Lampiran 3 Kode Program Model ddos.ipynb

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler,
MinMaxScaler, RobustScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import IsolationForest,
RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,  confusion_matrix,
roc_auc_score, classification_report,
precision_score,recall_score, f1_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from collections import Counter
import time

df = pd.read_parquet('ddos-collection.parquet')
df = df.drop(columns='Label')
df.head()
df.info()
df.keys()
df['ClassLabel'].value_counts()

# List of labels and their corresponding counts
labels_and_counts = {
    'Benign': 7186189,
    'DDoS': 1234729,
    'DoS': 397344,
    'Botnet': 145968,
    'Bruteforce': 103244,
    'Infiltration': 94857,
    'Webattack': 2995,
    'Portscan': 2255
}

# Check if rows with each label are unique
for label, count in labels_and_counts.items():
    rows_with_label = df[df['ClassLabel'] == label]

    if not rows_with_label.duplicated().any():
        print(f"All {count} rows with label '{label}' are
unique.")
    else:
        print(f"There are duplicates in {count} rows with label
'{label}'.")
        print(rows_with_label[rows_with_label.duplicated()])
    print("="*50)
```

```
# Hapus data duplicate
df.drop_duplicates(subset=df.columns[:-1], keep='first')
df.shape

# List of labels to keep
labels_to_keep = ['Benign', 'DoS', 'DDoS']

# Filter the DataFrame to only keep rows with specified labels
df = df[df['ClassLabel'].isin(labels_to_keep)]

# Replacing infinite values with NaN in DataFrame
df.replace([np.inf, -np.inf], np.nan, inplace=True)

#  Dropping NaN values
df.dropna(inplace=True)

# Create a larger and clearer figure
plt.figure(figsize=(16, 8))

# Distribution of the target variable
sns.countplot(x='ClassLabel', data=df, palette='viridis')

# Set title and labels
#plt.title('Distribution of the Target Labels', fontsize=16)
plt.xlabel('Target Labels', fontsize=20)
plt.ylabel('Count', fontsize=20)

# Increase the font size of x-axis tick labels
plt.xticks(fontsize=20)  # Adjust the font size according to your
preference

plt.savefig('Target_distribution_after_drop.png')
# Show the plot
plt.show()

df['ClassLabel'] = df['ClassLabel'].replace({
    'Benign': 0,
    'DDoS': 1,
    'DoS': 1
})

df['ClassLabel'].value_counts()

# Create a larger and clearer figure
plt.figure(figsize=(16, 8))

# Distribution of the target variable
sns.countplot(x='ClassLabel', data=df, palette='viridis')

# Set title and labels
#plt.title('Distribution of the Target Labels', fontsize=16)
plt.xlabel('Target Labels', fontsize=20)
plt.ylabel('Count', fontsize=20)

# Increase the font size of x-axis tick labels
plt.xticks(fontsize=20)  # Adjust the font size according to your
preference
```

```
plt.savefig('Target_distribution_after_drop.png')
# Show the plot
plt.show()

feature_mapping = {
    'Fwd Packet Length Std': 'fwd_pkt_len_std',
    'Total Backward Packets': 'tot_bwd_pkts',
    'Bwd Packets Length Total': 'totlen_bwd_pkts',
    'Fwd Act Data Packets': 'fwd_act_data_pkts',
    'Bwd IAT Total': 'bwd_iat_tot',
    'Bwd Packet Length Mean': 'bwd_pkt_len_mean',
    'Idle Mean': 'idle_mean',
    'Flow IAT Max': 'flow_iat_max'
}
df.rename(columns=feature_mapping, inplace=True)
features = list(feature_mapping.values())
print(len(features))
features

df[features].dtypes

X = df[features]
y = df['ClassLabel']

# Undersampling
rus = RandomUnderSampler(random_state=42)
X_resampled_under, y_resampled_under = rus.fit_resample(X, y)
print(f"Distribusi kelas setelah undersampling:
{Counter(y_resampled_under)}")

