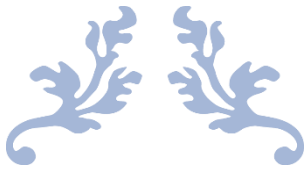


PRÁCTICA 1
RECOPILACIÓN, ESTRUCTURACIÓN Y ANÁLISIS DE DATOS

RELIZADA POR:
PABLO PASTOR, PABLO REDONDO, GABRIEL MEDRANO

ÍNDICE:

1. **Github:**

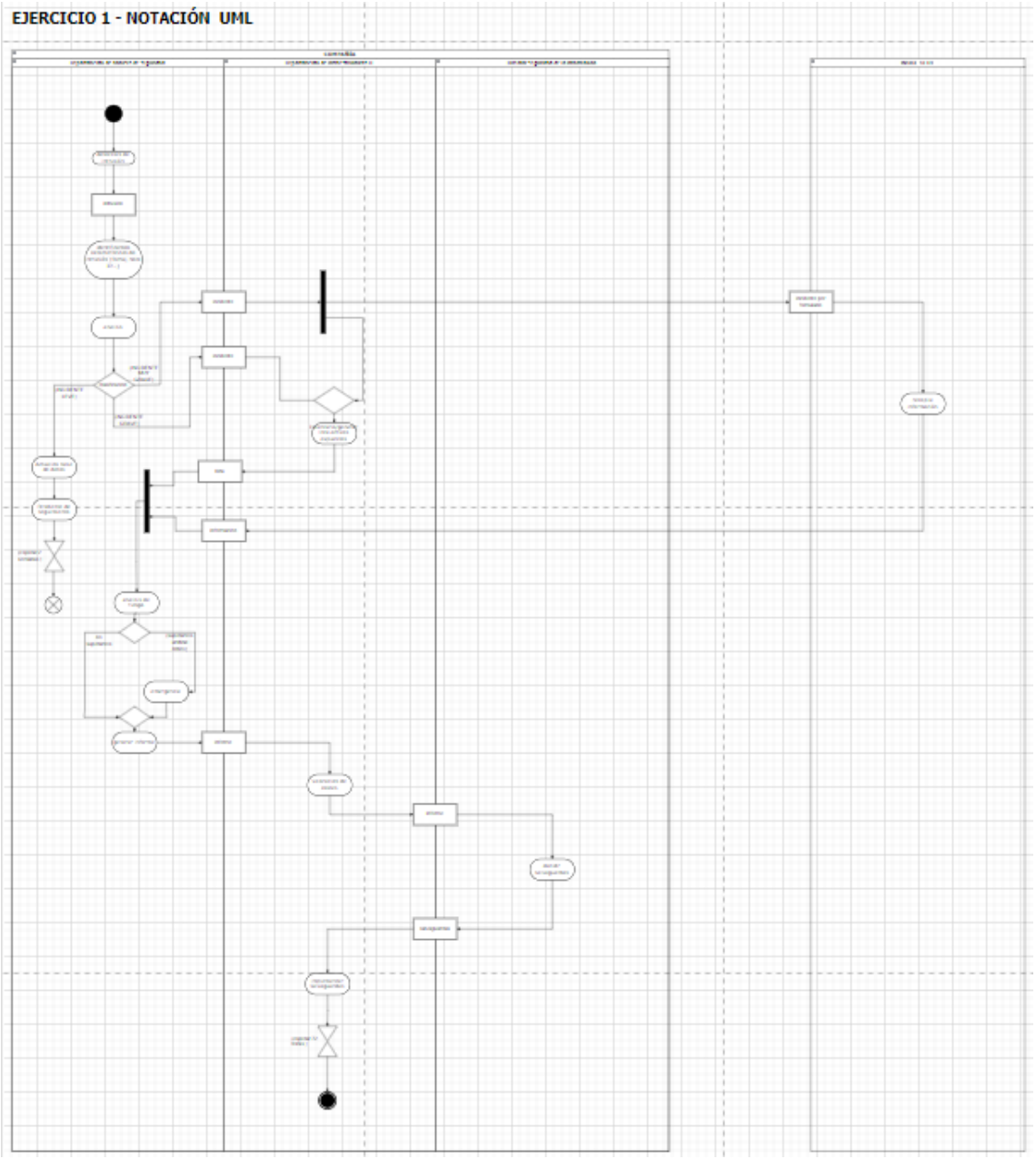
Link al proyecto común de github:

https://github.com/medranoGG/SI_Proyects

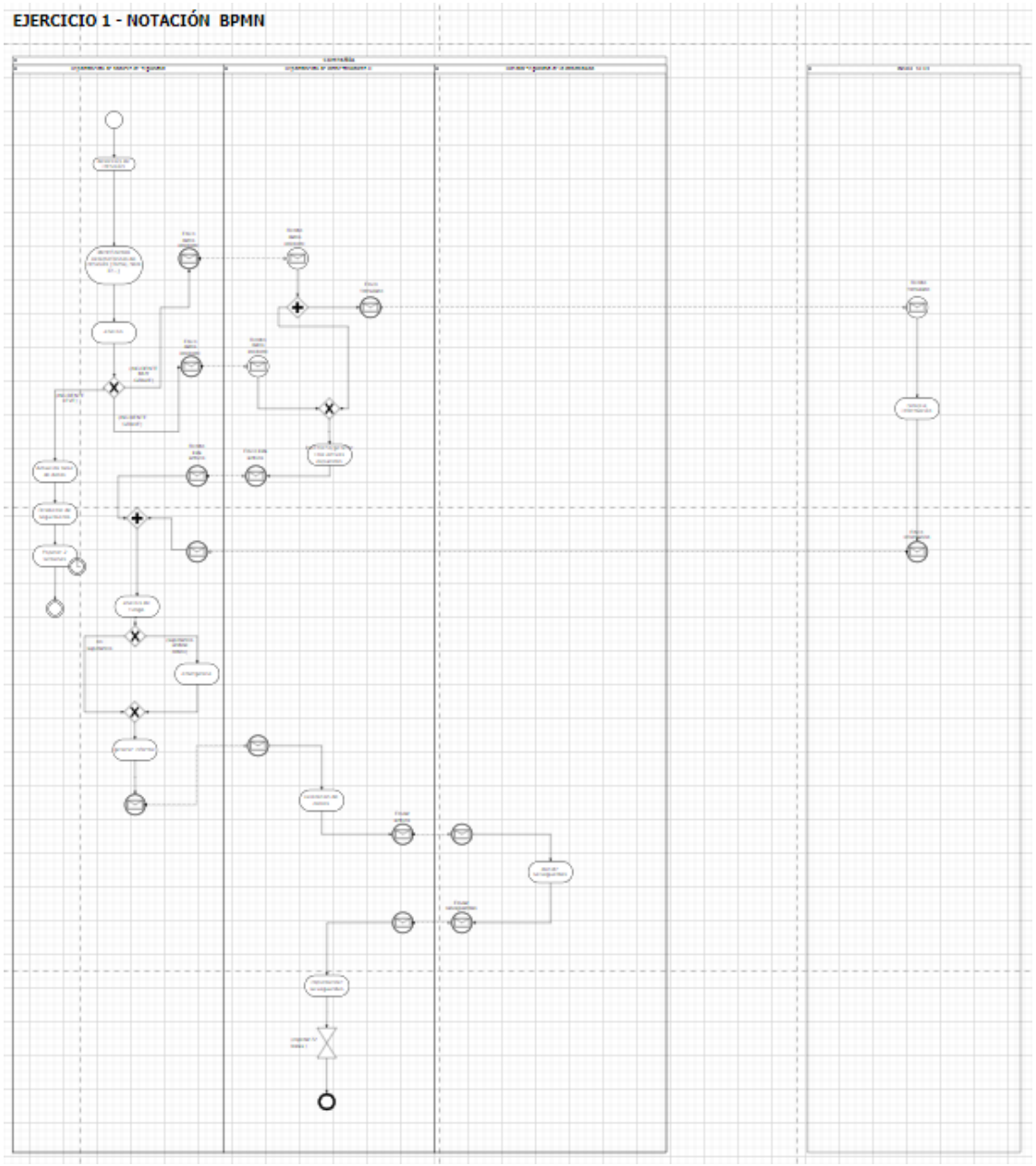
2. Ejercicio 1:

Desarrollaremos los dos modelos de negocio con la herramienta https://app.diagrams.net/. Entregados en el archivo ***Ejercicio1.drawio***.

Notación **UML**:

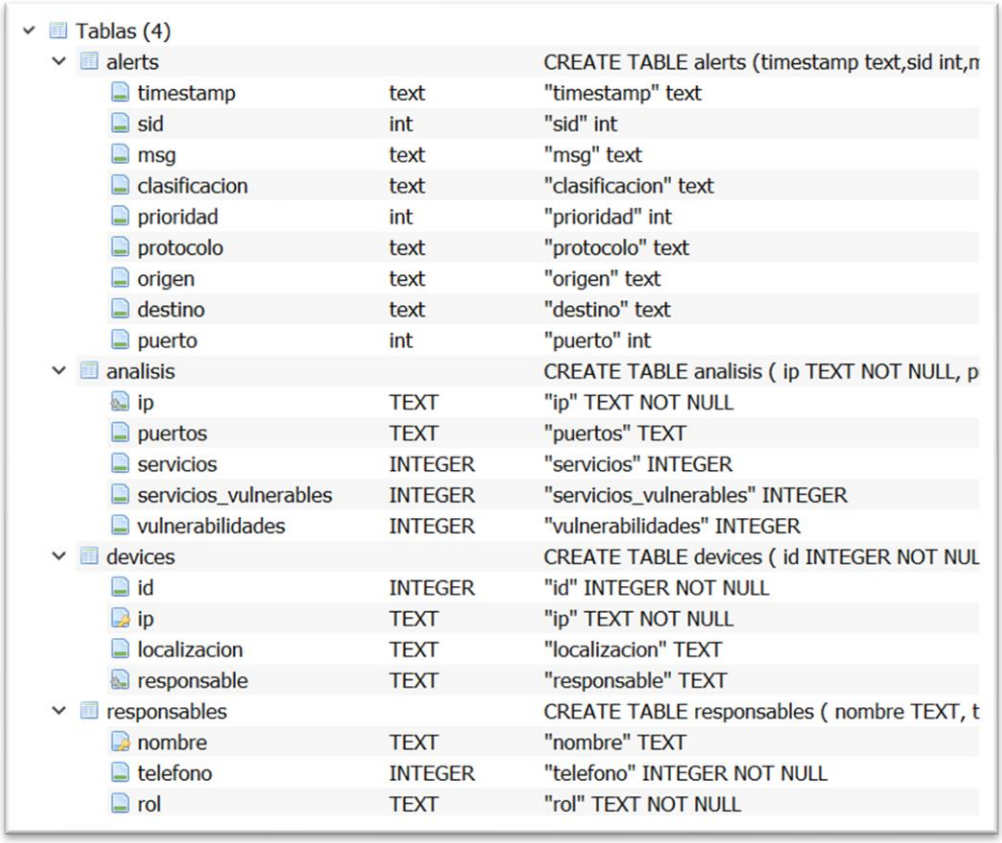


Notación **BPMN**:



3. Ejercicio 2:

En este ejercicio, leeremos y almacenaremos todos los archivos del conjunto de datos del enunciado en nuestra base de datos con el fichero ***files_to_db.py.*** En este, mediante los imports de *sqlite3, pandas y json* hemos volcado la información de los archivos dentro de nuestra base de datos ***base.db***, que tiene toda la información necesaria:



Tablas (4)		
alerts		CREATE TABLE alerts (timestamp text,sid int,n
timestamp	text	"timestamp" text
sid	int	"sid" int
msg	text	"msg" text
clasificacion	text	"clasificacion" text
prioridad	int	"prioridad" int
protocolo	text	"protocolo" text
origen	text	"origen" text
destino	text	"destino" text
puerto	int	"puerto" int
analisis		CREATE TABLE analisis (ip TEXT NOT NULL, p
ip	TEXT	"ip" TEXT NOT NULL
puertos	TEXT	"puertos" TEXT
servicios	INTEGER	"servicios" INTEGER
servicios_vulnerables	INTEGER	"servicios_vulnerables" INTEGER
vulnerabilidades	INTEGER	"vulnerabilidades" INTEGER
devices		CREATE TABLE devices (id INTEGER NOT NUL
id	INTEGER	"id" INTEGER NOT NULL
ip	TEXT	"ip" TEXT NOT NULL
localizacion	TEXT	"localizacion" TEXT
responsable	TEXT	"responsable" TEXT
responsables		CREATE TABLE responsables (nombre TEXT, t
nombre	TEXT	"nombre" TEXT
telefono	INTEGER	"telefono" INTEGER NOT NULL
rol	TEXT	"rol" TEXT NOT NULL

Ahora, leeremos los datos de nuestra ***base.db*** con las siguientes consultas almacenadas en dataframes. Para la creación de los dataframes, hemos usado la lib ***pandas***:

1. device_count.py -> Consulta número de dispositivos (y campos missing o None):

Creamos query: *(dataframe punto 1)*

```
# Querys to "devices" table
query_devices = 'SELECT ip FROM devices'
```

Salida de datos:

```
Número total de dispositivos:
7

Process finished with exit code 0
```

2. port_count.py -> Consulta media, desviación estándar y valor mínimo y máximo del total de puertos abiertos: *(en un mismo dataframe añadimos el punto 3 y el 6)*

Creamos query:

```
# Querys to "devices" table
query_puertos = 'SELECT puertos FROM analisis'
```

Calculamos datos:

```
# Calculate mean and standard dev. of open ports
media = df_puertos['puertos'].apply(len).mean()
desv_tipica = df_puertos['puertos'].apply(len).std()
max = df_puertos['puertos'].apply(lambda x: len(x)).max()
min = df_puertos['puertos'].apply(lambda x: len(x)).min()
```

Salida de datos:

```
Dataframe análisis de los puertos abiertos encontrados

  Media  Desv. estándar  Max  Min
    3.0        1.632993    5    0

Process finished with exit code 0
```

3. service_count.py -> Media y desviación estándar del número de servicios inseguros detectados. Media, desviación estándar y valor mínimo y máximo del número de vulnerabilidades detectadas: ()

Creamos query:

```
# Querys to "analisis" table
query_servicios = "SELECT servicios FROM analisis"
query_servicios_vulnerables = "SELECT servicios_vulnerables FROM analisis"
query_vulnerabilidades = "SELECT vulnerabilidades FROM analisis"
```

