LAMPIRAN

```
import numpy as np
import pandas as pd
import time
import matplotlib.pyplot as plt
from scipy import stats
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import StandardScaler

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
nRowsRead = None

# Membaca file CSV
df = pd.read_csv('CICDDOS2019/cicddos2019_dataset.csv')

print(df.shape)

# Menghapus duplikat baris
df = df.drop_duplicates(keep="first")

# Menghapus baris yang mengandung nilai NaN
df.dropna(inplace=True)

# Menghapus kolom yang mengandung NaN
df = df.dropna(axis=1)

# Menghapus nilai NaN, infinite, dan -infinite
df = df[~df.isin([np.nan, np.inf, -np.inf]).any(axis=1)]

# Grupkan berdasarkan label dan ambil label yang memiliki jumlah data >
10000
df = df.groupby('Label').filter(lambda x: len(x) > 10000)

# Konversi tipe data kolom integer menjadi int32 dan kolom float menjadi
float32
integer_cols = []
float_cols = []
for col in df.columns[:-1]:
    if df[col].dtype == "int64":
        integer_cols.append(col)
    else:
        float_cols.append(col)

df[integer_cols] = df[integer_cols].astype("int32")
df[float_cols] = df[float_cols].astype("float32")

# Pisahkan fitur dan target
x = df.drop(['Label'], axis=1)
y = df['Label']

def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr(numeric_only=True)  # Perbaikan disini
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:
              colname = corr_matrix.columns[i]
              col_corr.add(colname)
    return col_corr
```

```
# Melakukan correlation feature selection pada dataset x
corr_features_x = correlation(x, 0.6)

print("Jumlah fitur dengan korelasi tinggi:", len(corr_features_x))

x.drop(corr_features_x, axis=1, inplace=True)

# Membuat instance RandomUnderSampler dan melakukan undersampling pada x
dan y
rus = RandomUnderSampler(random_state=0)
x_resampled, y_resampled = rus.fit_resample(x, y)

# Mengubah skala fitur menggunakan Z-score
cols = list(x_resampled.columns)
for col in cols:
    x_resampled[col] = stats.zscore(x_resampled[col])

# Split data menjadi data latih dan data uji
from sklearn.model_selection import train_test_split, cross_val_score
X_train, X_test, Y_train, Y_test = train_test_split(x_resampled,
y_resampled, test_size=0.3, random_state=0)

print(np.any(np.isnan(X_train)))
print(np.all(np.isfinite(X_train)))

# Membuat instance SimpleImputer dan melakukan imputasi missing values
dengan mean
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Membuat instance StandardScaler dan melakukan standarisasi fitur
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.ensemble import RandomForestClassifier as RFC

start = time.time()
Random_Forest = RFC(n_estimators=100, max_depth=40)
Random_Forest.fit(X_train, Y_train)
print("Time taken to train model: ", time.time()-start," seconds")

from sklearn import metrics
Predict_X =  Random_Forest.predict(X_train)
scores = cross_val_score(Random_Forest, X_train, Y_train, cv=7)
accuracy = metrics.accuracy_score(Y_train,Predict_X)
confusion_matrix = metrics.confusion_matrix(Y_train, Predict_X)
classification = metrics.classification_report(Y_train, Predict_X)

# Menghitung nilai presisi, recall, dan F1-score pada data latih
precision_train = metrics.precision_score(Y_train, Predict_X,
average='macro')
recall_train = metrics.recall_score(Y_train, Predict_X, average='macro')
f1_train = metrics.f1_score(Y_train, Predict_X, average='macro')

print()
print('--------------------------- Results ------------------------------
--')
print()
print("Cross Validation Mean Score:" "\n", scores.mean())
print()
```

```
print("Model Accuracy Train:" "\n", accuracy)
print()
print("Confusion matrix:" "\n", confusion_matrix)
print()
print("Classification report:" "\n", classification)
print()
print("Train Precision: {:.2f}%".format( precision_train*100))
print("Train Recall: {:.2f}%".format( recall_train*100))
print("Train F1-score: {:.2f}%".format( f1_train*100))

def plot_confusion_matrix(cm, title, cmap=None, target=None,
normalize=False):

    import itertools
    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('viridis')
    plt.figure(figsize=(20, 20))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target is not None:
        ticks = np.arange(len(target))
        plt.xticks(ticks, target, rotation=45)
        plt.yticks(ticks, target)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="black" if cm[i, j] > thresh else "white")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="black" if cm[i, j] > thresh else "white")
    plt.grid(False)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f};
misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()
    plt.savefig(title, bbox_inches='tight', dpi=300)

plot_confusion_matrix(cm=confusion_matrix, title='Random Forest
Classification')

Predict_X =  Random_Forest.predict(X_test)
scores = cross_val_score(Random_Forest, X_test, Y_test, cv=7)
accuracy = metrics.accuracy_score(Y_test,Predict_X)
confusion_matrix = metrics.confusion_matrix(Y_test, Predict_X)
classification = metrics.classification_report(Y_test, Predict_X)

# Menghitung nilai presisi, recall, dan F1-score pada data uji
precision_test = metrics.precision_score(Y_test, Predict_X,
average='macro')
recall_test = metrics.recall_score(Y_test, Predict_X, average='macro')
```

```
f1_test = metrics.f1_score(Y_test, Predict_X, average='macro')

print()
print('---------------------------- Results -------------------------------
--')
print()
print ("Cross Validation Mean Score:" "\n", scores.mean())
print()
print ("Model Accuracy Test:{:.2f}%".format(accuracy*100))
print()
print("Confusion matrix:" "\n", confusion_matrix)
print()
print("Classification report:" "\n", classification)
print()
print("Test Precision: {:.2f}%".format(precision_test*100))
print("Test Recall: {:.2f}%".format(recall_test*100))
print("Test F1-score: {:.2f}%".format(f1_test*100))

def plot_confusion_matrix(cm, title, cmap=None, target=None,
normalize=False):

    import itertools
    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('viridis')
    plt.figure(figsize=(20, 20))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target is not None:
        ticks = np.arange(len(target))
        plt.xticks(ticks, target, rotation=45)
        plt.yticks(ticks, target)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="black" if cm[i, j] > thresh else "white")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="black" if cm[i, j] > thresh else "white")
    plt.grid(False)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f};
misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()
    plt.savefig(title, bbox_inches='tight', dpi=300)

plot_confusion_matrix(cm=confusion_matrix, title='Random Forest
Classification')
```