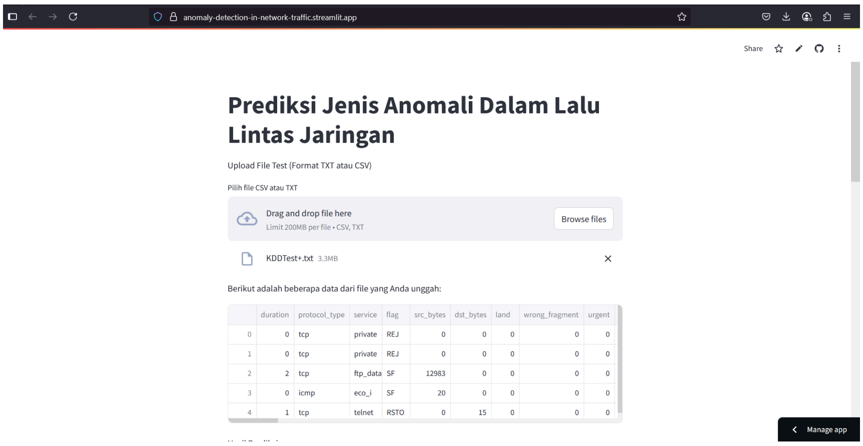
Lampiran

- **User Manual Guide**
  - Deskripsi umum:

    Aplikasi ini digunakan untuk memprediksi jenis anomali dalam lalu lintas jaringan komputer dengan memanfaatkan model *Support Vector Machine* (SVM). Pendekatan klasifikasi *multiclass* dilakukan dengan skema *One-vs-One*, sedangkan proses pelatihan model menggunakan metode optimasi *Primal Estimated sub-GrAdient SOlver for SVM* (Pegasos), yang menerapkan teknik *Stochastic Gradient Descent* (SGD) untuk menyelesaikan fungsi objektif SVM secara efisien. Kernel yang digunakan adalah *Radial Basis Function* (RBF), yang memungkinkan pemisahan kelas pada data non-linear.
  - Akses Aplikasi: https://anomaly-detection-in-network-traffic.streamlit.app
  - Fitur Utama:
    1. Upload file txt/csv
    2. Tabel hasil prediksi
    3. Visualisasi distribusi label
  - Cara Menggunakan:
    1. Download dataset KDDTest+.txt di kaggle untuk prediksi: https://www.kaggle.com/datasets/hassan06/nslkdd
    2. Buka url streamlit: https://anomaly-detection-in-network-traffic.streamlit.app
    3. Masukkan file KDDTest+.txt yang telah di download ke dalam streamlit
    4. Tunggu hingga hasil prediksi dan visualisasi distribusi label muncul
    5. Hasil muncul dalam bentuk tabel dan grafik
  - Contoh Tampilan:



Prediksi Jenis Anomali Dalam Lalu Lintas Jaringan

Upload File Test (Format TXT atau CSV)

Pilih file CSV atau TXT

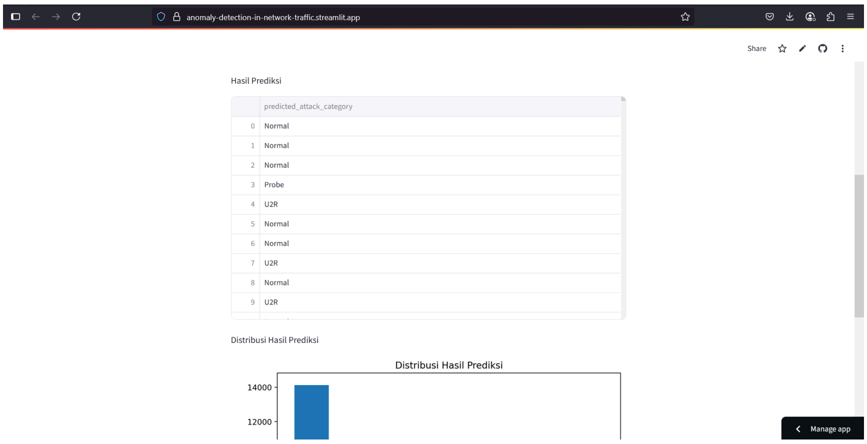
Drag and drop file here
Limit 200MB per file • CSV, TXT

KDDTest+.txt 3.3MB

Berikut adalah beberapa data dari file yang Anda unggah:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent
0	0	tcp	private	REJ	0	0	0	0	0
1	0	tcp	private	REJ	0	0	0	0	0
2	2	tcp	ftp_data	SF	12983	0	0	0	0
3	0	icmp	eco_i	SF	20	0	0	0	0
4	1	tcp	telnet	RSTO	0	15	0	0	0

○

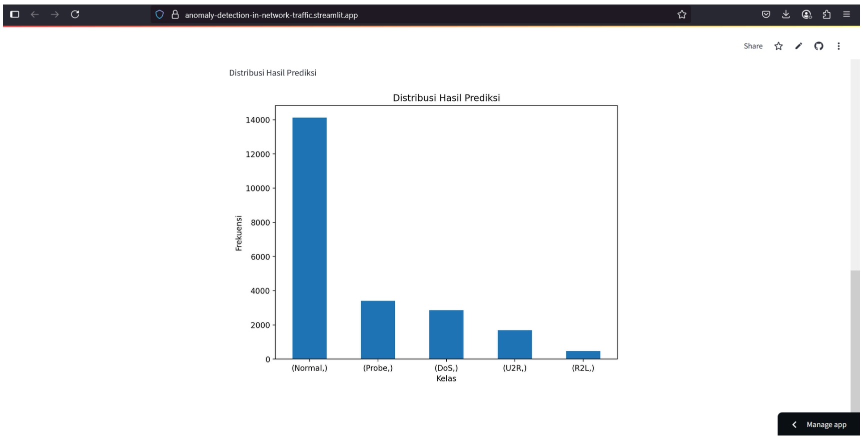

anomaly-detection-in-network-traffic.streamlit.app
Share
Manage app

Hasil Prediksi

	predicted_attack_category
0	Normal
1	Normal
2	Normal
3	Probe
4	U2R
5	Normal
6	Normal
7	U2R
8	Normal
9	U2R

