Sistema de Detección de Ciberataques DDoS por Análisis de Tráfico de Red

Master en IA sobre el sector de la Energía y las infraestructuras

Autor: Ramiro Bueno Martínez

Fecha: 05/05/2024

Version: R200

Descripcion: PARTE I: Ingeniería de Datos

Dataset

Conjunto de Datos relativo a la documentación de un Ataque de Denegación Distribuida de Servicio (DDoS), realizado a partir del conjunto de datos publicado por el Instituto de Ciber Seguridad Canadiense (CIC) el año 2019.

Licencia:

- Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak y Ali A. Ghorbani, " Developing Realistic Distributed Denial of Service of Service (DDoS) Attack Dataset and Taxonomy ", IEEE 53rd International Carnahan Conference on Security Technology, Chennai, India, 2019. DOI: 10.1109/CCST.2019.8888419

Resumen

El ataque de denegación de servicio (DDoS) es una amenaza a la seguridad de la red que tiene como objetivo agotar las redes de destino con tráfico malicioso. Aunque se han diseñado muchos métodos estadísticos para la detección de ataques DDoS, diseñar un detector en tiempo real con techos computacionales bajos sigue siendo una de las principales preocupaciones. Por otro lado, la evaluación de nuevos algoritmos y técnicas de detección se basa en gran medida en la existencia de conjuntos de datos bien diseñados.

Referencias estandares normativos

ISO/IEC FDIS 23053 Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML) Note: Under development ISO/IEC FDIS 23053

ISO/IEC CD 8183 Information technology — Artificial intelligence — Data life cycle framework Note: Under development ISO/IEC CD 8183

ISO/IEC DIS 24668 Information technology — Artificial intelligence — Process management framework for big data analytics Note: Under development ISO/IEC DIS 24668

ISO/IEC CD 5338 Information technology — Artificial intelligence — AI system life cycle processes Note: Under developmentISO/IEC CD 5338

Objetivo Principal:

El objetivo principal será determinar en tiempo real a través de monitorizacón en tiempo real de tramas de datos en tráfico de red, con programas de monitorización de red, como puede ser tshark, si estas tramas pueden estar sufriendo un ataque DDoS (Label=DDoS), o por el contrario el tráfico de red y las tramas generadas no tiene anomalía alguna (Label=BENIGN)

Objetivos Epecificos:

Definir y desarrollar las distintas Fases de un Proyecto Data Science para la Detección de anomalías en el tráfico de red

Medir las Prestaciones de distintos algoritmos, y determinar el mejor para este tipo de proyectos: (Parte II y Parte III

```
<ul>
    <li>Logistic Regression, (Parte II)</li>
    <li>Naïve Bayes,</li>
    <li>K-nearest neighbours,</li>
    <li>Multi Layer Perceptron,</li>
    <li>Ada Boost,</li>
    <li>Quadrant Discriminant Analysis,</li>
    <li>Random Forests,</li>
    <li>Pipelining and Ensemble,</li>
    <li>Deep Learning y Transfer Learning (Parte III)</li>
</ul>
```

Proponer y desarrollar un modelo predictivo a partir del algoritmo con una mayor precisión

Establecer Gráficas y Clasificaciones que permitan reforzar la verificación de la Hipotesis de inicio

Proponer una arquitectura descentralizada basada en Sistemas Autonomos y Multi-Agente con capacidades para una detección descentralizada de posibles amenazas (Parte IV)

Puesta en Servicio y Encapsulación en contenedores Dockers (Parte V)

Definición de un sistema machine learning (sistema de aprendizaje automático)

ISO/IEC 22989 define el aprendizaje automático como el proceso de optimización de los parámetros del modelo a través de técnicas computacionales, de modo que el comportamiento del modelo refleje los datos o la experiencia. Desde temprano Los modelos de neuronas de la década de 1940 y el desarrollo de programas informáticos que pueden aprender mientras se ejecutan tienen sido explorado. El aprendizaje automático es un campo de investigación en curso con nuevas aplicaciones emergentes en una amplia gama de sectores industriales. El acceso a grandes cantidades de datos y recursos computacionales es un importante habilitador para la realización y progresión del aprendizaje automático

Pero un sistema ML como tal, es un sistema complejo que conviene entender como un conjunto de partes que permiten realizar predicciones a partir de los datos que nos ofrece la experiencia. En la siguiente imagen es posible ver en su contexto la estructura de un proyecto ML

Fases del Proyecto:

Diseño y Desarrollo

La propuesta de diseño del proyecto de ciencia de datos,se basará en el pipeline de aprendizaje automático propuesto por el documento normativo ISO/IEC 23053

Fase I.- Adquisición de datos

Recolección de datos

A) Almacen de Datos CICDDoS2029 (Cloud/Kaggle)
B) PoC: ASDDoS2024 (Workstation)

Fase II.- Preparación de Datos

Preparación de Datos

C) Preprocesado
D) Limpieza
E) Exploracion de datos
F) Imputacion
G) Nomalización, escalado y distribución
H) Conformación de Conjuntos de Datos
I) Troceado del Conjunto de Datos
J) Aprendizaje Supervisado: Labeling
K) Otras transformaciones

L) Matriz de Confusion

Fase III.- Modelado

Selección de Atributos / Características

M) Matrices de Correlación
N) Detección de Anomalias / Estudio de Outliers
O) Tecnicas de Reducción de Dimensión, PCA y Clustering
P) Clasificadores

Entrenamiento del Model
Q) Regresion Logistica R) Exploración técnicas Sobremuestreo con SMOTE

Selección del Modelo

Fase IV.- Análisis de Resultados

Comprobación y Validación S) Testing with XGBoosting T) Neural Networks Testing (Undersampling vs Oversampling)

Introducción: Etapas del Estudio de Ciencia de Datos

1. Recolección de Datos Los datos se recopilan de diversas fuentes, como medidores individuales, APIs públicas de clima y demografía, o datos históricos de consumo y producción. Se colocan en un conjunto de datos en las instalaciones antes de ser manipulados. Todos los datos recopilados de los medidores inteligentes individuales son anónimos después del concentrador y no pueden identificar al consumidor.
2. Limpieza de Datos Los datos se limpian. Se recuperan los datos útiles. Se eliminan los datos irrelevantes o incorrectos.
3. Pre-procesamiento de Datos Se llenan los valores NA de los datos limpios y todas las variables categóricas se codifican en valores numéricos. Se definen las características de la capa de entrada de los algoritmos. Los datos que podrían identificar al consumidor se eliminan a nivel del concentrador.
4. Diseño e Implementación del Modelo Los diseños y parámetros del modelo se determinan según nuestras necesidades, pruebas estadísticas y análisis de series temporales.
5. Entrenamiento del Modelo Los datos se preparan para habilitar predicciones por hora. El entrenamiento inicial se realiza utilizando el 70% de los datos históricos y se procesa en centros de datos regionales junto a los centros de supervisión. El 15% de los datos se utiliza para la evaluación del modelo.
6. Pruebas del Modelo El 15% de los datos más recientes se utiliza para las pruebas.
7. Optimización Los hiperparámetros del modelo se optimizan mediante diferentes técnicas.

8. Evaluación del Modelo Desde una perspectiva de Aprendizaje Automático, el modelo se evalúa en datos dedicados con métricas como el Error Absoluto Medio o el Error Cuadrático Medio. Si el rendimiento del modelo no cumple con los resultados requeridos, se deben repetir los pasos anteriores.
9. Despliegue del Modelo El modelo se pone en producción en servidores dedicados e incorporado en una herramienta de panelización. Estos servidores están alojados en las instalaciones.
10. Monitoreo e Inferencia Una vez en producción, el modelo proporciona predicciones a los proveedores de electricidad. Estas predicciones pueden mostrarse a través de herramientas de panelización o incorporarse directamente en herramientas existentes para su reutilización.

FASE I.- Adquisicion de Datos

Los datos se recopilan de diversas fuentes, como medidores individuales, APIs públicas de clima y demografía, o datos históricos de consumo y producción. Se colocan en un conjunto de datos en las instalaciones antes de ser manipulados. Todos los datos recopilados de los medidores inteligentes individuales son anónimos después del concentrador y no pueden identificar al consumidor.

1.1 Conjunto de Datos empleado

CICDDoS2019 es un conjunto de datos que contiene ataques DDoS comunes, benignos y actualizados, que se asemejan a datos del mundo real (PCAPs). Sin embargo, esta versión, en formato de archivo CSV, incluye los resultados del análisis de tráfico de red utilizando CICFlowMeter-V3, con flujos etiquetados según la marca de tiempo, las direcciones IP de origen y destino, los puertos de origen y destino, los protocolos y el tipo de ataque.

1.2 Licencia

Segun la información proporcionada por el Instituto Canadiense de Ciberseguridad es posible redistribuir, volver a publicar y reflejar el conjunto de datos CICDDoS2019 en cualquier forma. https://www.unb.ca/cic/datasets/ddos-2019.html

Sin embargo, cualquier uso o redistribución de los datos debe incluir una cita al conjunto de datos CICDDoS2019 y el artículo publicado relacionado. Un trabajo de investigación en el que se esbozan los detalles de analizar el conjunto de datos IDS/IPS similar y principios relacionados:

- Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak y Ali A. Ghorbani, " Developing Realistic Distributed Denial of Service of Service (DDoS) Attack Dataset and Taxonomy ", IEEE 53rd International Carnahan Conference on Security Technology, Chennai, India, 2019.

```
#!pip install matplotlib seaborn scikit-learn "dask[complete]" tabulate
```

```
import concurrent.futures
import csv
import datetime
import ipaddress
import math
import multiprocessing
import os
import pickle
import psutil
import random
import sys
import time
import tracemalloc
import warnings

import dask.dataframe as dd
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib.gridspec import GridSpec
from tabulate import tabulate

from sklearn.calibration import CalibratedClassifierCV, CalibrationDisplay
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import (
    AdaBoostClassifier,
    GradientBoostingClassifier,
    RandomForestClassifier,
    StackingClassifier,
)
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.manifold import TSNE
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score,
    precision_score,
    recall_score,
)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
```

[A] DataSets: Almacenes de Datos del Conjunto de Datos CICDDoS2019

El Almacen de Datos conformado por el Conjunto de Datos CICDDoS2019 documenta las anomalías producidas en el Trafico de Red, experimentadas por una Red de Computadoras que esta experimentando distintos tipos de Ataques DDoS cuya finalidad es producir una Denegación Distribuida del Servicio de Red en este tipo de Redes.

```
from datetime import datetime

# ================================================================================
# Declaración de listas de características
# Se han eliminado las declaraciones duplicadas.
# ================================================================================

# Lista de características seleccionadas manualmente
selected_features = [
    'Source IP','Destination IP','Source Port','Destination Port','Protocol',
    'Fwd Packets/s','Flow Duration', 'Total Fwd Packets',
    'Total Backward Packets', 'Total Length of Fwd Packets',
    'Total Length of Bwd Packets', 'Fwd Packet Length Max', 'Fwd Packet Length Min',
    'Fwd Packet Length Mean', 'Bwd Packet Length Max', 'Bwd Packet Length Min',
    'Bwd Packet Length Mean', 'Flow Bytes/s', 'Flow Packets/s', 'FIN Flag Count',
    'SYN Flag Count', 'RST Flag Count', 'PSH Flag Count','ACK Flag Count',
    'URG Flag Count', 'CWE Flag Count', 'ECE Flag Count', 'Down/Up Ratio',
```

```
    'Average Packet Size', 'Init_Win_bytes_forward',
    'Init_Win_bytes_backward', 'act_data_pkt_fwd', 'min_seg_size_forward',
    'Active Mean', 'Active Std', 'Active Max', 'Active Min',
    'Idle Mean', 'Idle Std', 'Idle Max', 'Idle Min', 'SimillarHTTP',
    'Inbound','Label'
]

# Características de un conjunto extendido
extended_set_of_features = ['Timestamp','Source IP','Destination IP','Source Port',
                            'Destination Port','Protocol']

# Características de un conjunto reducido (Table 4. https://doi.org/10.26555/ijain.v7i1.553)
reduced_set_of_features = [
    'Fwd Packets/s', 'Total Backward Packets', 'Total Length of Fwd Packets',
    'Fwd Packet Length Std','Bwd Packet Length Std', 'Flow IAT Std','Bwd IAT Mean',
    'Bwd IAT Std','Bwd IAT Min','Fwd Header Length','Packet Length Mean',
    'SYN Flag Count','FIN Flag Count','PSH Flag Count','Avg Fwd Segment Size',
    'Fwd Avg Bytes/Bulk', 'Fwd Avg Packets/Bulk','Init_Win_bytes_forward','Init_Win_bytes_ba
    'Active Std','Idle Std'
]

# Características de otro conjunto reducido, con atributos de flujo
reduced_set_of_features_II = [
    'Timestamp','Source IP','Destination IP','Source Port','Destination Port',
    'Protocol', 'Fwd Packets/s','Bwd Packets/s','Flow Bytes/s', 'Flow Packets/s',
    'Total Fwd Packets','Total Backward Packets', 'Total Length of Fwd Packets',
    'Total Length of Bwd Packets', 'Fwd Packet Length Std','Bwd Packet Length Std',
    'Flow Duration', 'Flow IAT Std','Bwd IAT Mean','Bwd IAT Std','Bwd IAT Min',
    'SYN Flag Count','FIN Flag Count','PSH Flag Count', 'Active Std','Idle Std',
    'Fwd Header Length','Bwd Header Length','Fwd Header Length', 'Label'
]

# Atributos de PCA
PCA_25_Attributes = ['Fwd IAT Std', 'Fwd IAT Mean', 'Fwd IAT Total', 'Flow IAT Min',
                     'Flow IAT Max', 'Flow IAT Std', 'Flow IAT Mean', 'Flow Packets/s',
                     'Flow Bytes/s', 'Bwd Packet Length Std', 'Bwd Packet Length Mean',
                     'Bwd Packet Length Min', 'Bwd Packet Length Max',
                     'Fwd Packet Length Std', 'Fwd Packet Length Mean',
                     'Fwd Packet Length Min', 'Fwd Packet Length Max',
                     'Total Length of Bwd Packets', 'Total Length of Fwd Packets',
                     'Total Backward Packets', 'Total Fwd Packets', 'Flow Duration',
                     'Protocol', 'Destination Port', 'Source Port','Label'
```

```
                    ]

# Atributos para entrenamiento (son idénticos a Main_Attributes, por lo que
# se define uno y se asigna al otro para evitar redundancia)
Training_Attributes = ['Timestamp','Source IP','Destination IP','Protocol',
                       'Destination Port', 'Source Port', 'Flow Duration',''
                       'Flow Packets/s', 'Flow Bytes/s', 'Fwd IAT Std',
                       'Fwd IAT Mean', 'Fwd IAT Total', 'Flow IAT Min',
                       'Flow IAT Max', 'Flow IAT Std','Flow IAT Mean',
                       'Bwd Packet Length Std', 'Bwd Packet Length Mean',
                       'Bwd Packet Length Min', 'Bwd Packet Length Max',
                       'SYN Flag Count', 'FIN Flag Count', 'PSH Flag Count',
                       'ACK Flag Count', 'Fwd Packet Length Std',
                       'Fwd Packet Length Mean', 'Fwd Packet Length Min',
                       'Fwd Packet Length Max', 'Total Length of Bwd Packets',
                       'Total Length of Fwd Packets', 'Total Backward Packets',
                       'Total Fwd Packets','Label']

# Atributos principales
Main_Attributes = Training_Attributes

# Define the global variable PCA_8_Attributes
PCA_8_Attributes = [
    'Fwd Packet Length Mean', 'Fwd Packet Length Min', 'Fwd Packet Length Max',
    'Total Length of Bwd Packets', 'Total Length of Fwd Packets',
    'Total Backward Packets', 'Total Fwd Packets', 'Flow Duration'
]

# ================================================================================
# Asignaciones y lógica
# ================================================================================

# Se han consolidado las asignaciones a variables principales.
top_50_features = selected_features
top_25_features = reduced_set_of_features_II

fecha_formateada = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
print(f"FASE I.- [Exploracion Conjunto de Datos] CIC-DDoS2019")
```

```
2025-08-07 16:38:15 - FASE I.- [Exploracion Conjunto de Datos] - Atributos del Conjunto de Da
```

```
output_file = "/kaggle/working/training/Training_Ref_Attack_Mixed.csv"

# Lista de rutas de los archivos CSV originales
file_paths = [
    '/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_NetBIOS.csv',
    #'/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_SSDP.csv',
    #'/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_NTP.csv',
    #'/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/TFTP.csv',
    #'/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/UDPLag.csv',
    #'/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_UDP.csv',
    #'/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/Syn.csv',
    #'/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_MSSQL.csv',
    '/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_SNMP.csv',
    '/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_DNS.csv',
    '/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/01-12/DrDoS_LDAP.csv'
]

file_test_paths = [
    '/kaggle/input/cic-ddos2019-30gb-full-dataset-csv-files/03-11/UDP.csv'
]
```

