Imagen que contiene dibujo

Descripción generada automáticamente**Icono

Descripción generada automáticamente**

*JAVIER CUARTERO CORREDOR && GONZALO DE LOS REYES*

*LAB BOOK*

DATA WAREHOUSE

Tabla de contenidos:

[**LAB BOOK** 2](#_Toc191310995)

[**1. Extracción de los datos** 2](#_Toc191310996)

[**2. Limpieza de los datos** 2](#_Toc191310997)

[**3. Transformación de datos no válidos** 3](#_Toc191310998)

[**4. Tranformaciones de algunos datos** 5](#_Toc191310999)

[**5. Transformación de tipos de datos** 6](#_Toc191311000)

[**6. Loading en una BBDD** 7](#_Toc191311001)

[**7. Crear BBDD mediante Python** 8](#_Toc191311002)

[**8. Crear tablas mediante Python** 8](#_Toc191311003)

[**9. Inserción de datos en la BBDD** 11](#_Toc191311004)

[**10. Refactorización de código** 13](#_Toc191311005)

[**11. Consultas de nuestro DW** 14](#_Toc191311006)

[**ANEXO** 17](#_Toc191311007)

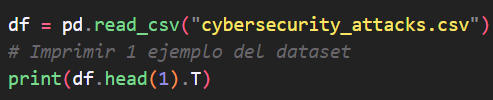
[**Tiempo dedicado** 17](#_Toc191311008)

[**Dificultades encontradas** 17](#_Toc191311009)

# **LAB BOOK**

## **1. Extracción de los datos**

Debemos obtener los datos del dataset público de ciberseguridad. Estos vienen en formato CSV. Utilizaremos la 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.

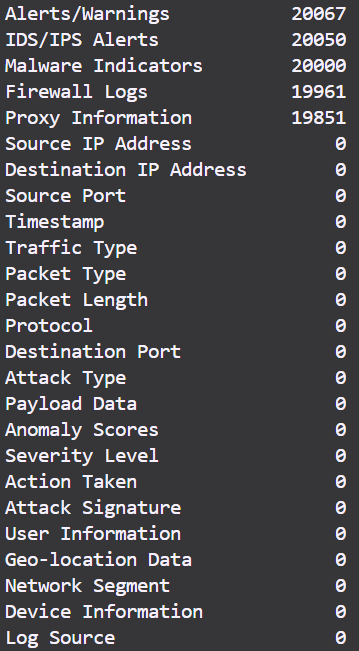
## **2. Limpieza de los datos**

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.

**6. 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.

Tablas de hechosTexto

El contenido generado por IA puede ser incorrecto.

**7. Crear BBDD mediante Python**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Resultados:**

****

**8. Crear tablas mediante Python**

cursor.execute("""

            CREATE TABLE dim\_origen (

                id\_origen INT AUTO\_INCREMENT PRIMARY KEY,

                ip\_origen VARCHAR(50),

                puerto\_origen INT,

                proxy VARCHAR(255),

                usuario VARCHAR(255)

            )

        """)

        cursor.execute("""

            CREATE TABLE dim\_destino (

                id\_destino INT AUTO\_INCREMENT PRIMARY KEY,

                ip\_destino VARCHAR(50),

                puerto\_destino INT,

                dispositivo VARCHAR(255),

                navegador VARCHAR(255)

            )

        """)

        cursor.execute("""

            CREATE TABLE dim\_protocolo (

                id\_protocolo INT AUTO\_INCREMENT PRIMARY KEY,

                protocolo VARCHAR(50)

            )

        """)

        cursor.execute("""

            CREATE TABLE dim\_tipo\_trafico (

                id\_tipo\_trafico INT AUTO\_INCREMENT PRIMARY KEY,

                tipo\_trafico VARCHAR(50)

            )

        """)

        cursor.execute("""

            CREATE TABLE dim\_malware (

                id\_malware INT AUTO\_INCREMENT PRIMARY KEY,

                indicador\_malware VARCHAR(255)

            )

        """)

        cursor.execute("""

            CREATE TABLE dim\_anomalia (

                id\_anomalia INT AUTO\_INCREMENT PRIMARY KEY,

                score\_anomalia FLOAT

            )

        """)

        cursor.execute("""

        CREATE TABLE dim\_severidad (

            id\_severidad INT AUTO\_INCREMENT PRIMARY KEY,

            nivel\_severidad VARCHAR(50)

        )

    """)

        cursor.execute("""

        CREATE TABLE dim\_dispositivo (

            id\_dispositivo INT AUTO\_INCREMENT PRIMARY KEY,

            tipo\_dispositivo VARCHAR(255)

