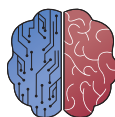




UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería de la Salud



INGENIERÍA
DE LA SALUD

**TFG del Grado en Ingeniería de la
Salud**

**Automatización de extracción
de datos de informes de
secuenciación masiva del
software *Oncomine Reporter*
Documentación Técnica**

Presentado por Lucía Vítores López
en Universidad de Burgos

3 de julio de 2023

Tutor: Antonio Jesús Canepa Oneto

Índice general

Índice general	i
Índice de figuras	iii
Índice de tablas	iv
Apéndice A Plan de Proyecto Software	1
A.1. Introducción.	1
Apéndice B Documentación de usuario	7
B.1. Requisitos software y hardware para ejecutar el proyecto. . .	7
B.2. Instalación / Puesta en marcha	7
B.3. Manuales y/o Demostraciones prácticas	21
Apéndice C Manual del programador	23
C.1. Introducción	23
C.2. Estructura de directorios	23
C.3. Compilación, instalación y ejecución del proyecto.	24
C.4. Pruebas del sistema	28
C.5. Instrucciones para la modificación o mejora del proyecto. . .	29
Apéndice D Descripción de adquisición y tratamiento de datos	31
D.1. Descripción formal de los datos.	33
D.2. Descripción clínica de los datos.	34
Apéndice E Manual de especificación de diseño	37
Apéndice F Especificación de Requisitos	39

F.1. Introducción.	39
F.2. Diagrama de casos de uso.	39
F.3. Explicación casos de uso.	46
Apéndice G Estudio experimental	53
Bibliografía	55

Índice de figuras

A.1. Acuerdo de confidencialidad HUBU-UBU para la realización del proyecto.	5
B.1. Distribución que se debe seguir a la hora de crear las carpetas para almacenar los datos y resultados obtenidos.	11
B.2. Distribución que se debe seguir teniendo en cuenta el archivo .exe.	22
E.1. Diagrama de clases, demostración del funcionamiento del código mediante un diagrama de clases de elaboración propia.	38
F.1. Requisito 1, verificación lectura correcta.	40
F.2. Requisito 2, generación variables.	41
F.3. Requisito 3, importación de ficheros que permitan modificaciones.	42
F.4. Requisito 4, formación DataFrames.	43
F.5. Requisito 5, exportación de los resultados.	44
F.6. Requisito 6, correcto funcionamiento del código.	45

Índice de tablas

A.1. Costes del personal.	3
A.2. Costes del hardware/software.	3
A.3. Costes totales del proyecto.	3
F.1. CU-1: Verificación lectura ficheros.	47
F.2. CU-2: Generación de variables.	48
F.3. CU-3: Importación de ficheros que permita modificaciones futuras.	49
F.4. CU-4: Creación de DataFrames.	50
F.5. CU-5: Creación de DataFrames.	51
F.6. CU-6: Exportación de los resultados.	52

Apéndice A

Plan de Proyecto Software

A.1. Introducción.

El proyecto se ha dividido en varias etapas, con el fin de que al unir las, se obtenga un buen resultado final.

Planificación temporal.

El proyecto se ha organizado en distintos *Milestones*, cada uno de ellos enfocado en una parte del proyecto.

- Primer *Milestone* o **Inicio del proyecto**: trabaja con información algo más general como la organización de la información a tratar en las tablas, los documentos necesarios para presentar al Comité de Bioética, el estudio de los datos necesarios para realizar el proyecto, etc.
- Segundo *Milestone* o **Desarrollo**: se tratan aspectos algo más específicos y se desarrolla el código necesario para el tratamiento de los datos.
- Tercer *Milestone* o **Proceso de entrega**: se parte de que una vez finalizado el código, cuál es la mejor manera de entregarlo al comité. Así como la finalización del desarrollo de la memoria y la mejora de los distintos ficheros a entregar, entre otros.

Dentro de cada uno, se pueden encontrar varios *Issues* (uno por cada reunión con el tutor). En cada uno se explica el contenido general de cada reunión y las metas a conseguir antes de la siguiente. Generalmente, las

reuniones son cada una o dos semanas, en función de la disponibilidad y la cantidad de trabajo. Hay *Issues* comenzados por el tutor con posibles mejoras a tener en cuenta para conseguir un mejor resultado.

También se realizan reuniones con el personal del HUBU encargado de dicho proyecto, con el fin de mejorar o responder preguntas fundamentales para el desarrollo y el entendimiento de los principales objetivos que buscan. Sin embargo, estas no cuentan como *Issues*, pero parte del contenido es explicado en *Issues* de otras reuniones.

No solo se han realizado reuniones. Todas las dudas que iban surgiendo a lo largo del proyecto eran preguntadas tanto al tutor como a la encargada del HUBU vía correo electrónico. De esta manera no era necesario tener que esperar a la siguiente reunión y se podía ir avanzando una vez resuelta la duda.

Se ha seguido este método de organización debido a que al tener varios *Issues* pertenecientes a un mismo área, agruparlos en un *Milestone* nos ayudaba a mantener la coherencia y organización. Además, es mucho más fácil de seguir el progreso que se iba llevando dentro de cada uno, ya que las tareas dentro de un *Milestone* están todas relacionadas entre sí y es necesario ir finalizando las primeras para poder seguir con el proyecto. El objetivo era ir cerrando *Issues* para llegar a finalizar los distintos *Milestones* y mediante esta técnica, ir acabando el proyecto.

Planificación económica.

Los costes se desglosarán en las siguientes categorías: costes del personal y costes de hardware/software de las herramientas.

Los costes del personal se puede ver en la tabla [A.1](#) Es una estimación de las horas que ha llevado el proyecto, multiplicado por el precio de la hora.

Para el desarrollo del proyecto no se ha adquirido ningún hardware nuevo, por lo que tan solo se incluirán en este apartado los costes del material con el que ya se contaba (asumiendo una amortización de aproximadamente 4 años) y calculando el coste de amortización correspondiente a la duración del proyecto (7 meses). No ha habido costes de software porque las herramientas usadas eran de código abierto. Se puede ver en la tabla [A.2](#).

El gasto total del proyecto se puede ver en la tabla [A.3](#).

CONCEPTO	COSTE
Horas	410 horas
Coste	7 euros
TOTAL 7 meses	2.870 euros

Tabla A.1: Costes del personal.

CONCEPTO	COSTE(€)	AMORTIZACIÓN(€)
Ordenador portátil	600	87.5
Licencia Office anual	100	14.58
TOTAL	102.08	

Tabla A.2: Costes del hardware/software.

CONCEPTO	COSTE(€)
Costes del persona	2.870
Costes del hardware/software	102.08
TOTAL	2.972,08

Tabla A.3: Costes totales del proyecto.


Viabilidad legal.

El 14 de marzo de 2023 se llegó a un acuerdo de colaboración temporal para la gestión de datos entre la investigadora Patricia Saiz López con vinculación laboral al Hospital Universitario de Burgos (HUBU) y los solicitantes Antonio Jesús Canepa Oneto y Lucía Vítores López, como vinculantes a la Universidad de Burgos (UBU).

Dicho acuerdo se enmarca dentro de la solicitud de ampliación de gestión de datos del proyecto **Secuenciación múltiple de última generación en tumores sólidos: utilidad clínica en un Hospital de tercer nivel** aprobado por el Comité de Ética de la Investigación con medicamentos de Áreas de Salud de Burgos y Soria, emitiendo favorable dicho dictamen. **A.1** La colaboración se ha establecido hasta junio de 2023.

Con motivo de la colaboración para la realización de un Trabajo de Fin de Grado (TFG) del alumnado de la titulación Grado en Ingeniería de la Salud de la Universidad de Burgos durante el curso académico 2022-2023, la investigadora Patricia Saiz López propone el estudio de la generación de un algoritmo capaz de automatizar la extracción de datos de informes de secuenciación masiva.

El estudio se realizará cumpliendo los criterios éticos internacionales recogidos en la Declaración de Helsinki, garantizando no difundir el material a terceros y la eliminación de los materiales o sus copias en un plazo máximo de 15 días naturales.

 Complejo Asistencial Universitario de Burgos	ACUERDO DE COLABORACIÓN
-------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------

Servicio Anatomía Patológica		
Burgos	14 de marzo, 2023	Página 1 de 2

Acuerdo entre el investigador: Dra. ~~RAFAELA GARCÍA GÓMEZ~~, DNI ~~XXXXXXXXXX~~, con vinculación laboral al Hospital Universitario de Burgos (HUBU),

y el solicitante: LUCÍA VITORES LÓPEZ, DNI: ~~XXXXXXXXXX~~, ANTONIO JESÚS CANEPA ONETO, DNI/NIE: ~~XXXXXXXXXX~~, con vinculación a la Universidad de Burgos (UBU),

para la gestión de datos bajo colaboración temporal, por el que se establece:

1. Motivo de colaboración y cesión de material:

Colaboración para la realización del Trabajo fin de Grado (TFG) de alumnado de la titulación Grado Ingeniería de la Salud de la Universidad de Burgos (UBU), curso 2022-2023; la investigadora Dra. ~~RAFAELA GARCÍA GÓMEZ~~, propone el estudio:

Generación de algoritmo que facilite la exportación automática a base de datos propia, a partir de los campos relevantes para el manejo de datos de secuenciación masiva, generados mediante software comercial en formato de informe en pdf.

Título provisional: "Automatización de extracción de datos de informes de secuenciación masiva".

Asignado para su realización a la alumna: Lucía Vitores López, DNI: ~~XXXXXXXXXX~~. Tutorizada en la UBU por: Tutor 1. Dr. Antonio Jesús Canepa Oneto. Conformando la parte investigadores de la UBU/receptora.

Tutorizada en el HUBU por: ~~Rafaela García Gómez~~. Conformando la parte investigadora del HUBU/ cedente.

Conformando todos ellos el equipo investigador.

2. Datos y/o material solicitados al HUBU: documentos formato .pdf, generados con el software OncoPrint™ Reporter. Solicitados mínimo 100 informes.

3. Costes derivados de cesión: esta cesión es gratuita para los investigadores.

4. Entrega del material y plazos: se realizará vía correo electrónico institucional salvo indicación de otra vía por la parte receptora, durante el primer semestre de 2023, conforme a los requisitos del solicitante.

Este documento es propiedad de ANATOMÍA PATOLÓGICA del Hospital Universitario de Burgos.
Queda prohibida su reproducción total o parcial, sin autorización expresa y escrita de la empresa.

Figura A.1: Acuerdo de confidencialidad HUBU-UBU para la realización del proyecto.

Apéndice *B*

Documentación de usuario

B.1. Requisitos software y hardware para ejecutar el proyecto.

El usuario debería clonar el repositorio para tener una copia completa de este en su dispositivo, lo que le permitiría conocer todos los cambios realizados desde el inicio del proyecto, las versiones anteriores del código y la propia evolución. De esta forma, si se realizan modificaciones posteriores, se tendría la última versión y no sería necesario volver a descargar nada.

Para poder ejecutar este proyecto, es necesario tener instalado tanto Python como Anaconda en el ordenador.

B.2. Instalación / Puesta en marcha

El notebook que contiene el código puede encontrarse [clicando](https://github.com/LuciaVitores/Automatizacion_PDF_scraping/blob/main/CodigoPython.ipynb) o en la URL https://github.com/LuciaVitores/Automatizacion_PDF_scraping/blob/main/CodigoPython.ipynb.

Para poder ejecutar correctamente el código es necesario tener instalado Python. Para ver si tenemos Python instalado en Windows, podemos ejecutar en el cmd o símbolo del sistema el comando **py --version** para saber que versión estamos usando. Si está instalado, devolverá la versión disponible.

En el caso de usar macOS se debe buscar la terminal dentro de la carpeta aplicaciones y ejecutar el comando **python3**, devolviendo la versión usada. Si no se encuentra, instalada devolverá un error. [1]

Si se está trabajando con Linux, se debe abrir el terminal y ejecutar **python --version**. [2]

En caso de no tenerlo instalado, es necesario instalar la última versión. [3]

Para poder usarlo, nos descargamos Anaconda, que contiene aplicaciones, librerías y conceptos diseñados para el desarrollo de la ciencia de datos con Python.

Tras descargar Anaconda Navigator aparecen una serie de aplicaciones que ya vienen por defecto, como por ejemplo Jupyter Notebook, que es con la que vamos a trabajar.

Una vez instalado, podemos acceder siguiendo la ruta de las carpetas a la carpeta que tiene toda la información del proyecto. Ahí podemos crear un nuevo fichero para el desarrollo del código necesario.

También es necesario saber si **pip** está disponible para ser usado. Generalmente ya viene instalado, pero en el caso de usar Linux e instalarlo desde un administrados de paquetes del sistema operativo, es posible que se deba instalar por separado. [4]

Pip es un sistema de administración de paquetes que se usa para instalar y administrar los paquetes de software de Python antes de su importación y uso. Debemos comprobar si está instalada, ya que sino deberíamos hacerlo.

En Windows se abre el símbolo del sistema (cmd) y se escribe comando **pip --version** Si tienes pip instalado, verás la versión de pip instalada en tu sistema. Si no está instalado, mostrará un mensaje de error indicando que el comando "pip"no se reconoce.

En macOS y Linux se abre la terminal y se ejecuta el mismo comando que en Windows **pip --version**. En Linux, es posible que pip no funcione y deba usarse pip3, ya que no son todos los comandos iguales.

Si no tienes pip instalado, puedes instalarlo siguiendo las instrucciones adecuadas para tu sistema operativo, las cuales se pueden encontrar [clicando](https://pip.pypa.io/en/stable/installation/) o en la URL <https://pip.pypa.io/en/stable/installation/>

Para poder instalar cualquier biblioteca en Windows, solo hay que poner **python -m pip install biblioteca** en la consola, siendo biblioteca el nombre de la biblioteca que nos interesa instalar.

En el caso de macOS se debería ejecutar **python -m pip --biblioteca**. Este código instalará la última versión de la biblioteca que queremos instalar. [5]

Para Linux se puede entrar en esta [dirección https://pypi.org/](https://pypi.org/) y escoger el paquete que se quiere instalar, se copia el comando de instalación y se ejecuta en la terminal. [6]

Otra posible opción para instalar estos paquetes es desde Anaconda, escogiendo el entorno de trabajo que vayamos a usar. En el apartado de búsqueda se escribe el nombre de la biblioteca a instalar y no es necesario usar comandos.

En nuestro caso habría que instalar Pandas, Numpy, Re, Os y PyMuPDF (fitz).

Una vez tenemos instalados los paquetes de interés, creamos un fichero para trabajar sobre él e importamos en la primera celda las distintas bibliotecas que vamos a ejecutar:

- **Import os:** [7] Este módulo proporciona una forma portátil de usar el sistema operativo. Se puede usar **open()** para leer o escribir un archivo, **os.path()** para modificar o manipular rutas, **file.input()** para leer todas las líneas de los archivos. Todas las funciones generan un **OSError** cuando las rutas o los nombres de los ficheros no son correctos o no existen. Esta biblioteca se ha usado principalmente para listar los nombres de a ruta y unir los PDF de una ruta en una cadena.
- **Import numpy as np:** es una biblioteca importada para trabajar principalmente con grandes conjuntos de datos, matrices multidimensionales, operaciones matemáticas, etc. Es muy rápida y eficiente. [8]. En este código se ha usado principalmente para la exportación de las tablas finales a un formato Excel y la importación de información sobre genes y diagnósticos.
- **Import fitz:** para esta importación es necesario tener instalada la biblioteca PyMuPDF, que actúa como enlace para la biblioteca MuPDF, la cual nos permite trabajar con ficheros PDF. El comando **fitz.open()** nos permite abrir el fichero en formato PDF con el que vamos a trabajar. A la hora de descargarlo, puede que de un problema debido a que tiene una interfaz que no está en el paquete, para ello ejecutamos **python -m pip install --upgrade pymupdf** y se eliminaría el problema. Esta biblioteca se ha usado para abrir los ficheros especificados en formato PDF.
- **Import re:** este módulo nos permite trabajar con expresiones regulares. Las expresiones regulares son un pequeño lenguaje dentro del lenguaje

de programación que especifica un conjunto de caracteres posibles que se desean hacer coincidir con el texto sobre el que se está trabajando. No todas las búsquedas se pueden realizar siguiendo este método, en algunos casos es mejor realizar un código de búsqueda que usar dichas expresiones, aunque sea un proceso más laborioso, puede llegar a ser más comprensible. Sin esta biblioteca no se pueden ejecutar las expresiones regulares como patrones de búsqueda.

- **Import pandas as pd:** la comunidad de Python ha adoptado una serie de nomenclaturas convencionales para los módulos de uso más común. Entre ellos se encuentra `import pandas as pd`, junto con `import numpy as np` o `import matplotlib.pyplot as plt`. [9] Pandas permite realizar muchas funciones dentro del análisis de datos, proporcionando estructuras de datos, Series y DataFrames. Hay otros parámetros que se pueden añadir a la función como **sheet-name**: permite escoger las hojas que se quieren usar y que en nuestro código dejamos lo que viene por defecto (cero) porque nos permite obtener todas, **usecols**: permite escoger con qué columnas queremos trabajar (como por ejemplo `^:E`, que en este caso indica que queremos trabajar con las columnas que van desde la A hasta la E, ambas incluidas) o **nrows**: determina el número de columnas con las que vamos a trabajar, aunque en este caso también dejamos el valor que viene por defecto que es None. En nuestro caso se ha usado para la importación de los datos de dos hojas de cálculo de un fichero Excel para crear distintos DataFrames usando `pd.read-excel(io = ruta/fichero/Excel.xlsx)`.

Una vez tenemos instalados los paquetes de interés, los importamos en el fichero en el que vamos a trabajar.

Todas las importaciones siguen una metodología común:[10]

`import + nombre/biblioteca + as + nombre/abreviado`

- **Import:** importa la funcionalidad o bibliotecas en el script en el que se está trabajando.
- **Nombre/biblioteca:** nombre de la biblioteca que se quiere importar.
- **As:** alias, es decir, permite tomar una palabra larga y hacer referencia a ella usando una palabra más corta.
- **Nombre/abreviado:** nombre abreviado estándar para hacer referencia al nombre de la biblioteca.

Ya ejecutada la celda donde se encuentran las importaciones, no debería devolvernos ningún mensaje.

En la siguiente celda se definen las carpetas con las que vamos a trabajar, ya sean de entrada o de salida. Para ello se necesitan colocar los ficheros y las carpetas según la distribución que indica [B.1](#):

- INPUT
 - DATOS
 - Diagnostico.xlsx
 - Genes.xlsx
 - INFORMES
 - (Aquí se colocarán todos los ficheros PDF que se quieran analizar)
- OUTPUT
 - (Aquí se generarán los ficheros con los resultados globales)
 - RESULTADOS
 - (Aquí se generarán los resultados parciales de los ficheros PDF analizados)

Figura B.1: Distribución que se debe seguir a la hora de crear las carpetas para almacenar los datos y resultados obtenidos.

La carpeta INPUT almacenará los ficheros necesarios para ejecutar el programa, mientras que la carpeta OUTPUT almacenará únicamente los resultados.

Dentro de INPUT podemos ver que hay dos subcarpetas, DATOS, que almacena los ficheros Excel creados con los nombres de los genes/diagnósticos y su número correspondiente e INFORMES, que es donde se guardan los informes que se quieren analizar.

La carpeta OUTPUT almacena los ficheros Excel generales resultantes y los parciales, pero estos últimos los guarda en una carpeta específica para una mejor organización.

De esta forma no es necesario introducir manualmente las rutas del ordenador, sino que teniendo las carpetas indicadas ya creadas, se usará la carpeta INPUT como datos de entrada y OUTPUT como carpeta de salida.

Para comenzar se crearon tres funciones nuevas, para encontrar los ficheros PDF de la ruta seleccionada y leerlos.

La función **LeerFicherosPDF** recibe una ruta como parámetros, con el objetivo de leer todos los archivos PDF que se encuentren tanto en la

ruta especificada como en sus subcarpetas. Se definen dos listas vacías denominadas `ficheros` y `subcarpetas`. Se utiliza el módulo `os.walk()` para recorrer la ruta y obtener todas las subcarpetas, directorios y archivos en esa ruta y en sus subcarpetas. El resultado es una tupla que contiene tres elementos: la ruta actual, una lista de directorios encontrados en la ruta actual y una lista de archivos encontrados en la ruta actual. Se ordena la lista de subcarpetas numéricamente utilizando la función `sorted`, asegurando que las subcarpetas se ordenen correctamente incluso si sus nombres contienen números. Se itera sobre cada subcarpeta y se construye la ruta completa de la subcarpeta utilizando `os.path.normpath()` y `os.path.join()`. Se itera sobre cada archivo en la subcarpeta para comprobar que tiene una extensión `.pdf` y en caso de ser así, lo añade a la variable `ficheros`. Después de procesar todas las subcarpetas, se itera sobre los archivos en la ruta principal. Si un archivo tiene una extensión `".pdf"`, se construye su ruta completa y se agrega a la lista `ficheros`, devolviendo una lista que contiene las rutas completas de todos los archivos PDF encontrados en la ruta y sus subcarpetas.

Con la función **LeerDocumento** con argumento `nombreFichero` (que es el nombre de cada archivo), se lee el contenido de cada argumento que se le pasa devolviendo una lista de cadenas de texto, donde cada cadena representa una línea del documento. Para ello se usa la librería PyMuPDF (`fitz`). Se crea un bucle con `with()`, que permite abrir el archivo especificado por `nombreFichero`, asegurando que el archivo se cierre correctamente una vez que se haya terminado de leer. Dentro del bucle, se inicializa una cadena vacía llamada `text` que almacenará el contenido del documento. Se itera sobre cada página del documento utilizando un bucle `for` y la variable `page`. Para cada página, se obtiene el texto utilizando el método `get_text()`, que extrae el texto de cada página del fichero. Después de procesar todas las páginas, se utiliza el método `split("")` para dividir la cadena `text` en una lista de cadenas. Esto divide el texto en líneas individuales. Finalmente, se devuelve la lista resultante que contiene cada línea del documento como un elemento de la lista. Con la función `fitz.open` se abre el documento denominado `nombreFichero` e inicializa la variable `text` como una cadena vacía. La finalidad de esto es que una vez se entre en el bucle `for`, se recorran todas las páginas del documento PDF y se vaya rellenando la variable `text` con el texto de los distintos PDF, devolviendo una lista con los fragmentos de texto encontrados después de la aparición del texto buscado.

La función **BuscarValor** permite buscar una palabra de interés en el texto que representa la información de cada fichero. El código creado busca la palabra que entra como argumento en la lista de cadenas denominada `lines`, para ello usa la biblioteca `re`. Se crea una lista vacía llamada `Encontrados` que

almacenará los fragmentos de texto encontrados. Se recorre cada línea en la lista `lines` y se realiza una búsqueda del texto especificado por `textoBuscar` en cada línea utilizando la función `re.search()`. Si se encuentra una coincidencia, se asigna el valor 1 a ese elemento en la lista `valores`, de lo contrario se asigna 0. Se utiliza otra comprensión de lista para obtener los índices de los elementos en `valores` que tienen el valor 1. Esto se consigue usando `enumerate(valores)` `if s==1 in valores`. La variable `i` representa el índice de un elemento en `valores` que contiene una coincidencia. Se itera sobre los valores encontrados en `valores` utilizando el bucle `for` y la variable `i`. Se añade a la lista `Encontrados` el fragmento de texto que se encuentra después de la aparición del texto buscado en cada línea. Para ello se usa el módulo `rfind` para encontrar la última aparición del texto buscado en la línea y luego se extrae el fragmento utilizando slicing y el método `strip()` para eliminar espacios en blanco al principio y al final. Devuelve la lista `Encontrados` con los fragmentos de texto encontrados después del texto buscado en cada línea.

La función comentada denominada **GenerarImagen** es un código adicional que permite obtener una imagen por cada una de las hojas que forman el PDF. En caso de querer usar este código solo habría que descomentarlo, eliminando las comillas iniciales y finales.

Una vez creadas las funciones se procede a importar los ficheros Excel proporcionados por el hospital, ya que estos contienen información fundamental para la correcta ejecución y obtención de datos.