Distribusi Hasil Prediksi

Distribusi Hasil Prediksi
14000
12000

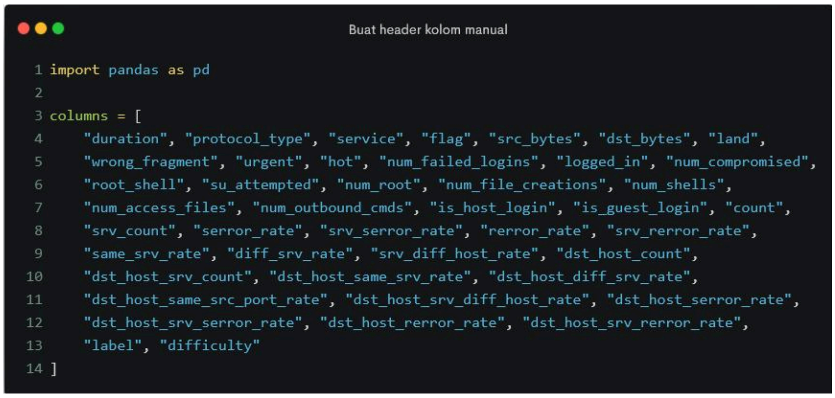


Distribusi Hasil Prediksi

- **Listing Code**

1. File preprosBaru.ipynb – Pemrosesan Data dan Klasifikasi Anomali

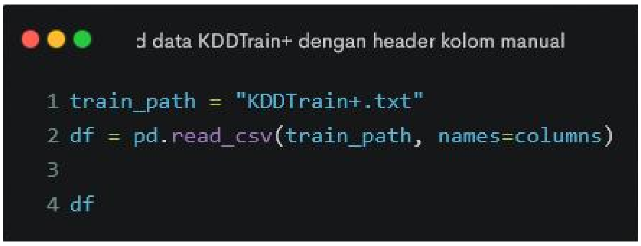
   a. Buat *header* kolom manual



Buat header kolom manual

```python
import pandas as pd

columns = [
    "duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land",
    "wrong_fragment", "urgent", "hot", "num_failed_logins", "logged_in", "num_compromised",
    "root_shell", "su_attempted", "num_root", "num_file_creations", "num_shells",
    "num_access_files", "num_outbound_cmds", "is_host_login", "is_guest_login", "count",
    "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate",
    "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate", "dst_host_count",
    "dst_host_srv_count", "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
    "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",
    "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate",
    "label", "difficulty"
]
```

   b. *Load* data KDDTrain+ dengan *header* kolom manual

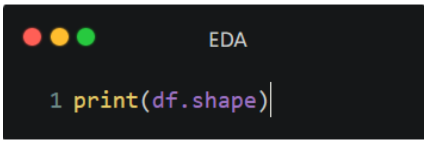


d data KDDTrain+ dengan header kolom manual

```python
train_path = "KDDTrain+.txt"
df = pd.read_csv(train_path, names=columns)

df
```

