



SISTEMAS DE INFORMACIÓN
GRADO EN INGENIERIA DE LA CIBERSEGURIDAD
CURSO ACADÉMICO 2024-2025
CONVOCATORIA 2Q

PRACTICA 1

AUTOR(A): Fuentes Contreras, Alejandro

AUTOR(A): García Márquez, Javier

AUTOR(A): Suárez Suárez, Juan Antonio

Contenido

- ENLACE REPOSITORIO GITHUB..... 3
- 1. EJERCICIO 1..... 3
- 2. EJERCICIO 2..... 5
 - 2.9. RESULTADOS 7
- 3. EJERCICIO 3..... 8
- 4. EJERCICIO 4..... 11
- 5. ENTORNO WEB..... 16
- BIBLIOGRAFIA 18

ENLACE REPOSITORIO GITHUB

[Práctica 1 Grupo 18 Github](#)

1. EJERCICIO 1

Esta parte del documento recoge el modelado del proceso de negocio para la gestión de incidencias de ciberseguridad en una empresa tecnológica. En concreto, se han realizado modelados en dos notaciones diferentes: *Business Process Modeling Notation* (BPMN) y *Unified Modeling Language* (UML).

El proceso de modelado comienza con la solicitud del gerente de operaciones de seguridad (SOC) a los analistas de seguridad para evaluar las incidencias recibidas. A partir de esta evaluación, se determina si se debe dar formación a los empleados, automatizar algún proceso o marcarlo para una revisión posterior.

La decisión tomada implicará a otros departamentos como recursos humanos y desarrollo, quienes implementarán las soluciones correspondientes.

Por último, se realiza un seguimiento de las mejoras para evaluar su impacto en la organización.

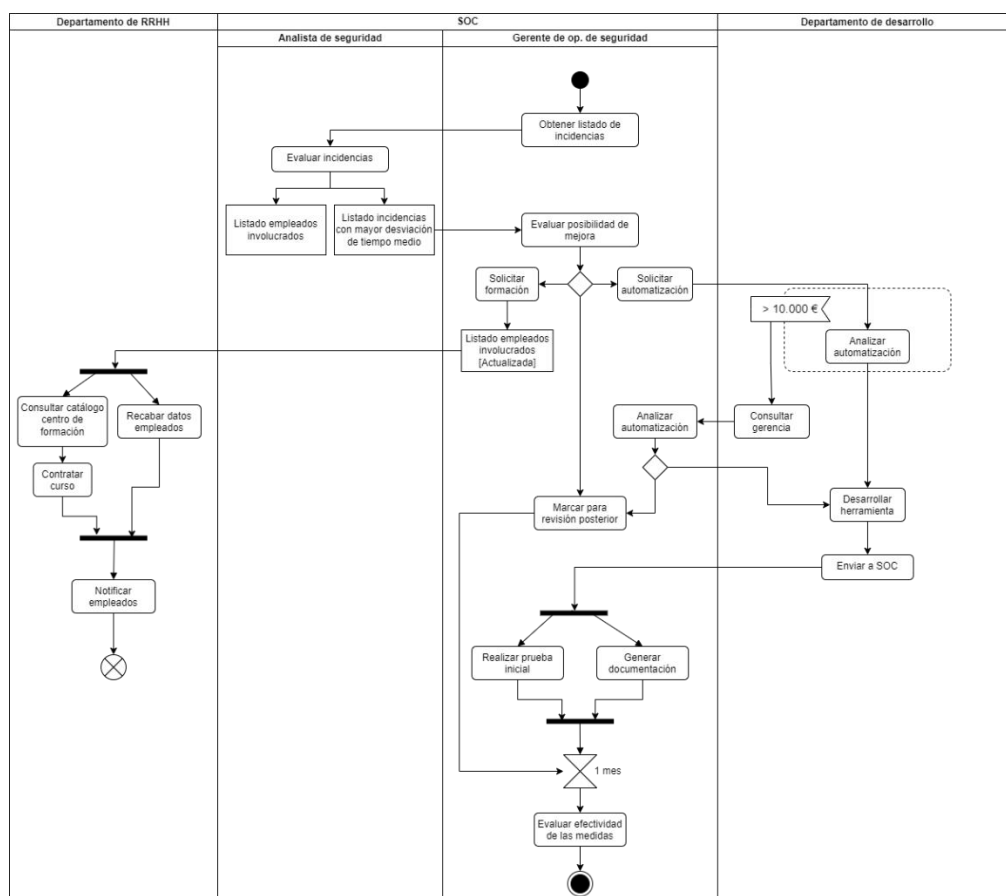


Ilustración 1. Modelado de proceso UML

Como se puede observar en el diagrama UML, una vez bifurcado el flujo en el gerente de operaciones, el departamento de RRHH realiza sus acciones y, una vez notificados los

empleados, termina su flujo, es decir, a partir de ahí ya no se realizan más tareas que afecten al flujo general.

