A.I. Lab Project

Instructor : Hurmat Hidayat

Laiba Niazi (20P-0029)
Shahzaib Niaz (20P-0558)

Steps for Achieving Results

- Files import
- Data -> dataframes
- Dataframes cleaning
  - Drop duplicates
  - Null values
  - Columns drop
  - Outliers
- Label Encoder (object -> int32)
- Standard scaling
- Plots (original vs scaled data)
- Data split -> Test and train
- Co-relation Matrix
- Heat Map

- KNN (3,5,7)
- Decision tree (Entropy and Gini)
- ANN
- K-mean clusters (Attacks)

Dataframes Cleaning

```
[16]  print(df.duplicated().sum())
```

```
20
```

```
[17]  df=df.drop_duplicates()
```

```
[18]  df.shape
```

```
(125964, 43)
```

```
[19]  print(df[df.duplicated()])
```

```
Empty DataFrame
Columns: [duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, num_fa
Index: []

[0 rows x 43 columns]
```

Dataframes Cleaning

```
[8]  df['attack_type']=df['attack_type'].fillna('normal')
```

```
[9]  df.head()
```

same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_r
0.17	0.00	0.00	0.00	0
0.88	0.00	0.00	0.00	0
0.00	0.00	1.00	1.00	0
0.03	0.04	0.03	0.01	0
0.00	0.00	0.00	0.00	0

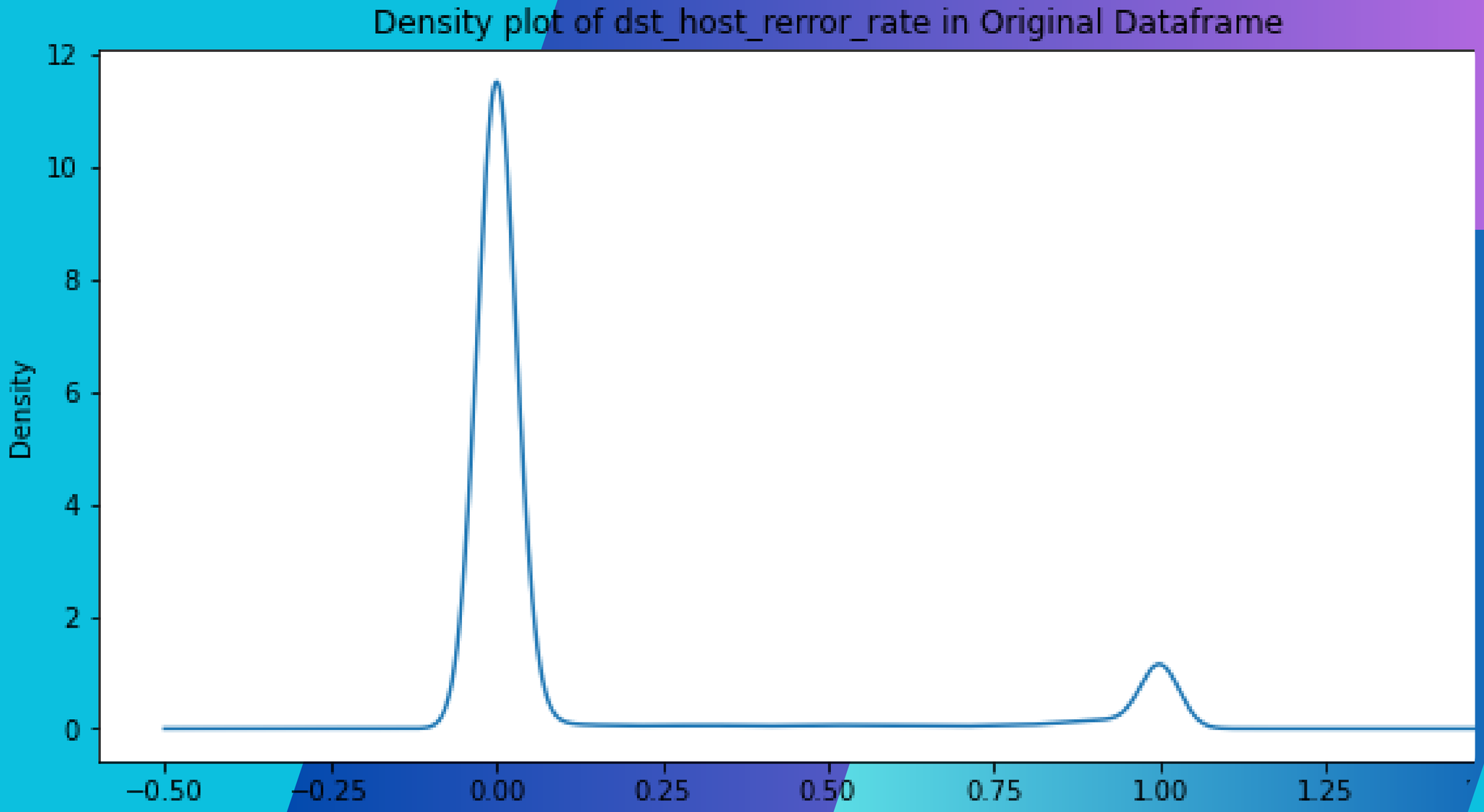
Dataframes Cleaning

```python
for col in df.columns:
    if df[col].nunique() == 1:
        df.drop(col, axis=1, inplace=True)
```

```python
df.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment
0	0	tcp	ftp_data	SF	491	0	0	0
1	0	udp	other	SF	146	0	0	0
2	0	tcp	private	S0	0	0	0	0
3	0	tcp	http	SF	232	8153	0	0
4	0	tcp	http	SF	199	420	0	0

Dataframes Cleaning



Label Encoder

```
le = LabelEncoder()
```

```
df[non_num_cols] = df[non_num_cols].apply(le.fit_transform)
```

```
df.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	u
0	0	1	20	9	491	0	0	0	
1	0	2	44	9	146	0	0	0	

Standard Scaler

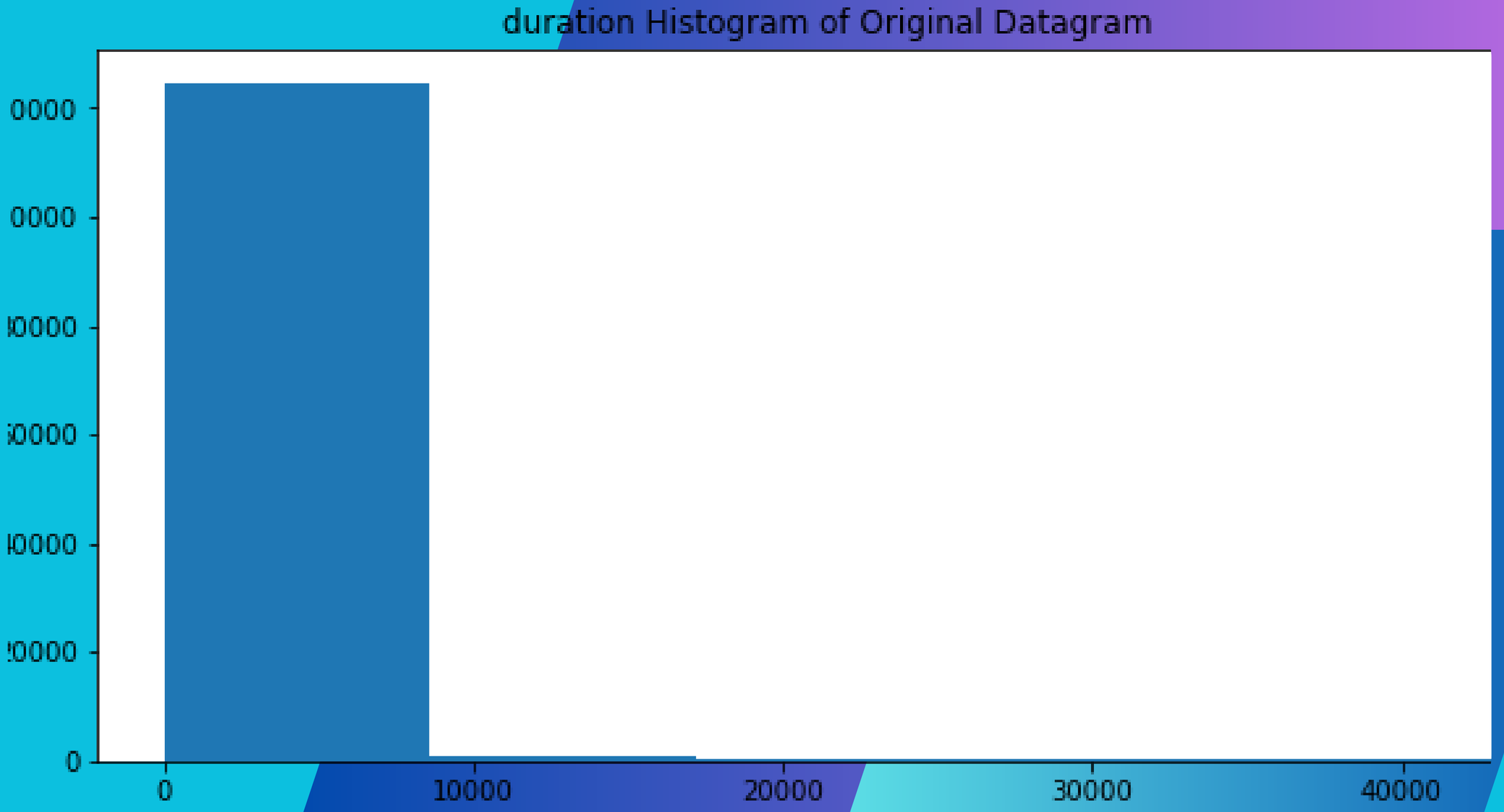
```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
```

```
df_scaled = pd.DataFrame(scaled_data, columns=df.columns)
```

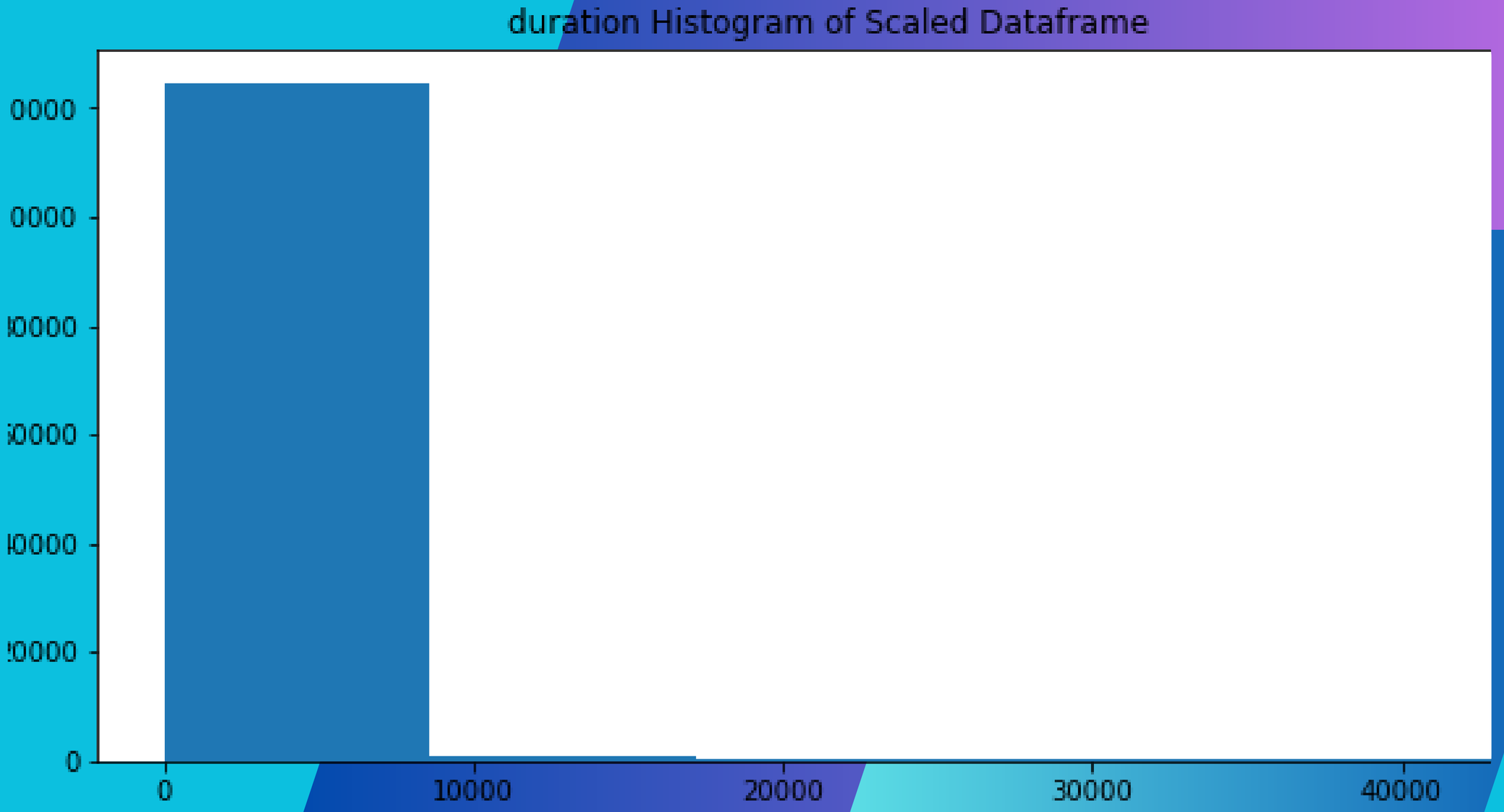
```
df_scaled.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wr
0	-0.110253	-0.124905	-0.686859	0.751153	-0.007679	-0.004919	-0.014089	
	0.110253	2.210530	0.781356	0.751153	0.007738	0.004919	0.014089	

Plot of Original Dataset



Plot of Scaled Dataset



Train Sets

```
y = df['attack_type']
```

```

```

```
X_train, X_test, y_train, y_test =train_test_split(df_scaled,y,test_size=0.3,random_state =0)
```

```

```

```
X_train.shape
```

```
(88174, 42)
```

```
X_test.shape
```

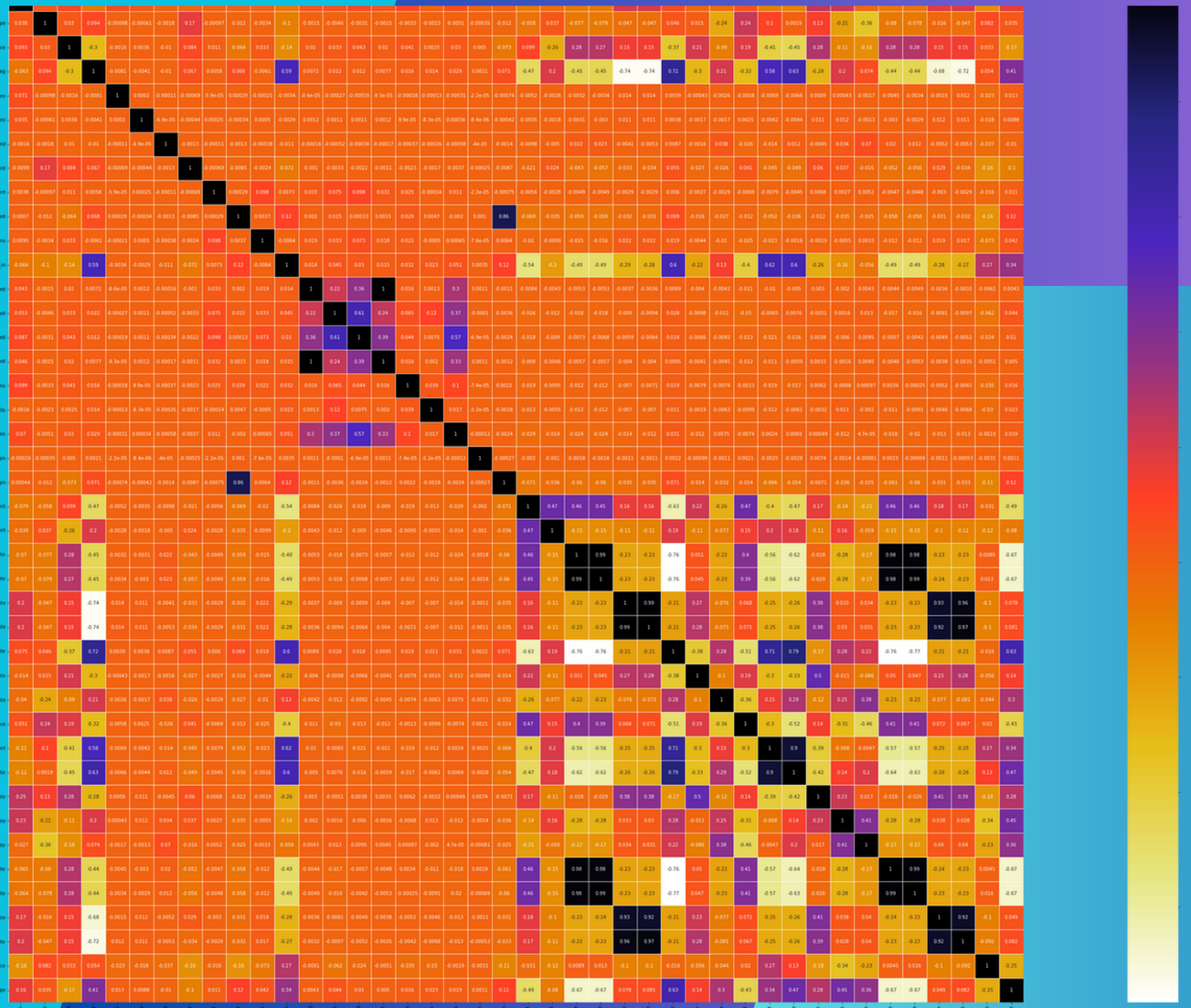
```
(37790, 42)
```

Co-relation Matrix

```
corr_matrix = df_scaled.corr()
print(corr_matrix)
```

```
                      duration  protocol_type   service      flag  \
duration              1.000000       0.038230  0.092853 -0.063385
protocol_type         0.038230       1.000000  0.029821  0.093823
service               0.092853       0.029821  1.000000 -0.303977
flag                 -0.063385       0.093823 -0.303977  1.000000
src_bytes             0.070737      -0.000975 -0.001632 -0.008114
dst_bytes             0.034878      -0.000609  0.003595 -0.004096
land                 -0.001553      -0.001760 -0.009953 -0.010372
wrong_fragment       -0.009867       0.169556  0.084401  0.067220
urgent                0.003830      -0.000966  0.010980  0.005811
hot                   0.000704      -0.011876 -0.064076  0.068444
num_failed_logins     0.009528      -0.003375  0.033042 -0.006061
```

Heat Map



Relevant Features

```
target_col = 'attack_type'
corr_with_target = corr_matrix[target_col]
relevant_features = corr_with_target[abs(corr_with_target) > 0.4].index.tolist()
print(relevant_features)
```

```
flag', 'count', 'serror_rate', 'srv_serror_rate', 'same_srv_rate', 'dst_host_count', 'dst_host_
```

```
relevant_features.remove(target_col)
```

```
X_train_new = X_train.drop(relevant_features,axis=1)
X_test_new = X_test.drop(relevant_features,axis=1)
```

K- Nearest Neighbour (K = 3, 5, 7)

```
k.append(3)
knn3 = KNeighborsClassifier(n_neighbors=3)
k.append(5)
knn5 = KNeighborsClassifier(n_neighbors=5)
k.append(7)
knn7 = KNeighborsClassifier(n_neighbors=7)
```

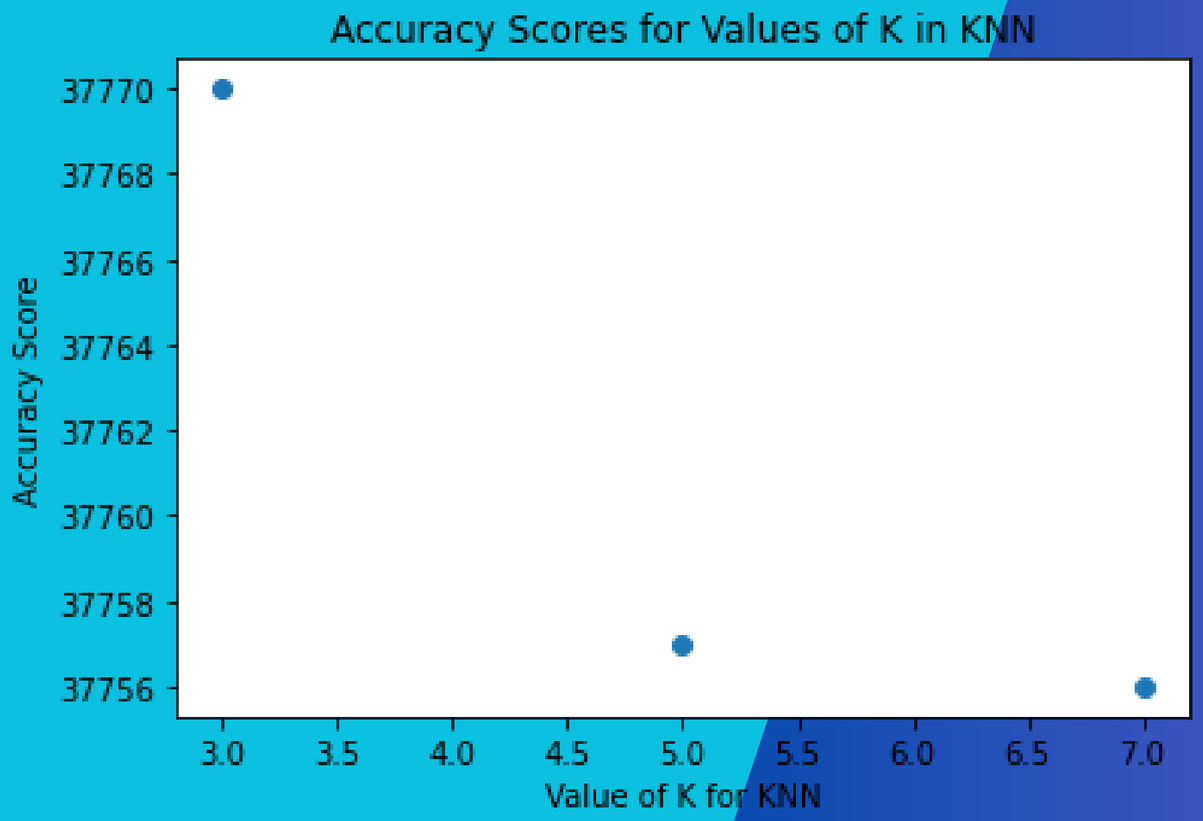
```
knn3.fit(X_train_new, y_train)
knn5.fit(X_train_new, y_train)
knn7.fit(X_train_new, y_train)
```

```
KNeighborsClassifier(n_neighbors=7)
```

```
X_test_new = X_test_new.dropna()
```

```
y_pred3 = knn3.predict(X_test_new)
y_pred5 = knn5.predict(X_test_new)
y_pred7 = knn7.predict(X_test_new)
```

Accuracy Score for Values of K




```
#For K = 3

accuracy.append(accuracy_score(y_test.round(), y_pred3.round(), normalize=False))

print("Accuracy:", accuracy)
```

[74]

```
Accuracy: [37770]
```

```
#For K = 5

accuracy.append(accuracy_score(y_test.round(), y_pred5.round(), normalize=False))

print("Accuracy:", accuracy)
```

[75]

```
Accuracy: [37770, 37757]
```

```
#For K = 7

accuracy.append(accuracy_score(y_test.round(), y_pred7.round(), normalize=False))

print("Accuracy:", accuracy)
```

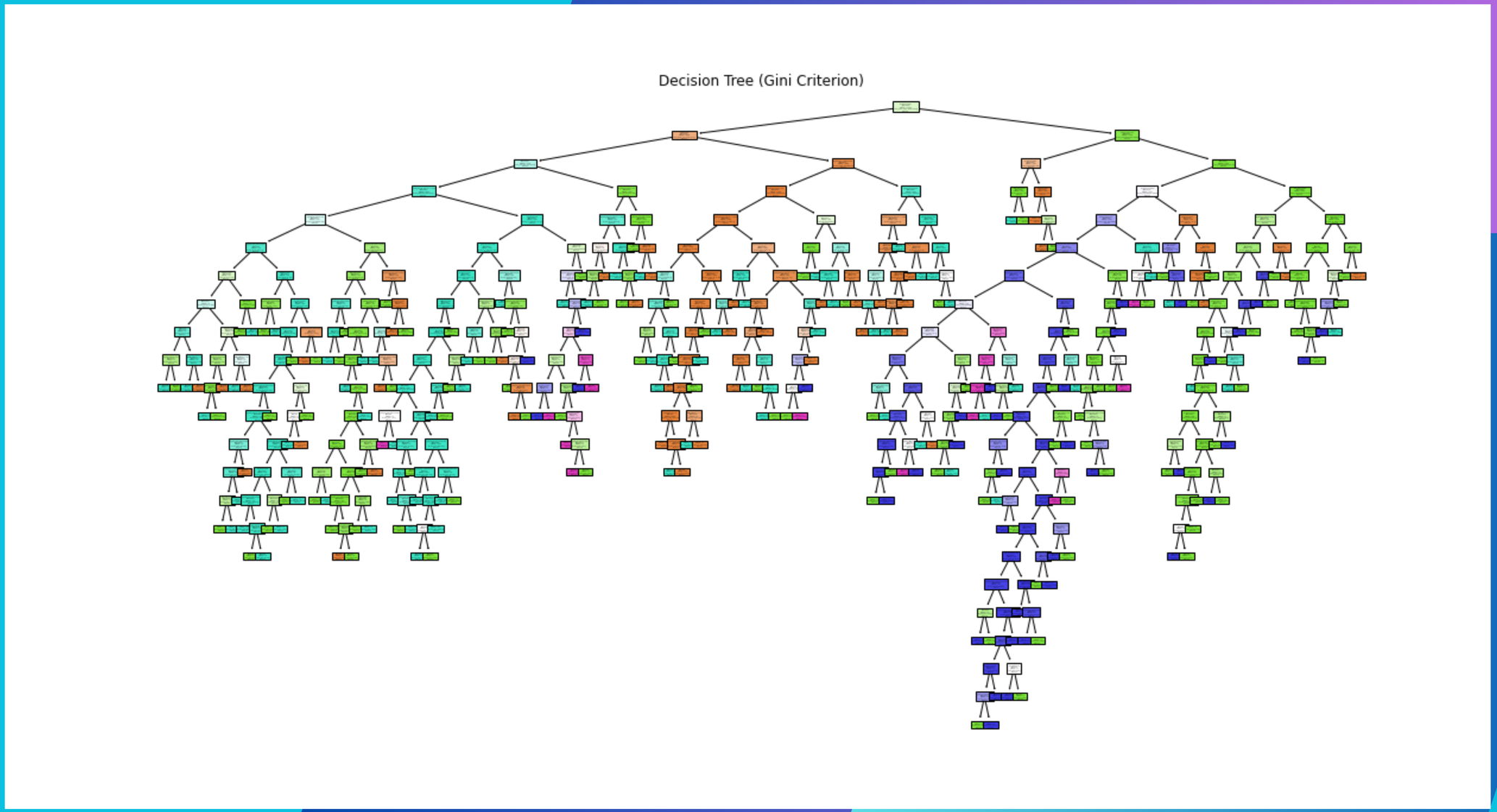
[76]

```
Accuracy: [37770, 37757, 37756]
```

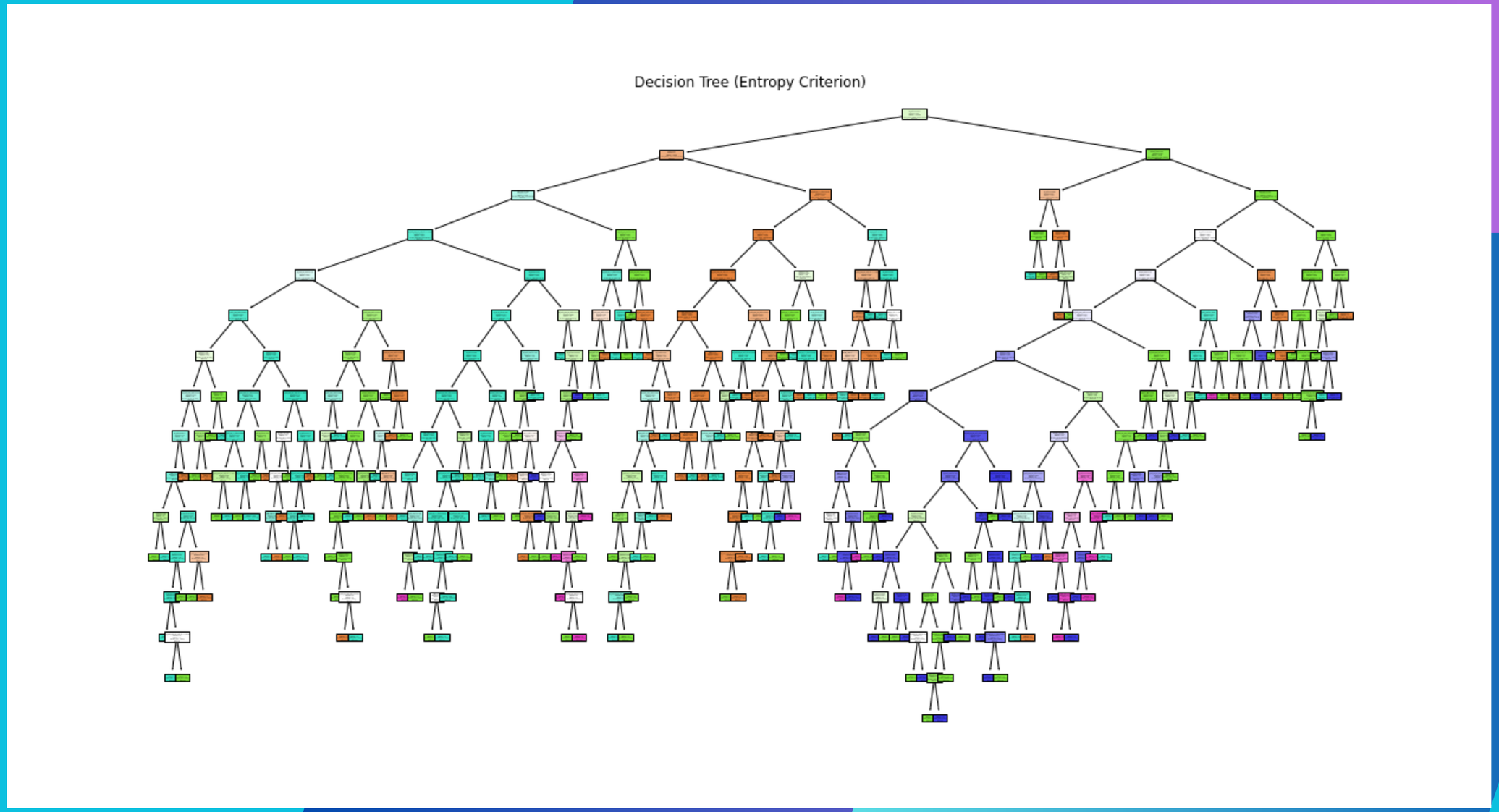
Decision Tree

```
[30]  ent = tree.DecisionTreeClassifier(criterion="entropy")

[31]  gin = tree.DecisionTreeClassifier(criterion="gini")

[32]  ent.fit(X_train_new, y_train)

      DecisionTreeClassifier(criterion='entropy')

[33]  gin.fit(X_train_new, y_train)

      DecisionTreeClassifier()
```

Decision Tree (Gini Criterion)



Decision Tree (Entropy Criterion)



ANN Using MLPClassifier

ANN

```
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
```

```
mlp.fit(X_train_new, y_train)
```

```
MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
```

```
y_pred = mlp.predict(X_test_new)
```

Classification Report Before Fine Tuning

```
[95]
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred,average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

```
[96]
print("Before Fine Tuning")
print("Accuracy: ", accuracy)
print("Precision Score: ", precision)
print("Recall Score: ",recall)
print("F1 Score: ",f1)
```

```
Before Fine Tuning
Accuracy:  0.9997618417570786
Precision Score:  0.99975553868720 86
Recall Score:  0.9997618417570786
F1 Score:  0.9997581792615621
```

Classification Report After Fine Tuning

```
accuracy = accuracy_score(y_test, new_y_pred)
precision = precision_score(y_test, new_y_pred,average='weighted')
recall = recall_score(y_test, new_y_pred,average='weighted')
f1 = f1_score(y_test, new_y_pred,average='weighted')
```

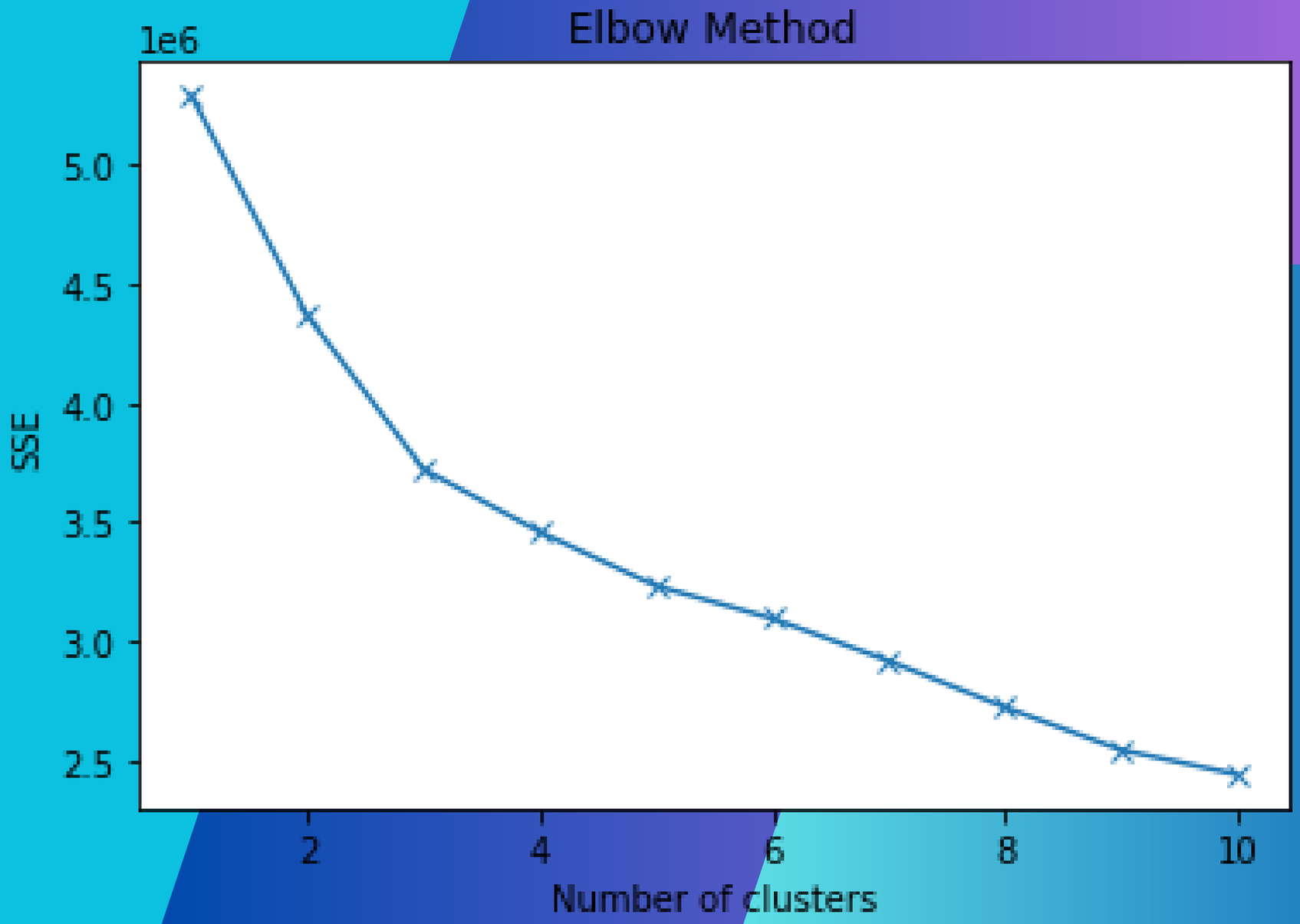
[105]

```
print("After Fine Tuning")
print("Accuracy of Fine-Tuned MLP: ", accuracy)
print("Precision Score of Fine-Tuned MLP: ", precision)
print("Recall Score of Fine-Tuned MLP: ",recall)
print("F1 Score of Fine-Tuned MLP: ",f1)
```

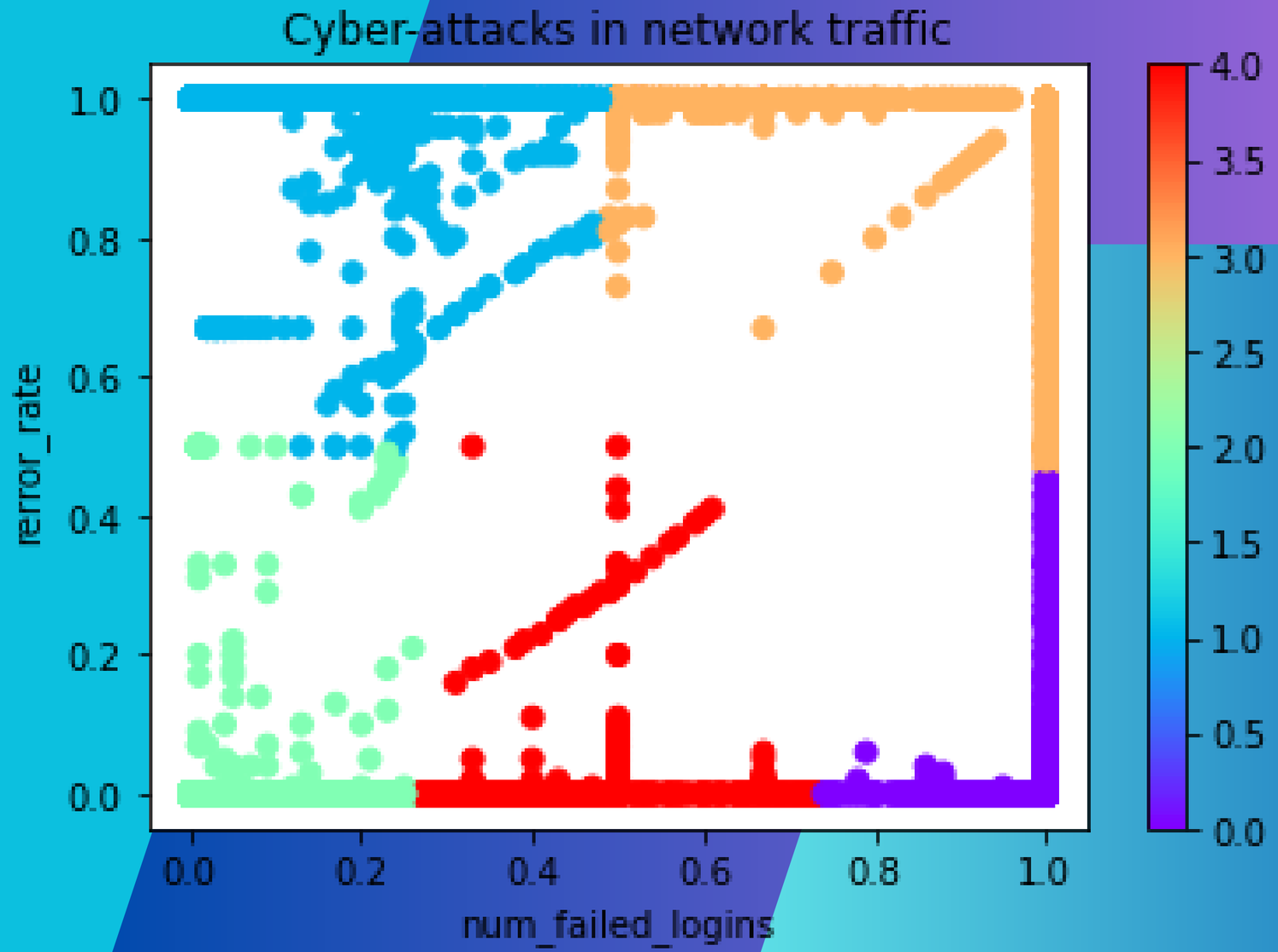
[106]

```
After Fine Tuning
Accuracy of Fine-Tuned MLP:  0.9998412278380524
Precision Score of Fine-Tuned MLP:  0.9998431177364935
Recall Score of Fine-Tuned MLP:  0.9998412278380524
F1 Score of Fine-Tuned MLP:  0.999841580540142
```

Elbow Method



K-Mean Clustering



Conclusion

- The MLP Classifier achieved the highest accuracy among the classification algorithms.
- The KMeans clustering algorithm was applied to the dataset, and the number of clusters was determined using the elbow method.

Thank You!