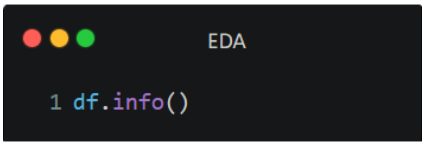
   c. EDA

      i. Cek jumlah baris dan kolom



```python
print(df.shape)
```

ii. Cek ringkasan informasi struktur DataFrame

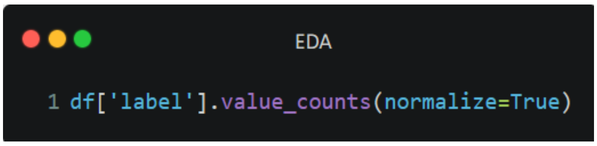


```python
df.info()
```

iii. Cek statistika deskriptif untuk kolom numerik

```python
df.describe()
```

iv. Cek distribusi class

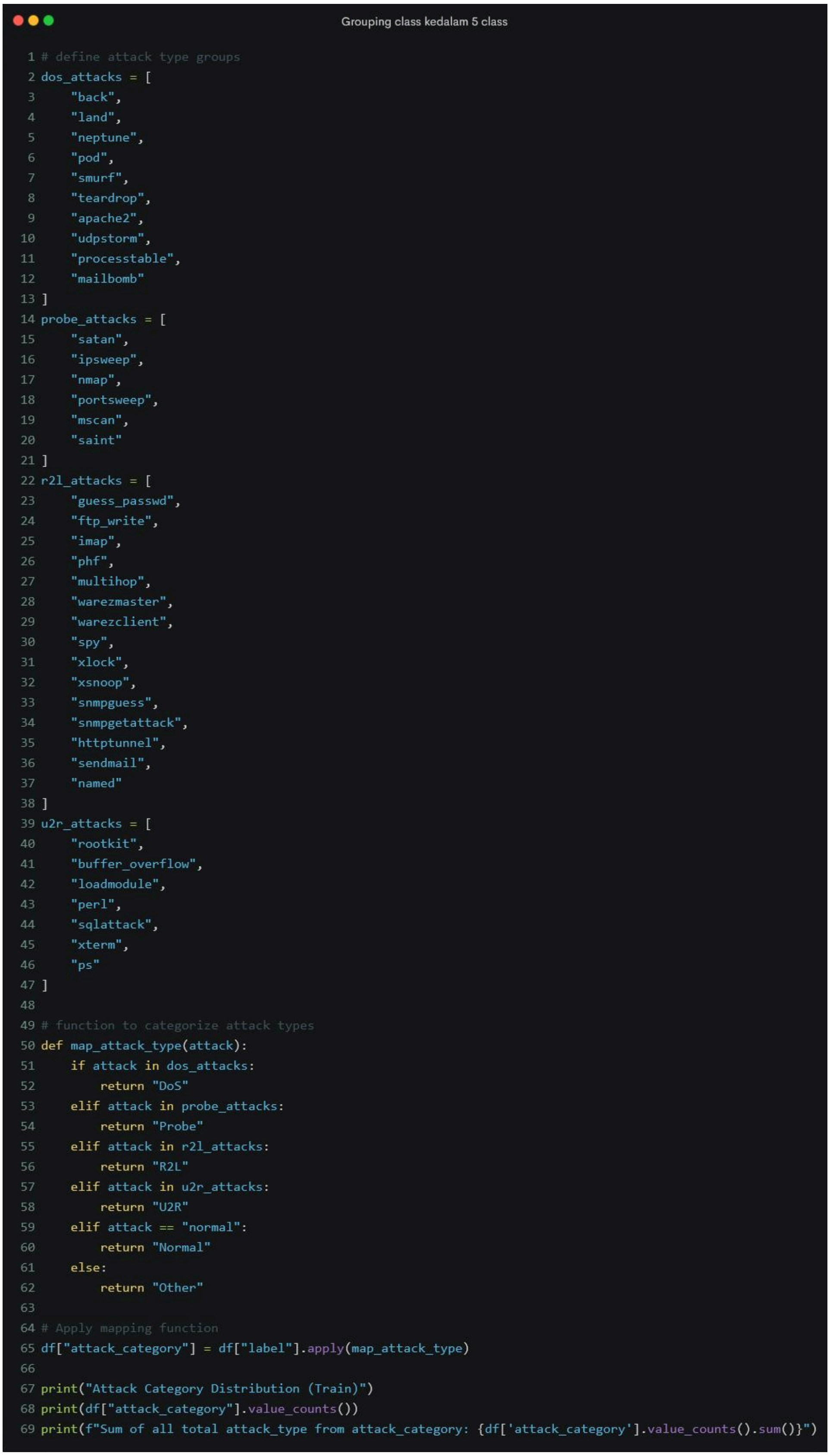


```python
df['label'].value_counts(normalize=True)
```

v. Plot bar chart

```python
from matplotlib.ticker import FuncFormatter

label_counts = df['label'].value_counts()
def format_k(x, pos):
    if x >= 1000:
        return f'{x/1000:.1f}k'
    return int(x)

plt.figure(figsize=(10, 6))
bars = plt.bar(label_counts.index, label_counts.values, color='#1e77b4')

plt.title('Distribusi Jumlah Label (Bar Chart)', pad=15)
plt.xlabel('Label')
plt.ylabel('Jumlah (ribuan)')
plt.ylim(0, max(label_counts.values) * 1.1)
plt.xticks(rotation=45)

plt.gca().yaxis.set_major_formatter(FuncFormatter(format_k))

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

d. Mengelompokkan semua label menjadi 5 label

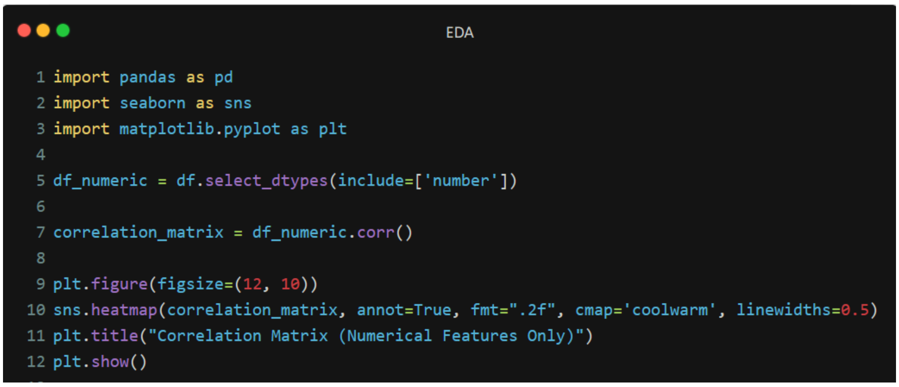


Grouping class kedalam 5 class

```python
# define attack type groups
dos_attacks = [
    "back",
    "land",
    "neptune",
    "pod",
    "smurf",
    "teardrop",
    "apache2",
    "udpstorm",
    "processtable",
    "mailbomb"
]
probe_attacks = [
    "satan",
    "ipsweep",
    "nmap",
    "portsweep",
    "mscan",
    "saint"
]
r2l_attacks = [
    "guess_passwd",
    "ftp_write",
    "imap",
    "phf",
    "multihop",
    "warezmaster",
    "warezclient",
    "spy",
    "xlock",
    "xsnoop",
    "snmpguess",
    "snmpgetattack",
    "httptunnel",
    "sendmail",
    "named"
]
u2r_attacks = [
    "rootkit",
    "buffer_overflow",
    "loadmodule",
    "perl",
    "sqlattack",
    "xterm",
    "ps"
]

# function to categorize attack types
def map_attack_type(attack):
    if attack in dos_attacks:
        return "DoS"
    elif attack in probe_attacks:
        return "Probe"
    elif attack in r2l_attacks:
        return "R2L"
    elif attack in u2r_attacks:
        return "U2R"
    elif attack == "normal":
        return "Normal"
    else:
        return "Other"