[B] PoC: Ingeniería de Datos en Conjuntos de Datos de Tráfico de Red

Para validar el diseño en una computadora local, se ha implementado el siguiente conjunto de datos, basado en el conjunto de Datos CICDDoS2019, que denominaremos ASDDoS2024, al tratarse de una versión actualizada y modificada, dirigida a desarrollar modelos predictivos que permita dotar a Agentes Autónomos de la Capacidad de Conciencia Situacional en un Ataque DDoS, implementando percepción (monitorización de tráfico de red), comprensión del tráfico de red, y la implementación de actuadores básicos (IDS) de señalización e identificación del tipo de ataque.

```
import os
#from google.colab import drive

# Montar Google Drive si aún no está montado
#try:
#    drive.mount('/content/drive')
#except:
#    print("Drive ya está montado.")

# Ficheros del Conjunto de datos ASDDoS2024
```

```python
_asddos_dataset_paths = [
    'C:/DataSets/Training/Training_Exp_Attack_TCP.csv',
    'C:/DataSets/Training/Training_Exp_Attack_UDP.csv',
    'C:/DataSets/Training/Training_Ref_Attack_MIXED.csv',
    'C:/DataSets/Training/Training_Ref_Attack_TCP.csv',
    'C:/DataSets/Training/Training_Ref_Attack_UDP.csv'
]

# Verificar la existencia de cada archivo
print("--- [IMPORTANTE] Verificando la existencia de los archivos ---")
all_files_exist = True
for file_path in _asddos_dataset_paths:
    if os.path.exists(file_path):
        print(f"El archivo existe: {file_path}")
    else:
        print(f"El archivo NO existe: {file_path}")
        all_files_exist = False

# Resumen
if all_files_exist:
    print("\n [CORRECTO] EL CONJUNTO DE DATOS ASDDOS-2019 ESTA DISPONIBLE")
else:
    print("\n [ERROR] EL CONJUNTO DE DATOS ASDDOS-2024 NO ESTA DISPONIBLE")

# Verificar el directorio
dirname = 'C:/DataSets/Training/'

print(f"\n--- [IMPORTANTE] Verificando el directorio: {dirname} ---")
if os.path.isdir(dirname):
    print(f"El directorio '{dirname}' existe.")
else:
    print(f"El directorio '{dirname}' NO existe. POR FAVOR, NO CONTINUE.")
```

```
--- [IMPORTANTE] Verificando la existencia de los archivos ---
El archivo NO existe: C:/DataSets/Training/Training_Exp_Attack_TCP.csv
El archivo NO existe: C:/DataSets/Training/Training_Exp_Attack_UDP.csv
El archivo NO existe: C:/DataSets/Training/Training_Ref_Attack_MIXED.csv
El archivo NO existe: C:/DataSets/Training/Training_Ref_Attack_TCP.csv
El archivo NO existe: C:/DataSets/Training/Training_Ref_Attack_UDP.csv

 [ERROR] EL CONJUNTO DE DATOS ASDDOS-2024 NO ESTA DISPONIBLE EN SU TOTALIDAD
```

```
--- [IMPORTANTE] Verificando el directorio: C:/DataSets/Training/ ---
El directorio 'C:/DataSets/Training/' existe.
```

```python
from datetime import datetime

file_production_paths = [
    'C:/DataSets/Produccion/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv'
]

output_file_reflection_tcp = 'C:/DataSets/Training/Training_Exp_Attack_TCP.csv'
output_file_reflection_udp ='C:/DataSets/Training/Training_Exp_Attack_UDP.csv'
output_file_reflection_mixed ='C:/DataSets/Training/Training_Ref_Attack_MIXED.csv'
output_file_exploitation_tcp = 'C:/DataSets/Training/Training_Ref_Attack_TCP.csv'
output_file_exploitation_udp = 'C:/DataSets/Training/Training_Ref_Attack_UDP.csv'

fecha_formateada = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
print(f"FASE I.- [Exploracion Conjunto de Datos] - Conjuntos de Datos ASDDoS2024")
```

```
2025-08-07 16:38:15 - FASE I.- [Exploracion Conjunto de Datos] - Conjuntos de Datos por Explo
```

```python
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from tabulate import tabulate
from datetime import datetime, timedelta
from matplotlib.gridspec import GridSpec
import matplotlib.patches as mpatches

# Ignorar las advertencias de pandas para mantener la salida limpia.
warnings.filterwarnings("ignore")

# --- Funciones de Cálculo de Métricas de Calidad del Modelo de Datos ---

def f_metricas_calidad_modelo_datos(df: pd.DataFrame) -> pd.DataFrame:
    """
    Calcula un conjunto de métricas de calidad para un DataFrame de datos de red.

    Esta función actúa como un orquestador, llamando a cada métrica individual y
    compilando los resultados en un DataFrame para una fácil visualización.
```

```
    Args:
        df (pd.DataFrame): El DataFrame de entrada que contiene los datos a evaluar.

    Returns:
        pd.DataFrame: Un DataFrame con las métricas de calidad y sus valores.
    """
    print("\n[Benchmarking] Calidad del Modelo de Datos")

    # Se pasa el DataFrame directamente a cada métrica para evitar variables globales.
    metricas = {
        'Exactitud': exactitud_metric(df),
        'Completitud': completitud_metric(df),
        'Consistencia': consistencia_metric(df),
        'Credibilidad': credibilidad_metric(df),
        'Actualidad': actualidad_metric(df),
        'Balanceado': balanceado_metric(df)
    }

    # Convertir las métricas a un DataFrame para una presentación clara.
    df_metricas = pd.DataFrame.from_dict(metricas, orient='index', columns=['Valor'])

    return df_metricas


def exactitud_metric(df: pd.DataFrame) -> float:
    """
    Calcula la exactitud como el porcentaje de filas sin valores nulos en el DataFrame.
    Un valor del 100% indica que todas las filas están completas.

    Args:
        df (pd.DataFrame): DataFrame a evaluar.

    Returns:
        float: El porcentaje de filas completas.
    """
    print("[Benchmarking] Exactitud del modelo")
    num_exactos = df.dropna().shape[0]
    num_total = len(df)
    print(f"Exactitud: Exactos {num_exactos} Totales {num_total}")
    return (num_exactos / num_total) * 100 if num_total > 0 else 0
```

```python
def completitud_metric(df: pd.DataFrame) -> float:
    """
    Calcula la completitud como el porcentaje de filas sin valores nulos.
    Esta métrica es funcionalmente similar a la exactitud en este contexto,
    evaluando si todos los campos de una fila están presentes.

    Args:
        df (pd.DataFrame): DataFrame a evaluar.

    Returns:
        float: El porcentaje de filas completas.
    """
    print("[Benchmarking] Completitud del modelo")
    num_completos = df.dropna().shape[0]
    num_total = len(df)
    print(f"Completitud: Completos {num_completos} Totales {num_total}")
    return (num_completos / num_total) * 100 if num_total > 0 else 0


def consistencia_metric(df: pd.DataFrame) -> float:
    """
    Calcula la consistencia asegurando que la IP de origen y la de destino
    no sean idénticas en la misma fila, lo que indicaría un bucle de tráfico
    inconsistente en los datos.

    Args:
        df (pd.DataFrame): DataFrame a evaluar ('Source IP' y 'Destination IP').

    Returns:
        float: El porcentaje de filas con IPs de origen y destino diferentes.
    """
    print("[Benchmarking] Consistencia del modelo")
    # Se filtran las filas donde la IP de origen es diferente de la de destino.
    num_consistentes = (df['Source IP'] != df['Destination IP']).sum()
    num_total = len(df)
    print(f"Consistencia: Consistentes {num_consistentes} Totales {num_total}")
    return (num_consistentes / num_total) * 100 if num_total > 0 else 0


def credibilidad_metric(df: pd.DataFrame) -> float:
    """
    Calcula la credibilidad verificando el formato de las IPs de origen y destino
```

```
    y la validez del campo 'Timestamp'.

    Args:
        df (pd.DataFrame): DataFrame a evaluar.

    Returns:
        float: Porcentaje de filas que cumplen con los criterios de credibilidad.
    """
    print("[Benchmarking] Credibilidad del modelo")

    # Convertir 'Timestamp' a formato de fecha y hora, los errores se convierten en NaT.
    df_temp = df.copy()
    df_temp['Timestamp'] = pd.to_datetime(df_temp['Timestamp'], errors='coerce')

    # Evaluar si las IPs tienen un formato básico de 4 octetos separados por puntos
    # y si el 'Timestamp' es un valor de fecha y hora válido.
    is_valid_ip_src = df_temp['Source IP'].str.count('.') == 3
    is_valid_ip_dst = df_temp['Destination IP'].str.count('.') == 3
    is_valid_timestamp = df_temp['Timestamp'].notna()

    num_veraces = (is_valid_ip_src & is_valid_ip_dst & is_valid_timestamp).sum()
    num_total = len(df)

    print(f"Credibilidad: Veraces {num_veraces} Totales {num_total}")
    return (num_veraces / num_total) * 100 if num_total > 0 else 0


def actualidad_metric(df: pd.DataFrame, current_year: int = 2024, data_year: int = 2019) -> i
    """
    Calcula una métrica de actualidad basada en una diferencia de años.
    Esta es una métrica heurística que penaliza los datos más antiguos.

    Args:
        df (pd.DataFrame): DataFrame a evaluar.
        current_year (int): El año actual para el cálculo.
        data_year (int): El año de referencia de los datos.

    Returns:
        float: Un valor de actualidad ajustado.
    """
    print("[Benchmarking] Actualidad del modelo")
    num_total = len(df)
```

```
    # Calcular el factor de penalización por antigüedad.
    # Por ejemplo, si la diferencia es de 5 años, el factor es (1 - 5/100) = 0.95.
    factor = 1 - (current_year - data_year) / 100

    # Se considera que todas las filas contribuyen a la actualidad
    # de manera ponderada por el factor de antigüedad.
    actualidad = (100 * factor) if num_total > 0 else 0

    print(f"Actualidad: Año de los datos {data_year}, Año actual {current_year}")
    return actualidad


def balanceado_metric(df: pd.DataFrame) -> float:
    """
    Calcula la métrica de balance de clases. Retorna el porcentaje de la
    clase más frecuente ('BENIGN' en este caso). Un valor más cercano a
    un balance ideal (por ejemplo, 50% en un problema binario) es mejor.

    Args:
        df (pd.DataFrame): DataFrame a evaluar (debe contener la columna 'Label').

    Returns:
        float: Porcentaje de la clase 'BENIGN' en el dataset.
    """
    print("[Benchmarking] Balanceado del modelo")

    # Imprimir el porcentaje de cada clase para un análisis detallado.
    print("Distribución de etiquetas:")
    total_samples = len(df)
    if total_samples > 0:
        for classificador, count in df['Label'].value_counts().items():
            print(f"  - Etiqueta '{classificador}': {round(count / total_samples * 100, 2)}%

        # Se retorna el porcentaje de la clase 'BENIGN' como métrica de balance.
        benign_percentage = df['Label'].value_counts(normalize=True).get('BENIGN', 0) * 100
    else:
        benign_percentage = 0

    return benign_percentage


# --- Funciones de Visualización ---
```

```python
def radar_plot_with_table(metricas_df1: pd.DataFrame, metricas_df2: pd.DataFrame, df1_name: s
    """
    Genera un gráfico de radar y una tabla comparativa para dos conjuntos de métricas.

    Args:
        metricas_df1 (pd.DataFrame): DataFrame con las métricas del primer conjunto de datos
        metricas_df2 (pd.DataFrame): DataFrame con las métricas del segundo conjunto de datos
        df1_name (str): Nombre del primer conjunto de datos.
        df2_name (str): Nombre del segundo conjunto de datos.
    """
    print("Graficando diagramas de radar ....")
    labels = metricas_df1.index
    num_vars = len(labels)

    # Convertir las métricas en listas para el gráfico.
    values1 = [float(x) for x in metricas_df1['Valor'].tolist()]
    values2 = [float(x) for x in metricas_df2['Valor'].tolist()]

    # Añadir el primer valor al final para cerrar el gráfico.
    values1 += values1[:1]
    values2 += values2[:1]

    # Calcular los ángulos de cada eje para el gráfico de radar.
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
    angles += angles[:1]

    # Crear los subplots para el gráfico de radar y la tabla.
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 8))

    # --- Gráfico de Radar ---
    ax1 = plt.subplot(121, polar=True)
    ax1.fill(angles, values1, color='red', alpha=0.25, label=df1_name)
    ax1.fill(angles, values2, color='blue', alpha=0.25, label=df2_name)
    ax1.plot(angles, values1, color='red', linewidth=2)
    ax1.plot(angles, values2, color='blue', linewidth=2)

    ax1.set_title('Métricas de Calidad del Modelo de Datos', size=16, color='black', y=1.1)
    ax1.set_yticklabels([])
    ax1.set_xticks(angles[:-1])
    ax1.set_xticklabels(labels, fontsize=12)
    ax1.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))
```

```
    # --- Tabla Comparativa ---
    table_data = {
        'KPI': labels,
        df1_name: [f'{v:.1f}%' for v in values1[:-1]],
        df2_name: [f'{v:.1f}%' for v in values2[:-1]]
    }
    df_table = pd.DataFrame(table_data)

    ax2.set_title(f'Comparativa de la Calidad del Modelo de Datos', size=16)
    ax2.axis('off')
    table = ax2.table(cellText=df_table.values, colLabels=df_table.columns,
                      cellLoc='center', loc='center')

    table.auto_set_font_size(False)
    table.set_fontsize(12)
    table.auto_set_column_width(col=list(range(len(df_table.columns))))
    table.scale(1, 1.8)

    plt.tight_layout()
    plt.show()


# --- Ejemplo de Uso ---
# NOTA: Asegúrate de tener los DataFrames 'df1' y 'df2' definidos.
# Calcular las métricas para ambos conjuntos de datos
# metricas_df1 = f_metricas_calidad_modelo_datos(df1)
# metricas_df2 = f_metricas_calidad_modelo_datos(df2)
# Llamada a la función para crear el gráfico de radar y la tabla
# radar_plot_with_table(metricas_df1, metricas_df2, df1_name='CICDDoS2019', df2_name='ASSG-DI
```

```
# Lectura Datos de Entrenamiento

def f_process_read_csv(file_path):

    print("1.1- [Data Collection] Process-CSV: [{}]\n".format(file_path))

    factor_sampling = 1 # 1-100
    try:
        filtered_df = pd.read_csv(file_path, low_memory=False, delimiter=',',
                                  quotechar='"')
        start_time = time.time()
        filtered_df = filtered_df.rename(columns=lambda x: x.strip())
```

```
        end_time = time.time()
        execution_time = end_time - start_time
        print("1.2.-[Data Collection] Finalización carga de Tiempo: {:.2f} secs, {}"
              .format(execution_time,file_path))

        filtered_df = filtered_df.iloc[::factor_sampling]

        print("1.4.- [Data Collection] Finalizada la carga: [{}]\n"
              .format(file_path))
        print('1.5.- [Data Collection] - Clasificador: ',
              filtered_df['Label'].unique())
        for classificador in filtered_df['Label'].unique():
            print('Etiqueta: ', classificador, round(filtered_df['Label']
                                                     .value_counts()[classificador]/len(filte
        return filtered_df

    except Exception as ex:
        print("1.1.- [Data Collection] Excepcion: ", ex)
        return None
```

```
# Testing
#_path = 'C:/DataSets/Training/Training_Exp_Attack_TCP.csv'
#_df = f_process_read_csv(_path)
```

```
#Lectura de Datos de Producción

def read_production_data(file_path):

    print("11.1- [Data Collection] Cargando conjunto de datos de producción: [{}]\n"
          .format(file_path))

    try:
        filtered_df = pd.read_csv(file_path, low_memory=False, delimiter=',',
                                  quotechar='"')
        start_time = time.time()
        filtered_df = filtered_df.rename(columns=lambda x: x.strip())
        print('1.2- [Data Collection] Atributos y Características del ' \
        'Conjunto de Datos')
        end_time = time.time()
        execution_time = end_time - start_time

        print("1.3.- [Data Collection] Finalización carga de  Tiempo: " \
```

```
        "{} secs".format(execution_time))
        print("1.4.- [Data Collection] Codificación campos no numericos ")
        label_encoder = LabelEncoder()
        filtered_df['Timestamp'] = label_encoder.fit_transform(filtered_df['Timestamp'])

        if 'Inbound' in filtered_df.columns:
            filtered_df.drop('Inbound', axis=1)

        #filtered_df = filtered_df.iloc[::10]  # Seleccionar solo las filas pares

        print(f"11.5.- [Data Collection] Finalizada la carga: {file_path}")

        print('11.6.- [Data Collection] - Clasificador: ', filtered_df['Label'].unique())
        for classificador in filtered_df['Label'].unique():
            print('Etiqueta: ', classificador, round(filtered_df['Label'].value_counts()[clas

        return filtered_df


    except Exception as ex:
        print("11.1.- [Data Collection] Excepcion: ", ex)

#Test
#_path = 'C:/DataSets/Produccion/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv'
#read_production_data(_path)
```

