Desarrollo de una interfaz para conectar con la base de datos de TCGA y favorecer la descarga de datos

Adolfo Casaña Carabot

MU Bioinf. y Bioest.
Desarrollo de programas y Aplicaciones

Nombre Tutor/a de TF
Jaime Sastre Tomas
Profesor/a responsable de la asignatura
Antoni Pérez Navarro

20/06/2023

Universitat Oberta de Catalunya

UNIVERSITAT DE BARCELONA

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de una interfaz para conectar con la base de datos de TCGA y favorecer la descarga de datos*
Nombre del autor:	*Adolfo Casaña Carabot*
Nombre del consultor/a:	*Jaime Sastre Tomas*
Nombre del PRA:	*Antoni Pérez Navarro*
Fecha de entrega (mm/aaaa):	*06/2023*
Titulación o programa:	*Máster Universitario en Bioinformática i Bioestadística*
Área del Trabajo Final:	*Desarrollo de programas y aplicaciones*
Idioma del trabajo:	*Castellano*
Palabras clave	*RStudio, Bioinformatics, TCGA, Database, Development, Package*

Resumen del Trabajo

El acceso a datos y modelos es esencial para la investigación científica en la actualidad, especialmente en el campo de la bioinformática. Sin embargo, los datos biológicos pueden ser complejos, difíciles de manejar y pueden estar disponibles en formatos muy diversos, lo que puede impedir el acceso a ellos y limitar la capacidad de los investigadores para interpretarlos y obtener nuevos conocimientos.

El objetivo inicial de este trabajo era el desarrollo de un módulo para el lenguaje de programación R que permitiese la interacción con la base de datos del GDC a través de las APIS publicadas por esta.

A raíz de las investigaciones iniciales, se descubrió que ya existía un componente que proporcionaba la interacción que se estaba desarrollando, por lo que se reorientaron los objetivos del mismo a la investigación y documentación de la estructura de dicha base de datos, del componente existente y a la creación de un tutorial de uso del mismo.

En el presente trabajo proporciona un resumen de la estructura de datos de la base de datos del GDC y se complementa con un tutorial interactivo desarrollado sobre R y Shiny que tiene como objetivo dotar a los usuarios de

los conocimientos necesarios para poder utilizar el módulo GenomicDataCommons en sus futuras investigaciones.

La herramienta se centra en las funciones más importantes de la librería y está complementada con ejemplos prácticos y un Dashboard que permite tener una primera impresión de los datos almacenados en la base de datos.

Abstract

Access to data and models is essential for scientific research today, especially in the field of bioinformatics. However, biological data can be complex, unwieldy, and available in many different formats, which can prevent access to it and limit researchers' ability to interpret it and get new insights.

The initial objective of this work was the development of a module for the R programming language that would allow interaction with the GDC database through the APIs published by it.

As a result of the initial investigations, it was discovered that there was already a component that provided the interaction that was being developed, so the objectives of the same were reoriented to the investigation and documentation of the structure of the GDC database, of the existing component and to the creation of a tutorial of its use.

This paper provides a summary of the data structure of the GDC database and is complemented by an interactive tutorial developed over R and Shiny that aims to provide users with the necessary knowledge to be able to use the GenomicDataCommons module in your future research.

The tool focuses on the most important functions of the library and it is complemented by some practical examples and a Dashboard that allows you to have a first impression of the data stored in the database.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

El acceso a datos y modelos es esencial para la investigación científica en la actualidad, especialmente en el campo de la bioinformática. Sin embargo, los datos biológicos son complejos, difíciles de manejar y pueden estar disponibles en formatos muy diversos, lo que puede impedir el acceso a ellos y limitar la capacidad de los investigadores para interpretarlos y obtener nuevos conocimientos.

Una de las ramas de la bioinformática es la de ayudar a simplificar el acceso a estos datos y modelos a través del desarrollo de herramientas y sistemas que simplifican el procesamiento y análisis de los mismos. Esto no solo mejora la eficiencia y la calidad de la investigación, sino que también fomenta la colaboración y el intercambio de datos entre científicos.

Por lo tanto, existe una necesidad de desarrollar soluciones bioinformáticas que simplifiquen el acceso a los modelos de datos para los científicos, lo que permite una mayor utilización y explotación de los datos existentes.

Una de las bases de datos con información biológica relevante es la que alberga "NCI Genomic Data Commons (GDC)" (https://docs.gdc.cancer.gov/), una organización que recibe, procesa y distribuye datos genómicos, clínicos y de biomuestras de programas de investigación del cáncer a través de su propia página web (https://docs.gdc.cancer.gov/) y mediante distintas APIs que ésta tiene publicadas.

Conjuntamente, una de las herramientas más extendidas universalmente en el ámbito del manejo, análisis de datos y elaboración de documentación científica es el lenguaje de programación R, para el que existen una gran cantidad de módulos desarrollados que amplían enormemente su funcionalidad.

El propósito de este trabajo es la investigación y documentación de la estructura de la base de datos proporcionada por GDC, la investigación y documentación del módulo de Bioconductor (https://www.bioconductor.org) llamado “GenomicDataCommons” (https://github.com/Bioconductor/GenomicDataCommons) para la consulta y

acceso a la información de dicha base de datos y la creación de una aplicación con R Shinny que sirva como tutorial de uso del módulo informado y proporcione un cuadro de mandos interactivo para la consulta y visualización de los datos de la Base de Datos.

1.2. Objetivos del Trabajo

1.2.1. Objetivos Generales

- Proporcionar a la comunidad bioinformática una aplicación para simplificar el acceso a la información proporcionada por TCGA.
- Proporcionar a la comunidad bioinformática las herramientas para visualizar de una forma amigable a la información proporcionada por TCGA.

1.2.2. Objetivos Específicos

- Estudio y documentación de la información disponible en la base de datos del TCGA.
- Estudio y documentación del módulo GenomicDataCommons de Bioconductor para el acceso a la base de datos del TCGA.
- Desarrollo y publicación en GIT de una aplicación en R Shiny con ejemplos de acceso y utilización del módulo GenomicDataCommons de Bioconductor.
- Desarrollo de un cuadro de mando en R Shiny para mostrar parte de los datos disponibles utilizando el módulo GenomicDataCommons de Bioconductor.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

La competencia de compromiso ético y global (CCEG) está definida a nivel de Máster como "*Actuar de manera honesta, ética, sostenible, socialmente responsable y respetuosa con los derechos humanos y la diversidad, tanto en la*

práctica académica como en la profesional, y diseñar soluciones para mejorar estas prácticas."

Y este sentido aborda tres grandes dimensiones:

- Sostenibilidad
- Comportamiento ético y responsabilidad social (RS)
- Diversidad (género, entre otros) y derechos humanos

Estas dimensiones están alineadas con los objetivos de desarrollo sostenible de la ONU (ONU, s.f.), con los que la UOC comprometida (UOC, Impacto global agenda 2030, s.f.)

Respecto a la sostenibilidad, el presente trabajo no puede aportar prácticamente nada ni a favor ni en contra, al tratarse de la elaboración de un componente de software.

Respecto al comportamiento ético y responsabilidad social, este trabajo puede llegar a tener un impacto significativo. El propósito final de este trabajo es el de proporcionar un acceso sencillo y universal a la información de la base de datos detallada anteriormente desde una herramienta de análisis conocida como es el lenguaje R, añadiéndole un cuadro de mandos para permitir su consulta e interpretación a perfiles con niveles técnicos inferiores, gracias a las funcionalidades especificadas en los objetivos del presente documento.

En relación al apartado de diversidad, el presente trabajo no puede aportar prácticamente nada a favor, más allá de asegurar el acceso a toda la información disponible de una forma totalmente transparente y sin ningún tipo de sesgo.

1.4. Enfoque y método seguido

Actualmente existen una gran cantidad de módulos para el lenguaje de programación R, pero ninguno de ellos proporciona acceso al base de datos de TCGA, es por ello que la estrategia escogida para el desarrollo del trabajo es la crear un nuevo paquete desde cero que proporcione esa la funcionalidad.

El paquete se creará una vez realizado el estudio de la información proporcionado por las APIs de TGA, con las siguientes premisas:

- Se aplicarán las buenas prácticas de desarrollo adquiridas a lo largo de los estudios junto a las que se puedan extraer de la bibliografía.
- Se desarrollará el código utilizando GIT como repositorio y control de fuentes.
- Tanto el código como su documentación, se desarrollarán en inglés para ampliar el alcance del mismo.
- El desarrollo seguirá un proceso de desarrollo iterativo incremental de acuerdo con el siguiente diagrama:

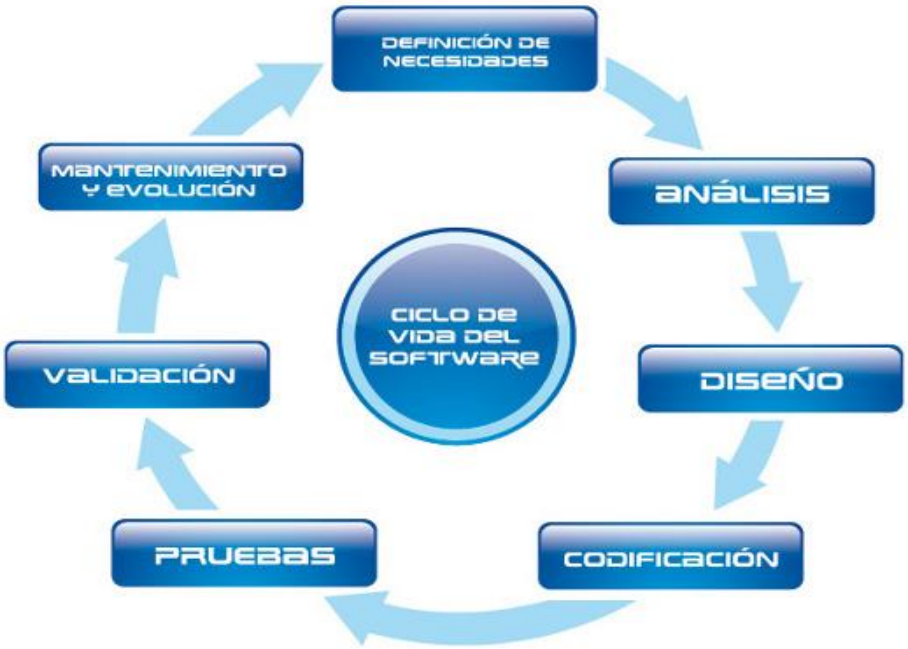

Ilustración 1 - Desarrollo iterativo incremental

Una vez desarrollado el paquete anterior, utilizando el mismo como origen de datos, se creará un cuadro de mandos desarrollado con R-Shinny, también en inglés, para mostrar las capacidades del mismo. La finalidad del cuadro de mandos será la de mostrar de la forma más sencilla posible la información obtenida de la base de datos indicada, permitiendo realizar consultas y filtros

sobre la misma por razón de sexo, edad y cualquier otro indicador que permita mostrar la información en formato gráfico o estadístico.

1.5. Planificación del Trabajo

1.5.1. Tareas

Con el fin de llevar a buen término los objetivos definidos anteriormente, se ha dividido el alcance del proyecto en las siguientes tareas que se han ordenado secuencialmente:

- Análisis de la documentación proporcionada por TCGA relativa a sus servicios web.
- Análisis de la capacidad de conexión con servicios web por parte de R.
- Documentación de la información proporcionada por la librería de TCGA.
- Desarrollo en R del paquete de conexión para R.
    - Primera versión del código (implementación de las funcionalidades básica y mínimas para el acceso a la base de datos)
    - Testeo unitario. Punto crítico detallado en el análisis de riesgos.
    - Creación del repositorio en GIT.
    - Desarrollo del resto de la integración.
    - Testeos unitarios del módulo.
- Documentación del código.
- Desarrollo del cuadro de mandos con R-Shiny.
- Finalización del documento entregable.
- Preparación de la presentación y defensa.

1.5.2. Calendario

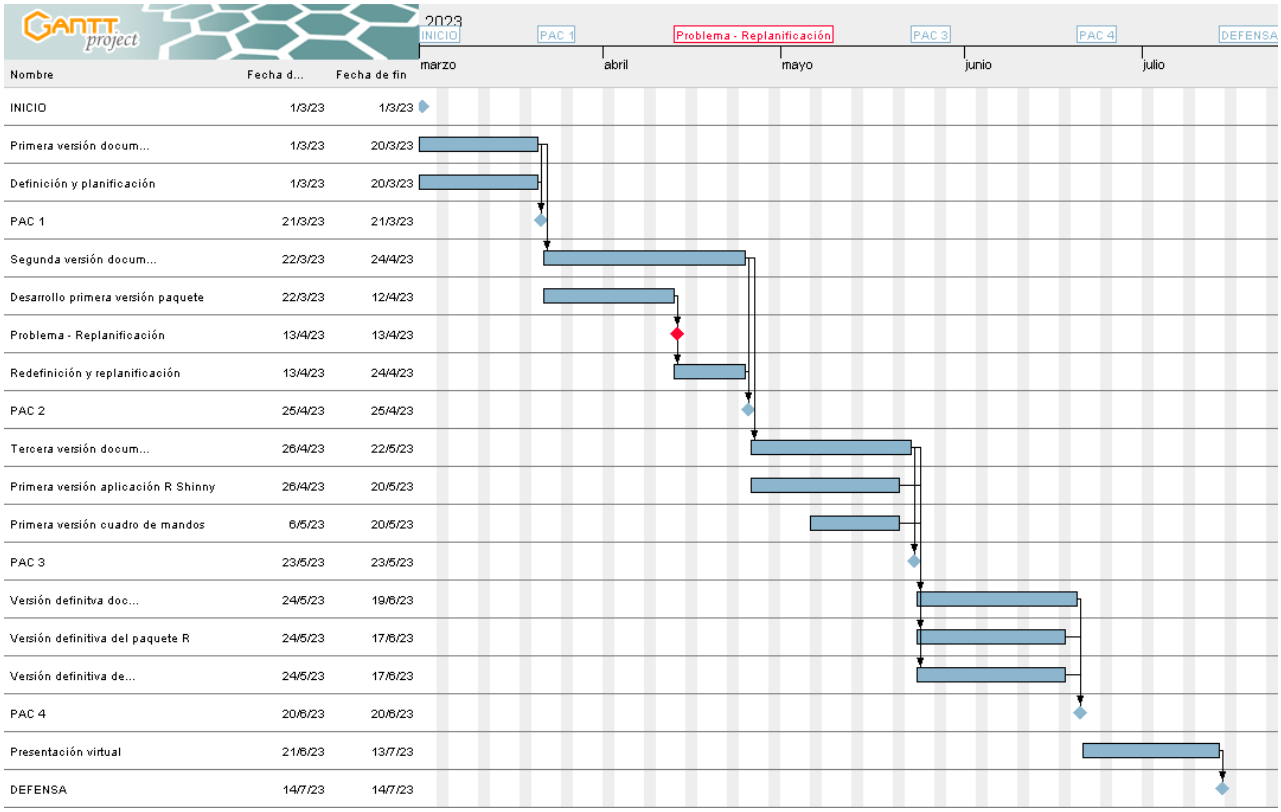

Ilustración 2 - Calendario en formato diagrama de Gantt

1.5.3. Hitos

Como se puede apreciar en el diagrama anterior, los hitos están alineados con las entregas definidas en el plan docente de la asignatura, los cuales se resumen en 4 Pruebas de Evaluación Continuada (PEC) y la defensa final del trabajo:

1- **PEC 1**
   - Primera versión del documento entregable.
   - Definición de la planificación del trabajo.

2- **PEC 2**
   - Segunda versión del documento entregable.
   - Redefinición y replanificación del trabajo.
   - Documentación de la base de datos del GDC
   - Documentación del módulo GenomicDataCommon de Bioconductor.
   - Pruebas iniciales de acceso a datos y R Shiny.

3- **PEC 3**
    - Tercera versión del documento entregable.
    - Entrega de la primera versión de la aplicación con ejemplos de acceso a la base de datos.
    - Entrega de la versión preliminar del cuadro de mandos.
4- **PEC 4**
    - Versión definitiva del documento entregable.
    - Entrega de las versiones definitivas de la aplicación.
    - Entrega de la versión definitiva del cuadro de mandos.
5- **PEC 5**
    - Presentación virtual

1.5.4. Análisis de riesgos

En todos los procesos de desarrollo de nuevas funcionalidades de software, existen ciertos riesgos que pueden surgir a lo largo del proceso de creación debido a multitud de factores, como pueden ser:

- Problemas de desarrollo relacionados con las capacidades del Entorno el lenguaje seleccionado.
- Problemas debidos a dependencias o indisponibilidades de terceros.
- Problemas de integración.