# Apply mapping function
df["attack_category"] = df["label"].apply(map_attack_type)

print("Attack Category Distribution (Train)")
print(df["attack_category"].value_counts())
print(f"Sum of all total attack_type from attack_category: {df['attack_category'].value_counts().sum()}")
```

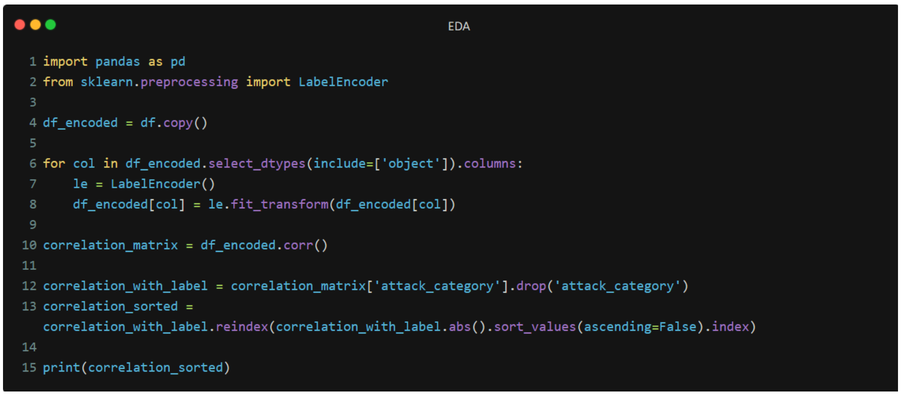
e. EDA

i. Cek korelasi matriks



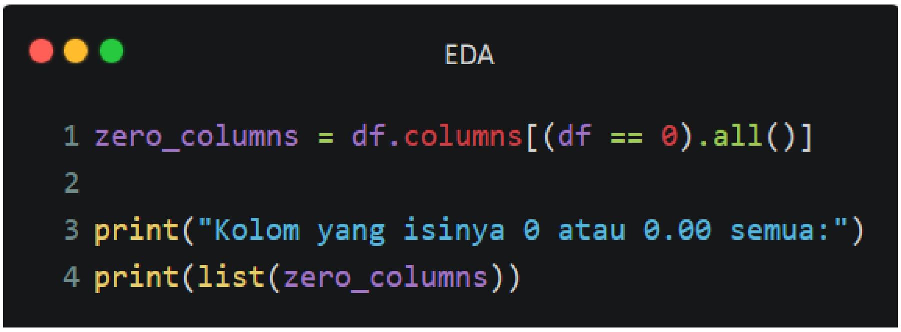
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df_numeric = df.select_dtypes(include=['number'])

correlation_matrix = df_numeric.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Matrix (Numerical Features Only)")
plt.show()
```

ii. Encoding kolom kategorikal, lalu menghitung dan mengurutkan korelasi di kolom attack_category



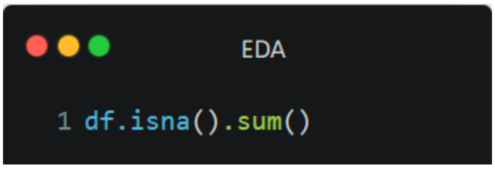
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df_encoded = df.copy()

for col in df_encoded.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])

correlation_matrix = df_encoded.corr()

correlation_with_label = correlation_matrix['attack_category'].drop('attack_category')
correlation_sorted = correlation_with_label.reindex(correlation_with_label.abs().sort_values(ascending=False).index)

print(correlation_sorted)
```

iii. Cek kolom dengan nilai yang berisi 0 atau 0.00



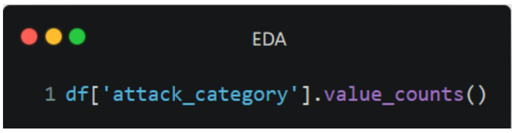
```python
zero_columns = df.columns[(df == 0).all()]

print("Kolom yang isinya 0 atau 0.00 semua:")
print(list(zero_columns))
```

iv. Cek Nan Values



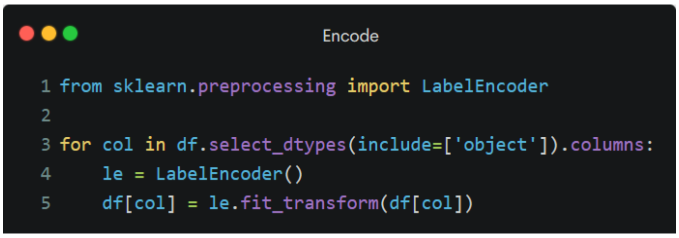
```python
df.isna().sum()
```

v. Cek jumlah kemunculan dari setiap kategori serangan di kolom attack_category



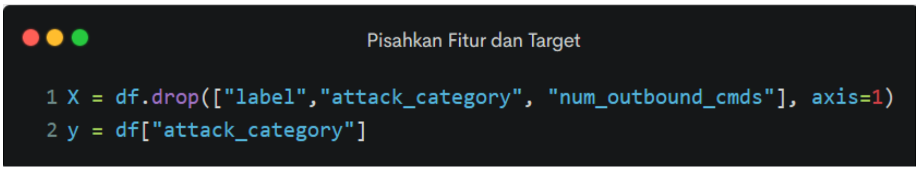
```python
df['attack_category'].value_counts()
```

f. Encode



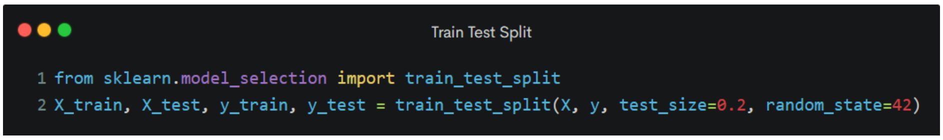
```python
from sklearn.preprocessing import LabelEncoder

for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

g. Pisahkan fitur dan target



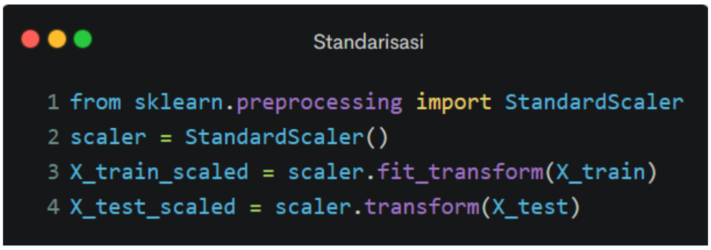
```python
X = df.drop(["label","attack_category", "num_outbound_cmds"], axis=1)
y = df["attack_category"]
```

