

PRÁCTICA 1 - SISTEMAS DE INFORMACIÓN

Alejandro Auni3n Ruiz, Lidia Cuellar Salinas, Daniel P3rez Prados, Fernando Rodr3guez Bellido

INGENIERÍA DE LA CIBERSEGURIDAD

Índice

Ejercicio 1.....	2
Diagrama UML:	2
Diagrama BPMN:	2
Ejercicio 2.....	3
2.1 Creación de tablas	3
2.2 Consultas	6
Ejercicio 3.....	7
Ejercicio 4.....	9
4.1 Media entre cambio de contraseñas	9
4.2 Usuarios más críticos	10
4.3 Políticas desactualizadas.....	11
4.4 Políticas según año de creación	13
Entrega de datos en entorno web.....	15
Conclusiones	21
Enlace a GitHub	22

Ejercicio 1

Diagrama UML:

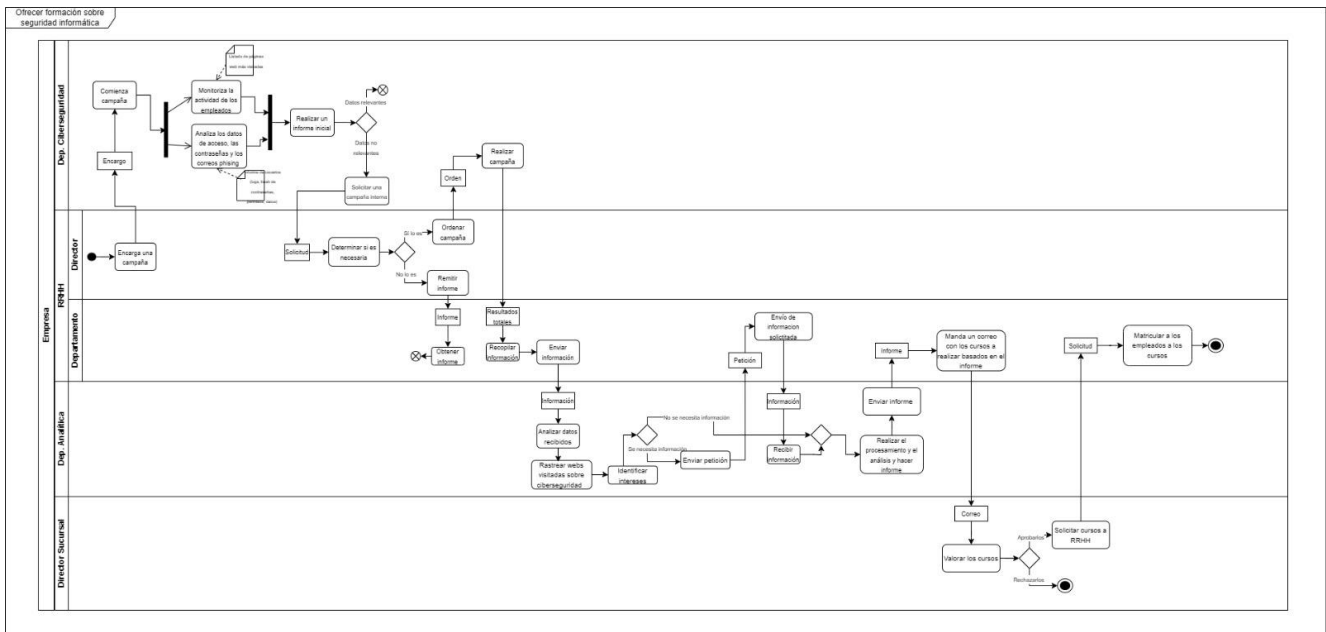
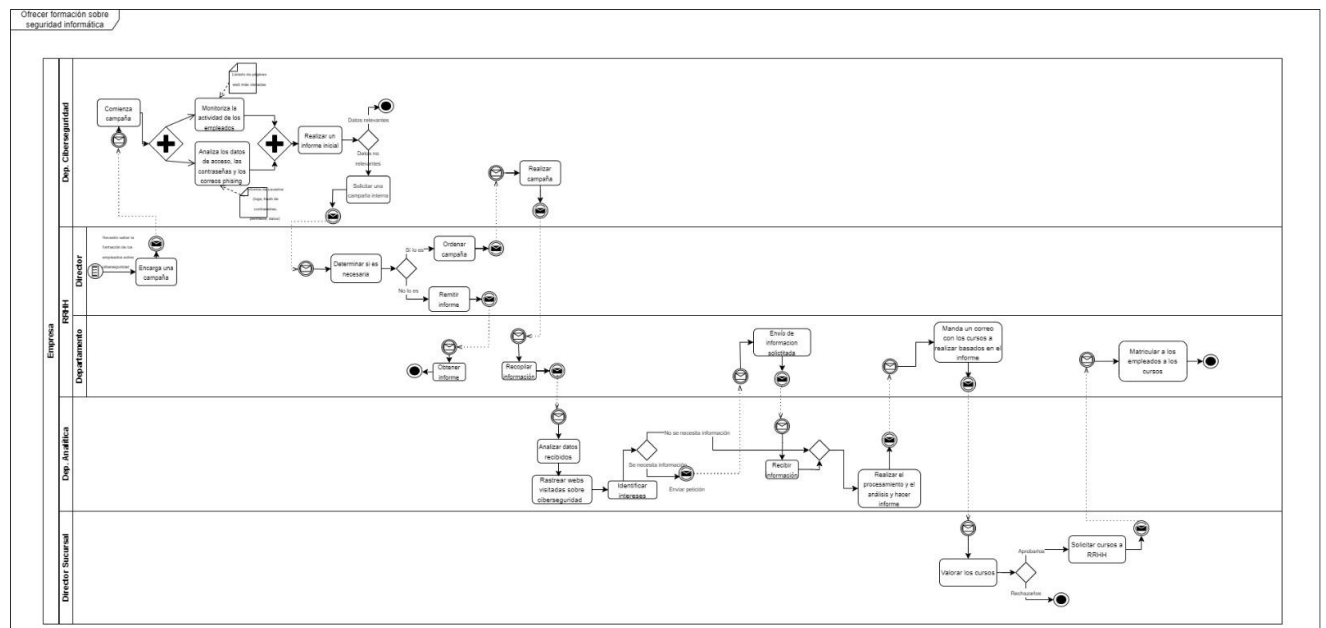


Diagrama BPMN:



Ejercicio 2

2.1 Creación de tablas

Para comenzar, debemos crear las tablas que consideramos hacen falta para poder responder los ejercicios. Comenzamos creando a partir de los JSON proporcionados, una tabla del apartado legal, donde vemos que abriremos el documento correspondiente. Realizamos un 'DROP TABLE IF EXISTS legal' para que en el caso de que exista una tabla denominada legal, no haya error y podamos operar. A continuación, creamos la tabla legal, con un id que irá en formato texto y será clave principal. Los demás atributos tendrán el tipo de datos con el que se presenten en el fichero JSON (cookies int, aviso int, proteccion_de_datos int, creación int).

Posteriormente introduciremos los datos del archivo JSON en la tabla legal, siendo, como ya hemos comentado, clave principal el ID.

```
def crearLegal():
    with open('data/legal_data_online.json', 'r') as f:
        datos = json.load(f)
        con = sqlite3.connect('datos.db')
        cur = con.cursor()

        cur.execute("DROP TABLE IF EXISTS legal;")

        cur.execute("CREATE TABLE IF NOT EXISTS legal("
                    "id TEXT PRIMARY KEY,"
                    "cookies INTEGER,"
                    "aviso INTEGER,"
                    "proteccion_de_datos INTEGER,"
                    "creacion INTEGER"
                    ");")
        con.commit()

        for elem in datos["legal"]:
            clave = list(elem.keys())[0]
            cur.execute("INSERT OR IGNORE INTO legal (id, cookies, aviso, proteccion_de_datos, creacion) \
                        VALUES ('%s', '%d', '%d', '%d', '%d') " %
                            (clave, int(elem[clave]['cookies']), int(elem[clave]['aviso']),
                             int(elem[clave]['proteccion_de_datos']), int(elem[clave]['creacion'])))
            con.commit()
        con.close()
```

Ahora crearemos la tabla usuarios, de nuevo, el ID en formato texto será nuestra clave principal, en formato texto los demás atributos serán en función de cómo se representen en el fichero JSON (teléfono int, contraseña text, segura int, está la hemos introducido posteriormente para determinar si la contraseña es o no segura; provincia text, permisos text, emails_totales int, emails_phishing int y emails_clickados int).

También creamos una tabla donde se guardaban las ip junto con las fechas vinculadas, añadimos una primary key autoincremental y la relacionamos con la tabla usuarios con una clave foránea que se referirá a la tabla usuarios, al ID del usuario. Los campos son un ide autoincremental int, fecha datetime, ip text y un user_id text, que hará de clave foránea.

```
def crearUsers():
    with open('data/users_data_online.json', 'r') as f:
        datos = json.load(f)
        con = sqlite3.connect('datos.db')
        cur = con.cursor()
        cur.execute("CREATE TABLE IF NOT EXISTS usuarios("
                    "id TEXT PRIMARY KEY,"
                    "telefono INTEGER,"
                    "contrasena TEXT,"
                    "segura INTEGER,"
                    "provincia TEXT,"
                    "permisos TEXT,"
                    "emails_totales INTEGER,"
                    "emails_phishing INTEGER,"
                    "emails_clickados INTEGER"
                    ");")
        con.commit()

        cur.execute("CREATE TABLE IF NOT EXISTS fechas_ips("
                    "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                    "fecha DATETIME,"
                    "ip TEXT,"
                    "user_id TEXT,"
                    "FOREIGN KEY (user_id) REFERENCES usuarios(id)"
                    ");")
        con.commit()
```

En cuanto a las inserciones de esta tabla, se trataba de hacer un proceso análogo a la tabla legal. No obstante, para determinar el atributo “segura”, que será 1 en caso de ser segura y 0 en caso contrario, hemos desarrollado una función que toma las contraseñas del diccionario *SmallRockyou.txt* (500 primeras líneas de *Rockyou.txt*), las hashea y las mete en un conjunto (dado que luego comprobar si una contraseña forma parte del conjunto de contraseñas de Rockyou.txt tiene una complejidad $O(1)$, mientras que hacerlo con una lista sería $O(n)$).

```
def hashComunes():
    hashes = set() # Meteremos los hash en un set, dado que comprobar si un hash pertenece al conjunto
    # Implica una complejidad O(1)
    with open('data/Smallrockyou.txt', 'r') as comunes:
        for comun in comunes:
            hashComun = hash.md5(comun[:-1].encode(encoding="utf-8")).hexdigest()
            # Evitamos incluir el salto de línea, porque el resultado del hash sería erróneo
            hashes.add(hashComun)
    return hashes
```

Luego, comprobaremos si cada contraseña de cada usuario es o no débil.

```

hashesInseguros = hashComunes()

for elem in datos["usuarios"]:
    clave = list(elem.keys())[0]

    # Para saber si el usuario tiene una contraseña segura o no
    seguridad = 0
    if elem[clave]['contrasena'] not in hashesInseguros: # Si no esta en el set de hashes inseguros, es segura
        seguridad = 1

    cur.execute(
        "INSERT OR IGNORE INTO usuarios (id, telefono, contrasena, segura, provincia, permisos, emails_totales, emails_phishi"
        "VALUES ('%s', '%s', '%s', '%d', '%s', '%s', '%d', '%d', '%d'))" %
        (clave, elem[clave]['telefono'], elem[clave]['contrasena'], seguridad,
        elem[clave]['provincia'], elem[clave]['permisos'], int(elem[clave]['emails']['total']),
        int(elem[clave]['emails']['phishing']), int(elem[clave]['emails']['clicados']))
    con.commit()

```

Como habíamos mencionado antes, la inserción es la misma que para la tabla legal, excepto en la parte en la que determinamos la seguridad de la contraseña mirando el conjunto creado con la función explicada.

```

i = 0
fechas = elem[clave]["fechas"]
for fecha in fechas:
    cur.execute("INSERT OR IGNORE INTO fechas_ips (fecha, ip, user_id)" \
                "VALUES ('%s', '%s', '%s')" %
                (fecha, elem[clave]["ips"][i], clave))
    if elem[clave]["ips"] != "None":
        i += 1
con.commit()

```

Cada usuario tenía una lista de Ips y fechas, las cuales no pueden ser insertadas en una única celda de una tabla y por ello, para cada usuario debemos recorrer estas listas para ir insertando en la tabla fechas_ips. Podía darse el caso de que no hubiera lista de ips (ips= 'None'), por ello ese condicional que evita hacer iteraciones inexistentes con el auxiliar "i".

2.2 Consultas

En este apartado se nos pide calcular diferentes valores, para ello necesitamos hacer una serie de consultas en la base de datos y sacar tres DataFrames de la siguiente forma:

```
1 usage  ↗ dprezp +2
def consultas2():

    # CONSULTAS EJ 2
    # conexión a la base de datos
    con = sqlite3.connect('datos.db')

    # Hacemos las consultas para sacar los dataFrames
    q_users = "SELECT * FROM usuarios"
    q_fechas = "SELECT * FROM fechas_ips WHERE user_id IN(SELECT id FROM usuarios)"
    q_admin = "SELECT * FROM usuarios WHERE permisos IS 1"

    # sacamos los dataFrames
    df_users = pd.read_sql_query(q_users, con)
    df_fechas = pd.read_sql_query(q_fechas, con)
    df_admins = pd.read_sql_query(q_admin, con)
```

A continuación, creamos un diccionario para más adelante poder devolverlo e imprimirlo de una forma más sencilla en la web flask. Guardamos también el número de muestras en el diccionario mediante el uso de la función count.

```
data = {} # podriamos hacerlo de golpe, pero así queda mas visual y sencillo de entender

# Numero de muestras
data["muestras_users"] = df_users['id'].count()
data["muestras_fechas"] = df_fechas['id'].count()
```

Utilizaremos también las funciones mean para calcular la media, std para la varianza y max y min para los máximos y mínimos que nos ayudarán a realizar el resto del ejercicio.

```
# Media y desviación estándar del total de fechas en las que se ha cambiado la contraseña
data["media_fechas"] = df_fechas.groupby('user_id').count().mean()['fecha']
data["desviacion_fechas"] = df_fechas.groupby('user_id').count().std()['fecha']

# Media y desviación estándar del total de IPS que se han detectado
data["media_ips"] = df_fechas.groupby('user_id').count().mean()['ip']
data["desviacion_ips"] = df_fechas.groupby('user_id').count().std()['ip']

# Media y desviación estándar del número de emails recibidos de phishing en los interactuó cualquier usuario
data["media_phishing"] = df_users['emails_phishing'].mean()
data["desviacion_phishing"] = df_users['emails_phishing'].std()

# Valor mínimo y valor máximo del total de emails recibidos
data["max_email"] = df_users['emails_totales'].max()
data["min_email"] = df_users['emails_totales'].min()

# Valor mínimo y valor máximo del número de emails phishing en los que ha interactuado un administrador
data["max_phishing_admin"] = df_admins['emails_clickados'].max()
data["min_phishing_admin"] = df_admins['emails_clickados'].min()
con.close()
return data
```

Ejercicio 3

En este ejercicio se nos pide calcular una serie de valores respecto a los mails de phishing recibidos por los usuarios. Nos piden calcular estos valores haciendo distinciones entre grupos, la primera comparativa será en función a qué permisos tienen los usuarios. Como sabemos, los usuarios solo pueden tener permisos tipo '0' o '1'.

```
def consultas3():  
    # CONSULTAS EJ 3  
    # conexion a la base de datos  
    con = sqlite3.connect('datos.db')  
  
    # Hacemos las consultas para sacar los dataFrames  
    #separamos por grupos en funcion de los permisos  
    q_users = "SELECT * FROM usuarios WHERE permisos = '0'"  
    q_admin = "SELECT * FROM usuarios WHERE permisos = '1'"  
  
    #sacamos los dataFrames  
    df_users = pd.read_sql_query(q_users, con)  
    df_admin = pd.read_sql_query(q_admin, con)
```

Primero realizaremos la conexión con la base de datos donde almacenamos la información de los usuarios. Tras conectar con esta, prepararemos las consultas que nos harán falta para separar a los usuarios en función de sus permisos, tal como indicaba el enunciado. Con las consultas, obtendremos después los DataFrames, para poder empezar a realizar el ejercicio.


```

# Numero de apariciones de Phishing en ambos grupos
print("El numero de apariciones de phishing en usuarios con permisos '0'")
print("Apariciones: ", end='')
print(df_users['emails_phishing'].sum())

print("El numero de apariciones de phishing de usuarios con permisos '1'")
print("Apariciones: ", end='')
print(df_admin['emails_phishing'].sum())

# Numero de valores ausentes de phishing en ambos grupos
print("El numero de valores ausentes de phishing en usuarios con permisos '0'")
print("Ausencias: ", end='')
print(df_users['emails_phishing'].isnull().sum())

print("El numero de valores ausentes de phishing de usuarios con permisos '1'")
print("Ausencias: ", end='')
print(df_admin['emails_phishing'].isnull().sum())

# Mediana de phishing encontrado en los usuarios de ambos grupos
print("La mediana de phishing en usuarios con permisos '0'")
print("Mediana: ", end='')
print(df_users['emails_phishing'].median())

print("La mediana de phishing de usuarios con permisos '1'")
print("Mediana: ", end='')
print(df_admin['emails_phishing'].median())

# Media de phishing encontrado en los usuarios de ambos grupos
print("La media de phishing en usuarios con permisos '0'")
print("Media: ", end='')
print(round(df_users['emails_phishing'].mean(), 2))

print("La media de phishing de usuarios con permisos '1'")
print("Media: ", end='')
print(round(df_admin['emails_phishing'].mean(), 2))

```

Comenzamos a obtener e imprimir los datos que nos solicita el enunciado mediante la aplicación de funciones como sum, median, mean, var, max y min.

Para la siguiente agrupación, vamos a determinar primero los usuarios que tienen una contraseña segura y los que no. Esto lo hacemos a la hora de insertar a los usuarios en la tabla (ejercicio 2, pág. 5), donde determinamos analizando el diccionario SmallRockYou si es segura o, por el contrario, no lo es. Si la contraseña es segura, el campo “segura” de la tabla de usuarios tomará el valor ‘1’, y si no lo fuera, tomará el valor ‘0’.

Siendo capaces de marcar estos dos grupos, las consultas serán igual que en los grupos separados por sus permisos, funcionando de la misma manera.

```

# Ahora, agrupamos por usuarios con contraseñas debiles y fuertes. Las consultas son las mismas

# Creamos las consultas
q_users_debil = "SELECT * FROM usuarios WHERE segura=0"
q_users_fuerte = "SELECT * FROM usuarios WHERE segura=1"

# Sacamos los dataFrames
df_users_fuerte = pd.read_sql_query(q_users_fuerte, con)
df_users_debil = pd.read_sql_query(q_users_debil, con)

```

Las consultas, por lo tanto, serán idénticas.

Ejercicio 4

4.1 Media entre cambio de contraseñas

Para encontrar la media de ambos tipos de usuarios (normales / permiso = 0 o administradores / permiso = 1), hacemos una consulta para crear dos DataFrames que nos permita manipular los datos.

```
def ej4_1():
    con = sqlite3.connect('datos.db')

    q_users = ("SELECT * FROM fechas_ips "
               "WHERE user_id IN "
               "(SELECT id FROM usuarios WHERE permisos == '0');")

    q_admins = ("SELECT * FROM fechas_ips "
                "WHERE user_id IN "
                "(SELECT id FROM usuarios WHERE permisos == '1');")

    # Crear dataframes a partir de las consultas
    df_users = pd.read_sql_query(q_users, con)
    df_admins = pd.read_sql_query(q_admins, con)
```

Como en la inserción de datos habíamos establecido nuestra columna fecha como tipo datetime, ajustamos el formato para poder ordenar las fechas para poder calcular la diferencia de tiempo entre los cambios de contraseña realizadas. Esta diferencia se almacena en una nueva columna, agrupándolas por cada usuario existente.

```
# Ajustar el formato de las fechas de string a datetime
df_users['fecha'] = pd.to_datetime(df_users['fecha'], format="%d/%m/%Y")
df_admins['fecha'] = pd.to_datetime(df_admins['fecha'], format="%d/%m/%Y")

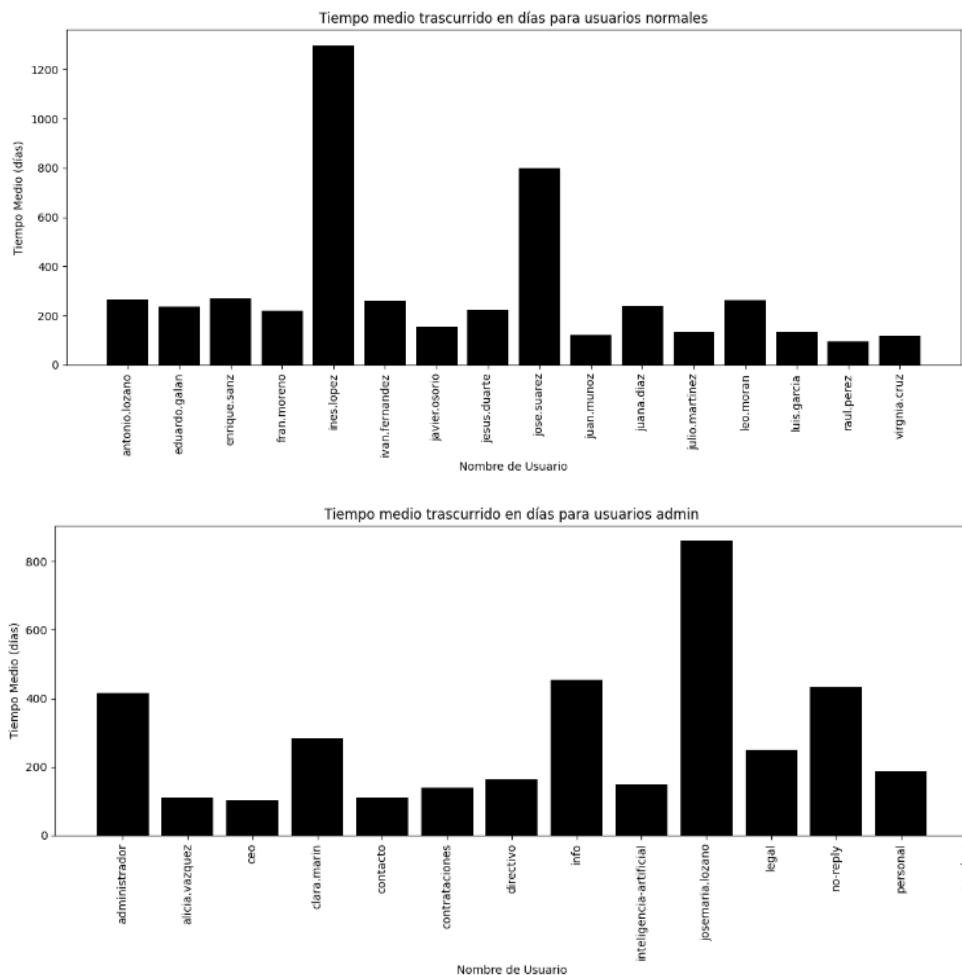
# Ordenar los dataframes segun el nombre de usuario y la fecha
df_users = df_users.sort_values(by=['user_id', 'fecha'])
df_admins = df_admins.sort_values(by=['user_id', 'fecha'])

# Crear la columna diferencia en los dataframes con la diferencia de días desde el cambio anterior
df_users['diferencia'] = pd.to_numeric(df_users.groupby('user_id')['fecha'].diff().dt.days)
df_admins['diferencia'] = pd.to_numeric(df_admins.groupby('user_id')['fecha'].diff().dt.days)
```

Realizamos la media de cada usuario, diferenciada por tipo.

```
# Hacer la media de las diferencias en cada usuario
media_users = df_users.groupby('user_id')['diferencia'].mean().reset_index()
media_admins = df_admins.groupby('user_id')['diferencia'].mean().reset_index()
```

Por último, creamos los gráficos para mostrarlas.



4.2 Usuarios más críticos

Para mostrar los usuarios más críticos, en primer lugar hacemos una consulta para crear el DataFrame con el nombre de usuario, los emails de phishing que han enviado al usuario y en cuantos ha hecho click.

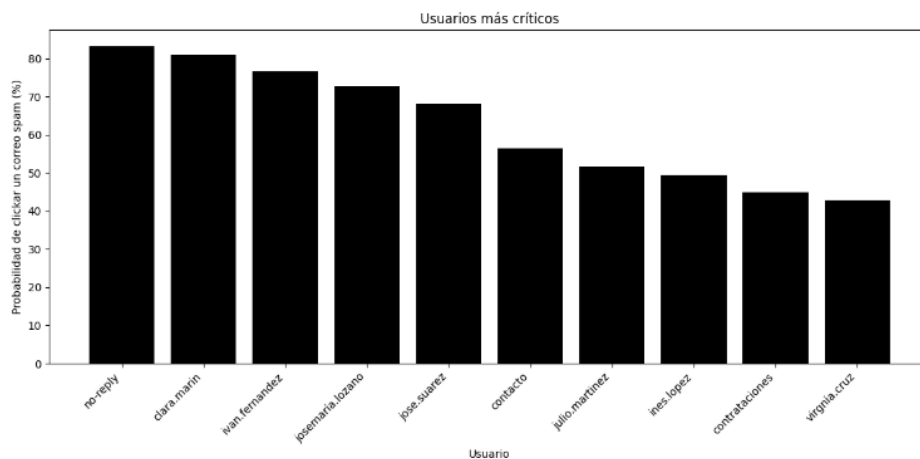
```
q_inseguro = ("SELECT id, emails_clickados, emails_phishing "
              "FROM usuarios WHERE segura IS 0 AND emails_clickados IS NOT 0 "
              "AND emails_phishing IS NOT 0 ;")

df_inseguro = pd.read_sql_query(q_inseguro, con)
```

Con estos datos, calcularemos las probabilidades de cada usuario de caer en este tipo de ataque. Nos quedamos con los 10 que mayor probabilidad tengan y los mostramos.

```
# Calcular probabilidad de éxito para ataques de phishing
df_inseguro['probabilidad'] = df_inseguro['emails_clickados'] / df_inseguro['emails_phishing'] * 100
df_inseguro.sort_values(by='probabilidad', ascending=False, inplace=True)

# Seleccionar los primeros 10 usuarios con más probabilidad
df_inseguro = df_inseguro.head(10)
```



4.3 Políticas desactualizadas

En este ejercicio se nos pide mostrar las 5 páginas web que contienen más políticas desactualizadas y representarlas en un gráfico de barras según las políticas.

Se entiende que las políticas desactualizadas se refieren a no tener implementado el sistema de cookies, avisos o protección de datos.

Para ello, hemos elaborado una función que comienza realizando una consulta para hacer un DataFrame de la siguiente manera:

```
def ej4_3():
    #conectamos la base de datos
    con = sqlite3.connect('datos.db')

    #Se realiza la consulta y se saca el dataframe
    q_webs='SELECT id, cookies, aviso, proteccion_de_datos FROM legal;'
    df_webs = pd.read_sql_query(q_webs, con)
```

Una vez que tenemos el dataframe le añadimos una columna llamada “deprecated” que contendrá el recuento de cuántas de las columnas cookies, avisos y protección de datos tienen el valor 0 en cada fila del DataFrame, es decir están desactualizadas, y las ordenamos de mayor a menor para quedarnos con las 5 primeras.

```
#Se calcula la cantidad de políticas deprecated
df_webs['deprecated'] = df_webs[['cookies', 'aviso', 'proteccion_de_datos']].apply(lambda row: sum(row==0), axis=1)

# ordenar las webs según deprecated (cogemos 5 primeros)
df_webs.sort_values(by=['deprecated'], inplace=True, ascending=False)
first_five = df_webs.head(5)
```

Más tarde nombramos la gráfica, establecemos los títulos de la X y la Y y creamos el gráfico de barras de la siguiente forma.

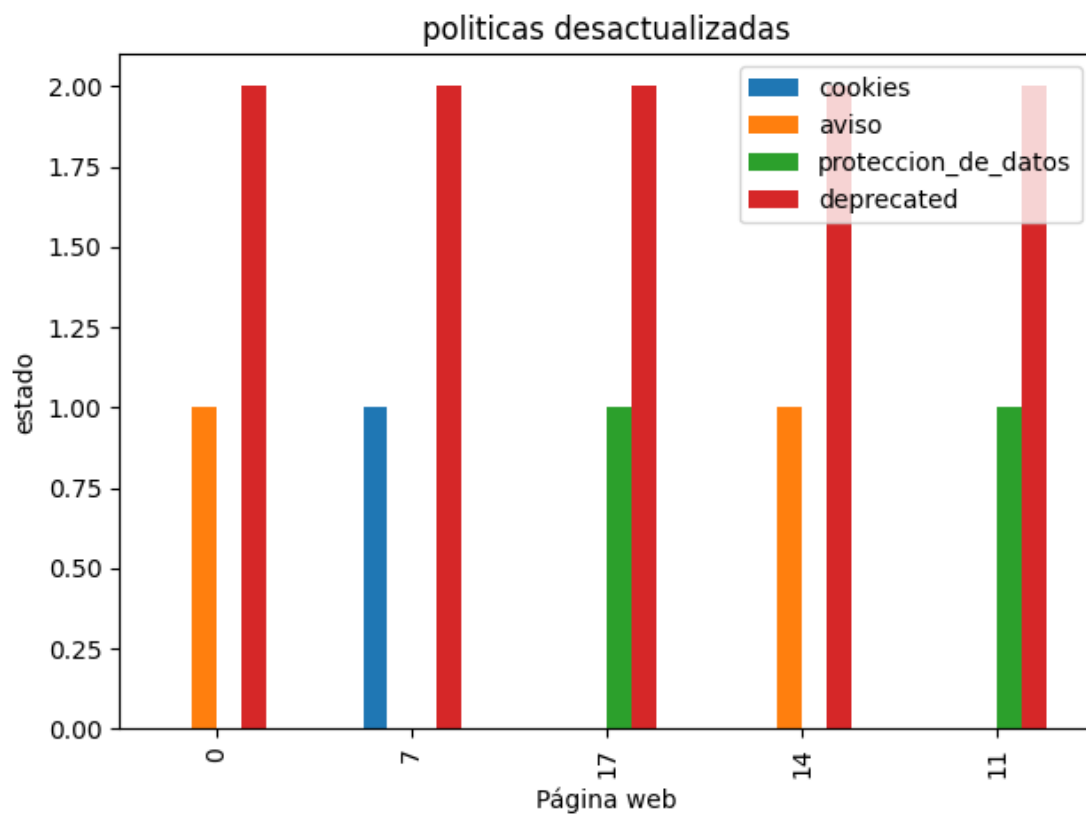
```

#nombre de la gráfica
df_webs = df_webs.set_index('id')
print(df_webs)

#crear gráfico de barras correspondientes
first_five.plot(kind='bar')
plt.title('políticas desactualizadas')
plt.xlabel('Página web')
plt.ylabel('estado')
plt.legend()
plt.tight_layout()
plt.show()
#cerrar base de datos
con.close()

```

El gráfico de barras resultante es el siguiente:



4.4 Políticas según año de creación

En este apartado mostraremos según el año de creación las webs que cumplen todas las políticas de privacidad frente a las que no lo hacen.

Se entiende que cumplen las políticas de seguridad cuando tienen presente las cookies, la protección de datos y los avisos.

Para ello, haremos una query para sacar un dataframe de las páginas web de la parte legal de la base de datos.

```
def ej4_4():
    con = sqlite3.connect('datos.db')

    #Se realiza la consulta y se saca el dataframe como en el ejercicio anterior
    q_webs='SELECT id, cookies, aviso, proteccion_de_datos, creacion FROM legal;'
    df_webs = pd.read_sql_query(q_webs, con)
```

Creamos un distintivo de políticas de seguridad en el DataFrame haciendo un distintivo de aquellas ids que coinciden que tengan la protección de datos, el aviso y las cookies.

```
#Creamos un distintivo de politicas de seguridad
df_webs['dist'] = (df_webs['proteccion_de_datos'] == 1) & (df_webs['aviso'] == 1) & (df_webs['cookies'] == 1)

#lo ordenamos por año de creación y con el distintivo
df_webs_ordenado = df_webs.groupby(['creacion', 'dist']).size().unstack()
print(df_webs_ordenado)
```

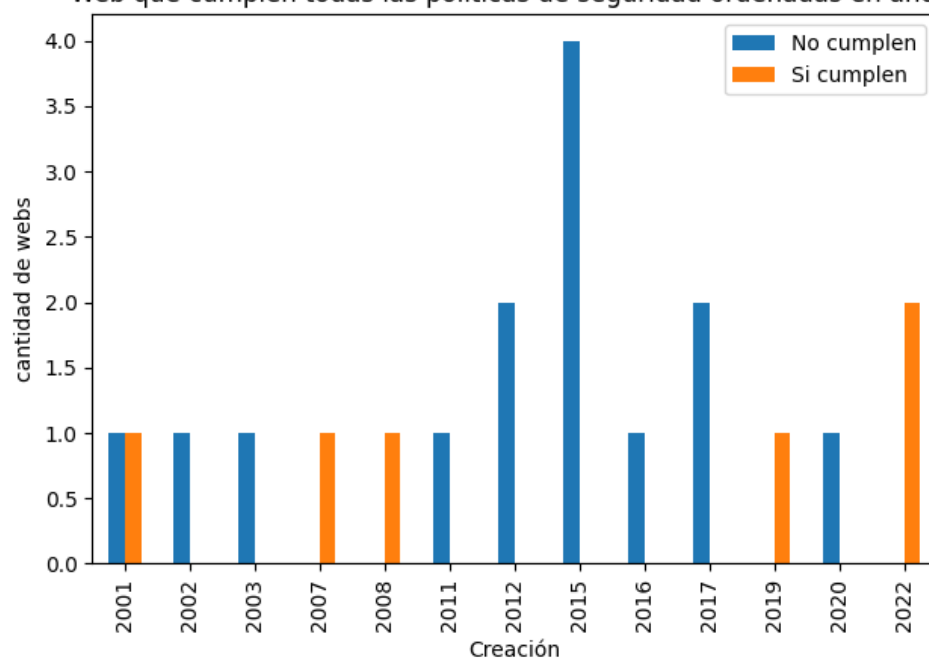
Ordenamos por el año de creación y con el distintivo además de la cantidad de webs que tienen o no las políticas y creamos el gráfico como en apartados anteriores.

```
#lo ordenamos por año de creación y con el distintivo
df_webs_ordenado = df_webs.groupby(['creacion', 'dist']).size().unstack()
print(df_webs_ordenado)

#Sacamos el gráfico
df_webs_ordenado.plot(kind='bar')
plt.title('Web que cumplen todas las políticas de seguridad ordenadas en años ')
plt.xlabel('Creación')
plt.ylabel('cantidad de webs')
plt.legend(['No cumplen', 'Si cumplen'])
plt.tight_layout()
plt.show()
con.close()
```

El gráfico que obtenemos es el siguiente:

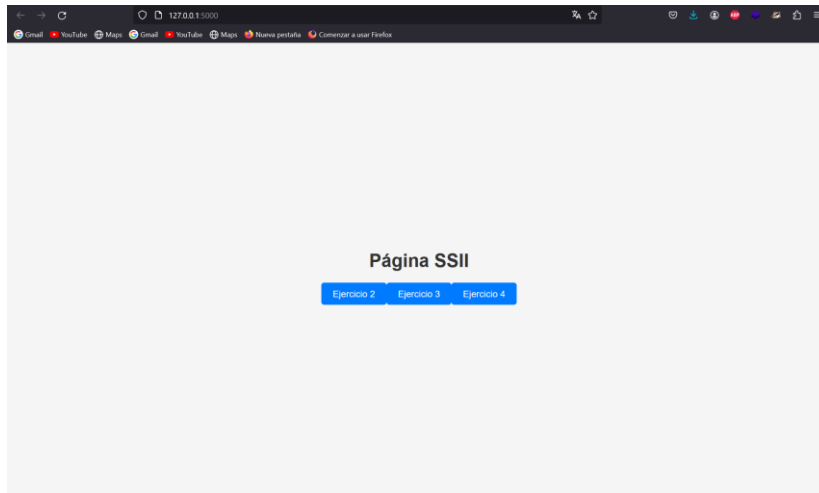
Web que cumplen todas las políticas de seguridad ordenadas en años



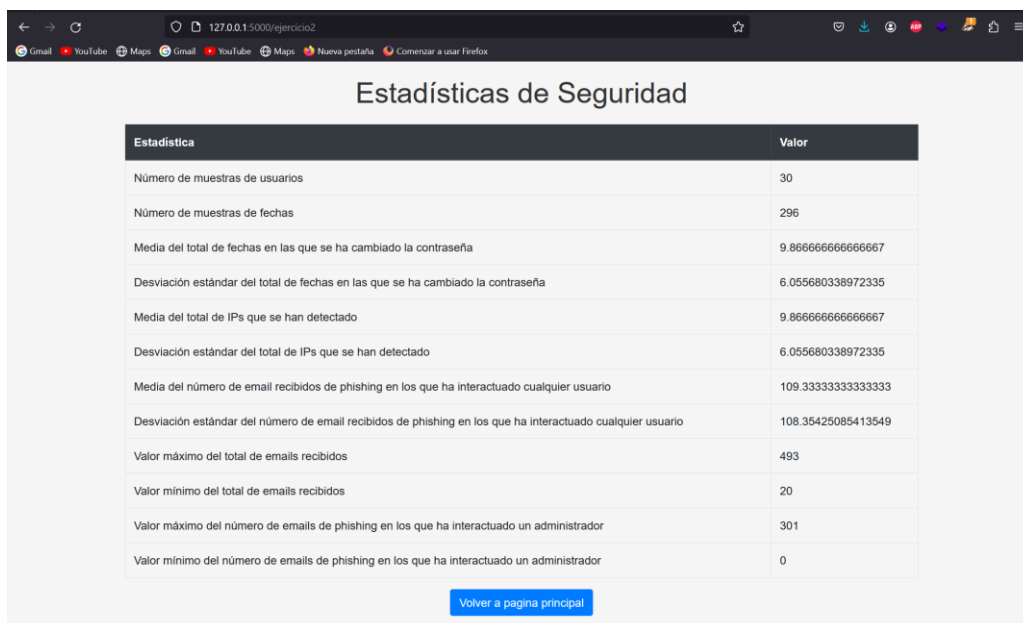
Entrega de datos en entorno web

Una vez realizados todos los ejercicios anteriores (los cuales se mostraban en la terminal de Python, para poder corroborar que funcionaban), llegamos a la última parte de esta práctica. Mediante Flask, mostraremos los resultados de apartados anteriores por medio de una página web estática.

Para darle un poco de estilización, hemos utilizado Bootstrap en los HTML; además de requerir las librerías *io* y *base64* para poder representar los gráficos del ejercicio 4 en el HTML (codificar en base64 al realizar los cálculos, y decodificar en el HTML).



Cuando ejecutamos la aplicación y vamos a la dirección localhost:5000 (impuesta por Flask), esto es lo que se nos muestra. Se trata de un HTML sencillo con 3 enlaces para acceder a la información de los apartados correspondientes. El aspecto de los enlaces está creado (como hemos mencionado antes) con Bootstrap.



Estadística	Valor
Número de muestras de usuarios	30
Número de muestras de fechas	296
Media del total de fechas en las que se ha cambiado la contraseña	9.866666666666667
Desviación estándar del total de fechas en las que se ha cambiado la contraseña	6.055680338972335
Media del total de IPs que se han detectado	9.866666666666667
Desviación estándar del total de IPs que se han detectado	6.055680338972335
Media del número de email recibidos de phishing en los que ha interactuado cualquier usuario	109.33333333333333
Desviación estándar del número de email recibidos de phishing en los que ha interactuado cualquier usuario	108.35425085413549
Valor máximo del total de emails recibidos	493
Valor mínimo del total de emails recibidos	20
Valor máximo del número de emails de phishing en los que ha interactuado un administrador	301
Valor mínimo del número de emails de phishing en los que ha interactuado un administrador	0

Volver a pagina principal

Esto es lo que vemos si entramos al ejercicio 2. Hemos organizado la información en una tabla, para que quede mejor estéticamente. Hay un botón para retroceder a la página principal.

Para poder pasarle la información correctamente a Flask y que se interprete debidamente en el *template*, hemos modificado las funciones de todos los ejercicios, para que, en vez de imprimir la información por pantalla en la terminal del IDE, se guarde toda en un diccionario. Veamos el ejemplo con el ejercicio 2 (el 3 es exactamente igual):

```
data = {} # podriamos hacerlo de golpe, pero asi queda mas visual y sencillo de entender

# Numero de muestras
data["muestras_users"] = df_users['id'].count()
data["muestras_fechas"] = df_fechas['id'].count()

# Media y desviación estándar del total de fechas en las que se ha cambiado la contraseña
data["media_fechas"] = df_fechas.groupby('user_id').count().mean()['fecha']
data["desviacion_fechas"] = df_fechas.groupby('user_id').count().std()['fecha']

# Media y desviación estándar del total de IPS que se han detectado
data["media_ips"] = df_fechas.groupby('user_id').count().mean()['ip']
data["desviacion_ips"] = df_fechas.groupby('user_id').count().std()['ip']

# Media y desviación estándar del número de emails recibidos de phishing en los interactuados cualquier usuario
data["media_phishing"] = df_users['emails_phishing'].mean()
data["desviacion_phishing"] = df_users['emails_phishing'].std()

# Valor mínimo y valor máximo del total de emails recibidos
data["max_email"] = df_users['emails_totales'].max()
data["min_email"] = df_users['emails_totales'].min()

# Valor mínimo y valor máximo del número de emails phishing en los que ha interactuado un administrador
data["max_phishing_admin"] = df_admins['emails_clickados'].max()
data["min_phishing_admin"] = df_admins['emails_clickados'].min()
con.close()
return data
```

Lo que antes era: “print(información)” ahora es data[“información_clave”]= información. Con esto, ya le podemos mandar la información a la plantilla, renderizarla con flask y mostrarla:

```
@app.route('/ejercicio2')
def ej2():
    data2 = tablas.consultas2()
    return render_template(template_name_or_list='ejercicio2.html', **data2)
```

Utilizamos ****data2**, porque las claves utilizadas en el diccionario son las mismas que tenemos en el *template* puestas entre llaves (“{{ clave }}”). Con esto, evitamos referenciar todas las claves que utiliza la plantilla:

```
<tr>
    <td>Valor máximo del número de emails de phishing en los que ha interactuado un administrador</td>
    <td>{{ max_phishing_admin }}</td>
</tr>
```

En cuanto a la tabla que se ve, es una tabla sencilla en HTML, con una clase de Bootstrap, la cual le da ese detalle extra de estilos.

Para el ejercicio 3 ocurre exactamente lo mismo:

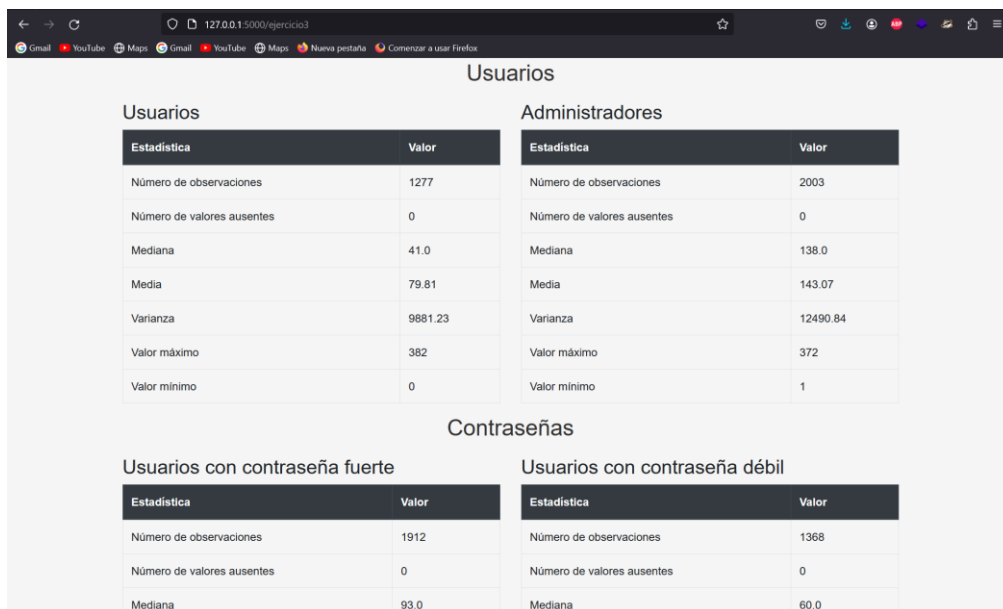
```
# Usuarios con contraseña debil

data["phishing_debil"] = df_users_debil['emails_phishing'].sum()
data["ausencia_debil"] = df_users_debil['emails_phishing'].isnull().sum()
data["mediana_debil"] = df_users_debil['emails_phishing'].median()
data["media_debil"] = round(df_users_debil['emails_phishing'].mean(), 2)
data["varianza_debil"] = round(df_users_debil['emails_phishing'].var(), 2)
data["max_debil"] = df_users_debil['emails_phishing'].max()
data["min_debil"] = df_users_debil['emails_phishing'].min()

con.close()
return data
```

Fragmento de código del ejercicio 3

Lo único diferente es la página estática:



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/ejercicio3'. The page content is organized into four tables under the heading 'Usuarios'.

Estadística	Valor
Número de observaciones	1277
Número de valores ausentes	0
Mediana	41.0
Media	79.81
Varianza	9881.23
Valor máximo	382
Valor mínimo	0

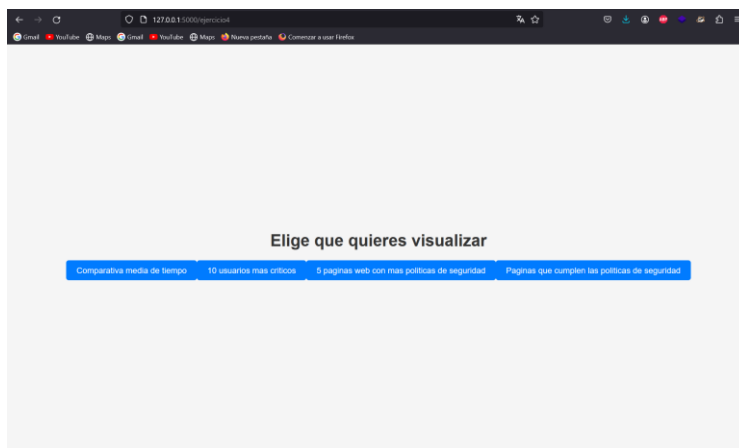
Estadística	Valor
Número de observaciones	2003
Número de valores ausentes	0
Mediana	138.0
Media	143.07
Varianza	12490.84
Valor máximo	372
Valor mínimo	1

Estadística	Valor
Número de observaciones	1912
Número de valores ausentes	0
Mediana	93.0

Estadística	Valor
Número de observaciones	1368
Número de valores ausentes	0
Mediana	60.0

Hemos puesto 4 tablas (idénticas a la anterior) con los datos requeridos en dos filas.

Para mostrar el ejercicio 4 hemos requerido pasar los gráficos a base 64. Luego, los decodificamos y mostramos.



Al igual que en el inicio, hemos separado el ejercicio 4 en apartados, para verlo mejor.

```
plt.tight_layout()
buf = io.BytesIO()
plt.savefig(*args: buf, format='png')
buf.seek(0)
data_uri = base64.b64encode(buf.read()).decode('utf-8') # pasamos el grafico a base64
# Esto lo hacemos para que pueda ser pasado en un formato aceptable al html para luego mostrarlo
plt.close()
con.close()

# Creamos un diccionario para poder enviarle a Flask los datos, con las mismas claves que las usadas en el html
return {"data_uri": data_uri, "df_inseguro": df_inseguro}
```

Esta es la modificación más relevante (y aplicable a los 4 apartados) con respecto a lo contado en el ejercicio 4. Una vez tenemos el gráfico creado tenemos que pasarlo a base64. Primeramente, generamos un buffer con binarios que necesitaremos para guardar la imagen en formato png (líneas 2 y 3). En buf por tanto, tenemos el gráfico. Con seek(0) posicionamos el puntero al inicio del buffer para comenzar a *encodear* en base64 posteriormente (y pasar los bytes a letras, de ahí el decode('UTF-8')). Esta información, junto a los datos obtenidos son enviados al controlador que renderizará la plantilla y mostrará el gráfico y la información por pantalla:



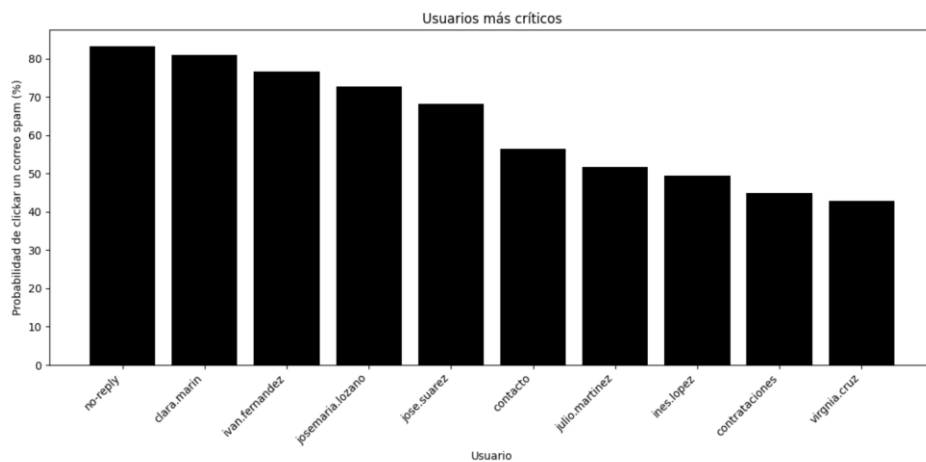
En este ejemplo (ejercicio 4.3) vemos como el gráfico se muestra correctamente, y bajo él, la tabla con las cinco políticas desactualizadas.

Webs que cumplen todas las políticas de seguridad ordenadas por años



4.4

Usuarios más críticos



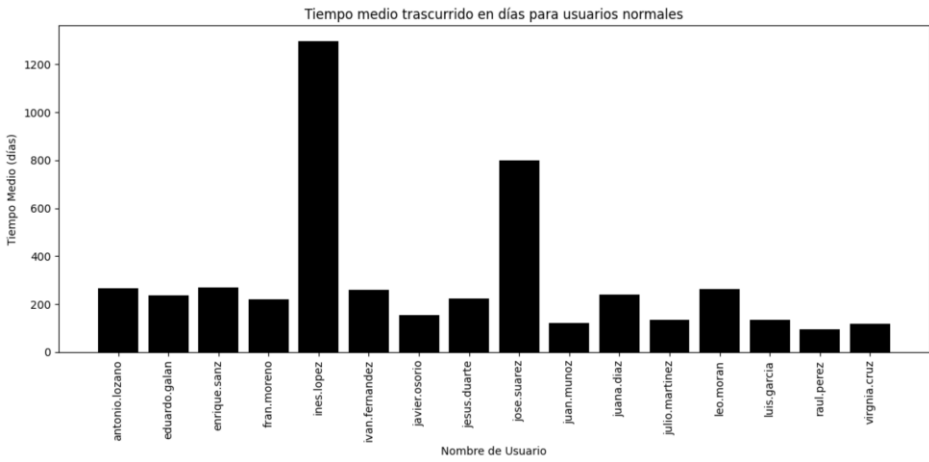
Datos de usuarios más críticos

Usuario	Emails Clickados	Emails Phishing	Probabilidad (%)
no-reply	25	30	83.33333333333334
clara.marin	301	372	80.91397849462365
ivan.fernandez	46	60	76.66666666666667
josemaria.lozano	16	22	72.72727272727273
jose.suarez	86	126	68.25396825396825
contacto	13	23	56.52173913043478
julio.martinez	117	226	51.76991150442478
ines.lopez	46	93	49.46236559139785
contrataciones	92	205	44.87804878048781
virginia.cruz	9	21	42.857142857142854

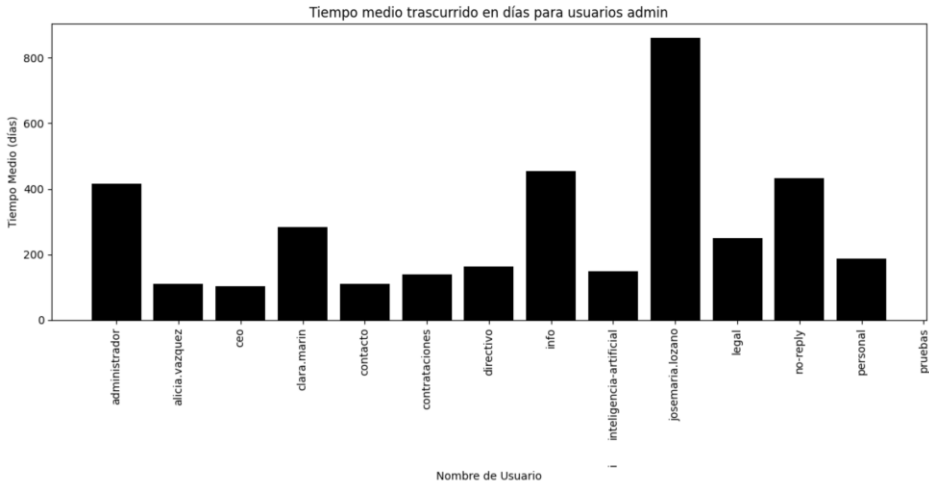
[Volver a pagina de ejercicio 4](#) [Volver a pagina principal](#)

4.2

Tiempo medio transcurrido en días para usuarios normales



Tiempo medio transcurrido en días para usuarios admin



Usuarios normales

Usuario	Tiempo Medio (días)
antonio.lozano	265.125
eduardo.galan	236.5
enrique.sanz	270.5
fran.moreno	220.375
ines.lopez	1298.0
ivan.fernandez	261.3333333333333
javier.osorio	156.2
jesus.duarte	225.0
jose.suarez	799.0
juan.munoz	120.61111111111111
juana.diaz	241.44444444444446
julio.martinez	134.69230769230768
leo.moran	261.875
luis.garcia	135.92857142857142
raul.perez	96.63157894736842
virgini.cruz	117.47058823529412

Usuarios admin

Usuario	Tiempo Medio (días)
administrador	416.5
alicia.vazquez	110.0
ceo	102.21052631578948
clara.marin	284.2
contacto	109.78947368421052
contrataciones	138.14285714285714
directivo	162.46153846153845
info	453.6666666666667
inteligencia-artificial	147.33333333333334
josemaria.lozano	862.0
legal	248.33333333333334
no-reply	432.6666666666667
personal	186.66666666666666
pruebas	nan

[Volver a pagina de ejercicio 4](#)

[Volver a pagina principal](#)

Conclusiones

En conclusión, como se ha demostrado, se puede identificar la importancia de un análisis completo y minucioso de la información disponible. Dados los resultados de esta revisión y la crítica de los datos, el lector adquiere una visión más clara y precisa de la situación en estudio. Junto con esto, es esencial resaltar la relevancia de las herramientas y métodos utilizados para analizar los datos. Se puede afirmar que la elección de la correcta metodología y las técnicas estadísticas aplicadas garantizan la validez y fiabilidad de los resultados que hemos obtenido.

También se ha evidenciado la importancia de incorporar el contexto y las variables adecuadas mientras se efectuaba el análisis. El medio en el que se realiza el estudio y factores que pueden determinar los resultados son aspectos que considerar, ya que pueden influir en la interpretación de los datos y las conclusiones. Además, otra conclusión es la necesidad de mantener una actitud crítica y reflexiva durante el análisis.

La exhaustividad en el análisis, el uso adecuado de herramientas y métodos, la consideración del contexto, la actitud crítica y la claridad en la comunicación de resultados son elementos clave a considerar en futuros proyectos.

Enlace a GitHub

El enlace a nuestro repositorio de github es el siguiente:

<https://github.com/dprezp/domadores>