En base a estos posibles riesgos, se han planteado las siguientes acciones para paliar estos posibles riesgos:

- La primera versión del código va a estar enfocada en cubrir todas las necesidades técnicas que puedan ser necesarias, permitiéndonos encontrar posibles limitaciones y aplicar, en el caso de ser necesario las medidas necesarias para subsanarlas, paliarlas, o en el peor de los casos, redefinir todo el proyecto.
- Se va a trabajar en el proyecto poniendo especial foco en los objetivos definidos, pero dotándolo de cierto nivel de flexibilidad en el alcance del desarrollo.

1.6. Resultados esperados

Al finalizar el trabajo, esperamos obtener los siguientes elementos:

- Un plan de trabajo que se ha seguido sin impedimentos.
- La memoria del Trabajo Final de Máster.
- La presentación para la defensa de Trabajo.
- El paquete de R y su documentación adjunta.

1.7. Breve sumario de productos obtenidos

- **Plan de trabajo:** Cronograma con las tareas de cada una de las fases del trabao.
- **Memoria:** Documento que se ha ido complementando a lo largo de todo el proyecto, actualizándose de forma constante en cada una de sus etapas.
- **Aplicación web**
    - Se encuentra publicada y disponible para cualquier usuario en la siguiente dirección: https://adocasuoc.shinyapps.io/GDC_Tutorial/
    - El código Fuente de la misma se encuentra disponible en el siguiente repositorio de GITHub: https://github.com/AdoCas/GDC_Tutorial
- **Presentación:** Documento con diapositivas en el que se resume el trabajo llevado a cabo a lo largo del proyecto.
- **Vídeo de Presentación:** Archivo de vídeo con la presentación del trabajo llevado a cabo a lo largo del proyecto y los resultados obtenidos.

1.8. Breve descripción de los otros capítulos de la memoria

- **Capítulo 2 - Estado del arte:** En este capítulo se realiza la investigación del estado del arte actual de la consulta de datos y el acceso a la base de datos del GDC.
- **Capítulo 3 - Estructura de la base de datos del GDC:** En este capítulo se realiza el estudio y la documentación del modelo de datos y las distintas entidades que contiene la base de datos del GDC.

- **Capítulo 4 - Módulo GenomicDataCommons:** En este capítulo se realiza el estudio y la documentación preliminar de las funciones más importantes del módulo GenomicDataCommons del paquete Bioconductor.
- **Capítulo 5 - Materiales y métodos:** En este capítulo se documentan los recursos que se van a utilizar para llevar a buen término la aplicación Shiny que se va a desarrollar.
- **Capítulo 6 - Aplicación Shiny obtenida:** En este capítulo se describen los elementos principales de la aplicación Shiny desarrollada y el cuadro de mandos complementario.

2. Estado del arte

2.1. Importancia de la consulta de datos

La consulta de datos es un proceso crítico en la investigación biomédica que permite a los investigadores extraer información valiosa de grandes conjuntos de datos. El uso de bases de datos públicas y privadas en este tipo de investigaciones se ha vuelto cada vez más común en las últimas décadas debido a la gran cantidad y variedad de datos disponibles. La consulta de datos también comprende la integración de datos de múltiples fuentes, lo que nos permite incrementar la precisión de los distintos análisis y sus posteriores conclusiones.

2.2. Utilización de R, RStudio y el paquete Bioconductor

R es un lenguaje de programación de código abierto y RStudio es un entorno de desarrollo integrado principalmente creado para R que ofrece una gran variedad de herramientas para el análisis de datos. El lenguaje R es ampliamente utilizado en la investigación biomédica debido a su capacidad para manejar grandes conjuntos de datos y su amplia gama de paquetes y bibliotecas disponibles para aumentar su funcionalidad.

Dentro de esos paquetes, podríamos destacar “Bioconductor”, un proyecto de software libre y abierto que proporciona herramientas para el análisis y la visualización de datos de biología molecular y genómica. Bioconductor incluye más de 2500 paquetes que proporcionan una amplia variedad de herramientas para el análisis de datos de secuenciación, expresión génica, epigénica, proteómica, metabólica, microarrays y muchos otros.

La combinación de R, RStudio y Bioconductor proporciona una solución muy completa y extremadamente potente para la investigación biomédica y la consulta de datos.

2.3. Base de datos del GDC

El Genomic Data Commons (GDC) es una plataforma de intercambio de datos propiedad del Instituto Nacional del Cáncer (NCI), el cual es un organismo gubernamental de Estados Unidos que forma parte de los Institutos Nacionales de la Salud (NIH). El NCI se dedica a la investigación, prevención y tratamiento del cáncer y está considerado como el principal centro de investigación del cáncer del mundo.

El Genomic Data Commons (GDC) es una red de conocimiento expandible que apoya la importación y estandarización de datos genómicos y clínicos de los programas de investigación del cáncer. Dentro de la misma, nos vamos a centrar en la funcionalidad que proporciona acceso a datos de secuenciación y de análisis molecular de más de 30 tipos de tumores y con más de 14,5 petabytes de datos. La plataforma alberga datos de proyectos financiados por el Instituto Nacional del Cáncer de Estados Unidos y otros proyectos relacionados con la misma enfermedad. GDC también proporciona a través de su página web una API pública para el acceso a los datos y una serie de herramientas para la visualización de los mismos.

La base de datos del GDC es una fuente muy valiosa de datos para la investigación en el campo del cáncer y la bioinformática en general. Proporciona acceso a grandes conjuntos de datos de secuenciación y análisis molecular que se pueden utilizar para la identificación de mutaciones somáticas, la identificación de subtipos de tumores y la identificación de dianas terapéuticas potenciales, entre otras aplicaciones.

2.4. La funcionalidad existente en el desarrollo inicial del prototipo

Durante el desarrollo inicial del prototipo para el módulo de consulta de la base de datos de GDC para RStudio, se descubrió la existencia de un módulo de Bioconductor llamado “GenimocDataCommons” que proporciona exactamente la funcionalidad que se estaba empezando a implementar. Esta funcionalidad

permite el acceso nativo desde R a los datos contenidos en la base de datos del GDC. Dado que esta funcionalidad ya existe, nos hemos visto obligados a enfocar de nuevo el objetivo principal del trabajo.

2.5. Nuevo enfoque del objetivo del prototipo

Como se ha comentado en el punto anterior, al existir la funcionalidad indicada, el objetivo del trabajo se ha reorientado a la realización de una breve documentación de la base de datos del GDC, una breve documentación e investigación de las funcionalidades del paquete encontrado y el desarrollo de una aplicación con una interfaz de usuario amigable y fácil de usar.

Esta aplicación ejemplificará el uso de del paquete y permitirá a los investigadores navegar y descargar datos de manera sencilla y rápida, tendrá, además, la capacidad de filtrar y buscar conjuntos de datos específicos.

2.5 Conclusiones

La consulta de datos es una tarea esencial en la investigación biomédica y la utilización de herramientas como R, RStudio y Bioconductor puede mejorar significativamente la eficacia y precisión de los análisis de datos.

Es importante tener en cuenta que ya existen soluciones y funcionalidades similares en el mercado, por lo que es importante investigar y evaluar antes de comenzar cualquier proyecto de desarrollo de software en el campo de la bioinformática.

El reenfoque del objetivo del prototipo permitirá el desarrollo de una herramienta útil y eficaz para los investigadores en el acceso y descarga de datos biomédicos disponibles públicamente.