FASE II.- Preparacion de Datos

[C] Preprocesado de Datos

Se llenan los valores NA de los datos limpios y todas las variables categóricas se codifican en valores numéricos. Se definen las características de la capa de entrada de los algoritmos. Los datos que podrían identificar al consumidor se eliminan a nivel del concentrador.

```
import pandas as pd
import numpy as np
import warnings

def f_preprocesado(data: pd.DataFrame) -> pd.DataFrame:
    """
```

```
    Realiza un preprocesado de datos verificando la presencia de valores nulos
    e infinitos, y muestra los tipos de datos de las columnas.

    Esta función es una fase de verificación para asegurar que el DataFrame
    esté listo para el modelado.

    Args:
        data (pd.DataFrame): El DataFrame de entrada para preprocesar.

    Returns:
        pd.DataFrame: El DataFrame procesado.
    """
    print("3.1.- [Preprocesado] Iniciando el preprocesado de datos.")

    try:
        # --- 3.1.1 Verificación de valores nulos ---
        nan_values_count = data.isnull().sum().sum()
        if nan_values_count > 0:
            print(f"3.1.1.- [Preprocesado] El DataFrame tiene {nan_values_count} nulos.")
            # Opcional: Podrías añadir una estrategia para manejar estos nulos, por ejemplo:
            # data = data.fillna(0)
        else:
            print("3.1.1.- [Preprocesado] Verificado: No hay valores nulos en el DataFrame.")

        # --- 3.1.2 Verificación de valores infinitos (CORREGIDO) ---
        # Se filtran las columnas para trabajar solo con datos numéricos.
        numeric_data = data.select_dtypes(include=[np.number])

        # Se ignora el warning de `isinf` si no hay columnas numéricas.
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            inf_values_count = np.isinf(numeric_data).sum().sum()

        if inf_values_count > 0:
            print(f"3.1.2.- [Preprocesado] El DataFrame contiene {inf_values_count} valores i
            # Opcional: Podrías añadir una estrategia para manejar estos infinitos, por ejem
            # data.replace([np.inf, -np.inf], np.nan, inplace=True)
        else:
            print("3.1.2.- [Preprocesado] Verificado: No hay valores infinitos en el DataFra

        # --- 3.1.3 Muestra de atributos del conjunto de datos ---
        print("\n3.1.3.- [Preprocesado] ATRIBUTOS DEL CONJUNTO DE DATOS PROCESADO:")
```

```
        data.info() # Usar data.info() es una forma más estándar y completa de mostrar esta i

        print("\n3.2.- [Preprocesado] ¡Preprocesado finalizado!")
        return data

    except Exception as ex:
        print(f"\n[Preprocesado] Excepción: {ex}")
        raise
```

[D] Limpieza de Datos

Los datos se limpian. Se recuperan los datos útiles. Se eliminan los datos irrelevantes o incorrectos.

```
import pandas as pd
import numpy as np
import ipaddress
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from typing import NoReturn

def _normalize_ip_address(ip: str) -> str:
    """
    Intenta normalizar una dirección IP. Si no es válida, la devuelve sin cambios.
    """
    try:
        # Se asegura de que la IP sea una cadena válida antes de procesarla.
        if isinstance(ip, str):
            return str(ipaddress.ip_address(ip))
        return str(ip)
    except ValueError:
        return ip
    except Exception:
        # En caso de otro error, devuelve la IP original para no detener el proceso.
        return ip

def f_cleaning(data: pd.DataFrame) -> pd.DataFrame:
    """
    Realiza un proceso de limpieza completo en un DataFrame.

    El proceso incluye la eliminación de duplicados, el manejo de valores
```

```
    infinitos y nulos, el etiquetado de la columna 'Label', la conversión
    de tipos de datos, la normalización de direcciones IP y la eliminación
    de columnas innecesarias.

    Args:
        data (pd.DataFrame): El DataFrame de entrada para limpiar.

    Returns:
        pd.DataFrame: El DataFrame limpio y procesado.

    Raises:
        Exception: Propaga cualquier excepción que ocurra durante la ejecución.
    """
    print("--- 2.1 [Data Cleaning] Iniciando proceso de limpieza de datos. ---")

    try:
        # Paso 1: Eliminar filas duplicadas
        initial_rows = len(data)
        data = data.drop_duplicates()
        removed_duplicates = initial_rows - len(data)
        print(f"-> 2.1.1 Eliminación de filas duplicadas. Filas eliminadas: {removed_duplicat

        # Paso 2: Manejo de valores infinitos y nulos
        print("-> 2.1.2 Manejando valores nulos, infinitos y tipos de datos.")

        # Reemplazar valores infinitos por NaN para una imputación posterior
        data.replace([np.inf, -np.inf], np.nan, inplace=True)

        # Identificar columnas numéricas y categóricas
        numeric_columns = data.select_dtypes(include=np.number).columns
        categorical_columns = data.select_dtypes(include='object').columns

        # Imputar valores nulos en campos numéricos (estrategia: mediana)
        num_imputer = SimpleImputer(strategy='median')
        data[numeric_columns] = num_imputer.fit_transform(data[numeric_columns])

        # Imputar valores nulos en campos categóricos (estrategia: moda)
        cat_imputer = SimpleImputer(strategy='most_frequent')
        data[categorical_columns] = cat_imputer.fit_transform(data[categorical_columns])


        # Paso 3: Proceso de Labeling
```

```
        if 'Label' in data.columns:
            print("-> 2.1.3 Procesando la columna 'Label'.")

            # Unificar etiquetas: todo lo que no sea 'BENIGN' se etiqueta como 'DDoS'
            data['Label'] = data['Label'].apply(lambda x: 'DDoS' if x != 'BENIGN' else x)

            # Presentar la distribución de las etiquetas
            print("-> Presentación de resultados: Distribución de Etiquetas")
            label_counts = data['Label'].value_counts(normalize=True) * 100
            for classificador, porcentaje in label_counts.items():
                print(f"    - Etiqueta: '{classificador}', Porcentaje: {porcentaje:.2f}% del
        else:
            print("-> Advertencia: La columna 'Label' no existe. Se omite el etiquetado.")


        # Paso 4: Conversión de campos y normalización de IPs
        print("-> 2.1.4 Conversión de campos y normalización de IPs.")

        # Convertir 'Timestamp' a formato de fecha y hora, si existe
        if 'Timestamp' in data.columns:
            data['Timestamp'] = pd.to_datetime(data['Timestamp'], errors='coerce')
        else:
            print("-> Advertencia: La columna 'Timestamp' no existe. Se omite la conversión.'

        # Normalizar direcciones IP
        ip_columns = [col for col in ['Source IP', 'Destination IP'] if col in data.columns]
        if ip_columns:
            for col in ip_columns:
                data[col] = data[col].astype(str).apply(_normalize_ip_address)
            print("-> IPs corruptas recuperadas y normalizadas.")
        else:
            print("-> Advertencia: Las columnas de IP no existen. Se omite la normalización.'


        # Paso 5: Eliminar columnas innecesarias
        columns_to_drop = [col for col in ['Unnamed: 0', 'Inbound'] if col in data.columns]
        if columns_to_drop:
            data = data.drop(columns=columns_to_drop)
            print(f"-> 2.1.5 Columnas eliminadas: {columns_to_drop}")

        print("--- 2.1 [Data Cleaning] Proceso de limpieza finalizado. ---")
        return data
```

```
    except Exception as ex:
        print(f"[Data Cleaning] Excepción: {ex}")
        raise # Propagar la excepción
```

[E] Exploración de Datos

Funciones para explorar los Conjuntos de Datos

[E.1] Exploración Gráfica de Datos: Anomalias a través de Diagramas de Caja

[E.2] Exploración Gráfica de Datos: Districución Ataques DDoS del Conjunto de Datos

[E.3] Exploración Gráfica de Datos: Graficos de Tendencia

[E.4] Exploración Gráfica de Datos: Histograma de la Distribucion de Atributos Numéricos

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from typing import List, Optional

# Se desactiva el warning para la conversión de valores infinitos a NaN
# Se desactiva el warning para la conversión de valores infinitos a NaN
warnings.filterwarnings('ignore', category=FutureWarning, message='use_inf_as_na')

def explorar_distribucion_clases(df: pd.DataFrame):
    """
    Realiza un análisis gráfico de la distribución de clases en el conjunto de datos.

    Genera dos gráficos:
    1. Un gráfico de barras que muestra la distribución porcentual de cada clase.
    2. Un gráfico circular que muestra el promedio de 'Flow Packets/s' por clase.

    Args:
        df: DataFrame de pandas con una columna 'Label'.
    """
```

```
print("\n---  Exploración Gráfica: Distribución de Clases ---")
try:
    if 'Label' not in df.columns:
        print("Error: La columna 'Label' no existe en el DataFrame.")
        return

    print("Clases únicas encontradas:", df['Label'].unique())
    for label, count in df['Label'].value_counts().items():
        porcentaje = (count / len(df)) * 100
        print(f"Etiqueta '{label}': {count} ocurrencias ({porcentaje:.2f}%)")

    fig, axes = plt.subplots(1, 2, figsize=(20, 8))

    # Gráfico de barras
    sns.countplot(data=df, x='Label', ax=axes[0], palette='viridis')
    axes[0].set_title('Distribución de Clases (Recuento Absoluto)', fontsize=14)
    axes[0].set_xlabel('Clase', fontsize=12)
    axes[0].set_ylabel('Recuento', fontsize=12)

    # --- CORRECCIÓN MÁS ROBUSTA ---
    # 1. Limpiar la columna antes de agrupar para manejar NaNs e Infinitos
    df['Flow Packets/s'] = df['Flow Packets/s'].replace([float('inf'), float('-inf')], 0)

    # 2. Calcular el promedio
    class_flow_packets = df.groupby('Label')['Flow Packets/s'].mean()

    # 3. Verificar si hay datos para graficar
    if not class_flow_packets.empty and class_flow_packets.sum() > 0:
        axes[1].pie(class_flow_packets, labels=class_flow_packets.index, autopct='%1.1f%%
        axes[1].set_title('Promedio de Flujo de Paquetes/s por Clase', fontsize=14)
    else:
        axes[1].set_title('No hay datos válidos para el gráfico de pastel', fontsize=14)
        axes[1].axis('off') # Ocultar ejes si no hay gráfico

    plt.suptitle('Análisis de la Distribución de Clases', fontsize=16, y=1.02)
    plt.tight_layout()
    plt.show()
    print("\n---  Proceso de exploración de clases finalizado.")

except KeyError as ke:
    print(f"Error de columna: {ke}. Asegúrate de que las columnas 'Label' y 'Flow Packets
except Exception as ex:
```

```
        print(f"Error inesperado durante la exploración de datos: {ex}")


#Testing
#explorar_distribucion_clases(_df)
```

```
import pandas as pd
from typing import List, Optional
from tabulate import tabulate

def presentar_estadisticas_descriptivas(df: pd.DataFrame, columns: Optional[List[str]] = None
    """
    Muestra un resumen conciso de las estadísticas descriptivas de las columnas
    en un formato de tabla legible.

