PRÁCTICA 1

Recopilación, estructuración y análisis de datos

Sistemas de Información

2022-2023

Ignacio Hermosa López, Nicole Reyes Rivero y Marina Parra Martínez

Grupo M

Contenido

1. Introducción

En esta práctica realizaremos y analizaremos diversos procesos de un sistema de información de un negocio de seguridad. En primer lugar, modelaremos el proceso gestión de alertas de seguridad detectadas en un IDS de su sistema de producción, esto lo haremos con las notaciones UML y BPMN vistos en las clases de teoría.

A continuación, desarrollaremos un sistema de información gerencial usando una base de datos SQLite y realizaremos diversas operaciones sobre los datos, entre ellas, extracción de información, agrupación y creación de graficas con la información de la base de datos.

Hemos realizado la práctica en un repositorio de GitHub, ahí tenemos ambos modelos de negocio en un archivo .drawio, el proyecto de PyCharm con todos los ejercicios implementados en el archivo main.py, esta memoria y el archivo SQLite de la base de datos.

https://github.com/ignaciohl/practicaSSII

2. Ejercicio 1

Realizaremos dos diagramas de modelo de negocio para la empresa, usaremos los modelos de UML y BPMN.

La acción para modelar será el análisis del proceso por el cual se gestionan las alertas de seguridad en la empresa, como implicados encontramos dos departamentos y la dirección de seguridad de la empresa y ajena a ella, el centro de respuesta a incidentes de seguridad INCIBE.

Hemos subido los diagramas al repositorio GitHub dedicado a la práctica, con el nombre **Práctica1DIAGRAMAS.drawio**

