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: 01/08/2025

Version: R200

Descripcion: Comparativa de Modelos de Aprendizaje Automático (AI/ML)

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:

Resumen

El presente cuaderno proporciona soporte a la comparativa desarrollada en el TFM centrado en el desarrollo de Sistemas de Detección y Mitigación de Amenazas para Smart Grids (SG) utilizando tecnologías avanzadas basadas en Inteligencia Artificial (IA). Desde técnicas de Aprendizaje Automático (ML) hasta soluciones basadas en la utilización de Metaheurísticas o Sistemas Autónomos y Multiagente (MAS), se exploran diferentes métodos estableciendo comparativas, que permitan identificar los métodos que ofrecen mejores prestaciones en la detección de anomalías en el tráfico de red.

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

```
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
_asddos_dataset_paths = [
    'C:/DataSets/asddos2024/Training/Training_Exp_Attack_TCP.csv',
    'C:/DataSets/asddos2024/Training/Training_Exp_Attack_UDP.csv',
    'C:/DataSets/asddos2024/Training/Training_Ref_Attack_MIXED.csv',
    'C:/DataSets/asddos2024/Training/Training_Ref_Attack_TCP.csv',
    'C:/DataSets/asddos2024/Training/Training_Ref_Attack_UDP.csv'
]

# Verificar la existencia de cada archivo
```

```python
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-2019 NO ESTA DISPONIBLE EN SU TOTALIDAD")

# Verificar el directorio
dirname = 'C:/DataSets/asddos2024/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 NO TIENE PERMISOS PARA ACCEDER AL CONJUNTO DE
```

```
--- [IMPORTANTE] Verificando la existencia de los archivos ---
El archivo existe: C:/DataSets/asddos2024/Training/Training_Exp_Attack_TCP.csv
El archivo existe: C:/DataSets/asddos2024/Training/Training_Exp_Attack_UDP.csv
El archivo existe: C:/DataSets/asddos2024/Training/Training_Ref_Attack_MIXED.csv
El archivo existe: C:/DataSets/asddos2024/Training/Training_Ref_Attack_TCP.csv
El archivo existe: C:/DataSets/asddos2024/Training/Training_Ref_Attack_UDP.csv
```

```
 [CORRECTO] EL CONJUNTO DE DATOS ASDDOS-2019 ESTA DISPONIBLE

--- [IMPORTANTE] Verificando el directorio: C:/DataSets/asddos2024/Training ---
El directorio 'C:/DataSets/asddos2024/Training' existe.
```

Sistemas basados en IA en Ciberseguridad de Infraestructuras Críticas

- Revision: R200

Introducción

El presente trabajo se centra en el desarrollo de un sistema de detección de intrusiones utilizando técnicas de Machine Learning para proteger infraestructuras críticas. En un contexto donde la digitalización avanza, la seguridad de estos sistemas es de vital importancia para garantizar el funcionamiento de servicios esenciales como la energía, el transporte o la sanidad.

El objetivo principal de este proyecto es analizar un conjunto de datos de tráfico de red para identificar patrones de ataques cibernéticos, como la inyección SQL y los ataques de denegación de servicio (DDoS), y entrenar varios modelos de aprendizaje automático para clasificarlos de manera efectiva. La meta es comparar el rendimiento de diferentes algoritmos y determinar cuál ofrece la mejor capacidad de detección para este tipo de amenazas.

La metodología a seguir incluye: 1. **Limpieza y Preprocesamiento de Datos:** Eliminación de datos irrelevantes para el entrenamiento de los modelos. 2. **Análisis Exploratorio de Datos (EDA):** Visualización y comprensión de la distribución de los datos y sus relaciones. 3. **Ingeniería de Características:** Creación de nuevas variables que aporten mayor valor predictivo. 4. **Modelado y Evaluación:** Entrenamiento de varios clasificadores y comparación de su rendimiento mediante métricas clave.

A continuación, se detalla el desarrollo práctico del proyecto.

FASE I: Exploración del Conjunto de Datos

Ingeniería de Datos

La Ingeniería de Datos es la disciplina que se encarga de diseñar, construir y mantener sistemas y arquitecturas para la recolección, almacenamiento y procesamiento de grandes volúmenes de datos. Su objetivo principal es asegurar que los datos estén disponibles, limpios y accesibles para los científicos de datos, analistas y aplicaciones de negocio. Esto incluye la creación de pipelines de datos y la automatización de flujos de trabajo para manejar la complejidad y el crecimiento constante de la información.

- Referencia Bibliográfica D. C. S. (2018). The Modern Data Stack: The Data Engineer's Handbook. O'Reilly Media.

```
import concurrent.futures
import csv
import datetime
import ipaddress
import math
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import multiprocessing
import numpy as np
import os
import pandas as pd
import pickle
import psutil
import random
import seaborn as sns
import sys
import time
import tracemalloc
```

```
import warnings
from matplotlib.gridspec import GridSpec
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, StackingClassifie
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
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
from tabulate import tabulate

import logging
# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)
```

1.- Adquisición de Datos y Elaboración de Datos de Entrenamiento

La primera tarea a desarrollar en la PoC es la adquisición de los datos del Conjunto de Datos disponible CICDDoS2019, previamente referenciado, en el almacenamiento local. El objetivo es obtener un Conjunto de Datos perfectamente estructrado para el desarrollo de la PoC, mediante un conjunto de Datos de Entrenamiento, estructurado en distintos DataSets

```
from datetime import datetime

# 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','Label'
]

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']

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']
                     # Eliminadas no aportan demasiada información 'Inbound',

# Lista de Atributos mas significativos para elaborar mi propio conjunto de datos
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'] #'Inbound'

Main_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']

# 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'
]


top_50_features = selected_features
top_25_features = reduced_set_of_features_II
extended_set_of_features = ['Timestamp','Source IP','Destination IP',
                            'Source Port', 'Destination Port','Protocol']

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

```
2025-08-09 09:12:43,501 - __main__ - INFO - FASE I.- [Exploracion Conjunto de Datos] - Atributos del Conjunto de Datos
```

```
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'

logger.info(f"FASE I.- [EXPLORACION] - Conjuntos de Datos por Explorar de CIC-DDoS2019")
```

```
2025-08-09 09:12:43,513 - __main__ - INFO - FASE I.- [EXPLORACION] - Conjuntos de Datos por Explorar de CIC-DDoS2019
```

Calidad del Modelo de Datos

La calidad de los datos es fundamental en el análisis de datos y el aprendizaje automático. Datos de alta calidad garantizan que los modelos sean precisos, confiables y tomen decisiones correctas. Los datos de mala calidad, en cambio, pueden llevar a resultados sesgados y a una toma de decisiones errónea, sin importar cuán sofisticado sea el algoritmo. La importancia de la calidad de los datos es reconocida a nivel internacional por estándares como la serie ISO/IEC 25012, que define las características de la calidad del producto de datos.

- Referencia International Organization for Standardization. (2008). ISO/IEC 25012:2008. Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Data quality model.

```
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


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 (debe contener '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) -> float:
    """
    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)}% del dataset")

        # 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
```

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

def radar_plot_with_table(metricas_df1: pd.DataFrame, metricas_df2: pd.DataFrame, df1_name: str, df2_name: str):
    """
    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()


# Llamada a la función para crear el gráfico de radar y la tabla
# Calcular las métricas para ambos conjuntos de datos
'''
input_file1 = "C:/Datasets/asddos2024/Training/Training_Exp_Attack_TCP.csv"
df1 = f_process_read_csv(input_file1)
input_file2 = "C:/Datasets/asddos2024/Training/Training_Exp_Attack_UDP.csv"
df2 = f_process_read_csv(input_file2)

metricas_df1 = f_metricas_calidad_modelo_datos(df1)
metricas_df2 = f_metricas_calidad_modelo_datos(df2)
radar_plot_with_table(metricas_df1, metricas_df2, df1_name='Flood TCP', df2_name='Flood UDP')
'''
```

```
'\ninput_file1 = "C:/Datasets/asddos2024/Training/Training_Exp_Attack_TCP.csv"\ndf1 = f_process_read_csv(input_file1)\
```

1.6 Estimación comparativa de métricas ambientales y recursos consumidos por el sistema

Justificación y Bibliografía

1. Tiempo de CPU: 0.000000004 kg CO2e/segundo El factor de emisión para el tiempo de CPU se basa en la energía consumida por los procesadores y la cantidad de CO2e emitida por unidad de energía consumida. Estudios como el de Barroso et al. (2013) y el informe de sostenibilidad de Google (2020) proporcionan datos sobre la eficiencia energética de los centros de datos y la huella de carbono asociada.

- Barroso, L.A., Clidaras, J., & Hölzle, U. (2013). The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool Publishers. Google Sustainability Report (2020). Carbon-Free Energy for Google Data Centers.

2. Memoria RSS y RAM: 0.00000000015 kg CO2e/MB El consumo de energía de la memoria RAM se ha estudiado en varios contextos. Según el trabajo de Meisner et al. (2009) y Gholkar & Xu (2013), el consumo energético de la RAM es significativo, y su huella de carbono puede estimarse a partir del consumo energético.

- Meisner, D., Gold, B.T., & Wenisch, T.F. (2009). PowerNap: Eliminating Server Idle Power. ACM SIGARCH Computer Architecture News. Gholkar, A., & Xu, Y. (2013). DRAM Power Consumption: A Significant Contributor to Data Center Power. IEEE.

3. Lectura y Escritura de Disco: 0.000000002 y 0.000000005 kg CO2e/MB Las operaciones de I/O de disco tienen un impacto en el consumo energético de los centros de datos. Liu et al. (2018) y otros estudios han medido el consumo energético de diferentes tipos de almacenamiento y han proporcionado factores de conversión de energía a emisiones de CO2e.

- Liu, Z., Zhang, M., & Meng, X. (2018). Measuring Disk Energy Consumption. IEEE Transactions on Computers.
- Pinheiro, E., Bianchini, R., Carrera, E.V., & Heath, T. (2001). Dynamic Cluster Reconfiguration for Power and Performance. ACM SIGOPS Operating Systems Review.

Referencias Bibliográficas

- Barroso, L.A., Clidaras, J., & Hölzle, U. (2013). The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool Publishers. Google Sustainability Report (2020). Carbon-Free Energy for Google Data Centers.
- Meisner, D., Gold, B.T., & Wenisch, T.F. (2009). PowerNap: Eliminating Server Idle Power. ACM SIGARCH Computer Architecture News.
- Gholkar, A., & Xu, Y. (2013). DRAM Power Consumption: A Significant Contributor to Data Center Power. IEEE.
- Liu, Z., Zhang, M., & Meng, X. (2018). Measuring Disk Energy Consumption. IEEE Transactions on Computers.
- Pinheiro, E., Bianchini, R., Carrera, E.V., & Heath, T. (2001). Dynamic Cluster Reconfiguration for Power and Performance. ACM SIGOPS Operating Systems Review.
- Koomey, J.G. (2011). Growth in Data Center Electricity Use 2005 to 2010. Analytics Press.

