LAMPIRAN

Lampiran 1. Format Data File Log eve.json


```
{
  "timestamp": "2024-01-02T14:37:32.472831+0800",
  "flow_id": 1255752592013055,
  "in_iface": "\\Device\\NPF_{34F31CFF-0F79-4655-BF50-BD7EB562C30C}",
  "event_type": "alert",
  "src_ip": "192.168.10.61",
  "src_port": 58888,
  "dest_ip": "192.168.10.128",
  "dest_port": 445,
  "proto": "TCP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1000001,
    "rev": 1,
    "signature": "Nmap scan detected",
    "category": "Attempted Information Leak",
    "severity": 2
  },
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 0,
    "bytes_toserver": 66,
    "bytes_toclient": 0,
    "start": "2024-01-02T14:37:32.472831+0800"
  }
}
{
  "timestamp": "2024-01-02T14:37:37.245532+0800",
  "flow_id": 598467973342777,
  "in_iface": "\\Device\\NPF_{34F31CFF-0F79-4655-BF50-BD7EB562C30C}",
  "event_type": "flow",
  "src_ip": "192.168.10.23",
  "src_port": 5353,
  "dest_ip": "224.0.0.251",
  "dest_port": 5353,
  "proto": "UDP",
  "app_proto": "failed",
  "flow": {
    "pkts_toserver": 2,
    "pkts_toclient": 0,
    "bytes_toserver": 200,
    "bytes_toclient": 0,
    "start": "2024-01-02T14:36:37.058937+0800",
    "end": "2024-01-02T14:36:37.058937+0800",
    "age": 0,
    "state": "new",
    "reason": "timeout",
    "alerted": false
  }
}
```

Lampiran 2. Sintaks Kode Parsing Data Json ke CSV

```
#Ubah Dataset Json ke Csv
import json
import pandas as pd

# Buka file Suricata EVE JSON
with open('eve.json', 'r') as file:
    # Baca setiap baris (objek JSON) dalam file
    eve_data_list = [json.loads(line) for line in file]

# Buat list of dictionaries untuk menyimpan data
data_list = []

# Loop melalui setiap objek JSON dalam list
for eve_data in eve_data_list:
    if 'flow_id' in eve_data:
        data_dict = {
            'timestamp': eve_data.get('timestamp', None),
            'flow_id': eve_data['flow_id'],
            'in_iface': eve_data.get('in_iface', None),
            'event_type': eve_data.get('event_type', None),
            'src_ip': eve_data.get('src_ip', None),
            'src_port': eve_data.get('src_port', None),
            'dest_ip': eve_data.get('dest_ip', None),
            'dest_port': eve_data.get('dest_port', None),
            'proto': eve_data.get('proto', None)
        }

        if 'alert' in eve_data:
            alert_data = eve_data['alert']
            data_dict['alert_action'] = alert_data.get('action', None)
            data_dict['alert_gid'] = alert_data.get('gid', None)
            data_dict['alert_signature_id'] = alert_data.get('signature_id', None)
            data_dict['alert_rev'] = alert_data.get('rev', None)
            data_dict['alert_signature'] = alert_data.get('signature', None)
            data_dict['alert_category'] = alert_data.get('category', None)
            data_dict['alert_severity'] = alert_data.get('severity', None)

        if 'flow' in eve_data:
            flow_data = eve_data['flow']
            data_dict['pkts_toserver'] = flow_data.get('pkts_toserver', None)
            data_dict['pkts_toclient'] = flow_data.get('pkts_toclient', None)
            data_dict['bytes_toserver'] = flow_data.get('bytes_toserver', None)
            data_dict['bytes_toclient'] = flow_data.get('bytes_toclient', None)
            data_dict['state'] = flow_data.get('state', None)
            data_dict['age'] = flow_data.get('age', None)
```

