

RECOPILACIÓN, ESTRUCTURACIÓN Y ANÁLISIS DE DATOS

PRÁCTICA 1



Ingeniería de la ciberseguridad.

Lucía Castro Rodríguez,

Oleg Fomenkov,

Mónica Ramos Martínez,

Álvaro Sierra Parra.

ÍNDICE

INTRODUCCIÓN	3
EJERCICIOS POR RESOLVER	6
1. EJERCICIO: UML y BPMN.....	6
2. EJERCICIO: Sistema ETL.....	8
3. EJERCICIO	11
4. EJERCICIO: Funciones del MIS	13
CONCLUSIONES	19

INTRODUCCIÓN

En esta práctica, vamos a analizar el funcionamiento y los datos de una empresa. Como se indica en el enunciado, empezamos con una modelación de cómo es el funcionamiento de la empresa, tanto en UML como en BPMN, y luego se utilizará *Python* y *SQLite* para crear un pequeño MIS (Management Information System) que analizará los datos en la BD mediante el uso de la librería *Pandas*.

El primer paso antes del desarrollo del código es la creación de la base de datos y la inserción de los datos presentes en los ficheros .json. Para ello, utilizamos la librería *sqlite3*, que nos permite crear una base de datos y manipularla utilizando el código en *Python*.

```
cursor_user.execute("""
CREATE TABLE IF NOT EXISTS user_data_online(
username TEXT PRIMARY KEY,
tel_num INTEGER,
hash_password TEXT NOT NULL,
pass_complexity INTEGER NOT NULL,
province TEXT,
permisos INTEGER,
emails_total INTEGER,
emails_phishing INTEGER,
emails_clicados INTEGER
);
""")

# La de las fechas y las IPs de los usuarios
cursor_user.execute("DROP TABLE IF EXISTS users_ips_fechas;")
cursor_user.execute("""
CREATE TABLE IF NOT EXISTS users_ips_fechas(
id INTEGER PRIMARY KEY AUTOINCREMENT,
username_access TEXT NOT NULL,
ip_address TEXT,
fecha DATETIME,
foreign key (username_access) REFERENCES user_data_online(username)
);
""")
```

Ilustración 1: Creación de las bases de datos *user_data_online* y *users_ips_fechas*.

<div> <div>Create Table</div> <div>Create Index</div> <div>Modify Table</div> <div>Delete Table</div> <div>Print</div> </div>		
Name	Type	Schema
Tables (4)		
legal_data_online		CREATE TABLE legal_data_online(web TEXT PRIMARY KEY,cookies INTEGER NOT NULL,aviso INTEGER NOT NULL)
web	TEXT	"web" TEXT
cookies	INTEGER	"cookies" INTEGER NOT NULL
aviso	INTEGER	"aviso" INTEGER NOT NULL
proteccion	INTEGER	"proteccion" INTEGER NOT NULL
creacion	INTEGER	"creacion" INTEGER NOT NULL
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
user_data_online		CREATE TABLE user_data_online(username TEXT PRIMARY KEY,teI_num INTEGER,hash_password TEXT NOT NULL)
username	TEXT	"username" TEXT
teI_num	INTEGER	"teI_num" INTEGER
hash_password	TEXT	"hash_password" TEXT NOT NULL
pass_complexity	INTEGER	"pass_complexity" INTEGER NOT NULL
province	TEXT	"province" TEXT
permisos	INTEGER	"permisos" INTEGER
emalis_total	INTEGER	"emalis_total" INTEGER
emalis_phishing	INTEGER	"emalis_phishing" INTEGER
emalis_clicados	INTEGER	"emalis_clicados" INTEGER
users_ips_fechas		CREATE TABLE users_ips_fechas(id INTEGER PRIMARY KEY AUTOINCREMENT,username_access TEXT NOT NULL)
id	INTEGER	"id" INTEGER
username_access	TEXT	"username_access" TEXT NOT NULL
ip_address	TEXT	"ip_address" TEXT
fecha	DATETIME	"fecha" DATETIME
Indices (0)		
Views (0)		
Triggers (0)		

Ilustración 2: Visualización de las tablas en SQLite.

Para la introducir los datos a la base de datos debemos iterar por los ficheros .json y crear sentencias 'INSERT INTO' de SQL. Durante el desarrollo nos enfrentamos a una duda, la clasificación de las contraseñas por nivel de complejidad. Para ello, consultando en Internet, hemos encontrado un fichero llamado 'RockYou.txt' (indicado en el enunciado de la práctica). En este fichero había las 512 contraseñas más comunes, de las cuales algunas han sido utilizadas por los usuarios, por lo que se guardó su hash en la base de datos.