- Koomey, J.G. (2008). Worldwide electricity used in data centers. Environmental Research Letters.

```
import time
import tracemalloc
import psutil
import os
from typing import Callable, Dict, Any, Union

# --- Factores de Emisión y Ponderación (Variables globales para evitar redefinición) ---
# Se justifican con la bibliografía proporcionada.

# Factores de ponderación para el cálculo del CEEC.
# Estos valores determinan la importancia relativa de cada recurso.
FACTORES_PONDERACION = {
    "tiempo_cpu": 0.7,
    "memoria_pico": 0.3,
    "lectura_disco": 0.3,
    "escritura_disco": 0.3
}

# Factores de emisión de CO2e por unidad de recurso consumido.
# Basados en la bibliografía proporcionada.
FACTORES_EMISION = {
    "tiempo_cpu": 0.000000004,     # kg CO2e/segundo
    "memoria_pico": 0.00000000015, # kg CO2e/MB
    "lectura_disco": 0.000000002,  # kg CO2e/MB
    "escritura_disco": 0.000000005  # kg CO2e/MB
}


# --- Funciones de Cálculo de Métricas ---
```

```python
def calcular_ceec(consumo_recursos: Dict[str, Union[float, int]]) -> float:
    """
    Calcula la Huella de Carbono del Equipo Computacional (CEEC).

    Multiplica el consumo de cada recurso por su factor de ponderación y de emisión.

    Args:
        consumo_recursos (Dict[str, Union[float, int]]): Diccionario con el
            consumo medido de cada recurso (ej. 'tiempo_cpu', 'memoria_pico').

    Returns:
        float: El valor total del CEEC en kg de CO2e.
    """
    ceec = 0.0
    for recurso, consumo in consumo_recursos.items():
        ponderacion = FACTORES_PONDERACION.get(recurso, 1)
        factor_emision = FACTORES_EMISION.get(recurso, 0)
        ceec += consumo * ponderacion * factor_emision
    return ceec


def ejecutar_y_calcular_metricas(nombre_funcion: str, descripcion: str, funcion_a_ejecutar: Callable, *args: Any):
    """
    Ejecuta una función y mide su consumo de recursos y huella ambiental.

    Mide el tiempo de ejecución, el uso de memoria (pico), el uso de CPU y la
    actividad de disco para luego calcular la Huella de Carbono (CEEC).

    Args:
        nombre_funcion (str): Nombre de la función que se está ejecutando.
```

```
    descripcion (str): Breve descripción de la tarea de la función.
    funcion_a_ejecutar (Callable): La función a ejecutar.
    *args (Any): Argumentos posicionales para la función.
"""
print(f"--- Iniciando medición para: '{nombre_funcion}' ---")

# Iniciar la medición de memoria y obtener la información del proceso actual.
tracemalloc.start()
process = psutil.Process(os.getpid())

# Medir el uso inicial de disco (lectura/escritura).
io_counters_start = process.io_counters()

# Medir el tiempo de inicio.
t1 = time.perf_counter_ns()

# --- Ejecutar la función ---
funcion_a_ejecutar(*args)

# --- Finalizar mediciones ---
t2 = time.perf_counter_ns()
io_counters_end = process.io_counters()

# Obtener el uso de memoria pico.
# El valor 'small_memory' representa la memoria actual, 'peak_memory' el valor máximo.
small_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()

# --- Calcular métricas de recursos ---
# Tiempo de ejecución en segundos.
exec_time_s = (t2 - t1) / 1e9
```

```python
# Uso de CPU. `cpu_percent` mide el porcentaje de uso desde la última llamada.
# Se llama con un intervalo para obtener un valor representativo.
cpu_usage = process.cpu_percent(interval=1)

# Uso de memoria pico en MB. Se considera el pico como la métrica más relevante.
mem_peak_mb = peak_memory / (1024 * 1024)

# Uso de disco en MB. Se calcula la diferencia entre el inicio y el fin.
disk_usage_read_mb = (io_counters_end.read_bytes - io_counters_start.read_bytes) / (1024 * 1024)
disk_usage_write_mb = (io_counters_end.write_bytes - io_counters_start.write_bytes) / (1024 * 1024)

# --- Consumo de recursos para el cálculo del CEEC ---
consumo_recursos = {
    "tiempo_cpu": exec_time_s,
    "memoria_pico": mem_peak_mb,
    "lectura_disco": disk_usage_read_mb,
    "escritura_disco": disk_usage_write_mb
}

# Cálculo del CEEC
ceec = calcular_ceec(consumo_recursos)

# --- Mostrar resumen de las métricas ---
print("\n--- Resumen de Métricas de Recursos y Huella Ambiental ---")
print(f"Función: {nombre_funcion}")
print(f"Descripción: {descripcion}")
print("-" * 50)
print(f"Tiempo de ejecución: {exec_time_s:.2f} segundos")
print(f"Pico de memoria (tracemalloc): {mem_peak_mb:.2f} MB")
print(f"Uso de CPU: {cpu_usage:.2f}%")
```

```
    print(f"Lectura de disco: {disk_usage_read_mb:.2f} MB")
    print(f"Escritura de disco: {disk_usage_write_mb:.2f} MB")
    print(f"Huella ambiental (CEEC): {ceec:.10f} kg CO2e")
    print("-" * 50)
```

```
import pandas as pd
from typing import Dict, Any

def compare_df_formats(df1: pd.DataFrame, df2: pd.DataFrame) -> Dict[str, Any]:
    """
    Compara el formato de dos DataFrames de pandas, verificando si tienen
    las mismas columnas y los mismos tipos de datos.

    Args:
        df1 (pd.DataFrame): El primer DataFrame para comparar.
        df2 (pd.DataFrame): El segundo DataFrame para comparar.

    Returns:
        Dict[str, Any]: Un diccionario que resume la comparación, incluyendo:
            - 'same_columns' (bool): True si las columnas son idénticas, False en caso contrario.
            - 'df1_only_columns' (list): Columnas presentes solo en df1.
            - 'df2_only_columns' (list): Columnas presentes solo en df2.
            - 'same_dtypes' (bool): True si los tipos de datos de las columnas compartidas son idénticos.
            - 'dtype_mismatches' (dict): Un diccionario con las columnas y sus tipos de datos
                                         cuando hay una discrepancia.
    """
    logger.info("Iniciando la comparación de formatos de los DataFrames...")

    format_comparison = {}

    # Obtener los conjuntos de columnas para una comparación eficiente.
```

```
    columns1 = set(df1.columns)
    columns2 = set(df2.columns)

    # 1. Comparación de columnas.
    df1_only_columns = list(columns1 - columns2)
    df2_only_columns = list(columns2 - columns1)

    format_comparison['same_columns'] = not df1_only_columns and not df2_only_columns
    format_comparison['df1_only_columns'] = df1_only_columns
    format_comparison['df2_only_columns'] = df2_only_columns

    # 2. Comparación de tipos de datos solo para las columnas compartidas.
    common_columns = list(columns1.intersection(columns2))
    dtype_mismatches = {}

    for col in common_columns:
        if df1[col].dtype != df2[col].dtype:
            dtype_mismatches[col] = {
                'df1_dtype': str(df1[col].dtype),
                'df2_dtype': str(df2[col].dtype)
            }

    format_comparison['same_dtypes'] = not dtype_mismatches
    format_comparison['dtype_mismatches'] = dtype_mismatches

    # Mostrar un resumen de los resultados
    logger.info("\n--- Resumen de la Comparación de Formatos ---")
    if format_comparison['same_columns'] and format_comparison['same_dtypes']:
        logger.info("El formato de ambos DataFrames es idéntico.")
    else:
        if not format_comparison['same_columns']:
```

```
            logger.info("Discrepancia en las columnas:")
            if format_comparison['df1_only_columns']:
                logger.info(f"  - Columnas en df1 pero no en df2: {format_comparison['df1_only_columns']}")
            if format_comparison['df2_only_columns']:
                logger.info(f"  - Columnas en df2 pero no en df1: {format_comparison['df2_only_columns']}")
        else:
            logger.info("Las columnas de ambos DataFrames son idénticas.")

        if not format_comparison['same_dtypes']:
            print("\nDiscrepancia en los tipos de datos:")
            for col, dtypes in format_comparison['dtype_mismatches'].items():
                print(f"  - Columna '{col}': df1 tiene {dtypes['df1_dtype']}, df2 tiene {dtypes['df2_dtype']}")
        else:
            print("\nLos tipos de datos de las columnas compartidas son idénticos.")

    return format_comparison
```

1.7.- Balanceado: Sobremuestreo de Datasets

El balanceo de datasets es una etapa crítica en la **Ingeniería de Datos** y el **aprendizaje automático**, especialmente cuando se trabaja con datos desequilibrados. Un dataset se considera desequilibrado cuando la distribución de clases en la variable objetivo no es uniforme, es decir, una clase (mayoritaria) tiene un número significativamente mayor de instancias que otra(s) clase(s) (minoritaria). En contextos como la **detección de intrusiones de red**, esto es común, ya que los ataques suelen ser mucho menos frecuentes que el tráfico benigno.

El desequilibrio puede llevar a que los modelos de clasificación se sesguen hacia la clase mayoritaria. Como resultado, el modelo podría obtener una alta precisión general simplemente clasificando todas las instancias como la clase

mayoritaria, pero fallaría en detectar la clase minoritaria, que a menudo es la más importante (ej. un fraude, un ataque DDoS, etc.).

La función implementada aborda este problema mediante una estrategia híbrida de **sobremuestreo (*oversampling*) y submuestreo (*undersampling*)**:

1. **Sobremuestreo con SMOTE**: El código utiliza la técnica **SMOTE (Synthetic Minority Over-sampling Technique)**. En lugar de simplemente duplicar las instancias existentes de la clase minoritaria, SMOTE genera nuevas instancias sintéticas a lo largo de las líneas que conectan a un punto con sus vecinos más cercanos. Esto crea un conjunto de datos más rico y diverso para la clase minoritaria, reduciendo el sesgo.
2. **Filtrado y Submuestreo**: Después de aplicar SMOTE, la función realiza un filtrado estadístico para eliminar posibles *outliers* sintéticos que puedan haber sido generados y luego submuestrea el conjunto de datos para alcanzar un tamaño objetivo y una proporción de clases deseada.

Este enfoque combinado no solo equilibra el dataset, sino que también controla su tamaño, lo que mejora la eficiencia computacional y la calidad del conjunto de datos de entrenamiento. El resultado es un modelo más robusto, capaz de detectar tanto la clase mayoritaria como la minoritaria con mayor precisión.