        )

    """)

        cursor.execute("""

        CREATE TABLE dim\_segmento (

            id\_segmento INT AUTO\_INCREMENT PRIMARY KEY,

            segmento VARCHAR(255)

        )

    """)

        cursor.execute("""

        CREATE TABLE dim\_geo (

            id\_geo INT AUTO\_INCREMENT PRIMARY KEY,

            ubicacion VARCHAR(255)

        )

    """)

        cursor.execute("""

        CREATE TABLE hechos\_ataques (

            id\_ataque INT AUTO\_INCREMENT PRIMARY KEY,

            timestamp DATETIME NOT NULL,

            id\_origen INT,

            id\_destino INT,

            id\_protocolo INT,

            id\_tipo\_trafico INT,

            id\_malware INT,

            id\_anomalia INT,

            id\_severidad INT,

            id\_dispositivo INT,

            id\_segmento INT,

            id\_geo INT,

            longitud\_paquete INT,

            numero\_alertas INT,

            FOREIGN KEY (id\_origen) REFERENCES dim\_origen(id\_origen),

            FOREIGN KEY (id\_destino) REFERENCES dim\_destino(id\_destino),

            FOREIGN KEY (id\_protocolo) REFERENCES dim\_protocolo(id\_protocolo),

            FOREIGN KEY (id\_tipo\_trafico) REFERENCES dim\_tipo\_trafico(id\_tipo\_trafico),

            FOREIGN KEY (id\_malware) REFERENCES dim\_malware(id\_malware),

            FOREIGN KEY (id\_anomalia) REFERENCES dim\_anomalia(id\_anomalia),

            FOREIGN KEY (id\_severidad) REFERENCES dim\_severidad(id\_severidad),

            FOREIGN KEY (id\_dispositivo) REFERENCES dim\_dispositivo(id\_dispositivo),

            FOREIGN KEY (id\_segmento) REFERENCES dim\_segmento(id\_segmento),

            FOREIGN KEY (id\_geo) REFERENCES dim\_geo(id\_geo)

        )

    """)

        cursor.execute("SHOW TABLES")

        tables = cursor.fetchall()

        print("📋 Tablas en la base de datos:")

        for table in tables:

            print(table[0])

        print("✅ Tablas creadas con éxito en 'ciberseguridad\_db'.")

        conn.commit()

**Resultados:**

**Tabla

El contenido generado por IA puede ser incorrecto.**

**9. Inserción de datos en la BBDD**

*def* insert\_dim\_data(*cursor*, *table*, *columns*, *data*):

    query = *f*"""

        INSERT IGNORE INTO {table} ({', '.join(columns)})

        VALUES ({', '.join(['%s'] \* len(columns))})

    """

    cursor.executemany(query, data)

*def* load\_dimensions(*df*):

    conn = connect\_db()

    cursor = conn.cursor()

    insert\_dim\_data(cursor, "dim\_origen", ["ip\_origen", "puerto\_origen", "proxy", "usuario"], df[['Source IP Address', 'Source Port', 'Proxy Information', 'User Information']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_destino", ["ip\_destino", "puerto\_destino", "dispositivo", "navegador"], df[['Destination IP Address', 'Destination Port', 'Targeted Device', 'Browser']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_protocolo", ["protocolo"], df[['Protocol']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_tipo\_trafico", ["tipo\_trafico"], df[['Traffic Type']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_malware", ["indicador\_malware"], df[['Malware Indicators']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_anomalia", ["score\_anomalia"], df[['Anomaly Scores']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_severidad", ["nivel\_severidad"], df[['Severity Level']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_dispositivo", ["tipo\_dispositivo"], df[['Targeted Device']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_segmento", ["segmento"], df[['Network Segment']].drop\_duplicates().values.tolist())

    insert\_dim\_data(cursor, "dim\_geo", ["ubicacion"], df[['Geo-location Data']].drop\_duplicates().values.tolist())

    conn.commit()

    cursor.close()

    conn.close()

Resultados las tablas llenadas de los datos limpiados.

