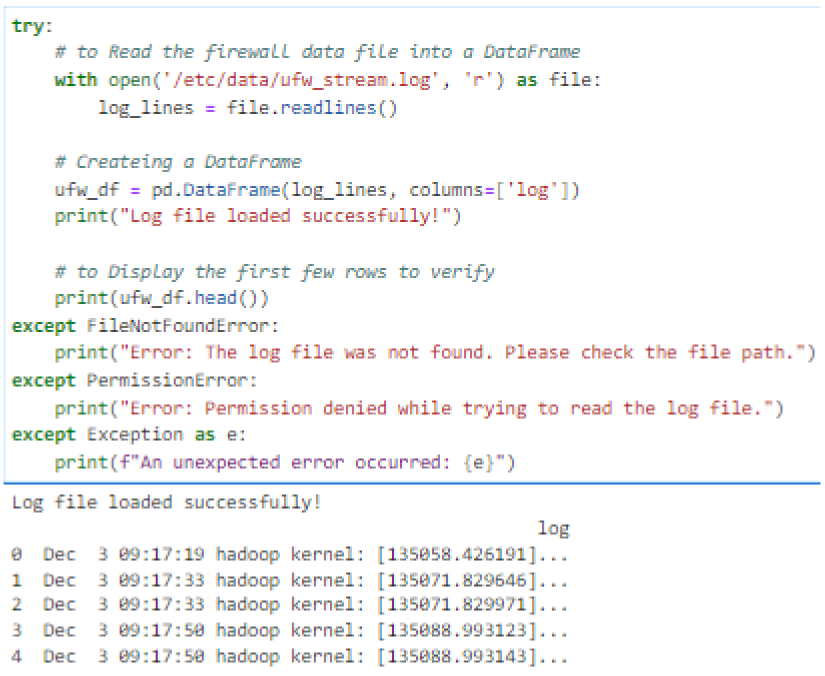
```
#importing libraries
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# making usernames into a single string for the word cloud
invalid_user_text = " ".join(invalid_usernames)

# Checking if there are any usernames to generate the word cloud
if len(invalid_user_text) > 0:
    # Generate the word cloud
    wordcloud = WordCloud(
        width=800, height=400,
        background_color='white',
        colormap='plasma'
    ).generate(invalid_user_text)

    # Ploting the word cloud
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title("Word Cloud of Invalid Users")
    plt.show()
else:
    print("No usernames found in logs with 'Invalid user' to generate a word cloud.")
```



6. Implementation of Network Data

Importing the data

Importing the firewall data using exception handling and creating it into a dataframe