Referencia Bibliográfica

- **Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321-357.**


```
#!pip install pandas scikit-learn imbalanced-learn
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
import warnings
from typing import Tuple, Dict, List
```

```python
def f_balancear_dataset_oversampling(
    df: pd.DataFrame,
    # Se utiliza un diccionario para mayor flexibilidad en los ratios de cualquier número de clases
    ratios: Dict[str, float],
    target_samples: int = 30000,
    random_state: int = 42
) -> pd.DataFrame:
    """
    Balancea un conjunto de datos desequilibrado con SMOTE para oversampling
    y luego submuestrea para alcanzar un tamaño objetivo con proporciones
    de clases específicas.

    Args:
        df (pd.DataFrame): El DataFrame de entrada. Debe contener una columna 'Label'.
        ratios (Dict[str, float]): Un diccionario con las proporciones deseadas para cada clase.
                                   Ejemplo: {'BENIGN': 0.5, 'DDoS': 0.5}.
                                   La suma de los valores debe ser 1.0.
        target_samples (int): El número total de muestras en el dataset final.
        random_state (int): Semilla para la reproducibilidad de los resultados.

    Returns:
        pd.DataFrame: Un nuevo DataFrame balanceado con los datos sintéticos filtrados.

    Raises:
        ValueError: Si las validaciones iniciales fallan (columnas faltantes o ratios incorrectos).
    """
    logger.info("--- [Balanceo] Iniciando el proceso de balanceo del conjunto de datos ---")

    # 1. Validación inicial de entrada
```

```
    if 'Label' not in df.columns:
        raise ValueError("Error: El DataFrame de entrada no contiene la columna 'Label'.")

    # Se valida que la suma de los ratios sea 1.0
    if not np.isclose(sum(ratios.values()), 1.0):
        raise ValueError("Error: La suma de los valores en el diccionario 'ratios' debe ser 1.0.")

    try:
        initial_rows, initial_cols = df.shape
        logger.info(f"Dimensiones iniciales del DataFrame: {initial_rows} x {initial_cols}")

        # 2. Separación de características y etiquetas
        X = df.drop('Label', axis=1)
        y = df['Label']

        # 3. Mostrar balance inicial y preparar codificador
        logger.info("\n[Balanceo] Balance inicial del conjunto de datos:")
        label_counts = y.value_counts(normalize=True) * 100
        for label, percentage in label_counts.items():
            print(f"  - Etiqueta '{label}': {percentage:.2f}%")

        le = LabelEncoder()
        y_encoded = le.fit_transform(y)

        # 4. Aplicación de SMOTE (se usa la estrategia 'auto' para oversamplear todas las minoritarias)
        logger.info("\n[Balanceo] Aplicando oversampling con SMOTE para igualar las clases.")
        smote_sampler = SMOTE(sampling_strategy='auto', random_state=random_state)
        X_resampled, y_resampled_encoded = smote_sampler.fit_resample(X, y_encoded)

        df_resampled = pd.DataFrame(X_resampled, columns=X.columns)
        df_resampled['Label'] = le.inverse_transform(y_resampled_encoded)
```

```
        logger.info(f"Oversampling con SMOTE finalizado. Nuevas dimensiones: {len(df_resampled)} x {len(df_resampled.c

        # 5. Filtrado de datos atípicos sintéticos
        logger.info("\n[Balanceo] Filtrando datos atípicos sintéticos.")
        outlier_threshold = 3.0

        numeric_features = df_resampled.select_dtypes(include=['float64', 'int64']).columns.drop('Label', errors='igno

        # Se calcula la media y la desviación estándar solo de los datos originales para evitar sesgos
        original_stats = df.drop('Label', axis=1).describe()

        for feature in numeric_features:
            if feature in original_stats.columns:
                mean = original_stats.loc['mean', feature]
                std = original_stats.loc['std', feature]
                lower_bound = mean - (std * outlier_threshold)
                upper_bound = mean + (std * outlier_threshold)

                df_resampled = df_resampled.loc[
                    (df_resampled[feature] >= lower_bound) & (df_resampled[feature] <= upper_bound)
                ]
            else:
                logger.info(f"  - Advertencia: La característica '{feature}' no se encontró en el DataFrame original.'

        # 6. Remuestreo final para alcanzar el tamaño objetivo
        logger.info(f"\n[Balanceo] Remuestreando para un tamaño final de {target_samples} muestras.")

        dfs_to_concat = []
        for label, ratio in ratios.items():
            data = df_resampled[df_resampled['Label'] == label]
            num_samples = int(target_samples * ratio)
```

```
            # Se usa `min` para no solicitar más muestras de las que existen tras el filtrado
            num_samples = min(num_samples, len(data))

            # Realizar el muestreo aleatorio
            sampled_data = data.sample(n=num_samples, replace=(len(data) < num_samples), random_state=random_state)
            dfs_to_concat.append(sampled_data)

        # Concatenar y mezclar el conjunto de datos final
        df_balanced = pd.concat(dfs_to_concat).sample(frac=1, random_state=random_state).reset_index(drop=True)

        # 7. Mostrar el balance final
        logger.info("\n[Balanceo] Balance final del conjunto de datos:")
        label_counts_final = df_balanced['Label'].value_counts(normalize=True) * 100
        for label, percentage in label_counts_final.items():
            print(f"  - Etiqueta '{label}': {percentage:.2f}%")

        final_rows, final_cols = df_balanced.shape
        logger.info(f"Proceso finalizado. Dimensiones del DataFrame de salida: {final_rows} x {final_cols}")

        return df_balanced

    except Exception as ex:
        logger.info(f"\n[Balanceo]   Excepción: {ex}")
        raise

# --- Ejemplo de uso ---
'''
if __name__ == '__main__':
```

```
    input_file = "C:/Datasets/asddos2024/Training/Training_Exp_Attack_TCP.csv"
    df = f_process_read_csv(input_file)
    df = f_cleaning(df)
    df = f_preprocesado(df)
    logger.info("--- DataFrame Original ---")
    print(df['Label'].value_counts())
    print("-" * 30)
    # Definir los ratios deseados para el balanceo
    ratios_deseados = {'BENIGN': 0.7, 'DDoS': 0.3}


    # Llamar a la función mejorada de balanceo
    try:
        df_balanceado = f_balancear_dataset_oversampling(
            df=df[PCA_25_Attributes],
            ratios=ratios_deseados,
            target_samples=500000
        )
        logger.info("\n--- DataFrame Balanceado ---")
        print(df_balanceado['Label'].value_counts())
    except ValueError as e:
        logger.info(e)
'''
```

```
'\nif __name__ == \'__main__\':\n    \n    \n    input_file = "C:/Datasets/asddos2024/Training/Training_Exp_Attack_TCP
```

1.8 Exploración Gráfica del Conjunto de Datos

Funciones y metodos de exploración del conunto de datos

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

def f_exploracion_grafica_datos(df: pd.DataFrame) -> NoReturn:
    """
    Realiza una exploración gráfica de los datos para analizar la distribución
    de las etiquetas y el balance del flujo de red.

    Genera dos gráficos:
    1. Un gráfico de barras que muestra la distribución porcentual de la columna 'Label'.
    2. Un gráfico de pastel que muestra el promedio de paquetes por segundo ('Flow Packets/s')
       para cada etiqueta.

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

    Raises:
        ValueError: Si las columnas 'Label' o 'Flow Packets/s' no se encuentran en el DataFrame.
    """
    logger.info("\n--- [Exploración Gráfica] Iniciando la exploración de datos ---")

    # 1. Validación de columnas
    if 'Label' not in df.columns:
        raise ValueError("Error: La columna 'Label' no se encuentra en el DataFrame.")
    if 'Flow Packets/s' not in df.columns:
        raise ValueError("Error: La columna 'Flow Packets/s' no se encuentra en el DataFrame.")

    try:
        # 2. Impresión del balance inicial de clasificadores
```

```
        logger.info("[Exploración Gráfica] Balance de clasificadores:")
        label_counts_normalized = df['Label'].value_counts(normalize=True) * 100
        for label, percentage in label_counts_normalized.items():
            print(f"  - Etiqueta '{label}': {percentage:.2f}% del dataset")

        # 3. Creación de los gráficos
        fig, axes = plt.subplots(1, 2, figsize=(20, 8))

        # Gráfico 1: Distribución de Clases (Gráfico de Barras)
        class_distribution = df['Label'].value_counts()
        sns.barplot(x=class_distribution.index, y=class_distribution.values, ax=axes[0], palette='viridis')
        axes[0].set_title('Distribución de Clases')
        axes[0].set_xlabel('Etiqueta (Label)')
        axes[0].set_ylabel('Número de Muestras')

        # Añadir las etiquetas de porcentaje encima de las barras
        total_samples = len(df)
        for i, p in enumerate(axes[0].patches):
            percentage = '{:.2f}%'.format(100 * p.get_height() / total_samples)
            x = p.get_x() + p.get_width() / 2 - 0.1
            y = p.get_y() + p.get_height() * 1.01
            axes[0].annotate(percentage, (x, y), ha='center')

        # Gráfico 2: Promedio de Paquetes por Segundo (Gráfico de Pastel)
        # Se calcula el promedio de 'Flow Packets/s' por cada etiqueta de 'Label'.
        class_flow_packets = df.groupby('Label')['Flow Packets/s'].mean()
        axes[1].pie(class_flow_packets, labels=class_flow_packets.index, autopct='%1.1f%%', startangle=90, colors=sns.
        axes[1].set_title('Promedio de Paquetes por Segundo por Clase')

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

```
        logger.info("\n[Exploración Gráfica] ¡Proceso de exploración finalizado!")

    except Exception as ex:
        logger.info(f"\n[Exploración Gráfica] Excepción: {ex}")
        raise
```

Reducción de Dimensiones: Matriz de Correlacion:

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.

```
def f_create_label_map(dataset):
    logger.info('[Correlation Matrix] - Creación de etiquetas')
    unique_labels = sorted(dataset['Label'].unique())
    label_map = {label: idx+1 for idx, label in enumerate(unique_labels)}
    return label_map

def f_print_screen_correlation_matrix(corr):

    try:
        #Correlaciones Negativas
        k = 9 # number of features to select
        features_selected = corr.nsmallest(k, 'Label')['Label'].index
        logger.info(f'[Correlation Matrix] Características con una Correlación Negativa: {features_selected}')
        #Correlaciones Positivas
        k = 10 # number of features to select
        features_selected = corr.nlargest(k, 'Label')['Label'].index
```

```
        logger.info(f'[Correlation Matrix] Características con una Correlación Positiva: {features_selected}')

        logger.info('[Correlation Matrix] - Presentacion de Resultados')
        f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))
        # Entire DataFrame
        sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':10}, ax=ax1)
        ax1.set_title("Matriz de Correlación con variables no equilibradas \n (NO UTILIZAR COMO REFERENCIA)", fontsize
        sub_sample_corr = corr.corr()
        sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':10}, ax=ax2)
        ax2.set_title('Matriz de Correlación Subconjunto \n (UTILIZAR COMO REFERENCIA)', fontsize=14)

        plt.show()

    except Exception as ex:
        logger.info("[Correlation Matrix] - Exception print screen: ",ex)




def f_make_correlation_matrix(dataset):

    try:
        n_dim = 15 # Numero de dimensiones a reducir
        logger.info('[Matriz de Correlación] Iniciando proceso ...')
        # Crear un mapeo de etiquetas
        label_map = f_create_label_map(dataset)
        dataset['Label'] = dataset['Label'].map(label_map)
        print('[Matriz de Correlación] - Porcentaje: ', dataset['Label'].value_counts())
        # Calcular la matriz de correlación
        correlation_matrix = dataset.corr(numeric_only=True)
        logger.info('[Matriz de Correlación] - Presentación de resultados')
```

```
        # Crear el dataframe con los atributos más correlacionados
        top_corr = correlation_matrix.drop('Label')['Label'].nlargest(n_dim)
        top_corr = pd.concat([top_corr, correlation_matrix.drop('Label')['Label']])
        top_corr = pd.DataFrame(top_corr)
        # Crear el heatmap
        plt.figure(figsize=(10, 5))
        sns.heatmap(top_corr.head(n_dim), annot=True, cmap="RdBu", fmt=".1f", square=True)
        plt.show()
        # Split data into x(input) and y(output)
        top_15_attributes = list(top_corr.index)[:n_dim]
        x = dataset[top_15_attributes]
        y = dataset['Label']
        # Obtener los atributos principales

        # Agregar los atributos especificados
        atributos_principales = top_15_attributes
        atributos_principales.extend(['Source Port', 'Destination Port', 'Timestamp', 'Label'])

        print(f'[Matriz de Correlación] - Reducción dimensiones completada: {atributos_principales}')

        logger.info("[Matriz de Correlación] - Matrices de Correlación con Sub-Conjuntos de Datos Equilibrado y Sin Ec

        f_print_screen_correlation_matrix(correlation_matrix)

        logger.info("[Matriz de Correlación] - Proceso completado ...")
        return dataset[atributos_principales]


    except Exception as ex:
        print("M [Correlation Matrix] - Exception ", ex)
```

Detección de Anomalías / Detección de Outliers

El principal objetivo de esta sección será eliminar las posibles anomalías que encontremos en los extremos de la representación de los Diagramas de Cajas / Blackbox, en las características que tienen un elevado grado de correlación con nuestras clases. Esto podría tener un impacto positivo a la hora de incrementar la precisión del modelo

Metodo de Rango Intercuartile (IRQ):

Rango Intercuartile (IQR): Es posible calcular a partir de la direrencia entre el 75th percentile y el 25th percentile. El objetivo es crar un humbral mas alla de los percentiles 75th y 25th, y en el caso de que existan valores fuera del rango de valores definidos como validos, sean borrados.

Diagramas de Caja/ Boxplots: Es una repesentación gráfica, que permite un visión rápida de quellos valores que están dentro de un rango prestablecido 25th y 75th percentiles (both end of the squares), que se configuran como extremos, y ver las posibles anomalías o outliers que están fuera de los valores aceptables, tanto por el humbral superior como por el humbral inferior

Borrado de Anomalías :

En el proceso de borrado hay que ser cuidadoso en la forma de eliminar outliers que están fuera de los humbrales aceptables de información sensible. Se determinará el humbral, multiplicando un factor (e.j 1,5) por el Rango Intercuartile. El valor mas elevado de este humbral es en el que menor número de anomalias / outliers que se detectarán (calculo que se realizará multiplicando por un factor mas elevado e.j 3) y en el humbral menor es en que encontraremos un mayor número de outliers o anomalias Notas de aclaración: En el menor de los humbrales las anomalías proximas al humbral que serán borradas, sin embargo nos centraremos en los extremos de anomalias. Puesto que desde el punto de vista de información sensible al modelo, existe la posibilidad de sean mas valiosas, que en las anomalías mas proximas a los humbrales. Y podría producirse el caso de perder información relevante del modelo, y perder precisión, para ello se tendrán que realizar análisis y ver como afectan a la precisión y los modelos de clasificación.

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: Despues de la implementación de reducción de anomalias, la precisión de nuestro modelo habrá sido mejorada entorno a un 3%, algunos outliers pueden distorsionar la precisión del modelo, pero hay que tener presente evitar la perdida de información sensible al modelo, aun corriendo el riesgo de infraajuste.

Funciones y métodos:

- f_cleaning_outliers_simple: Es una implementación básica que calcula los outliers basándose en la clase de ataque y los elimina de todo el DataFrame. Su lógica es simple y puede ser propensa a errores al no separar la clase benigna.
- f_cleaning_outliers_complex: Es una versión más robusta y flexible. Aísla las clases benignas de las de ataque, lo que previene la eliminación de datos válidos. Además, permite configurar los atributos de limpieza y los factores del IQR, haciéndola más reutilizable.

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

```python
import pandas as pd
import numpy as np

def f_cleaning_outliers_simple(df: pd.DataFrame, atributos: list) -> pd.DataFrame:
    """
    Identifica y elimina outliers en un DataFrame basándose en el rango intercuartílico (IQR).

    Este método opera específicamente en los datos de la clase de ataque (etiqueta '1'),
    calcula los límites para cada atributo y elimina los valores atípicos de todo el DataFrame.

    Args:
        df (pd.DataFrame): DataFrame de entrada que contiene los datos.
        atributos (list): Lista de nombres de columnas sobre las que se aplicará la limpieza.

    Returns:
        pd.DataFrame: Un nuevo DataFrame con las filas que contienen outliers eliminadas.
    """
    print("--- [Limpieza de Outliers] Proceso iniciado ---")

    # Crea una copia del DataFrame para evitar modificar el original
    processed_df = df.copy()

    try:
        # 1. Validar que la columna 'Label' y los atributos existan en el DataFrame
        if 'Label' not in processed_df.columns:
            raise ValueError("El DataFrame no contiene la columna 'Label'.")

        for atributo in atributos:
            if atributo not in processed_df.columns:
                print(f"  - Advertencia: El atributo '{atributo}' no se encuentra en el DataFrame.")
                continue  # Salta al siguiente atributo
```

```
        # 2. Calcular los límites para los outliers usando la clase de ataque (Label == 1)
        # Se aísla la clase de ataque para calcular el IQR y los límites de forma específica.
        atributo_attack_data = processed_df.loc[processed_df['Label'] == 1, atributo]

        # Si no hay datos de ataque para un atributo, se salta el proceso
        if atributo_attack_data.empty:
            print(f"  - No se encontraron datos de la clase de ataque para el atributo '{atributo}'.")
            continue

        q25, q75 = np.percentile(atributo_attack_data, 25), np.percentile(atributo_attack_data, 75)
        iqr = q75 - q25
        cut_off = iqr * 1.5
        lower_bound, upper_bound = q25 - cut_off, q75 + cut_off

        print("-" * 50)
        print(f"Estadísticas del Atributo '{atributo}' (en la clase de ataque):")
        print(f"  - Quartil 25 (Q1): {q25:.4f}")
        print(f"  - Quartil 75 (Q3): {q75:.4f}")
        print(f"  - Rango Intercuartílico (IQR): {iqr:.4f}")
        print(f"  - Límite Inferior: {lower_bound:.4f}")
        print(f"  - Límite Superior: {upper_bound:.4f}")

        # 3. Identificar y contar los outliers
        # Se filtran los outliers en todo el DataFrame, pero usando los límites
        # calculados con los datos de ataque.
        outlier_rows = processed_df[(processed_df[atributo] < lower_bound) | (processed_df[atributo] > upper_bound

        print(f"\nSe encontraron {len(outlier_rows)} outliers para el atributo '{atributo}'.")
        print(f"  - Estos outliers se eliminarán de todo el DataFrame.")
```

```
            # 4. Eliminar los outliers del DataFrame
            # El método `.drop()` se utiliza con los índices de las filas identificadas.
            processed_df = processed_df.drop(outlier_rows.index)

            print(f"  - Número de registros restantes después de la limpieza: {len(processed_df)}")

    except Exception as ex:
        print(f"\n  [Limpieza de Outliers] - Excepción: {ex}")
        # Se lanza la excepción para que el programa se detenga si algo sale mal.
        raise

    print("-" * 50)
    print("  [Limpieza de Outliers] Proceso finalizado. Outliers eliminados.")
    return processed_df

import pandas as pd
import numpy as np
from typing import NoReturn, List

def f_cleaning_outliers_complex(df: pd.DataFrame,
                        atributos: List[str] = ['Flow Packets/s', 'Flow Bytes/s'],
                        iqr_factor: float = 1.5) -> pd.DataFrame:
    """
    Limpia los valores atípicos (outliers) de un DataFrame en las variables
    especificadas utilizando el método del rango intercuartílico (IQR).

    El proceso aísla las clases de ataque, calcula los límites de outliers para
    cada una de ellas individualmente y elimina las filas que caen fuera de
    estos límites. La clase 'BENIGN' se mantiene intacta.

    Args:
```

```
        df (pd.DataFrame): El DataFrame de entrada que contiene los datos a limpiar.
        atributos (List[str]): Lista de nombres de columnas a las que se aplicará la limpieza.
                               Por defecto, ['Flow Packets/s', 'Flow Bytes/s'].
        iqr_factor (float): El factor para multiplicar el IQR y determinar los límites
                            de los outliers. El valor estándar es 1.5.

    Returns:
        pd.DataFrame: Un nuevo DataFrame con los outliers de las clases de ataque eliminados.

    Raises:
        ValueError: Si las columnas necesarias no se encuentran en el DataFrame de entrada.
    """
    print("\n--- [Limpieza de Outliers] Iniciando el proceso ---")

    # 1. Validación de columnas y manejo de errores
    if 'Label' not in df.columns:
        raise ValueError("Error: El DataFrame de entrada debe contener la columna 'Label'.")

    missing_cols = [col for col in atributos if col not in df.columns]
    if missing_cols:
        raise ValueError(f"Error: Faltan los siguientes atributos en el DataFrame: {missing_cols}")

    # Creamos una copia para no modificar el DataFrame original
    df_cleaned = df.copy()

    try:
        # 2. Separar las clases de manera eficiente
        df_benign = df_cleaned[df_cleaned['Label'] == 'BENIGN']
        df_attack = df_cleaned[df_cleaned['Label'] != 'BENIGN']

        print(f"Número de registros BENIGN: {len(df_benign)}")
```

```
print(f"Número de registros de Ataque antes de la limpieza: {len(df_attack)}")

# 3. Limpieza de outliers para cada atributo en las clases de ataque
outliers_indices = pd.Index([])

# Iterar sobre las etiquetas de ataque únicas para un análisis más granular
attack_labels = df_attack['Label'].unique()

for atributo in atributos:
    print(f"\n[Limpieza de Outliers] Procesando el atributo '{atributo}'...")

    for label in attack_labels:
        data_clase = df_attack.loc[df_attack['Label'] == label, atributo]

        # Omitir el cálculo si la clase tiene menos de 2 muestras
        if len(data_clase) <= 1:
            print(f"  - Advertencia: La clase '{label}' tiene solo {len(data_clase)} muestras. Se omite la lim
            continue

        # 4. Calcular los límites del IQR
        q25, q75 = np.percentile(data_clase, 25), np.percentile(data_clase, 75)
        iqr = q75 - q25
        lower_bound = q25 - iqr * iqr_factor
        upper_bound = q75 + iqr * iqr_factor

        # 5. Identificar y acumular los índices de los outliers
        outliers = df_attack.loc[
            (df_attack['Label'] == label) &
            ((df_attack[atributo] < lower_bound) | (df_attack[atributo] > upper_bound))
        ]
        outliers_indices = outliers_indices.union(outliers.index)
```

```
            print(f"  - Clase '{label}': se identificaron {len(outliers)} outliers.")

    # 6. Eliminar los outliers y reconstruir el DataFrame
    df_attack_cleaned = df_attack.drop(outliers_indices)

    print(f"\n[Limpieza de Outliers] Total de outliers eliminados: {len(outliers_indices)}")
    print(f"Número de registros de Ataque después de la limpieza: {len(df_attack_cleaned)}")

    final_df = pd.concat([df_attack_cleaned, df_benign], ignore_index=True)

    print("\n[Limpieza de Outliers] Proceso finalizado. El DataFrame ha sido reconstruido.")
    print(f"Dimensiones del DataFrame final: {final_df.shape[0]} x {final_df.shape[1]}")

    return final_df

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

Tecnicas de Reducción de Dimensiones y Escalado Multidimensional

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.

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.)

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='coerce')
    # 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("[O].- Otras técnicas de Reducción de Dimensiones: ")

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

    # Verificar si todas las características en reduced_set_of_features están presentes en _df
    missing_features = [feature for feature in reduced_set_of_features_2 if feature not in _df.columns]
    if missing_features:
        raise ValueError(f"Las siguientes características no están presentes en el DataFrame: {missing_features}")

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


    print("[O.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('[O.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='Tráfi
axes[0].scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(_df['Label'] == 1), cmap='coolwarm', label='Tráfi
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
        axes[1].scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(_df['Label'] == 1), cmap='coolwarm', label='Tráfico
        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
        axes[2].scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(_df['Label'] == 1), cmap='coolwarm', label='Tráfico
        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 II: Comparativa modelos predictivos basados en AI/ML

Ingeniería del Modelo

La presente PoC establece una comparativa entre los distintos Algoritmos Predictivos ML/AI, que mejor pudiesen adaptarse a las caracteristicas de los diferentes Conjuntos de Datos que se han desarrollado (ASDDoS) a partir del Ataque DDoS documentado por el CIC, (CIDDDoS2019)

[A] Lectura del Conjunto de Datos

El codigo que a continuación se muestra realiza la lectura de un Conjunto de Datos

- [A.1] Lectura del Conjunto de Datos

```
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\n [{}]"
              .format(execution_time,file_path))

        filtered_df = filtered_df.iloc[::factor_sampling]  # Realizar un submuestreo 1 de cada 3 muestras

        print("1.4.- [Data Collection] Finalizada la carga del conjunto de datos: [{}]\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(filtered_df) * 100,2), '% del
        return filtered_df

    except Exception as ex:
```

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

[B] Preparacion de los Datos

La preparación de los datos obtenidos a través de los distintos Datasets son acondicionados, limpiados y corregidos mediante las funciones y métodos que se indican a continuación:

- [B.1] Limpieza del Conjunto de Datos: f_cleaning
- [B.2] Preprocesado y validación de los Datos: f_preprocesado(df)

```
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_duplicates}")
```

```
# 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 dataset")
```

```
        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

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 contiene {nan_values_count} valores 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 infinitos.")
            # Opcional: Podrías añadir una estrategia para manejar estos infinitos, por ejemplo:
            # data.replace([np.inf, -np.inf], np.nan, inplace=True)
        else:
            print("3.1.2.- [Preprocesado] Verificado: No hay valores infinitos en el DataFrame.")

        # --- 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 información.

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

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

```python
def explore_datasets_in_directory(directory_path: str, attributes: list):
    """
    Recorre un directorio, explora cada dataset CSV y genera un informe.
    """
    if not os.path.isdir(directory_path):
        print(f"Error: El directorio '{directory_path}' no existe.")
        return

    print("================================================================================")
    print("FASE I.- [Exploración de Conjuntos de Datos] - Inicio")
    print("================================================================================")

    # Obtener la lista de archivos en el directorio
    files = [f for f in os.listdir(directory_path) if f.endswith('.csv')]

    if not files:
        print(f"No se encontraron archivos CSV en el directorio: {directory_path}")
        return

    for file_name in files:
        input_file = os.path.join(directory_path, file_name)
        print(f"\n--- Procesando archivo: {file_name} ---")

        try:
            # 1. Carga inicial del dataset
            df = f_process_read_csv(input_file)
            if df is None:
                continue

            initial_rows, initial_cols = df.shape
            print(f"Filas iniciales: {initial_rows}, Columnas iniciales: {initial_cols}")
```

```
#print("Exploracion del Conjunto de Datos sin Procesar: ")
#for column, dtype in df.dtypes.items():
  #print(f"Columna: {column}, Tipo: {dtype}")

# 2. Limpieza de datos
df = f_cleaning(df)

final_rows, final_cols = df.shape
print(f"Filas Intermedias: {final_rows}, Columnas Intermedias: {final_cols}")

# 3. Selección de atributos
if not all(attr in df.columns for attr in attributes):
    missing_attrs = [attr for attr in attributes if attr not in df.columns]
    print(f"Advertencia: Faltan los siguientes atributos en el DataFrame: {missing_attrs}. Continuando con
    df = df.loc[:, df.columns.intersection(attributes)]
else:
    df = df[attributes]

# 4. Preprocesamiento final
df = f_preprocesado(df)
final_rows, final_cols = df.shape

final_rows, final_cols = df.shape
print(f"Filas Procesadas: {final_rows}, Columnas Procesadas: {final_cols}")

# 5. Generar y mostrar el informe de exploración
report = [
    ["Métrica", "Valor"],
    ["Nombre del Archivo", file_name],
    ["Filas Iniciales", initial_rows],
```

```
                ["Columnas Iniciales", initial_cols],
                ["Filas Finales", final_rows],
                ["Columnas Finales", final_cols],
                ["Total de Filas Eliminadas", initial_rows - final_rows],
                ["Total de Columnas Eliminadas", initial_cols - final_cols]
            ]

            final_rows, final_cols = df.shape
            print(f"Filas Finales: {final_rows}, Columnas Finales: {final_cols}")


            print("\n--- Informe de Exploración ---")
            print(tabulate(report, headers="firstrow", tablefmt="fancy_grid"))

        except Exception as e:
            print(f"Error inesperado al procesar '{file_name}': {e}")

    print("\n========================================================================")
    print("FASE I.- [Exploración de Conjuntos de Datos] - Fin")
    print("========================================================================")

'''
if __name__ == "__main__":
    # Ruta del directorio con los datasets de entrenamiento
    data_directory = 'C:/Datasets/asddos2024/Training'

    # Ejecutar la exploración
    explore_datasets_in_directory(data_directory, PCA_25_Attributes)
'''
```

'\nif __name__ == "__main__":\n   # Ruta del directorio con los datasets de entrenamiento\n   data_directory = \'C:/

[C] Exploracion Grafica del Conjunto de Datos

A continuación para estudiar la estructura del conjunto de datos se realizará una exploración gráfica, utilizando las siguientes técnicas

- [C.1] Histogramas: f_show_histograms
- [C.2] Diagramas de Burbujas (Clustering): f_exploracion_grafica_analisis_clustering
- [C.3] Diagramas de Cajas: f_exploracion_grafica_datos_boxplot

```
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)
```

```
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=14)
    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=14)
    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}. Asegúrate de que todas las columnas requeridas existen en el DataFrame.")
    except Exception as ex:
        print(f"Error inesperado al generar los scatter plots: {ex}")

#f_exploracion_grafica_analisis_clustering(_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).dropna()

    # 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_bound)])
                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)
```

[D] Presentación de Resultados

El Siguiente codigo permite a través de diagramas de Radar una visualización fácil e intuitiva de resultados que permitan identifica la intensidad de la variante del ataque dentro del conjunto de datos, tal y como sigue mediante un diccionario donde las claves son los nombres de las variantes de ataque (str) y los valores son las métricas de precisión (float o int).

Ejemplo: {'TCP Reflection': 95.5, 'UDP Flood': 98.2, ...,'TCP Flood': 98.2}

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

def f_resultados_v_and_v(results_dict: Dict[str, Union[float, int]]) -> None:
    """
    Genera un gráfico de radar y una tabla comparativa para visualizar
    la precisión de la detección de diferentes variantes de ataques DDoS.

    Args:
        results_dict (Dict[str, Union[float, int]]): Un diccionario donde las
            claves son los nombres de las variantes de ataque (str) y los
            valores son las métricas de precisión (float o int).
            Ejemplo: {'TCP Reflection': 95.5, 'UDP Flood': 98.2, ...}
```

```
    Raises:
        ValueError: Si el diccionario de resultados está vacío.
    """
    if not results_dict:
        raise ValueError("El diccionario de resultados no puede estar vacío.")

    # Nombres de las variantes de ataque y sus valores
    variantes = list(results_dict.keys())
    valores = list(results_dict.values())

    # Asegurarse de que los valores sean numéricos
    try:
        valores = [float(v) for v in valores]
    except (TypeError, ValueError) as e:
        print(f"Error: Los valores en el diccionario de resultados deben ser numéricos. Excepción: {e}")
        return

    # Preparar datos para el gráfico de radar
    num_vars = len(variantes)
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()

    # Cerrar el círculo del gráfico de radar
    valores += valores[:1]
    angles += angles[:1]

    # Configuración del estilo de Matplotlib
    plt.style.use('seaborn-v0_8-whitegrid')

    # Crear la figura y los subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 8))
```

```
# ------------------- Gráfico de Radar -------------------
# El subplot se crea directamente con la proyección polar
ax1 = fig.add_subplot(1, 2, 1, polar=True)
ax1.set_title('Precisión de Detección por Variante de Ataque', fontsize=16, pad=20)

# Rellenar y dibujar la línea del gráfico de radar
ax1.fill(angles, valores, color='blue', alpha=0.25)
ax1.plot(angles, valores, color='blue', linewidth=2, label='Precisión de Detección')

# Configurar las etiquetas del gráfico de radar
ax1.set_yticklabels([])
ax1.set_xticks(angles[:-1])
ax1.set_xticklabels(variantes, fontsize=12)
ax1.legend(loc='upper right', bbox_to_anchor=(0.1, 0.1))

# ------------------- Tabla Comparativa -------------------
# Ocultar los ejes del segundo subplot para mostrar la tabla
ax2.set_title('Tabla Comparativa de Resultados', fontsize=16)
ax2.axis('off')

# Crear un DataFrame para la tabla
table_data = {
    'Variante del Ataque': variantes,
    '% Probabilidad DDoS': [f'{v:.2f}%' for v in valores[:-1]]
}
df_table = pd.DataFrame(table_data)

# Crear la tabla en el subplot
table = ax2.table(cellText=df_table.values,
                  colLabels=df_table.columns,
```

```
                        cellLoc='center',
                        loc='center')

    # Ajustar el formato de la tabla
    table.auto_set_font_size(False)
    table.set_fontsize(12)
    table.auto_set_column_width(col=list(range(len(df_table.columns))))
    table.scale(1.2, 1.8) # Ajustar la escala general de la tabla

    # Asegurar que el layout se ajuste
    plt.tight_layout()
    plt.show()

# Ejemplo de uso de la función mejorada:
# resultados = {
#     'TCP Reflection': 95.5,
#     'UDP Reflection': 98.2,
#     'Mixed Reflection': 91.0,
#     'TCP Flood': 99.1,
#     'UDP Flood': 97.8
# }
# f_resultados_v_and_v(resultados)
```

[E] Ingeniería del Modelo

En esta fase se desarrolla el código necesario para la evaluación de las prestaciones de los modelos ML/AI propuestos

Logistic Regression, (Parte II)

Naïve Bayes,

K-nearest neighbours,

Multi Layer Perceptron,

Ada Boost,

Quadrant Discriminant Analysis,

Random Forests,

Pipelining and Ensemble,

```
import pandas as pd
import os
import joblib
import pickle
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tabulate import tabulate
from typing import Tuple, Any, List, Dict, NoReturn
from xgboost import XGBClassifier  # Importar XGBoost

from matplotlib import pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import History
```

```
from keras.optimizers import Adam

import logging

# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# --- Funciones de Evaluación y Visualización ---

def plot_training_history(history: History) -> NoReturn:
    """
    Genera y muestra gráficos de la pérdida y la exactitud del entrenamiento
    a partir del objeto `History` de Keras.