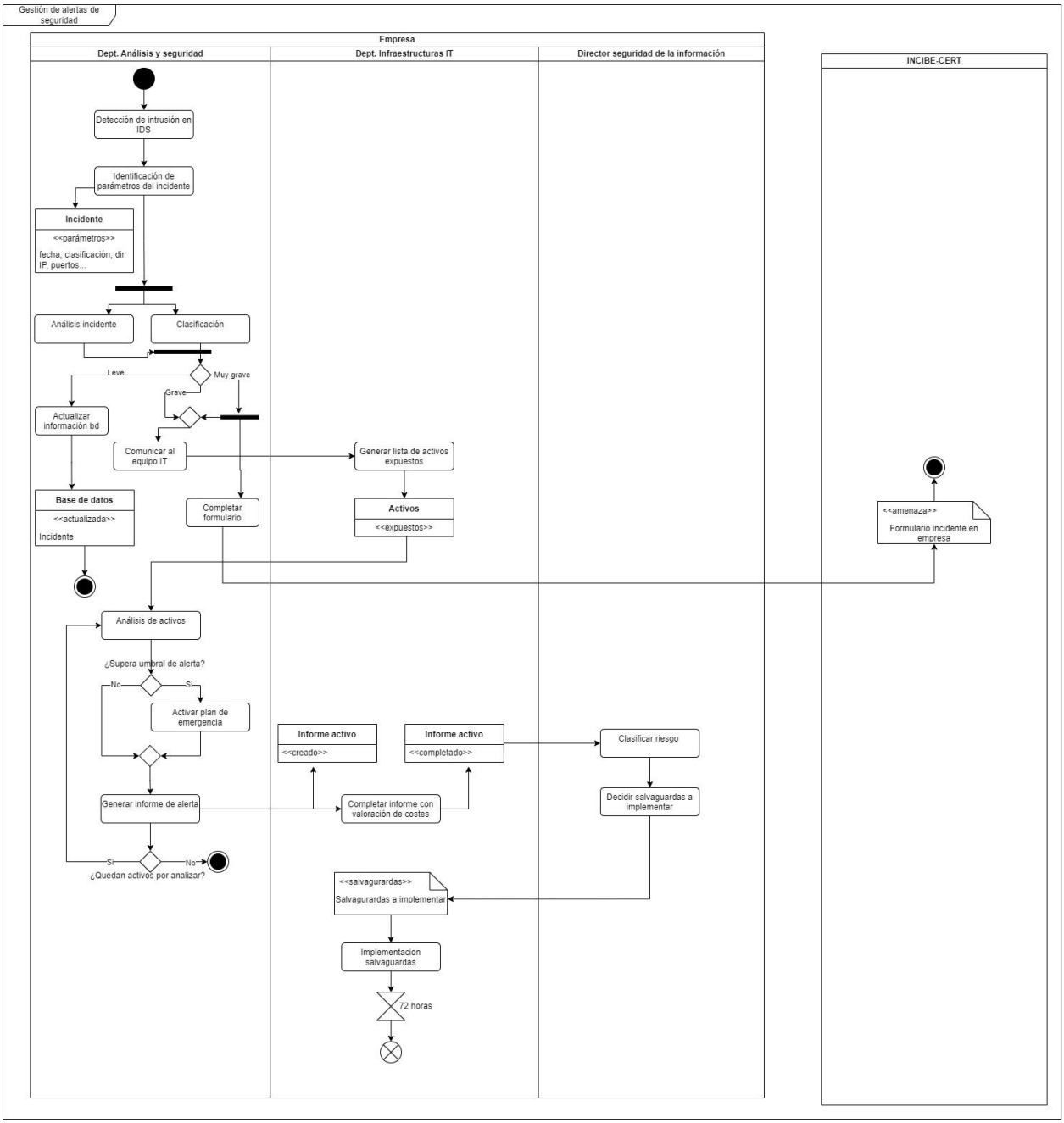


Ilustración 1. Diagrama UML

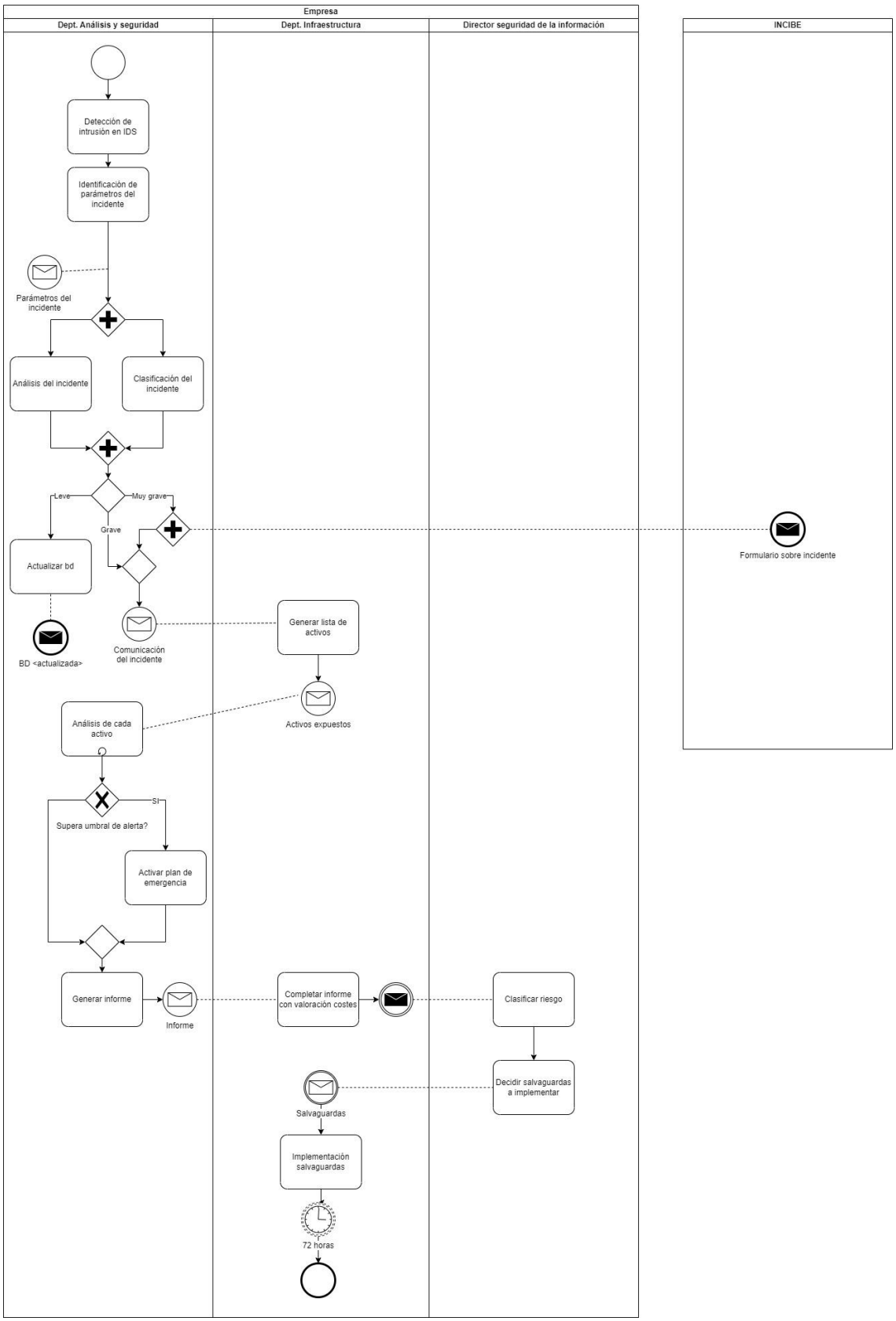


Ilustración 2. Diagrama BPMN

3. Ejercicio 2

Queremos desarrollar un sistema ETL sencillo. Lo primero es conectar la base de datos que hemos creado, “practica.db”.

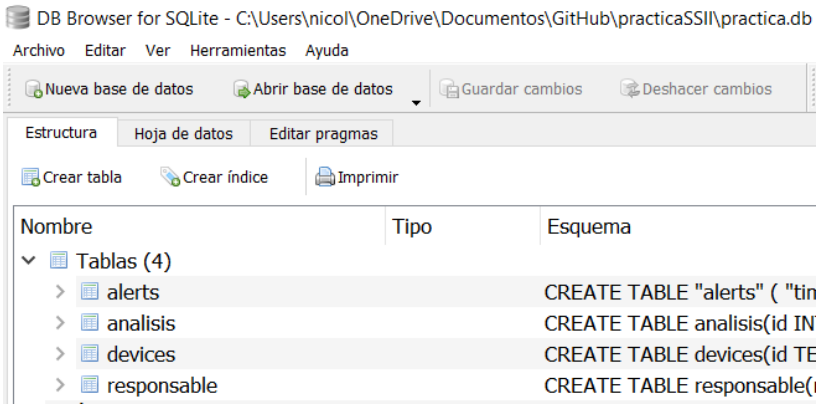
```
#Conectar la bd creada con sqlite
con= sqlite3.connect('practica.db')
```

Seguimos diseñando las tablas en la base de datos. Creamos una tabla para los datos del fichero alerts y otra para devices. A partir de ésta última tenemos la necesidad de crear otras dos: responsable y análisis. Para crear las cuatro tablas utilizamos “CREATE TABLE”:

```
cur.execute("CREATE TABLE IF NOT EXISTS responsable(nombre TEXT PRIMARY_KEY,
telefono TEXT, rol TEXT)")

cur.execute("CREATE TABLE IF NOT EXISTS analisis(id TEXT PRIMARY_KEY,
puertos_abiertos TEXT, numPuertosAbiertos TEXT, servicios TEXT,
servicios_inseguros TEXT, vulnerabilidades_detectadas INTEGER)")

cur.execute("CREATE TABLE IF NOT EXISTS devices(id TEXT, ip TEXT,
localizacion TEXT,responsable_id TEXT, analisis_id TEXT, FOREIGN
KEY(responsable_id) REFERENCES responsable(nombre), FOREIGN KEY(analisis_id)
REFERENCES analisis(id))")

cur.execute("CREATE TABLE IF NOT EXISTS alerts(timestamp TEXT, sid TEXT, msg
TEXT,clasificacion TEXT, prioridad TEXT, protocolo TEXT, origen TEXT, destino
TEXT, puerto TEXT  )")
```

Como las tablas responsable y análisis “salen” de la tabla devices, añadimos análisis_id y responsable_id como FOREIGN KEYS.

