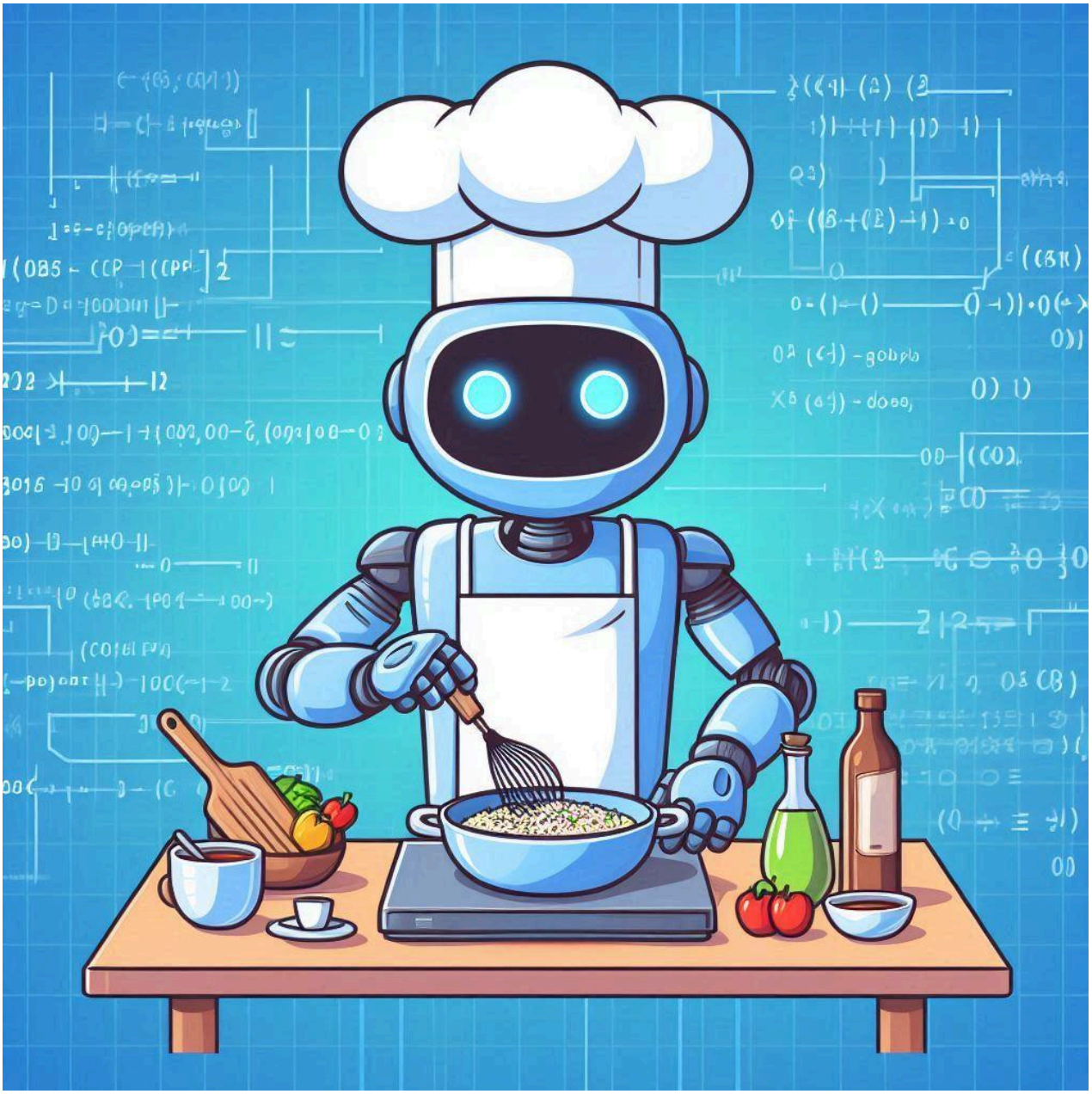
# SISTEMA DE RECOMENDACIÓN DE RECETAS BASADO EN SURPRISE



David Fernández Prieto 44663898Z davidvispo4211@gmail.com
Daniel Barandela Cachafeiro 45163395C danielbaracacha@gmail.com

# Índice

# Introducción al problema

## Objetivos

-Desarrollar un sistema de recomendación en base a reseñas de usuarios.
-Manejar la incertidumbre en las preferencias de los usuarios, realizando predicciones.
-Establecer una buena interacción con el usuario, con recomendaciones relevantes y sencillas.

## Explicación del caso.

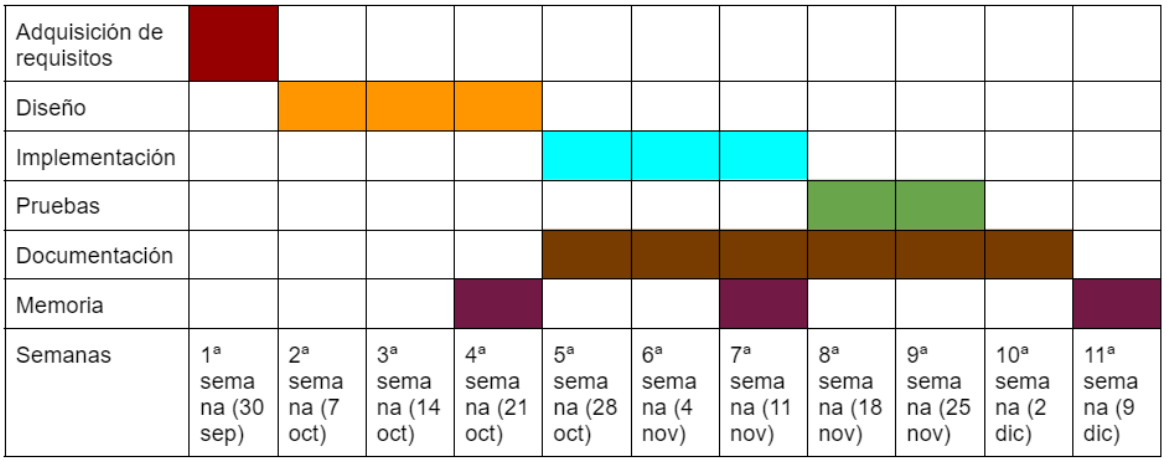
A pesar de que originalmente se tratase de recomendación de restaurantes, ya que no encontramos ningún dataset adecuado al tema, decidimos orientarlo a recomendación de recetas.

