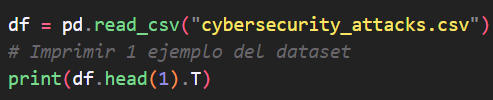
**LAB BOOK**

**1. Extracción de los datos**

Debemos obtener los datos del dataset público de ciberseguridad. Estos vienen en formato CSV. Utilizaremos librería pandas de Python como ‘pd’ para extraer los datos como se muestra en el experimento:



Resultados:

Texto

El contenido generado por IA puede ser incorrecto.

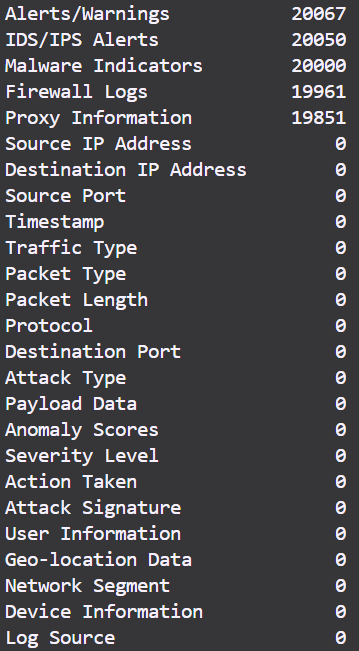
**2. Limpieza de los datos**

La limpieza de los datos nulos o no existentes en algunas columnas del dataset, para ello usaremos:

*# Check for missing values*

print(df.isnull().sum().sort\_values(*ascending*=False))

Resultados:



**3. Transformación de datos no válidos**

Alerts/Warnings, IDS/IPS Alerts, Malware Indicators, Firewall Logs, Proxy Information son las columnas que hemos comprobado que tienen valores no válidos o inexistentes. Por lo que trataremos de limpiarlos o darles un formato válido y universal:

*# Transform the data*

missing\_columns = ['Alerts/Warnings', 'IDS/IPS Alerts', 'Malware Indicators', 'Firewall Logs', 'Proxy Information']

*# fill missing values*

fillvalues = ['None', 'No Data', 'No Detected', 'No Data', 'No Proxy Data']

for i in range(len(fillvalaues)):

    df.fillna({missing\_columns[i]: fillvalaues[i]}, *inplace*=True)

De esta forma los datos no existentes se rellenarán según hemos definido en el vector fillvalues

Resultados:  
Texto

El contenido generado por IA puede ser incorrecto.

**4. Tranformaciones de algunos datos**

Para mejor comprensión de algunas columnas, hemos transformado en más columnas, por ejemplo, la columna de ‘Timestamp’:

*# Rename columns for better understanding*

df.rename(*columns*={'Timestamp':'Datetime'}, *inplace*=True)

df['Datetime'] = df['Datetime'].apply(*lambda* *x*: pd.to\_datetime(x))

*# Transform time information to new columns of the dataframe*

df['year'] = df['Datetime'].dt.year

df['month'] = df['Datetime'].dt.month

df['day'] = df['Datetime'].dt.day

df['dayofweek'] = df['Datetime'].dt.dayofweek

df['hour'] = df['Datetime'].dt.hour

df['minute'] = df['Datetime'].dt.minute

df['second'] = df['Datetime'].dt.second

Esto no es lo único, también hemos transformado ‘Browser’ y ‘Device’ para mejor comprensión de la columna ‘Device Information’.Para ello, usamos librería de expresiones regulares: re de Python.

*def* device\_identifier(*user\_agent*):

    user\_agent = user\_agent.strip()

    for device in devices:

        matching = re.findall(device, user\_agent, re.IGNORECASE)

        if matching:

            return matching[0]

    return 'Unknown Device'

devices = [

*r*'Windows',

*r*'Macintosh',

*r*'Linux',

*r*'iPhone',

*r*'iPod',

*r*'iPad',

*r*'Android'

]

*# Transform 'Device Browser' from 'Device Information'*

df['Browser'] = df['Device Information'].str.split('/').str[0]

*# Transform 'Device' from 'Device Information'*

df['Targeted Device'] = df['Device Information'].apply(device\_identifier)

df['Targeted Device'].unique()

Resultados:

Forma

El contenido generado por IA puede ser incorrecto.

**5. Transformación de tipos de datos**

Para nuestra BBDD relacional MySQL, debemos transformar los objetos de tipo ‘object’ para poder almacenarlo en tablas relacionales.

Para ellos convertiremos los datos a ‘string’ o si llega el caso a ‘int’ o ‘float’.

*# Transformar objetos a tipos para introducir en la base de datos*

df['Source IP Address'] = df['Source IP Address'].astype(pd.StringDtype())

df['Destination IP Address'] = df['Destination IP Address'].astype(pd.StringDtype())

df['Protocol'] = df['Protocol'].astype(pd.StringDtype())

df['Packet Type'] = df['Packet Type'].astype(pd.StringDtype())

df['Traffic Type'] = df['Traffic Type'].astype(pd.StringDtype())

*#df['Payload'] = df['Payload'].astype(pd.StringDtype())*

df['Malware Indicators'] = df['Malware Indicators'].astype(pd.StringDtype())

df['Alerts/Warnings'] = df['Alerts/Warnings'].astype(pd.StringDtype())

df['Attack Signature'] = df['Attack Signature'].astype(pd.StringDtype())

df['Action Taken'] = df['Action Taken'].astype(pd.StringDtype())

*#df['Security Level'] = df['Security Level'].astype(pd.StringDtype())*

df['User Information'] = df['User Information'].astype(pd.StringDtype())

*#df['Device Information'] = df['Device Information'].astype(pd.StringDtype())*

df['Network Segment'] = df['Network Segment'].astype(pd.StringDtype())

df['Geo-location Data'] = df['Geo-location Data'].astype(pd.StringDtype())

df['Proxy Information'] = df['Proxy Information'].astype(pd.StringDtype())

df['Firewall Logs'] = df['Firewall Logs'].astype(pd.StringDtype())

df['IDS/IPS Alerts'] = df['IDS/IPS Alerts'].astype(pd.StringDtype())

df['Log Source'] = df['Log Source'].astype(pd.StringDtype())

\* Como podemos observar, hay varios datos que no hemos podido convertir, en otro experimento los analizaremos más profundamente.

Resultados:

Imagen que contiene texto, tabla, grande, teléfono

El contenido generado por IA puede ser incorrecto.

**5 Loading en una BBDD**

Una vez limpios y transformados los datos, se almacenan en el Data Warehouse. Dependiendo de la infraestructura, podemos usar:

* Bases de Datos Relacionales: PostgreSQL, MySQL, SQL Server.
* Sistemas de Big Data: Amazon Redshift, Google BigQuery, Snowflake.

- Ejemplo de carga en PostgreSQL usando Python:

from sqlalchemy import create\_engine

engine = create\_engine('postgresql://usuario:contraseña@host:puerto/base\_de\_datos')

df.to\_sql('cybersecurity\_data', engine, if\_exists='replace', index=False)