Confirmamos en SQLite que las tablas están creadas:



Para almacenar los datos de los ficheros en la base de datos, utilizamos el siguiente comando SQL:

```
"INSERT INTO nombreTabla VALUES ('valor1', 'valor2','valorN')"
```

```
#Datos tabla responsable
cur.execute("INSERT INTO responsable VALUES ('admin', '656445552','Administracion de sistemas')")
cur.execute("INSERT INTO responsable VALUES ('Paco Garcia', '640220120','Direccion')")
cur.execute("INSERT INTO responsable VALUES ('Luis Sanchez', 'None','Desarrollador')")
cur.execute("INSERT INTO responsable VALUES ('admin', '656445552','Administracion de sistemas')")
cur.execute("INSERT INTO responsable VALUES ('admiin', 'None','None')")
cur.execute("INSERT INTO responsable VALUES ('admin', '656445552','Administracion de sistemas')")
cur.execute("INSERT INTO responsable VALUES ('admin','656445552','Administracion de sistemas')")
```

```
#Datos tabla analisis
cur.execute("INSERT INTO analisis VALUES(1, '80/TCP, 443/TCP, 3306/TCP, 40000/UDP', 4, 3, 0, 15)")
cur.execute("INSERT INTO analisis VALUES(2, 'None', 0, 0, 0, 4)")
cur.execute("INSERT INTO analisis VALUES(3, '1194/UDP, 8080/TCP,8080/UDP, 40000/UDP',4, 1, 1, 52)")
cur.execute("INSERT INTO analisis VALUES(4, '443/UDP, 80/TCP',2, 1, 0,3)")
cur.execute("INSERT INTO analisis VALUES(5, '80/TCP, 67/UDP, 68/UDP', 3, 2, 2, 12)")
cur.execute("INSERT INTO analisis VALUES(6, '8080/TCP, 3306/TCP, 3306/UDP', 3,2, 0, 2)")
cur.execute("INSERT INTO analisis VALUES(7, '80/TCP, 443/TCP, 9200/TCP, 9300/TCP, 5601/TCP', 5,3, 2, 21)")
```

```
#Datos tabla devices
cur.execute("INSERT INTO devices VALUES('web', '172.18.0.0', 'None','admin', 1)")
cur.execute("INSERT INTO devices VALUES('paco_pc', '172.17.0.0', 'Barcelona','Paco Garcia', 2)")
cur.execute("INSERT INTO devices VALUES('luis_pc', '172.19.0.0', 'Madrid','Luis Sanchez', 3)")
cur.execute("INSERT INTO devices VALUES('router1', '172.1.0.0', 'None','admin', 4)")
cur.execute("INSERT INTO devices VALUES('dhcp_server', '172.1.0.1', 'Madrid','admiin', 5)")
cur.execute("INSERT INTO devices VALUES('mysql_db', '172.18.0.1', 'None','admin', 6)")
cur.execute("INSERT INTO devices VALUES('ELK', '172.18.0.2', 'None','admin', 7)")
```

Para leer del fichero “alerts.csv” hacemos uso de las funciones que nos proporciona la librería panda. Primero

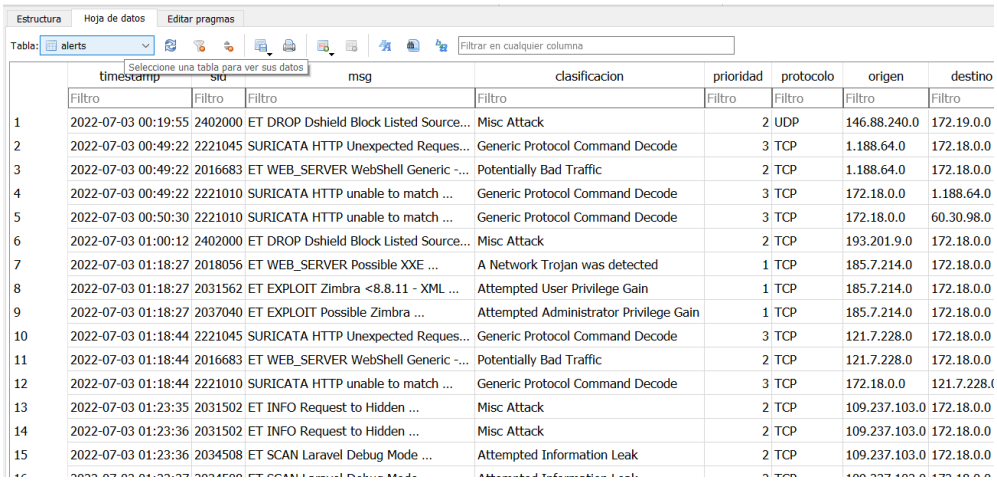
```
df_alertas= pd.read_csv('alerts.csv')
```

que almacena los datos de dicho fichero en un DataFrame. Luego, los almacenamos en nuestra tabla SQL “alerts”:

```
df_alertas.to_sql('alerts',con,if_exists='replace',index=False)
```

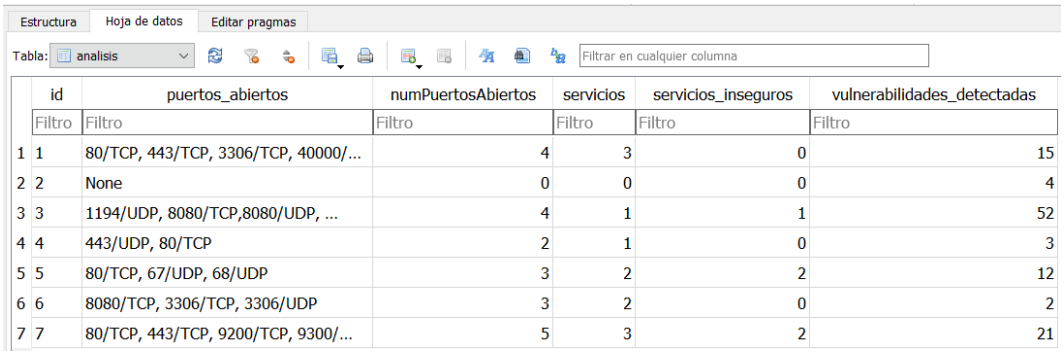
Comprobamos que las tablas ya contienen los datos:

Tabla alerts



	timestamp	sid	msg	clasificacion	prioridad	protocolo	origen	destino
	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	2022-07-03 00:19:55	2402000	ET DROP Dshield Block Listed Source...	Misc Attack	2	UDP	146.88.240.0	172.19.0.0
2	2022-07-03 00:49:22	2221045	SURICATA HTTP Unexpected Reques...	Generic Protocol Command Decode	3	TCP	1.188.64.0	172.18.0.0
3	2022-07-03 00:49:22	2016683	ET WEB_SERVER WebShell Generic -...	Potentially Bad Traffic	2	TCP	1.188.64.0	172.18.0.0
4	2022-07-03 00:49:22	2221010	SURICATA HTTP unable to match ...	Generic Protocol Command Decode	3	TCP	172.18.0.0	1.188.64.0
5	2022-07-03 00:50:30	2221010	SURICATA HTTP unable to match ...	Generic Protocol Command Decode	3	TCP	172.18.0.0	60.30.98.0
6	2022-07-03 01:00:12	2402000	ET DROP Dshield Block Listed Source...	Misc Attack	2	TCP	193.201.9.0	172.18.0.0
7	2022-07-03 01:18:27	2018056	ET WEB_SERVER Possible XXE ...	A Network Trojan was detected	1	TCP	185.7.214.0	172.18.0.0
8	2022-07-03 01:18:27	2031562	ET EXPLOIT Zimbra <8.8.11 - XML ...	Attempted User Privilege Gain	1	TCP	185.7.214.0	172.18.0.0
9	2022-07-03 01:18:27	2037040	ET EXPLOIT Possible Zimbra ...	Attempted Administrator Privilege Gain	1	TCP	185.7.214.0	172.18.0.0
10	2022-07-03 01:18:44	2221045	SURICATA HTTP Unexpected Reques...	Generic Protocol Command Decode	3	TCP	121.7.228.0	172.18.0.0
11	2022-07-03 01:18:44	2016683	ET WEB_SERVER WebShell Generic -...	Potentially Bad Traffic	2	TCP	121.7.228.0	172.18.0.0
12	2022-07-03 01:18:44	2221010	SURICATA HTTP unable to match ...	Generic Protocol Command Decode	3	TCP	172.18.0.0	121.7.228.(
13	2022-07-03 01:23:35	2031502	ET INFO Request to Hidden ...	Misc Attack	2	TCP	109.237.103.0	172.18.0.0
14	2022-07-03 01:23:36	2031502	ET INFO Request to Hidden ...	Misc Attack	2	TCP	109.237.103.0	172.18.0.0
15	2022-07-03 01:23:36	2034508	ET SCAN Laravel Debug Mode ...	Attempted Information Leak	2	TCP	109.237.103.0	172.18.0.0

Tabla análisis



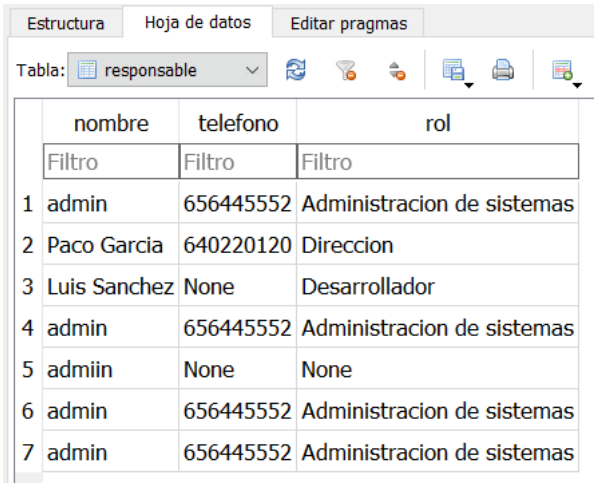
Estructura | Hoja de datos | Editar pragmas

Tabla: analisis | Filtrar en cualquier columna

	id	puertos_abiertos	numPuertosAbiertos	servicios	servicios_inseguros	vulnerabilidades_detectadas
	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	1	80/TCP, 443/TCP, 3306/TCP, 40000/...	4	3	0	15
2	2	None	0	0	0	4
3	3	1194/UDP, 8080/TCP,8080/UDP, ...	4	1	1	52
4	4	443/UDP, 80/TCP	2	1	0	3
5	5	80/TCP, 67/UDP, 68/UDP	3	2	2	12
6	6	8080/TCP, 3306/TCP, 3306/UDP	3	2	0	2
7	7	80/TCP, 443/TCP, 9200/TCP, 9300/...	5	3	2	21

A esta tabla le hemos añadido la columna numPuertosAbiertos para poder realizar el ejercicio. Lo explicamos con más detalle más adelante.

Tabla responsable



Estructura | Hoja de datos | Editar pragmas

Tabla: responsable

	nombre	telefono	rol
	Filtro	Filtro	Filtro
1	admin	656445552	Administracion de sistemas
2	Paco Garcia	640220120	Direccion
3	Luis Sanchez	None	Desarrollador
4	admin	656445552	Administracion de sistemas
5	admiin	None	None
6	admin	656445552	Administracion de sistemas
7	admin	656445552	Administracion de sistemas

Tabla devices

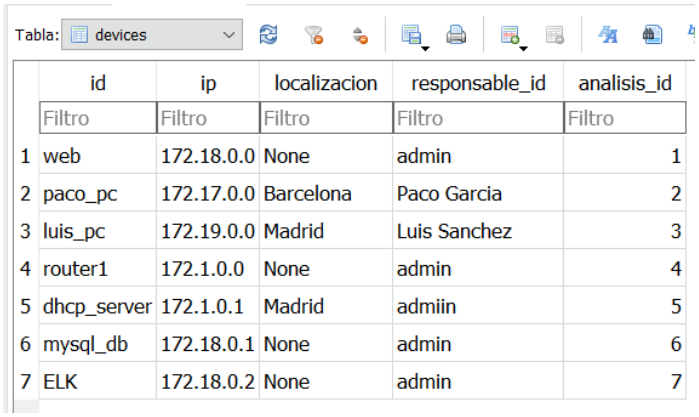


Tabla: devices

	id	ip	localizacion	responsable_id	analisis_id
	Filtro	Filtro	Filtro	Filtro	Filtro
1	web	172.18.0.0	None	admin	1
2	paco_pc	172.17.0.0	Barcelona	Paco Garcia	2
3	luis_pc	172.19.0.0	Madrid	Luis Sanchez	3
4	router1	172.1.0.0	None	admin	4
5	dhcp_server	172.1.0.1	Madrid	admiin	5
6	mysql_db	172.18.0.1	None	admin	6
7	ELK	172.18.0.2	None	admin	7

Una vez tenemos todos los datos de todos los ficheros en la base de datos, procedemos a leer los datos de nuestra BD y los almacenamos en dataframes. Para ello, usamos el comando sql “select * from nombreTabla” y la función de panda “read_sql_query”:

```
#Consultas
print("### CONSULTAS ###")
df_dispositivos = pd.read_sql_query("SELECT * from devices", con)
df_analisis=pd.read_sql_query("SELECT * FROM analisis",con)
```

Ahora, debemos calcular los valores que nos piden.