En el apartado **2.2 Importación de ficheros Excel**, se importan dos ficheros Excel. Uno con información sobre los diagnósticos y el número correspondiente a cada diagnóstico y otro con los genes y los números correspondientes a cada gen. La primera línea del código usa `os.path.normpath` y `os.path.join` para construir una ruta de archivo completa. Tomando las variables `PathBase`, `CarpetaEntrada`, `CarpetaDatos` y el nombre del archivo `Diagnostico.xlsx` o `Genes.xlsx` y las combina en una ruta normalizada. Posteriormente se usa la función `pd.read_excel` de Pandas seguido de la ruta relativa donde se encuentra cada uno de los ficheros que se desea importar. Se crea un diccionario por cada fichero importado usando `dict` (función que permite crear diccionarios) junto con `zip` (permite tomar dos o más secuencias y combinarlas en una tupla de dos elementos).

En el caso de las mutaciones, se crea una única variable que almacene solo los nombres de las funciones, ya que servirá posteriormente para detectar las mutaciones en el texto. En ambos casos hacemos un `for` para almacenar cada clave junto con su valor correspondiente.

Una vez importados los ficheros, se procede a buscar las variables de interés a lo largo de los ficheros encontrados en la ruta.

En el apartado **2.3. Definición de algunas variables** definimos la ruta donde tenemos los documentos para que sea usada en todo el código. Inicializamos las variables que vamos a obtener en este apartado como listas vacías, usando para ello dos corchetes (`[]`). Creamos una variable nueva llamada `ficheros` que usando la función anterior ‘LeerFicherosPDF’ con argumento `ruta`, nos lee los ficheros de esa ruta. Hacemos un bucle `for` para que en cada uno de los ficheros use la función ‘LeerDocumento’ para tener almacenadas las listas de texto de cada uno de los ficheros. Posteriormente se usa la función ‘BuscarValor’ que tiene como argumento la palabra que se quiere buscar y la variable `lines` definida anteriormente. En caso de querer crear las imágenes, a parte de descomentar el código anterior, también es necesario descomentar la fila comentada que contiene `GenerarImagen(ruta, ficheroPDF)`. Finalmente uso `print` para imprimir las variables obtenidas, llamadas `NHC_Data`, `Nbiopsia_Data`, `fecha_Data` y `texto_Data`.

Como vemos, el resultado de cada variable es una lista que contiene ocho sublistas en su interior (una por cada uno de los ficheros). En estos casos solo nos interesa tener un valor por fichero, no tantas repeticiones como veces aparece la palabra.

Para ello en el apartado **2.3.1. Diagnóstico** definimos dos variables denominadas `textoDiag` y `numeroDiag` y creamos un bucle `for` para recorrer los elementos de la variable `texto_Data` obtenida en el apartado anterior. Se utiliza el comando `list(set())` para eliminar los elementos duplicados en la lista `iy` se meten en una lista llamada `sinduplicados` que contiene los elementos únicos de la lista `i`. Se vuelve a iterar sobre cada elemento en la lista `i` y se agrega el valor solo si también aparece en la variable `sinduplicados`. Se accede al primer elemento usando `[0]`. Finalmente, se añade el primer elemento no duplicado de cada una de las listas a `textoDiag` usando el método `append()`.

Posteriormente se itera en cada uno de los elementos almacenados en `textoDiag`. En caso de que coincida el diagnóstico del texto con alguna de las claves del diccionario, se obtiene el valor respectivo. Los valores se almacenan en una función llamada `numeroDiag`.

En el apartado **2.3.2. NHC** se crea una variable dentro de un bucle `for`, denominada `sinduplicadosNHC`, que permite almacenar una única copia de los valores almacenados en `NHC_Data`. Los valores almacenados en esta variable se sacarán del bucle y se añadirán a la variable `NHC`.

De esta forma solo se ha creado una lista con solo una copia de los resultados siguiendo el orden original.

En la subsección **2.3.3 Biopsia** se define una lista vacía llamada `lista_resultante` y un conjunto vacío denominado `elementos_vistos` usando `set()`. Se itera sobre los elementos de `Nbiopsia_Data` y dentro del bucle, se crea una lista vacía llamada `sublist_sin_duplicados` para almacenar los elementos recorridos sin repetir. Se vuelve a iterar, pero esta vez sobre los elementos en cada sublista de `Nbiopsia_Data`. Verifica si el elemento no está presente en el conjunto `elementos_vistos` (para evitar duplicados). Si el elemento no está en `elementos_vistos`, se añade a la variable `sublist_sin_duplicados` y a `elementos_vistos`. `Sublist_sin_duplicados` se añade a la lista `lista_resultante` mediante el método **append**. Una vez obtenido este resultado, nos interesa que sea una lista, no una lista con sublistas. Para ello se crea una lista llamada `NB_values` donde recorre la lista anterior sacándolas de la sublista.

Como nos interesa saber el tercer caracter de cada valor, lo que hacemos es recorrer cada palabra y sacar el tercer elemento (`x[2]`) y almacenamos los resultados en una nueva variable denominada `biopsia`.

En el caso de biopsia sólida vista en el apartado **2.3.4 Biopsia sólida** se ve cómo se define una lista denominada `Biopsia_solida` y a cada una de las tres posibles opciones se la da un valor numérico. Se crea también un bucle `for` que itera sobre la variable `biopsia` del punto anterior para comparar cada letra y almacenar su número correspondiente en una nueva variable denominada `Biopsia_solida`.

Finalmente, en el apartado **2.3.5 Fechas** se itera sobre la lista `fecha_Data` y se crea una nueva lista llamada `fechas` que contiene solo el primer elemento de cada sublista.

Una vez que pasamos a la sección **2.4. Definimos el resto de variables**, la primera subsección que encontramos es la de **2.4.1 Ensayos clínicos y tratamientos disponibles**. Para obtener las variables que contengan los valores de ensayos clínicos y tratamientos disponibles se usan expresiones regulares.

En ambos casos se ha usado el mismo patrón, ya que nos interesaba que la cadena de texto contenga cualquier número de 0 al 9 ambos incluidos y se encontrara antes de la palabra definida posteriormente. Si nos fijamos bien, en todos los casos aparece el número seguido de la palabra Ensayos o Tratamiento, lo que nos da una ventaja a la hora de realizar el patrón de búsqueda.

Para esta parte del código definimos dos listas vacías. Iteramos sobre ficheros para leer cada uno de los ficheros e inicializar una nueva variable denominada ensayos/tratamientos a 0. Se itera sobre cada línea en lines y se busca el patrón definido anteriormente con el método `re.search()`. De esta forma, si se encuentra un resultado se extrae el número entero usando `int(resultado.group(1))` y se valor se añade a la variable igualada a 0. La lista ensayos o tratamientos se añade a la lista `lista_ensayos` o `lista_tratamientos` para tener el número exacto de tratamientos/ensayos que hay en cada fichero.

El código es igual en ambos casos, solo cambia el nombre de las variables y la palabra siguiente a la expresión regular del patrón, de forma que se explican conjuntamente. Como también nos interesa binarizar la variable que almacena los números, se itera sobre esa misma lista añadiendo un 1 a la variable `ensayos_finales/tratamientos_finales` cuando el resultado sea un 1 o cualquier número mayor o se añade un 0 cuando el resultado sea un 0.

Todas estas variables se encontraban dentro de los PDF, sin embargo, hay cierta información que depende únicamente del nombre que le asigna el software a cada uno de los ficheros que crea.

Para calcular el número de chip y de paciente en el punto **2.4.2 Número de chip y de paciente** nos tenemos que fijar en el nombre del fichero. Primero iteramos sobre la variable `ficheros`. Con el condicional `if` podemos ver si cada elemento de la lista `ficheros` pertenece o no a un archivo existente de la ruta dada. Para ello tenemos dos condiciones, en la primera usamos `os.path.isfile()` para combinar la ruta con cada uno de los ficheros, sobre esto realizamos `os.path.normpath()` para normalizar la ruta y finalmente `os.path.endswith()` para ver si es un archivo existente dentro de la ruta con extensión `.pdf`. Una vez se ha cumplido esto, podemos obtener tanto el número de paciente como el de chip.

Para sacar el número de paciente hay que crear una nueva lista vacía para que almacene dicho numero. Una vez creada, se itera sobre los archivos separa el nombre de la extensión usando `os.path.split()` y obtenemos únicamente el nombre del archivo con el comando `os.path.splitext()` accediendo con `[0]` al primer elemento. Dentro del nombre del fichero, obtenemos el séptimo valor usando `[7]` y agregamos este valor a la variable `numero_paciente`.

Para el caso del chip nos interesa saber que este valor se encuentra entre una barra baja seguida de una v (esto es así por defecto). De forma que usamos esta información para crear un patrón con una expresión regular que nos permita buscar cualquier valor que esté entre dos `_` y comience por

v. Los resultados que cumplan estas condiciones son añadidos a una nueva lista llamada `chip2`.

Una vez que hemos sido capaces de obtener todos los valores de interés, nos centramos un poco más en el tema de las mutaciones.

En el subapartado **2.4.3.1. Mutaciones totales** se llama a la función ‘LeerFicherosPDF’ definida anteriormente para obtener las listas de los ficheros de esa ruta. Se definen varias variables como son `max_mut` inicializado a 0, `genes_mut2` para crear un diccionario donde las claves serán los nombres de los ficheros y los valores las mutaciones que hay en cada uno y finalmente la variable `frecuencias_totales` para almacenar las frecuencias alélicas de cada mutación. En este caso también se usa una expresión regular para buscar valores que sigan el formato: dos números, un punto y otros dos números. Se itera sobre la lista `ficheros` para leer el contenido de cada fichero usando ‘LeerDocumento’ e inicializar nuevas variables llamadas `total_mut`, `encontrados2` y `lista_frec`.

Se vuelve a iterar sobre la variable `mutaciones` definida anteriormente en el apartado **2.2 Importación de ficheros Excel**. para ver si coincide la mutación del fichero con alguna de las mutaciones de la variable. En caso afirmativo, se obtienen la posición de la mutación usando `index(mutacion)`. Si la mutación coincide con FGFR4 se verifica si la siguiente posición es p.(P136L). En caso de que sea así, se omite, ya que esa mutación no interesa. En caso de que no se cumpla, a la variable `total_mut` se le suma 1 y se agrega la mutación a la variable `encontrados2`.

En cualquier otra mutación, se verifica si aparece la palabra Benign en alguna de las posiciones de la línea. En caso afirmativo, no interesa. En caso de que no aparezca se incrementa en uno la variable `total_mut` y se agrega la mutación a `encontrados2`. También se busca en un rango de diez posiciones, un patrón definido para buscar %, lo que indica la frecuencia alélica de cada mutación. Este valor se añade a `lista_frec` para almacenar todas las frecuencias de las mutaciones de interés. Se usa la variable `genes_mut2` como clave del diccionario y `encontrados2` como valor. También se tiene en cuenta el valor de `total_mut`, ya que si `max_mut` es mayor, se cambia el valor, indicando el número total de mutaciones de interés que hay en cada fichero. Finalmente se añade la lista `lista_frec` a `frecuencias_totales` para tener una variable con las frecuencias de los genes de interés. `Mut` es una nueva variable que se crea para almacenar únicamente los nombres de los genes (usando el comando `values()`). Si ejecutamos las celdas `frecuencias_totales` y `mut`, vemos que no coincide el número de mutaciones con el número de frecuencias. Esto se debe a que en algunos casos, hay mutaciones que no

tienen frecuencia alélica porque en su lugar tienen números de lectura o números de copias, no porque el código funcione mal. También se crea la variable `num_mutaciones` para calcular el número de mutaciones dentro de cada fichero. Para ello aplicamos la función `len()` en la variable anterior `mut`.

Como ya tenemos un diccionario con clave las mutaciones y valor el número correspondiente a cada mutación, se crea una nueva lista denominada `numero_iden`. Por cada gen que coincida con una clave, se añade su valor a la nueva variable. En caso de no encontrarse, se añade un 0. De esta forma, al imprimir la variable `numero_iden` se imprimen los valores correspondientes a las claves.

Hay casos en los que no se trabaja con genes, sino con fusiones. En estos casos se trabaja de una manera especial, ya que el código anterior no es capaz de detectar dichas fusiones. Para ello se ha desarrollado un código nuevo donde se crea una lista llamada `fusiones`. Y como en el caso anterior, se lee cada uno de los ficheros usando ‘LeerDocumento’ y almacenándolo en `lines`. Se inicializa una lista vacía llamada `variantes`, donde se irán metiendo las variables encontradas al recorrer cada archivo. Se itera sobre `lines` y posteriormente sobre mutaciones para definir un nuevo patrón que sea capaz de encontrar cualquier palabra que cumpla la condición de tener cualquier letra (ya sea mayúscula o minúscula) o número que aparezca una o más veces y vaya seguida de - y cualquiera de las mutaciones. En el caso de que se encuentre alguna coincidencia, esta se almacena en una variable denominada `gen` y se vuelve a definir otro patrón para buscar el ID de cada elemento almacenado anteriormente. Para el patrón que busque los ID basta con buscar el nombre del gen seguido de un punto y un conjunto de letras (mayúsculas o minúsculas) combinada con números (esto aparece dos veces). Si se encuentra esta gran coincidencia, se almacena dentro de la variable `variante` que esta a su vez se añade a la lista `variantes`. Una vez que se sale del bucle `for`, la lista `variantes` se añade a la lista `fusiones`.

Una vez que ya tenemos el apartado de las mutaciones desarrollado, va a ser mucho más fácil desarrollar el apartado **2.4.3.2. Genes patogénicos**. En este apartado se pretende hacer lo mismo que en el anterior, pero teniendo en cuenta solo los genes patogénicos.

En este caso lo que hacemos una vez determinamos la posición es iterar sobre las posiciones (de la uno hasta la diez) para ver si la cadena `Pathogeni` está en alguna de las posiciones. En caso de encontrarla, se imprime el fichero con la mutación.

Posteriormente se crea un nuevo patrón que busca dos dígitos seguidos de un punto y otros dos dígitos. Se crea también una nueva lista vacía llamada `frecuenciasPato` para almacenar únicamente las frecuencias de las mutaciones patogénicas. Se itera sobre los ficheros y se lee cada uno de ellos almacenando el texto en líneas. Se inicializa también una lista vacía llamada `lista_fec` donde se van almacenando las frecuencias del archivo actual sobre el que se está trabajando para luego añadir esta información a la variable `frecuenciasPato` una vez se haya salido del bucle. Igual que en el caso anterior, iteramos sobre las mutaciones y en caso de encontrar alguna que coincida en el texto, se usa su posición para ver si en las posiciones posteriores a esa se encuentra la palabra `Pathogeni`. En caso de que sí, se usa el patrón para determinar su frecuencia y añadir estos valores a `lista_frec`. Al final del bucle `lista_frec` se añade a la `frecuenciasPato`.

Posteriormente se crea un diccionario vacío llamado `patogen` donde se almacenarán las mutaciones patogénicas de cada uno de los ficheros. En este caso el código es igual que el anterior, lo único que se modifica es que se va añadiendo el nombre de la mutación a la variable `genpato2` y una vez se sale del bucle, se añade a `patogen` para tener las mutaciones encontradas que cumplen estas características dentro de cada fichero.

Como nos interesa tener una variable únicamente con los nombres, se cogen solo los valores del diccionario `patogen`, usando `values()`.

Como estas mutaciones también se encuentran en el fichero importado, a cada una de ellas le corresponde un número. Para hacer esta asignación se ha creado una celda donde se inicializa una lista vacía llamada `numero_iden_pato` para almacenar los valores asociados a esas mutaciones. Se itera sobre la lista `patológicos` y se vuelve a iterar sobre cada uno de los elementos de esta lista para crear un diccionario llamado `mutaciones_dic` para obtener el valor numérico asociado a esa mutación. Estos resultados se añaden a la variable `numero_iden_pato`.

Finalmente contamos el número de mutaciones patogénicas que hay en cada fichero usando `len()`.

En este apartado se procede a realizar un resumen de las variables encontradas que se van a usar, para comprobar que todas están dentro de una lista con el mismo número de resultados que número de ficheros con los que se está trabajando. En el caso de que las longitudes no coincidan, se devolverá un error a la hora de crear las tablas.

El apartado siguiente **2.5. Variables de interés** hace una recopilación de todas las variables de interés obtenidas que van a ser necesarias para la formación de los DataFrames.

Lo siguiente que se debe hacer es agrupar la información obtenida en función de su similitud.

En el siguiente apartado **3. Creación de DataFrames** se crean los distintos DataFrames para su posterior unión. Para crear cada uno de ellos se usa la función `pd.DataFrames(...)`. Los tres puntos suspensivos indican que ahí van las columnas que van a formar el DataFrame. Para ello se indica primero el nombre entre comillas simples, seguido de dos puntos verticales y posteriormente el nombre de la variable. Se pueden poner tantas columnas como se quiera. Una vez creados todos los necesarios, se usa el método `join` para su unión. Para ello se determina el nombre de la tabla que se quiere formar, seguido de un igual y la forma de unión, es decir, se indica la primera tabla que se quiere unir seguida de `.join(nombre-segunda-tabla.set_index([claves]))`. Realizamos este mismo proceso todas las veces que haga falta para obtener el DataFrame final. Finalmente conseguimos dos tablas finales, una con toda la información sobre los genes patogénicos y otra con toda la información de todos los genes.

En el apartado **4. Exportación** se indica cómo exportar las tablas a un formato Excel. Con la función `tabla-a-exportar.to_excel('nombre-tabla.xlsx')` se exporta el DataFrame a un archivo Excel sin índices, almacenando los resultados en descargas. Se exportan dos tablas, una final de los genes patogénicos y otra final con todos los genes.

Como tener una tabla con toda la información de todos los ficheros puede ser algo difícil para trabajar, también se ha tenido en cuenta el crear distintas tablas que en lugar de almacenar todos los datos, tengan solo 80 líneas (lo que correspondería con 8 chips).

Para ello las dos últimas celdas se han enfocado en obtener estos resultados.

Se usa la función `np.array_split` para dividir el DataFrame que contiene toda la información en distintos DataFrames con menos contenido. La división se realiza cogiendo fragmentos de un máximo de 80. Estos fragmentos se almacenan en la variable `fragmentos`. Se genera un bucle sobre la variable anterior usando la función `enumerate` para obtener el índice del fragmento como el fragmento en sí. Para cada uno de los fragmentos se crea un archivo Excel específico usando el método anterior `to_excel`.

Una vez entendido y copiado todo el código en el fichero, se ejecuta. Para ejecutar el código basta con dar al botón **Run** que sale en la parte superior del fichero o usando las teclas Ctrl + Enter.

Como resultados, en el apartado **2.5. Variables de interés** podemos ver el resultado que almacena cada variable y en el **3. Creación de DataFrames** vemos los DataFrames finales creados.

Podemos ver que se ha creado una carpeta denominada OUTPUT que en su interior contiene un fichero Excel con los resultados generales obtenidos (TablaGeneral.xlsx) y otro con los resultados solo de los genes patogénicos (TablaPato.xlsx). A su vez, vemos que también se ha creado una subcarpeta denominada RESULTADOS, donde se almacenan los resultados parciales.

B.3. Manuales y/o Demostraciones prácticas

Para la entrega de este proyecto se ha creado una máquina virtual para que el tribunal pueda reproducir el programa sin tener que realizar ninguna descarga, evitando problemas de compatibilidad o dependencias no instaladas, ya que todo el entorno necesario para la ejecución del programa se encuentra dentro. Además, es compatible con los distintos sistemas operativos, por lo que no habría ningún problema a la hora de ejecutarlo.

Para ello se había usado Virtual Box, un software desarrollado por Oracle Corporation que permite la ejecución de máquinas virtuales con diferentes características disponible gratuitamente como software de código abierto [11].

Sin embargo, se estudió una forma más fácil de entregarlo. Por ello se creó un script de Python (fichero .py). Una vez creado el script en Python, debemos de generar un fichero ejecutable independiente, para que pueda ser arrancado desde la consola del sistema. Para ello instalamos la utilidad **pyinstaller** usando **pip install nbconvert**.

Una vez instalada, debemos de ejecutarla para generar el fichero .exe con la siguiente instrucción **pyinstaller --onefile .CodigoPython.py**, siendo CodigoPython.py el nombre del fichero .py creado.

Para que la aplicación funcione correctamente necesitamos colocar todos los ficheros y carpetas según la distribución **B.2**:

- CodigoPython.exe
- INPUT
 - DATOS
 - Diagnostico.xlsx
 - Genes.xlsx
 - INFORMES
 - (Aquí se colocarán todos los ficheros PDF que se quieran analizar)
- OUTPUT
 - (Aquí se generarán los ficheros con los resultados globales)
 - RESULTADOS
 - (Aquí se generarán los resultados parciales de los ficheros PDF analizados)

Figura B.2: Distribución que se debe seguir teniendo en cuenta el archivo .exe.

Apéndice *C*

Manual del programador

C.1. Introducción

Este anexo tiene como finalidad detallar cómo funciona el código, que realiza cada función, qué resultados se obtienen, etc.

El proyecto está disponible en el repositorio GitHub disponible en este [hiper-vínculo](https://github.com/LuciaVitores/Automatizacion_PDF_scraping/blob/main/CodigoPython.ipynb) o en la URL https://github.com/LuciaVitores/Automatizacion_PDF_scraping/blob/main/CodigoPython.ipynb.

C.2. Estructura de directorios

Entre los ficheros de entrega constarán:

1. **Código Python:** el código en Python en un script llamado CodigoPython.ipynb para que cualquier persona que quiera probarlo pueda ejecutarlo sin ningún problema, con comentarios para entender el funcionamiento y la finalidad de cada función.
2. **CodigoPython.py:** fichero que guarda el script de Python.
3. **README.md:** fichero que resume la información más relevante del proyecto, incluyendo nombre del tutor, resumen del proyecto, universidad, etc.
4. **Memorias:** es una carpeta que contiene dos ficheros PDF, uno de ellos recoge toda la información sobre los objetivos, introducción, metodología, conclusiones y líneas futuras para el correcto desarrollo

del proyecto y el otro almacena toda la información correspondiente a los anexos del proyecto.

5. **INPUT**: es una carpeta que contiene cinco subcarpetas. La primera de ellas denominada **Datos** que contiene los ficheros Excel para la importación de sus datos, la segunda carpeta **DATOS** que contiene información sobre los distintos tipos de cáncer en un periodo de tiempo, la tercera **HUBU** que contiene el Power Point usado para la presentación del proyecto en el hospital, la cuarta **Imágenes** que contiene imágenes usadas a lo largo de la memoria y la quinta **Informes** contiene los informes aportados por el Hospital Universitario de Burgos (ya anonimizados) para tener una idea sobre cómo se organizan y poder trabajar sobre ellos.
6. **OUTPUT**: carpeta que almacena los resultados obtenidos tras la automatización de extracción de los datos de los informes.

C.3. Compilación, instalación y ejecución del proyecto.

Una vez tenemos instalado tanto Python como su interfaz gráfica, accedemos a Jupyter notebook para cargar el código.

En nuestro caso habría que instalar las librerías:

- Pandas
- Re
- Os
- PyMuPDF
- Numpy

Una vez tenemos instalados los paquetes de interés, los importamos en la primera celda. Cuando se ha ejecutado la celda sin ningún problema, podemos ejecutar el código que viene a continuación.

Definimos las carpetas que vamos a usar a lo largo del proyecto, tanto de entrada como de salida.

La función **LeerFicherosPDF** recorre todos los archivos y directorios de la ruta especificada, comprobando que los archivos tengan una extensión .pdf y almacenándolos en una variable para usarlos posteriormente.

La función **LeerDocumento** usa la biblioteca PyMuPDF para abrir cada uno de los archivos .pdf y extraer el texto que se encuentra en cada una de las páginas que lo forman para devolver una variable con ese contenido.

La función **BuscarValor** es capaz de buscar una palabra dentro de la variable que almacena el texto de cada archivo.

La función **GenerarImagen** genera una imagen por cada una de las hojas que tiene cada uno de los ficheros de la ruta. Para poder usar esta función basta con descomentarla.

Estas funciones van a ser llamadas posteriormente el otras funciones para su ejecución, ya que por sí solas no devuelven nada.

En el apartado **2.2 Importación de ficheros Excel**, se importan dos ficheros Excel distintos. Uno con información sobre los diagnósticos y el número correspondiente a cada diagnóstico y otro con los genes y los números correspondientes a cada gen. Creando un diccionario en ambos casos para trabajar mejor con la información.