    Args:
        df: DataFrame de pandas.
        columns: Lista de nombres de columnas. Si es None, muestra las
        estadísticas de todas.
    """

    try:
        df_to_describe = df[columns] if columns else df

        # Seleccionamos las estadísticas más importantes para una vista rápida
        descriptive_stats = df_to_describe.describe().loc[['mean', 'std', 'min',
                                                           '25%', '50%', '75%', 'max']]
        # --- CAMBIO AQUÍ: Se utiliza el formato 'github' para una tabla más limpia ---
        #print(tabulate(descriptive_stats, headers='keys', tablefmt='github'))
        print(f"\n---  Resumen de Estadísticas Descriptivas ---\n {descriptive_stats}")
        print("\n---  Proceso de estadísticas descriptivas finalizado.")

    except KeyError as ke:
        print(f"Error: Una o más columnas no se encontraron en el DataFrame: {ke}")
    except Exception as ex:
        print(f"Error inesperado al generar estadísticas: {ex}")

#presentar_estadisticas_descriptivas(_df, PCA_8_Attributes)
```

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

```
def f_show_histograms(_df: pd.DataFrame):
    """
    Genera y muestra histogramas para los 25 atributos de PCA_25_Attributes.

    Args:
        _df (pd.DataFrame): El DataFrame que contiene los datos a visualizar.
    """
    # Configuración de estilo
    try:
        plt.style.use('seaborn-v0_8')
    except:
        plt.style.use('ggplot')

    sns.set_theme(style="whitegrid")
    sns.set_palette("husl")
    plt.rcParams['figure.figsize'] = [25, 25]  # Ajuste de tamaño para una cuadrícula 5x5
    plt.rcParams['figure.dpi'] = 100

    # Atributos de PCA_25_Attributes (excluyendo los que no son aptos para histogramas)
    numeric_pca_attributes = [
        'Fwd IAT Std', 'Fwd IAT Mean', 'Fwd IAT Total', 'Flow IAT Min', 'Flow IAT Max',
        'Flow IAT Std', 'Flow IAT Mean', 'Flow Packets/s', 'Flow Bytes/s',
        'Bwd Packet Length Std', 'Bwd Packet Length Mean', 'Bwd Packet Length Min',
        'Bwd Packet Length Max', 'Fwd Packet Length Std', 'Fwd Packet Length Mean',
        'Fwd Packet Length Min', 'Fwd Packet Length Max', 'Total Length of Bwd Packets',
        'Total Length of Fwd Packets', 'Total Backward Packets', 'Total Fwd Packets',
        'Flow Duration'
    ]

    # Crear la cuadrícula de histogramas (5 filas y 5 columnas para los 25 atributos)
    fig, axes = plt.subplots(5, 5, figsize=(25, 25))
    axes = axes.ravel()  # Aplanar los ejes para iterar fácilmente

    # Bucle para generar cada histograma
    for i, var in enumerate(numeric_pca_attributes):
        if var in _df.columns:
            # Eliminar infinitos y nulos para evitar errores
            data_cleaned = _df[var].replace([float('inf'), -float('inf')],
                                            pd.NA).dropna()

            sns.histplot(data=data_cleaned, kde=True, ax=axes[i], bins=50)
            axes[i].set_title(f'Distribución de {var}', fontsize=12)
```

```
            axes[i].set_xlabel(var, fontsize=10)
            axes[i].set_ylabel('Frecuencia', fontsize=10)

            # Añadir línea de media si el valor es válido
            mean_val = data_cleaned.mean()
            if pd.notna(mean_val):
                axes[i].axvline(mean_val, color='red', linestyle='--',
                                label=f'Media: {mean_val:.2f}')
                axes[i].legend()

    # Ocultar subgráficos no utilizados
    for i in range(len(numeric_pca_attributes), len(axes)):
        fig.delaxes(axes[i])

    plt.tight_layout()
    plt.suptitle('Histogramas de los 25 Atributos de PCA', y=1.02, fontsize=20)
    plt.show()


#Testing
#_show_histograms(_df)
```

```
##############################################################################
## Nombre:  f_exploracion_grafica_datos_boxplot
## Descripción: Exploración gŕafica anomalias a través de Diagramas de Caja
##############################################################################

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Optional

def f_exploracion_grafica_datos_boxplot(_df: pd.DataFrame):
    """
    Genera diagramas de caja para los 8 atributos clave del DataFrame, agrupados
    por 'Label'. Calcula y muestra el recuento de outliers para cada clase.

    Args:
        _df (pd.DataFrame): El DataFrame de entrada.
    """
    print("\n---  Exploración Gráfica de Diagramas de Caja ---")
    try:
```

```
        # 1. Atributos clave a visualizar (los 8 atributos más importantes)
        variables = [
            'Fwd Packet Length Mean', 'Fwd Packet Length Min', 'Fwd Packet Length Max',
            'Total Length of Bwd Packets', 'Total Length of Fwd Packets',
            'Total Backward Packets', 'Total Fwd Packets', 'Flow Duration'
        ]

        # 2. Configuración del gráfico
        fig, axes = plt.subplots(2, 4, figsize=(25, 12))
        fig.suptitle('Análisis de Outliers en Atributos Clave', fontsize=20, y=1.02)

        axes_flat = axes.flatten()  # Aplanar la matriz de ejes para facilitar la iteración

        # 3. Limpieza de datos antes de graficar (para evitar errores con NaNs e Infinitos)
        df_cleaned = _df.copy()
        for var in variables:
            df_cleaned[var] = df_cleaned[var].replace([float('inf'), -float('inf')], pd.NA).d

        # 4. Bucle para generar cada diagrama de caja
        for i, var in enumerate(variables):
            ax = axes_flat[i]

            # Crear el boxplot agrupado por 'Label'
            sns.boxplot(x="Label", y=var, data=df_cleaned, ax=ax, palette="viridis")
            ax.set_title(f'Diagrama de Caja de "{var}"', fontsize=12)
            ax.set_ylabel(var, fontsize=10)
            ax.set_xlabel('Clase', fontsize=10)

            # 5. Calcular y mostrar outliers por clase
            labels = df_cleaned['Label'].unique()
            y_pos = 0.95
            for label_index, label in enumerate(labels):
                subset = df_cleaned[df_cleaned['Label'] == label][var].dropna()

                if not subset.empty:
                    q1 = subset.quantile(0.25)
                    q3 = subset.quantile(0.75)
                    iqr = q3 - q1
                    lower_bound = q1 - 1.5 * iqr
                    upper_bound = q3 + 1.5 * iqr

                    outliers_count = len(subset[(subset < lower_bound) | (subset > upper_bou
```

```
                    total_count = len(subset)

                    if total_count > 0:
                        percentage = (outliers_count / total_count) * 100
                    else:
                        percentage = 0

                    ax.text(
                        x=label_index, y=y_pos,
                        s=f'Outliers ({label}):\n{outliers_count} ({percentage:.1f}%)',
                        horizontalalignment='center',
                        verticalalignment='top',
                        transform=ax.get_xaxis_transform(), # Alineación al eje x
                        bbox=dict(facecolor='white', alpha=0.8, edgecolor='none')
                    )
                    y_pos -= 0.15 # Espacio entre etiquetas

        plt.tight_layout(rect=[0, 0, 1, 0.98]) # Ajuste para que el título no se superponga
        plt.show()

        print("\n---  Proceso de exploración de boxplots finalizado.")

    except Exception as ex:
        print(f"[D.1 Exploracion Gráfica de Diagramas de Caja] - Excepción: {ex}")


#f_exploracion_grafica_datos_boxplot(_df)
```

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

def f_exploracion_grafica_scatterplots_ataques(_df: pd.DataFrame):
    """
    Genera dos scatter plots para visualizar las relaciones entre atributos clave
    y diferenciar entre tráfico normal y de ataque.

    Los gráficos muestran:
    1.  Total de paquetes de avance vs. Longitud media del paquete.
    2.  Tasa de paquetes por segundo vs. Tasa de bytes por segundo.

    Args:
```

```
        _df (pd.DataFrame): DataFrame de pandas que contiene los datos.
    """
    print("\n---  Exploración Gráfica de Scatter Plots para Ataques ---")

    required_cols = [
        'Total Fwd Packets', 'Fwd Packet Length Mean', 'Flow Duration', 'Label',
        'Flow Packets/s', 'Flow Bytes/s'
    ]

    try:
        # Validación de que las columnas necesarias existen
        if not all(col in _df.columns for col in required_cols):
            missing_cols = [col for col in required_cols if col not in _df.columns]
            raise KeyError(f"Columnas faltantes en el DataFrame: {missing_cols}")

        fig, axes = plt.subplots(1, 2, figsize=(18, 8))

        # Paleta de colores dinámica
        n_labels = _df['Label'].nunique()
        palette = sns.color_palette("viridis", n_labels)

        # Scatter Plot 1: Total de paquetes de avance vs. Longitud media del paquete
        sns.scatterplot(
            data=_df,
            x='Total Fwd Packets',
            y='Fwd Packet Length Mean',
            size='Flow Duration',
            sizes=(50, 500),
            hue='Label',
            ax=axes[0],
            palette=palette
        )
        axes[0].set_title('Total de paquetes de avance vs. Longitud media', fontsize=14)
        axes[0].set_xlabel('Total de paquetes de avance', fontsize=12)
        axes[0].set_ylabel('Longitud media del paquete', fontsize=12)
        axes[0].legend(title='Clase', loc='best')

        # Scatter Plot 2: Tasa de paquetes por segundo vs. Tasa de bytes por segundo
        sns.scatterplot(
            data=_df,
            x='Flow Packets/s',
            y='Flow Bytes/s',
```

```
            size='Flow Duration',
            sizes=(50, 500),
            hue='Label',
            ax=axes[1],
            palette=palette
        )
        axes[1].set_title('Tasa de paquetes/s vs. Tasa de bytes/s', fontsize=14)
        axes[1].set_xlabel('Paquetes/s', fontsize=12)
        axes[1].set_ylabel('Bytes/s', fontsize=12)
        axes[1].legend(title='Clase', loc='best')

        plt.suptitle('Análisis de Comportamiento de Flujos', fontsize=16, y=1.02)
        plt.tight_layout()
        plt.show()

        print("\n---  Proceso de visualización de scatter plots finalizado.")

    except KeyError as ke:
        print(f"Error de columna: {ke}. Asegúrate de que todas las columnas requeridas existe
    except Exception as ex:
        print(f"Error inesperado al generar los scatter plots: {ex}")


#f_exploracion_grafica_scatterplots_ataques(_df)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from typing import List

def f_exploracion_grafica_analisis_clustering(_df: pd.DataFrame):
    """
    Genera una cuadrícula de cuatro scatter plots para visualizar las relaciones
    entre atributos de tráfico de red y su distribución por clase (Label).

    Los gráficos se enfocan en la distribución del tráfico hacia adelante y
    hacia atrás, y en las fluctuaciones de los tiempos entre paquetes (IAT).

    Args:
        _df (pd.DataFrame): DataFrame de pandas con los datos a visualizar.
    """
```

```
print("\n---  Distribución de Indicadores en Flujos de Tráfico de Red ---")

required_cols = [
    'Total Length of Bwd Packets', 'Total Backward Packets',
    'Total Length of Fwd Packets', 'Total Fwd Packets', 'Fwd IAT Max',
    'Flow IAT Std', 'Bwd IAT Max', 'Flow Duration', 'Label'
]

try:
    # Validación de que todas las columnas necesarias existen
    if not all(col in _df.columns for col in required_cols):
        missing_cols = [col for col in required_cols if col not in _df.columns]
        raise KeyError(f"Columnas faltantes en el DataFrame: {missing_cols}")

    fig, axes = plt.subplots(2, 2, figsize=(20, 15))
    fig.suptitle('Análisis de Clustering por Atributos de Flujo', fontsize=20, y=1.02)

    # Paleta de colores dinámica para el número de etiquetas presentes
    n_labels = _df['Label'].nunique()
    palette = sns.color_palette("viridis", n_labels)

    # 1. Gráfica: Tráfico hacia atrás (longitud vs. total de paquetes)
    sns.scatterplot(
        data=_df, x='Total Length of Bwd Packets', y='Total Backward Packets',
        size='Flow Duration', sizes=(50, 500), hue='Label', ax=axes[0, 0],
        palette=palette, legend=False
    )
    axes[0, 0].set_title('Tráfico hacia atrás: Longitud vs. Paquetes', fontsize=14)
    axes[0, 0].set_xlabel('Longitud total de paquetes de vuelta', fontsize=12)
    axes[0, 0].set_ylabel('Total de paquetes de vuelta', fontsize=12)

    # 2. Gráfica: Tráfico hacia adelante (longitud vs. total de paquetes)
    sns.scatterplot(
        data=_df, x='Total Length of Fwd Packets', y='Total Fwd Packets',
        size='Flow Duration', sizes=(50, 500), hue='Label', ax=axes[0, 1],
        palette=palette, legend=False
    )
    axes[0, 1].set_title('Tráfico hacia adelante: Longitud vs. Paquetes', fontsize=14)
    axes[0, 1].set_xlabel('Longitud total de paquetes de avance', fontsize=12)
    axes[0, 1].set_ylabel('Total de paquetes de avance', fontsize=12)

    # 3. Gráfica: Variabilidad temporal (IAT Max Fwd vs. IAT Std)
```

```
        sns.scatterplot(
            data=_df, x='Fwd IAT Max', y='Flow IAT Std',
            size='Flow Duration', sizes=(50, 500), hue='Label', ax=axes[1, 0],
            palette=palette, legend=False
        )
        axes[1, 0].set_title('Variabilidad de tiempos: IAT máx. avance vs. IAT std', fontsize
        axes[1, 0].set_xlabel('Tiempo máximo entre paquetes de avance', fontsize=12)
        axes[1, 0].set_ylabel('Desviación estándar de tiempos entre paquetes', fontsize=12)

        # 4. Gráfica: Variabilidad temporal (IAT Max Bwd vs. IAT Std)
        sns.scatterplot(
            data=_df, x='Bwd IAT Max', y='Flow IAT Std',
            size='Flow Duration', sizes=(50, 500), hue='Label', ax=axes[1, 1],
            palette=palette, legend=True
        )
        axes[1, 1].set_title('Variabilidad de tiempos: IAT máx. vuelta vs. IAT std', fontsize
        axes[1, 1].set_xlabel('Tiempo máximo entre paquetes de vuelta', fontsize=12)
        axes[1, 1].set_ylabel('Desviación estándar de tiempos entre paquetes', fontsize=12)

        # Ajuste de leyenda y layout
        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
        plt.show()

        print("\n---  Proceso de visualización de scatter plots finalizado.")

    except KeyError as ke:
        print(f"Error de columna: {ke}. Asegurar que existen las columnas.")
    except Exception as ex:
        print(f"Error inesperado al generar los scatter plots: {ex}")

#f_exploracion_grafica_analisis_clustering(_df)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from typing import List

def f_plot_variable_evolution(_df: pd.DataFrame):
    """
    Grafica la evolución temporal de atributos críticos, normalizando los datos
    para comparar sus tendencias.
```

```
    Args:
        _df (pd.DataFrame): El DataFrame de entrada con una columna 'Timestamp'.
    """
    print("\n---  Evolución Temporal de Atributos Críticos ---")

    attributes_to_plot = [
        'Flow IAT Std', 'Bwd IAT Std', 'SYN Flag Count', 'Bwd Packet Length Std',
        'Fwd Packet Length Std', 'Total Length of Fwd Packets'
    ]

    try:
        # Validación de que las columnas necesarias existen
        if not all(col in _df.columns for col in ['Timestamp'] + attributes_to_plot):
            missing_cols = [
                col for col in ['Timestamp'] + attributes_to_plot
                if col not in _df.columns
            ]
            raise KeyError(f"Columnas faltantes: {missing_cols}")

        # Crear una copia para evitar SettingWithCopyWarning
        df_normalized = _df.copy()

        # Eliminar NaN e infinitos antes de normalizar
        df_normalized = df_normalized.replace([float('inf'), -float('inf')], pd.NA).dropna(su

        # Normalizar los datos
        scaler = MinMaxScaler()
        df_normalized[attributes_to_plot] = scaler.fit_transform(df_normalized[attributes_to_

        # Configurar la cuadrícula de gráficos
        fig, axes = plt.subplots(3, 3, figsize=(20, 15))
        axes = axes.ravel()

        for i, attribute in enumerate(attributes_to_plot):
            axes[i].plot(df_normalized['Timestamp'], df_normalized[attribute], label=attribut
            axes[i].set_xlabel('Timestamp', fontsize=10)
            axes[i].set_ylabel('Valor Normalizado', fontsize=10)
            axes[i].set_title(f'Evolución temporal de {attribute}', fontsize=12)
            axes[i].legend(loc='upper left')

        # Marcar puntos de ataque si la clase 'DrDoS_SNMP' existe
        if 'DrDoS_SNMP' in df_normalized['Label'].unique():
```

```
            attack_data = df_normalized[df_normalized['Label'] == 'DrDoS_SNMP']
            axes[len(attributes_to_plot)].scatter(
                attack_data['Timestamp'],
                attack_data['Total Length of Fwd Packets'],  # Se usa un atributo de ejemplo
                color='red', marker='o', label='Ataque: DrDoS_SNMP'
            )
            axes[len(attributes_to_plot)].set_xlabel('Timestamp', fontsize=10)
            axes[len(attributes_to_plot)].set_ylabel('Total de Paquetes Avance', fontsize=10)
            axes[len(attributes_to_plot)].set_title('Eventos de Ataque', fontsize=12)
            axes[len(attributes_to_plot)].legend(loc='upper left')

        # Ocultar subgráficos no utilizados
        for i in range(len(attributes_to_plot) + 1, len(axes)):
            fig.delaxes(axes[i])

        plt.suptitle('Tendencias de los Indicadores Críticos', fontsize=16, y=1.02)
        plt.tight_layout()
        plt.show()

        print("\n---  Proceso de visualización de tendencias finalizado.")

    except KeyError as ke:
        print(f"Error de columna: {ke}. Por favor, verifica el nombre de los atributos.")
    except Exception as ex:
        print(f"Error inesperado al generar los gráficos de evolución: {ex}")

#f_plot_variable_evolution(_df)
```