Para comprobar si una contraseña es conocida o no, creamos una función que itera por el fichero de contraseñas, los hashea y los guarda en un array. Posteriormente, para la comprobación, se itera por este array y si la contraseña analizada forma parte de este array, se marca como una contraseña no segura.

```

hashes = []
palabras = open('palabras_comunes.txt', 'r')
for palabra in palabras:
    # Para hashearlas usamos md5
    hash_pass = hash.md5(palabra[:-1].encode(encoding="utf-8")).hexdigest()
    hashes.append(hash_pass)

```

Ilustración 3: Creación del array de contraseñas conocidas hasheadas.

```

# Función para verificar si una contraseña es segura o no
# AlexJ *
def if_secure(password, hashes):
    for h in hashes:
        if password == h:
            return 0
    return 1

```

Ilustración 4: Función de comprobación de la contraseña.

Posteriormente, se hace *insert* de todos los datos sacados de los ficheros .json y se hace commit a la base de datos. Este código se puede encontrar en el fichero *creacionDeTablas.py* y la base de datos en el fichero *users_data_online.db*. Las contraseñas conocidas se encuentran en el fichero *palabras_comunes.txt*.

EJERCICIOS POR RESOLVER

1. EJERCICIO: UML y BPMN

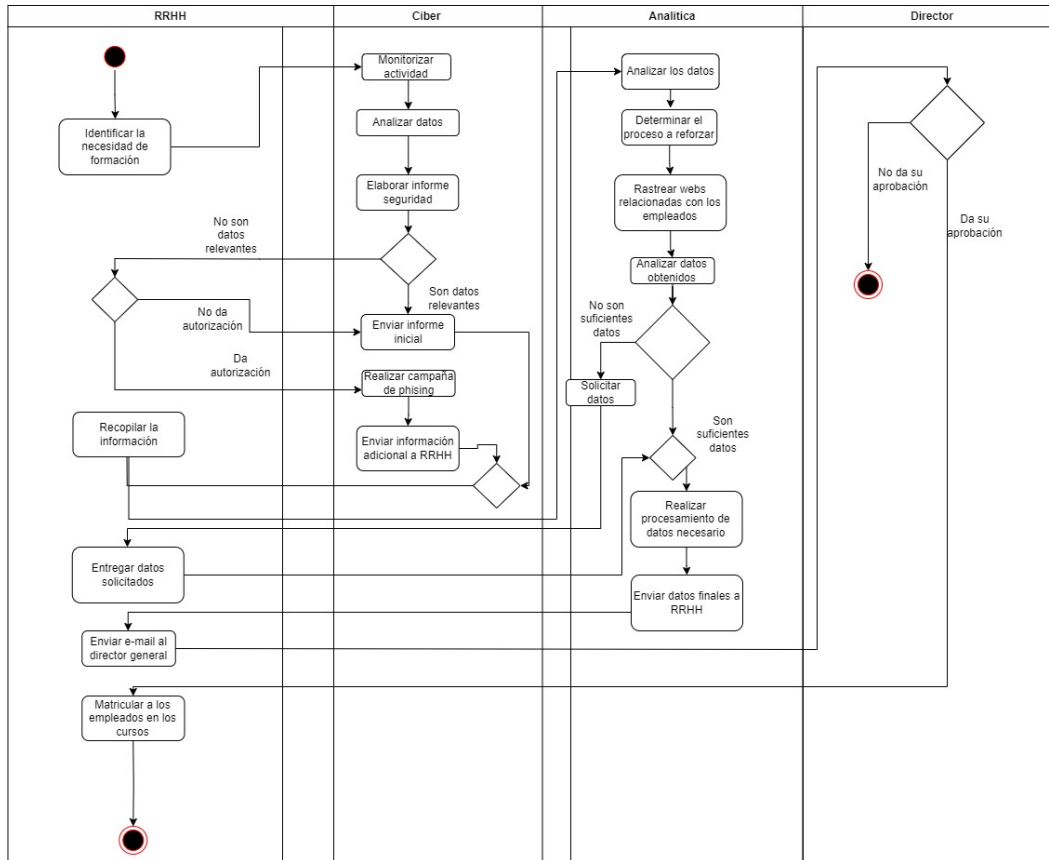


Ilustración 5: UML

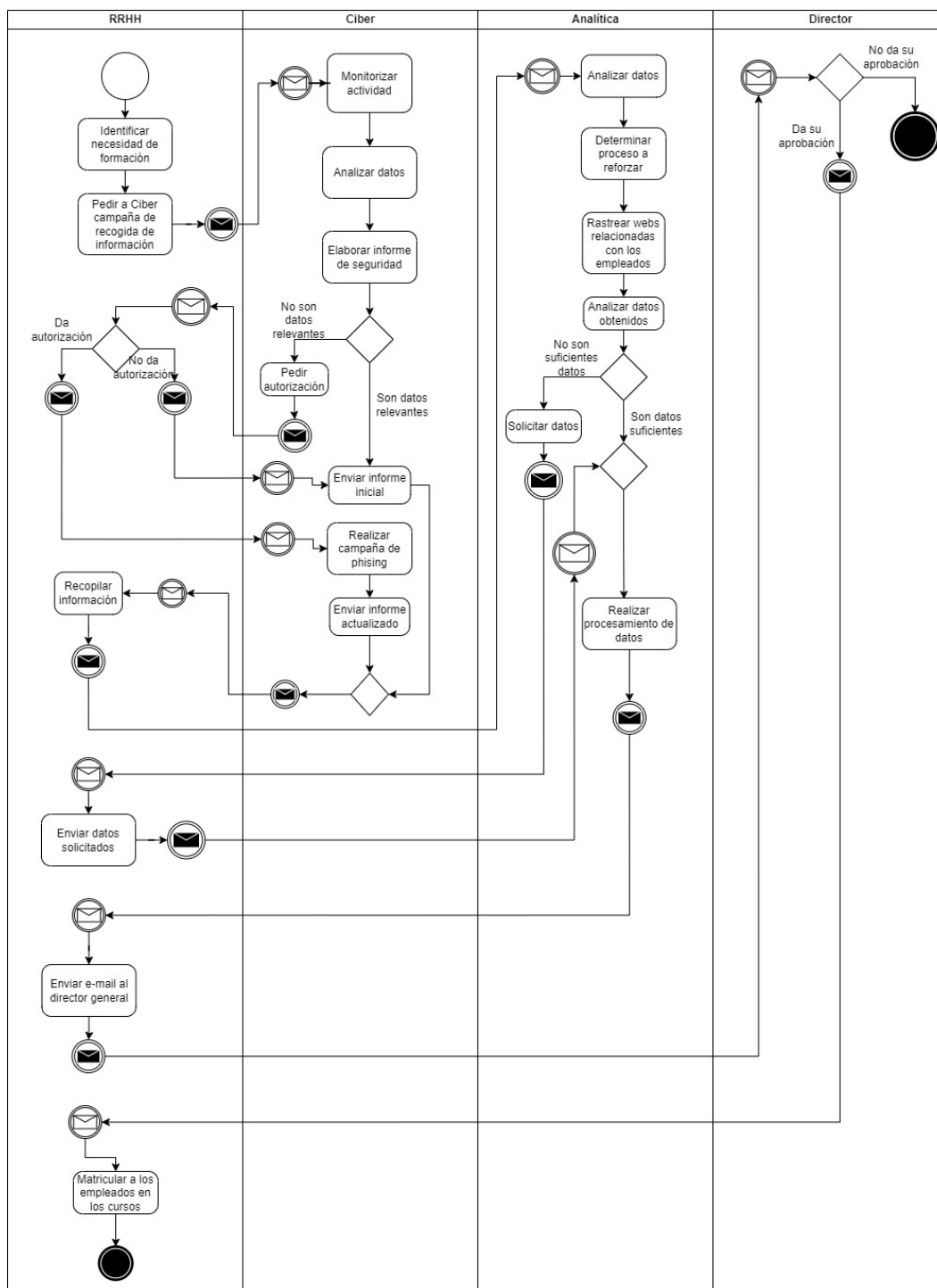


Ilustración 6: BPMN

2. EJERCICIO: Sistema ETL

En este ejercicio desarrollaremos un sencillo sistema ETL, donde almacenamos los datos en diferentes DataFrames (df_users y df_fechas) para poder manipularlos. Los datos han sido extraídos de los ficheros .json que posteriormente han sido almacenados en la BD en distintas tablas.

```
# Conectamos con la base de datos
conn = sqlite3.connect('users_data_online.db')

# CONSULTAS SIN FILTRADO

# Hacemos las consultas para obtener la información necesaria de las tablas de usuarios e IPs-fechas
query_users = "SELECT * FROM user_data_online;"
query_ips = "SELECT * FROM users_ips_fechas WHERE username_access IN (SELECT username FROM user_data_online);"

# Creamos los DataFrames de pandas con los resultados de las consultas
df_users = pd.read_sql_query(query_users, conn)
df_fechas = pd.read_sql_query(query_ips, conn)
```