Al inicio de la actividad del departamento de desarrollo nos encontramos una región interrumpible. En este caso, si el coste de la automatización supera lo 10.000 € habrá que consultar con el gerente de operaciones, en otro caso, se continuará con su desarrollo.

Por último, todo el flujo viene a converger en un punto de espera de 1 mes, después del cual se valorará la efectividad de las medidas tomadas y pondremos final al flujo.

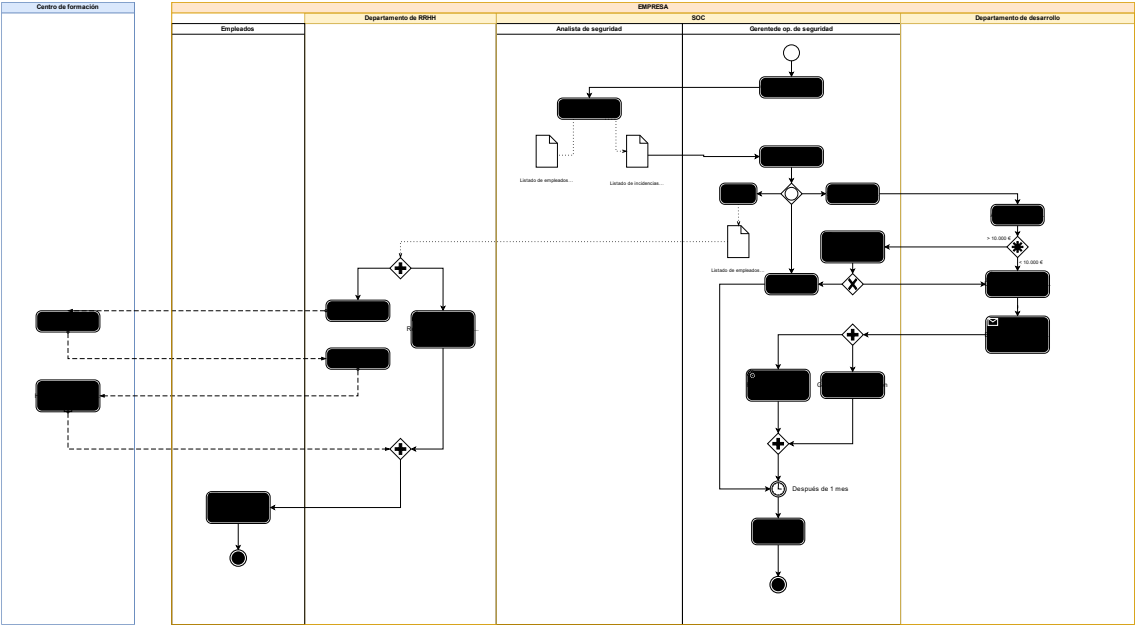


Ilustración 2. Modelado de proceso BPMN


Como se puede observar el diagrama BPMN nos ofrece mucha más flexibilidad y posibilidades de representar de manera más correcta parte del flujo.

Lo que podemos destacar del diagrama es la adición de dos actores más, los empleados y el centro de formación. Este último tienen su *pool* separado del resto pues en BPMN los *pools* pertenecientes a una misma organización estarán juntos, mientras que los que sean externos estarán en otro diferente.

Ahora, el flujo de RRHH intervendrá con los empleados y con el centro de formación.

Las distintas actividades utilizadas son:

SCRIPT		Esta tarea es automatizada, por ello el uso de esta actividad.
SUB-PROCESS		Esta tarea muestra una tarea que debería dividirse en más subprocesos.
USER		Esta actividad requiere la intervención de un usuario para su realización.
SERVICE		Esta tarea es ejecutada automáticamente por un sistema externo o un servicio sin intervención humana.

SEND		Esta actividad envía un mensaje a otro sistema o entidad.
------	-----------------------------------------------------------------------------------	-----------------------------------------------------------

En este caso, en vez de una región de interrupción, tenemos una bifurcación “compleja”, es decir, según las condiciones que se den (si el presupuesto es mayor o menor de 10.000 €) se ramificará hacia un lado u otro. Además, si el presupuesto se supera y pasa a revisión del gerente de operaciones, este deberá decidir si se continúa con el desarrollo o se marca para una revisión posterior, de ahí la bifurcación ‘*exclusive*’, que sólo permitirá tomar un flujo.

Los dos diagramas anteriormente presentados se encuentran subidos en el repositorio de GitHub de esta práctica: <https://github.com/juansrz/SI-Practica1-Grupo18/tree/main>

2. EJERCICIO 2

En esta parte de la práctica, se implementará un sencillo sistema ETL que nos permita estructurar y analizar los datos provenientes de un archivo JSON que se nos proporciona, almacenarlos en una base de datos SQLite y, posteriormente, facilitar su consulta para la toma de decisiones en la gestión de incidencias.