[E] Borrado de anomalías en la distribución

[E.1] Matriz de Confusión

Función para crear y evaluar una matriz de confusión utilizando K-Nearest Neighbors (KNN) Classification.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import warnings
import ipaddress

def _normalize_ip_address(ip: str) -> str:
    """
    Intenta normalizar una dirección IP. Si no es válida, la devuelve sin cambios.
    """
    try:
        if isinstance(ip, str):
            return str(ipaddress.ip_address(ip))
        return str(ip)
    except ValueError:
        return ip
    except Exception:
        return ip

def f_make_confusion_matrix_with_knn(_df: pd.DataFrame):
    """
    Realiza una clasificación KNN y visualiza una matriz de confusión para un
    DataFrame dado.

    Args:
        _df (pd.DataFrame): El DataFrame de entrada.
    """

    warnings.filterwarnings("ignore", category=UserWarning)

    print("\n---  Generando Matriz de Confusión con Clasificador KNN ---")

    try:
        data = _df.copy()

        print("E.1.- [Preprocesamiento] Limpieza y codificación de datos.")

        required_columns = ['Source IP', 'Destination IP', 'Label']
        if not all(col in data.columns for col in required_columns):
            missing_cols = [col for col in required_columns if col not in data.columns]
            raise ValueError(f"Columnas requeridas faltantes: {missing_cols}")

        data.dropna(inplace=True)
```

```
label_encoder = LabelEncoder()
data['Label_encoded'] = label_encoder.fit_transform(data['Label'])
class_labels = label_encoder.classes_

# --- CORRECCIÓN CLAVE ---
# Definir las columnas de características y la variable objetivo
feature_cols = [col for col in data.columns if pd.api.types.is_numeric_dtype(data[col
X_features = data[feature_cols]
y_target = data['Label_encoded']

# --- NUEVA CORRECCIÓN ---
# 1. Reemplazar valores infinitos con NaN para poder eliminarlos.
X_features.replace([np.inf, -np.inf], np.nan, inplace=True)

# 2. Eliminar las filas que ahora contienen NaN.
# Esto asegura que no haya valores problemáticos en el conjunto de datos.
X_features.dropna(inplace=True)
y_target = y_target[X_features.index] # Asegurar que y_target se alinee con las filas

if X_features.empty:
     raise ValueError("El DataFrame de características numéricas está vacío después d
if y_target.empty:
     raise ValueError("La serie de la variable objetivo está vacía después de la limp

print("E.2.- [Modelo] Entrenando clasificador KNN.")

X_train, X_test, y_train, y_test = train_test_split(
    X_features, y_target, test_size=0.2, random_state=42
)

k_value = 5
knn_classifier = KNeighborsClassifier(n_neighbors=k_value)
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)

print("\nE.3.- [Resultados] Evaluación del modelo.")

accuracy = accuracy_score(y_test, y_pred)
print(f"  Precisión del modelo (Accuracy): {accuracy:.4f}")

print("\nInforme de Clasificación:")
print(classification_report(y_test, y_pred, target_names=class_labels))
```

```
        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(10, 8))
        sns.heatmap(
            cm,
            annot=True,
            fmt='d',
            cmap='Blues',
            xticklabels=class_labels,
            yticklabels=class_labels
        )
        plt.title('Matriz de Confusión', fontsize=16)
        plt.xlabel('Predicho', fontsize=12)
        plt.ylabel('Verdadero', fontsize=12)
        plt.show()

    except ValueError as ve:
        print(f"  Error de valor: {ve}. No se pudo continuar con la clasificación.")
    except Exception as ex:
        print(f"  Error inesperado: {ex}. Por favor, revisa datos y dependencias.")
#f_make_confusion_matrix_with_knn(_df)
```

Fase III.- Modelado

[Feature Engineering] - Seleccion de Atributos/Caracteristicas

[M] Matriz de Correlacion: Reducción de Dimensiones

En esta Fase del Proyecto de Ingeniería de Dátos se realizará un estudio de las Matrices de Correlación con el objetivo de identificar las variables con una mayor influencia en una operación específica identificada como fraudulenta.

Sin embargo, será de vital importancia la utilización de un DataFrame (Sub-Conjunto de Datos) para poder observar de una manera visual que carácteristicas / variables tienen una correlación positiva o negativa con respecto a la incidencia del tráfico de red asociado a un posible ataque de Denegación Distribuida del Servicio.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Dict, Any
```

```
def plot_correlation_matrix(_df: pd.DataFrame, title: str, ax):
    """
    Función auxiliar para graficar una matriz de correlación en un subgráfico.

    Args:
        _df (pd.DataFrame): DataFrame con los datos de correlación.
        title (str): Título del gráfico.
        ax: Eje de Matplotlib para dibujar el gráfico.
    """
    sns.heatmap(_df, cmap='coolwarm_r', annot=True, annot_kws={'size': 8}, fmt=".2f", ax=ax,
    ax.set_title(title, fontsize=12)
    ax.tick_params(axis='both', which='major', labelsize=8)

def f_make_correlation_analysis(_df: pd.DataFrame) -> pd.DataFrame:
    """
    Realiza un análisis completo de correlación, incluyendo:
    1. Codificación de la columna 'Label'.
    2. Cálculo de la matriz de correlación.
    3. Identificación de las variables más y menos correlacionadas.
    4. Visualización de las matrices de correlación.
    5. Retorno de un subconjunto del DataFrame con las características principales.

    Args:
        _df (pd.DataFrame): El DataFrame de entrada. Debe contener la columna 'Label'.

    Returns:
        pd.DataFrame: Un nuevo DataFrame con las columnas principales identificadas.
    """

    print("\n---  Análisis de Matriz de Correlación ---")

    # Validar que el DataFrame no esté vacío y contenga la columna 'Label'
    if _df.empty or 'Label' not in _df.columns:
        print("  Error: El DataFrame está vacío o no contiene la columna 'Label'.")
        return pd.DataFrame()

    try:
        # Copiar el DataFrame para no modificar el original
        dataset = _df.copy()

        # 1. Preprocesamiento: Codificar la columna 'Label'
        print("M.1.- [Análisis] Codificando la columna 'Label'...")
```

```
        unique_labels = sorted(dataset['Label'].unique())
        label_map = {label: idx for idx, label in enumerate(unique_labels)}
        dataset['Label'] = dataset['Label'].map(label_map)

        print(f"M.1.1.- [Análisis] Etiquetas codificadas: {label_map}")

        # 2. Calcular la matriz de correlación
        print("M.2.- [Análisis] Calculando la matriz de correlación...")
        correlation_matrix = dataset.corr(numeric_only=True)

        # Validación de que se pudo calcular la matriz
        if correlation_matrix.empty:
            raise ValueError("No se pudo calcular la matriz de correlación. ¿Hay suficientes

        # 3. Presentación de correlaciones principales
        print("\nM.3.- [Análisis] Atributos más correlacionados con la etiqueta 'Label'...")
        corr_with_label = correlation_matrix['Label'].sort_values(ascending=False)

        # Correlaciones positivas más altas (incluyendo 'Label' misma)
        top_positive_corr = corr_with_label.iloc[1:11]
        print("  Correlaciones Positivas:")
        print(top_positive_corr)

        # Correlaciones negativas más bajas
        top_negative_corr = corr_with_label.tail(10)
        print("\n  Correlaciones Negativas:")
        print(top_negative_corr)

        # 4. Visualización de matrices de correlación
        print("\nM.4.- [Visualización] Generando gráficos de la matriz de correlación.")

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 10))

        # Gráfico 1: Matriz completa
        plot_correlation_matrix(correlation_matrix, "Matriz de Correlación Completa", ax1)

        # Gráfico 2: Matriz de correlación del subconjunto de características principales
        main_features = list(top_positive_corr.index)
        if len(main_features) > 1:
            sub_sample_corr = dataset[main_features].corr(numeric_only=True)
            plot_correlation_matrix(sub_sample_corr, "Matriz de Correlación del Subconjunto d
        else:
```

```
            print("No hay suficientes características principales para graficar un subconjunt
            fig.delaxes(ax2) # Eliminar el segundo eje si no se usa

        plt.tight_layout()
        plt.show()

        # 5. Obtener los atributos principales para retornar
        # Seleccionar las 15 características más correlacionadas (positiva y negativamente)
        n_dim = 15
        top_15_attributes = corr_with_label.iloc[1:].abs().nlargest(n_dim).index.tolist()

        print(f"\nM.5.- [Análisis] Proceso completado. Atributos principales seleccionados (
        print(top_15_attributes)

        # Retornar un subconjunto con las columnas principales
        # Se añaden columnas clave que pueden no tener una alta correlación pero son importa
        # para la identificación de flujos, como IPs y puertos.
        #important_cols = ['Source IP', 'Destination IP', 'Source Port', 'Destination Port',
        important_cols =[]
        final_cols = list(set(top_15_attributes + important_cols + ['Label']))
        final_df = dataset[final_cols]

        return final_df

    except Exception as ex:
        print(f"  Error inesperado durante el análisis de correlación: {ex}")
        return pd.DataFrame()


#f_make_correlation_analysis(_df)
```

[N] Detección de Anomalías / Detección de Outliers

La detección de anomalías, también conocida como detección de outliers, es un proceso esencial en el análisis de datos. Su principal objetivo es identificar y, en muchos casos, eliminar valores atípicos que pueden distorsionar los resultados de los modelos de aprendizaje automático. En el contexto de la ciberseguridad, esto nos permite refinar los datos de red, centrándonos en aquellos patrones que son representativos del comportamiento normal, lo que a su vez puede mejorar la precisión en la detección de ataques.

[N.1] Metodo de Rango Intercuartile (IRQ):

Rango Intercuartile (IQR): El Rango Intercuartílico (IQR) es una herramienta estadística fundamental para la detección de anomalías. A diferencia de otros métodos que se basan en la media y la desviación estándar, el IQR es robusto a los valores extremos, lo que lo hace ideal para datos sesgados.

Definición: El IQR es la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1), que representan los percentiles 75 y 25, respectivamente. Esto captura el 50% central de los datos: IQR=Q3−Q1

Diagramas de Caja/ Boxplots: Boxplots: Los diagramas de caja (boxplots) son la representación gráfica más común para visualizar el IQR y los outliers. La "caja" del diagrama representa el IQR, y los "bigotes" se extienden hasta los límites aceptables de los datos. Cualquier punto que se encuentre fuera de estos bigotes se considera un outlier.

[N.2] Borrado de Anomalías :

Para aplicar el método IQR de manera efectiva, se sigue una estrategia estructurada:

- Cálculo de los Umbrales: Se definen límites superior e inferior para identificar los outliers. Estos umbrales se calculan usando una fórmula simple:

Umbral Inferior = Q1−(Factor×IQR)

Umbral Superior = Q3+(Factor×IQR)

El factor más común es 1.5. Un valor de 3.0 se usa para detectar outliers más extremos.

- Visualización Inicial: Antes de cualquier modificación, se visualiza la distribución de las características mediante diagramas de caja. Esto nos da una idea clara de la cantidad y la ubicación de los outliers.
- Eliminación Condicional: Se crea un procedimiento para eliminar de forma sistemática cualquier registro (fila) que tenga valores que caigan por debajo del umbral inferior o por encima del umbral superior en las características seleccionadas.
- Visualización Final: Después de la eliminación, se genera un nuevo diagrama de caja para la misma característica. Esto permite verificar que el proceso de limpieza fue exitoso y que los datos están ahora más concentrados, sin los valores extremos que podían sesgar el modelo.

Sumario:

Visualización de Distribuciones: Se comenzara por la visualización de la distribución de la carácteristica, que vamos a utilizar para eliminar algunos de las anomalías/outliers.

Determinaremos el Humbral: Despues se decidirá que número se utilizará para multiplicar con el IQR para eliminar los valores proximos, mas numerosos, al humbral definido. La forma de proceder será determinando el valor elevado y el valor inferior, sustrayendo el humbral q25 / 25th (Humbral Extremo Inferior) y sumando el humbral q75 / 75th (Humbral Extremo Superior).

Borrado Condicional: Finalmente, se creará un procedimiento borrado condicioanal, por el cual se borraran las instancias que excedan por ambos extremos

Representación den Diagramas de Cajas: Visualizaremos a través del diagrama de cajas el número de anomalías en los extremos, observando que han sido reducidas consideramente.