Ilustración 7

En primer lugar, para que el desarrollo del sistema MIS sea correcto, hemos calculado, como indica el enunciado de la práctica, lo siguiente: el número de muestras, la media y la desviación estándar del total de fechas en las que se ha cambiado la contraseña, la media y la desviación estándar del total de IPs que se han detectado, la media y la desviación estándar del número de email recibidos de phishing en los que ha interactuado cualquier usuario, el valor mínimo y el valor máximo del total de emails recibidos, y el valor mínimo y el valor máximo del número de emails de phishing en los que ha interactuado un administrador.

Para realizar los cálculos, hemos utilizado diferentes funciones que pertenecen a la librería Panda que está especializada en el manejo y análisis de estructuras de datos.

```
# pregunta 2
print("Media de fechas por usuario:")
# agrupamos las fechas por usuarios y se calcula la cantidad media por usuario
print(df_fechas.groupby('username_access').count().mean()['fecha'])
print("Desviación estándar de fechas por usuario:")
print(df_fechas.groupby('username_access').count().std()['fecha'])
print()
```

Ilustración 8

Por otro lado, hemos realizado el ejercicio de dos formas distintas, la primera de ellas ha sido utilizando todos los datos recogidos en los DataFrames, y la

otra, realizando un filtrado donde descartamos los usuarios que alguno de sus campos eran None.

Se destaca que antes de realizar el filtrado, nos encontramos con un total de 30 usuarios con 292 fechas-ips, y tras el filtrado, únicamente, hay 19 usuarios y 170 muestras de fechas-ips, Por lo que observamos, que el filtrado recude casi a la mitad del total de las muestras.

DATOS SIN FILTRAR NONE

#####

Numero de muestras de usuarios después del filtrado:

30

Numero de muestras de fechas-ips después del filtrado:

292

Media de fechas por usuario:

9.73333333333333

Desviación estándar de fechas por usuario:

6.141623582562273

Media de IPs por usuario:

9.73333333333333

Desviación estándar de IPs por usuario:

6.141623582562273

La media de los emails recibidos de phishing por usuario:

109.33333333333333

El valor mínimo de los emails recibidos en total:

20

El valor máximo de los emails recibidos en total:

20

El valor máximo de los emails recibidos en total:

493

DATOS FILTRANDO NONE
#####
Numero de muestras de usuarios después del filtrado: 19
Numero de muestras de fechas-ips después del filtrado: 170
Media de fechas por usuario: 8.947368421052632
Desviación estándar de fechas por usuario: 5.929902418257696
Media de IPs por usuario: 8.947368421052632
Desviación estándar de IPs por usuario: 5.929902418257696
La media de los emails recibidos de phishing por usuario: 112.15789473684211
El valor mínimo de los emails recibidos en total: 35
El valor máximo de los emails recibidos en total: 448
Valor mínimo de emails de phishing clicados por usuario con permisos de administrador: 4
Valor máximo de emails de phishing clicados por usuario con permisos de administrador: 301

3. EJERCICIO

En este ejercicio haremos dos agrupaciones, por tipo de permiso (usuario y administrador), y por contraseña (débil o no). Para ello, hacemos diferentes DataFrames como en el ejercicio anterior:

Primero, lo hacemos para los tipos de permiso y creamos df_p0 y df_p1.

```
# Hacemos las consultas para obtener los usuarios con permiso 0 o 1 respectivamente
query_permisos_0 = "SELECT * FROM user_data_online WHERE permisos=0"
query_permisos_1 = "SELECT * FROM user_data_online WHERE permisos=1"

# Y se crean los correspondientes DataFrames
df_p0 = pd.read_sql_query(query_permisos_0, conn)
df_p1 = pd.read_sql_query(query_permisos_1, conn)
```