X[df['ClassLabel'] == 1]

print(X.shape)
X.columns

# Split the data into training, testing, and validation sets
X_train, X_temp, y_train, y_temp =
train_test_split(X_resampled_under, y_resampled_under,
test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp,
test_size=0.33, random_state=42)

scaler = StandardScaler()

# Fit and transform the training data
X_train_normalized = scaler.fit_transform(X_train)

# Transform the validation data
X_val_normalized = scaler.transform(X_val)

# Random Forest
rf_model = RandomForestClassifier()
rf_start_time_train = time.time()
rf_model.fit(X_train_normalized, y_train)
rf_training_time = time.time() - rf_start_time_train

rf_start_time_pred = time.time()
rf_y_pred = rf_model.predict(X_val_normalized)
```

```
rf_y_prob = rf_model.predict_proba(X_val_normalized)
rf_prediction_time = time.time() - rf_start_time_pred
rf_accuracy = accuracy_score(y_val, rf_y_pred)
rf_classification_rep = classification_report(y_val, rf_y_pred)

cm = confusion_matrix(y_val, rf_y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g',
            xticklabels=['BENIGN', 'DDos'], yticklabels=['BENIGN',
'DDos'])
plt.title('RF Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Print Random Forest results
print("Random Forest Model")
print(f"Training time: {rf_training_time:.4f} seconds")
print(f"Prediction time: {rf_prediction_time:.4f} seconds")
print(f"Normalization Method: StandardScaler")
print(f"Accuracy: {rf_accuracy:.4f}")
# print(f"ROC AUC: {rf_roc_auc:.4f}")
print("Classification Report:\n", rf_classification_rep)
print("="*50)

# K-Nearest Neighbors
knn_model = KNeighborsClassifier()
knn_start_time_train = time.time()
knn_model.fit(X_train_normalized, y_train)
knn_training_time = time.time() - knn_start_time_train
knn_start_time_pred = time.time()
knn_y_pred = knn_model.predict(X_val_normalized)
knn_accuracy = accuracy_score(y_val, knn_y_pred)
knn_prediction_time = time.time() - knn_start_time_pred
knn_classification_rep = classification_report(y_val, knn_y_pred)

# Print K-Nearest Neighbors results
print("K-Nearest Neighbors Model")
print(f"Training time: {knn_training_time:.4f} seconds")
print(f"Prediction time: {knn_prediction_time:.4f} seconds")
print(f"Normalization Method: StandardScaler")
print(f"Accuracy: {knn_accuracy:.4f}")
print("Classification Report:\n", knn_classification_rep)
print("="*50)

cm = confusion_matrix(y_val, knn_y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g',
            xticklabels=['BENIGN', 'DDos'], yticklabels=['BENIGN',
'DDos'])
plt.title('KNN Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
from sklearn.naive_bayes import GaussianNB

# Assuming X_train and X_val are your training and validation
datasets

# Gaussian Naive Bayes
nb_model = GaussianNB()
nb_start_time_train = time.time()
nb_model.fit(X_train_normalized, y_train)
nb_training_time = time.time() - nb_start_time_train
nb_start_time_pred = time.time()
nb_y_pred = nb_model.predict(X_val_normalized)
nb_prediction_time = time.time() - nb_start_time_pred
nb_accuracy = accuracy_score(y_val, nb_y_pred)
nb_classification_rep = classification_report(y_val, nb_y_pred)

# Print Naive Bayes results
print("Gaussian Naive Bayes Model")
print(f"Training time: {nb_training_time:.4f} seconds")
print(f"Prediction time: {nb_prediction_time:.4f} seconds")
print(f"Normalization Method: StandardScaler")
print(f"Accuracy: {nb_accuracy:.4f}")
print("Classification Report:\n", nb_classification_rep)
print("="*50)

cm = confusion_matrix(y_val, knn_y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g',
            xticklabels=['BENIGN', 'DDos'], yticklabels=['BENIGN',
'DDos'])
plt.title('Naive Bayes Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

pickle.dump(rf_model, open("rf.pkl", "wb"))
pickle.dump(scaler, open("scaler.pkl", "wb"))
```