Nota: La eliminación de outliers, cuando se hace correctamente, puede mejorar la precisión del modelo en varios puntos porcentuales. Sin embargo, es vital ser cauteloso. Eliminar demasiados puntos podría resultar en una pérdida de información valiosa, especialmente en un contexto de ciberseguridad, donde un "outlier" podría ser el indicio de un ataque real. Por ello, se recomienda un análisis cuidadoso y pruebas exhaustivas para determinar el factor IQR ideal.

Referncias: Mas información Metodo IRQ: How to Use Statistics to Identify Outliers in Data by Jason Brownless (Machine Learning Mastery blog)

```python
import numpy as np
import pandas as pd
from typing import List

def cleaning_outliers(new_df: pd.DataFrame, atributos: List[str]) -> pd.DataFrame:
    """
    Identifica y elimina outliers en múltiples atributos de un DataFrame usando el método IQ

    El proceso itera sobre una lista de atributos, calcula los umbrales IQR y
    elimina las filas que contienen valores fuera de esos umbrales.

    Args:
        new_df (pd.DataFrame): El DataFrame de entrada.
        atributos (List[str]): Una lista de nombres de columnas numéricas a limpiar.

    Returns:
        pd.DataFrame: Un DataFrame con los outliers eliminados.
    """
```

```
    try:
        print('N.- [Limpieza de Outliers]: Proceso iniciado')
        print('--------------------------------------------------------------------------------

        # Crear una copia del DataFrame para no modificar el original
        # y evitar el error SettingWithCopyWarning
        df_cleaned = new_df.copy()

        for atributo in atributos:
            # Validar que el atributo exista y sea numérico
            if atributo not in df_cleaned.columns:
                print(f" Error: El atributo '{atributo}' no existe en el DataFrame. Se omite
                continue
            if not pd.api.types.is_numeric_dtype(df_cleaned[atributo]):
                print(f" Advertencia: El atributo '{atributo}' no es numérico. Se omite.")
                continue

            # Calcular los cuartiles y el rango intercuartílico (IQR)
            q25, q75 = np.percentile(df_cleaned[atributo], 25), np.percentile(df_cleaned[atri
            atributo_iqr = q75 - q25

            # Calcular el umbral (cut-off) y los límites para los outliers
            atributo_cut_off = atributo_iqr * 1.5
            atributo_lower, atributo_upper = q25 - atributo_cut_off, q75 + atributo_cut_off

            print(f"Estadísticas para el atributo '{atributo}':")
            print(f"  - Quartil 25 (Q1): {q25:.2f}")
            print(f"  - Quartil 75 (Q3): {q75:.2f}")
            print(f"  - IQR: {atributo_iqr:.2f}")
            print(f"  - Umbral (Cut-off): {atributo_cut_off:.2f}")
            print(f"  - Límite Inferior: {atributo_lower:.2f}")
            print(f"  - Límite Superior: {atributo_upper:.2f}")

            # Identificar los outliers
            outliers_condition = (df_cleaned[atributo] > atributo_upper) | (df_cleaned[atribu
            outliers_indices = df_cleaned[outliers_condition].index

            print('N.1 - [Limpieza de Outliers]: Identificación de Outliers')
            print(f'  - Número de outliers identificados en {atributo}: {len(outliers_indices

            # Borrar los outliers del DataFrame
            print('N.2 - [Limpieza de Outliers]: Borrado de Outliers')
```

```
            df_cleaned.drop(outliers_indices, inplace=True)

            print(f'  - Número de registros después del borrado de outliers en {atributo}: {l
            print('-------------------------------------------------------------------------

        print('N.- [Limpieza de Outliers]: Outliers eliminados. Proceso completado.')
        return df_cleaned

    except Exception as ex:
        print(f"[Limpieza de Outliers] - Excepción: {ex}")
        return new_df

#_df_cleaned = cleaning_outliers(_df, PCA_25_Attributes)
```

[O] Reducción dimensional y clustering

[O.1] t-distributed Stochastic Neighbor Embedding (t-SNE)

El algoritmo t-SNE fue desarrollado por Van der Maaten y Hinton en 2008, como una herramienta innovadora para un escalado multidimensional. Esta técnica es muy popular para el desarrollo de algoritmos ML, debido a que puede ser utiliada para integrar un elebado número de dimensiones de datos, en un número inferior, tal y según es el caso del presente estudio.

1. Distancia euclídea
2. Probabilidad condicional
3. Ploteado de la Normal y la T-Distribution

Sumario del Procedimiento:

1. El Algorimo t-SNE puede proporcionar una información bastante ajustada a través de una análisis basado en clustering de un conjunto de datos como el sometido a estudio, distinguiendo gráficamente las transacciones fraudulentas de las genuinas.
2. A traveś de una submuestra menor y ajustada a nuestras necesidades, el algoritmos t-SNE algorithm es capaz de detectar clusters bastante precisos de cada escenario (Ataque DDoS / Benigno)
3. La utilidad es proporcionar un indicador que nos permita realizar la diferenciacion y separación de casos de transacciones fraudulentas de las que son genuinas.

[O.2] Análisis en Componentes Principales

Conviene introducir el concepto de Análisis de Componentes Fundamentales, antés de de centrarse en el proceso de procesamiento y limpieza de carácterísticas PCA: PCA es una forma eficiente de reducir el número de dimensiones que no aportan información al modelado

de un sistema, permitiendo eliminar la correlación de características 1. PCA Transformation: La transformación PCA, como se puede deducir a partir de la previa definición es la que nos permitirá en el modelado de detección de posibles fraudes, reducir las dimensiones de características, excepto la dimensión de tiempo y cantidad 2. Scaling: Keep in mind that in order to implement a PCA transformation features need to be previously scaled. (In this case, all the V features have been scaled or at least that is what we are assuming the people that develop the dataset did.)

[O.3] TruncatedSVD

TruncatedSVD (Descomposición de Valores Singulares Truncados) es una técnica de reducción de dimensionalidad que se utiliza comúnmente en conjuntos de datos dispersos o matrices dispersas de alta dimensionalidad. Funciona de manera similar a la descomposición de valores singulares (SVD), pero en lugar de calcular todos los componentes principales, solo calcula los primeros k componentes principales.

1. El objetivo principal de TruncatedSVD es reducir la dimensionalidad de los datos al preservar la mayor cantidad posible de varianza. Esto se logra seleccionando los k componentes principales más importantes, que capturan la mayor parte de la varianza en los datos originales.
2. TruncatedSVD es especialmente útil cuando se trabaja con grandes conjuntos de datos, ya que es más eficiente computacionalmente que otras técnicas de reducción de dimensionalidad como PCA (Análisis de Componentes Principales) en matrices dispersas.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD
import matplotlib.patches as mpatches
from sklearn.preprocessing import StandardScaler

def clean_numeric_fields(new_df):

    # Paso 1: Identificar columnas no numéricas
    non_numeric_columns = new_df.select_dtypes(exclude=['number']).columns
    print("X.Xx.- [Cleanning] Eliminación de atributos no numéricos")
    # Paso 2: Eliminar filas con valores no numéricos en esas columnas
    new_df = new_df.dropna(subset=non_numeric_columns, how='any')
    # Paso 3: Convertir las columnas a tipo de dato numérico
    new_df[non_numeric_columns] = new_df[non_numeric_columns].apply(pd.to_numeric, errors='c
    # Eliminar filas con valores NaN después de la conversión
```

```
    new_df.dropna(inplace=True)
    # Seleccionar solo las columnas de tipo int64
    int_columns = new_df.select_dtypes(include=['int64']).columns
    return new_df[int_columns]




def f_dimensionality_reduction_comparison(_df):
    try:
        print("[0].- Otras técnicas de Reducción de Dimensiones: ")

        reduced_set_of_features_2 = ['Protocol', 'Bwd Header Length', 'Min Packet Length','Fw
         'Fwd Packet Length Mean', 'Avg Fwd Segment Size', 'Average Packet Size',
         'Packet Length Mean', 'Fwd Packet Length Max', 'Max Packet Length','Total Length of
         'Subflow Fwd Bytes', 'Flow Bytes/s', 'Fwd Packets/s','Label']

        # Verificar si todas las características en reduced_set_of_features están presentes e
        missing_features = [feature for feature in reduced_set_of_features_2 if feature not i
        if missing_features:
            raise ValueError(f"Las siguientes características no están presentes en el DataFr

        new_df = _df[reduced_set_of_features_2]
        X = new_df.drop('Label', axis=1)
        y = new_df['Label']


        print("[0.1].- Normalización y Escalado de variables")

        # Normalización de los datos
        scaler = StandardScaler()
        features_scaled = scaler.fit_transform(new_df)


        # Reducción de dimensionalidad con T-SNE
        print('[0.2] t-distributed Stochastic Neighbor Embedding (t-SNE)')
        t0 = time.time()
        tsne = TSNE(n_components=2, random_state=42)
        X_reduced_tsne = tsne.fit_transform(features_scaled)
        t1 = time.time()
        print("Tiempo: T-SNE {:.2f} secs".format(t1 - t0))
```

```
# Reducción de dimensionalidad con PCA
print('[0.2] Principal Component Analysis (PCA)')
t0 = time.time()
pca = PCA(n_components=2, random_state=42)
X_reduced_pca = pca.fit_transform(features_scaled)
t1 = time.time()
print("Tiempo: PCA {:.2f} secs".format(t1 - t0))

# Reducción de dimensionalidad con TruncatedSVD
print('[0.3] TruncatedSVD')
t0 = time.time()
svd = TruncatedSVD(n_components=2, algorithm='randomized', random_state=42)
X_reduced_svd = svd.fit_transform(features_scaled)
t1 = time.time()
print("Tiempo: TruncatedSVD {:.2f} secs".format(t1 - t0))


# Gráfico de dispersión
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
fig.suptitle('Reducción Dimensional basada en Análisis de Clustering', fontsize=14)

blue_patch = mpatches.Patch(color='#0A0AFF', label='Tráfico Benigno')
red_patch = mpatches.Patch(color='#AF0000', label='Tráfico Ataque DDoS')

# Gráfico de dispersión T-SNE
axes[0].scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(_df['Label'] == 2),
                cmap='coolwarm', label='Tráfico Benigno', linewidths=2)
axes[0].scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(_df['Label'] == 1),
                cmap='coolwarm', label='Tráfico Ataques DDoS', linewidths=2)
axes[0].set_title('T-SNE', fontsize=14)
axes[0].grid(True)
axes[0].legend(handles=[blue_patch, red_patch])

# Gráfico de dispersión PCA
axes[1].scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(_df['Label'] == 2),
                cmap='coolwarm', label='Tráfico Benigno', linewidths=2)
axes[1].scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(_df['Label'] == 1),
                cmap='coolwarm', label='Tráfico Ataque DDoS', linewidths=2)
axes[1].set_title('PCA', fontsize=14)
axes[1].grid(True)
axes[1].legend(handles=[blue_patch, red_patch])
```

```
        # Gráfico de dispersión Truncated SVD
        axes[2].scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(_df['Label'] == 2),
                        cmap='coolwarm', label='Tráfico Benigno', linewidths=2)
        axes[2].scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(_df['Label'] == 1),
                        cmap='coolwarm', label='Tráfico Ataque DDoS', linewidths=2)
        axes[2].set_title('Truncated SVD', fontsize=14)
        axes[2].grid(True)
        axes[2].legend(handles=[blue_patch, red_patch])

        plt.show()

    except Exception as ex:
        print("[0].- Otras técnicas de Reducción de Dimensiones - Exception:", ex)
```

FASE III: Análisis de Resultados (Ataques DDoS)

Puesta en Servicio del Modelo Predictivo: Ajuste del Modelo, Validación y Verificación de los Datos

Este conjunto de datos incluye una variedad de ataques DDoS contemporáneos, principalmente de tipo **reflejado (reflective)** y **de inundación (flood)**, organizados en tres categorías principales:

1. Ataques de Inundación (Flood Attacks)

Técnicas que saturan directamente los recursos del objetivo mediante tráfico masivo:

Tipo	Características Clave	Impacto
SYN Flood	Envío masivo de paquetes SYN sin completar handshake TCP	Agota conexiones disponibles en servidores
UDP Flood	Inundación con datagramas UDP sin sesión	Consumo de ancho de banda y recursos de procesamiento

Tipo	Características Clave	Impacto
UDP-Lag	Variante pulsante que causa picos de latencia	Interrumpe servicios sensibles al tiempo (VoIP, gaming)

Mecanismo común: - Tráfico directo víctima-atacante - No requiere intermediarios - Fácil de generar pero también de filtrar

2. Ataques de Reflexión/Amplificación

Ataques que utilizan servidores legítimos como multiplicadores de tráfico:

Protocolos Explotados

Protocolo	Factor Amplificación	Puerto	Ejemplo Comando
DNS	28-54x	53	`dig ANY example.com @resolver`
NTP	556x	123	`ntpdc -n -c monlist`
LDAP	46-55x	389	Búsquedas sin autenticación
MSSQL	10-50x	1434	Consultas a servidores expuestos

Características clave: - IP spoofing obligatorio - Respuestas > solicitudes (ratio de amplificación) - Usan protocolos UDP principalmente

Top 3 Ataques por Impacto Potencial:

1. **NTP Monlist** (Mayor ratio amplificación)
2. **DNS Reflection** (Infraestructura global disponible)
3. **SSDP** (Dispositivos IoT vulnerables)

3. Otros Vectores de Ataque

WebDDoS (Layer 7)

- Ataques HTTP/S complejos
- Bajo volumen pero alta efectividad
- Mimickea comportamiento usuario legítimo
- Ejemplos: Slowloris, RUDY

PortScan (Reconocimiento)

- Pre-ataque para identificación de objetivos
- Falsos positivos comunes en detección
- Clasificado como "Unknown" en el modelo

```
def creacion_datos_entrenamiento():

    print("FASE I.- Adquisición de Datos")
    print('1.- [Data Collection] - Procesado multiproceso del conjunto de datos: \n {}'.forma

    ################################################
    ## Crear un pool de procesos
    pool = multiprocessing.Pool()

    ## Ejecutar los procesos en paralelo para procesar los archivos CSV
    results = [pool.apply_async(process_csv, args=(file_path)) for file_path in zip(file_patl

    ## Obtener los resultados de los procesos
    training_data = [result.get() for result in results]

    ## Cerrar el pool de procesos
    pool.close()
    pool.join()

    # Concatenar todos los DataFrames en uno solo
    print("1- [Data Collection] Lectura Completada. Fin pruebas de carga de Datasets 0605202

    # Concatenar los DataFrames resultantes
    training_data = pd.concat(training_data, ignore_index=True)
    training_data.head()

    print('2.- [Limpieza de Datos] - Limpieza del Conjunto de Datos')
```

```
    # Aplicar las operaciones de limpieza
    training_data = f_cleaning(training_data)
    print('2.10.- [Preprocesado] - Preprocesado de los Conjuntos de Datos')
    f_preprocesado(training_data)

    print('3.- [Datos de Entrenamiento] - Creación del Conjunto de datos: ',output_file)
    _df = pd.DataFrame(training_data[top_25_features])
    _df.to_csv(output_file, index=False)

    print("3.1- [Datos de Entrenamiento] Conjunto de Datos: {}\n - ATRIBUTOS: {}".format(outp
    return