3. Estructura de la base de datos del GDC

3.1. Gráfico de estructura

Como se ha comentado previamente, la base de datos del GDC contiene y proporciona una gran cantidad de información, por lo que el modelo de datos que se puede observar en la siguiente imagen es considerablemente complejo:

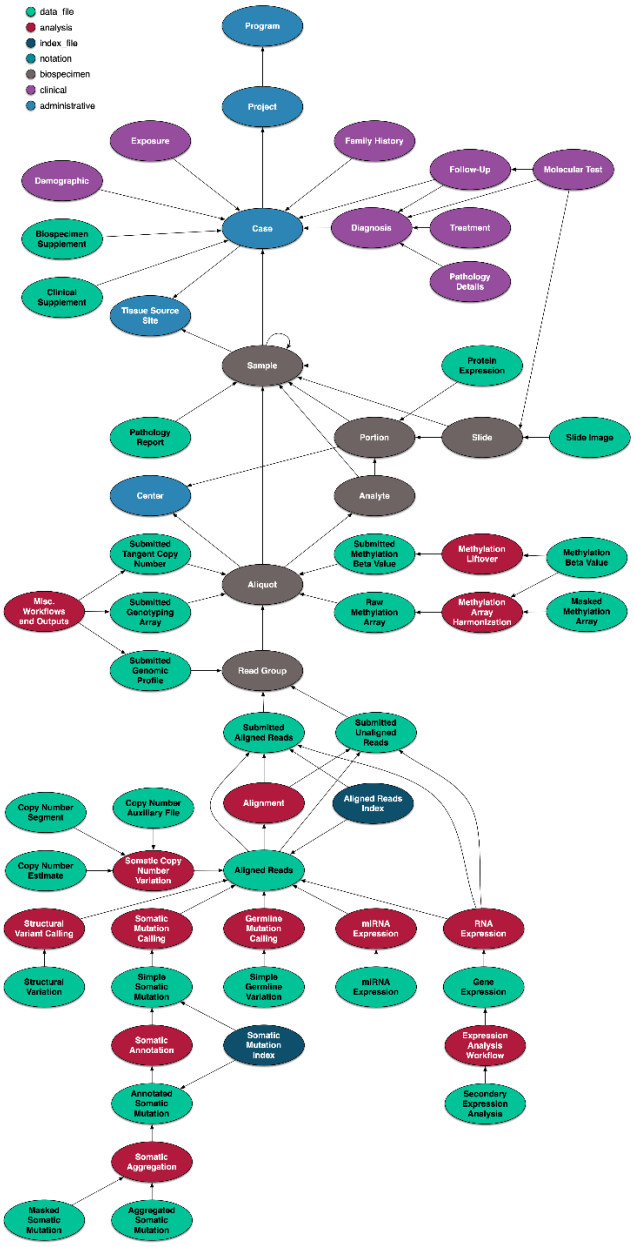

Ilustración 3 - Modelo de datos del GDC a 01 de febrero de 2020

Si nos centramos en el volumen de información albergado en la misma, podemos extraer de su página principal un resumen estadístico de los datos almacenados en ella:

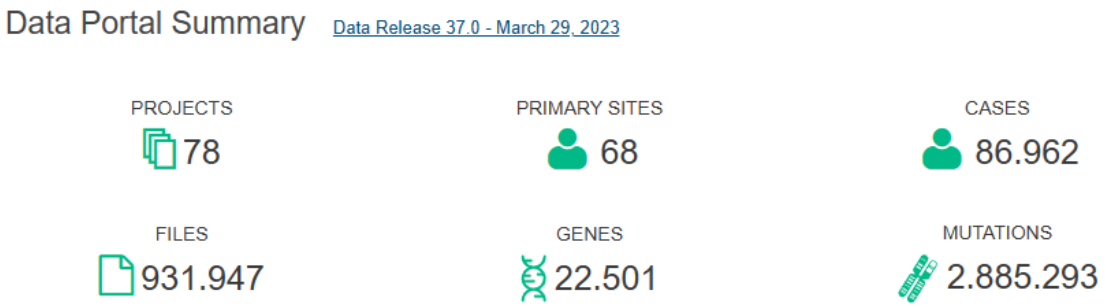

Ilustración 4 - Cantidad de datos del GDC a 29 de marzo de 2023

3.2. Análisis y detalle de la estructura de datos

La estructura de datos de la base de datos del GDC se divide en dos secciones principales: metadatos y datos. Los metadatos describen los atributos y características de los datos almacenados en la base de datos, mientras que los datos contienen los archivos de datos brutos generados a partir del análisis genómico y proteómico de las muestras de tejido.

3.2.1. Metadatos:

Esta sección se divide en cuatro partes principales: Caso, Muestra, Archivo y Liberación de datos.

3.2.1.1. Caso

La sección de Caso describe información sobre el paciente, incluyendo la edad, el género, la raza, el estado de supervivencia, el estadio de la enfermedad y el tipo de cáncer. Los principales campos disponibles en esta sección son:

Campo	Descripción
Case ID	Identificador único asignado a cada caso en la base de datos

Project ID	Identificador que indica el proyecto al que se asignó el caso
Diagnosis ID	Identificador que indica el tipo de cáncer que se está investigando
Age at Diagnosis	Edad del paciente en el momento del diagnóstico
Gender	Género del paciente
Race	Raza del paciente
Vital Status	Estado de supervivencia del paciente
Disease Type	Tipo de cáncer que se está investigando
Primary Site	Sitio primario del tumor
Tumor Stage	Estadio del tumor
Histological Type	Tipo histológico del tumor
Treatment Type	Tipo de tratamiento recibido por el paciente
Sample Type	Tipo de muestra utilizada para el análisis genómico
Submitting Institution	Institución que proporcionó los datos

3.2.1.2. Muestra

La sección de Muestra describe las muestras de tejido utilizadas en los análisis, incluyendo el tipo de muestra, método de recolección, calidad de la muestra y cantidad de material disponible. Los principales campos disponibles en esta sección son:

Campo	Descripción
Sample ID	Identificador único asignado a cada muestra en la base de datos
Case ID	Identificador del caso al que pertenece la muestra
Sample Type	Tipo de muestra (por ejemplo, tumor, sangre, tejido normal, etc)
Tissue Type	Tipo de tejido de la muestra (por ejemplo, pulmón, hígado, etc)
Source Site	Sitio de origen de la muestra
Is FFPE	Indicador que indica si la muestra fue fijada en formalina y embebida en parafina
Tumor Content	Proporción de células tumorales en la muestra
Total Nucleic Acid Input	Cantidad de material genómico disponible para el análisis
Total Nucleic Acid Input	Cantidad de material genómico disponible para el análisis

3.2.1.3. Archivo

La sección de Archivo contiene información sobre los archivos de datos brutos generados a partir de la secuenciación y otros análisis genómicos y proteómicos de las muestras de tejido. Los principales campos disponibles en esta sección son:

Campo	Descripción
File ID	Identificador único asignado a cada archivo en la base de datos
Case ID	Identificador del caso al que pertenece el archivo
Sample ID	Identificador de la muestra a partir de la cual se generó el archivo
Data Category	Categoría de datos (secuenciación de exoma, secuenciación de ARN, proteómica, etc)
Data Type	Tipo de datos (secuencias de lectura única, secuencias de lectura pareja, proteínas identificadas, etc)
Data Format	Formato de archivo utilizado para almacenar los datos (FASTQ, BAM, VCF, etc)
File Name	Nombre del archivo
File Size	Tamaño del archivo en bytes
MD5 Checksum	Valor hash que se utiliza para verificar la integridad del archivo

3.2.1.4. Liberación de datos

La sección de Liberación de datos contiene información sobre cuándo se hicieron públicos los datos y si se aplicaron restricciones de acceso a los mismos. Los principales campos disponibles en esta sección son:

Campo	Descripción
Release ID	Identificador único asignado a cada versión de liberación de datos
Release Date	Fecha en que se hizo pública la versión de liberación de datos
Data Level	Nivel de procesamiento de los datos (nivel 1: datos brutos, nivel 2: datos procesados, nivel 3: datos analizados)
Access	Indica si los datos están disponibles públicamente o si se aplican restricciones de acceso

3.2.2. Datos

La sección de datos contiene los archivos de datos brutos generados a partir del análisis genómico y proteómico de las muestras de tejido. Estos archivos se organizan en dos categorías principales: Secuenciación y Proteómica.

3.2.2.1. Secuenciación

La categoría de Secuenciación contiene los datos brutos generados a partir de la secuenciación de ADN y ARN. Los tipos de datos disponibles en esta categoría incluyen:

- **Secuenciación de Exoma:** Datos de secuenciación generados a partir de la secuenciación del exoma (es decir, la parte del genoma que codifica para proteínas).
- **Secuenciación de Genoma Completo:** Datos de secuenciación generados a partir de la secuenciación del genoma completo (es decir, todo el ADN de una célula).
- **Secuenciación de ARN:** Datos de secuenciación generados a partir de la secuenciación de ARN, que proporcionan información sobre la expresión génica en una muestra.
- **Secuenciación de Epigenoma:** Datos de secuenciación generados a partir de la secuenciación del epigenoma, que proporcionan información sobre la estructura de la cromatina y la regulación génica.
- **Datos de Anotación:** Datos que proporcionan información sobre la ubicación y función de los genes y otros elementos genómicos.

3.2.2.2. Proteómica

La categoría de Proteómica contiene los datos brutos generados a partir del análisis proteómico de las muestras de tejido. Los tipos de datos disponibles en esta categoría incluyen:

- **Identificación de Proteínas:** Datos generados a partir de la identificación de proteínas presentes en una muestra.
- **Cuantificación de Proteínas:** Datos generados a partir de la cuantificación de las proteínas presentes en una muestra.
- **Modificación Post-Traduccional:** Datos generados a partir de la identificación de modificaciones post-traduccionales en las proteínas presentes en una muestra.

3.3. Conclusión

La estructura de datos de la base de datos del GDC está altamente organizada y proporciona a los investigadores el acceso a datos genómicos y proteómicos de alta calidad de una amplia variedad de tipos de tumores. La base de datos contiene información detallada sobre cada muestra de tejido, incluyendo información clínica y de tratamiento, y proporciona acceso a datos brutos generados a partir de la secuenciación y otros análisis genómicos y proteómicos. Los campos disponibles en la base de datos permiten a los investigadores buscar y filtrar datos según sus necesidades específicas y realizar análisis avanzados para descubrir nuevas relaciones entre la genómica del cáncer y la respuesta al tratamiento.

4. Módulo GenomicDataCommons

4.1. Descripción del paquete

El módulo GenomicDataCommons de Bioconductor es una herramienta que permite a los investigadores acceder, descargar y procesar los conjuntos de datos genómicos alojados en la base de datos del GDC de una manera eficiente y programática encapsulando las peticiones a la interfaz de programación de aplicaciones (API) que la plataforma proporciona. Gracias a éste, los usuarios pueden descargar datos genómicos relevantes para su investigación y manipularlos en R sin tener que acceder a ellos manualmente desde el propio sitio web del GDC.

El módulo también ofrece una serie de funciones para filtrar, procesar y transformar los datos genómicos descargados, las cuales permiten a los investigadores seleccionar datos específicos basados en criterios específicos como el tipo de tumor, el estado de la enfermedad, la edad del paciente y el tipo de datos genómicos.

El módulo ofrece herramientas de manipulación de datos que permiten a los usuarios procesar y transformar los datos descargados para su posterior análisis. Además, permite que los usuarios puedan fusionar diferentes conjuntos de datos para integrar información clínica y genómica, lo que puede permitir identificar asociaciones entre las características genómicas y clínicas del cáncer, mejorando la comprensión de la biología de la enfermedad.

Una ventaja adicional del módulo es que se actualiza automáticamente a medida que se agregan nuevos datos al GDC, lo que permite a los usuarios acceder a los datos más recientes y actualizar sus análisis en consecuencia

4.2. Funciones disponibles

El módulo GenomicDataCommons ofrece una gran cantidad de funciones para acceder a la base de datos del GDC, de ellas, destacamos las siguientes:

- **search():** Esta función permite buscar datos de muestras de tejido en la base de datos del GDC. Las búsquedas se pueden realizar utilizando diferentes filtros, como el tipo de tumor, la información clínica, tratamiento, etc.
- **gdcdata():** Esta función permite descargar datos genómicos y proteómicos de las muestras de tejido seleccionadas. Se pueden descargar diferentes tipos de datos, como de expresión génica, metilación del ADN, variación genética y datos proteómicos.
- **filter():** Esta función permite filtrar los datos descargados para seleccionar los que son relevantes para el análisis que se desea realizar. Permite la aplicación de filtros por diferentes criterios, como la ubicación del gen, la mutación específica, la expresión génica, etc.

- **facets() y aggegations():** Estas funciones proporcionan resúmenes estadísticos de los datos descargados, lo que puede ayudar a los usuarios a comprender mejor los datos y a tomar decisiones informadas sobre cómo analizarlos.
- **results() y results_all():** Estas funciones permiten obtener la información resultante de las consultas una vez aplicados los correspondientes filtros y agrupaciones.

Como se ha comentado previamente, el módulo nos ofrece una gran cantidad de funciones y aquí únicamente hemos documentado aquellas que hemos considerado que son fundamentales para operar poder entender la filosofía del mismo y poder empezar a operar con él.

En la aplicación desarrollada podremos observar y comprender de una forma más sencilla como se utilizan y estas y otras que posiblemente no se han reflejado en este análisis inicial.

En el caso de querer profundizar más en las distintas funciones disponibles, existe una gran cantidad de información técnica en la página oficial del módulo.

5. Materiales y métodos

5.1. Tecnología empleada para el desarrollo

Una de las principales ventajas de utilizar RShiny para desarrollar una aplicación es su capacidad para crear visualizaciones interactivas y dinámicas, lo que permite al usuario interactuar directamente con los datos, explorar diferentes variables y ajustar parámetros para analizar su impacto en tiempo real. Esta interactividad facilita enormemente la comprensión y la interpretación de los datos.

Uno de los módulos que R nos proporciona nos permite utilizar el lenguaje Markdown, definido a continuación, el cual nos permite incluir texto explicativo,

imágenes, ecuaciones matemáticas y código fuente en un formato muy fácil de entender.

> Markdown es un lenguaje de marcado ligero creado por John Gruber en 2004, que se destaca por su enfoque en la legibilidad y la simplicidad. El lenguaje tiene una sintaxis basada en texto plano que permite a los usuarios escribir documentos de texto simple utilizando caracteres especiales y convenciones de formato para representar elementos estructurales, como encabezados, párrafos, listas y enlaces. Su facilidad de uso y capacidad de conversión a diversos formatos, como HTML y PDF, lo hacen altamente compatible y portable. Además, al ser independiente de plataforma, los documentos Markdown pueden ser editados y visualizados en cualquier editor de texto simple.

Vista que la combinación de RShiny con Markdown nos ofrece un enfoque flexible y muy potente para la creación de informes claros y coherentes, éstas son las tecnologías que se van a utilizar para llevar a cabo la aplicación objeto de este trabajo.

5.2. Publicación de la aplicación

Shinnyapps.io es una plataforma en línea que brinda a los investigadores y académicos una manera sencilla de crear, publicar y compartir aplicaciones interactivas basadas en R y RStudio. Está diseñada específicamente para aquellos que desean presentar sus análisis y resultados de una manera interactiva y accesible.

Con el objetivo de simplificar el acceso y las pruebas a la aplicación desarrollada, se ha dado de alta una aplicación llamada “GDC Tutorial” en la siguiente URL https://adocasuoc.shinyapps.io/GDC_Tutorial/. A medida que los distintos apartados de la aplicación se vayan completando, se irá actualizando la aplicación en dicha ubicación.

5.3. Gestor de código fuente

Los gestores de código fuente son herramientas utilizadas en el desarrollo de software para controlar y gestionar las versiones del código fuente de un proyecto. Su función principal es rastrear y registrar los cambios realizados en los archivos de código a lo largo del tiempo, hecho que permite a los desarrolladores colaborar de manera eficiente, mantener un historial detallado de las modificaciones y revertir a versiones anteriores si es necesario.

Una de las plataformas de alojamiento de repositorios de código fuente más conocidas es GitHub, la cual está basada en la tecnología de control de versiones Git y nos proporciona una interfaz web junto a herramientas colaborativas para administrar y compartir proyectos de software.

Otra de las ventajas de GitHub es que no tiene coste para los usuarios individuales y nos permite configurar el acceso de cada uno de nuestros proyectos alojados en él de manera pública o privada.

Para el desarrollo de este proyecto se ha creado un repositorio en la siguiente URL https://github.com/AdoCas/GDC_Tutorial en el que se irán subiendo los distintos cambios en el proyecto a medida que éstos sean funcionales.