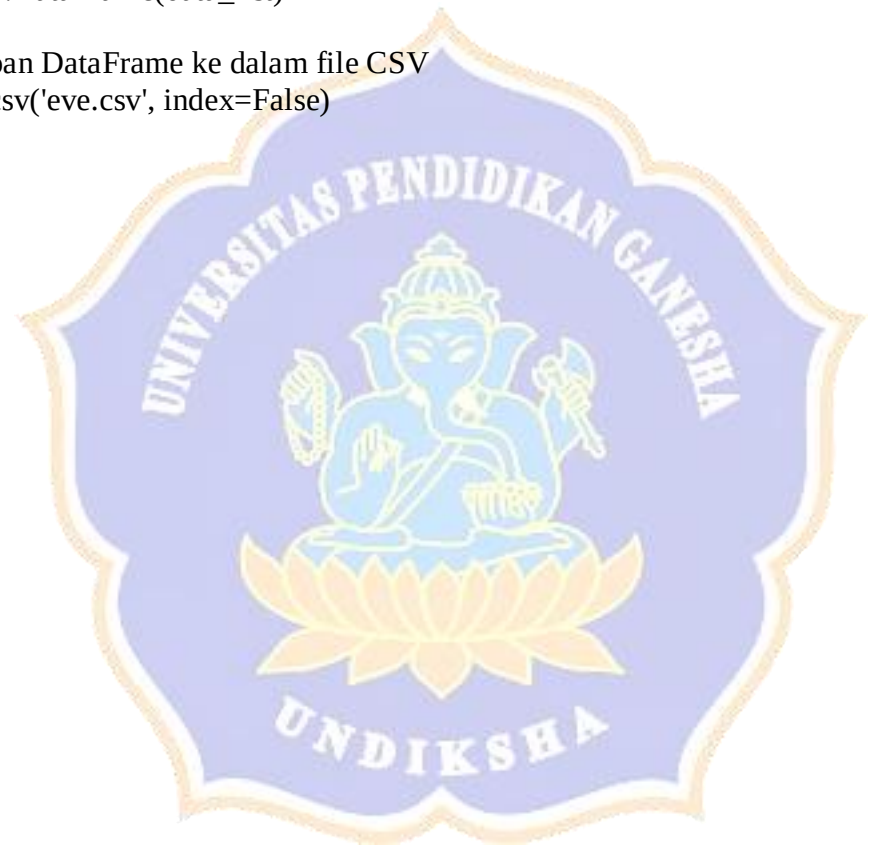
```
            data_dict['alerted'] = flow_data.get('alerted', None)

        if 'anomaly' in eve_data:
            anomaly_data = eve_data['anomaly']
            data_dict['alert_category'] = anomaly_data.get('app_proto', None)
            data_dict['alert_signature'] = anomaly_data.get('event', None)
            data_dict['alert_action'] = anomaly_data.get('type', None)

        data_list.append(data_dict)

# Buat DataFrame dari list of dictionaries
df = pd.DataFrame(data_list)

# Simpan DataFrame ke dalam file CSV
df.to_csv('eve.csv', index=False)
```



Lampiran 3. Sintaks Kode Grid Search Cross Validation

```
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.svm import OneClassSVM
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Baca dataset
df = pd.read_csv('z-score.csv')

# Tentukan kolom target (label)
label_column = 'label'  # Ganti 'label' dengan nama kolom target sesuai dataset Anda

# Ubah label menjadi 1 untuk data normal dan -1 untuk data anomali
df[label_column] = np.where(df[label_column] == 0, 1, -1)

# Pilih fitur yang diinginkan
selected_features = ['alert_gid', 'alert_signature_id', 'alert_rev', 'alert_severity', 'hour', 'pkts_toserver', 'pkts_toclient', 'bytes_toserver', 'bytes_toclient', 'state', 'alerted']

# Pisahkan fitur (X) dan target (y) menggunakan fitur yang dipilih
X = df[selected_features]
y = df[label_column]

# Tangani nilai yang hilang
X = X.dropna()

# Bagi dataset menjadi 80% training dan 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Ambil sampel acak dari data training (misalnya, 20% dari data training)
sample_frac = 0.2
X_train_sampled = X_train.sample(frac=sample_frac, random_state=42)
y_train_sampled = y_train[X_train_sampled.index]

# Tentukan model One-Class SVM
model = OneClassSVM()

# Tentukan hyperparameter yang akan diuji
param_grid = {
    'kernel': ['rbf'],
    'nu': [0.01, 0.1, 0.2],
    'gamma': [0.001, 0.01, 0.1]
```

```
}

# Inisiasi GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# Lakukan Grid Search Cross Validation pada sampel data training
grid_search.fit(X_train_sampled, y_train_sampled)

# Tampilkan parameter terbaik
print("Parameter terbaik:", grid_search.best_params_)

# Tampilkan skor terbaik
print("Skor terbaik:", grid_search.best_score_)

# Mendapatkan hasil Grid Search
results = pd.DataFrame(grid_search.cv_results_)

# Menampilkan kolom-kolom yang relevan
relevant_columns = ['params', 'mean_test_score', 'std_test_score',
'rank_test_score']
print(results[relevant_columns].sort_values(by='rank_test_score'))
```