    Args:
        history (History): El objeto History devuelto por el método `fit()` de Keras.
    """
    history_dict: Dict[str, Any] = history.history
    logger.info("--- [Visualización] Generando gráficos de pérdida y exactitud ---")

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

    # Gráfico de la pérdida
    ax1.set_title('Pérdida (Cross Entropy Loss)')
    ax1.plot(history_dict.get('loss', []), color='blue', label='Entrenamiento')
    if 'val_loss' in history_dict:
        ax1.plot(history_dict['val_loss'], color='orange', label='Validación')
```

```
    ax1.set_xlabel('Época')
    ax1.set_ylabel('Pérdida')
    ax1.legend()
    ax1.grid(True)

    # Gráfico de la exactitud
    ax2.set_title('Exactitud (Accuracy)')
    ax2.plot(history_dict.get('accuracy', []), color='blue', label='Entrenamiento')
    if 'val_accuracy' in history_dict:
        ax2.plot(history_dict['val_accuracy'], color='orange', label='Validación')
    ax2.set_xlabel('Época')
    ax2.set_ylabel('Exactitud')
    ax2.legend()
    ax2.grid(True)

    plt.tight_layout(pad=3.0)
    plt.show()

    logger.info("[Visualización] Gráficos generados exitosamente.")

def specificity_score(y_true, y_pred):
    """Calcula la especificidad (TNR) a partir de la matriz de confusión."""
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    if (tn + fp) == 0:
        return 0.0
    return tn / (tn + fp)

def benchmark_and_present_results(X: pd.DataFrame, y: pd.Series) -> Tuple[Any, str]:
    """
    Entrena y evalúa una lista de clasificadores de Machine Learning,
    y presenta los resultados de los KPIs en una tabla clara.
```

```
    Args:
        X (pd.DataFrame): DataFrame de características.
        y (pd.Series): Serie de etiquetas.

    Returns:
        Tuple[Any, str]: Una tupla que contiene el mejor modelo entrenado
                         y su nombre.
    """
    logger.info("[Benchmarking] Algoritmos de clasificación de amenazas ---")

    # Corregido: Uso de np.unique() para manejar tanto Series como ndarrays
    if len(np.unique(y)) > 2:
        logger.warning("El problema no es de clasificación binaria. Se usará 'average' = 'weighted' para las métricas.
        average_method = 'weighted'
    else:
        average_method = 'binary'

    # Inicialización de todos los clasificadores
    lr = LogisticRegression(max_iter=1000, random_state=42)
    gnb = GaussianNB()
    kn = KNeighborsClassifier()
    mlp = MLPClassifier(max_iter=1000, random_state=42)
    ada = AdaBoostClassifier(n_estimators=100, random_state=42)
    rfc = RandomForestClassifier(n_estimators=100, random_state=42)
    svc = SVC(probability=True, max_iter=1000, random_state=42)
    id3 = DecisionTreeClassifier(criterion="entropy", random_state=42)
    xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
    stacking = StackingClassifier(estimators=[('lr', lr), ('rfc', rfc), ('gnb', gnb)])

    # Se incluye el modelo de Keras en la lista de clasificadores
```

```
keras_model = Sequential([
    Dense(64, activation='relu', input_shape=(X.shape[1],)),
    Dense(1, activation='sigmoid')
])
keras_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

clf_list = [
    (lr, "Regresión Logística"), (gnb, "Naive Bayes"), #(svc, "SVC"),
    (kn, "K-nearest neighbours"), (mlp, "Multi Layer Perceptron"),
    (ada, "Ada Boost"), (rfc, "Random Forests"), (id3, "ID3"),
    (xgb, "XGBoost"), (stacking, "Pipelining and Ensemble"),
    (keras_model, "Keras Sequential")
]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

kpis = {
    'Exactitud': accuracy_score,
    'Precision': lambda yt, yp: precision_score(yt, yp, average=average_method, zero_division=0),
    'Recall': lambda yt, yp: recall_score(yt, yp, average=average_method, zero_division=0),
    'F1-Score': lambda yt, yp: f1_score(yt, yp, average=average_method, zero_division=0),
    'Especificidad': specificity_score
}
kpi_results = {}
best_model_name = ""
best_f1_score = -1
best_model = None

logger.info("[Benchmarking] - Iniciando entrenamiento y evaluación ---")
for clf, name in clf_list:
    print(f"-> Entrenando y evaluando: {name}")
```

```
        try:
            if isinstance(clf, Sequential):
                history = clf.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_split=0.2)
                plot_training_history(history)
                pred = (clf.predict(X_test) > 0.5).astype("int32").flatten()
                logger.info(f"[Evaluación] Informe de clasificación para {name}:")
                print(classification_report(y_test, pred, zero_division=0))
            else:
                clf.fit(X_train, y_train)
                pred = clf.predict(X_test)
                print(f"[Evaluación] Informe de clasificación para {name}:")
                print(classification_report(y_test, pred, zero_division=0))

            kpi_scores = {kpi: metric(y_test, pred) for kpi, metric in kpis.items()}
            kpi_results[name] = kpi_scores

            if kpi_scores['F1-Score'] > best_f1_score:
                best_f1_score = kpi_scores['F1-Score']
                best_model_name = name
                best_model = clf

            print(f"    -> Exactitud de {name}: {kpi_scores['Exactitud']:.4f}")
        except Exception as e:
            logger.error(f"    -> Error al entrenar/evaluar {name}: {e}")
            kpi_results[name] = {kpi: None for kpi in kpis.keys()}

    kpi_df = pd.DataFrame(kpi_results).T.applymap(lambda x: f"{x * 100:.2f}%" if pd.notnull(x) else "N/A")
    kpi_df.index.name = 'Clasificador'

    print("\n\n" + "="*80)
    logger.info("RESUMEN DE RENDIMIENTO DE LOS CLASIFICADORES")
```

```
    print("="*80)
    print(tabulate(kpi_df, headers='keys', tablefmt='psql'))

    print(f"\nEl mejor modelo encontrado es: '{best_model_name}' con un F1-Score de {best_f1_score*100:.2f}%")
    print("="*80)

    return best_model, best_model_name


def export_model(model: Any, model_name: str, dataset_name: str, export_methods: List[str] = ['joblib', 'pickle']):
    """
    Exporta el modelo usando diferentes métodos, con nombres de archivo descriptivos.

    Args:
        model (Any): El modelo entrenado a exportar.
        model_name (str): El nombre del modelo (ej. 'Random Forests').
        dataset_name (str): El nombre del dataset de entrenamiento.
        export_methods (List[str]): Lista de métodos de exportación a utilizar.

    Returns:
        List[str]: Rutas a los archivos exportados.
    """
    logger.info("[Ingenieria del Modelo] Exportación del modelo")
    print(f"\nExportando el modelo '{model_name}' usando métodos: {export_methods}")

    os.makedirs('models', exist_ok=True)
    exported_files = []
    sanitized_model_name = model_name.replace(" ", "_").lower()
    sanitized_dataset_name = dataset_name.replace(" ", "_").lower()

    if 'joblib' in export_methods:
```

```
        joblib_path = f'models/{sanitized_model_name}_{sanitized_dataset_name}.joblib'
        try:
            joblib.dump(model, joblib_path)
            exported_files.append(joblib_path)
            logger.info(f"   Modelo exportado con joblib: {joblib_path}")
        except Exception as e:
            logger.error(f"   Error al exportar con joblib: {e}")

    if 'pickle' in export_methods:
        pickle_path = f'models/{sanitized_model_name}_{sanitized_dataset_name}.pkl'
        try:
            with open(pickle_path, 'wb') as f:
                pickle.dump(model, f)
            exported_files.append(pickle_path)
            logger.info(f"   Modelo exportado con pickle: {pickle_path}")
        except Exception as e:
            logger.error(f"   Error al exportar con pickle: {e}")

    return exported_files

def process_single_dataset(file_path: str, attributes: List[str]) -> None:
    """
    Procesa un único archivo CSV, entrena y evalúa modelos, y exporta el mejor.

    Args:
        file_path (str): Ruta completa al archivo CSV.
        attributes (List[str]): Lista de nombres de columnas a utilizar como características.
        f_process_read_csv (callable): Función para cargar y preprocesar el CSV inicial.
        f_cleaning (callable): Función para limpiar el DataFrame.
        f_preprocesado (callable): Función para el preprocesamiento final de los datos.
    """
```

```
    file_name = os.path.basename(file_path)
    print(f"\n--- Procesando archivo: {file_name} ---")

    try:
        # 1. Carga, limpieza y preprocesamiento de datos
        print(f"Carga, limpieza y preprocesamiento de datos: {file_name}")

        df = f_process_read_csv(file_path)
        if df is None:
            return
        df = f_cleaning(df)
        # Filtrado de atributos
        df_columns = set(df.columns)
        attr_set = set(attributes)
        if not attr_set.issubset(df_columns):
            missing_attrs = list(attr_set - df_columns)
            logger.warning(f"Advertencia: Faltan los siguientes atributos en el DataFrame: {missing_attrs}. Se usarán
            df = df.loc[:, df_columns.intersection(attr_set)]
            return    # Salida Controlada
        else:
            df = df[attributes]


        # 2. Preparación de datos para el modelado
        print(f"Preparación de datos para el modelado: {file_name}")
        df = f_preprocesado(df)
        ratios_deseados = {'BENIGN': 0.7, 'DDoS': 0.3}
        df_balanceado = f_balancear_dataset_oversampling(df=df[PCA_25_Attributes], ratios=ratios_deseados, target_samp
        if 'Label' not in df.columns:
            logger.error("Error: La columna 'Label' es necesaria para el entrenamiento.")
            return
```

```
        X = df.drop('Label', axis=1)
        y = df['Label']
        label_encoder = LabelEncoder()
        y = label_encoder.fit_transform(y)

        # 3. Benchmarking de clasificadores
        print(f"Iniciando la elaboración del Modelo Predictivo sobre el Dataset: {file_name}")
        best_model, best_model_name = benchmark_and_present_results(X, y)

        # 4. Exportar el mejor modelo
        if best_model and best_model_name:
            dataset_name = os.path.splitext(file_name)[0]
            export_model(best_model, best_model_name, dataset_name)
        else:
            logger.info("No se pudo exportar el modelo, ya que no se encontró el mejor modelo.")

    except Exception as ex:
        logger.error(f"Error al procesar el archivo '{file_name}': {ex}")

def f_create_machine_learning_models(data_directory: str, attributes: List[str]):
    """
    Recorre un directorio, explora cada dataset CSV, entrena y evalúa
    múltiples modelos de aprendizaje automático y exporta el mejor de ellos.