h. Split data



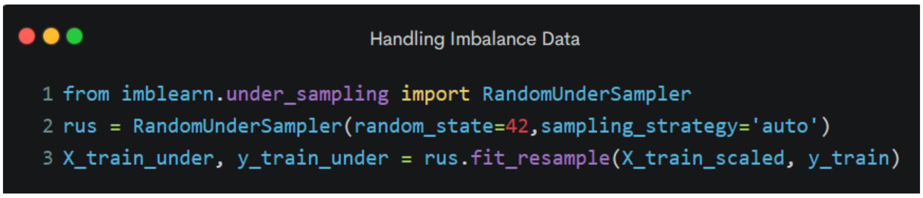
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

i. Standarisasi



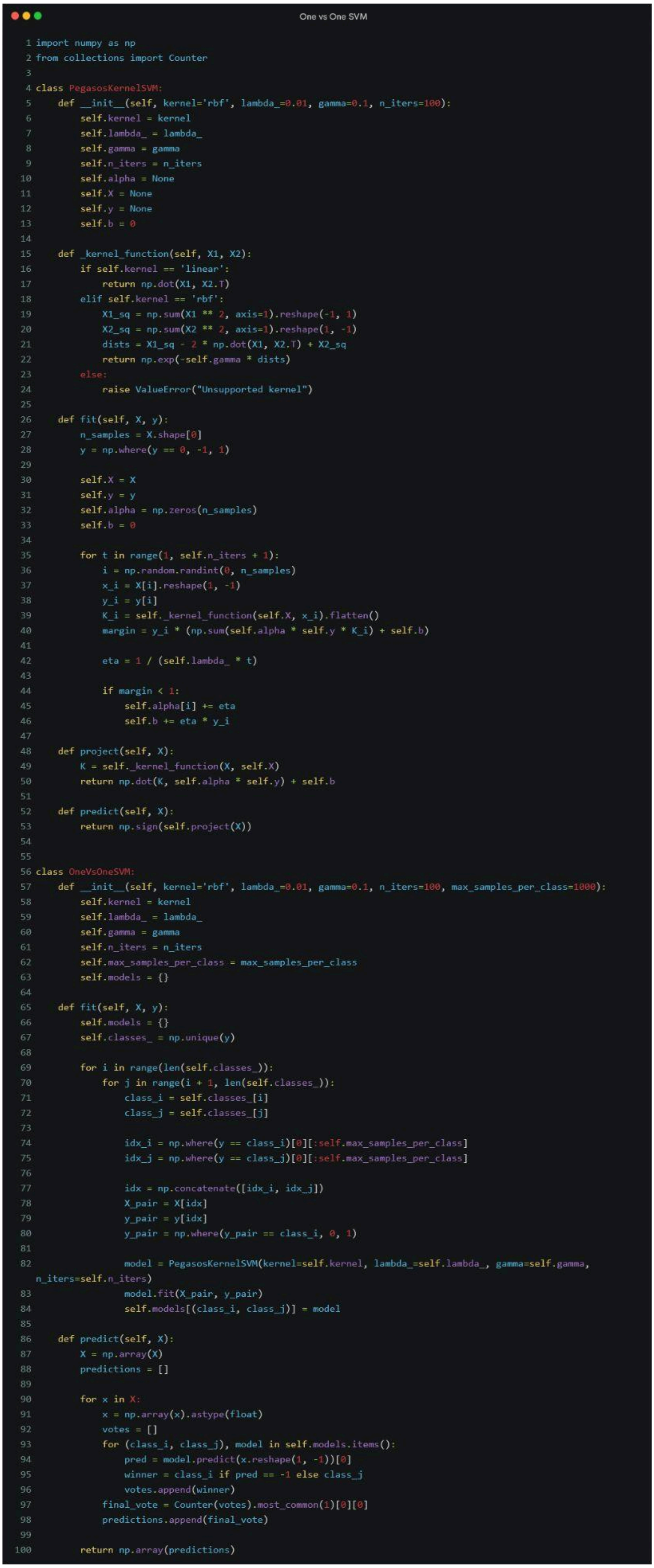
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

j. Handling Imbalance Data



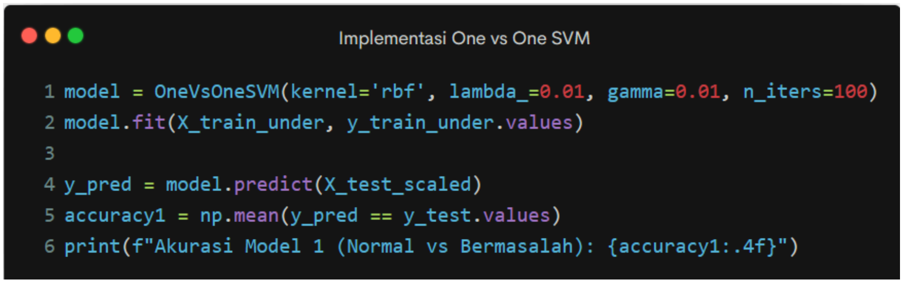
```python
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42,sampling_strategy='auto')
X_train_under, y_train_under = rus.fit_resample(X_train_scaled, y_train)
```

k. Model SVM



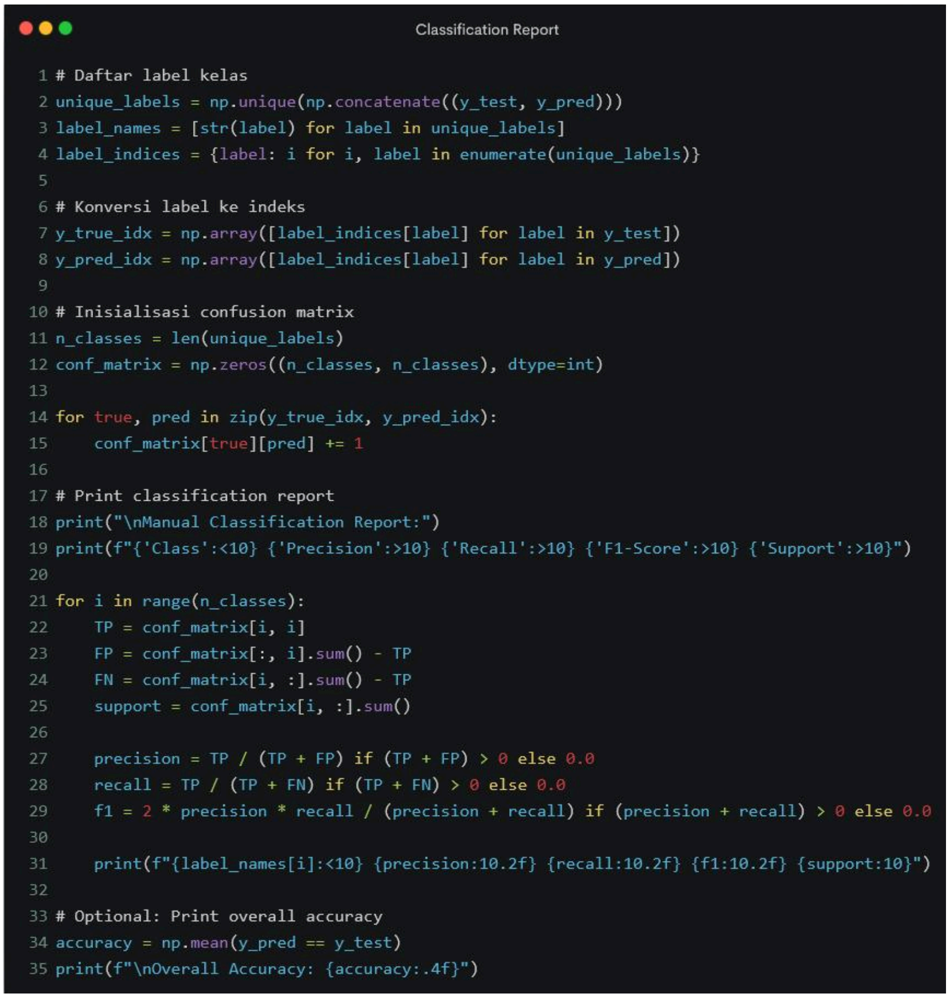
```python
import numpy as np
from collections import Counter

class PegasosKernelSVM:
    def __init__(self, kernel='rbf', lambda_=0.01, gamma=0.1, n_iters=100):
        self.kernel = kernel
        self.lambda_ = lambda_
        self.gamma = gamma
        self.n_iters = n_iters
        self.alpha = None
        self.X = None
        self.y = None
        self.b = 0

    def _kernel_function(self, X1, X2):
        if self.kernel == 'linear':
            return np.dot(X1, X2.T)
        elif self.kernel == 'rbf':
            X1_sq = np.sum(X1 ** 2, axis=1).reshape(-1, 1)
            X2_sq = np.sum(X2 ** 2, axis=1).reshape(1, -1)
            dists = X1_sq - 2 * np.dot(X1, X2.T) + X2_sq
            return np.exp(-self.gamma * dists)
        else:
            raise ValueError("Unsupported kernel")

    def fit(self, X, y):
        n_samples = X.shape[0]
        y = np.where(y == 0, -1, 1)

        self.X = X
        self.y = y
        self.alpha = np.zeros(n_samples)
        self.b = 0

        for t in range(1, self.n_iters + 1):
            i = np.random.randint(0, n_samples)
            x_i = X[i].reshape(1, -1)
            y_i = y[i]
            K_i = self._kernel_function(self.X, x_i).flatten()
            margin = y_i * (np.sum(self.alpha * self.y * K_i) + self.b)

            eta = 1 / (self.lambda_ * t)

            if margin < 1:
                self.alpha[i] += eta
                self.b += eta * y_i

    def project(self, X):
        K = self._kernel_function(X, self.X)
        return np.dot(K, self.alpha * self.y) + self.b

    def predict(self, X):
        return np.sign(self.project(X))


class OneVsOneSVM:
    def __init__(self, kernel='rbf', lambda_=0.01, gamma=0.1, n_iters=100, max_samples_per_class=1000):
        self.kernel = kernel
        self.lambda_ = lambda_
        self.gamma = gamma
        self.n_iters = n_iters
        self.max_samples_per_class = max_samples_per_class
        self.models = {}

    def fit(self, X, y):
        self.models = {}
        self.classes_ = np.unique(y)

        for i in range(len(self.classes_)):
            for j in range(i + 1, len(self.classes_)):
                class_i = self.classes_[i]
                class_j = self.classes_[j]

                idx_i = np.where(y == class_i)[0][:self.max_samples_per_class]
                idx_j = np.where(y == class_j)[0][:self.max_samples_per_class]

                idx = np.concatenate([idx_i, idx_j])
                X_pair = X[idx]
                y_pair = y[idx]
                y_pair = np.where(y_pair == class_i, 0, 1)

                model = PegasosKernelSVM(kernel=self.kernel, lambda_=self.lambda_, gamma=self.gamma,
n_iters=self.n_iters)
                model.fit(X_pair, y_pair)
                self.models[(class_i, class_j)] = model

    def predict(self, X):
        X = np.array(X)
        predictions = []

        for x in X:
            x = np.array(x).astype(float)
            votes = []
            for (class_i, class_j), model in self.models.items():
                pred = model.predict(x.reshape(1, -1))[0]
                winner = class_i if pred == -1 else class_j
                votes.append(winner)
            final_vote = Counter(votes).most_common(1)[0][0]
            predictions.append(final_vote)

        return np.array(predictions)
```