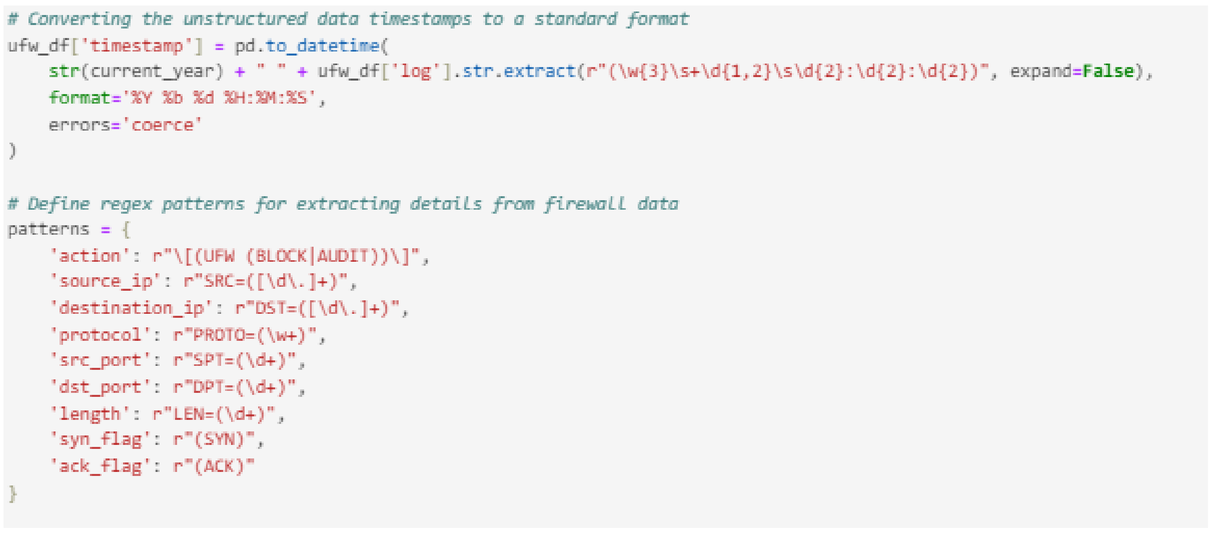


```
try:
    # to Read the firewall data file into a DataFrame
    with open('/etc/data/ufw_stream.log', 'r') as file:
        log_lines = file.readlines()

    # Createing a DataFrame
    ufw_df = pd.DataFrame(log_lines, columns=['log'])
    print("Log file loaded successfully!")

    # to Display the first few rows to verify
    print(ufw_df.head())
except FileNotFoundError:
    print("Error: The log file was not found. Please check the file path.")
except PermissionError:
    print("Error: Permission denied while trying to read the log file.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

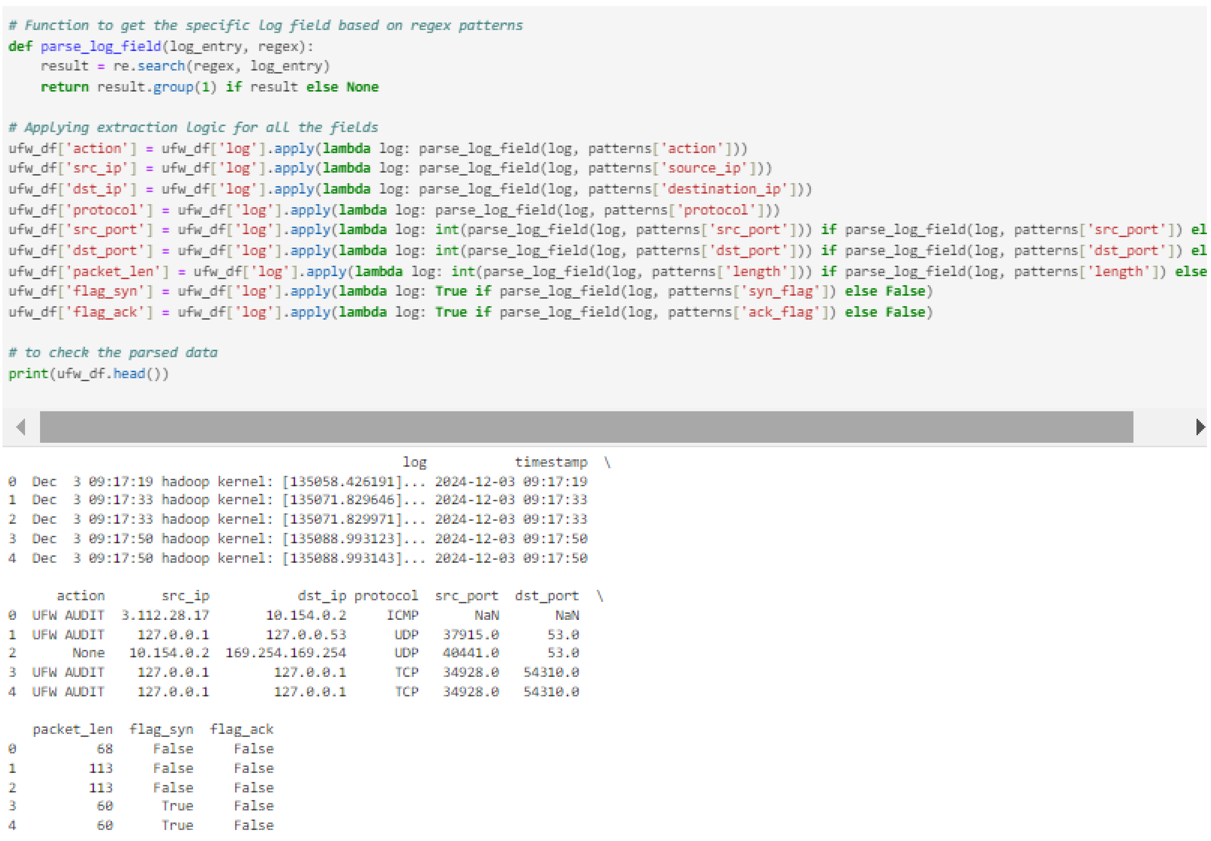
```
Log file loaded successfully!
                                                    log
0  Dec  3 09:17:19 hadoop kernel: [135058.426191]...
1  Dec  3 09:17:33 hadoop kernel: [135071.829646]...
2  Dec  3 09:17:33 hadoop kernel: [135071.829971]...
3  Dec  3 09:17:50 hadoop kernel: [135088.993123]...
4  Dec  3 09:17:50 hadoop kernel: [135088.993143]...
```

Data preprocessing

Coverting the unstructured data timestamps to a standard format by using regex