Ilustración 9

Después, para las contraseñas con df_pwd_sec y df_pwd_nosec.

```
# Lo mismo para las contraseñas seguras e inseguras
query_pass_sec = "SELECT * FROM user_data_online WHERE pass_complexity=true"
query_pass_noSec = "SELECT * FROM user_data_online WHERE pass_complexity=false"

df_pwd_sec = pd.read_sql_query(query_pass_sec, conn)
df_pwd_nosec = pd.read_sql_query(query_pass_noSec, conn)
```

Ilustración 10

Tras hacer las dos agrupaciones, calcularemos lo requerido para la variable dentro del email de phishing, utilizando las funciones de la librería Pandas al igual que en el ejercicio anterior, y obtenemos estos resultados:

PERMISOS DE USUARIO
Numero de observaciones de emails phishing:
16

Numero missings:
1

Mediana:
41.0

Media:
79.8125

Varianza:
9881.229166666666

Valor maximo:
382

Valor minimo:
0

PERMISOS DE ADMIN
Numero de observaciones de emails phishing:
14

Numero missings:
0

Mediana:
138.0

Media:
143.07142857142858

Varianza:
12490.840659340658

Valor maximo:
372

Valor minimo:
1

CONTRASEÑA SEGURA
Numero de observaciones de emails phishing:
16

Numero missings:
0

Mediana:
99.0

Media:
119.5

Varianza:
12071.066666666668

Valor maximo:
382

Valor minimo:
1

CONTRASEÑA DEBIL
Numero de observaciones de emails phishing:
14

Numero missings:
1

Mediana:
45.0

Media:
97.71428571428571

Varianza:
11989.912087912086

Valor maximo:
372

Valor minimo:
0

4. EJERCICIO: Funciones del MIS

En este ejercicio, se utilizará la librería Matplotlib.pyplot para hacer los diferentes gráficos, y la librería Panda.

A continuación, se programan diferentes funciones del MIS para:

- Mostrar la media de tiempo entre cambios de contraseña por usuario de usuarios normales frente a las de usuarios administradores

Primero, como en los anteriores ejercicios, se hacen las consultas para obtener los datos y se almacenan en los DataFrames.

```
# Se hacen las consultas para obtener los datos de usuarios normales y admins
query_ips_normal = "SELECT * FROM users_ips_fechas WHERE username_access IN (SELECT username FROM user_data_online WHERE pe
query_ips_admin = "SELECT * FROM users_ips_fechas WHERE username_access IN (SELECT username FROM user_data_online WHERE per
# Y se crean los DataFrames correspondientes
df_ips_normal = pd.read_sql_query(query_ips_normal, conn)
df_ips_admin = pd.read_sql_query(query_ips_admin, conn)
```

Ilustración 11

Una vez creados los DataFrames, se convierten la columna “fecha” en datetime para poder ordenarlos según la fecha, y el usuario, y calcular la diferencia en días de las fechas consecutivas.

```
# Convertir la columna "fecha" al tipo datetime
df_ips_normal['fecha'] = pd.to_datetime(df_ips_normal['fecha'], format="%d/%m/%Y")
df_ips_admin['fecha'] = pd.to_datetime(df_ips_admin['fecha'], format="%d/%m/%Y")

# Ordenar los DataFrames según el nombre de usuario y la fecha
df_ips_normal = df_ips_normal.sort_values(by=['username_access', 'fecha'])
df_ips_admin = df_ips_admin.sort_values(by=['username_access', 'fecha'])

# Calcular la diferencia en días de las fechas consecutivas, por usuario
df_ips_normal['diferencia'] = pd.to_numeric(df_ips_normal.groupby('username_access')['fecha'].diff().dt.days)
df_ips_admin['diferencia'] = pd.to_numeric(df_ips_admin.groupby('username_access')['fecha'].diff().dt.days)
```