Todo el código se puede consultar en el siguiente repositorio de GitHub: <https://github.com/juansrz/SI-Practica1-Grupo18/tree/main>

En primer lugar, en el archivo main.py, comenzaremos con la extracción y transformación de datos.

El código comienza con la creación de la base de datos mediante SQLite, se crea la tabla y las filas y columnas necesarias para la representación de los datos de nuestro JSON.

Después procedemos a la apertura del archivo “datos.json”, que contiene la información sobre clientes, empleados, tipos de incidentes y tickets emitidos. A continuación, utilizamos json.load() para cargar los datos en memoria en forma de diccionario.

Cada conjunto de datos dentro del JSON (clientes, empleados, tickets, etc.) se almacena en listas dentro del diccionario.

La carga de datos sigue un orden estratégico para garantizar la integridad:

1. Eliminamos las tablas preexistentes con DROP TABLE IF EXISTS para evitar conflictos de posibles datos obsoletos.
2. Creamos las tablas con CREATE TABLE, asegurándonos de que las relaciones entre ellas están correctamente definidas mediante claves primarias y foráneas.
3. Insertamos los datos en las tablas correspondientes mediante INSERT INTO.

Por último, para validar la carga de datos, usaremos Pandas para leer y mostrar información básica:

- `pd.read_sql_query("SELECT * FROM tickets;", conn)`: Obtiene todos los tickets registrados.
- `len(df_tickets)`: Muestra el número total de incidencias registradas.
- `pd.read_sql_query("SELECT * FROM contactos_empleados;", conn)`: Permite analizar qué empleados han participado en la solución de incidencias.

2.1. Análisis de Datos en analisis.py

Una vez que los datos están almacenados en la base de datos, se extraen para realizar diferentes cálculos estadísticos.

2.2. Número total de muestras

Este valor indica cuántas incidencias están registradas en la base de datos.

```
# 1) Número de incidencias totales
total_incidencias = len(df_tickets)
```

Ilustración 3. Total incidencias.

2.3. Media y desviación estándar del total de incidentes con valoración ≥ 5

- Filtramos las incidencias con `satisfaccion_cliente` ≥ 5 .
- `groupby("cliente").size()` agrupa las incidencias por cliente y cuenta cuántas tiene cada uno.
- Calculamos la media (`mean()`) y la desviación estándar (`std()`).

```
df_satis_5 = df_tickets[df_tickets["satisfaccion_cliente"] >= 5]
num_satis_5 = len(df_satis_5)

# Media y desviación std del total de incidencias (con sat >=5) por cliente
group_satis = df_satis_5.groupby("cliente").size()
media_satis_5 = group_satis.mean()
std_satis_5 = group_satis.std(ddof=1)
```

Ilustración 4. Satisfacción cliente

2.4. Media y desviación estándar del número de incidentes por cliente

- Se agrupan todas las incidencias por cliente.
- Se calcula la media de incidencias por cliente y la desviación estándar.

```
group_incidencias = df_tickets.groupby("cliente").size()
media_incid = group_incidencias.mean()
std_incid = group_incidencias.std(ddof=1)
```

Ilustración 5. Media y desviación

2.5. Media y desviación estándar del total de horas realizadas en cada incidente

- Se agrupan los tiempos trabajados (`tiempo`) por cada incidente (`id_ticket`).
- Se calcula la media y desviación estándar de horas trabajadas en cada incidencia.

```

horas_por_ticket = df_contactos.groupby("id_ticket")["tiempo"].sum().reset_index()
horas_por_ticket.columns = ["id_ticket", "horas_totales_incidente"]
media_horas = horas_por_ticket["horas_totales_incidente"].mean()
std_horas = horas_por_ticket["horas_totales_incidente"].std(ddof=1)

```

Ilustración 6. Media y desviación horas trabajadas

2.6. Valores mínimos y máximos de horas trabajadas por empleado

- Se agrupa el tiempo trabajado por empleado (id_emp).
- Se obtiene el mínimo y máximo de horas trabajadas.

```

horas_por_empleado = df_contactos.groupby("id_emp")["tiempo"].sum().reset_index()
min_horas = horas_por_empleado["tiempo"].min()
max_horas = horas_por_empleado["tiempo"].max()

```

Ilustración 7. Min y Max horas

2.7. Valores mínimos y máximos del tiempo entre apertura y cierre de incidente

- Se calcula la duración de cada incidencia como la diferencia entre fecha_cierre y fecha_apertura.
- Se obtiene el mínimo y máximo de días que ha durado una incidencia.

```

df_tickets["duracion_dias"] = (df_tickets["fecha_cierre"] - df_tickets["fecha_apertura"]).dt.days
min_duracion = df_tickets["duracion_dias"].min()
max_duracion = df_tickets["duracion_dias"].max()