Servicios:

Calculamos datos servicios:

```
# Calculate mean and standard dev. of services
media = df_servicios['servicios'].mean()
desv_tipica = df_servicios['servicios'].std()
```

Calculamos datos servicios vulnerables:

```
# Calculate mean and standard dev. of vulnerable services
media = df_servicios_vulnerables['servicios_vulnerables'].mean()
desv_tipica = df_servicios_vulnerables['servicios_vulnerables'].std()
```

Vulnerabilidades:

Calculamos datos vulnerabilidades:

```
# Calculate mean,standard dev. max & min of the vulnerabilities found
media = df_vulnerabilidades['vulnerabilidades'].mean()
desv_tipica = df_vulnerabilidades['vulnerabilidades'].std()
max = df_vulnerabilidades['vulnerabilidades'].max()
min = df_vulnerabilidades['vulnerabilidades'].min()
```

Salida de datos:

```
Análisis de los servicios encontrados

   Media  Desv. estándar
1.714286        1.112697


Análisis de los servicios vulnerables encontrados

   Media  Desv. estándar
0.714286         0.95119


Análisis de las vulnerabilidades encontradas

    Media  Desv. estándar  Max  Min
15.571429       17.539072   52    2
```

4. <u>alert_count.py</u> -> Numero de alertas: *(dataframe punto 2)*

Creamos query:

```
# Querys to "devices" table
query_alerts = 'SELECT sid FROM alerts'
```

Salida de datos:

```
Número total de alertas
200225

Process finished with exit code 0
```

FUNCIONES RELEVANTES UTILIZADAS:

Hemos necesitado utilizad diversas funciones, algunas de las más relevantes son:

- *pd.read_sql_query()*
- *Dataframe.shape[]*
- *Dataframe.apply(lambda x: eval(x) if x != None else []) Porque uno de los valores "None" no se cuenta como entero*
- *.mean()*
- *.std()*
- *Dataframe.drop()*
- *Dataframe.rename()*
- *Dataframe.fillna()*
- *pd.merge()*

Entre otras

4. Ejercicio 3:

En este ejercicio, analizaremos las alertas mediante agrupaciones de datos en función de su prioridad y su fecha. Con este sistema, daremos un sentido a nuestro análisis. Las agrupaciones han sido más sencillas puesto que se podía hacer la selección desde las propias consultas SQL

Hemos creado dos ficheros Python en la carpeta **Agrupaciones**, donde guardaremos lo siguiente:

1. prioridad.py -> Donde agruparemos los datos por prioridad de alerta, donde 1 son las graves, 2 las medianas y 3 las bajas:

Creamos querys:

```
# Create the querys to the SQLite3
query_dataframe_prioridad_1 = 'SELECT * FROM alerts WHERE prioridad == 1'

query_dataframe_prioridad_2 = 'SELECT * FROM alerts WHERE prioridad == 2'

query_dataframe_prioridad_3 = 'SELECT * FROM alerts WHERE prioridad == 3'
```

Creamos dataframes con la conexión a la base de datos:

```
# Create the dataframes using pandas
df_prioridad_1 = pd.read_sql_query(query_dataframe_prioridad_1, conn)
df_prioridad_2 = pd.read_sql_query(query_dataframe_prioridad_2, conn)
df_prioridad_3 = pd.read_sql_query(query_dataframe_prioridad_3, conn)
```

Salida de datos mayor prioridad:

```
Alteras de mayor prioridad
                timestamp      sid  ...     destino puerto
0     2022-07-03 01:18:27  2018056  ...  172.18.0.0     80
1     2022-07-03 01:18:27  2031562  ...  172.18.0.0     80
2     2022-07-03 01:18:27  2037040  ...  172.18.0.0     80
3     2022-07-03 03:57:41  2011465  ...  172.18.0.0     80
4     2022-07-03 03:57:41  2034125  ...  172.18.0.0     80
...                   ...      ...  ...         ...    ...
2586  2022-09-05 00:50:30  2010698  ...  172.18.0.0     80
2587  2022-09-05 00:50:30  2019309  ...  172.18.0.0     80
2588  2022-09-05 00:50:30  2020899  ...  172.18.0.0     80
2589  2022-09-05 03:26:30  2006402  ...  172.18.0.0     80
2590  2022-09-05 03:26:30  2011669  ...  172.18.0.0     80
```

Salida de datos media prioridad:

```
Alertas de media prioridad
                timestamp      sid  ...     destino puerto
0     2022-07-03 00:19:55  2402000  ...  172.19.0.0   1194
1     2022-07-03 00:49:22  2016683  ...  172.18.0.0     80
2     2022-07-03 01:00:12  2402000  ...  172.18.0.0    443
3     2022-07-03 01:18:44  2016683  ...  172.18.0.0     80
4     2022-07-03 01:23:35  2031502  ...  172.18.0.0     80
...                   ...      ...  ...         ...    ...
6580  2022-09-05 03:44:19  2403320  ...  172.18.0.0    443
6581  2022-09-05 03:49:41  2016683  ...  172.18.0.0     80
6582  2022-09-05 04:37:03  2402000  ...  172.18.0.0     80
6583  2022-09-05 04:44:37  2016683  ...  172.18.0.0     80
6584  2022-09-05 05:50:56  2402000  ...  172.18.0.0     80
```