Comenzamos con el número de dispositivos. Para ello utilizamos “nunique()”

```
numDispositivos = df_dispositivos['id'].nunique()
print("Número de dispositivos: " +str(numDispositivos))
#numDispositivos=7
```

Para el número de alertas:

```
numAlertas= len(df_alertas)
print("Número de alertas: " + str(numAlertas))
# numAlertas = 200225
```

Para las medias hemos usado “mean()” y para las desviaciones estándar “std()”. Para mínimos y máximos “min()” y “max()”.

```
77  mediaPuertos=df_analisis['numPuertosAbiertos'].mean()
78  desvPuertos=df_analisis['numPuertosAbiertos'].std()
79  print("Media de puertos: " + str(mediaPuertos)+"     Desviacion estándar: " + str(desvPuertos))
80  #media y desviacion puertos abiertos
81
82
83  mediaServicios=df_analisis['servicios_inseguros'].mean()
84  desvServicios=df_analisis['servicios_inseguros'].std()
85  mediaVulner=df_analisis['vulnerabilidades_detectadas'].mean()
86  desvVulner=df_analisis['vulnerabilidades_detectadas'].std()
87  minPuertos=df_analisis['numPuertosAbiertos'].min()
88  maxPuertos=df_analisis['numPuertosAbiertos'].max()
89  minVulner=df_analisis['vulnerabilidades_detectadas'].min()
90  maxVulner=df_analisis['vulnerabilidades_detectadas'].max()
```

Para el obtener la media y desviación estándar del total de puertos abiertos, hemos añadido una nueva columna a la tabla análisis. Este campo determina el número de puertos abiertos según el campo que ya viene explícitamente de puertos_abiertos. Así, si tenemos por ejemplo que los puertos abiertos son:

```
'1194/UDP, 8080/TCP,8080/UDP, 40000/UDP'
```

El campo numPuertosAbiertos lo ponemos a 4.

puertos_abiertos	numPuertosAbiertos
Filtro	Filtro
80/TCP, 443/TCP, 3306/TCP, 40000/...	4
None	0
1194/UDP, 8080/TCP,8080/UDP, ...	4
443/UDP, 80/TCP	2
80/TCP, 67/UDP, 68/UDP	3
8080/TCP, 3306/TCP, 3306/UDP	3
80/TCP, 443/TCP, 9200/TCP, 9300/...	5

Esta columna es necesaria también para calcular el valor mínimo y máximo del total de puertos abiertos. De esta manera, obtenemos los siguientes resultados:

```
C:\Users\nicol\AppData\Local\Programs\Python\Python37\python.exe C:\Users\nicol\OneDrive\Documentos\GitHub\practicaSSII\main.py
### CONSULTAS ###
Número de dispositivos: 7
Número de alertas: 200225
Media de puertos: 3.0     Desviacion estándar: 1.5689290811054724
Media servicios inseguros: 0.7142857142857143   Desviacion estándar: 0.9138735334633754
Media vulnerabilidades detectadas: 15.571428571428571   Desviación estandar: 16.85099534021786
Mínimo puertos abiertos: 0     Máximo: 5
Mínimo vulnerabilidades: 2   Máximo: 52

Process finished with exit code 0
```

4. Ejercicio 3

Para agrupar las alertas por prioridad lo primero que hacemos es unir las tablas de *analisis*, *devices* y *alerts*, ya que necesitamos ciertos campos de las tres tablas (devices.localizacion, analisis.vulnerabilidades_detectadas y alerts.prioridad). Teniendo el dataframe de la unión solo nos queda ir haciendo las consultas con las funciones.

Lo que hemos hecho para imprimir los datos de cada prioridad es hacer un bucle *for* donde la variable *i* vaya marcando que prioridad es, así nos evitamos tener que crear dataframes por cada tipo de prioridad que exista. Para el apartado de campos missing y none hemos tomado el campo devices.localizacion ya que es el único que tenía valores none.

Agrupación por prioridad de alerta:

```
#Join tablas para tener informacion relacionada en una única
df_tabla3=pd.read_sql_query("SELECT * FROM alerts JOIN devices ON (alerts.origen = devices.ip) JOIN analisis ON devices.analisis_id = analisis.id", con)

#Agrupacion por prioridad
print("### ANALISIS BASADO EN PRIORIDAD DE ALERTA ###")
for i in range(1,4):
    prioridad=df_tabla3.loc[df_tabla3['prioridad']==i]
    print("# ALERTAS PRIORIDAD "+ str(i))
    print("Numero de observaciones:", str(len(prioridad)))
    print("Numero de valores ausentes:", str(len(prioridad.loc[prioridad['localizacion']=='None'])))
    print("Mediana:", prioridad['vulnerabilidades_detectadas'].median())
    print("Media:",prioridad['vulnerabilidades_detectadas'].mean())
    print("Varianza:", prioridad['vulnerabilidades_detectadas'].var())
    print("Minimo",prioridad['vulnerabilidades_detectadas'].min())
    print("Maximo:",prioridad['vulnerabilidades_detectadas'].max())
```

Para la agrupación por fechas usamos el dataframe del apartado anterior y mediante la función de pandas to_datetime y .dt.month podremos ir sacando los meses de la tabla del campo timestamp. El procedimiento es el mismo, mediante un for que empiece en 7 y que solo haga dos iteraciones (7 para julio y 8 para agosto) y que por cada iteracion se vayan imprimiendo los valores necesarios.