En el apartado **2.3. Definición de algunas variables** definimos la ruta donde tenemos los documentos para que sea usada en todo el código. Se llama a las tres funciones creadas para leer el texto de cada uno de los ficheros y buscar las palabras de interés seguidas de :.

Como vemos, el resultado de cada variable es una lista que contiene ocho sublistas en su interior (una por cada uno de los ficheros). En estos casos solo nos interesa tener un valor por fichero, no tantas repeticiones como veces aparece la palabra.

Para ello en el apartado **2.3.1. Diagnóstico** recorreremos los elementos obtenidos de la celda anterior y eliminar los elementos duplicados en la lista. Se accede al primer elemento usando [0] para tener un único valor para cada archivo.

También se realiza la asignación del valor numérico a cada diagnóstico, empleando para ello el diccionario creado a la hora de importar el Excel con la información de los diagnósticos.

En los apartados **2.3.2. NHC**, **2.3.3 Biopsia** y **2.3.5 Fechas** el objetivo es el mismo, obtener únicamente un valor para cada variable por cada fichero. En el caso de biopsia, también se tiene en cuenta el tercer elemento de cada

valor, ya que este posteriormente definirá de forma numérica si se trata de biopsia, punción o citología.

Una vez que pasamos a la sección **2.4. Definimos el resto de variables**, la primera subsección que encontramos es la de **2.4.1 Ensayos clínicos y tratamientos disponibles**. Mediante expresiones regulares podemos obtener el número de tratamientos disponibles y ensayos clínicos que hay para paciente, en función de sus genes y diagnóstico. Estos valores también se binarizan para proporcionar un estudio más sencillo.

Para calcular el número de chip y de paciente en el punto **2.4.2 Número de chip y de paciente** nos tenemos que fijar en el nombre del fichero.

Primero iteramos sobre la variable ficheros.

Para sacar el número de paciente hay que crear una nueva lista vacía para que almacene dicho número. Lo obtenemos accediendo con [0] al primer elemento. Dentro del nombre del fichero, obtenemos el séptimo valor usando [7].

Para el caso del chip nos interesa saber que este valor se encuentra entre una barra baja seguida de una v (esto es así por defecto). De forma que usamos esta información para crear un patrón con una expresión regular que nos permita buscar cualquier valor que esté entre dos `_` y comience por v.

Una vez que hemos sido capaces de obtener todos los valores de interés, nos centramos un poco más en el tema de las mutaciones.

En el subapartado **2.4.3.1. Mutaciones totales** se llama a la función **LeerFicherosPDF** definida anteriormente para obtener las listas de los ficheros de esa ruta y se itera sobre la lista de ficheros para leer el contenido de cada uno usando **LeerDocumento**. Se comprueba que las mutaciones de los ficheros coinciden con las del archivo Excel importado en el apartado **2.2 Importación de ficheros Excel**. En caso afirmativo, se obtienen la posición de la mutación. Si la mutación coincide con FGFR4 se verifica si la siguiente posición es p.(P136L). En caso de que sea así, se omite, ya que esa mutación no interesa. En caso contrario, se almacena.

En cualquier otra mutación, se verifica si aparece la palabra Benign en alguna de las posiciones de la línea. En caso afirmativo, no interesa. En caso de que no aparezca se busca en un rango de diez posiciones, un patrón definido para buscar %, lo que indica la frecuencia alélica de cada mutación. Posteriormente se crea una variable para almacenar únicamente los nombres de los genes y sacamos la longitud del número de genes que cumplen estas condiciones en cada uno de los ficheros.

Como ya tenemos un diccionario con clave las mutaciones y valor el número correspondiente a cada mutación, se crea una nueva lista donde por cada gen que coincida con una clave, se añade su valor a la nueva variable. En caso de no encontrarse, se añade un 0. De esta forma se imprimen los valores correspondientes a las claves.

Hay casos en los que no se trabaja con genes, sino con fusiones. En estos casos se trabaja de una manera especial, ya que el código anterior no es capaz de detectar dichas fusiones.

Se ha desarrollado un código nuevo donde se crea una lista, se lee cada uno de los ficheros usando **LeerDocumento** y almacena su contenido en **lines**. Se inicializa una lista vacía donde se irán metiendo las variables encontradas al recorrer cada archivo. Se itera sobre **lines** y posteriormente sobre las mutaciones para definir un nuevo patrón que sea capaz de encontrar cualquier palabra que cumpla la condición de tener cualquier letra (ya sea mayúscula o minúscula) o número que aparezca una o más veces y vaya seguida de - y cualquiera de las mutaciones. En el caso de que se encuentre alguna coincidencia, esta se almacena. Se vuelve a definir otro patrón para buscar el ID de cada elemento almacenado anteriormente. Para el patrón que busque los ID basta con buscar el nombre del gen seguido de un punto y un conjunto de letras (mayúsculas o minúsculas) combinada con números (esto aparece dos veces). Si se encuentra esta gran coincidencia, se almacena.

Una vez que ya tenemos el apartado de las mutaciones desarrollado, va a ser mucho más fácil desarrollar el apartado **2.4.3.2. Genes patogénicos..** En este apartado se pretende hacer lo mismo que en el anterior, pero teniendo en cuenta solo los genes patogénicos.

En este caso lo que hacemos una vez determinamos la posición es iterar sobre las posiciones (de la uno hasta la diez) para ver si la cadena Pathogeni está en alguna de las posiciones. En caso de encontrarla, se imprime el fichero con la mutación.

Posteriormente se usa un nuevo patrón que busca dos dígitos seguidos de un punto y otros dos dígitos, que nos permite obtener la frecuencia de cada uno de los genes patogénicos.

Como estas mutaciones también se encuentran en el fichero importado, a cada una de ellas le corresponde un número.

Finalmente contamos el número de mutaciones patogénicas que hay en cada fichero.

El apartado siguiente **2.5. Variables de interés** hace una recopilación de todas las variables de interés obtenidas que van a ser necesarias para la formación de los DataFrames.

En el siguiente apartado **3. Creación de DataFrames** se crean los distintos DataFrames para su posterior unión. Para crear cada uno de ellos se usa la función `pd.DataFrames(...)`. Los tres puntos suspensivos indican que ahí van las columnas que van a formar el DataFrame.

Una vez creados todos los necesarios, se usa el método `join` para su unión.

Realizamos este mismo proceso todas las veces que haga falta para obtener el DataFrame final. Finalmente conseguimos dos tablas finales, una con toda la información sobre los genes patogénicos y otra con toda la información de todos los genes.

En el apartado **4. Exportación** se indica cómo exportar las tablas a un formato Excel. Con la función `tabla-a-exportar.to_excel('nombre-tabla.xlsx')` se exporta el DataFrame a un archivo Excel sin índices, almacenando los resultados en descargas. Se exportan dos tablas, una final de los genes patogénicos y otra final con todos los genes.

Como tener una tabla con toda la información de todos los ficheros puede ser algo difícil para trabajar, también se ha tenido en cuenta el crear distintas tablas que en lugar de almacenar todos los datos, tengan solo 80 líneas (lo que correspondería con 8 chips).

Para ello las dos últimas celdas se han enfocado en obtener estos resultados.

C.4. Pruebas del sistema

El martes 13 de junio se ha presentado el trabajo frente al Comité Molecular del HUBU, ya que la colaboración para el TFG la han enmarcado dentro del proyecto de NGS de dicho Comité.

Se ha realizado una breve exposición sobre el trabajo, junto con una demostración en vivo de los informes anonimizados.

El personal ha comentado que sería un programa de gran ayuda y que les facilitaría mucho el trabajo que realizan diariamente, evitándoles el proceso de extracción manual de datos.

C.5. Instrucciones para la modificación o mejora del proyecto.

Una posible mejora puede que ser reducir el código, es decir, en lugar de implementarlo en distintas celdas, crear un código que almacene todo lo relacionado con mutaciones en una única celda, todo lo relacionado con las mutaciones patogénicas en otra celda, etc. En este caso se ha usado este método para tener el código más claro y poder explicarlo mejor.

También se ha visto que sería de gran ayuda el añadir una nueva columna a la tabla final en la que se indicara el cambio de aminoácido de cada uno de los genes mutados. En este caso sí se ha tenido en cuenta el cambio de aminoácido P.(P136L) para el gen FGFR4, ya que este gen aparece bastantes veces y se ha demostrado que no es influyente en ningún caso. Pero sería de ayuda si apareciera una nueva columna al lado de los genes para saber donde se ha producido exactamente el cambio del aminoácido y poder estudiar las consecuencias a distintos niveles.

También sería conveniente crear dos columnas nuevas. Una para determinar el número de lecturas de cada fusión que se encuentre en la tablas fusiones de genes (ARN) y otra en la que se determinen el número de copias de cada gen perteneciente a la tabla de variaciones del número de copias.

Otra posible mejora, basada en la estética del resultado, sería transformar los valores obtenidos en algunos resultados de las tablas por 0, - o la palabra vacío. La aparición de este símbolo indica que en ese apartado no se han encontrado resultados, sin embargo, hay otras formas un poco más claras o no tan cargadas de indicarlo.

Finalmente, la creación de una base de datos sería la forma más correcta de almacenar los resultados. Esto se debe a que son capaces de almacenar grandes cantidades de una forma ordenada, fácil de entender y rápida a la hora de realizar consultas. Además, son capaces de garantizar la protección de los datos. Sin embargo, en este caso es necesario que el usuario sepa hacer consultas y trabajar sobre los resultados, por lo no se descarta el uso de las tablas finales creadas como, ya que son de uso sencillo.

Para crear una base de datos en PostgreSQL usando Python se necesitará instalar `psycopg2`, una biblioteca que os facilita el trabajo a la hora de realizar este proceso.

Apéndice D

Descripción de adquisición y tratamiento de datos

El HUBU cederá distintos informes en formato .pdf generados por el software Oncomine Reporter. Dicha cesión será realizada por vía correo electrónico institucional durante el primer semestre de 2023, cumpliendo los requisitos.

Al trabajar con información sensible, se ha considerado la idea de anonimizar los datos (aunque los colaboradores de la Universidad de Burgos no tendrán acceso para asociarlo con otros datos personales de los pacientes.) El estudio se realiza siguiendo los criterios éticos internacionales recogidos en la Declaración de Helsinki.

La anonimización es un proceso en el que es imposible la vinculación de datos con la persona real a la que identifican. Un tipo es la seudonimización, cuyo objetivo se basa en limitar la trazabilidad entre el conjunto de datos tratados y la persona física a la que corresponden dichos datos, y al ser un proceso reversible, es posible identificar a la persona real. Es una de las técnicas de enmascaramiento que garantizan mayor seguridad a la hora de tratar los datos y una de las más usadas en el ámbito médico.

Sin embargo, hay otras técnicas de anonimización a parte de la desarrollada anteriormente disponibles en [\[12\]](#)

1. **Enmascaramiento de datos:** permite ocultar ciertos datos usando caracteres aleatorios en su lugar. Se sustituye la palabra por una clave.
2. **Intercambio de datos:** se basa en la variación del orden de los elementos de un conjunto ordenado, es decir, reordena valores de forma

que sigan estando presentes en el conjunto, pero no corresponden con el registro de datos originales.

3. **Datos sintéticos:** no son una técnica de anonimización real, más bien se usan para tratar con datos personales de forma que no interfiera con la ley. Un algoritmo crea un conjunto de datos sin ningún tipo de relación con los datos originales.
4. **Perturbación de datos:** agrega ruido a las bases de datos originales aportando confidencialidad a los registros. Puede sumar un valor a todos los valores numéricos del conjunto de datos que van a usarse para no trabajar directamente con los reales, pero hay que tener cuidado con las bases de datos iniciales porque si son demasiado grandes o demasiado pequeñas, es posible que los datos no se reconozcan bien y no se anonimicen.
5. **Generalización:** se basa en la eliminación de ciertos identificadores.

Tras el estudio y la comparación de los distintos tipos, se ha llegado a la conclusión de que la mejor para este proyecto es la seudonimización.