El modelo, a partir de todas las reseñas que han dejado los usuarios y, en caso de que ya haya sido entrenado sobre ellos, los gustos del propio usuario que solicita la recomendación (al que nos referiremos en adelante como cliente), predeciría la nota que el cliente daría a las recetas que no ha probado, generando unas recomendaciones de platos que no ha probado personalizadas para este.

El dataset que usaremos para desarrollar y entrenar el sistema está conformado por la información sobre las reseñas de recetas (subidas por los propios usuarios) e información de las mismas subidas a la página web Food.com.

Para la implementación del modelo se utilizará la librería de python **surprise.**

# Planificación




Adquisición de requisitos:

Comprobamos que la librería surprise para python se ajustase a nuestras necesidades para el sistema, leyendo su documentación y funciones básicas. Buscamos una base de datos adecuada para el modelo pensado.

Diseño:

Preparar la arquitectura de todo el código, la documentación y la memoria. Se realizarán esquemas para explicar dichas estructuras y cómo se relacionan las partes del código entre sí. Además, quedarán diseñadas las pruebas que deberá pasar el modelo.

Implementación:

Programar el código como quedará pensado en la fase de diseño, además de solucionar los problemas que aparezcan durante el proceso, esta fase se repetirá las veces que sean necesarias dependiendo del resultado de las pruebas.

Pruebas:

Realizar las pruebas necesarias sobre el código para asegurar el funcionamiento satisfactorio, se volverá a la fase de diseño o de implementación según las pruebas que no logre superar el modelo.

Documentación:

Se documentará el código realizado así como las pruebas a medida que se avance en dichas fases, la 10ª semana se empleará para realizar el manual de usuario y limpiar la documentación de código.

Memoria:

La memoria se actualizará intermitentemente durante el proyecto hasta llegar a la semana final en la que se dejará completamente preparada para su presentación, una vez finalizadas las demás partes del proyecto.

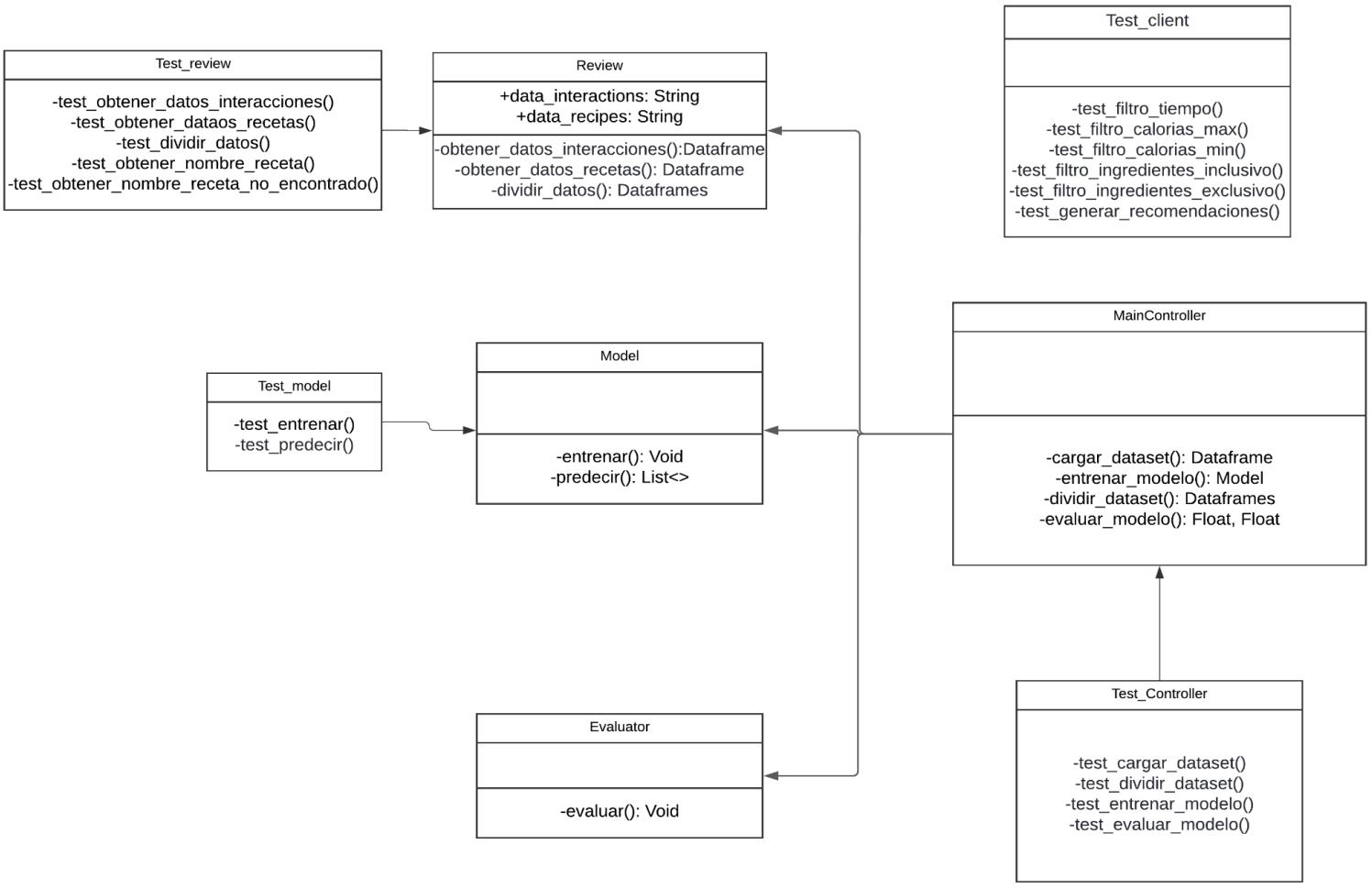
# Diseño

Para el diseño del programa se hicieron diagramas de clases para representar la estructura que sigue la implementación y diagramas de flujo para representar los procesos y su orden durante la ejecución tanto del entrenamiento como de la recomendación.

En esta parte tan solo se explicará el funcionamiento general de los módulos del programa así como la forma en la que se relacionan, la implementación será desarrollada en su propio apartado.

Diagramas de clases:

<u>Fase de entrenamiento</u>



El entrenamiento se lleva a cabo desde el módulo **maincontroller,** que controla todos los procesos que se realizan en esta fase.

El controlador carga los datos que se usarán para entrenar el modelo **(cargar_dataset)**, llamando a las funciones de review necesarias **(obtener_datos_interacciones, obtener_datos_recetas)**. Estos datos son divididos en entrenamiento, validación y test **(dividir_dataset)**.

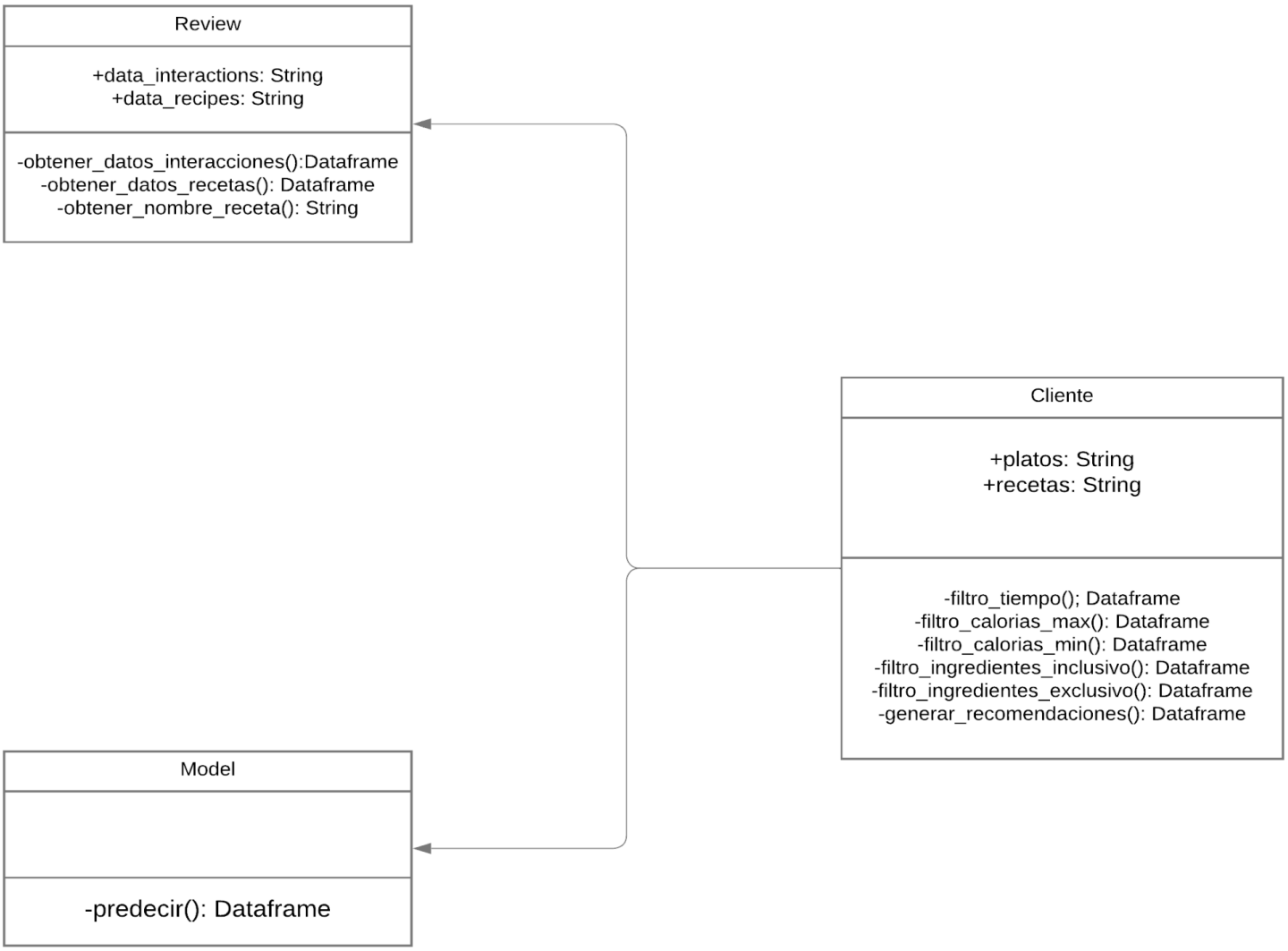
Una vez los datos están listos para ser utilizados por el modelo, el **maincontroller** llama a este para que sea entrenado **(entrenar_modelo).**

Por último, el **maincontroller** evalúa el modelo **(evaluar_modelo)**, retornando el RMSE (raíz del error cuadrático medio) en las predicciones.

En este diagrama se muestran los tests, ya que se realizan tan solo en la fase de entrenamiento, incluso sobre los módulos que no participan en dicho proceso, como en el caso del **Client**.

Los módulos de test tan sólo interaccionan con el módulo que deben probar.

Fase de recomendación

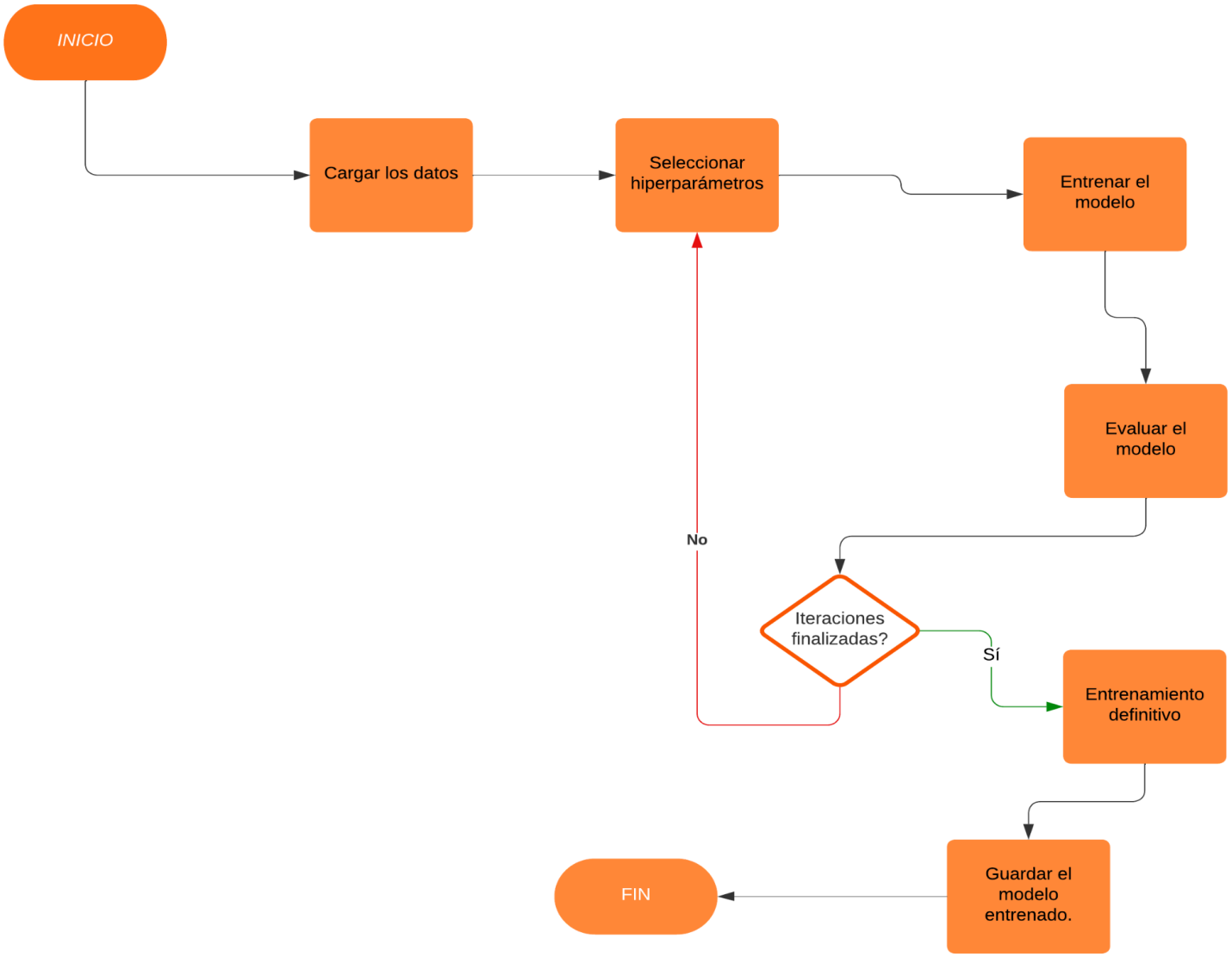



La recomendación se lleva a cabo desde el módulo **Cliente.** Éste obtiene las peticiones del cliente **(filtro_tiempo, filtro_calorias_max, filtro_calorias_min, filtro_ingredientes_inclusivo, filtros_ingredientes_exclusivo)** y los datos sobre los que predecir (al crear la clase Client el constructor llama a **obtener_datos_interacciones** y **obtener_datos_recetas**).

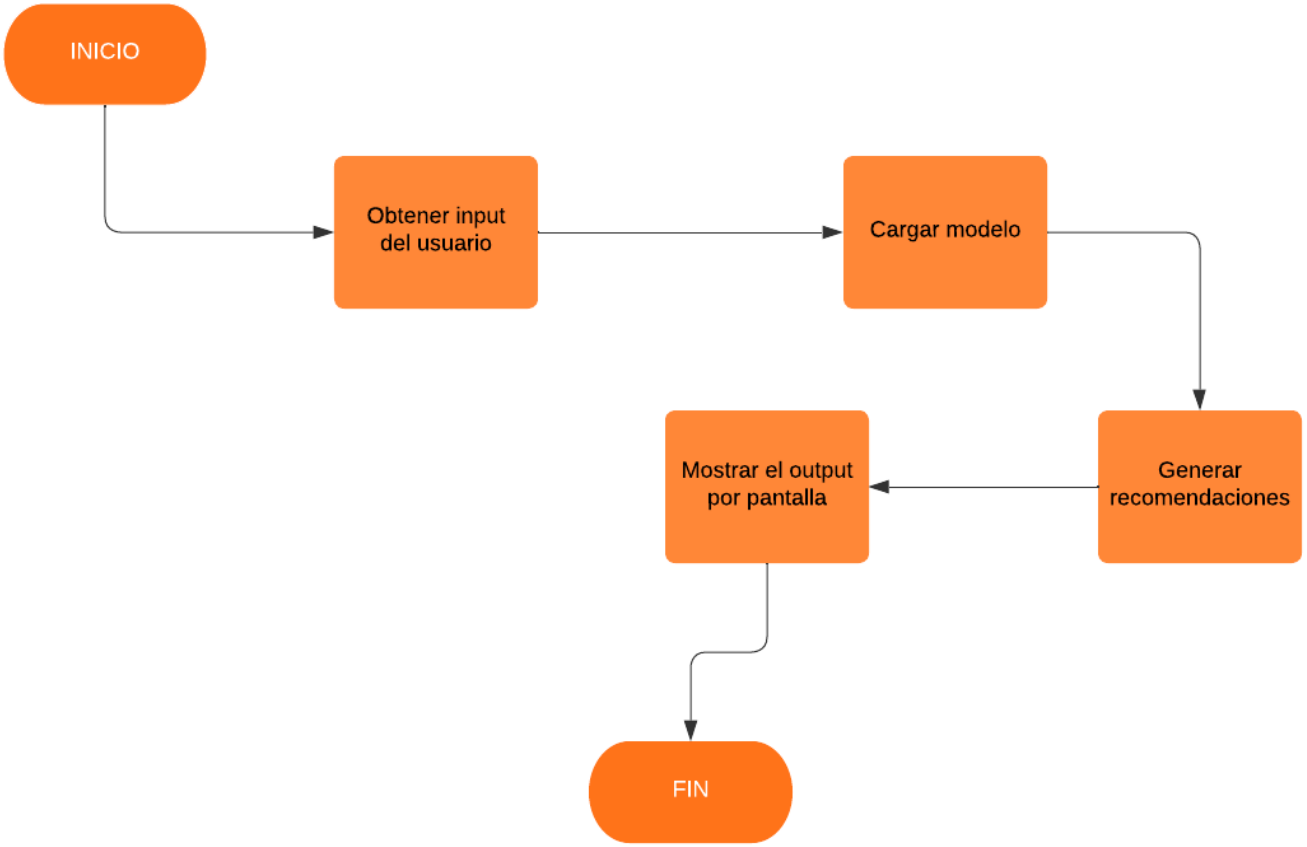
Finalmente, genera las predicciones para el usuario con el ID introducido (función **predecir** de la clase **Model**) que cumplan los filtros posteriores y las muestra por pantalla ordenadas según su calificación, junto con un link a la receta original.

Diagramas de flujo:

## Fase de entrenamiento



Fase de recomendación



# Implementación

## Tecnologías usadas

### Surprise

Para la implementación del modelo se utilizó la librería surprise, que implementa muchas funciones y clases útiles para el caso, concretamente:

·Relacionadas con el entrenamiento del modelo

-Reader: la clase Reader interpreta el dataframe que usará el modelo. Permite establecer el rango de valores entre los que estarán las valoraciones.

-Dataset: maneja el dataframe de entrada, convirtiéndolo a un formato entendible para surprise. Incorpora varias fuciones como build_full_trainset o construct_testset.

-SVD: es el modelo de recomendación que decidimos usar (ya que tras varias pruebas nos resultaba en el menor error). Se basa en la descomposición matricial e incluye las funciones fit y test.

-RandomizedSearch: hace una búsqueda exhaustiva con distintos hiperparámetros, permitiendo usar la mejor combinación para el entrenamiento final y devolver el mejor resultado. A diferencia de GridSearch, que prueba con todas las posibles combinaciones (por lo tanto, es óptimo, pero tarda mucho tiempo), éste prueba un número determinado de combinaciones aleatorias.

·Relacionadas con la evaluación

-accuracy: esta función devuelve el error medio en las predicciones que se le pasen. Permite cuantificar el error con distintas medidas, como el MAE, RMSE o FCP.

### Otras

Para la implementación de las pruebas se utilizó la librería unittest, que aporta métodos para realizar pruebas unitarias sobre funciones.

También hicimos uso de pandas, que permite trabajar sencillamente con dataframes.

# Preprocesado de los datos

Antes de empezar con toda la elaboración del proyecto, tenemos que prestar atención a la calidad de los datos que usaremos porque determinará en gran parte si obtenemos un buen sistema de recomendación.

Para limpiar los datos y que la codificación fuera lo más sencilla posible, filtramos ambas partes del dataset, en el módulo **preprocesado.py**, en la subcarpeta de datasets.

Primero, descartamos las columnas innecesarias de las reviews y de las recetas.

Luego, eliminamos las reviews de todo usuario con menos de un número determinado de reviews, ya que a la hora de entrenar un modelo estos datos pueden ser considerados ruido y aumentar el overfitting. Tras varias pruebas midiendo el error de salida, decidimos dejar el número mínimo en 8, ya que no reducía demasiado el tamaño del dataset (de 960 mil a 850 mil) y el error bajaba considerablemente (de 0.89-0.9 a 0.74-0.75).

Por último, filtramos el dataset de recetas eliminando todas aquellas que no tengan ninguna review dedicada, ya que serían datos que no usaríamos y también el dataset de interacciones, borrando aquellas reseñas que hagan referencia a recetas de las que no tenemos información.

Pensamos en que se podría juntar ambos datasets en uno solo, para hacerlo más manejable a la hora de codificar, pero no lo hicimos ya que al hacer un merge entre ambos datasets tendríamos muchísimos datos redundantes y el modelo sería mucho más lento (para cada review tendríamos los datos de su receta, por lo que cada misma receta aparecería un promedio de 4 a 5 veces).

# Codificación

## Client:

**__init__:** las instancias de la clase contienen los datos sobre los que se harán las predicciones **(self.platos,** que se obtienen instanciando el método **obtener_datos_interacciones** de la clase **Review**) y el resto de información para los filtros **(self.recetas,** que se obtienen instanciando el método **obstener_datos_recetas** de la clase **Review).**

**generar_recomendaciones:** recibe el id del usuario, para comprobar si el modelo ya conoce sus gustos (si ha sido entrenado sobre ellos) y el número de recetas que quiere. Consulta los filtros que quiere aplicar (calorías, ingredientes y tiempo de preparación). Después de que el modelo genere las predicciones sobre las recetas, se eliminan las que no cumplan los requisitos solicitados.
Sólo se mostrarán aquellas que cumplan los requisitos incluso si son menos de las que pidió el cliente. En caso de que no haya ninguna receta que cumpla con los filtros pedidos, se devuelve un None al main para que muestre un mensaje de aviso.
Todos los requisitos son opcionales y están preparados para que no salten errores con entradas incorrectas.

**filtro_tiempo, filtro_calorias_max y filtro_calorias_min:** las tres funcionan de la misma forma, solicitan un número al usuario que no tiene por que ser un entero. En el caso de no querer establecer estos filtros basta con introducir un 0.

**filtro_ingredientes_inclusivo y filtro_ingredientes_exclusivo:** ambas funcionan de manera similar, reciben los ingredientes separados por ‘,’ y sin espacios que se quieren incluir/excluir respectivamente. En el caso de no querer incluir este filtro basta con introducir una cadena vacía.

**main:** en el main se carga el modelo ya entrenado, se instancian las clases **Cliente** y **Review,** se obtienen los inputs necesarios para la función **generar_recomendaciones** y se llama a dicha clase. Una vez se tienen las recomendaciones finales, se muestran por consola junto con un link a su respectiva página web donde se puede visualizar la receta al completo (con fotos, ingredientes y pasos a seguir). En el caso de no haber recetas que cumplan con los filtros establecidos por el usuario, se mostrará un mensaje.

## Controller:

**cargar_dataset:** carga el dataset que se utilizará para entrenar el modelo mediante el método **obtener_datos_interacciones** de la clase **Review.**

**dividir_dataset:** divide los datos en entrenamiento, validación y test para llevar a cabo todo el proceso de entrenamiento y optimización del modelo.

**entrenar_modelo:** ejecuta el entrenamiento del modelo llamando a la función **entrenar** de la clase **Model.**

**evaluar_modelo:** lleva a cabo la evaluación del modelo resultante y muestra los resultados por pantalla y para esto llama a la función **evaluar** de la clase **Evaluator**.

**main:** se ejecutan las funciones necesarias para el entrenamiento (**cargar_dataset, dividir_dataset, entrenar_modelo y evaluar_modelo)**, se avisa de la finalización del proceso y por último se guarda el modelo entrenado.

## Model:

**__init__:** en la instancia de esta clase se guardará el modelo de predicción.

**entrenar:** en esta función se separan los atributos del dataset que se usarán para el entrenamiento (user_id, recipe_id y corrected_rating). Antes de entrenar el modelo definitivamente, usamos el método **RandomizedSearchCV** (ya implementado por la librería surprise) para buscar la mejor combinación de los hiperparámetros: n_factors, n_epochs, lr_all y rg_all. Finalmente se entrena el modelo.

**predecir:** esta función genera unas predicciones sobre los datos de test, que serán usadas por el **maincontroller** para calcular el error del modelo y evaluarlo.

## Review:

**__init__:** se almacenan los paths a los datasets que se usarán tanto en recomendación como en entrenamiento.

**obtener_datos_interacciones:** se cargan los datos sobre las reviews en un dataframe.

**obtener_datos_recetas:** se cargan los datos sobre las recetas en un dataframe.

**dividir_datos:** divide los datos que se le pasarán al modelo para entrenar en entrenamiento, validación y test.

**obtener_nombre_receta:** una función para obtener los nombres de las recetas a partir del id.
**obtener_tiempo_receta:** una función para obtener el tiempo de preparación de las recetas a partir del id.

## Evaluator:

**evaluar:** función para medir el error en una predicción. Nosotros usaremos el RMSE.

# Tests

## TestClient:

**setup:** creamos los dataframes con datos aleatorios para las pruebas.

**test_filtro_tiempo, test_filtro_calorias_max, test_filtro_calorias_min, test_filtro_ingredientes_inclusivo, test_filtro_ingredientes_exclusivo:** todos estos tests comprueban que se almacenan correctamente las entradas en sus funciones respectivas.

**test_generar_recomendaciones:** se hacen llamadas de prueba a las funciones que se utilizan el método **generar_recomendaciones** para verificar que funcionan correctamente y se comprueba que las recomendaciones se generan y almacenan sin problemas.

## TestMainController:

**test_cargar_dataset:** comprueba que se llama a la función obtener_datos_interacciones.

**test_dividir_dataset:** comprueba que si envía un conjunto de datos a dividir, se devuelven tres conjuntos distintos.

**test_entrenar_modelo:** mira si se ha llamado alguna vez a los datos de entrenamiento y comprueba si se crea un modelo correctamente.

**test_evaluar_modelo:** simula la evaluación de un conjunto de test y validación artificiales y verifica que se llama a la función ‘evaluar’ para ambos conjuntos. Para poder hacer estas comprobaciones, se deben transformar temporalmente los dataframes en diccionarios, ya que unittest no admite comparar dataframes a menos que sean exactamente el mismo objeto (en las anteriores funciones sí podíamos porque solo se llamaba a una función externa, pero en este caso se modifica internamente la referencia y ya no son el mismo objeto).