Salida de datos baja prioridad:

```
Alertas de baja prioridad
                 timestamp      sid  ...      destino puerto
0       2022-07-03 00:49:22  2221045  ...   172.18.0.0     80
1       2022-07-03 00:49:22  2221010  ...   1.188.64.0  48224
2       2022-07-03 00:50:30  2221010  ...   60.30.98.0  47325
3       2022-07-03 01:18:44  2221045  ...   172.18.0.0     80
4       2022-07-03 01:18:44  2221010  ...  121.7.228.0  60193
...                     ...      ...  ...          ...    ...
191044  2022-09-05 04:35:13  2029054  ...   172.18.0.0     80
191045  2022-09-05 04:44:37  2221045  ...   172.18.0.0     80
191046  2022-09-05 04:44:37  2221010  ...  45.191.79.0  53395
191047  2022-09-05 05:39:31  2260002  ...  111.7.100.0  65477
191048  2022-09-05 05:39:31  2230018  ...   172.18.0.0    443
```

2. <u>fecha.py</u> -> Donde agruparemos los datos según fechas definidas. En este caso, estableceremos el rango del mes de julio y del mes de agosto:

Creamos query:

```python
# Query the database to get all the data
df = pd.read_sql_query("SELECT * FROM alerts", conn)
```

Filtramos por mes respectivamente (julio = 7, agosto = 8):

```python
# Filter the data to get alerts from July and August
july_df = df[df['timestamp'].dt.month == 7]
august_df = df[df['timestamp'].dt.month == 8]
```

Salida de datos mes de Julio:

```
Alertas recibidas en Julio
                timestamp      sid  ...     destino puerto
0     2022-07-03 00:19:55  2402000  ...  172.19.0.0   1194
1     2022-07-03 00:49:22  2221045  ...  172.18.0.0     80
2     2022-07-03 00:49:22  2016683  ...  172.18.0.0     80
3     2022-07-03 00:49:22  2221010  ...  1.188.64.0  48224
4     2022-07-03 00:50:30  2221010  ...  60.30.98.0  47325
...                   ...      ...  ...         ...    ...
11485 2022-07-31 23:31:09  2403374  ...  172.18.0.0    443
11486 2022-07-31 23:34:41  2031499  ...  172.18.0.0     80
11487 2022-07-31 23:49:06  2029054  ...  172.18.0.0     80
11488 2022-07-31 23:55:02  2403341  ...  172.18.0.0    443
11489 2022-07-31 23:57:17  2402000  ...  172.18.0.0     80
```

Salida de datos mes de Agosto:

```
Alertas recibidas en Agosto
                timestamp      sid  ...     destino puerto
11490  2022-08-01 00:02:15  2029054  ...  172.18.0.0     80
11491  2022-08-01 00:18:38  2402000  ...  172.19.0.0   1194
11492  2022-08-01 00:54:08  2018056  ...  172.18.0.0     80
11493  2022-08-01 00:54:08  2031562  ...  172.18.0.0     80
11494  2022-08-01 00:54:08  2037040  ...  172.18.0.0     80
...                   ...      ...  ...         ...    ...
199246 2022-08-31 23:48:13  2029022  ...  172.18.0.0     80
199247 2022-08-31 23:48:13  2030093  ...  172.18.0.0     80
199248 2022-08-31 23:56:26  2221045  ...  172.18.0.0     80
199249 2022-08-31 23:56:26  2016683  ...  172.18.0.0     80
199250 2022-08-31 23:56:27  2221010  ...  45.191.79.0  54271
```

Ahora, en la carpeta **Análisis vulnerabilidades** encontramos el fichero ***vulnerabilidades.py***, el cual nos calculará el número de observaciones y de valores ausentes, la media, la desviación típica, la varianza y los valores máximos y mínimos para la variable de vulnerabilidades detectada en los dispositivos, donde;

Hemos creado un Dataframe con todos los campos de nuestra tabla "análisis". Indexando a través de la columna vulnerabilidades hemos extraido los datos de la misma y realizado las operaciones pertinentes:

Calculamos query:

```
# Querys to "analisis" table
query_vulns = 'SELECT vulnerabilidades FROM analisis'
```

Almacenamos en dataframe con la conexión a la basede datos:

```
# Read the querys into the dataframe
df_vulns = pd.read_sql_query(query_vulns, conn)
```

Salida de datos vulnerabilidades:

```
Numero de observaciones:  7
Numero de valores ausentes:  0
Mediana:  12.0
Media:  15.571428571428571
Varianza:  307.61904761904765
Maximo:  52  Minimo:  2
```

5. Ejercicio 4:

En este último ejercicio generaremos gráficos para los siguientes casos, con la librería ***matplotlib***.