Ilustración 12

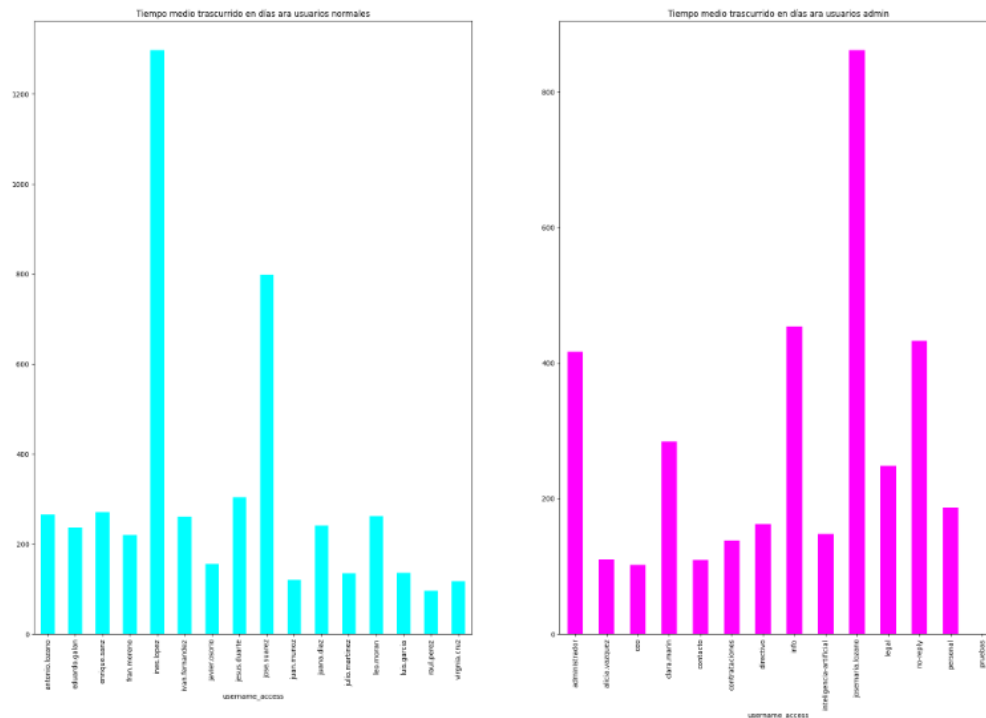
Posteriormente, se muestran los datos y se crean los gráficos de barras para comparar resultados:

La media de tiempo medio entre cambios de contraseña por usuario normal (en días):

username_access	
antonio.lozano	265.125000
eduardo.galan	236.500000
enrique.sanz	270.500000
fran.moreno	220.375000
ines.lopez	1298.000000
ivan.fernandez	261.333333
javier.osorio	156.200000
jesus.duarte	303.666667
jose.suarez	799.000000
juan.munoz	120.611111
juana.diaz	241.444444
julio.martinez	134.692308
leo.moran	261.875000
luis.garcia	135.928571
raul.perez	96.631579
virgnia.cruz	117.470588

La media de tiempo medio entre cambios de contraseña por usuario administrador (en días):

username_access	
administrador	416.500000
alicia.vazquez	110.000000
ceo	102.210526
clara.marin	284.200000
contacto	109.789474
contrataciones	138.142857
directivo	162.461538
info	453.666667
inteligencia-artificial	147.333333
josemaria.lozano	862.000000
legal	248.333333
no-reply	432.666667
personal	186.666667
pruebas	NaN



- Mostrar los 10 usuarios más críticos.

Creamos el DataFrame con las contraseñas inseguras y calculamos la probabilidad de éxito para el ataque de phishing. Finalmente, se eliminan las columnas que no se mostraran en la gráfica y limitamos a los primeros 10 usuarios.

```
# Hacer consultas de usuarios con contraseñas inseguras y crear DataFrame
query_inseguros = 'SELECT username, emails_clicados, emails_phishing FROM user_data_online WHERE pass_complexity IS'
df_inseguros = pd.read_sql_query(query_inseguros, conn)

# Calcular probabilidad de éxito para ataques de phishing
df_inseguros['probabilidad_phishing'] = (df_inseguros['emails_clicados']/df_inseguros['emails_phishing'])*100
df_inseguros.sort_values(by=['probabilidad_phishing'])

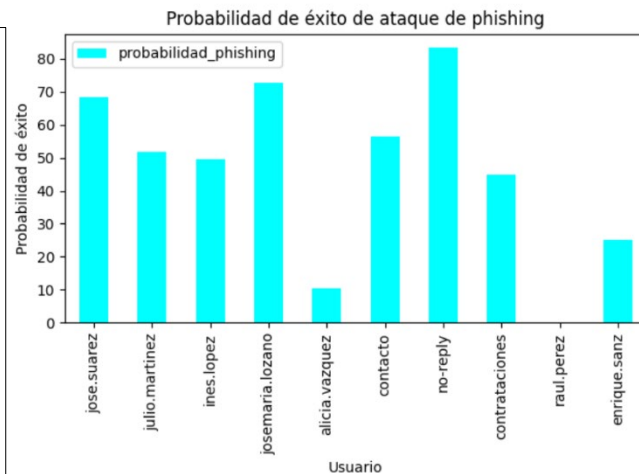
# Quitar columnas que no se mostrarán en la gráfica
df_inseguros.drop(labels=['emails_clicados', 'emails_phishing'], axis=1, inplace=True)
answer += (df_inseguros.to_string())+'\n'

# Limitar DataFrame a los primeros 10 usuarios
df_inseguros = df_inseguros.head(10)
```