Finalmente, insertamos la tabla de hechos en la BBDD:

*def* get\_dimension\_id(*cursor*, *table*, *id\_column*, *column*, *value*):

    cursor.execute(*f*"SELECT {id\_column} FROM {table} WHERE {column} = %s LIMIT 1", (value,))

    result = cursor.fetchone()

    return result[0] if result else None

*def* insert\_fact\_table(*df*):

    conn = connect\_db()

    cursor = conn.cursor()

    batch\_data = []

    for index, row in df.iterrows():

        if index % 500 == 0:

            print(*f*"⏳ Procesando fila {index}...")

        ids = {

            "id\_origen": get\_dimension\_id(cursor, 'dim\_origen', 'id\_origen', 'ip\_origen', row['Source IP Address']),

            "id\_destino": get\_dimension\_id(cursor, 'dim\_destino', 'id\_destino', 'ip\_destino', row['Destination IP Address']),

            "id\_protocolo": get\_dimension\_id(cursor, 'dim\_protocolo', 'id\_protocolo', 'protocolo', row['Protocol']),

            "id\_tipo\_trafico": get\_dimension\_id(cursor, 'dim\_tipo\_trafico', 'id\_tipo\_trafico', 'tipo\_trafico', row['Traffic Type']),

            "id\_malware": get\_dimension\_id(cursor, 'dim\_malware', 'id\_malware', 'indicador\_malware', row['Malware Indicators']),

            "id\_anomalia": get\_dimension\_id(cursor, 'dim\_anomalia', 'id\_anomalia', 'score\_anomalia', row['Anomaly Scores']),

            "id\_severidad": get\_dimension\_id(cursor, 'dim\_severidad', 'id\_severidad', 'nivel\_severidad', row['Severity Level']),

            "id\_dispositivo": get\_dimension\_id(cursor, 'dim\_dispositivo', 'id\_dispositivo', 'tipo\_dispositivo', row['Targeted Device']),

            "id\_segmento": get\_dimension\_id(cursor, 'dim\_segmento', 'id\_segmento', 'segmento', row['Network Segment']),

            "id\_geo": get\_dimension\_id(cursor, 'dim\_geo', 'id\_geo', 'ubicacion', row['Geo-location Data'])

        }

        if None not in ids.values():

            batch\_data.append((row['Datetime'], \*ids.values(), row['Packet Length'], row['Alerts/Warnings']))

    if batch\_data:

        print("⏳ Insertando datos en hechos\_ataques...")

        cursor.executemany("""

            INSERT INTO hechos\_ataques (timestamp, id\_origen, id\_destino, id\_protocolo, id\_tipo\_trafico, id\_malware, id\_anomalia, id\_severidad, id\_dispositivo, id\_segmento, id\_geo, longitud\_paquete, numero\_alertas)

            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

        """, batch\_data)

    conn.commit()

    cursor.close()

    conn.close()

    print("✅ Carga de hechos completada.")

Y todos los datos habrán sido introducidos en nuestra BBDD MySQL.

**10. Refactorización de código**

**Texto

El contenido generado por IA puede ser incorrecto.**

**Resultados:**

Un código más elegible que podrá ser mejor entendido por usuarios o compañeros de trabajo.

## **11. Consultas de nuestro DW**

Para realizar nuestras consultas, nos apoyaremos otra vez en Python para lograr ver de manera clara con la librería matplotlib datos de nuestro almacén de datos con valor para ayudarnos a entenderla:

**Resultados:**

**Trafico más frecuente:**

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

**Direcciones IP con más ataques:**

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

Aquí debemos comentar que como podemos observar no es muy útil esta consulta debido a que nuestro dataset contiene todas las ip de ataques distintas. Pero en una empresa en la que tenemos un par de ips, nos interesaría saber por donde nos atacan más porque quizás es una red más débil o menos protegida.

**Segmentos de red más atacados:**

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

Como podemos observar se atacan todos los segmentos de red casi en igual cantidad.

**Ataques por nivel de Severidad:**

Gráfico, Gráfico circular

El contenido generado por IA puede ser incorrecto.

Como podemos observar, al igual que los segmentos de red, no hay un claro vencedor de nivel de severidad de ataque.

# **ANEXO**

## **Tiempo dedicado**

El tiempo total sumando las horas de ambos compañeros (Gonzalo y Javier) ha sido de 20 horas.

## **Dificultades encontradas**

Una de las dificultades en la que más tiempo hemos dedicado es a la hora de insertar datos en la BBDD con MySQL: Hemos tenido que limpiar todos los datos y convertirlos en tipos soportados por MySQL ya que no podíamos guardar los objetos que teníamos en el dataframe con pandas.

Además, al tener distinto sistema operativo (Windows y Mac OS) hemos encontrado un problema al insertar datos de tipo STRING. Finalmente pudimos resolverlo revisando las columnas de las tablas y viendo si coincidían en nombre al introducir los datos.

Como ultimo problema comentar que hemos intentado insertar datos de manera paralela con threads, pero no hemos obtenido el resultado de rendimiento mejorado esperado, por lo que hemos optado por hacer rollback y dejar lo que funcionaba (aunque tarde 5 minutos en insertar completamente los datos en la tabla de hechos).