```
# Converting the unstructured data timestamps to a standard format
ufw_df['timestamp'] = pd.to_datetime(
    str(current_year) + " " + ufw_df['log'].str.extract(r"(\w{3}\s+\d{1,2}\s\d{2}:\d{2}:\d{2})", expand=False),
    format='%Y %b %d %H:%M:%S',
    errors='coerce'
)

# Define regex patterns for extracting details from firewall data
patterns = {
    'action': r"\[(UFW (BLOCK|AUDIT))\]",
    'source_ip': r"SRC=([\d\.]+)",
    'destination_ip': r"DST=([\d\.]+)",
    'protocol': r"PROTO=(\w+)",
    'src_port': r"SPT=(\d+)",
    'dst_port': r"DPT=(\d+)",
    'length': r"LEN=(\d+)",
    'syn_flag': r"(SYN)",
    'ack_flag': r"(ACK)"
}
```

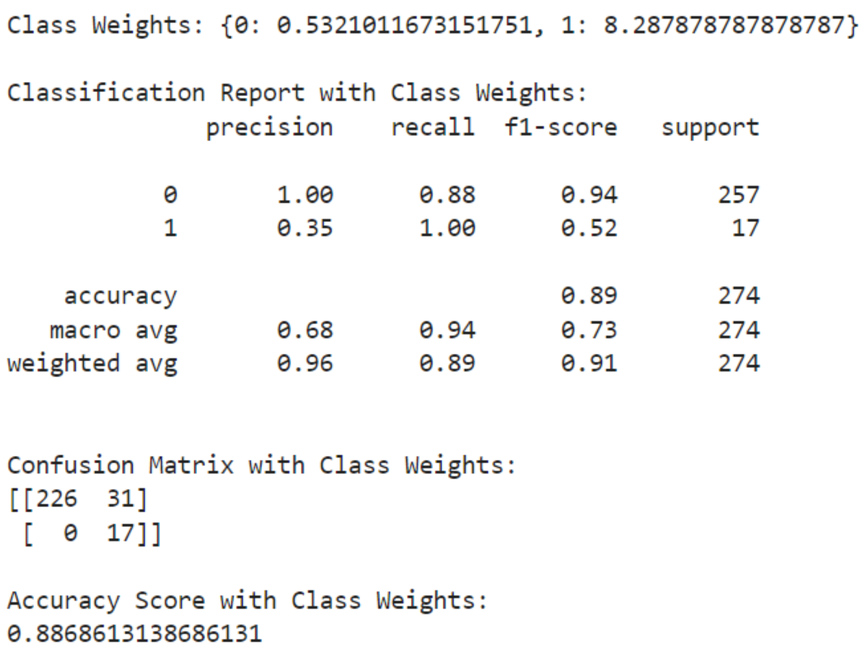