Agrupación por fechas:

```
#Agrupacion por fechas (solo JULIO 7 y AGOSTO 8)
print("### ANALISIS BASADO EN RANGOS MENSUALES ###")
for i in range(7,9):
    if (i==7):
        print("# ALERTAS JULIO: ")
    else:
        print("# ALERTAS AGOSTO")
    fecha=df_tabla3.loc[(pd.to_datetime(df_tabla3['timestamp']).dt.month==i)]
    print("Numero de observaciones", str(len(fecha)))
    print("Numero de valores ausentes", str(len(fecha.loc[fecha['localizacion']=='None'])))
    print("Mediana:",fecha['vulnerabilidades_detectadas'].median())
    print("Media:", fecha['vulnerabilidades_detectadas'].mean())
    print("Varianza:", fecha['vulnerabilidades_detectadas'].var())
    print("Minimo", fecha['vulnerabilidades_detectadas'].min())
    print("Maximo:", fecha['vulnerabilidades_detectadas'].max())
```

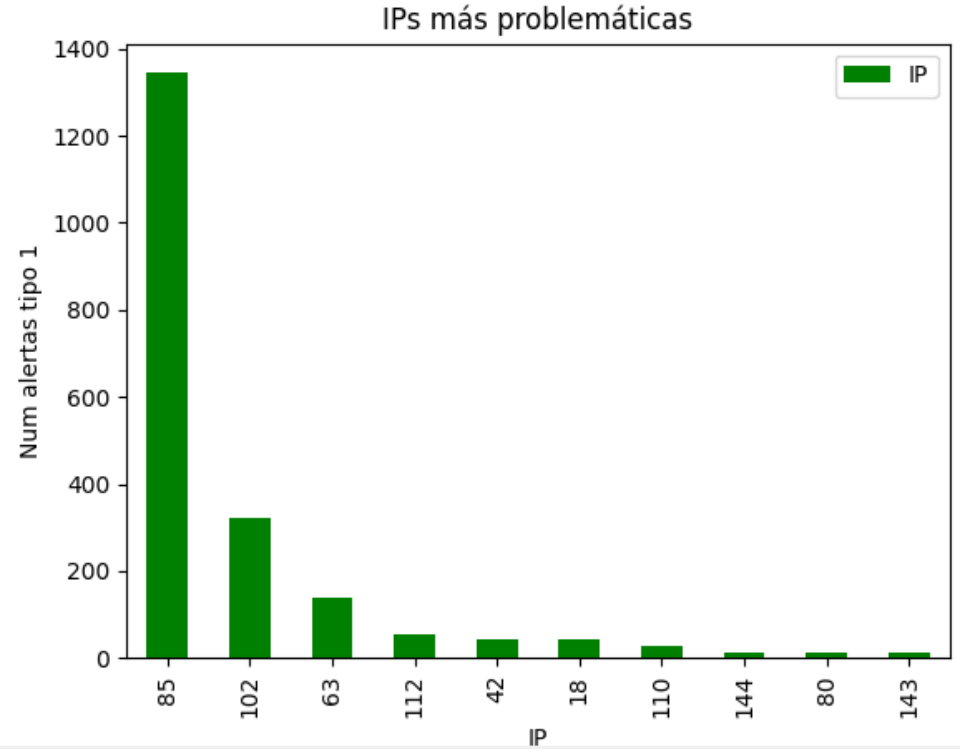
5. Ejercicio 4

En este apartado vamos a generar diversas gráficas.

a. Mostrar las 10 IP de origen más problemáticas.

Para sacar las ip que son más problemáticas lo primero que tenemos que hacer es sacar a una variable aquellas ip que son de prioridad 1. Teniendo ya en la variable los campos que necesitamos podemos agruparlos por origen y así acumular todas las ip iguales en una misma barra del gráfico. Por último, nos queda ordenar los valores y usar la función head para que solo salgan las 10 primeras ip. Usamos la librería matplotlib.pyplot para todo este apartado.

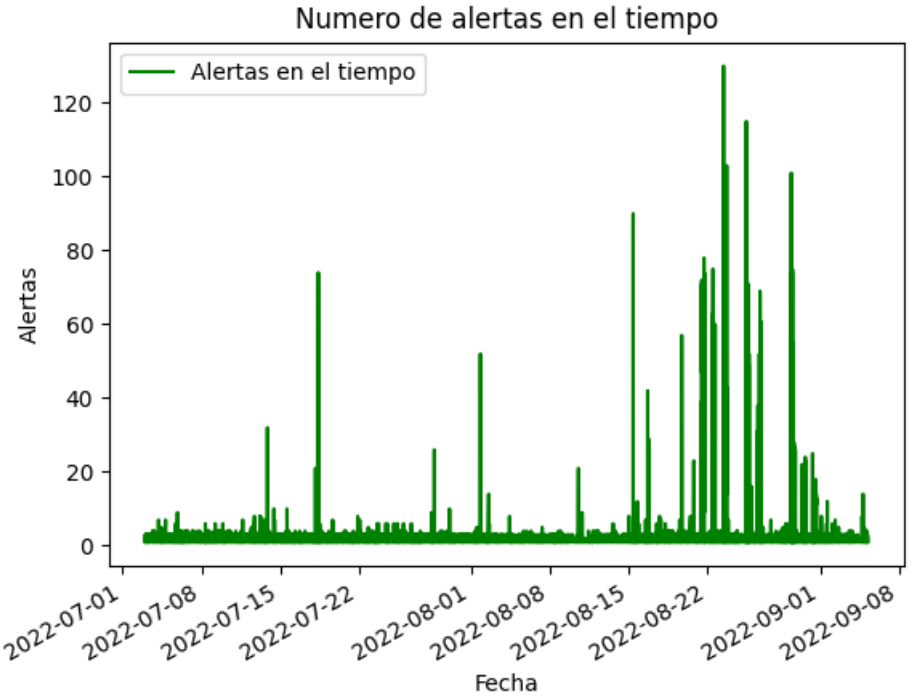
```
#Apartado A
apartadoIp = df_alertas[df_alertas['prioridad']==1]
apartadoIp = apartadoIp.groupby('origen')['sid'].count().reset_index(name='IP')
apartadoIp.sort_values(by=['IP'],ascending=False, inplace=True)
apartadoIp.head(10).plot(kind='bar',color='green')
plt.title('IPs más problemáticas')
plt.xlabel('IP')
plt.ylabel('Num alertas tipo 1')
plt.show()
```



b. Número de alertas en el tiempo

Para representar las alertas que han ido ocurriendo a lo largo del tiempo lo que hacemos es agrupar en una variable el campo timestamp de alertas con la función count para saber el número de apariciones que tiene cada fecha en las tablas. Luego establecemos este campo como índice y creamos el gráfico. Como tienen que ir en orden cronológico no tiene sentido esta vez ordenarlas por número de alertas.