Ilustración 13

Estos son los resultados:

username	probabilidad_phishing
0 jose.suarez	68.253968
1 julio.martinez	51.769912
2 ines.lopez	49.462366
3 josemaria.lozano	72.727273
4 alicia.vazquez	10.303030
5 contacto	56.521739
6 no-reply	83.333333
7 contrataciones	44.878049
8 raul.perez	NaN
9 enrique.sanz	25.000000



- Mostrar las 5 páginas web que contienen más políticas (cookies, protección de datos o aviso legal) desactualizadas.

Tras realizar las consultas, en este caso en la tabla de legal, y crear el DataFrame, se calcula la cantidad de políticas desactualizadas, se ordena y se escogen los 5 primeros para mostrarlo en el gráfico.

```
# Realizan consultas para obtener datos de la tabla de legal
query_webs = 'SELECT web, cookies, aviso, proteccion FROM legal_data online;'
df_webs = pd.read_sql_query(query_webs, conn)

# Calcular la cantidad de políticas desactualizadas
df_webs['desactualizadas'] = df_webs[['cookies', 'aviso', 'proteccion']].apply(lambda row: sum(row == 0), axis=1)

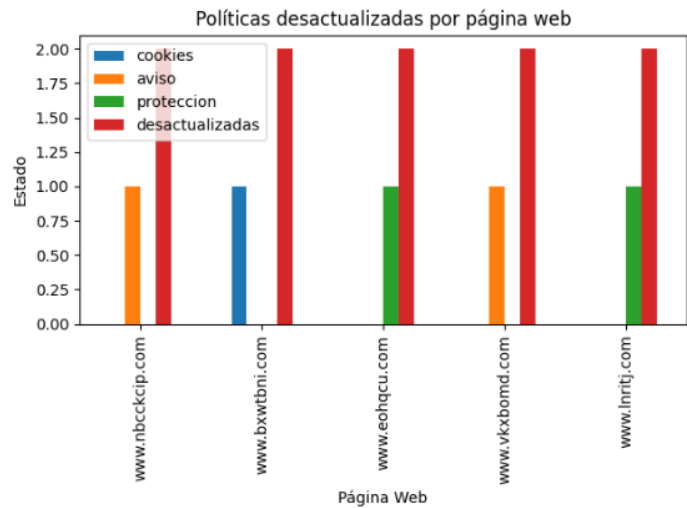
# Para que en la gráfica aparezca con el nombre del sitio web
df_webs = df_webs.set_index('web')
answer+=(df_webs.to_string())+'\n'

# Ordenar las webs según el número de políticas desactualizadas y quedarse con los 5 primeros
df_webs.sort_values(by=['desactualizadas'], inplace=True, ascending=False)
primeros_cinco = df_webs.head(5)
```

Ilustración 14

Los resultados son los siguientes:

cookies	aviso	proteccion	desactualizadas
www.nbckcip.com	0	1	0
www.zrejm.com	1	1	0
www.hocptvc.com	1	1	0
www.ldxtgwxw.com	1	0	1
www.cwngoo.com	0	1	1
www.tnvpt.com	1	0	1
www.lxkwaajoz.com	1	0	0
www.bxwtbni.com	1	0	0
www.kijwwl.com	0	0	1
www.amnoit.com	1	1	1
www.gqhnb.com	0	1	0
www.lnritj.com	0	0	1
www.qoqu.com	1	1	0
www.zzvuuuq.com	1	1	1
www.vkxbomd.com	0	1	0
www.gjww.com	1	1	0
www.jvcl.com	1	1	1
www.eohqcu.com	0	0	1
www.xshd.com	1	1	1
www.aetzc.com	1	0	1



- Mostrar según el año de creación las webs que cumplen todas las políticas de privacidad, frente a las que no cumplen la política de privacidad.