5.4. Guion de la aplicación

Con el fin de poder mostrar las principales funcionalidades de la librería GenomicDataCommons, hemos definido el siguiente guion sobre el que se va a basar el tutorial desarrollado:

- Instalación del paquete GenomicDataCommons: pasos necesarios para instalar el paquete GenomicDataCommons y sus dependencias.
- Obtención de credenciales de acceso al GDC: cómo obtener credenciales de acceso para utilizar la plataforma GDC desde R.
- Consulta de datos en GDC: cómo realizar consultas de datos en la plataforma GDC utilizando la función GDCquery.
- Agrupación de datos: cómo utilizar las funciones *facets* y *aggregations* para realizar consultas avanzadas y obtener resultados agregados.

- Filtrado de datos: cómo utilizar la función GDCquery para filtrar los datos de interés en función de criterios específicos, como tipo de tumor, tipo de muestra, etc.
- Descarga de datos de GDC: cómo descargar los datos filtrados de GDC en R utilizando la función GDCdata.
- Análisis integrado de datos de GDC: cómo integrar los datos de GDC con otros tipos de datos, como datos clínicos o de expresión génica, para obtener una imagen más completa del conjunto de datos.
- Análisis exploratorio de datos de GDC: cómo utilizar herramientas estadísticas y gráficas para explorar los datos descargados de GDC y obtener información relevante en base a consultas planteadas.
- DashBoard de ejemplo: mostrará un dashboard con información proveniente de distintas consultas a la base de datos GDC.

6. Resultados

Basándonos en los objetivos generales y específicos definidos en los apartados 1.2.1 y 1.2.2 y junto a los materiales y métodos definidos en el apartado 5, se han conseguido obtener los siguientes resultados:

- Se ha realizado el estudio y la documentación de la estructura de la base de datos del GDC. (Apartado 3)
- Se ha realizado el estudio y la documentación de las funciones principales del módulo de Bioconductor GenomicDataCommons para la conexión con la base de datos del GDC. (Apartado 4)
- Se ha desarrollado una aplicación con Shiny que sirve de tutorial para el uso del módulo GenomicDataCommons.
    - El código fuente de la aplicación Shiny se ha publicado en GitHub y está disponible en la siguiente dirección: *https://github.com/AdoCas/GDC_Tutorial*
    - La misma aplicación se ha dejado accesible a todo el mundo en la plataforma shinyapps.io en la siguiente dirección: https://adocasuoc.shinyapps.io/GDC_Tutorial/
- Se ha desarrollado un Dashboard para mostrar información proveniente de la base de datos del GDC utilizando el módulo GenomicDataCommons, en este Dashboard se muestran datos acumulados que coinciden con la información mostrada en la página del GDC.
    - El Dashboard se ha integrado dentro de la aplicación Shiny comentada en el punto anterior.

6.1. Aplicación Shiny: GDC Tutorial

6.1.1. Visión general de la aplicación.

La aplicación se ha planteado como una serie de pasos con el objetivo de proporcionar a los usuarios los conocimientos necesarios para el correcto uso del módulo GenomicDataCommons y se complementa con un apartado final

llamado Dashboard en el que se visualiza un resumen de los datos contenidos en GDC.

En los siguientes apartados se realiza un resumen de cada uno de los pasos del tutorial, en cada uno de ellos se explica el contenido principal del mismo, las funcionalidades que proporcionan y se complementan con algunas capturas de pantalla de la aplicación. En el caso de querer acceder a todo el contenido, éste se encuentra detallado en el Anexo 10.1, en la aplicación publicada y en el repositorio de fuentes indicado anteriormente.

6.1.2. Paso 1 – Instalación

El primer paso del tutorial se centra en explicarle al usuario los requisitos necesarios para la instalación del paquete GenomicDataCommons, estos son:

- La versión de R sobre la que trabaja el tutorial.
- Instalación de dependencias.
- Instalación de BioConductor.
- Instalación del módulo GenomicDataCommons.
- Carga del módulo instalado.

Ilustración 5 - GDC Tutorial - Paso 1

6.1.3. Paso 2 – Obtención de credenciales

En el segundo paso del tutorial nos centramos en los pasos necesarios para la obtención de credenciales que nos permitan acceder a la información de acceso restringido o controlado de la base de datos del GDC, estos son:

- Obtener una cuenta eRA Commons Account.
- Acceder a la base de datos bdGaP.
- Acceder a un proyecto de investigación.
- Registrarse en GDC.
- Obtener el Token y utilizarlo para conectar desde R.

Todos los pasos se han detallado y complementado con los enlaces que se han considerado necesarios para poder obtener el Token de autenticación, no obstante, ha sido totalmente imposible crear una cuenta eRA de ejemplo al no aceptar la plataforma usuarios individuales no vinculados a entidades oficiales y existir un problema con la autenticación federada entre la plataforma y la Universitat Oberta de Catalunya.

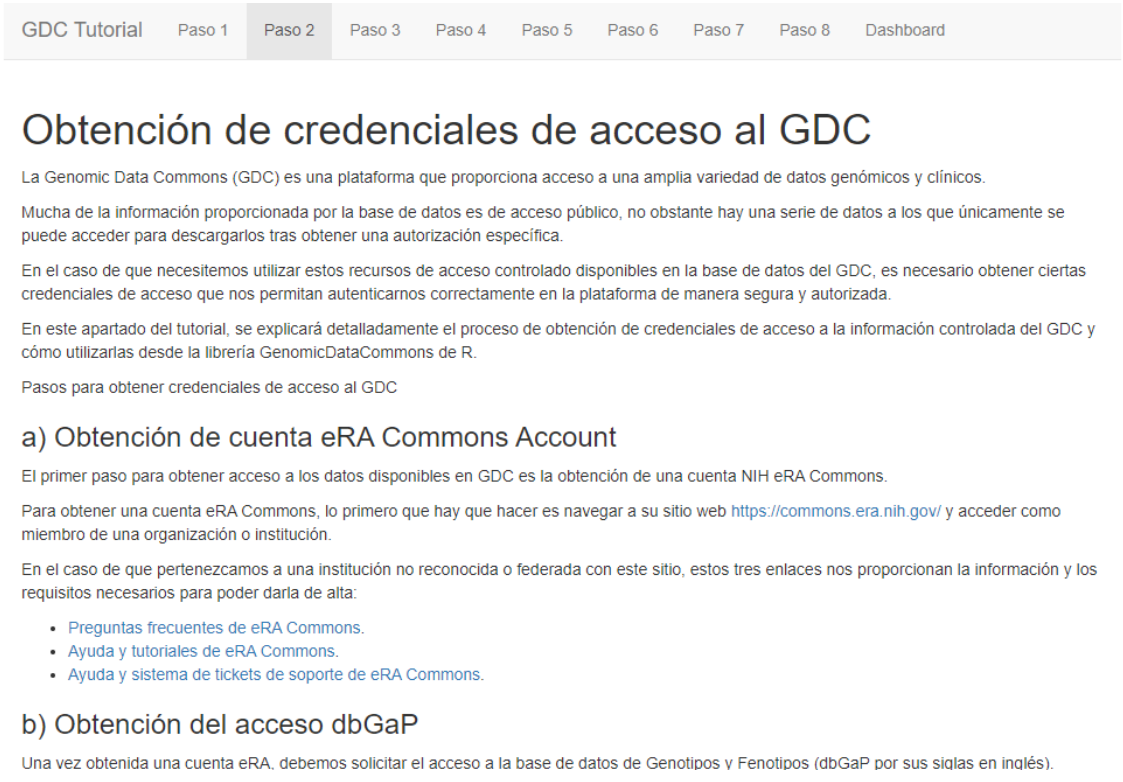

Ilustración 6 - GDC Tutorial - Paso 2

6.1.4. Paso 3 – Función Query

En el tercer paso del tutorial nos centramos en la función *query*, la más importante del tutorial. Es la que realmente nos permite acceder a la información proporcionada por la base de datos.

Ilustración 7 - GDC Tutorial - Paso 3.1

Tras las explicaciones de los distintos parámetros y constructores específicos proporcionados por la herramienta, se muestra un apartado interactivo en el que los usuarios pueden observar la información que proporcionan por defecto las distintas entidades de la base de datos.

Ilustración 8 - GDC Tutorial - Paso 3.2

6.1.5. Paso 4 – Agrupación de resultados

En el cuarto paso del tutorial nos centramos en las funciones facets y aggregations, que nos permiten obtener resultados agregados y agrupaciones en base a las características que nos interesen de los datos genómicos.