Lampiran 4. Sintaks Kode Pelatihan dan Pengujian Model

```
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.svm import OneClassSVM
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report

# Baca dataset
df = pd.read_csv('z-score.csv')

# Tentukan kolom target (label)
label_column = 'label'  # Ganti 'label' dengan nama kolom target sesuai dataset Anda

# Pilih fitur yang diinginkan
selected_features = ['alert_gid', 'alert_signature_id', 'alert_rev', 'alert_severity', 'hour', 'pkts_toserver', 'pkts_toclient', 'bytes_toserver', 'bytes_toclient', 'state', 'alerted']

# Pisahkan fitur (X) dan target (y) menggunakan fitur yang dipilih
X = df[selected_features]
y = df[label_column]

# Tangani nilai yang hilang
X_selected = X.dropna()

# Bagi dataset menjadi 80% training dan 20% testing dengan label 0
X_train, X_test, y_train, y_test = train_test_split(X_selected[y == 0], y[y == 0], test_size=0.2, random_state=42)

# Gabungkan data testing label 0 dengan data testing label 1
X_test_label_1 = X[selected_features][y == 1]
y_test_label_1 = y[y == 1]

X_test_final = pd.concat([X_test, X_test_label_1], axis=0)
y_test_final = pd.concat([y_test, y_test_label_1], axis=0)

# Gunakan parameter terbaik yang ditemukan dari GridSearchCV
best_kernel = 'rbf'
best_nu = 0.01
best_gamma = 0.001

# Inisiasi model One-Class SVM dengan parameter terbaik
best_model = OneClassSVM(gamma=best_gamma, kernel=best_kernel, nu=best_nu)
```

```
# Fitting model pada seluruh data training
best_model.fit(X_train)

# Prediksi pada data testing
y_pred = best_model.predict(X_test_final)

# Ubah nilai anomali agar konsisten dengan nilai sebenarnya
prediksi = [1 if i == -1 else 0 for i in y_pred]

# Masukkan hasil prediksi ke dalam DataFrame menggunakan best_model
df['prediction'] = best_model.predict(X_selected)

# Tampilkan DataFrame dengan hasil prediksi
print("DataFrame dengan Hasil Prediksi:")
print(df)

# Tampilkan classification report
print("Classification Report:")
print(classification_report(y_test_final, prediksi))

# Simpan DataFrame yang sudah diberi label ke dalam file CSV
df.to_csv('hasil_prediksi.csv', index=False)
```

Lampiran 5. Sintaks Kode Evaluasi Model

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Hitung Confusion Matrix
conf_matrix = confusion_matrix(y_test, prediksi)

# Visualisasi Confusion Matrix menggunakan seaborn
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'Anomali'], yticklabels=['Normal', 'Anomali'])
plt.title('Confusion Matrix')
plt.xlabel('Prediksi')
plt.ylabel('Aktual')
plt.show()

# Tampilkan classification report
print("Classification Report:")
print(classification_report(y_test, prediksi))
```

Lampiran 6. Sintaks Kode Program Live Analysis

```
import streamlit as st
import time
import threading
import numpy as np
import json
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import psutil
import joblib
from sklearn.preprocessing import LabelEncoder