Realizamos las consultas en la tabla de legal y se construye el DataFrame, ahora creamos una columna adicional para indicar si se cumplen las políticas de seguridad o no. Posteriormente se agrupan las webs por año de creación y si satisfacen la condición anterior, contando el número de webs de cada categoría. Finalmente creamos el gráfico para mostrarlo.

```
# Consulta para obtener los datos de la tabla de legal
query_webs = 'SELECT web, creacion, cookies, aviso, proteccion FROM legal_data_online;'
df_webs = pd.read_sql_query(query_webs, conn)

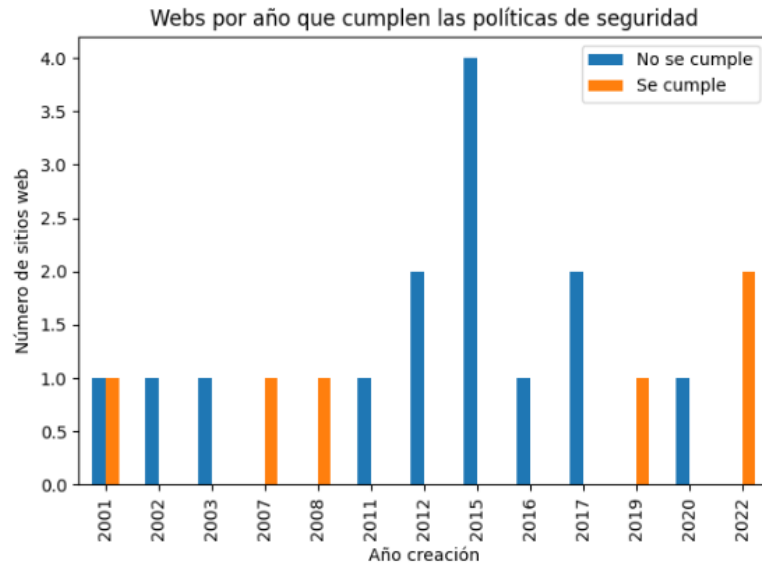
# Creación de columna para indicar si se cumplen las políticas de seguridad
df_webs['satisfacen'] = (df_webs['cookies'] == 1) & (df_webs['aviso'] == 1) & (df_webs['proteccion'] == 1)

# Agrupan las webs según el año de creación y si satisfacen las políticas, contando el número de webs de cada categoría
df_webs_agrupado_dos_columnas = df_webs.groupby(['creacion', 'satisfacen']).size().unstack()
answer = str(df_webs_agrupado_dos_columnas) + '\n'
```

Ilustración 15

Los resultados obtenidos son:

satisface	False	True
creacion		
2001	1.0	1.0
2002	1.0	NaN
2003	1.0	NaN
2007	NaN	1.0
2008	NaN	1.0
2011	1.0	NaN
2012	2.0	NaN
2015	4.0	NaN
2016	1.0	NaN
2017	2.0	NaN
2019	NaN	1.0
2020	1.0	NaN
2022	NaN	2.0



Finalmente, mostramos un ejemplo de cómo se han creado los diferentes gráficos para los apartados anteriores:

```
# Gráfico de barras correspondiente
df_webs_agrupado_dos_columnas.plot(kind='bar')
plt.title('Webs por año que cumplen las políticas de seguridad')
plt.xlabel('Año creación')
plt.ylabel('Número de sitios web')
plt.legend(['No se cumple', 'Se cumple'])
plt.tight_layout()
plt.savefig("static/images/grafico4.png")
```

Ilustración 16

CONCLUSIONES

La práctica se centra en profundizar en conceptos relacionados con la gestión de procesos de negocio y el desarrollo de sistemas de información. El escenario que se plantea implica el análisis de la necesidad de formación en seguridad de informática para los empleados de una compañía que gestionan diferentes sitios webs.

Generamos información valiosa a través de la modelización de proceso de negocio, usando UML y BPMN, el desarrollo de un sistema de información gerencial (MIS), mediante el uso de Python y sus diferentes librerías como Panda, y una base de datos SQLite. Además, se usó Flask para crear una web estática para la presentación de los diferentes informes, de una forma más visual.

En conclusión, esta práctica nos proporcionó la oportunidad de aplicar los conocimientos teóricos obtenidos en el desarrollo de las diferentes clases en un contexto empresarial real. Destacando la importancia de la integración de la información las diferentes áreas funcionales en una única base de datos, además del uso de herramientas tecnológicas para el desempeño en materia de seguridad informática y gestión de procesos de negocio.

Finalmente, este es el enlace para el repositorio de Github:

<https://github.com/fomenkoleg/PracticaSSII>