Para la creación de estos gráficos, hemos creado dataframes por cada problema. Mediante la librería matplotlib accedemos a los diferentes campos del gráfico para personalizarlo - Título - Descripción - Color – etc

1. top_ips.py -> Mostramos y representamos en gráfica las 10 primeras IPs de origen más problemáticas, es decir, las de prioridad 1.

Creamos query:

```
# Querys to "alerts" table
query_alert = 'SELECT * FROM alerts'
```

Seleccionamos las alertas de prioridad 1, filtramos por origen y nos guardamos las 10 primeras:

```
# Select priority 1
df_p1 = df_alerts[df_alerts['prioridad'] == 1]

# Count the aparitions of the 'origen'IP column
df_counts = df_p1.groupby('origen').size().reset_index(name='counts')

# Select top 10
df_top10 = df_counts.nlargest(10, 'counts')
```

Salida de datos de las 10 IPs:

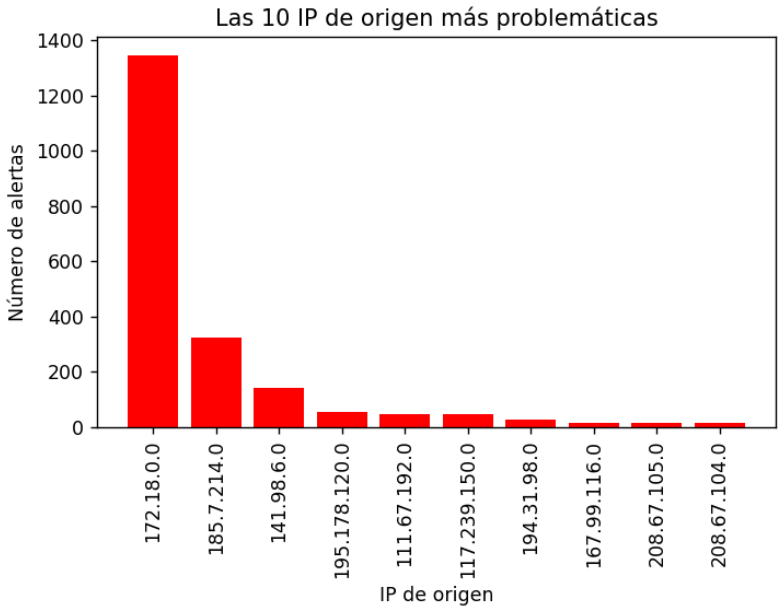
```
            origen  counts
85      172.18.0.0    1345
102    185.7.214.0     322
63      141.98.6.0     140
112  195.178.120.0      54
18    111.67.192.0      44
42   117.239.150.0      44
110    194.31.98.0      26
80    167.99.116.0      14
144   208.67.105.0      14
143   208.67.104.0      13

Process finished with exit code 0
```

Con ***plt*** creamos el gráfico de barras y lo mostramos:

```
# Crear un graph (simple bar graph with titles)
plt.bar(df_top10['origen'], df_top10['counts'], color='red')
plt.xticks(rotation=90)
plt.xlabel('IP de origen')
plt.ylabel('Número de alertas')
plt.title('Las 10 IP de origen más problemáticas')
plt.show()
```

Gráfico final:



2. <u>alerts_temporal.py</u> -> Mostramos en el gráfico el número de alertas en función de su fecha. Clasificación temporal.

Creamos query:

```
# Querys to "alerts" table
query_alert = 'SELECT * FROM alerts'
```

Filtramos en función del tiempo y prioridad D:

```python
# Change dates to date time
df_alerts['timestamp'] = pd.to_datetime(df_alerts['timestamp'])

# Set the index as the timestamp
df_alerts = df_alerts.set_index('timestamp')

# Group x day and count
alerts_per_day = df_alerts['prioridad'].resample('D').count()
```

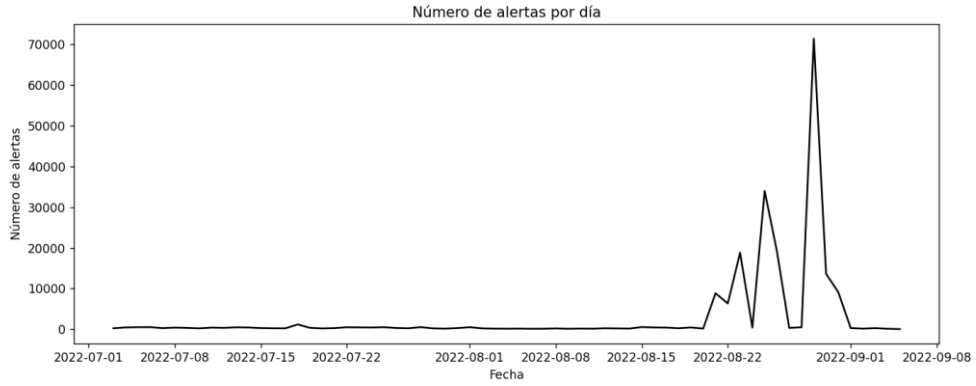
Salida de datos:

```
timestamp
2022-07-03    253
2022-07-04    469
2022-07-05    522
2022-07-06    536
2022-07-07    296
              ...
2022-09-01    307
2022-09-02    173
2022-09-03    293
2022-09-04    139
2022-09-05     62
```