l. Implementasi model SVM



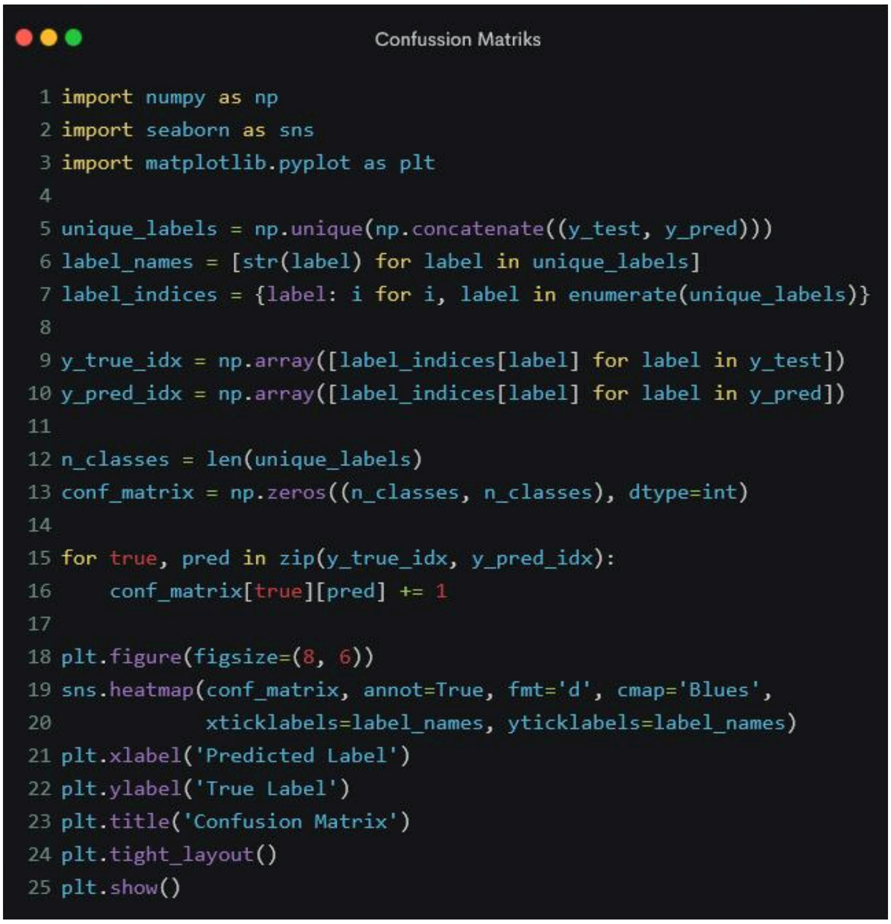
```
model = OneVsOneSVM(kernel='rbf', lambda_=0.01, gamma=0.01, n_iters=100)
model.fit(X_train_under, y_train_under.values)

y_pred = model.predict(X_test_scaled)
accuracy1 = np.mean(y_pred == y_test.values)
print(f"Akurasi Model 1 (Normal vs Bermasalah): {accuracy1:.4f}")
```

m. Classification report



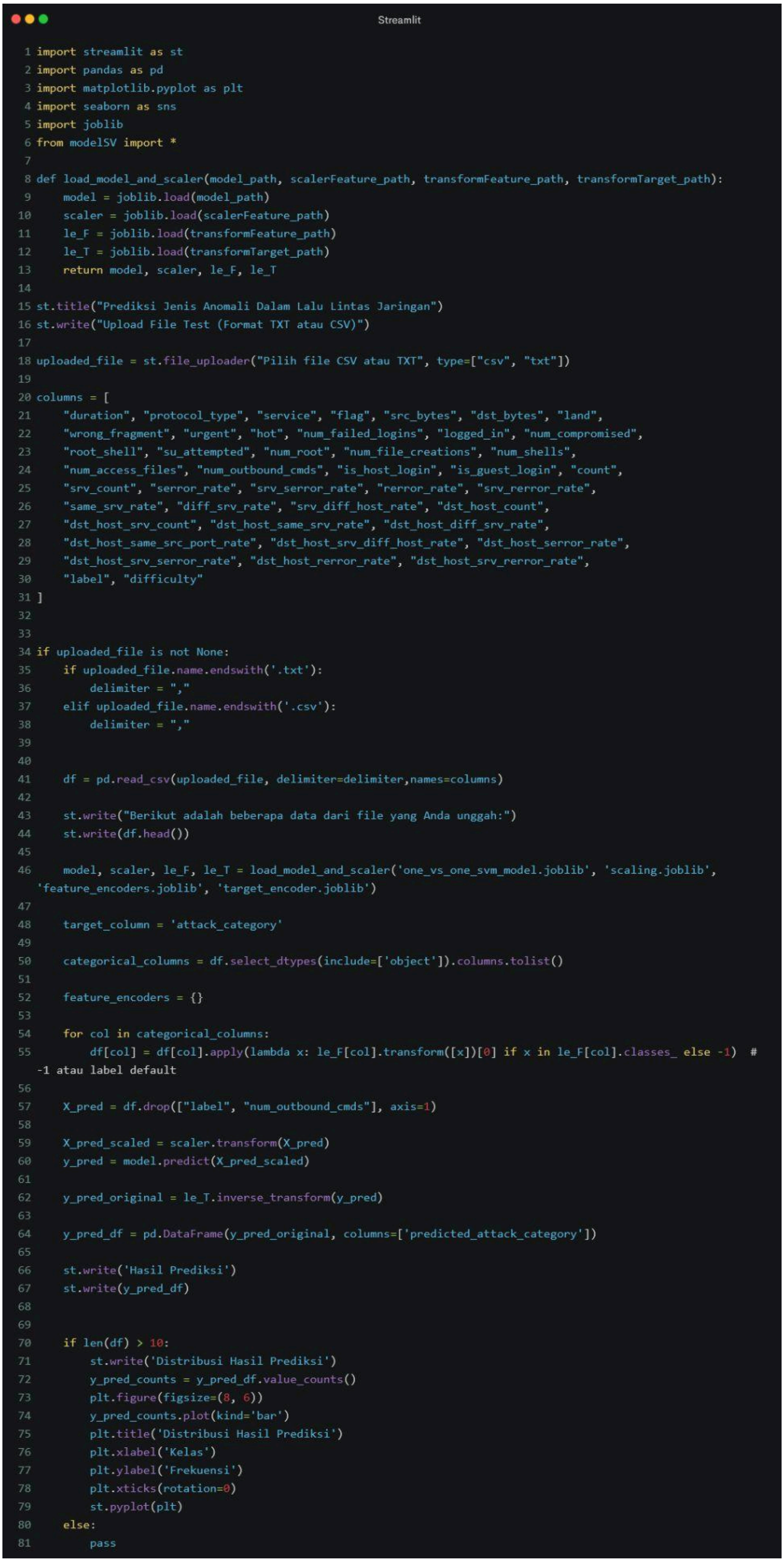
```
# Daftar label kelas
unique_labels = np.unique(np.concatenate((y_test, y_pred)))
label_names = [str(label) for label in unique_labels]
label_indices = {label: i for i, label in enumerate(unique_labels)}

# Konversi label ke indeks
y_true_idx = np.array([label_indices[label] for label in y_test])
y_pred_idx = np.array([label_indices[label] for label in y_pred])

# Inisialisasi confusion matrix
n_classes = len(unique_labels)
conf_matrix = np.zeros((n_classes, n_classes), dtype=int)

for true, pred in zip(y_true_idx, y_pred_idx):
    conf_matrix[true][pred] += 1

# Print classification report
print("\nManual Classification Report:")
print(f"{'Class':<10} {'Precision':>10} {'Recall':>10} {'F1-Score':>10} {'Support':>10}")

for i in range(n_classes):
    TP = conf_matrix[i, i]
    FP = conf_matrix[:, i].sum() - TP
    FN = conf_matrix[i, :].sum() - TP
    support = conf_matrix[i, :].sum()

    precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
    recall = TP / (TP + FN) if (TP + FN) > 0 else 0.0
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0.0

    print(f"{label_names[i]:<10} {precision:10.2f} {recall:10.2f} {f1:10.2f} {support:10}")

# Optional: Print overall accuracy
accuracy = np.mean(y_pred == y_test)
print(f"\nOverall Accuracy: {accuracy:.4f}")
```

n. Confusion matriks



```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

unique_labels = np.unique(np.concatenate((y_test, y_pred)))
label_names = [str(label) for label in unique_labels]
label_indices = {label: i for i, label in enumerate(unique_labels)}

y_true_idx = np.array([label_indices[label] for label in y_test])
y_pred_idx = np.array([label_indices[label] for label in y_pred])

n_classes = len(unique_labels)
conf_matrix = np.zeros((n_classes, n_classes), dtype=int)

for true, pred in zip(y_true_idx, y_pred_idx):
    conf_matrix[true][pred] += 1

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()
```

2. File app.py – Deploy via Streamlit



```python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
from modelSV import *

def load_model_and_scaler(model_path, scalerFeature_path, transformFeature_path, transformTarget_path):
    model = joblib.load(model_path)
    scaler = joblib.load(scalerFeature_path)
    le_F = joblib.load(transformFeature_path)
    le_T = joblib.load(transformTarget_path)
    return model, scaler, le_F, le_T

st.title("Prediksi Jenis Anomali Dalam Lalu Lintas Jaringan")
st.write("Upload File Test (Format TXT atau CSV)")

uploaded_file = st.file_uploader("Pilih file CSV atau TXT", type=["csv", "txt"])

columns = [
    "duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land",
    "wrong_fragment", "urgent", "hot", "num_failed_logins", "logged_in", "num_compromised",
    "root_shell", "su_attempted", "num_root", "num_file_creations", "num_shells",
    "num_access_files", "num_outbound_cmds", "is_host_login", "is_guest_login", "count",
    "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate",
    "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate", "dst_host_count",
    "dst_host_srv_count", "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
    "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",
    "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate",
    "label", "difficulty"
]


if uploaded_file is not None:
    if uploaded_file.name.endswith('.txt'):
        delimiter = ","
    elif uploaded_file.name.endswith('.csv'):
        delimiter = ","


    df = pd.read_csv(uploaded_file, delimiter=delimiter,names=columns)

    st.write("Berikut adalah beberapa data dari file yang Anda unggah:")
    st.write(df.head())

    model, scaler, le_F, le_T = load_model_and_scaler('one_vs_one_svm_model.joblib', 'scaling.joblib', 'feature_encoders.joblib', 'target_encoder.joblib')

    target_column = 'attack_category'

    categorical_columns = df.select_dtypes(include=['object']).columns.tolist()

    feature_encoders = {}

    for col in categorical_columns:
        df[col] = df[col].apply(lambda x: le_F[col].transform([x])[0] if x in le_F[col].classes_ else -1)  # -1 atau label default

    X_pred = df.drop(["label", "num_outbound_cmds"], axis=1)

    X_pred_scaled = scaler.transform(X_pred)
    y_pred = model.predict(X_pred_scaled)

    y_pred_original = le_T.inverse_transform(y_pred)

    y_pred_df = pd.DataFrame(y_pred_original, columns=['predicted_attack_category'])

    st.write('Hasil Prediksi')
    st.write(y_pred_df)


    if len(df) > 10:
        st.write('Distribusi Hasil Prediksi')
        y_pred_counts = y_pred_df.value_counts()
        plt.figure(figsize=(8, 6))
        y_pred_counts.plot(kind='bar')
        plt.title('Distribusi Hasil Prediksi')
        plt.xlabel('Kelas')
        plt.ylabel('Frekuensi')
        plt.xticks(rotation=0)
        st.pyplot(plt)
    else:
        pass
```