# Fungsi untuk mendapatkan penggunaan memori dan CPU
def get_system_usage():
    cpu_percent = psutil.cpu_percent()
    memory_info = psutil.virtual_memory()
    return cpu_percent, memory_info.percent

# Function to set up session state
def setup_session_state():
    if 'is_authenticated' not in st.session_state:
        st.session_state.is_authenticated = False

# Page configuration
st.set_page_config(page_title="SIKAMI Polkeska", page_icon="□",
layout="wide")

# Set up session state
setup_session_state()

# Page layout
st.write("# Welcome to SIKAMI Polkeska! □")
st.sidebar.header("□ SIKAMI Polkeska")

# Check if authenticated
if not st.session_state.is_authenticated:
    st.sidebar.success("□□ Silahkan Login!")

    st.markdown(
        """
        SIKAMI (Sistem Keamanan Informasi) Polkeska merupakan sistem live
analysis keamanan jaringan komputer menggunakan algoritma Machine Learning
One Class Support Vector Machines.
        ##### **Masukkan Username dan Password** untuk menjalankan sistem
live analysis!
```

```
        """
    )

    # Form for login
    username = st.text_input("Username")
    password = st.text_input("Password", type="password")

    # Login button
    login_button = st.button("Login")
    if login_button:
        # Check login credentials (Replace with your authentication logic)
        if username == "admin" and password == "kartinibali":
            setup_session_state()  # Initialize session state
            st.session_state.is_authenticated = True
            st.success("Login Successful!")
            st.sidebar.info(f"Logged in as {username}")

            # Clear the login form
            username = password = ""  # Clear the input fields

            # Rerun the app to hide the login form
            st.rerun()

# Tambahkan widget untuk menampilkan penggunaan memori dan CPU
cpu_usage, memory_usage = get_system_usage()
st.sidebar.write(f"CPU Usage: {cpu_usage}%")
st.sidebar.write(f"Memory Usage: {memory_usage}%")

# After login, show different pages
if st.session_state.is_authenticated:
    # Display a message indicating that the user is logged in
    st.sidebar.success("You are logged in!")

    selected_page = st.sidebar.selectbox("Navigation SIKAMI", ["Home",
"Anomali Detection"])

    # Add logout button
    logout_button = st.sidebar.button("Logout", use_container_width=True)
    if logout_button:
        st.session_state.is_authenticated = False
        st.sidebar.success("Logout Successful!")
        st.sidebar.info("□□ Silahkan Login! Klik Logout")

    if selected_page == "Home":
        # Buka file Suricata EVE JSON
        with open('eve.json', 'r') as file:
```

```
    # Baca setiap baris (objek JSON) dalam file
    eve_data_list = [json.loads(line) for line in file]

# Buat list of dictionaries untuk menyimpan data
data_list = []

# Loop melalui setiap objek JSON dalam list
for eve_data in eve_data_list:
    if 'flow_id' in eve_data:
        data_dict = {
            'timestamp': eve_data.get('timestamp', None),
            'flow_id': eve_data['flow_id'],
            'in_iface': eve_data.get('in_iface', None),
            'event_type': eve_data.get('event_type', None),
            'src_ip': eve_data.get('src_ip', None),
            'src_port': eve_data.get('src_port', None),
            'dest_ip': eve_data.get('dest_ip', None),
            'dest_port': eve_data.get('dest_port', None),
            'proto': eve_data.get('proto', None)
        }

        if 'alert' in eve_data:
            alert_data = eve_data['alert']
            data_dict['alert_action'] = alert_data.get('action', None)
            data_dict['alert_gid'] = alert_data.get('gid', None)
            data_dict['alert_signature_id'] = alert_data.get('signature_id', None)
            data_dict['alert_rev'] = alert_data.get('rev', None)
            data_dict['alert_signature'] = alert_data.get('signature', None)
            data_dict['alert_category'] = alert_data.get('category', None)
            data_dict['alert_severity'] = alert_data.get('severity', None)