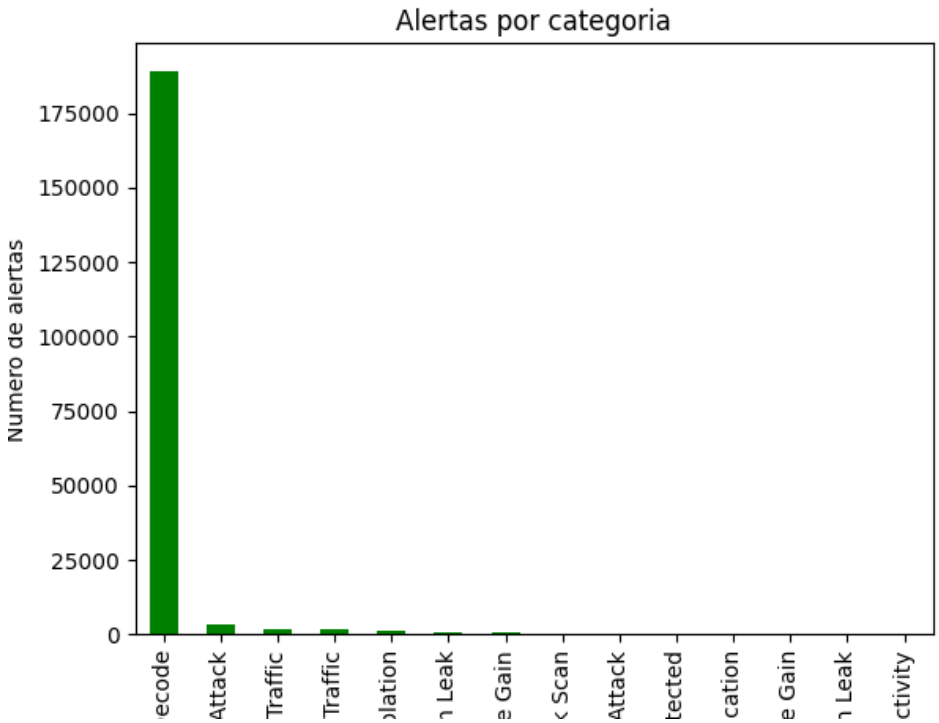
```
#Apartado B
timeAlerts = df_alertas.groupby('timestamp')['timestamp'].count().reset_index(name='Alertas en el tiempo')
timeAlerts['timestamp'] = pd.to_datetime(timeAlerts['timestamp'])
timeAlerts = timeAlerts.set_index('timestamp')
timeAlerts.plot(kind='line',color='green')
plt.xlabel('Fecha')
plt.ylabel('Alertas')
plt.title('Numero de alertas en el tiempo')
plt.show()
```



c. Numero de alertas por categoría

Si queremos representar un gráfico de alertas por categoría, lo que tenemos que hacer es agrupar el campo clasificación de la tabla de alertas y ordenar los valores. El procedimiento es igual a los apartados anteriores.

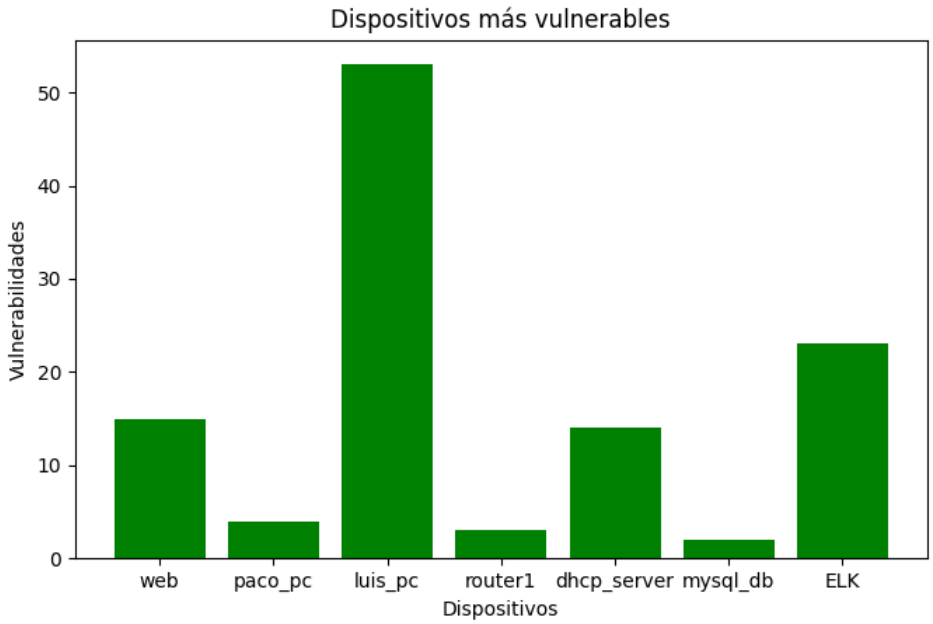
```
#Apartado C
alertasCateg= df_alertas.groupby('clasificacion')['sid'].count()
alertasCateg=alertasCateg.sort_values(ascending=False)
alertasCateg.plot(kind='bar',color='green')
plt.title('Alertas por categoria')
plt.xlabel('Categoria')
plt.ylabel('Numero de alertas')
plt.show()
```



d. Dispositivos más vulnerables

En este apartado lo primero que tenemos que hacer es crear una nueva columna que sea la suma de servicios vulnerables más las vulnerabilidades detectadas, esto lo guardamos en “total_vulnerabilidades”, que será el campo del eje y. Para crear el gráfico esta vez usamos la función plt.bar que nos permite asignar a cada eje un campo que nosotros queremos. Al eje de las x le ponemos el campo dispositivos.id y como las tablas de dispositivos y análisis vienen ordenadas de la misma manera, si no ordenamos los datos de mayor a menor de ninguna tabla, coincidirán.

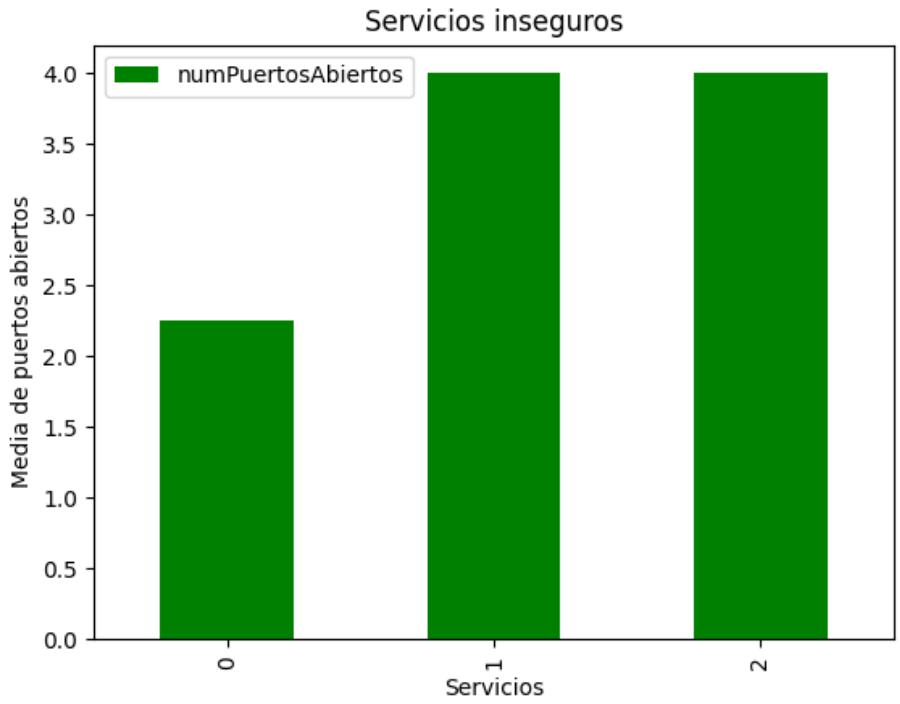
```
#Apartado D
df_analisis['total_vulnerabilidades']=df_analisis['servicios_inseguros']+df_analisis['vulnerabilidades_detectadas']
plt.bar(df_dispositivos['id'],df_analisis['total_vulnerabilidades'],color='green')
plt.title('Dispositivos más vulnerables')
plt.xlabel('Dispositivos')
plt.ylabel('Vulnerabilidades')
plt.show()
```



e. Media de puertos abiertos frente a servicios inseguros y frente al total de servicios detectados.

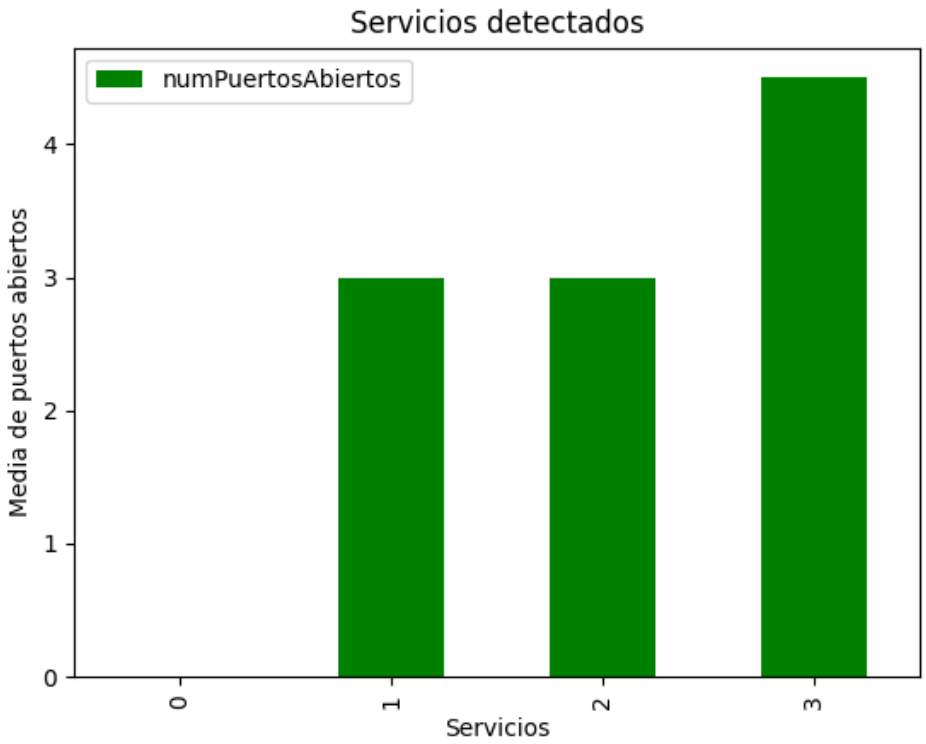
Este apartado lo hemos dividido en dos, un gráfico de los puertos abiertos frente a los servicios inseguros y otro frente al total de servicios detectados. El procedimiento es el mismo, pero cambiando los datos que leemos del dataframe de análisis. Para los servicios inseguros usaremos el campo servicios_inseguros de analisis y numPuertosAbiertos, haremos uso de la función mean para poder sacar la media y crearemos el gráfico como en los anteriores apartados.

```
#Apartado E.1
df_inseguros = df_analisis[['numPuertosAbiertos', 'servicios_inseguros']]
df_inseguros = df_inseguros.groupby('servicios_inseguros').mean()
df_inseguros.plot(kind='bar', color='green')
plt.title('Servicios inseguros')
plt.xlabel('Servicios')
plt.ylabel('Media de puertos abiertos')
plt.show()
```



Para el gráfico del total de servicios detectados solo necesitamos cambiar el campo servicios_inseguros por servicios:

```
#Apartado E.2
df_servicios = df_analisis[['numPuertosAbiertos', 'servicios']]
df_servicios = df_servicios.groupby('servicios').mean()
df_servicios.plot(kind='bar', color='green')
plt.title('Servicios detectados')
plt.xlabel('Servicios')
plt.ylabel('Media de puertos abiertos')
plt.show()
```


Servicios detectados
numPuertosAbiertos
Media de puertos abiertos
0
1
2
3
4
0
1
2
3
Servicios