Usamos ***plt*** para crear nuestra gráfica:

```python
# Create a graph
plt.plot(alerts_per_day.index, alerts_per_day.values, color='black')
plt.xlabel('Fecha')
plt.ylabel('Número de alertas')
plt.title('Número de alertas por día')
plt.show()
```

Gráfico final:



3. <u>alerts_type.py</u> -> Mostramos en el gráfico el número de alertas en función de la clasificación de las alertas.

Creamos query:

```
# Querys to "alerts" table
query_alert = 'SELECT * FROM alerts'
```

Conectamos la query a la base de datos con pandas, y filtramos en función de la clasificación:

```
# Read the query into a df
df_alerts = pd.read_sql_query(query_alert, conn)

# Group by the classification and count
alertas_por_clasificacion = df_alerts.groupby('clasificacion')['msg'].count()
```

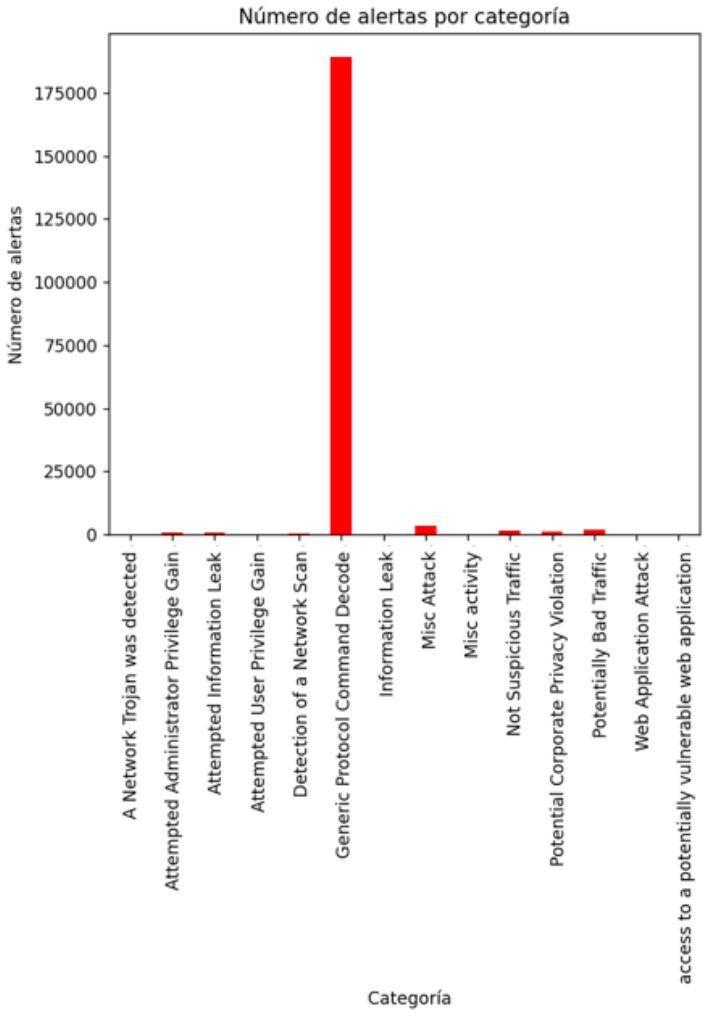
Salida de datos:

```
clasificacion
A Network Trojan was detected                          143
Attempted Administrator Privilege Gain                 742
Attempted Information Leak                             978
Attempted User Privilege Gain                           62
Detection of a Network Scan                            370
Generic Protocol Command Decode                     188985
Information Leak                                        14
Misc Attack                                           3505
Misc activity                                            2
Not Suspicious Traffic                                1692
Potential Corporate Privacy Violation                 1378
Potentially Bad Traffic                               1950
Web Application Attack                                 266
access to a potentially vulnerable web application     138
Name: msg, dtype: int64
```

Con ***plt*** creamos el gráfico en función de la variable filtrada anteriormente y lo mostramos:

```
# Create a graph (simple bar graph)
alertas_por_clasificacion.plot(kind='bar', color='red')
plt.title('Número de alertas por categoría')
plt.xlabel('Categoría')
plt.ylabel('Número de alertas')
plt.show()
```

Gráfico final:



4. <u>most_vulned_devides.py</u> -> En este código, mostramos los dispositivos vulnerables en un gráfico. Tanto los servicios vulnerables como las vulnerabilidades detectadas.