Podemos identificar cinco tipos distintos de seudonimización [13]:

1. **Cifrado con clave secreta:** se usa una clave capaz de generar un conjunto de datos que almacena dichos datos personales pero cifrados. En el momento en el que se conoce la clave de descryptación, es posible revertir el proceso.
2. **Función hash:** se basa en el uso de un algoritmo, donde partiendo de uno o varios input, genera un output alfanumérico que resume la información obtenida. Solo es posible recuperar los valores originales si se conocen los valores de entrada iniciales que forman los input.
3. **Cifrado determinista/función has con clave de borrado:** se genera un número aleatorio por cada uno de los atributos/valores originales que se quieren sustituir, borrando la tabla que los relaciona, de modo que es irreversible.
4. **Función clave almacenada:** se asocia una clave secreta a cada valor original, de forma que, conociendo las claves, es posible identificar al sujeto original.

5. **Descomposición en tokens:** se basa en reemplazar los números de interés por otros valores usando tres métodos: mecanismos de cifrado unidireccional, números de secuencias mediante funciones de índice o números generados aleatoriamente.

La idea principal era anonimizar los datos, por lo que se buscó distinta información sobre el tema para elegir el mejor tipo y saber como llevarlo a cabo. Sin embargo, al pasar el código al ordenador del hospital, no va a ser necesario anonimizar los datos. La idea de anonimizarlo era para poder trabajar con datos reales, pero como los ficheros usados para el desarrollo del proyecto no contenían información real no se ha visto necesario realizarlo.

D.1. Descripción formal de los datos.

Los ficheros iniciales son unos archivos PDF donde se encuentra la información sin seguir ningún tipo de estructura. La idea es crear un script para obtener un resultado final donde los datos estén organizados lo mejor posible. El primer resultado que obtenemos si descomentamos la función GenerarImagen son distintas imágenes en formato .png (una imagen por cada página del PDF). Esto nos permite trabajar con una mayor gama de aplicaciones y plataformas donde el archivo inicial no está permitido. También se pueden usar distintas herramientas para editar y modificar la imagen o subrayar las partes importantes. Además, ayuda a mantener el formato porque en el caso de PDF puede llegar a alterarse a la hora de visualizarlo.

Los diccionarios creados en Python partiendo de los ficheros Excel permiten asociar claves (diagnóstico y genes) a valores (número de diagnóstico y número de gen). Estos diccionarios permiten añadir, eliminar o actualizar elementos de forma muy eficiente. Además, era la mejor forma de hacerlo ya que partiendo de la clave obtenida en el texto, podíamos asignarle el valor correspondiente en cada caso.

La mayoría de las variables usadas son listas. Esto se debe a que como en un chip se procesan 8 ficheros, habrá un mínimo de ocho resultados dentro de la variable (uno por cada fichero).

La mejor forma de unir estas variables es usando DataFrames, estructuras de datos que permiten formar tablas. Siguen una estructura tabular de filas y columnas, donde cada columna representa una variable y cada fila un fichero. En cada columna pueden contener distintos tipos de datos como

fehcas, booleanos... y ofrecen una gran cantidad de operaciones y funciones para trabajar con los datos.

D.2. Descripción clínica de los datos.

Los datos obtenidos por el Software Oncomire Reporter se obtienen en carpetas comprimidas y al abrirlas, habrá ocho PDF por cada una de ellas. El nombre de los PDF está formado por el número de paciente y el número de chip. Un ejemplo de esto sería **Sample-1-v100** que es uno de los ejemplos que se va a usar para desarrollar el código. El uno indica el número de paciente (al haber ocho pacientes por carpeta, los números serán del uno al ocho) y el cien sería el número de chip. Todas las carpetas siguen el mismo formato, numeración del uno al ocho para el número de paciente y distintos número de chip.

Sin embargo, al abrir los PDF podemos ver que no todos siguen un formato estándar y esto se debe a las variaciones de cada persona. Vemos que hay tres tablas distintas, una de variantes de secuencia de ADN (que indica los cambios permanentes de la secuencia de ADN que forma un gen), fusiones de genes ARN (cambios en la secuencia de ARN) y variaciones en el número de copias (el número de copias de un segmento específico de ADN varía entre distintos genomas individuales). En función de los resultados obtenidos en estas tablas, se determinará el diagnóstico.

Sin embargo, todos ellos tienen información común como son:

- Número de historia clínica. (NHC)
- Fecha de informe.
- Número de biopsia.
- Biopsia sólida.
- Mutaciones detectadas (en las distintas tablas).
- Diagnóstico.
- Porcentaje de frecuencia alélica por cada mutación.
- Fármaco aprobado.
- Ensayos clínicos.

El número de historia clínica es un identificador único para cada persona y es asignado al paciente cuando se elabora su historia clínica manteniéndose de por vida. Este es uno de los valores de carácter sensible, por lo que habría se anonimizarlo.

La fecha de informe indica cuándo se ha llevado a cabo la extracción de la información de la muestra, es decir, cuando se ha creado el informe.

El número de biopsia es específico para cada biopsia tomada. En algunos casos es posible que el número venga acompañado de -A1 como en alguno de los ejemplos y esto se debe a que de una biopsia se pueden obtener distintos cortes y a cada uno se le asigna un sufijo distinto. De forma que varias muestras pueden tener el mismo nombre, pero distinto sufijo.

La biopsia sólida depende del valor de la tercera posición del número de biopsia, ya que este puede ser C (citología), P (punción) o B (biopsia). Nos interesa que sea B, ya que sino no entraría en nuestro algoritmo.

Las mutaciones detectadas vienen en las distintas tablas. En la tabla de variaciones de secuencias de ADN, nos interesan aquellas mutaciones que sean patogénicas, conflictivas o incluso vacías (una excepción en este caso sería el gen FGFR4 p.(P136L), que se ha determinado que no supone ningún cambio ni complicación para los distintos tipos de diagnósticos), pero en ninguno de los casos las benignas, ya que estas no supondrían ningún peligro. En el caso de fusiones de genes ARN y variaciones en el número de copias, cogemos todas. Se ha creado un diccionario para poder relacionar las mutaciones con un número específico y enriquecer los resultados, haciéndolos más simples. De esta forma las tablas contienen dos columnas, una con el nombre del gen y otra con el número que le corresponde.

El diagnóstico viene determinado ya por el software. Sin embargo, nos interesa asignar a cada diagnóstico un número específico con el fin de obtener un estudio más organizado. Para ello, se ha realizado una búsqueda de todos los diagnósticos que es capaz de determinar y junto con la investigadora, se han determinado cuáles son los de mayor interés. Al ser tantos, el hospital no usa todos los tipos y es posible que se generalice (un ejemplo es el cáncer de colon o el cáncer rectal que se diagnosticarían con el nombre de cáncer colorrectal). Para asignar a cada diagnóstico un valor numérico, se ha creado un diccionario que almacene diagnóstico-valor para poder usarlo en el código y devolver dos columnas, una con la clave (diagnóstico) y otra con el valor (número del diagnóstico).

El porcentaje de frecuencia alélica es único de cada mutación en cada paciente y sirve para indicar el número de veces que aparece el alelo, di-

vidiendo el número entre el número total de copias del gen. En el caso de tratarse de fusiones de ARN o variaciones en el número de copias no hay porcentaje de frecuencia alélica, en su lugar aparecen número de lecturas y número de copias respectivamente. Debemos tener en cuenta que en algunos casos no aparecen genes, sino fusiones. Para cada gen podemos encontrar fusiones, ya que un gen tiene varios posibles compañeros de fusión". También hay que tener en cuenta la localización de la fusión. Generalmente suelen tener únicamente un nexo de unión, pero hay casos (como por ejemplo el primer fichero) donde están fusionados por distintos localizadores, por eso es posible que aparezcan dos veces. En estos casos, hay que indicar el ID de la variante, para poder identificar la variante. Estas fusiones no vienen indicadas en la tabla de genes porque no se consideran genes, por lo que para el su obtención en el código habría que tener en cuenta esta excepción.

El fármaco aprobado indica los distintos tipos de fármacos que hay para cada diagnóstico. Podemos conocer esta información en el apartado de tratamientos relevantes en este tipo de cáncer. También ha sido interesante binarizar los resultados, siendo 0 si no hay ningún fármaco o 1 si hay uno o más.

Los ensayos clínicos aportan información sobre el número de ensayos que hay y en algunos casos, la fase en la que se encuentra cada uno. En este caso también se binarizan los resultados, siguiendo el mismo código que en el caso anterior.

Vemos que a parte de esta información, también podemos encontrar otros datos como el exón, el locus o el transcripto, pero estos valores no nos interesan para nuestro algoritmo, por eso no son tenidos en cuenta.

La idea final era ser capaz de obtener dos tablas: una con todas las variables de interés de todos los genes y otra teniendo en cuenta solo las mutaciones patogénicas.

Apéndice E

Manual de especificación de diseño

Para realizar este apartado, se ha decidido crear un diagrama de clases, es decir, una representación visual de los objetos de clases en un sistema de modelo, clasificados por tipos de clases. [14]

Dicho diagrama se encuentra disponible en la figura [E.1](#).

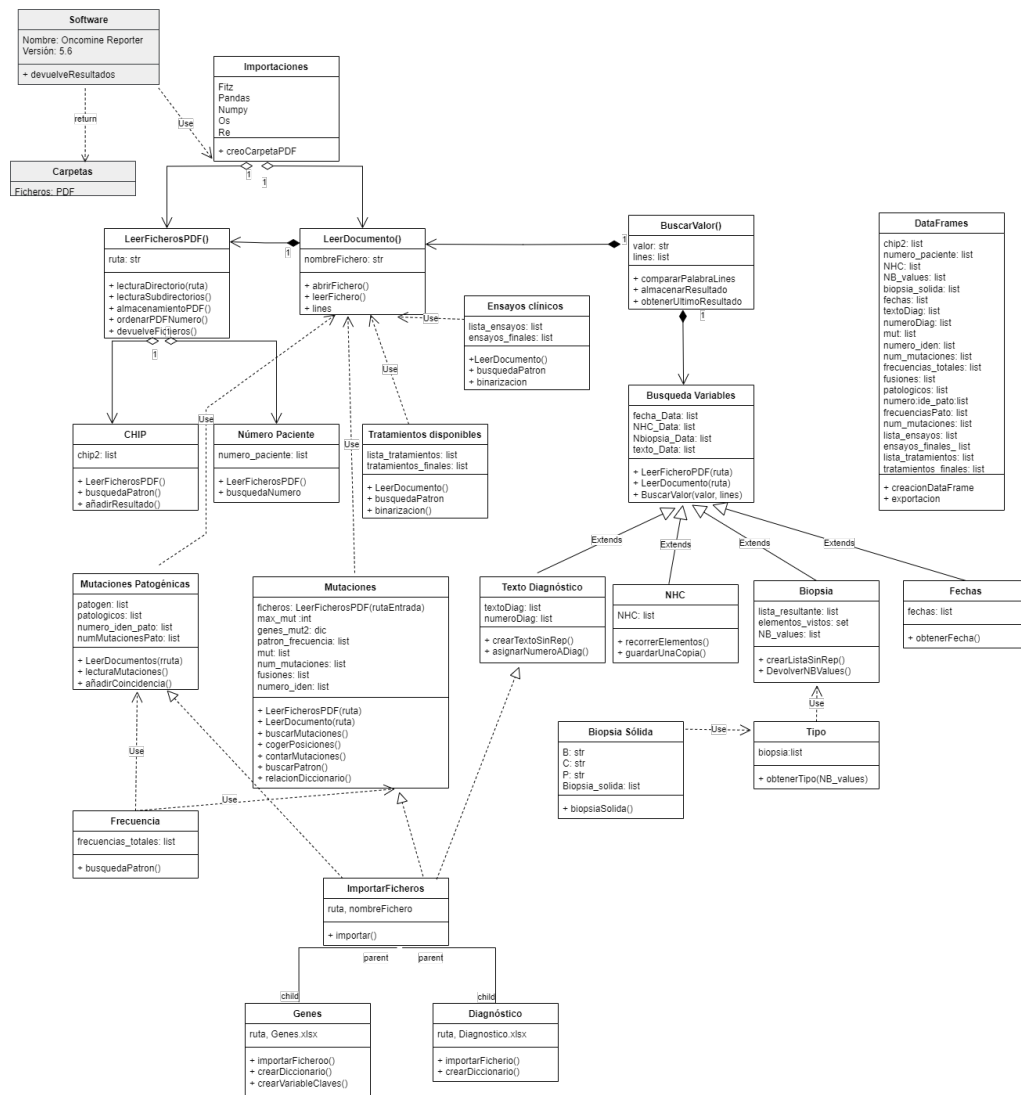


Figura E.1: Diagrama de clases, demostración del funcionamiento del código mediante un diagrama de clases de elaboración propia.

