POR HACER:

1. Explicación de cómo hemos usado Scrum:
   1. Roles de cada miembro del equipo.
   2. Explicación de cómo se ha puesto en práctica Scrum, describiendo los eventos tal como los hemos puesto en práctica, incluyendo un calendario de eventos y ejemplos de problemas encontrados y como los hemos resuelto, como mínimo.
   3. Listado de los artefactos de Scrum.
2. Enlace al repositorio de código grupal à [PabloHernandezUDC/practica-is (github.com)](https://github.com/PabloHernandezUDC/practica-is)
   1. Descripción de cómo hemos gestionado los repositorios (git) individuales y grupal.
   2. Flujo y política de creación de ramas y gestión de los merges para integrar el código de cada miembro.
   3. Política de nombrado de commits
3. Explicación del código desarrollado, qué hace y cómo lo hace (funcionalidades):
   1. Patrones, aspectos de diseño relevantes (\*)
   2. Pruebas y planes de pruebas creados.
   3. Guías o normas de estilo utilizadas.
   4. Descripción del proceso de despliegue (qué tendría que hacer el cliente para poder ejecutar la aplicación desarrollada).
   5. Manual de usuario à no es necesario porque ya lo vamos a poner en el README del repositorio de código grupal

***¿Cómo hemos usado Scrum?***

*Roles*

En cuanto a los roles, todos fuimos Desarrolladores, y además Marcos fue el Scrum Master. Los Desarrolladores se ocuparon de decidir de puntuar y elegir las tareas adecuadas antes de cada Sprint, además de realizarlas durante el transcurso del mismo.

*Aplicación de Scrum y resolución de problemas*

La aplicación del framework Scrum a la práctica fue bastante simple. Para ello, se emplearon Sprints semanales donde se realizaban una fracción de todas las historias del Product Backlog. Cada viernes lectivo se llevaba a cabo una reunión en la que se evaluaban los resultados del Sprint anterior, y se reflexionaba sobre el trabajo durante el mismo. A continuación, se puntuaban y asignaban las historias del próximo Sprint, y durante esa semana, los desarrolladores las completaban.

Siempre que el equipo se topó con algún tipo de duda, problema o conflicto, lo intentó resolver consultando a las personas adecuadas. Por ejemplo, si alguien tenía una pregunta sobre el funcionamiento de una parte del código que había elaborado otra persona, le preguntaba a ella. Por otro lado, hubo una ocasión en la que tuvimos que consultar a nuestro profesor sobre problemas técnicos con GitHub.

En cuanto a los artefactos de Scrum, el Product Backlog fue elaborado por el Product Owner, que también proporcionó el Objetivo del Producto. Esto fue clave a la hora de entender las historias y tareas. El Sprint Backlog fue elaborado por los desarrolladores a escala semanal y permitió al equipo avanzar con el proyecto a un ritmo más realista y consistente gracias al Objetivo del Sprint. Por último, los incrementos fueron cada uno de los avances concretos que acercaban el proyecto al Objetivo del Producto, y que fueron testeados individualmente para asegurar su funcionamiento.

***Repositorio (***[***enlace***](https://github.com/PabloHernandezUDC/practica-is)***)***

*Uso*

Durante la realización del proyecto se utilizó un repositorio de GitHub para sincronizar el código entre desarrolladores y archivar los cambios de manera eficiente. Cada desarrollador, a su vez, usaba su repositorio local para mantener los cambios durante el proceso de edición.

*Ramas*

La política de creación de ramas fue más bien sencilla. Además de *main*, cada desarrollador creó una rama a su nombre, y él/ella debía asegurarse de trabajar única y exclusivamente en su rama hasta que el código estuviese 100% preparado para integrarse en la rama principal *main*. Cada uno debía asegurarse de trabajar sobre código que estuviese al día. Resolver los conflictos que se le pudiesen presentar al fusionar (hacer “merge”) sus avances con *main*.

(\*)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Creacionales | Estructurales | De comportamiento |
| *classModel.py* | No presenta patrones creacionales. | Facade no se implementa explícitamente, pero la clase proporciona métodos simples para acceder y modificar atributos, lo que se asemeja al objetivo del patrón. | No presenta patrones de comportamiento. |
| *dataOp.py* | No presenta patrones creacionales. | Las librerías pandas y tkinter proporcionan funcionalidades que coinciden con el patrón Facade. | En el bucle for dentro de la función getColumns() se recorren las columnas de un DataFrame, lo que refleja ciertos aspectos del patrón Iterator. En la función createColumns() hay una observación de eventos en la interfaz gráfica que desencadena una acción al hacer click en un botón, lo que se asemeja al patrón Observer. |
| *guiOp.py* | Factory Method no está directamente presente, pero la función readFile() y loadModelFromPickleObject() de los módulos dataOp y modelOp respectivamente podrían ser considerados métodos de fábrica si internamente crean y devuelven objetos de manera específica según ciertos parámetros. | Existe un uso potencial del patrón Facade en el módulo customTkinter. La creación de widgets personalizados como CTkButton proporciona una interfaz simplificada para trabajar con los elementos de la interfaz gráfica. | No presenta patrones de comportamiento. |
| *main.py* | No presenta patrones creacionales. | No presenta patrones estructurales. | No presenta patrones de comportamiento. |
| *modelOp.py* | La función makeModel() podría considerarse similar al patrón Factory Method. | La librería customTkinter se asemeja al patrón Facade al proporcionar una interfaz simple para un subsistema más grande y complejo (tkinter en este caso). La función loadModelFromPickledObject() podría considerarse como una forma del patrón Proxy ya que actúa como un intermediario entre la solicitud de cargar el modelo y la presentación de la información correspondiente. | La función makeAndShowGraph() podría interpretarse como un patrón Observer, donde el gráfico es el observador que se actualiza cuando el modelo cambia. |
| *prediction.py* | Factory Method no se utiliza explícitamente, pero la función makePrediction() muestra similitudes. | Se usa el patrón Facade con la función makePrediction() para encapsular la complejidad de la interfaz de usuario para realizar una predicción. | No presenta patrones de comportamiento. |
| *readDbOp.py* | Factory Method no se muestra explícitamente pero la función readSQL() se asemeja. | La función readSQL() puede considerarse como una implementación del patrón Facade. | Las funciones readRows() y readOrdered() muestran un comportamiento similar al patrón Iterator. Además, la estructura de las funciones sigue un patrón con un método común, lo cual tiene similitudes con el patrón Template Method. |
| *regression.py* | No presenta patrones creacionales. | No presenta patrones estructurales. | No presenta patrones de comportamiento. |

Patrón iterator à bucles for sin el range (mencionar alguno de nuestro código)