        if 'flow' in eve_data:
            flow_data = eve_data['flow']
            data_dict['pkts_toserver'] = flow_data.get('pkts_toserver', None)
            data_dict['pkts_toclient'] = flow_data.get('pkts_toclient', None)
            data_dict['bytes_toserver'] = flow_data.get('bytes_toserver', None)
            data_dict['bytes_toclient'] = flow_data.get('bytes_toclient', None)
            data_dict['state'] = flow_data.get('state', None)
            data_dict['alerted'] = flow_data.get('alerted', None)

        if 'anomaly' in eve_data:
            anomaly_data = eve_data['anomaly']
            data_dict['alert_category'] = anomaly_data.get('app_proto', None)
            data_dict['alert_signature'] = anomaly_data.get('event', None)
            data_dict['alert_action'] = anomaly_data.get('type', None)
```

```
            data_list.append(data_dict)

        # Buat DataFrame dari list of dictionaries
        df = pd.DataFrame(data_list)

        # Tampilkan DataFrame di Streamlit dengan lebar penuh
        st.title('Sistem Keamanan Informasi (SIKAMI Polkeska)')
        st.dataframe(df)

        # Set up layout dengan 3 kolom
        col1, col2, col3 = st.columns(3)

        # Hitung total in_iface, src_ip, dan src_port di kolom kiri
        total_in_iface = df['in_iface'].nunique()
        total_src_ip = df['src_ip'].nunique()
        total_src_port = df['src_port'].nunique()

        # Kolom 1: Total Sensor
        clicked_in_iface = col1.metric(label="Total Sensor", value=total_in_iface)
        if col1.button("Tampilkan DataFrame Sensor"):
            with col1.empty():
                col1.write(df.groupby('in_iface').size().reset_index(name='Jumlah'))

        # Kolom 2: Total src IP Address
        clicked_src_ip = col2.metric(label="Total src IP Address",
value=total_src_ip)
        if col2.button("Tampilkan DataFrame src IP Address"):
            with col2.empty():
                col2.write(df.groupby('src_ip').size().reset_index(name='Jumlah'))

        # Kolom 3: Total src Port
        clicked_src_port = col3.metric(label="Total src Port", value=total_src_port)
        if col3.button("Tampilkan DataFrame src Port"):
            with col3.empty():
                col3.write(df.groupby('src_port').size().reset_index(name='Jumlah'))

        # Hitung persentase untuk kolom proto
        proto_percent = df['proto'].value_counts(normalize=True) * 100

        # Tampilkan grafik untuk kolom proto di kolom kanan
        st.title('Persentase Protokol (proto)')
        st.bar_chart(proto_percent)

    elif selected_page == "Anomali Detection":
        st.write("# Deteksi Anomali Jaringan Komputer Polkeska BALI")
        st.sidebar.info("Anomaly Detection Page.")
```

```
# Baca dataset CSV (jika sudah ada) atau buat DataFrame kosong
csv_file_path = r'C:\Program Files\Suricata\log\eve.csv'
try:
    df_existing = pd.read_csv(csv_file_path)
except FileNotFoundError:
    df_existing = pd.DataFrame()

with open('eve.json', 'r') as file:
    # Baca setiap baris (objek JSON) dalam file
    eve_data_list = [json.loads(line) for line in file]

# Buat list of dictionaries untuk menyimpan data
data_list = []

# Loop melalui setiap objek JSON dalam list
for eve_data in eve_data_list:
    if 'flow_id' in eve_data:
        # Periksa apakah 'flow_id' sudah ada dalam DataFrame yang sudah ada
        if eve_data['flow_id'] not in df_existing['flow_id'].values:
            data_dict = {
                'timestamp': eve_data.get('timestamp', None),
                'flow_id': eve_data['flow_id'],
                'in_iface': eve_data.get('in_iface', None),
                'event_type': eve_data.get('event_type', None),
                'src_ip': eve_data.get('src_ip', None),
                'src_port': eve_data.get('src_port', None),
                'dest_ip': eve_data.get('dest_ip', None),
                'dest_port': eve_data.get('dest_port', None),
                'proto': eve_data.get('proto', None)
            }