## TestModel:

**test_entrenar:** pone a prueba que se cree correctamente el modelo.

**test_predecir:** comprueba que si predices sobre 5 instancias de reviews, devuelva 5 predicciones.

# Manual de usuario

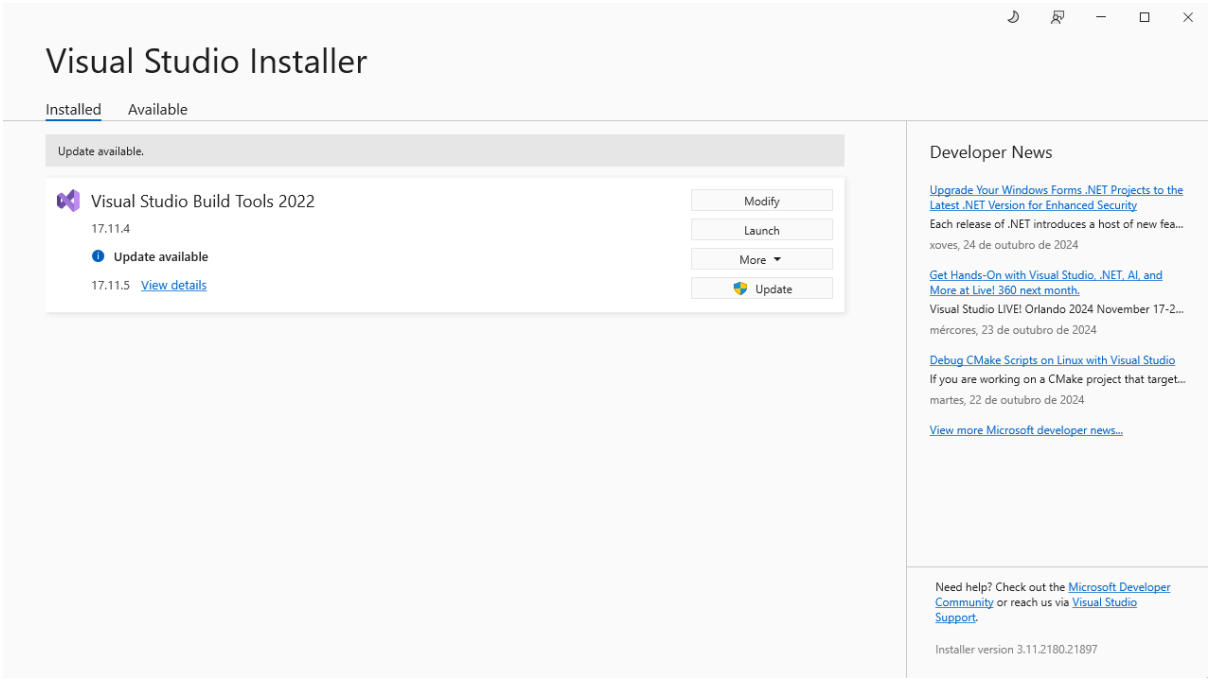
## Requisitos del programa

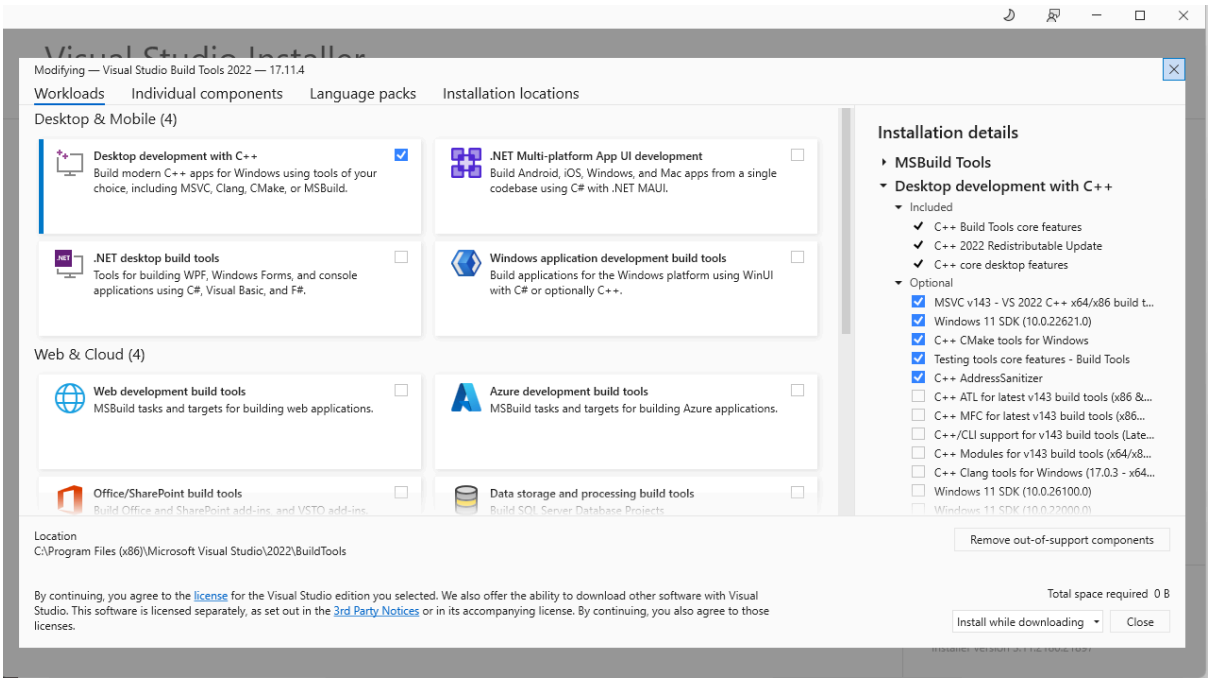
El programa necesita Python y las librerías surprise, pandas y unittest para funcionar.

Se puede descargar la última versión de python en: https://www.python.org/downloads/

Para poder utilizar surprise en windows:

https://visualstudio.microsoft.com/es/downloads/




Surprise requiere tener instalado el Desktop Development with C++ (con las descargas opcionales que se muestran a la derecha), que se puede instalar desde la aplicación gratuita de Visual Studio Installer.

Comando para instalar surprise: **$ pip install scikit-surprise**

En ubuntu se debe instalar surprise y cambiar la versión de numpy a una anterior a la 2.0 (al instalar surprise se instala una versión de numpy incompatible).

Comando para instalar pandas: **$  pip install pandas**

Unittest ya está incluido en los módulos estándar de python.

# Guia de ejecución.

Una vez se ha descargado y descomprimido la carpeta del programa se debe ejecutar el archivo **maincontroller.py** para entrenar el modelo, que al ejecutarse debe generar un archivo llamado **modelo_entrenado.**

Una vez esté el modelo entrenado, debe ejecutarse el archivo **client.py.** El programa preguntará primero cuantas recetas desea el cliente, le solicitará que aporte su ID de usuario y después se le consultará sobre los filtros que quiere aplicar: el tiempo de preparación (en minutos), las calorías máximas y mínimas, los ingredientes que desea incluir y los que desea excluir.

Una vez aportada toda la información solicitada por el programa, se mostrarán por consola las recetas que obtuvieron mejor valoración en las predicciones y un link que llevará a la página web donde se explica el proceso de elaboración de la receta y otras especificaciones.

# Problemas encontrados

## Problemas en diseño

Al principio no separamos los procesos de entrenamiento y recomendación, cuando tienen fases completamente diferentes. Tuvimos que agregar/eliminar unas cuantas funciones a las clases respecto a cómo estaba pensado originalmente, pero sin desviarnos del esquema inicial.

## Problemas en implementación

Ciertos procesos de la codificación fueron algo difíciles, ya que algunas funciones de surprise nos devolvían errores extraños, pero fueron todos arreglados. Probablemente, el principal problema en esta fase fuese conseguir descargar y usar la propia librería.

Tuvimos también varias complicaciones realizando el apartado de tests, ya que no funciona correctamente con dataframes.

# Conclusiones

Tras desarrollar este sistema de recomendación, podríamos concluir que estos son una muy buena forma de tratar la incertidumbre en sistemas basados en reviews de usuarios. Surprise es una buena herramienta para realizar el entrenamiento de esta clase de modelos, ya que aporta muchas funciones útiles para ello y además está muy optimizada, reduciendo los tiempos de entrenamiento y predicción al mínimo.

Durante todo el proceso de preprocesado y codificación también pudimos comprobar que el resultado obtenido depende enormemente de la calidad de los datos usados para el entrenamiento y de los hiperparámetros que se utilicen.

Pensamos que este tipo de sistemas, unos más avanzados que este, podrían tener aplicaciones reales para distintas empresas que busquen la satisfacción de sus usuarios con recomendaciones personalizadas de calidad.

Finalmente, pensamos que este sistema recomendador podría ser ampliado con distintos enfoques, como: añadiendo nuevos filtros que den más juego, como tags/etiquetas o, si quisiéramos una aplicación verdaderamente práctica, la posibilidad de añadir feedback del propio usuario, actualizando la base de datos periódicamente con la opción de añadir nuevas reseñas a las recetas.

# Bibliografía

-Documentación de surprise: https://surprise.readthedocs.io/en/stable/

-Base de Datos del proyecto:
https://www.kaggle.com/datasets/iamnotwhale/food-com-recsys-dataset?resource=download

-Fuente para resolver algunos conflictos durante la codificación:
https://stackoverflow.com