    Args:
        data_directory (str): Ruta al directorio que contiene los datasets CSV.
        attributes (List[str]): Lista de nombres de columnas a utilizar como características.
        f_process_read_csv (callable): Función para cargar y preprocesar el CSV inicial.
        f_cleaning (callable): Función para limpiar el DataFrame.
        f_preprocesado (callable): Función para el preprocesamiento final de los datos.
```

```
    """
    if not os.path.isdir(data_directory):
        logger.error(f"Error: El directorio '{data_directory}' no existe.")
        return

    print("="*80)
    print("FASE II.- [INGENIERIA DEL MODELO] - Inicio")
    print("="*80)

    files = [f for f in os.listdir(data_directory) if f.endswith('.csv')]

    if not files:
        logger.info(f"No se encontraron archivos CSV en el directorio: {data_directory}")
        return

    for file_name in files:
        input_file = os.path.join(data_directory, file_name)
        process_single_dataset(
            file_path=input_file,
            attributes=attributes
        )

    print("="*80)
    print("FASE II.- [INGENIERIA DEL MODELO] - Fin")
    print("="*80)


_filename = 'C:/DataSets/asddos2024/Training/Training_Exp_Attack_TCP.csv'
_attributes = PCA_25_Attributes
logger.info(f"PoC: Desarrollo de un Modelo Predictivo. Training Dataset {_filename}")
process_single_dataset(_filename, _attributes)
```

```
logger.info(f"PoC: FIN - Proceso completado ...")
```

```
2025-08-09 09:18:25,587 - __main__ - INFO - PoC: Desarrollo de un Modelo Predictivo. Training Dataset C:/DataSets/asdo
2025-08-09 09:18:25,589 - __main__ - INFO -
--- Procesando archivo: Training_Exp_Attack_TCP.csv ---

Carga, limpieza y preprocesamiento de datos: Training_Exp_Attack_TCP.csv
1.1- [Data Collection] Process-CSV: [C:/DataSets/asddos2024/Training/Training_Exp_Attack_TCP.csv]

1.2.- [Data Collection] Finalización carga de Tiempo: 0.40 secs
 [C:/DataSets/asddos2024/Training/Training_Exp_Attack_TCP.csv]
1.4.- [Data Collection] Finalizada la carga del conjunto de datos: [C:/DataSets/asddos2024/Training/Training_Exp_Attac

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. ---
Preparación de datos para el modelado: Training_Exp_Attack_TCP.csv
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 26 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   Fwd IAT Std                  1582681 non-null  float64
 1   Fwd IAT Mean                 1582681 non-null  float64
 2   Fwd IAT Total                1582681 non-null  float64
 3   Flow IAT Min                 1582681 non-null  float64
 4   Flow IAT Max                 1582681 non-null  float64
 5   Flow IAT Std                 1582681 non-null  float64
 6   Flow IAT Mean                1582681 non-null  float64
 7   Flow Packets/s               1582681 non-null  float64
 8   Flow Bytes/s                 1582681 non-null  float64
 9   Bwd Packet Length Std        1582681 non-null  float64
 10  Bwd Packet Length Mean       1582681 non-null  float64
 11  Bwd Packet Length Min        1582681 non-null  float64
 12  Bwd Packet Length Max        1582681 non-null  float64
 13  Fwd Packet Length Std        1582681 non-null  float64
 14  Fwd Packet Length Mean       1582681 non-null  float64
 15  Fwd Packet Length Min        1582681 non-null  float64
 16  Fwd Packet Length Max        1582681 non-null  float64
 17  Total Length of Bwd Packets  1582681 non-null  float64
 18  Total Length of Fwd Packets  1582681 non-null  float64
 19  Total Backward Packets       1582681 non-null  float64
 20  Total Fwd Packets            1582681 non-null  float64
 21  Flow Duration                1582681 non-null  float64
 22  Protocol                     1582681 non-null  float64
```

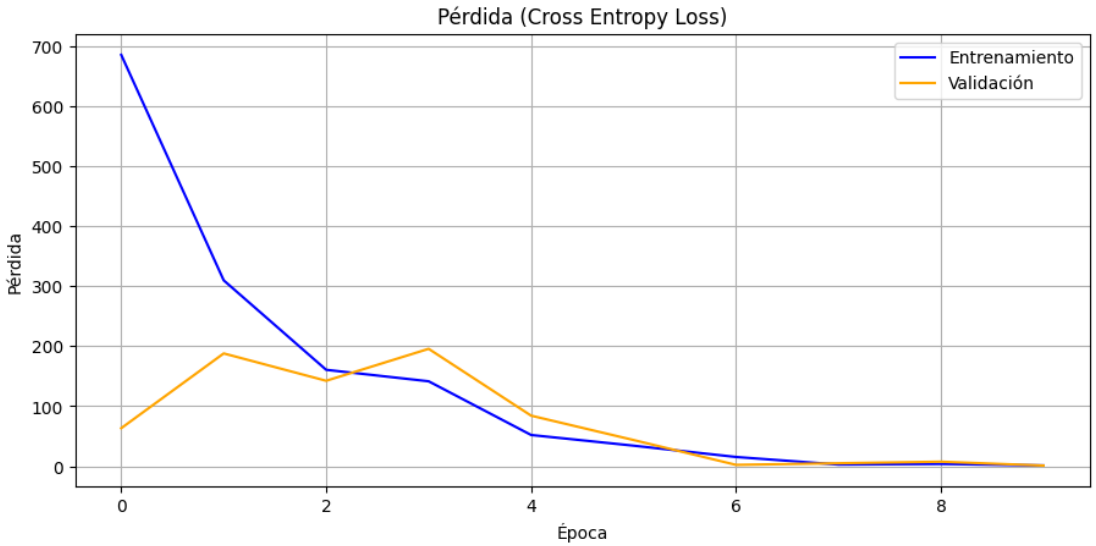
```
 23  Destination Port             1582681 non-null  float64
 24  Source Port                  1582681 non-null  float64
 25  Label                        1582681 non-null  object
dtypes: float64(25), object(1)
memory usage: 313.9+ MB

3.2.- [Preprocesado] ¡Preprocesado finalizado!


2025-08-09 09:19:41,607 - __main__ - INFO - --- [Balanceo] Iniciando el proceso de balanceo del conjunto de datos ---
2025-08-09 09:19:41,612 - __main__ - INFO - Dimensiones iniciales del DataFrame: 1582681 x 26
2025-08-09 09:19:41,804 - __main__ - INFO -
[Balanceo] Balance inicial del conjunto de datos:


  - Etiqueta 'DDoS': 99.98%
  - Etiqueta 'BENIGN': 0.02%


2025-08-09 09:19:42,265 - __main__ - INFO -
[Balanceo] Aplicando oversampling con SMOTE para igualar las clases.
2025-08-09 09:19:44,730 - __main__ - INFO - Oversampling con SMOTE finalizado. Nuevas dimensiones: 3164578 x 26
2025-08-09 09:19:44,732 - __main__ - INFO -
[Balanceo] Filtrando datos atípicos sintéticos.
2025-08-09 09:19:59,189 - __main__ - INFO -
[Balanceo] Remuestreando para un tamaño final de 100000 muestras.
2025-08-09 09:20:00,084 - __main__ - INFO -
[Balanceo] Balance final del conjunto de datos:
2025-08-09 09:20:00,095 - __main__ - INFO - Proceso finalizado. Dimensiones del DataFrame de salida: 100000 x 26


  - Etiqueta 'BENIGN': 70.00%
  - Etiqueta 'DDoS': 30.00%
```

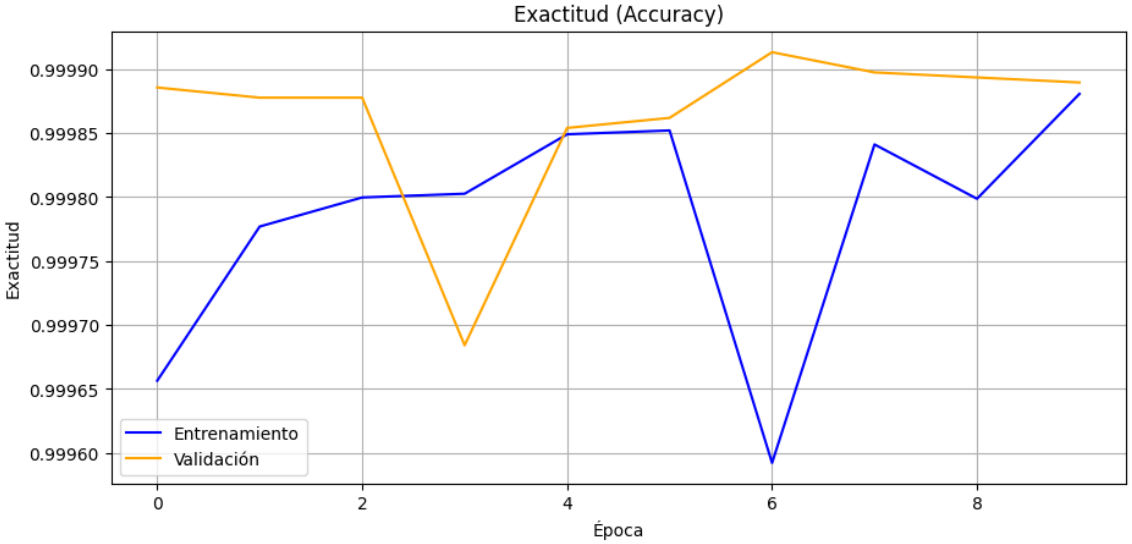
```
2025-08-09 09:20:00,811 - __main__ - INFO - [Benchmarking] Algoritmos de clasificación de amenazas ---

Iniciando la elaboración del Modelo Predictivo sobre el Dataset: Training_Exp_Attack_TCP.csv