Apéndice F

Especificación de Requisitos

F.1. Introducción.

En este anexo se describirán todos los requisitos que se pretendían cubrir para poder cumplir el objetivo general.

1. **Requisito 1.** Verificar una lectura correcta de los ficheros independientemente de su carpeta.
2. **Requisito 2.** Generar las variables necesarias con los datos de interés.
3. **Requisito 3.** Correcta importación de ficheros que permita modificaciones futuras.
4. **Requisito 4.** Formación de distintos DataFrames para su posterior unión.
5. **Requisito 5.** Exportación de los resultados en la ruta especificada.
6. **Requisito 6.** Correcto funcionamiento del código.

F.2. Diagrama de casos de uso.

Para cada uno de los requisitos se ha creado un diagrama de casos de uso específico. Dichos requisitos se encuentran disponibles en las figuras **F.1**, **F.2**, **F.3**, **F.4**, **F.5** y **F.6** respectivamente.

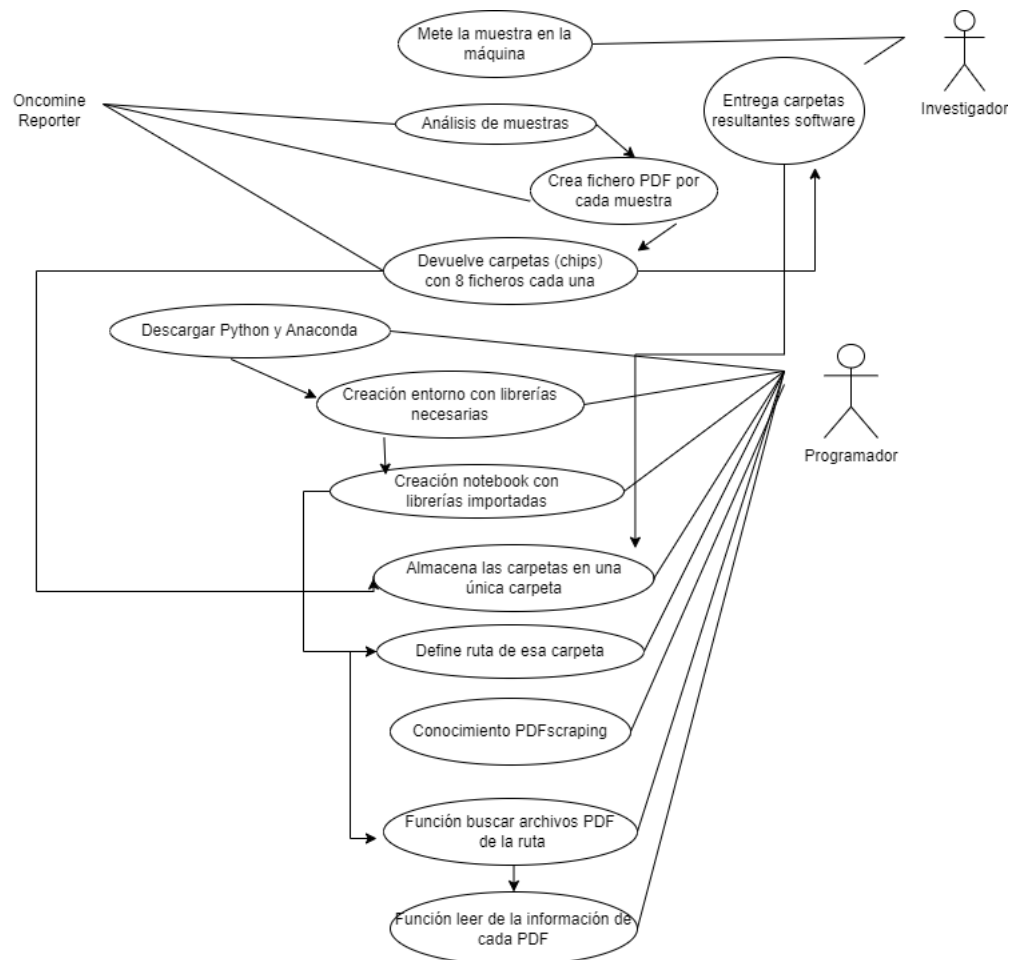


Figura F.1: Requisito 1, verificación lectura correcta.

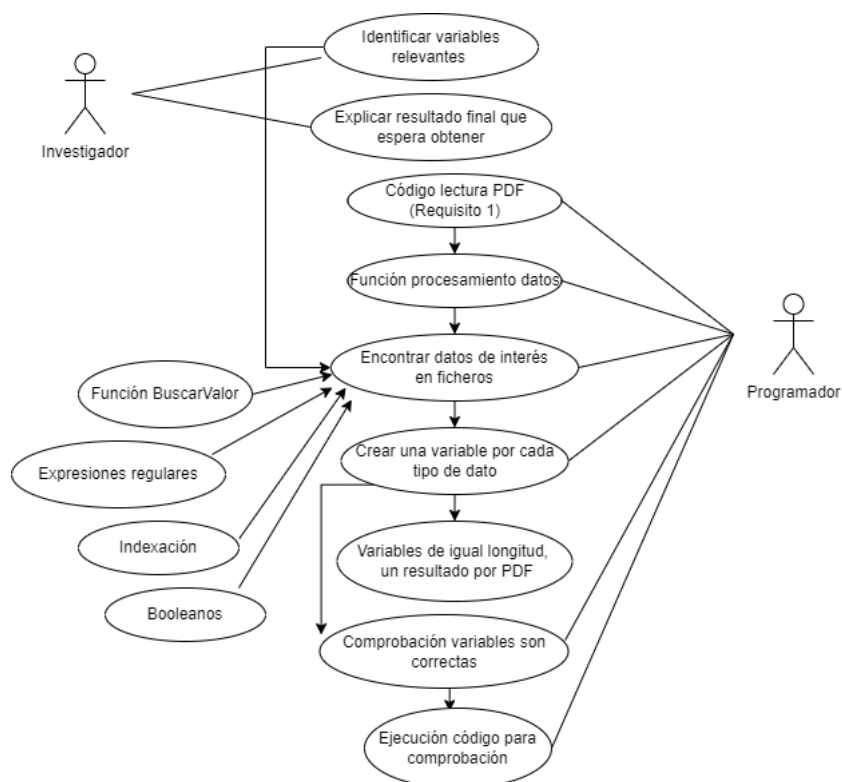


Figura F.2: Requisito 2, generación variables.

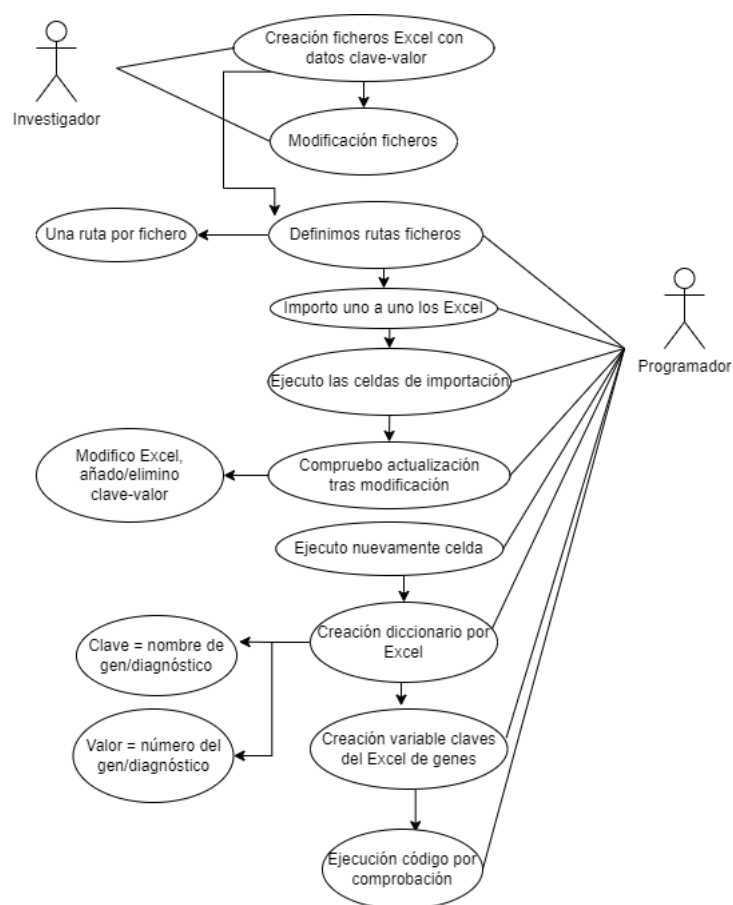


Figura F.3: Requisito 3, importación de ficheros que permitan modificaciones.

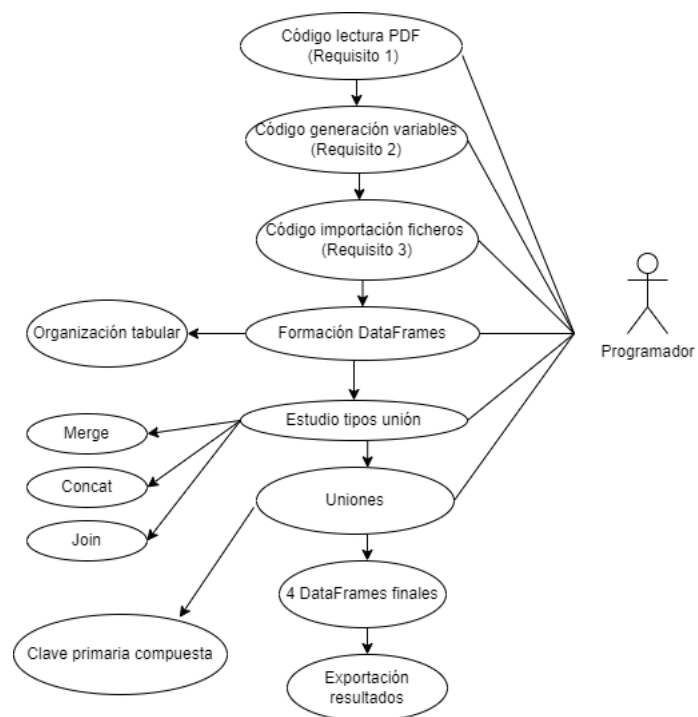


Figura F.4: Requisito 4, formación DataFrames.



Figura F.5: Requisito 5, exportación de los resultados.