#creacion_datos_entrenamiento()
```

```
#!pip install xgboost
```

```
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from typing import List

import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import LabelEncoder
from typing import List, Dict

# --- Variables globales y funciones de apoyo (se asumen definidas en el contexto) ---
# _testing_dataset_path
# _atributos
# best_model
global_attributes = ''

def _preprocess_validation_data(file_path: str, attributes: List[str]) -> pd.DataFrame:
    """
    Pre-procesa el conjunto de datos de validación para asegurar la compatibilidad con el mo
    """
```

```
    print(f"  -> Leyendo el conjunto de datos de validación desde: {file_path}")
    try:
        validation_data = pd.read_csv(file_path)
    except FileNotFoundError:
        raise FileNotFoundError(f"El archivo no se encuentra en la ruta: {file_path}")

    validation_data.columns = validation_data.columns.str.strip()
    print("  -> Validacion de Atributos")

    if 'Timestamp' in validation_data.columns:
        print("Convirtiendo la columna 'Timestamp' a formato numérico (timestamp).")
        validation_data['Timestamp'] = pd.to_datetime(validation_data['Timestamp']).astype('

    required_cols = list(attributes)
    if 'Label' not in attributes:
        required_cols.append('Label')

    if not all(col in validation_data.columns for col in required_cols):
        missing_cols = [col for col in required_cols if col not in validation_data.columns]
        raise ValueError(f"Columnas requeridas faltantes en el archivo de validación: {missi

    validation_data.replace([np.inf, -np.inf], np.nan, inplace=True)
    validation_data.dropna(inplace=True)

    label_encoder = LabelEncoder()
    categorical_cols = validation_data.select_dtypes(
        include=['object', 'category']).columns.drop('Label', errors='ignore')

    for col in categorical_cols:
        # Convertimos explícitamente la columna a tipo string antes de encodear
        validation_data[col] = validation_data[col].astype(str)
        validation_data[col] = label_encoder.fit_transform(validation_data[col])

    # Mapear las etiquetas
    label_mapping = {'BENIGN': 0}
    unique_labels = set(validation_data['Label'].unique()) - set(label_mapping.keys())
    for label in unique_labels:
        label_mapping[label] = 1

    if 'Label' in validation_data.columns:
        validation_data['Label'] = validation_data['Label'].map(label_mapping)
        validation_data.dropna(subset=['Label'], inplace=True)
```

```
    print("  -> Preparación de los datos de validación completada.")
    return validation_data

def validar_modelo(best_model: xgb.XGBClassifier, testing_dataset_path: str, atributos: List
    """
    Valida un modelo entrenado con un conjunto de datos de prueba, calculando métricas clave

    Args:
        best_model (xgb.XGBClassifier): El modelo XGBoost entrenado.
        testing_dataset_path (str): La ruta al archivo CSV del conjunto de datos de prueba.
        atributos (List[str]): Lista de las características esperadas por el modelo.
    """
    try:
        print('--- FASE DE VALIDACIÓN: Evaluando el modelo con datos de prueba ---')

        # 1. Pre-procesar los datos de prueba
        validation_data = _preprocess_validation_data(testing_dataset_path, atributos)

        # 2. Separar características (X) y variable objetivo (y)
        # La línea 'X_test = validation_data[atributos]' ya es correcta.
        # No necesitas el 'if atributos in validation_data.columns:'
        X_test = validation_data[atributos]
        y_test = validation_data['Label']

        print("  -> Datos de prueba listos para la validación.")

        # 3. Realizar predicciones
        y_pred = best_model.predict(X_test)

        # 4. Calcular y mostrar las métricas de rendimiento
        print('\n--- Resultados de la Validación ---')
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
        recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
        f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

        print(f"  Accuracy: {accuracy * 100:.2f}%")
        print(f"  Precision: {precision * 100:.2f}%")
        print(f"  Recall: {recall * 100:.2f}%")
        print(f"  F1-Score: {f1 * 100:.2f}%")

    except FileNotFoundError as e:
```

```
        print(f"  Error de archivo: {e}")
    except ValueError as e:
        print(f"  Error de datos: {e}")
    except Exception as e:
        print(f"  Error inesperado durante la validación: {e}")

# --- Ejemplo de uso en el flujo principal ---
# Suponiendo que 'best_model' ya ha sido entrenado y 'testing_dataset_path' está definido
# best_model = train_model_with_xgboost(X_train, y_train)
# validar_modelo(best_model, _testing_dataset_path, _atributos)

def train_model_with_xgboost(X_train: pd.DataFrame, y_train: pd.Series):
    """
    Entrena un modelo de clasificación con XGBoost.

    Args:
        X_train (pd.DataFrame): Datos de entrenamiento.
        y_train (pd.Series): Etiquetas de entrenamiento.

    Returns:
        xgb.XGBClassifier: El modelo XGBoost entrenado.
    """

    # 1. Reemplazar valores infinitos y nulos
    X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
    # XGBoost puede manejar valores nulos directamente, pero es una buena práctica
    # asegurarse de que los datos estén limpios.
    X_train.dropna(inplace=True)
    y_train = y_train.loc[X_train.index] # Asegurar que 'y_train' tenga el mismo índice

    # 2. Inicializar y entrenar el modelo XGBoost
    # Los hiperparámetros se pueden ajustar para optimizar el rendimiento.
    # Aquí se usan valores de referencia.
    xgboost_classifier = xgb.XGBClassifier(
        objective='binary:logistic',  # Para clasificación binaria
        eval_metric='logloss',
        use_label_encoder=False,
        n_estimators=100,             # Número de árboles
        learning_rate=0.1,            # Tasa de aprendizaje
        max_depth=5,                  # Profundidad máxima de cada árbol
        random_state=42
    )
```

```
    xgboost_classifier.fit(X_train, y_train)

    print("  El modelo XGBoost ha sido entrenado exitosamente.")

    return xgboost_classifier

# --- Ejemplo de uso en el flujo principal del código ---
# best_model = train_model_with_xgboost(X_train, y_train)
```

```
_file_paths = _asddos_dataset_paths
_index = 1
_model_path = './models/modelo_predictivo.pkl'

_training_dataset_path = 'C:/Datasets/Training/Syn.csv'
_testing_dataset_path  = 'C:/Datasets/Testing/Syn.csv'
#_testing_dataset_path  = 'C:/Datasets/Training/Syn.csv'
# --- Código principal ---
def main():
    print("FASE I.- Adquisición de Datos")
    print('1.- [Data Collection] - Carga y Pre-Procesado del Conjunto de Datos de Entrenamien

    # 1. Carga de datos
    #dataset_path = _file_paths[_index]
    df = f_process_read_csv(_training_dataset_path)
    df = f_cleaning(df)
    training_data = f_preprocesado(df)

    print('2.- [Data Engineering] - Exploracion del Conjunto de Datos')
    # Corregido: Se usaba 'trainig_data' en vez de 'training_data'
    f_show_histograms(training_data)
    f_exploracion_grafica_datos_boxplot(training_data)
    f_exploracion_grafica_analisis_clustering(training_data)

    print('3.- [Feature Engineering] - Seleccionando atributos')
    # Corregido: La llamada era incorrecta. f_make_correlation_matrix debe devolver el DataFr
    '''
    if 'Timestamp' in training_data.columns:
        training_data.drop('Timestamp', axis=1)
    '''

    training_data_reduced = f_make_correlation_analysis(training_data)
```

```
training_data_cleaned = training_data_reduced.copy()  # Se asume que ya está "limpio"

#f_exploracion_grafica_datos_boxplot(training_data_cleaned)

print('6.- [Confussion Matrix] - Imprimiendo matriz')
#f_make_confusion_matrix_with_knn(training_data_cleaned)

print('7.- [Modelado] - Comparativa de distintos modelos predictivos')
#benchmarking_clasificadores(training_data_cleaned)

# --- Preprocesamiento del modelo ---
# Identificar todas las columnas no numéricas para su codificación
categorical_columns = training_data_cleaned.select_dtypes(include=['object', 'category'])

# Aplicar LabelEncoder a cada columna categórica
label_encoder = LabelEncoder()
for col in categorical_columns:
    training_data_cleaned[col] = label_encoder.fit_transform(training_data_cleaned[col])

# Reemplazar valores infinitos y nulos para la limpieza final
training_data_cleaned.replace([np.inf, -np.inf], np.nan, inplace=True)
training_data_cleaned.dropna(inplace=True)

# 5. Modelado y entrenamiento del clasificador ID3
print('\n7.1- [Model Engineering] - Entrenamiento del Modelo Seleccionado: XGBOOST')

# Asegurar que la columna 'Label' exista
if 'Label' not in training_data_cleaned.columns:
    raise ValueError("La columna 'Label' no se encuentra en el DataFrame después del prep

# Separar características (X) y la variable objetivo (y)
X = training_data_cleaned.drop('Label', axis=1)
y = training_data_cleaned['Label']

# División de los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42
)



# Conversión de la columna de fecha y hora a un timestamp numérico
```

```
    if 'Timestamp' in X_train.columns:
        print("Convirtiendo la columna 'Timestamp' a formato numérico (timestamp).")
        X_train['Timestamp'] = pd.to_datetime(X_train['Timestamp']).astype('int64') // 10**9
    if 'Timestamp' in X_test.columns:
        print("Convirtiendo la columna 'Timestamp' a formato numérico (timestamp).")
        X_test['Timestamp'] = pd.to_datetime(X_test['Timestamp']).astype('int64') // 10**9


    _atributos = X_train.columns
    global_attributes = _atributos
    best_model = train_model_with_xgboost(X_train, y_train)
    validar_modelo(best_model, _testing_dataset_path, _atributos)
    print(f'\n7.2.-  Modelo generado exitosamente. Los atributos empleados son: {_atributos}

    # 6. Guardar el modelo entrenado
    try:
        with open(_model_path, 'wb') as archivo:
            pickle.dump(best_model, archivo)
        print(f"8.-  Modelo guardado exitosamente en: {_model_path}")
    except Exception as e:
        print(f"8.-  Error al guardar el modelo: {e}")

if __name__ == '__main__':
    main()
```

```
FASE I.- Adquisición de Datos
1.- [Data Collection] - Carga y Pre-Procesado del Conjunto de Datos de Entrenamiento
1.1- [Data Collection] Process-CSV: [C:/Datasets/Training/Syn.csv]

1.2.- [Data Collection] Finalización carga de Tiempo: 0.28 secs
 [C:/Datasets/Training/Syn.csv]
1.4.- [Data Collection] Finalizada la carga del conjunto de datos: [C:/Datasets/Training/Syn

1.5.- [Data Collection] - Clasificador:  ['Syn' 'BENIGN']
Etiqueta:  Syn 99.98 % del dataset
Etiqueta:  BENIGN 0.02 % del dataset
--- 2.1 [Data Cleaning] Iniciando proceso de limpieza de datos. ---
-> 2.1.1 Eliminación de filas duplicadas. Filas eliminadas: 0
-> 2.1.2 Manejando valores nulos, infinitos y tipos de datos.
-> 2.1.3 Procesando la columna 'Label'.
-> Presentación de resultados: Distribución de Etiquetas
    - Etiqueta: 'DDoS', Porcentaje: 99.98% del dataset
```

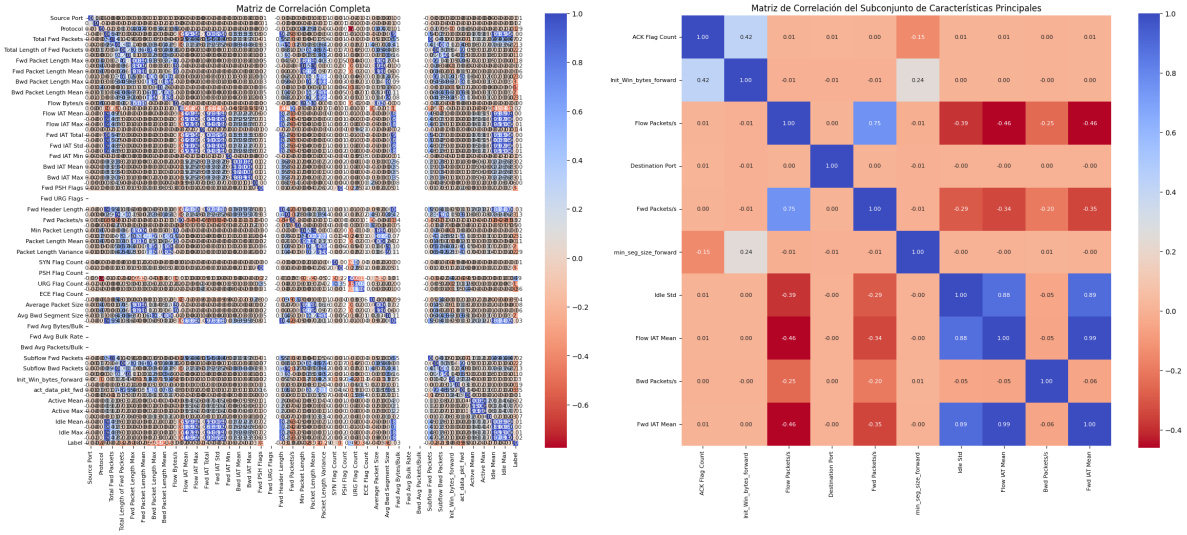
```
    - Etiqueta: 'BENIGN', Porcentaje: 0.02% del dataset
-> 2.1.4 Conversión de campos y normalización de IPs.
-> IPs corruptas recuperadas y normalizadas.
-> 2.1.5 Columnas eliminadas: ['Unnamed: 0', 'Inbound']
--- 2.1 [Data Cleaning] Proceso de limpieza finalizado. ---
3.1.- [Preprocesado] Iniciando el preprocesado de datos.
3.1.1.- [Preprocesado] Verificado: No hay valores nulos en el DataFrame.
3.1.2.- [Preprocesado] Verificado: No hay valores infinitos en el DataFrame.