2025-08-09 09:20:01,934 - __main__ - INFO - [Benchmarking] - Iniciando entrenamiento y evaluación ---

-> Entrenando y evaluando: Regresión Logística
[Evaluación] Informe de clasificación para Regresión Logística:
              precision    recall  f1-score   support

           0       1.00      0.08      0.16        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       1.00      0.54      0.58    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de Regresión Logística: 0.9998
-> Entrenando y evaluando: Naive Bayes
[Evaluación] Informe de clasificación para Naive Bayes:
              precision    recall  f1-score   support

           0       0.01      0.14      0.01        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       0.50      0.57      0.51    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de Naive Bayes: 0.9954
-> Entrenando y evaluando: K-nearest neighbours
```

```
[Evaluación] Informe de clasificación para K-nearest neighbours:
              precision    recall  f1-score   support

           0       0.95      0.85      0.90        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       0.98      0.92      0.95    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de K-nearest neighbours: 1.0000
-> Entrenando y evaluando: Multi Layer Perceptron
[Evaluación] Informe de clasificación para Multi Layer Perceptron:
              precision    recall  f1-score   support

           0       0.81      0.65      0.72        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       0.90      0.82      0.86    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de Multi Layer Perceptron: 0.9999
-> Entrenando y evaluando: Ada Boost
[Evaluación] Informe de clasificación para Ada Boost:
              precision    recall  f1-score   support

           0       0.98      0.89      0.93        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
```

```
   macro avg       0.99      0.94      0.97    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de Ada Boost: 1.0000
-> Entrenando y evaluando: Random Forests
[Evaluación] Informe de clasificación para Random Forests:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       1.00      0.98      0.99    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de Random Forests: 1.0000
-> Entrenando y evaluando: ID3
[Evaluación] Informe de clasificación para ID3:
              precision    recall  f1-score   support

           0       0.97      0.96      0.96        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       0.99      0.98      0.98    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de ID3: 1.0000
-> Entrenando y evaluando: XGBoost
[Evaluación] Informe de clasificación para XGBoost:
              precision    recall  f1-score   support
```

```
           0       0.70      0.73      0.72        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       0.85      0.87      0.86    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de XGBoost: 0.9999
-> Entrenando y evaluando: Pipelining and Ensemble
[Evaluación] Informe de clasificación para Pipelining and Ensemble:
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       1.00      0.93      0.96    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de Pipelining and Ensemble: 1.0000
-> Entrenando y evaluando: Keras Sequential

2025-08-09 10:15:30,763 - __main__ - INFO - --- [Visualización] Generando gráficos de pérdida y exactitud ---
```


Pérdida (Cross Entropy Loss)
Entrenamiento
Validación
Pérdida
Época
0
100
200
300
400
500
600
700
0
2
4
6
8


Exactitud (Accuracy)
Entrenamiento
Validación
Exactitud
Época
0.99960
0.99965
0.99970
0.99975
0.99980
0.99985
0.99990
0
2
4
6
8

```
2025-08-09 10:15:31,577 - __main__ - INFO - [Visualización] Gráficos generados exitosamente.

9892/9892            9s 899us/step

2025-08-09 10:15:45,903 - __main__ - INFO - [Evaluación] Informe de clasificación para Keras Sequential:
2025-08-09 10:15:46,060 - __main__ - INFO - RESUMEN DE RENDIMIENTO DE LOS CLASIFICADORES
2025-08-09 10:15:46,099 - __main__ - INFO - [Ingenieria del Modelo] Exportación del modelo
2025-08-09 10:15:46,169 - __main__ - INFO -    Modelo exportado con joblib: models/random_forests_training_exp_attack
2025-08-09 10:15:46,182 - __main__ - INFO -    Modelo exportado con pickle: models/random_forests_training_exp_attack

              precision    recall  f1-score   support

           0       1.00      0.59      0.74        71
           1       1.00      1.00      1.00    316466

    accuracy                           1.00    316537
   macro avg       1.00      0.80      0.87    316537
weighted avg       1.00      1.00      1.00    316537

    -> Exactitud de Keras Sequential: 0.9999


================================================================================
================================================================================
+-------------------------+-------------+-------------+----------+------------+-----------------+
| Clasificador            | Exactitud   | Precision   | Recall   | F1-Score   | Especificidad   |
|-------------------------+-------------+-------------+----------+------------+-----------------|
| Regresión Logística     | 99.98%      | 99.98%      | 100.00%  | 99.99%     | 8.45%           |
| Naive Bayes             | 99.54%      | 99.98%      | 99.56%   | 99.77%     | 14.08%          |
| K-nearest neighbours    | 100.00%     | 100.00%     | 100.00%  | 100.00%    | 84.51%          |
| Multi Layer Perceptron  | 99.99%      | 99.99%      | 100.00%  | 99.99%     | 64.79%          |
```

```
| Ada Boost                | 100.00%     | 100.00%     | 100.00%  | 100.00%    | 88.73%          |
| Random Forests           | 100.00%     | 100.00%     | 100.00%  | 100.00%    | 95.77%          |
| ID3                      | 100.00%     | 100.00%     | 100.00%  | 100.00%    | 95.77%          |
| XGBoost                  | 99.99%      | 99.99%      | 99.99%   | 99.99%     | 73.24%          |
| Pipelining and Ensemble | 100.00%     | 100.00%     | 100.00%  | 100.00%    | 85.92%          |
| Keras Sequential         | 99.99%      | 99.99%      | 100.00%  | 100.00%    | 59.15%          |
+--------------------------+-------------+-------------+----------+------------+-----------------+

El mejor modelo encontrado es: 'Random Forests' con un F1-Score de 100.00%
================================================================================

Exportando el modelo 'Random Forests' usando métodos: ['joblib', 'pickle']

2025-08-09 10:15:46,224 - __main__ - INFO - PoC: FIN - Proceso completado ...
```

```
!dir models
```

```
 El volumen de la unidad C no tiene etiqueta.
 El n£mero de serie del volumen es: 34FA-F652

 Directorio de C:\Users\Usuario\OneDrive - Universidad de Alcala\Estudios Doctorado 1er A¤o\Cuadernos\CICDDoS\models

09/08/2025  10:15    <DIR>          .
09/08/2025  10:15    <DIR>          ..
01/08/2025  20:37            66.316 ada_boost_training_exp_attack_tcp.joblib
01/08/2025  20:37           (56.428) ada_boost_training_exp_attack_tcp.pkl
05/08/2025  20:21         1.560.080 lstm_optimized_training_exp_attack_tcp.csv.h5
07/08/2025  16:40            73.884 modelo_predictivo.pkl
09/08/2025  10:15           739.289 random_forests_training_exp_attack_tcp.joblib
09/08/2025  10:15           731.108 random_forests_training_exp_attack_tcp.pkl
01/08/2025  20:48       (1.533.049) random_forests_training_exp_attack_udp.joblib
```

```
01/08/2025  20:48       (1.524.852) random_forests_training_exp_attack_udp.pkl
               8 archivos      6.285.006 bytes
               2 dirs  354.955.464.704 bytes libres
```

```
'''
PoC 1: Comparativa modelos predictivos basados en AI/ML
    DESCRIPCION: Entrena y evalúa una lista de clasificadores de Machine Learning,
    y presenta los resultados de los KPIs en una tabla clara.
'''
def main():
    # Lista de etapas a ejecutar, incluyendo los argumentos necesarios para cada función
    data_directory = 'C:/DataSets/asddos2024/Training/'
    attributes = PCA_25_Attributes
    stages: List[Dict] = [
        {
            "title": "2.1- [Ingeniería del Modelo] Comparativa Modelos Machine Learning Predictivos",
            "function": f_create_machine_learning_models,
            "description": "Creación de los Modelos de Entrenamiento",
            "args": {
                "data_directory": data_directory,
                "attributes": attributes
            }
        }
    ]

    # Ejecutar cada etapa en el orden definido
    for stage in stages:
        logger.info(f"--- Ejecutando etapa: {stage['title']} ---")

        # Obtener la función y los argumentos del diccionario de la etapa
        function_to_call = stage["function"]
```

```
        args = stage.get("args", {})

        # Llamar a la función directamente con sus argumentos
        function_to_call(**args)

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

[F] V&V - Validacion y Verificación

En esta sección se implementará el código, mediante las funciones necesarias, que permitan mediante la utilización de un Conjunto de Datos con datos reales de producción, identificar de forma y manera univoca posibles ataques sobre la infraestructura de red

```
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


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: {missing_cols}")
else:
    print(" -> El conjunto de datos tiene las caracteristicas necesarias")

# 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='ign
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']

        if 'Label' in X_production.columns:
            X_production = X_production.drop('Label', axis=1)
```

```
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(f"Atributos: {X_production.columns}")
print(f"Caracteristicas deseadas: {atributos}")

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"\n ¡ALERTA! Se han detectado {num_ataques_detectados} instancias de ataques en el tráfico de red.
    print("--------------------------------------------------------------------------")
    # Puedes mostrar las instancias de ataque si lo deseas
    # ataques_detectados = production_data.iloc[produccion_predicciones == 1]
    # print("Registros de ataques detectados:")
    # print(ataques_detectados)
else:
    print("\n  No se han detectado patrones de Ataque DDoS por Inundación TCP en el tráfico analizado")
    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_divisio
```

```
        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}")
```

```
'''
PoC 2: Verificación y Validacion del Modelo con Datos de Producción Reales
    DESCRIPCION: Evalua las prestaciones de un modelo concreto con datos reales.
'''
# ----------------- 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.
def main():
    _atributos = PCA_25_Attributes
    _path_production_dataset: str = "C:/Datasets/Produccion/Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv"
    _model_path: str = './models/random_forests_training_exp_attack_tcp.pkl'
    fase_produccion(_model_path, _path_production_dataset, _atributos)

if __name__ == '__main__':
```

```
    main()
```

```
--- 11. [FASE DE PRODUCCIÓN] - Análisis del tráfico en tiempo real ---
  -> Leyendo el conjunto de datos de producción desde: C:/Datasets/Produccion/Friday-WorkingHours-Afternoon-DDos.pcap_
  -> Limpiando y preparando los datos de producción.
 -> El conjunto de datos tiene las caracteristicas necesarias
  -> Preparación de datos de producción completada.
11.2- [Data Collection] Conversión de tipos y limpieza completada.
Atributos: Index(['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'],
      dtype='object')
Caracteristicas deseadas: ['Fwd IAT Std', 'Fwd IAT Mean', 'Fwd IAT Total', 'Flow IAT Min', 'Flow IAT Max', 'Flow IAT S
11.3.- [Producción] - Realizando predicciones sobre el conjunto de producción.

  ¡ALERTA! Se han detectado 62586 instancias de ataques en el tráfico de red.
----------------------------------------------------------------------

11.4.- [Producción] - Calculando métricas de rendimiento
11.5.- [Producción] Métricas de rendimiento del modelo:
Accuracy: 56.84%
Precision: 63.90%
Recall: 56.84%
F1-Score: 54.84%
```