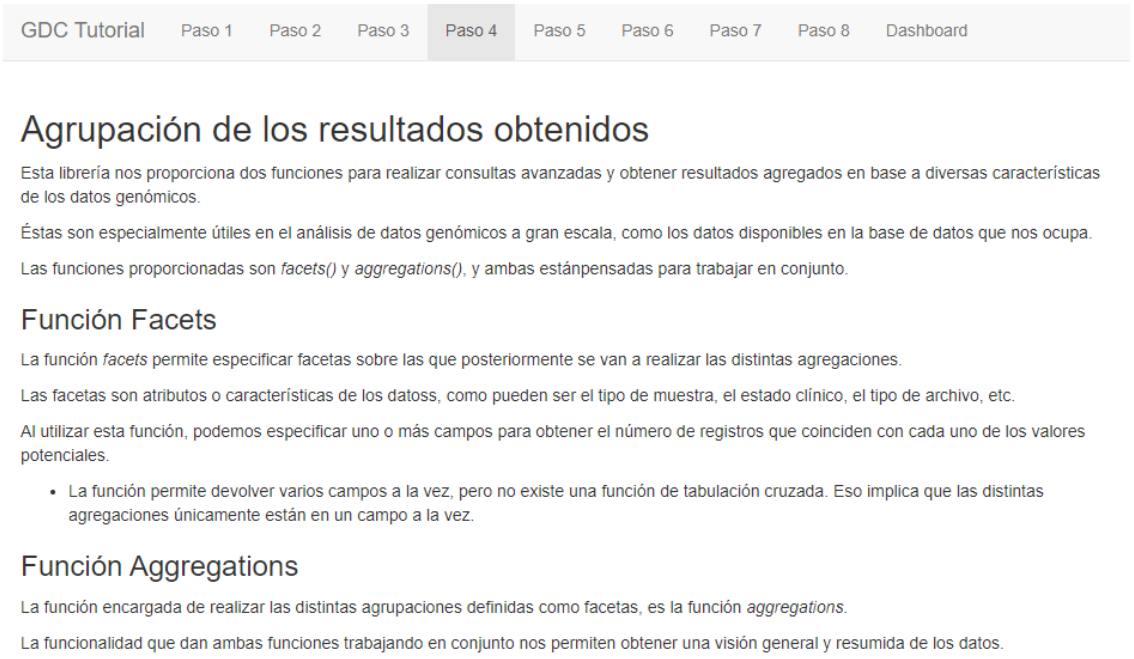

Ilustración 9 - GDC Tutorial - Paso 4.1

Tras las correspondientes explicaciones de las funciones, se muestra un apartado interactivo en el que los usuarios pueden consultar el resultado de aplicar varias agrupaciones sobre los datos relativos a los ficheros disponibles.

Ilustración 10 - GDC Tutorial - Paso 4.2

6.1.6. Paso 5 – Filtrado de los datos obtenidos

En el quinto paso del tutorial nos centramos en otra de las funciones más importantes del módulo, la función *filter*.

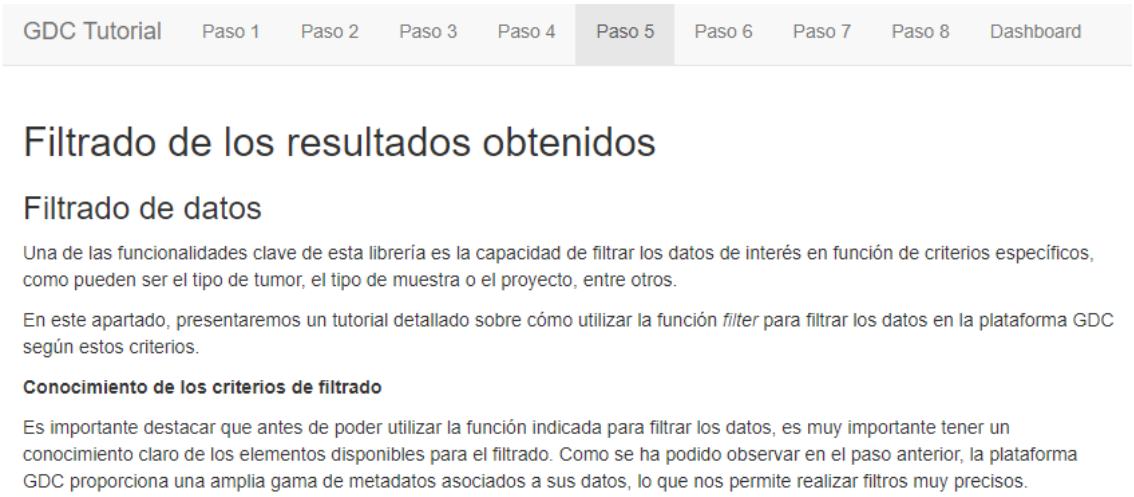

Ilustración 11 - GDC Tutorial - Paso 5.1

Como el uso de esta función no tiene prácticamente ninguna complejidad y depende principalmente de los conocimientos que tenga el usuario de los datos de la plataforma, tras las explicaciones se ha montado un sistema de consultas dinámicas con el fin de que el usuario se pueda familiarizar de una forma muy sencilla con el modelo de datos.

El sistema montado permite al usuario realizar siguientes acciones en secuencia:

- Seleccionar cualquier tipo de entidad con la que trabajar:

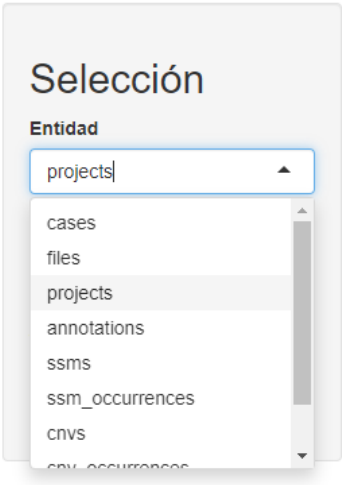

Ilustración 12 - GDC Tutorial - Paso 5.2

- Una vez seleccionada la entidad, se le permite seleccionar uno de sus campos principales:

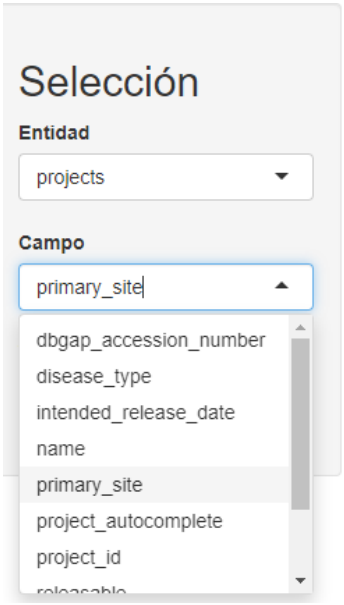

Ilustración 13 - GDC Tutorial - Paso 5.3

- Una vez seleccionado el campo, seleccionar uno de los valores disponibles en ese campo para filtrar los resultados:

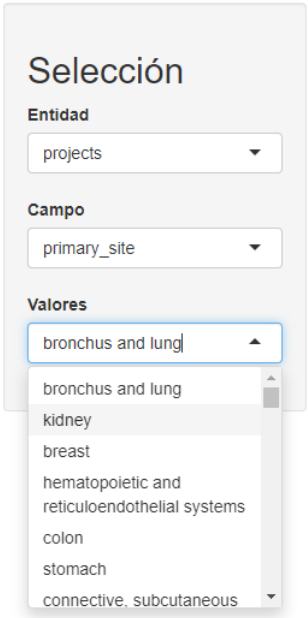

Ilustración 14 - GDC Tutorial - Paso 5.4

Tras realizar las selecciones indicadas, la aplicación muestra la consulta que se va a ejecutar y devuelve los datos resultantes.

6.1.7. Paso 6 – Descarga de datos

En el sexto paso nos centramos en la función proporcionada por la herramienta para la descarga de datos *gdcdata*.

Se empieza con una breve explicación de distintos motivos para plantearse la descarga de los datos de la aplicación, antes de entrar a explicar cómo funciona y las opciones que ésta nos ofrece.