```

Ilustración 8. Min y Max días incidencia

2.8. Valores mínimos y máximos del número de incidencias atendidas por empleado

- Se cuentan cuántas incidencias únicas (nunique()) ha atendido cada empleado.
- Se obtiene el mínimo y máximo de incidencias atendidas.

```

tickets_por_empleado = df_contactos.groupby("id_emp")["id_ticket"].nunique().reset_index()
min_incid_emp = tickets_por_empleado["id_ticket"].min()
max_incid_emp = tickets_por_empleado["id_ticket"].max()

```

Ilustración 9. Min y Max incidencias atendidas

2.9. RESULTADOS

Para continuar con el ejercicio, pasamos al archivo analisis.py, donde se calculan los siguientes valores con sus respectivos resultados:

Cálculo	Resultado
Numero de muestras totales.	88
Media y desviación estándar del total de incidentes en los que ha habido una valoración mayor o igual a 5 por parte del cliente.	7.89
Media y desviación estándar del total del número de incidentes por cliente.	8.80

Media y desviación estándar del número de horas totales realizadas en cada incidente.	5.00
Valor mínimo y valor máximo del total de horas realizadas por los empleados.	14.00 - 45.00
Valor mínimo y valor máximo del tiempo entre apertura y cierre de incidente.	0 – 4
Valor mínimo y valor máximo del número de incidentes atendidos por cada empleado.	14 - 15

Tabla 1. Resultados cálculos ejercicio 2

3. EJERCICIO 3

En este ejercicio analizaremos los datos que tengan como tipo de incidencia “Fraude” basándonos en agrupaciones. Para cada agrupación calcularemos el número de incidencias, el número de actuaciones realizadas por los empleados y un análisis estadísticos de los resultados.

Todos los cálculos realizados se podrán calcular con el script en python analisis.py, y el código de este apartado se encontrará en el apartado agrupaciones en el mismo archivo.

Por empleado

Para el primer cálculo, primero de todo necesitamos un dataframe en el que guardar solo los datos que cumplan con el tipo de incidencia “Fraude”. Para ello, del dataframe df_tickets guardaremos solo los que tengan el valor “5” en la columna tipo_incidencia.

A continuación, del dataframe contactos guardaremos solo las entradas que estén dentro del conjunto df_fraude y así guardaremos en group_fraude esas entradas.

Con el grupo con las entradas por empleado de los casos de fraude calcularemos las incidencias y actuaciones. Para las incidencias agruparemos por id_emp las entradas y contaremos los id_tickets de manera única (Si un empleado trabaja varias veces en un id_ticket se cuenta una sola vez). Para las actuaciones agruparemos por id_emp las entradas y contaremos los id_tickets de manera repetida (Si un empleado trabaja varias veces en un id_ticket se cuentan todas las actuaciones)

Por último, calculamos las estadísticas de las incidencias por empleado e imprimimos todo por pantalla.


```

Agrupacion por Empleado:
  id_emp  Incidencias  Actuaciones
0    101           1           1
1    102           1           1
2    103           1           1
3    104           2           2
4    105           2           2
5    108           1           1
6    109           1           1
7    110           1           1
8    111           3           3
9    112           3           3
10   113           2           2
11   114           4           4
12   115           4           4

Estadísticas básicas por empleado:
Mediana: 2.0, Media: 2.0, Varianza: 1.3333333333333333, Max: 4, Min: 1

```

Ilustración 10. Resultados análisis de incidencias de “Fraude” por empleado

Como podemos observar, el número de incidencias y actuaciones es el mismo porque ningún empleado ha trabajado varias veces en el mismo incidente. Si hubiera algún empleado que ha trabajado varias veces en un incidente, en las actuaciones se vería reflejado cada una de ellas.

Por tipo de empleado

Para este caso, reutilizamos los cálculos realizados anteriormente que cuentan las incidencias y actuaciones por empleado, pero como tenemos que agruparlo por nivel de empleado realizaremos dos pasos extra.

Primero añadiremos la columna de nivel al grupo con merge. Una vez tenemos toda la información necesaria en el grupo, agruparemos las incidencias y actuaciones por nivel y las sumaremos para cada nivel 1,2 o 3.

Calcularemos de nuevo las estadísticas de las incidencias por nivel e imprimiremos por pantalla los resultados.

```

Agrupacion por nivel de empleado:
  nivel  Incidencias  Actuaciones
0      1           10           10
1      2            6            6
2      3           10           10

Estadísticas básicas por nivel de empleado:
Mediana: 2.0, Media: 8.666666666666666, Varianza: 5.3333333333333334, Max: 10, Min: 6

```

Ilustración 11. Resultados análisis de incidencias de “Fraude” por nivel empleado

Como podemos observar en la imagen y lo que hemos dicho anteriormente, un empleado no actúa varias veces en ningún incidente por lo que los resultados de incidencias y actuaciones serán los mismos

Por cliente

Para este caso, no hace falta que utilicemos el `group_fraude` que hemos utilizado anteriormente, porque para agrupar por cliente lo podemos hacer directamente desde el dataframe `df_fraude`. Es por eso por lo que contamos los incidentes agrupando por cliente.

Para calcular las actuaciones por cliente, sí que necesitamos la información de `id_ticket`, ya que a diferencia de en las agrupaciones por empleado, un incidente sí puede tener varias actuaciones si agrupamos por cliente. Por eso hacemos merge con `id_ticket` y agrupamos por la columna que queramos, ya que simplemente va a contar cuantas entradas hay por cliente, que son las actuaciones.

Calcularemos de nuevo las estadísticas de las incidencias e imprimiremos los resultados.

```
Agrupacion por cliente:
  cliente  Incidencias  Actuaciones
0        2           2           6
1        5           2           6
2        6           2           6
3        7           2           8

Estadísticas básicas por cliente:
Mediana: 2.0, Media: 2.0, Varianza: 0.0, Max: 2, Min: 2
```

Ilustración 12. Resultados análisis de incidencias de “Fraude” por cliente

Como podemos observar, por cada cliente que tiene al menos una incidencia de “Fraude”, se obtienen el número de incidencias y de actuaciones, que obviamente será mayor, ya que por cada incidencia puede haber varias actuaciones de diferentes empleados.

Por tipo de incidente

Para este apartado seguiremos la misma estructura que en la agrupación por cliente.

Para las incidencias simplemente las agruparemos por `tipo_incidencia` en el dataframe de fraude, aunque todas las incidencias serán de este tipo.

Para las actuaciones haremos merge de `id_ticket`, agruparemos, sin importar la columna debido a que solo contaremos las entradas, y contaremos el número de entradas para ver cuántas actuaciones ha habido para las incidencias de tipo “Fraude”

Calcularemos de nuevo las estadísticas e imprimiremos los resultados.

```
Agrupacion por tipo de incidencia:
  tipo_incidencia  Incidencias  Actuaciones
0                5           8           26

Estadísticas básicas por tipo de incidencia:
Mediana: 8.0, Media: 8.0, Varianza: nan, Max: 8, Min: 8
```

Ilustración 13. Resultados análisis de incidencias de “Fraude” por tipo de incidencia

Como podemos observar en los resultados, el número de incidencias de tipo “Fraude” es 8 y el número de actuaciones es de 26, que comprobando los apartados anteriores es la suma de todas las actuaciones de los clientes, empleados y tipo de empleados

Por día de la semana

Para este apartado necesitamos crear una columna que corresponda a los diferentes días de la semana. Para ello crearemos una copia del dataframe `group_fraude` y con la función `.dt.dayweek` transformamos una fecha en el día de la semana que le corresponde del 0-6 (lunes-domingo).

Para calcular los incidentes, suponemos que el incidente utiliza la `fecha_apertura`, o en nuestro caso, utilizamos la primera fecha de actuación y se le asigna ese incidente. Al resto de actuaciones posteriores de ese incidente, no se le asigna como incidente. Para ello agrupamos por `id_ticket`, y con `min()` seleccionamos la primera entrada para ese `id_ticket`.

Para las actuaciones se agruparán en el dataframe sin los cambios realizados anteriormente y se contarán todas las entradas como actuaciones.

Debido a que el lunes no hay ningún incidente, si no seleccionamos el rango que queremos mostrar del dataframe, el lunes no se muestra.

Calcularemos las estadísticas e imprimiremos por pantalla.

```
Agrupacion por día de la semana:
  dia_semana  Incidencias  Actuaciones
0          0           0.0           3
1          1           2.0           4
2          2           1.0           4
3          3           1.0           4
4          4           2.0           4
5          5           1.0           3
6          6           1.0           4

Estadísticas básicas por día de la semana:
Mediana: 1.0, Media: 1.1428571428571428, Varianza: 0.4761904761904761, Max: 2.0, Min: 0.0
```

Ilustración 14. Resultados análisis de incidencias de “Fraude” por día de la semana

Como podemos observar, hay 8 incidencias que es el total visto anteriormente y 26 actuaciones.

4. EJERCICIO 4

Para el último ejercicio vamos a generar gráficos, con `matplotlib`, de las nuevas funcionalidades del MIS.

Todos los cálculos realizados se podrán calcular con el script en `python graficos.py`, y el código de este apartado se encontrará en el apartado gráficos en el mismo archivo.

Media de tiempo (apertura-cierre) de los incidentes según mantenimiento o no mantenimiento

Primero de todo vamos a calcular el tiempo de apertura-cierre en días. Para ello restaremos cada columna de apertura y cierre por cada entrada del dataframe `df_tickets` y lo guardaremos en una nueva columna. Agruparemos los resultados por `es_mantenimiento` y haremos una media.

Para representarlo establecemos el 0 a no mantenimiento y el 1 a mantenimiento. Además, estableceremos el tamaño, los datos de las barras, los títulos y el inicio del gráfico.

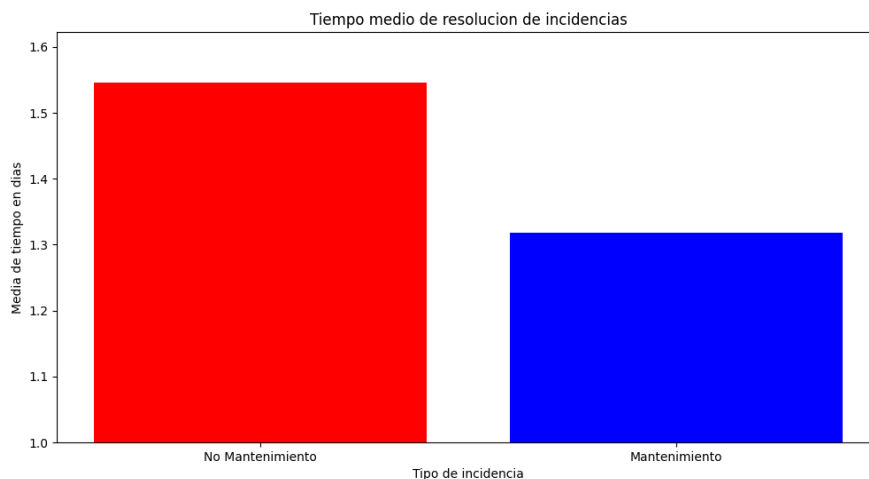


Ilustración 15. Gráfico del tiempo medio

Como podemos observar, los incidentes de no mantenimiento tienen una media de 1.55 días y los de mantenimiento de 1.3 días aproximadamente.

Boxplot de tiempos de resolución de incidentes con percentiles 5% y 90%

Para la creación del gráfico boxplot primero agruparemos por `tipo_incidencia` los tiempos de resolución y los añadiremos a una lista.

Después, para cada tipo de incidencia pondremos las entradas de los tiempos de resolución y configuraremos parámetros como la representación en vertical, el aspecto y los títulos.

Para cada incidencia calcularemos los percentiles 5% y 90%

Por último, estableceremos los títulos del gráfico

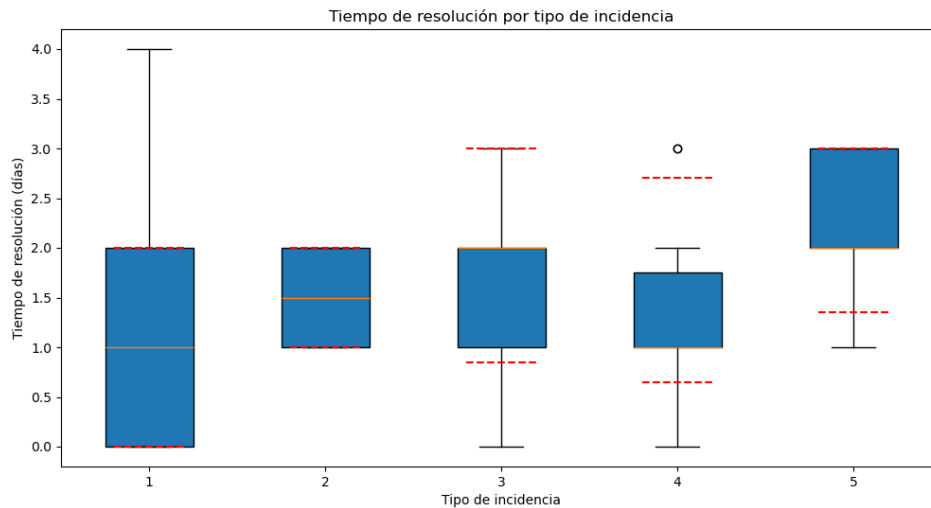


Ilustración 16. Gráfico boxplot

Como conclusiones podemos sacar que el tipo de incidencia 1 tiene una varianza muy alta, los tipos de incidencia 2,3 y 4 tienen una mediana de resolución muy similar, exceptuando un valor atípico en el tipo de incidencia 4, y el tipo de incidencia 5 tiene una mediana más alta que el resto.

Mostrar los 5 clientes más críticos

Para cumplir con los requisitos del enunciado crearemos un nuevo dataframe `df_criticos` donde guardaremos los incidentes que sean de mantenimiento y de tipo de incidente distinto de 1.

Agrupamos por cliente, contamos las entradas de cada uno y hacemos merge con el dataframe `df_clientes` de las columnas `id_cli` y `nombre` para así mostrar los nombres de los clientes en el gráfico. Por último, seleccionamos solo las 5 primeras entradas (Los más críticos).

Para representarlo seguiremos la misma estructura que el resto de los gráficos de barras.

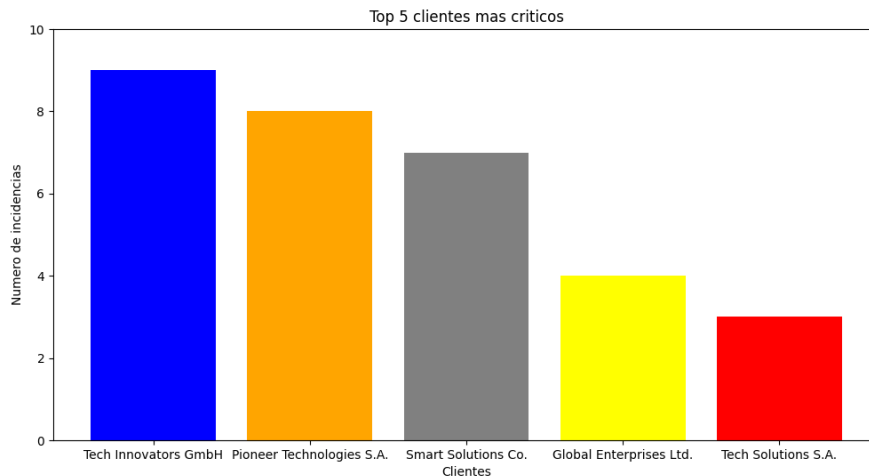


Ilustración 17. Gráfico los 5 clientes más críticos

Como podemos observar, obtenemos los clientes con más incidencias, siguiendo los parámetros vistos por el enunciado, con sus respectivos nombres y número de incidencias.

Mostrar las actuaciones realizadas por los empleados

Agruparemos los empleados por `id_emp` e igual que antes haremos un merge con el dataframe `df_empleados`, donde están los nombres de los empleados.

En nuestro caso, por espacio y facilidad para ver los gráficos hemos usado los IDs de los empleados, pero para ver el nombre de los empleados simplemente tendremos que cambiar los parámetros de `plt.bar()`.

Para la creación del gráfico de barras seguiremos la misma estructura que el resto.

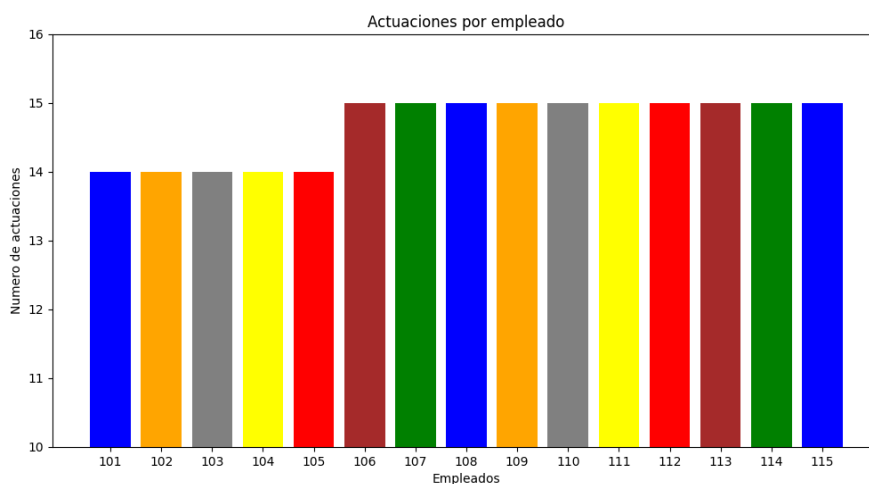


Ilustración 18. Gráfico de actuaciones por empleado

Como podemos observar, el número de actuaciones por empleado es bastante similar, habiendo dos principales grupos.

Mostrar las actuaciones realizadas según el día de la semana

Igual que en el apartado del anterior ejercicio, necesitamos pasar los datetimes a días de la semana. Una vez añadida la columna, agruparemos por día de la semana y contaremos el número de entradas para cada uno.

Para facilitar la comprensión de la gráfica mapearemos cada día de la semana con su día en texto.

Para la creación del gráfico de barras seguiremos la misma estructura que el resto.

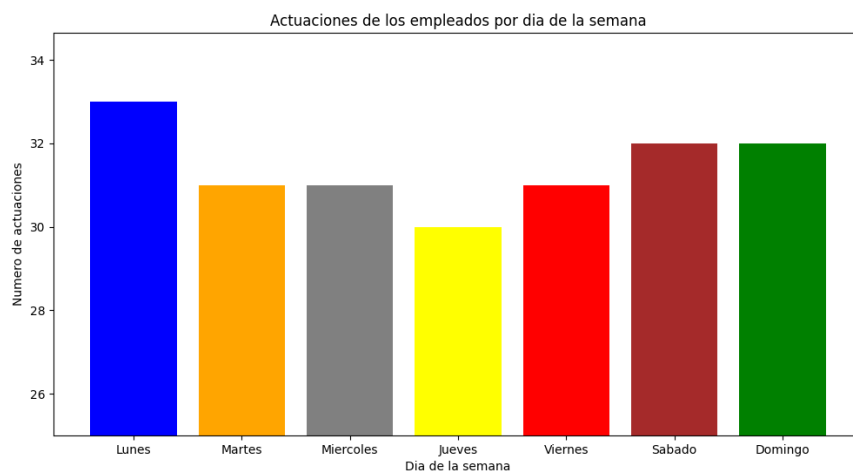


Ilustración 19. Gráfico actuaciones según el día de la semana

Como podemos observar se representan el número de incidentes según el día de la semana, viendo que el lunes, seguido del sábado y domingo, son los días que más actuaciones se realizan.

5. ENTORNO WEB

Flask es un microframework para Python que permite crear aplicaciones web de manera sencilla y flexible. A diferencia de otros frameworks más grandes, Flask proporciona solo los componentes esenciales (enrutamiento, servidor web y motor de plantillas), permitiendo al desarrollador construir la aplicación a medida sin imponer una estructura rígida.



Ilustración 20. Página inicio Flask

Estructura del Proyecto

Un entorno web construido con Flask se organiza comúnmente en tres partes principales:

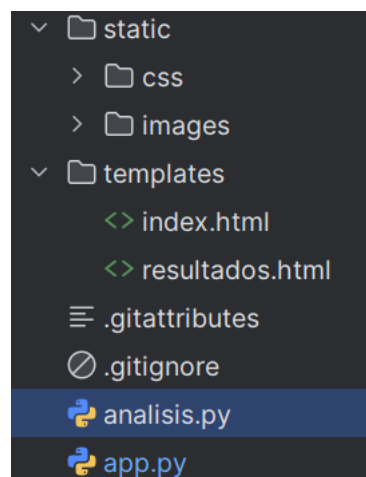


Ilustración 21. Estructura del Entorno Web

- **app.py:**
Es el punto de entrada de la aplicación. Aquí se definen las rutas, se configura Flask y se arranca el servidor. Por ejemplo, mediante la decoración `@app.route("/")` se indica la URL asociada a la función `index()`, que llama a `render_template("index.html")` para cargar la vista correspondiente.
- **templates/:**
Esta carpeta contiene las vistas HTML que se renderizan utilizando el motor de plantillas Jinja2. Las plantillas permiten incluir variables y estructuras de control para mostrar contenido dinámico. Por ejemplo, en la página de inicio se pueden incluir diagramas UML y BPMN, y en la ruta `/resultados` se muestran estadísticas y agrupaciones calculadas en otro módulo.

- **static/:**

Aquí se alojan los archivos estáticos, como hojas de estilo (CSS), scripts de JavaScript y recursos gráficos (imágenes). Estos recursos se vinculan en las plantillas mediante la función `url_for`, lo que facilita su gestión y actualización.

Definición de Rutas y Envío de Datos a las Plantillas

Dentro de **app.py** se inicia la aplicación y se implementan las rutas. Por ejemplo:

- **Ruta raíz ("/"):**

La anotación `@app.route("/")` asocia la URL principal con la función `index()`. Al llamar a `render_template("index.html")`, Flask busca el archivo `index.html` en la carpeta **templates/**, mostrando la página de inicio, en la que se pueden incluir diagramas UML y BPMN.

```
@app.route("/")  # Juan Antonio Suárez Suárez
def index():
    return render_template("index.html")
```

Ilustración 22. Route a la Página principal

- **Ruta de Resultados ("/resultados"):**

Esta ruta muestra cómo enviar datos a una plantilla mediante parámetros. Se utilizan variables obtenidas del módulo `analisis.py` para mostrar estadísticas y agrupaciones. Por ejemplo, al utilizar la sintaxis Agrupación por `{{ key }}`, se mostrará "Agrupación por empleado" si la clave `key` tiene el valor "empleado". Este enfoque permite renderizar datos dinámicos en el HTML mediante Jinja2.

```
@app.route("/resultados")
def resultados():
```

Ilustración 23. Route a la Página resultados

Mejora en la Modularidad

Una mejora potencial es la creación de un módulo adicional que se encargue de formatear las agrupaciones antes de enviarlas a la plantilla. Esto favorecería un desarrollo modular y ordenado, ya que se separaría la lógica de análisis (en `analisis.py`) de la de presentación, facilitando el mantenimiento y la escalabilidad de la aplicación.

El uso de Flask para crear un entorno web permite separar claramente las responsabilidades:

- **La lógica de negocio** (cálculos y análisis) se gestiona en módulos específicos, como `analisis.py`.
- **La lógica de presentación** se implementa en `app.py` y las plantillas HTML.
- **Los recursos estáticos** se organizan en la carpeta **static/**.

Este enfoque modular no solo mejora la organización del código, sino que también facilita la integración de nuevas funcionalidades y el mantenimiento futuro del proyecto.

BIBLIOGRAFIA

<https://www.geeksforgeeks.org/box-plot-in-python-using-matplotlib/>

https://www.w3schools.com/python/pandas/ref_df_merge.asp

<https://www.scaler.com/topics/matplotlib/boxplot-matplotlib/>