Creamos query:

```
# Querys to "devices" table & "alerts" table
query_analisis = 'SELECT ip, servicios_vulnerables, vulnerabilidades FROM analisis'
```

Metemos la query en el dataframe y filtramos todos los datos necesarios para el gráfico:

```
# Create a new column with the sum of vulnerabilities
df_analisis['Total Vulnerabilities'] = df_analisis['servicios_vulnerables'] + df_analisis['vulnerabilidades']
df_analisis = df_analisis.sort_values(by='Total Vulnerabilities', ascending=False)
df_analisis.rename(columns={'ip': 'IP', 'servicios_vulnerables': 'Vunerable Services', 'vulnerabilidades': 'Vulnerbility number'}, inplace=True)
```

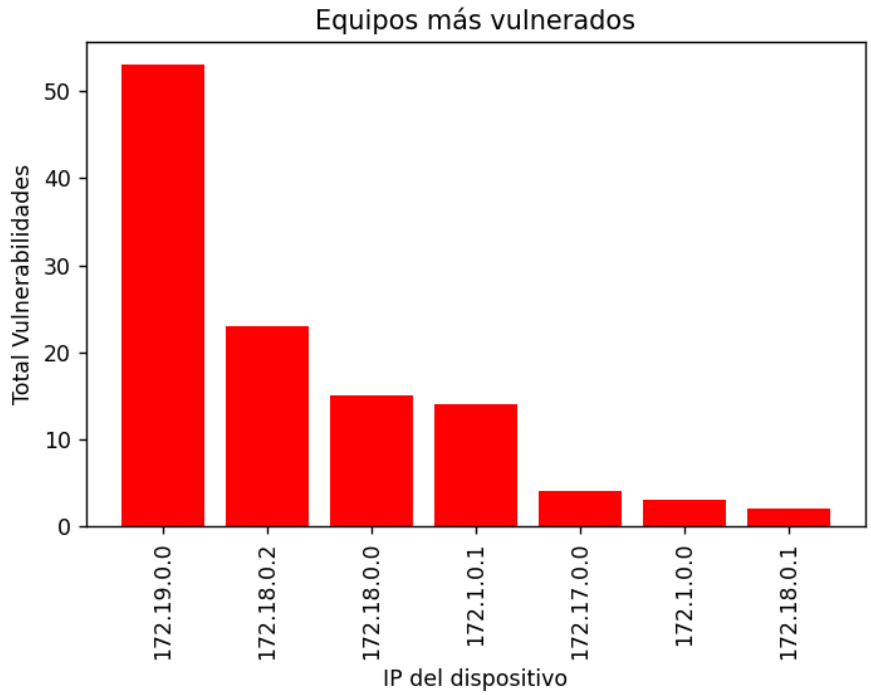
Salida de datos:

	IP	Vunerable Services	Vulnerbility number	Total Vulnerabilities
2	172.19.0.0	1	52	53
6	172.18.0.2	2	21	23
0	172.18.0.0	0	15	15
4	172.1.0.1	2	12	14
1	172.17.0.0	0	4	4
3	172.1.0.0	0	3	3
5	172.18.0.1	0	2	2

Usamos ***plot*** para crear y mostrar nuestro gráfico:

```
# Create a graph (simple bar chart)
plt.bar(df_analisis['IP'], df_analisis['Total Vulnerabilities'], color='red')
plt.xticks(rotation=90)
plt.xlabel('IP del dispositivo')
plt.ylabel('Total Vulnerabilidades')
plt.title('Equipos más vulnerados')
plt.show()
```

Gráfico final:



5. puertos_vulns.py -> Mostramos la media de puertos abiertos frente a servicios inseguros y frente al total de servicios detectados en un único gráfico.

Creamos query:

```
# Querys to "devices" table & "alerts" table
query_analisis = 'SELECT * FROM analisis'
```

Creamos dataframe:

```
# Dataframe of the ports
df_puertos = pd.read_sql_query(query_puertos, conn)
df_puertos['puertos'] = df_analisis['puertos'].apply(lambda x: eval(x) if x != None else [])
df_puertos['numero_puertos'] = df_puertos['puertos'].apply(lambda x: len(x))
```

Configuramos ejes de gráfico:

```
# Data configure
x = df_puertos['numero_puertos']
y1 = df_analisis['servicios_vulnerables']
y2 = df_analisis['servicios']
```

Usamos ***plot*** para crear y mostrar nuestro gráfico:

```
# Graph configuration
fig, ax = plt.subplots(figsize=(8, 5))
ax.set_title('Media de puertos abiertos frente a servicios inseguros y frente al total de servicios detectados')
ax.set_xlabel('Media de puertos abiertos')
ax.set_ylabel('Número de servicios')
ax.grid(True)

# Data plotting
ax.bar(x, y2, color='red', label='Total de servicios')
ax.stem(x, y1, markerfmt='black', basefmt='black', label='Servicios inseguros')
```

Gráfico final: