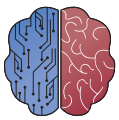
UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería de la Salud



INGENIERÍA
DE LA SALUD

TFG del Grado en Ingeniería de la Salud

Automatización de extracción de datos de informes de secuenciación masiva y análisis. Documentación Técnica

Presentado por Lucía Vítores López
en Universidad de Burgos

5 de junio de 2023

Tutor: Antonio Jesús Canepa Oneto

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción.

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

Planificación temporal.

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

- Primer *Millestone* 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...
- Segundo *Millestone* o *Desarrollo*: se tratan aspectos algo más específicos y se desarrolla el código necesario para el tratamiento de los datos.
- Tercer *Millestone* 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.

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. También hay *Issues* comenzados por el tutor con posibles mejoras o cosas a tener en cuenta para una mejora de resultado.

También se realizan reuniones con el persona del HUBU encargado de dicho proyecto, con el fin de mejorar o responder a 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 Patricia 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 *Millestone* 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 *Millestone* 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 *Millestones* 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 A1. 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 A2.

El gasto total del proyecto se puede ver en la tabla A3.

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

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. 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.

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 aquí.

Otro factor importante a tener en cuenta es el tema de las rutas. Las rutas especificadas en este script son las usadas en el ordenador donde se ha desarrollado, pero al ejecutar este código en cualquier otro, al no tener las mismas carpetas ni el mismo usuario, no funcionará. Hay que tener en cuenta que la carpeta donde se guardan los ficheros no tiene por qué ser la misma donde se encuentran los documentos Excel ni la misma donde se almacenarán los resultados una vez ejecutado el código. Por ello es interesante crear carpetas de la forma más ordenada posible para saber dónde se encuentra cada fichero.

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. [9]

Si se está trabajando con Linux, se debe abrir el terminal y ejecutar **python –version**. [8]

En caso de no tenerlo instalado, es necesario instalar la última versión. [7]

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 terminar 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 aquí.

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. [6]

Para Linux se puede entrar en esta dirección y escoger el paquete que se quiere instalar, se copia el comando de instalación y se ejecuta en la terminal. [5]

Otra posible opción para instalar estos paquetes es desde Anaconda, escogiendo el entorno de trabajamos 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.

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

- **Import os**: [1] 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, 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... Es muy rápida y eficiente. [10]. En este código se ha usado principalmente para la exportación de las tablas finales a un formato Excel, con la finalidad de crear distintos ficheros de salida, cada uno con un número aproximado de 80 pacientes.

- **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**: [2] 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. 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**: que permite escoger las hojas que se quieren usar y que en nuestro dejamos lo que viene por defecto (cero) porque nos permite obtener todas, **usecols**: que nos permite escoger con qué columnas queremos trabajar (como por ejemplo .ᴬ:E", que en este caso indica que queremos trabajar con las columnas que van desde la A hasta la E ambas incluidas) o **nrows** para determinar el número de columnas con el 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 DataFraes 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:[11]

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.

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

La función 'LeerFicherosPDF' que se encuentra dentro del apartado *2.1. Definición de funciones* del fichero CodigoPython.ipynb contiene el código necesario para crear una lista vacía llamada ficheros que almacene los ficheros de la ruta que se pide como argumento. Se usa la función **os.walk(ruta)** para iterar sobre los archivos y directorios de la ruta. Devuelve una tupla con la ruta actual (raiz), una lista de nombres de subdirectorios (directorios) y una lista con los nombres de los archivos (archivos). Se itera sobre cada uno de los archivos encontrados y para cada uno se comprueba la extensión usando **endswith(.'pdf)**. Con el comando **.append(os.path.join(raiz, archivo))** si es PDF, se añade a la lista ficheros usando el método **os.path.join()** para combinar la ruta con el nombre del archivo. Finalmente obtenemos una lista con todos los archivos de la ruta definida.

Con la función 'LeerDocumento' con argumento nombreFichero se lee el contenido de cada argumento que se le pasa, usando la libreria PyMuPDF (fitz). 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.

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. Recorre todas las posiciones comparando cada palabra con la palabra de interés. En caso de encontrar la palabra pone un 1 en una nueva variable llamada valores, en caso de no encontrar pone un 0. Recorremos cada uno de los elementos de la variable anterior (formada por unos y ceros) y se hace un for para recorrer la lista y extraer el valor que hay tras la palabra de interés. Strip permite eliminar los espacios en blanco.

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 porporcionados 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. Para ello se usa la función **pd.read_excel** seguido de la ruta 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 definen dos listas vacías llamadas NHC_Data_order y NHC_Data_val y se itera sobre los elementos en NHC_Data. Dentro del bucle, se vuelve a itera sobre los valores en cada elemento i. Si el valor no está en NHC_Data_val, lo agrega a ambas listas y se le asigna el contenido de NHC_Data_order a NHC_Data_val.

NHC_Data_order se escribe sobre NHC_Data_val manteniendo el mismo orden.

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.join()** para combinar la ruta con cada uno de los ficheros, sobre esto realizamos **os.path.normpath()** para normalizar la ruta y finalmente **os.path.isfile()** para ver si es un archivo existente dentro de la ruta y en la segunda nos aseguramos de que la extensión del fichero sea .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 nievas 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 aplicacmos la función **en()** 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 los 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 lines. 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 modificar 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 devovlerá 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 posterir 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 ente 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([çlaves"])**. 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 ínidices, 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 lineas (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. Como los hemos exportado, si vamos a la ruta elegida, podemos ver estos mismos DataFrames en formatos Excel, listos para trabajar sobre ellos.

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 progra-

ma 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 ha 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. [3]

Para usar la máquina virtual es tan fácil como descargar el fichero y entrar en VirtualBox e ir a archivo -> Importar servicio virtualizado. Se escoge el fichero .ova y se da a siguiente. Una vez que se abra, se puede ver que hay una carpeta que contiene los ficheros con los que se trabaja, un fichero llamado PDFscraping??? que es donde se encuentra el código y una carpeta de resultados vacía. Cuando ejecutamos el fichero, solo es necesario pinchar sobre este y vemos que la carpeta de resultados se llena.

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...
El proyecto está disponible en el repositorio GitHub pinchando aquí.

C.2. Estructura de directorios

Entre los ficheros de entrega constarán:

1. **Código Python**: el código en Python en un fichero 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. **FuncionesGit.txt**: fichero que explica las principales funciones usadas en el funcionamiento interno de GitHub.
3. **README.md**: fichero que resume la información más relevante del proyecto, incluyendo nombre del tutor, resumen del proyecto, universidad...
4. **MemoriaAnexos.pdf**: es un fichero PDF que almacena toda la información correspondiente a los anexos del proyecto.
5. **MemoriaTEX.pdf**: otro fichero en formato PDF que recoge toda la información sobre los objetivos, introducción, metodología, conclusiones y líneas futuras para el correcto desarrollo del proyecto.

6. **INPUT**: es una carpeta que contiene cuatro subcarpetas. La primera de ellas denominada *Datos* que contiene los ficheros Excel para la importación de sus datos, la segunda carpeta *Imágenes* contiene una imagen sobre los distintos tipos de uniones que hay dentro de la función join, la tercera *Informes* contiene los informes aportados por el Hospital Universitario de Burgos (ya anonimizados) para tener una idea sobre cómo se organizan y porder trabajar sobre ellos, y finalmente la última carpeta *Resultados* que es donde se almacenan los resultados obtenidos al final del código.

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 ejecuatado la celda sin ningún problema, podemos ejecutar el código que viene a continuación. La función 'LeerFicherosPDF' recorre todos los archivos y directorios de la ruta especificada, comprobando que los archivos tengan una extensión .pdf.

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* recorremos 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.

En el apartado *2.3.2. NHC* se definen dos listas vacías llamadas NHC_Data_order y NHC_Data_val y se itera sobre los elementos en NHC_Data. Dentro del bucle, se vuelve a itera sobre los valores en cada elemento i. Si el valor no está en NHC_Data_val, lo agrega a ambas listas y se le asigna el contenido de NHC_Data_order a NHC_Data_val.

NHC_Data_order se escribe sobre NHC_Data_val manteniendo el mismo orden.

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.

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.join()** para combinar la ruta con cada uno de los ficheros, sobre esto realizamos **os.path.normpath()** para normalizar la ruta y finalmente **os.path.isfile()** para ver si es un archivo existente dentro de la ruta y en la segunda nos aseguramos de que la extensión del fichero sea .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 nievas 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 aplicacmos la función **en()** 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 los 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 lines. 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 modificar 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()**.

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 posterir 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 ente 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([çlaves"])**. 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 ínidices,

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 lineas (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. Como los hemos exportado, si vamos a la ruta elegida, podemos ver estos mismos DataFrames en formatos Excel, listos para trabajar sobre ellos.

Una vez tenemos la ruta sobre la que se encuentran los ficheros, se realizan distintos métodos para obtener las variables que interesan para formar los DataFrames finales.

C.4. Pruebas del sistema

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

Una posible mejora puede que se reduzca 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... En este caso se ha usado este método para tener el código más claro y poder explicarlo mejor.

Sin embargo, una de las posibles mejoras que se ha visto ha sido añadir una nueva columna a la tabla final en la que se indicara el cambio de aminoácido de cada uno de lo 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 variable al lado de los genes para saber donde se ha producido exactamente el cambio del aminoácido para poder estudiar las consecuencias a distintos niveles.

También sería conveniente crear dos columnas nuevas. Una para determinar el número de lecturas 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 rápido a la hora de realizar consultas. Además. son capaces de garantizar la protección de los datos. 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 [13]

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 [14]:

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 desencriptación, es posible revertir el proceso.

2. Función hash: se basa en el uso de un algoritmo, donde partiendo de uno o varios inputs, 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 inputs.

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 la hora de trabajar con datos reales, pero como los ficheros usados para el desarrollo del proyecto no contenían información real y el código se ejecutará directamente en el ordenador con los ficheros reales, 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 deforma 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 varaible 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 çompañeros 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 fusione 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

E.1. Planos

E.2. Diseño arquitectónico

Si procede.

Diagramas de clases, diagramas de despliegue ...

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.

F.3. Explicación casos de uso.

Se puede describir mediante el uso de tablas o mediante lenguaje natural.

Una muestra de cómo podría ser una tabla de casos de uso:

F.4. Prototipos de interfaz o interacción con el proyecto.

CU-1	**Ejemplo de caso de uso**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-xx, RF-xx
Descripción	La descripción del CU
Precondición	Precondiciones (podría haber más de una)
Acciones	1. Pasos del CU 2. Pasos del CU (añadir tantos como sean necesarios)
Postcondición	Postcondiciones (podría haber más de una)
Excepciones	Excepciones
Importancia	Alta o Media o Baja...

Tabla F.1: CU-1 Nombre del caso de uso.

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.* [12]

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] os — miscellaneous operating system interfaces.

[2] pandas.

[3] Virtual box.

[4] Installing pip/setuptools/wheel with linux package managers, 2021.

[5] How to install packages in python on linuz?, 2022.

[6] How to install packages in python on macos?, 2022.

[7] Properly installing python, 2022.

[8] How to check python version for mac, windows, and linux, 2023.

[9] How to install python, 2023.

[10] Datatrained. Import numpy as np(numerical python) | datatrained.

[11] Data independent. Import pandas as pd – bring pandas to python, 2022.

[12] Ignacio (dir.) Palacios Martínez. *Diccionario electrónico de enseñanza y aprendizaje de lenguas.* 2019.

[13] Pangeanic. 6 técnicas de anonimización de datos personales que debe conocer, 2023.

[14] Icaria technology. ¿qué es la seudonimización de los datos?, 2022.