            if 'alert' in eve_data:
                alert_data = eve_data['alert']
                data_dict['alert_action'] = alert_data.get('action', None)
                data_dict['alert_gid'] = alert_data.get('gid', None)
                data_dict['alert_signature_id'] = alert_data.get('signature_id', None)
                data_dict['alert_rev'] = alert_data.get('rev', None)
                data_dict['alert_signature'] = alert_data.get('signature', None)
                data_dict['alert_category'] = alert_data.get('category', None)
                data_dict['alert_severity'] = alert_data.get('severity', None)

            if 'flow' in eve_data:
                flow_data = eve_data['flow']
                data_dict['pkts_toserver'] = flow_data.get('pkts_toserver', None)
                data_dict['pkts_toclient'] = flow_data.get('pkts_toclient', None)
```

```
                data_dict['bytes_toserver'] = flow_data.get('bytes_toserver', None)
                data_dict['bytes_toclient'] = flow_data.get('bytes_toclient', None)
                data_dict['state'] = flow_data.get('state', None)
                data_dict['alerted'] = flow_data.get('alerted', None)

            if 'anomaly' in eve_data:
                anomaly_data = eve_data['anomaly']
                data_dict['alert_category'] = anomaly_data.get('app_proto', None)
                data_dict['alert_signature'] = anomaly_data.get('event', None)
                data_dict['alert_action'] = anomaly_data.get('type', None)

            data_list.append(data_dict)

    # Buat DataFrame dari list of dictionaries
    df_new_data = pd.DataFrame(data_list)

    # Gabungkan data lama dan baru
    df_combined = pd.concat([df_existing, df_new_data], ignore_index=True)

    # Simpan DataFrame ke dalam file CSV
    df_combined.to_csv(csv_file_path, index=False)

    # Load trained model
    model = joblib.load('ocsvm.joblib')

    # Example: Load new data for prediction
    new_data = pd.read_csv('eve.csv')

    # Ubah tipe data kolom alerted ke numerik (True: 1, False: 0)
    # Gantilah nilai yang kosong pada kolom alerted dengan 1
    new_data['alerted'] = new_data['alerted'].map({True: 1, False: 0})
    new_data['alerted'].fillna(1, inplace=True)

    # Ubah tipe data kolom state ke numerik (established: 2, closed: 1, new: 0)
    # Gantilah nilai yang kosong pada kolom state dengan nilai sesuai kebutuhan
    # Misalnya, gantilah dengan 0 untuk nilai yang kosong
    new_data['state'] = new_data['state'].map({'established': 2, 'closed': 1, 'new':
0})
    new_data['state'].fillna(1, inplace=True)

    # Konversi kolom timestamp
    new_data['timestamp'] = pd.to_datetime(new_data['timestamp'],
errors='coerce')
    new_data['year'] = new_data['timestamp'].dt.year
    new_data['month'] = new_data['timestamp'].dt.month
    new_data['day'] = new_data['timestamp'].dt.day
```

```
        new_data['hour'] = new_data['timestamp'].dt.hour
        new_data['minute'] = new_data['timestamp'].dt.minute
        new_data['second'] = new_data['timestamp'].dt.second
        new_data = new_data.drop(columns=['timestamp'])

        # Konversi kolom-kolom kategorikal menggunakan Label Encoding
        label_encoder = LabelEncoder()
        new_data['in_iface'] = label_encoder.fit_transform(new_data['in_iface'])
        new_data['alert_action'] =
label_encoder.fit_transform(new_data['alert_action'])
        new_data['alert_signature'] =
label_encoder.fit_transform(new_data['alert_signature'])
        new_data['alert_category'] =
label_encoder.fit_transform(new_data['alert_category'])

        # Mengisi data kosong hanya pada kolom-kolom numerik dengan nilai
khusus (misalnya, 0)
        kolom_numerik = new_data.select_dtypes(include='number').columns
        new_data[kolom_numerik] = new_data[kolom_numerik].fillna(0)

        kolom_non_numerik = new_data.select_dtypes(exclude='number').columns
        for kolom in kolom_non_numerik:
            nilai_khusus = 'unknown'  # Ganti dengan nilai khusus yang sesuai
            new_data[kolom] = new_data[kolom].fillna(nilai_khusus)