3.1.3.- [Preprocesado] ATRIBUTOS DEL CONJUNTO DE DATOS PROCESADO:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1582681 entries, 0 to 1582680
Data columns (total 86 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   Flow ID                      1582681 non-null  object
 1   Source IP                    1582681 non-null  object
 2   Source Port                  1582681 non-null  float64
 3   Destination IP               1582681 non-null  object
 4   Destination Port             1582681 non-null  float64
 5   Protocol                     1582681 non-null  float64
 6   Timestamp                    1582681 non-null  datetime64[ns]
 7   Flow Duration                1582681 non-null  float64
 8   Total Fwd Packets            1582681 non-null  float64
 9   Total Backward Packets       1582681 non-null  float64
 10  Total Length of Fwd Packets  1582681 non-null  float64
 11  Total Length of Bwd Packets  1582681 non-null  float64
 12  Fwd Packet Length Max        1582681 non-null  float64
 13  Fwd Packet Length Min        1582681 non-null  float64
 14  Fwd Packet Length Mean       1582681 non-null  float64
 15  Fwd Packet Length Std        1582681 non-null  float64
 16  Bwd Packet Length Max        1582681 non-null  float64
 17  Bwd Packet Length Min        1582681 non-null  float64
 18  Bwd Packet Length Mean       1582681 non-null  float64
 19  Bwd Packet Length Std        1582681 non-null  float64
 20  Flow Bytes/s                 1582681 non-null  float64
 21  Flow Packets/s               1582681 non-null  float64
 22  Flow IAT Mean                1582681 non-null  float64
 23  Flow IAT Std                 1582681 non-null  float64
 24  Flow IAT Max                 1582681 non-null  float64
 25  Flow IAT Min                 1582681 non-null  float64
 26  Fwd IAT Total                1582681 non-null  float64
 27  Fwd IAT Mean                 1582681 non-null  float64
```

```
28  Fwd IAT Std                 1582681 non-null  float64
29  Fwd IAT Max                 1582681 non-null  float64
30  Fwd IAT Min                 1582681 non-null  float64
31  Bwd IAT Total               1582681 non-null  float64
32  Bwd IAT Mean                1582681 non-null  float64
33  Bwd IAT Std                 1582681 non-null  float64
34  Bwd IAT Max                 1582681 non-null  float64
35  Bwd IAT Min                 1582681 non-null  float64
36  Fwd PSH Flags               1582681 non-null  float64
37  Bwd PSH Flags               1582681 non-null  float64
38  Fwd URG Flags               1582681 non-null  float64
39  Bwd URG Flags               1582681 non-null  float64
40  Fwd Header Length           1582681 non-null  float64
41  Bwd Header Length           1582681 non-null  float64
42  Fwd Packets/s               1582681 non-null  float64
43  Bwd Packets/s               1582681 non-null  float64
44  Min Packet Length           1582681 non-null  float64
45  Max Packet Length           1582681 non-null  float64
46  Packet Length Mean          1582681 non-null  float64
47  Packet Length Std           1582681 non-null  float64
48  Packet Length Variance      1582681 non-null  float64
49  FIN Flag Count              1582681 non-null  float64
50  SYN Flag Count              1582681 non-null  float64
51  RST Flag Count              1582681 non-null  float64
52  PSH Flag Count              1582681 non-null  float64
53  ACK Flag Count              1582681 non-null  float64
54  URG Flag Count              1582681 non-null  float64
55  CWE Flag Count              1582681 non-null  float64
56  ECE Flag Count              1582681 non-null  float64
57  Down/Up Ratio               1582681 non-null  float64
58  Average Packet Size         1582681 non-null  float64
59  Avg Fwd Segment Size        1582681 non-null  float64
60  Avg Bwd Segment Size        1582681 non-null  float64
61  Fwd Header Length.1         1582681 non-null  float64
62  Fwd Avg Bytes/Bulk          1582681 non-null  float64
63  Fwd Avg Packets/Bulk        1582681 non-null  float64
64  Fwd Avg Bulk Rate           1582681 non-null  float64
65  Bwd Avg Bytes/Bulk          1582681 non-null  float64
66  Bwd Avg Packets/Bulk        1582681 non-null  float64
67  Bwd Avg Bulk Rate           1582681 non-null  float64
68  Subflow Fwd Packets         1582681 non-null  float64
69  Subflow Fwd Bytes           1582681 non-null  float64
70  Subflow Bwd Packets         1582681 non-null  float64
```

```
 71  Subflow Bwd Bytes            1582681 non-null  float64
 72  Init_Win_bytes_forward       1582681 non-null  float64
 73  Init_Win_bytes_backward      1582681 non-null  float64
 74  act_data_pkt_fwd             1582681 non-null  float64
 75  min_seg_size_forward         1582681 non-null  float64
 76  Active Mean                  1582681 non-null  float64
 77  Active Std                   1582681 non-null  float64
 78  Active Max                   1582681 non-null  float64
 79  Active Min                   1582681 non-null  float64
 80  Idle Mean                    1582681 non-null  float64
 81  Idle Std                     1582681 non-null  float64
 82  Idle Max                     1582681 non-null  float64
 83  Idle Min                     1582681 non-null  float64
 84  SimillarHTTP                 1582681 non-null  object
 85  Label                        1582681 non-null  object
dtypes: datetime64[ns](1), float64(80), object(5)
memory usage: 1.0+ GB

3.2.- [Preprocesado] ¡Preprocesado finalizado!
2.- [Data Engineering] - Exploracion del Conjunto de Datos
3.- [Feature Engineering] - Seleccionando atributos

---   Análisis de Matriz de Correlación ---
M.1.- [Análisis] Codificando la columna 'Label'...
M.1.1.- [Análisis] Etiquetas codificadas: {'BENIGN': 0, 'DDoS': 1}
M.2.- [Análisis] Calculando la matriz de correlación...

M.3.- [Análisis] Atributos más correlacionados con la etiqueta 'Label'...
  Correlaciones Positivas:
ACK Flag Count            0.494067
Init_Win_bytes_forward    0.027547
Flow Packets/s            0.019984
Destination Port          0.018275
Fwd Packets/s             0.015243
min_seg_size_forward      0.011576
Idle Std                  0.001474
Flow IAT Mean             0.001290
Bwd Packets/s             0.001089
Fwd IAT Mean             -0.000035
Name: Label, dtype: float64

  Correlaciones Negativas:
Bwd URG Flags          NaN
```

```
FIN Flag Count          NaN
PSH Flag Count          NaN
ECE Flag Count          NaN
Fwd Avg Bytes/Bulk      NaN
Fwd Avg Packets/Bulk    NaN
Fwd Avg Bulk Rate       NaN
Bwd Avg Bytes/Bulk      NaN
Bwd Avg Packets/Bulk    NaN
Bwd Avg Bulk Rate       NaN
Name: Label, dtype: float64
```

M.4.- [Visualización] Generando gráficos de la matriz de correlación.



M.5.- [Análisis] Proceso completado. Atributos principales seleccionados (15):
['Bwd Packet Length Min', 'ACK Flag Count', 'Bwd Packet Length Mean', 'Avg Bwd Segment Size'
6.- [Confussion Matrix] - Imprimiendo matriz
7.- [Modelado] - Comparativa de distintos modelos predictivos

```
7.1- [Model Engineering] - Entrenamiento del Modelo Seleccionado: XGBOOST
  El modelo XGBoost ha sido entrenado exitosamente.
--- FASE DE VALIDACIÓN: Evaluando el modelo con datos de prueba ---
  -> Leyendo el conjunto de datos de validación desde: C:/Datasets/Testing/Syn.csv
  -> Validacion de Atributos
Convirtiendo la columna 'Timestamp' a formato numérico (timestamp).
  -> Preparación de los datos de validación completada.
```

```
  -> Datos de prueba listos para la validación.

--- Resultados de la Validación ---
  Accuracy: 99.12%
  Precision: 98.25%
  Recall: 99.12%
  F1-Score: 98.69%

7.2.-  Modelo generado exitosamente. Los atributos empleados son: Index(['Fwd Packet Length
       'act_data_pkt_fwd', 'Fwd PSH Flags', 'Bwd Packet Length Std',
       'RST Flag Count', 'Protocol', 'CWE Flag Count', 'Packet Length Std',
       'Max Packet Length', 'Avg Bwd Segment Size', 'Bwd Packet Length Max',
       'URG Flag Count', 'Bwd Packet Length Min'],
      dtype='object')
8.-  Modelo guardado exitosamente en: ./models/modelo_predictivo.pkl
```

::: {.cell _kg_hide-output='true'}

```
import pandas as pd
import numpy as np
import pickle
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import LabelEncoder
from typing import List, Dict

# ----------------- VARIABLES GLOBALES CONFIGURABLES -----------------
# Atributos (columnas) que tu modelo espera para hacer predicciones.
# Es crucial que esta lista coincida con las características usadas en el entrenamiento.
_atributos: List[str] = ['Fwd Packet Length Std', 'ACK Flag Count', 'Bwd Packet Length Mean'
       'act_data_pkt_fwd', 'Fwd PSH Flags', 'Bwd Packet Length Std',
       'RST Flag Count', 'Protocol', 'CWE Flag Count', 'Packet Length Std',
       'Max Packet Length', 'Avg Bwd Segment Size', 'Bwd Packet Length Max',
       'URG Flag Count', 'Bwd Packet Length Min']
_path_production_dataset: str = "C:/Datasets/Produccion/Friday-WorkingHours-Afternoon-DDos.p
_model_path: str = './models/modelo_predictivo.pkl'


def _preprocess_production_data(file_path: str, attributes: List[str]) -> pd.DataFrame:
    """
    Pre-procesa un archivo CSV de producción para su validación.
    """
    print(f"  -> Leyendo el conjunto de datos de producción desde: {file_path}")
```

```
    try:
        production_data = pd.read_csv(file_path)
    except FileNotFoundError:
        raise FileNotFoundError(f"El archivo no se encuentra en la ruta: {file_path}")

    production_data.columns = production_data.columns.str.strip()
    print("  -> Limpiando y preparando los datos de producción.")

    # 1. Asegurar que las columnas del modelo existen en el DataFrame
    required_cols = attributes + ['Label']
    if not all(col in production_data.columns for col in required_cols):
        missing_cols = [col for col in required_cols if col not in production_data.columns]
        raise ValueError(f"Columnas requeridas faltantes en el archivo de producción: {missi

    # 2. Reemplazar valores infinitos y nulos para una limpieza robusta
    production_data.replace([np.inf, -np.inf], np.nan, inplace=True)
    production_data.dropna(inplace=True)

    # 3. Codificar automáticamente columnas no numéricas (si las hubiera)
    label_encoder = LabelEncoder()
    categorical_cols = production_data.select_dtypes(
        include=['object', 'category']).columns.drop('Label', errors='ignore')
    for col in categorical_cols:
        production_data[col] = label_encoder.fit_transform(production_data[col])

    # 4. Mapear las etiquetas para la clasificación binaria (0 = BENIGNO, 1 = ATAQUE)
    label_mapping = {'BENIGN': 0}
    # Mapear todas las demás etiquetas (ataques) a 1
    unique_labels = set(production_data['Label'].unique()) - set(label_mapping.keys())
    for label in unique_labels:
        label_mapping[label] = 1

    if 'Label' in production_data.columns:
        production_data['Label'] = production_data['Label'].map(label_mapping)
        production_data.dropna(subset=['Label'], inplace=True)

    print("  -> Preparación de datos de producción completada.")
    return production_data


def fase_produccion(name_of_model: str, path_production_dataset: str, atributos: List[str]):
    """
```

```
    Valida el modelo entrenado con un conjunto de datos de producción
    y detecta si hay ataques.

    Args:
        name_of_model (str): La ruta del archivo pickle del modelo.
        path_production_dataset (str): La ruta del archivo CSV con datos de producción.
        atributos (List[str]): Lista de las características esperadas por el modelo.
    """
    try:
        print('\n--- 11. [FASE DE PRODUCCIÓN] - Análisis del tráfico en tiempo real ---')

        # 1. Pre-procesar los datos de producción
        production_data = _preprocess_production_data(path_production_dataset, atributos)

        # 2. Separar características y variable objetivo
        X_production = production_data[atributos]
        y_production = production_data['Label']

        print("11.2- [Data Collection] Conversión de tipos y limpieza completada.")

        # 3. Cargar el modelo desde el archivo
        with open(name_of_model, 'rb') as archivo:
            best_model = pickle.load(archivo)

        print('11.3.- [Producción] - Realizando predicciones sobre el conjunto de producción
        produccion_predicciones = best_model.predict(X_production)

        # 4. Mostrar el resultado de la detección
        num_ataques_detectados = np.sum(produccion_predicciones == 1)
        if num_ataques_detectados > 0:
            print(f"  ¡ALERTA! Se han detectado {num_ataques_detectados} trazas")
            print(f" -- Patrones, de un posible Ataque DDoS")
            print("-----------------------------------------------------------------------")

        else:
            print("\n  No se han detectado patrones de Ataque DDoS )
            print("-----------------------------------------------------------------------")

        # 5. Calcular métricas de rendimiento (opcional en producción, útil para validación)
        print('\n11.4.- [Producción] - Calculando métricas de rendimiento')
        accuracy_production = accuracy_score(y_production, produccion_predicciones)
        precision_production = precision_score(y_production, produccion_predicciones,
```

```
                                                    average='weighted', zero_division=0)
        recall_production = recall_score(y_production, produccion_predicciones,
                                         average='weighted', zero_division=0)
        f1_production = f1_score(y_production, produccion_predicciones,
                                 average='weighted', zero_division=0)

        print("11.5.- [Producción] Métricas de rendimiento del modelo:")
        print(f"Accuracy: {accuracy_production * 100:.2f}%")
        print(f"Precision: {precision_production * 100:.2f}%")
        print(f"Recall: {recall_production * 100:.2f}%")
        print(f"F1-Score: {f1_production * 100:.2f}%")

    except FileNotFoundError as e:
        print(f"  Error de archivo: {e}")
    except ValueError as e:
        print(f"  Error de datos: {e}")
    except Exception as e:
        print(f"  Error inesperado durante la fase de producción: {e}")


if __name__ == '__main__':
    fase_produccion(_model_path, _path_production_dataset, _atributos)
```

```
--- 11. [FASE DE PRODUCCIÓN] - Análisis del tráfico en tiempo real ---
  -> Leyendo el conjunto de datos de producción desde: C:/Datasets/Produccion/Friday-Working
  -> Limpiando y preparando los datos de producción.
  -> Preparación de datos de producción completada.
11.2- [Data Collection] Conversión de tipos y limpieza completada.
11.3.- [Producción] - Realizando predicciones sobre el conjunto de producción.

  ¡ALERTA! Se han detectado 225711 instancias de ataques en el tráfico de red.
-----------------------------------------------------------------------

11.4.- [Producción] - Calculando métricas de rendimiento
11.5.- [Producción] Métricas de rendimiento del modelo:
Accuracy: 56.72%
Precision: 32.17%
Recall: 56.72%
F1-Score: 41.06%

:::
```

Otras Referencias:

En el desarrollo del presente Cuaderno que conforma una PoC centrada en explorar distintos metodos y algoritmos previamente desarrollado en otro tipo de contextos, como metodos empleados en la detección de Fraude Bancario. También se han utilizado distintos Modelos LLMs y AI como Copilot o Gemini, en la mejora y optimización del código implementado. Algunas de estas referencias se muestran a continuación:

Credit Card Fraud Detection using Machine Learning Algorithms by Vaishnavi NathDornadula

Anonymized credit card transactions labeled as fraudulent or genuine

Credit Card Fraud Detection using Pipeling and Ensemble Learning

Credit Fraud || Dealing with Imbalanced Datasets by Janio Martinez Bachmann · (Kaggle Notebook)

Hands on Machine Learning with Scikit-Learn & TensorFlow by Aurélien Géron (O'Reilly). CopyRight 2017 Aurélien Géron

Machine Learning - Over-& Undersampling - Python/ Scikit/ Scikit-Imblearn by CodingManiac

auprc, 5-fold c-v, and resampling methods by Jeremy Lane (Kaggle Notebook)