```
# Function to get the specific log field based on regex patterns
def parse_log_field(log_entry, regex):
    result = re.search(regex, log_entry)
    return result.group(1) if result else None

# Applying extraction logic for all the fields
ufw_df['action'] = ufw_df['log'].apply(lambda log: parse_log_field(log, patterns['action']))
ufw_df['src_ip'] = ufw_df['log'].apply(lambda log: parse_log_field(log, patterns['source_ip']))
ufw_df['dst_ip'] = ufw_df['log'].apply(lambda log: parse_log_field(log, patterns['destination_ip']))
ufw_df['protocol'] = ufw_df['log'].apply(lambda log: parse_log_field(log, patterns['protocol']))
ufw_df['src_port'] = ufw_df['log'].apply(lambda log: int(parse_log_field(log, patterns['src_port'])) if parse_log_field(log, patterns['src_port']) el
ufw_df['dst_port'] = ufw_df['log'].apply(lambda log: int(parse_log_field(log, patterns['dst_port'])) if parse_log_field(log, patterns['dst_port']) el
ufw_df['packet_len'] = ufw_df['log'].apply(lambda log: int(parse_log_field(log, patterns['length'])) if parse_log_field(log, patterns['length']) else
ufw_df['flag_syn'] = ufw_df['log'].apply(lambda log: True if parse_log_field(log, patterns['syn_flag']) else False)
ufw_df['flag_ack'] = ufw_df['log'].apply(lambda log: True if parse_log_field(log, patterns['ack_flag']) else False)

# to check the parsed data
print(ufw_df.head())
```

```
                                                 log           timestamp  \
0  Dec  3 09:17:19 hadoop kernel: [135058.426191]... 2024-12-03 09:17:19
1  Dec  3 09:17:33 hadoop kernel: [135071.829646]... 2024-12-03 09:17:33
2  Dec  3 09:17:33 hadoop kernel: [135071.829971]... 2024-12-03 09:17:33
3  Dec  3 09:17:50 hadoop kernel: [135088.993123]... 2024-12-03 09:17:50
4  Dec  3 09:17:50 hadoop kernel: [135088.993143]... 2024-12-03 09:17:50

      action       src_ip           dst_ip protocol  src_port  dst_port  \
0  UFW AUDIT  3.112.28.17       10.154.0.2     ICMP       NaN       NaN
1  UFW AUDIT    127.0.0.1       127.0.0.53      UDP   37915.0      53.0
2       None   10.154.0.2  169.254.169.254      UDP   40441.0      53.0
3  UFW AUDIT    127.0.0.1        127.0.0.1      TCP   34928.0   54310.0
4  UFW AUDIT    127.0.0.1        127.0.0.1      TCP   34928.0   54310.0

   packet_len  flag_syn  flag_ack
0          68     False     False
1         113     False     False
2         113     False     False
3          60      True     False
4          60      True     False
```

Evaluation

Here the logestic regression is used for the firewall data



```
Class Weights: {0: 0.5321011673151751, 1: 8.287878787878787}

Classification Report with Class Weights:
              precision    recall  f1-score   support

           0       1.00      0.88      0.94       257
           1       0.35      1.00      0.52        17

    accuracy                           0.89       274
   macro avg       0.68      0.94      0.73       274
weighted avg       0.96      0.89      0.91       274


Confusion Matrix with Class Weights:
[[226  31]
 [  0  17]]

Accuracy Score with Class Weights:
0.8868613138686131
```