        # Hapus baris dengan event_type selain 'alert', 'flow', dan 'anomaly'
        new_data = new_data[(new_data['event_type'] == 'alert') |
(new_data['event_type'] == 'flow')]

        # Pilih kolom-kolom yang diperlukan untuk prediksi
        columns_for_prediction = ['alert_gid', 'alert_signature_id', 'alert_rev',
'alert_severity', 'hour', 'pkts_toserver', 'pkts_toclient', 'bytes_toserver',
'bytes_toclient', 'state', 'alerted']

        # Pilih hanya kolom yang diperlukan untuk prediksi
        new_data_for_prediction = new_data[columns_for_prediction]

        # Example: Make predictions using the trained model
        predictions = model.predict(new_data_for_prediction)

        # Tambahkan kolom prediksi ke DataFrame
        new_data['prediction'] = predictions

        # Tampilkan DataFrame dan hasil prediksi
        st.write("Informasi Hasil Prediksi Data dengan One-Class Support Vector
Machine :")
```

```
        st.write(new_data)

        # Menampilkan jumlah total prediksi -1 dan 1
        total_pred_minus_1 = new_data[new_data['prediction'] == -1].shape[0]
        total_pred_1 = new_data[new_data['prediction'] == 1].shape[0]

        # Menampilkan jumlah total proto
        total_proto = new_data['proto'].nunique()

        # Layout dua kolom
        col1, col2 = st.columns(2)

        # Menampilkan jumlah dengan button di kolom pertama
        if col1.button("Tampilkan Chart Jumlah Prediksi"):
            # Tampilkan total prediksi
            col1.write(f"Total Predictions with -1: {total_pred_minus_1}")
            col1.write(f"Total Predictions with 1: {total_pred_1}")

            # Plot chart
            fig, ax = plt.subplots()
            labels = ['Anomaly', 'Normal']
            sizes = [total_pred_minus_1, total_pred_1]
            colors = ['#ff9999', '#66b3ff']
            ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90,
colors=colors)
            ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
            col1.pyplot(fig)

        # Tombol untuk menampilkan chart proto di kolom kedua
        if col2.button("Tampilkan Chart Jumlah Protokol"):
            # Hitung frekuensi setiap proto
            proto_counts = new_data['proto'].value_counts()

            # Plot chart
            fig, ax = plt.subplots()
            proto_counts.plot(kind='bar', ax=ax)
            ax.set_xlabel('Proto')
            ax.set_ylabel('Count')
            ax.set_title('Proto Distribution')
            col2.pyplot(fig)

        # Pilih kolom-kolom yang diperlukan untuk prediksi
        columns_for_prediction = ['flow_id', 'proto', 'src_ip', 'src_port', 'dest_ip',
'dest_port', 'pkts_toserver', 'pkts_toclient',
                                  'bytes_toserver', 'bytes_toclient', 'state', 'alerted', 'alert_gid',
'alert_signature_id', 'alert_rev', 'alert_severity',
```

```
                    'hour', 'minute', ]

        # Pilih hanya kolom yang diperlukan untuk prediksi
        new_data_for_prediction = new_data[['prediction'] +
columns_for_prediction]

        # Bagi tampilan menjadi dua kolom
        col1, col2 = st.columns(2)

        # Menampilkan DataFrame dengan prediksi -1 pada kolom pertama
        with col1:
            st.write("Informasi Data Anomali :")
            df_pred_minus_1 =
new_data_for_prediction[new_data_for_prediction['prediction'] == -1]
            st.write(df_pred_minus_1)

        # Menampilkan DataFrame dengan prediksi 1 pada kolom kedua
        with col2:
            st.write("Informasi Data Normal :")
            df_pred_1 =
new_data_for_prediction[new_data_for_prediction['prediction'] == 1]
            st.write(df_pred_1)

        # Streamlit widgets automatically run the script from top to bottom. Since
        # this button is not connected to any other logic, it just causes a plain
        # rerun.
        st.button("Re-run")
```