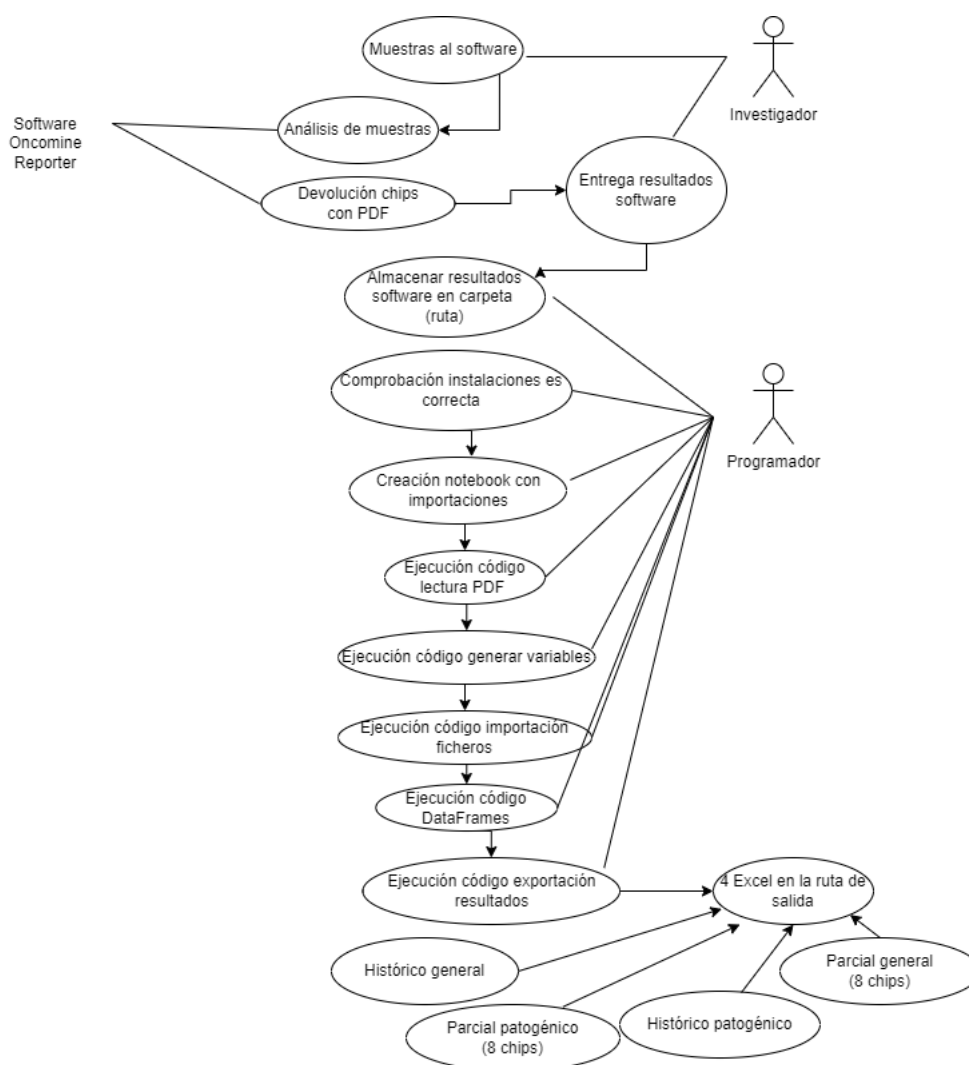


Figura F.6: Requisito 6, correcto funcionamiento del código.

F.3. Explicación casos de uso.

En esta sección vamos a explicar los requisitos del proyecto mediante casos de uso (CU), donde cada requisito vendrá explicado mediante un caso de uso.

CU-1	Verificación correcta lectura.
Versión	1.0
Autor	Lucía Vítores López
Descripción	Código que garantice la correcta lectura de los ficheros PDF obtenidos del software Oncomine Reportes. La lectura debe ser independiente de los directorios en los que se encuentre.
Precondiciones	<ol style="list-style-type: none"> 1. Conocimiento sobre PDFscraping. 2. Descarga de Python. 3. Instalación de Anaconda. 4. Creación de un entorno o enviroment. 5. Descargar los paquetes necesarios. 6. Creación de un notebook con los paquetes anteriores importados. 7. Obtención de los chips con los ficheros PDF exportados por el software. 8. Almacenamiento de dicha exportación en una carpeta, usada posteriormente como ruta de entrada para su lectura.
Acciones	<ol style="list-style-type: none"> 1. Definición de la ruta de entrada donde se almacenan los informes para su posterior lectura. 2. Creación de una función capaz de encontrar los archivos en formato PDF de la ruta definida, independientemente de los subdirectorios que esta contenga. 3. Creación de una función para la lectura de los archivos, almacenando el contenido de estos en una variable.
Postcondición	<ol style="list-style-type: none"> 1. No devuelve nada, es una función solo de lectura.
Importancia	Alta.

Tabla F.1: CU-1: Verificación lectura ficheros.

CU-2	Generación de variables.
Versión	1.0
Autor	Lucía Vítores López
Requisitos asociados	CU-1
Descripción	Tras la lectura de los ficheros, se deben extraer los datos específicos y almacenarlos en variables según su contenido (como puede ser NHC, número de biopsia, diagnóstico...).
Precondiciones	<ol style="list-style-type: none"> 1. Conocer el resultado final que pide el hospital. 2. Saber los datos que se quieren extraer. 3. Tener conocimiento acerca de expresiones regulares.
Acciones	<ol style="list-style-type: none"> 1. Una vez leídos los ficheros y almacenado su contenido, se buscan los datos de interés. 2. Como no todos los ficheros siguen la misma estructura interna, se han usado distintos métodos para la búsqueda de los datos. 3. Se crea una variable por cada tipo de dato a obtener. 4. En algunos casos puede aparecer el mismo valor repetido, por lo que se determina que solo se escoja uno. 5. No tienen por qué existir todos los valores, algunos pueden ser nulos. Pero sí tener la misma longitud. 6. Comprobamos que las variables son correctas.
Postcondición	<ol style="list-style-type: none"> 1. Por cada variable se debe devolver una lista con un valor específico del tipo de dato con el que se está trabajando. Un valor por fichero. 2. Las variables deben tener la misma longitud.
Excepciones	No todas las variables deben tener un único valor por fichero, hay excepciones donde hay más.
Importancia	Alta.

Tabla F.2: CU-2: Generación de variables.

CU-3	Importación de ficheros que permita modificaciones futuras.
Versión	1.0
Autor	Lucía Vítóres López
Requisitos asociados	CU-1, CU-2
Descripción	Como cada vez se va teniendo más información sobre los genes, se ha visto necesario usar un fichero fácil de usar y que se pueda importar cada vez que se realice una modificación.
Precondiciones	<ol style="list-style-type: none"> 1. Ficheros Excel con formato clave (gen/diagnóstico) y valor (número de gen/diagnóstico). 2. Conocer las rutas de los ficheros.
Acciones	<ol style="list-style-type: none"> 1. Para importar los ficheros hay que usar la función de python preparada para ello seguido de la ruta de cada fichero. 2. Creo un diccionario por cada fichero importado para usarlo posteriormente. 3. Para comprobar que se actualiza cada vez que lo importo, añado una clave-valor a cada uno de los ficheros. 4. Vuelvo a ejecutarlo y compruebo que se ha añadido la nueva información. 5. Creación de una variable que almacene únicamente las claves del diccionario de genes.
Postcondición	<ol style="list-style-type: none"> 1. Cada vez que se realice un cambio en el fichero Excel, se debe realizar automáticamente el mismo cambio en el diccionario. 2. Devuelve dos diccionarios, uno por fichero.
Importancia	Media.

Tabla F.3: CU-3: Importación de ficheros que permita modificaciones futuras.

CU-4	Creación de DataFrames.
Versión	1.0
Autor	Lucía Vítores López
Requisitos asociados	CU-1, CU-2, CU-3
Descripción	Una vez obtenidas las variables, se pretende almacenar estas en DataFrame.
Precondiciones	<ol style="list-style-type: none"> 1. Obtención de las variables correctas. 2. Formato previo de las tablas que se quieren crear. 3. Ventajas e inconvenientes de los distintos métodos de unión. 4. Tener determinada la clave primaria.
Acciones	<ol style="list-style-type: none"> 1. Creamos distintos DataFrames con las variables obtenidas. En este caso 5. 2. Voy uniendo los DataFrames usando la clave primaria (compuesta) para conseguir otros de mayor tamaño. 3. Finalmente se crean 4. El histórico general, histórico patogénico, parcial general y parcial patogénico.
Postcondición	<ol style="list-style-type: none"> 1. No debe haber columnas espurias en los resultados. 2. Visualizar los 4 DataFrames creados. 3. Los DataFrames serán los resultados finales.
Importancia	Alta.

Tabla F.4: CU-4: Creación de DataFrames.

CU-5	Exportación de los resultados.
Versión	1.0
Autor	Lucía Vítores López
Requisitos asociados	CU-1, CU-2, CU-3, CU-4
Descripción	Partiendo de los 4 DataFrames finales, se exportarán en formato Excel para que sea más fácil su modificación.
Precondiciones	<ol style="list-style-type: none"> 1. DataFrames creados. 2. Conocer la ruta de salida.
Acciones	<ol style="list-style-type: none"> 1. Exportación de DataFrames en formato Excel. 2. Creamos una función que permite exportar los ficheros cada 80 líneas (8 chips).
Postcondición	<ol style="list-style-type: none"> 1. Comprobar en la ruta de salida que se han creado los ficheros Excel correspondientes. 2. Se crearán más o menos ficheros parciales, todo depende el número de chips usados.
Excepciones	Puede haber ficheros parciales de 16 filas, esto se debe a que el número de chips usados es menor que 8.
Importancia	Alta.

Tabla F.5: CU-5: Creación de DataFrames.

CU-6	Exportación de los resultados.
Versión	1.0
Autor	Lucía Vítóres López
Requisitos asociados	CU-1, CU-2, CU-3, CU-4, CU-5
Descripción	Ejecución del código en conjunto para comprobar que funciona correctamente al unirlo y no solo por fragmentos.
Precondiciones	<ol style="list-style-type: none"> 1. Tener instalado Python y Anaconda. 2. Crear un entorno e instalar las librerías necesarias. 3. Tener los informes resultantes del software guardados en una carpeta. 4. Definir la ruta de los informes, las de los ficheros y la ruta final.
Acciones	<ol style="list-style-type: none"> 1. Ejecutamos una a una las celdas que forman el código. 2. Comprobamos que no devuelve ningún mensaje de error.
Postcondición	<ol style="list-style-type: none"> 1. Comprobar en la ruta de salida que se han creado los ficheros Excel correspondientes.
Excepciones	Puede haber ficheros parciales incompletos.
Importancia	Alta.

Tabla F.6: CU-6: Exportación de los resultados.

Apéndice G

Estudio experimental

La definición más precisa para estudio experimental sería la siguiente.

Tipo de estudio en el que el investigador manipula deliberadamente algún factor o circunstancia, y así puede comprobar qué efecto produce esta modificación en otro fenómeno. [15]

Como los datos con los que se ha realizado el proyecto no han sido manipulados o modificados de ninguna forma, no se llevará a cabo el desarrollo de este anexo. Tan solo se ha mejorado el resultado final con el objetivo de facilitar el estudio y manejo de los resultados, no se ha modificado ningún factor ni circunstancia ni se ha llevado a cabo un seguimiento de los pacientes.

Bibliografía

- [1] *How to Install Python*. (2023). URL: <https://www.datacamp.com/blog/how-to-install-python>.
- [2] *How to Check Python Version for Mac, Windows, and Linux*. (2023). URL: <https://blog.amphy.com/how-to-check-python-version/#:~:text=To%20check%20if%20you%20have,type%20the%20command%20python%20%E2%80%93version..>
- [3] *Properly Installing Python*. (2022). URL: <https://docs.python-guide.org/starting/installation/#installation>.
- [4] *Installing pip/setuptools/wheel with Linux Package Managers*. (2021). URL: <https://packaging.python.org/en/latest/guides/installing-using-linux-tools/>.
- [5] *How to Install Packages in Python on MacOS?* (2022). URL: <https://www.geeksforgeeks.org/how-to-install-packages-in-python-on-macos/>.
- [6] *How to Install Packages in Python on LINUX?* (2022). URL: <https://www.geeksforgeeks.org/how-to-install-packages-in-python-on-linux/>.
- [7] *os — Miscellaneous operating system interfaces*. URL: <https://docs.python.org/3/library/os.html>.
- [8] Datatrained. *Import Numpy as np(Numerical Python) | DataTrained*. URL: <https://www.datatrained.com/post/import-numpy-as-np/#:~:text=In%20order%20to%20get%20started,np%20offers%20you%20right%20away!>
- [9] *Pandas*. URL: https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html.

- [10] Data independent. *Import pandas as pd – Bring Pandas to Python*. (2022). URL: <https://dataindependent.com/pandas/import-pandas-as-pd-bring-pandas-to-python/>.
- [11] Virtual Box. URL: <https://www.virtualbox.org/>.
- [12] Pangeanic. *6 técnicas de anonimización de datos personales que debe conocer*. (2023). URL: <https://blog.pangeanic.com/es/6-tecnicas-de-anonimizacion-de-datos-personales>.
- [13] Icaria technology. *¿Qué es la seudonimización de los datos?* (2022). URL: <https://icariatechnology.com/que-es-la-seudonimizacion-de-los-datos/>.
- [14] Leticia Fonseca. *Cómo crear un diagrama de clases*. (2022). URL: <https://es.venngage.com/blog/diagrama-de-clases/#what>.
- [15] Ignacio Palacios Martínez. *Diccionario electrónico de enseñanza y aprendizaje de lenguas*. (2019). URL: <https://www.dicenlen.eu/es>.