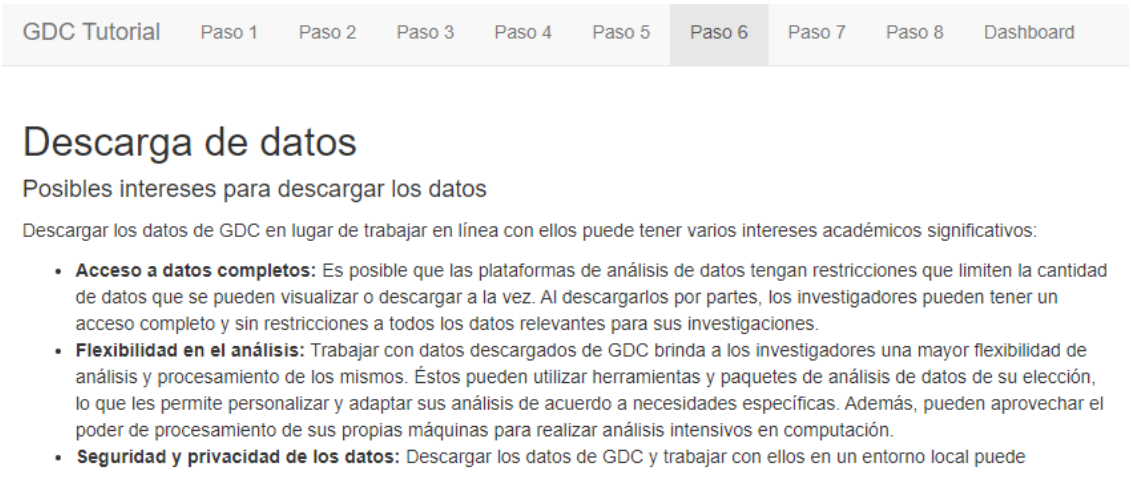

Ilustración 15 - GDC Tutorial - Paso 6.1

Se finaliza el apartado mostrando un ejemplo de código con todos los pasos necesarios para la descarga de un conjunto de datos específico. En este caso, el código no es interactivo para evitar descargar archivos innecesarios:

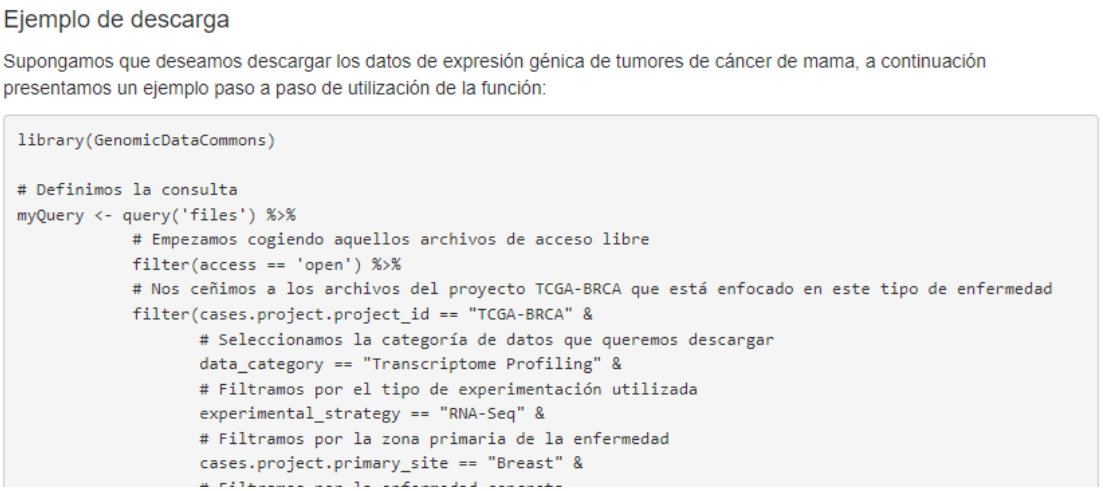

Ilustración 16 - GDC Tutorial - Paso 6.2

6.1.8. Paso 7 – Análisis integrado de datos

En el séptimo paso nos centramos en como enfocar el análisis integrado de datos.

Este punto tiene un enfoque puramente teórico y pretende proporcionarle al usuario una serie de pasos para poder realizar este tipo de análisis en condiciones.

Los pasos que se detallan son:

- Preparación de los datos
- Exploración y limpieza de los datos
- Integración de los datos
- Análisis integrado
- Visualización de resultados

Ilustración 17 - GDC Tutorial - Paso 7

6.1.9. Paso 8 – Análisis exploratorio de los datos

El octavo paso pretende mostrarle al usuario un uso práctico de la librería GenomicDataCommons para realizar un análisis exploratorio de los datos que la base de datos contiene.

Para conseguir eso, se han planteado una serie de preguntas relacionadas con el contenido y las instrucciones necesarias para la obtención y visualización de la información buscada.

Las preguntas planteadas son:

- ¿Cuántos proyectos tiene cada uno de los programas existentes en GDC?
- ¿Cuántos casos tiene registrado cada uno de los proyectos del programa TCGA?
- ¿De qué trata exactamente el proyecto TCGA?
- ¿De dónde proceden las distintas muestras asociadas al proyecto?
- ¿A qué género, raza y etnia pertenecen las muestras del proyecto?
- ¿Cuántos proyectos hay con muestras de cáncer de pecho, además del TCGA-BRCA?
- ¿Cuáles serían los códigos de estos proyectos?
- ¿Cuántos casos relacionados tiene cada uno de ellos?

Ilustración 18 - GDC Tutorial - Paso 8

6.1.10. Dashboard

La última opción de la aplicación es la correspondiente al Dashboard, en la que se muestra información estadística relativa a la demografía y al origen de las muestras de los casos analizados.

Este apartado está dividido en dos zonas claramente diferenciadas, el panel de filtrado y los resultados estadísticos.

Panel de filtrado

El panel de filtrado permite especificar el origen de los datos que queremos que se visualicen en el área de resultados, éste ofrece tres apartados de selección:

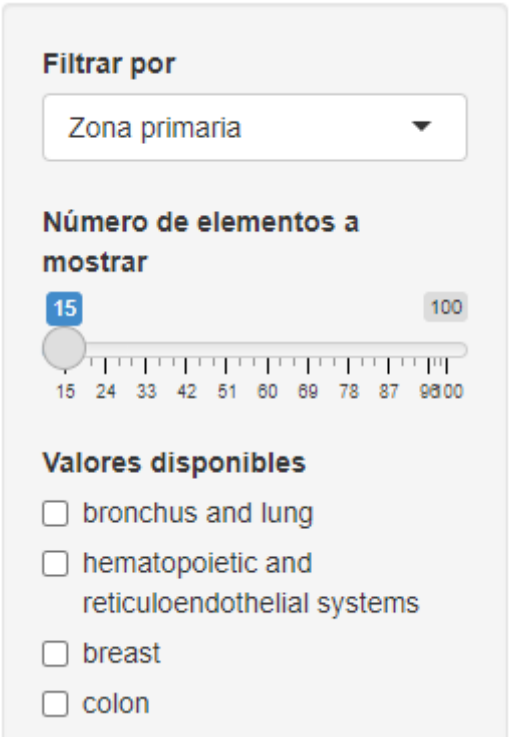

Ilustración 19 - GDC Tutorial - Dashboard 1

- **Filtrar por:** Este apartado permite seleccionar entre tres metadatos diferentes como filtro para los resultados de los distintos casos disponibles.
- **Número de elementos a mostrar:** Este apartado limita la cantidad de valores disponibles para filtrar los elementos por el metadato seleccionado.

- **Valores disponibles:** En este apartado se muestran tantos valores disponibles para seleccionar como el número de elementos a mostrar seleccionado en el punto anterior. Los valores mostrados se corresponden a los valores disponibles para el metadato seleccionado, y aparecen ordenados, de forma descendente, en función de la cantidad de casos que tiene cada uno de los metadatos asociado.

La selección de uno o varios valores disponibles genera el recálculo automático de todos los resultados estadísticos mostrados.

Resultados estadísticos

El panel de resultados estadísticos es el encargado de visualizar la información estadística en función de lo seleccionado en el panel de filtrado, éste divide la información en dos partes:

Tabla “Elementos encontrados”

Esta tabla contiene todos los valores disponibles para el metadato de filtrado seleccionado por el usuario y la cantidad de casos asociados a cada uno de ellos.

La tabla le permite al usuario seleccionar la cantidad de elementos a mostrar en cada página, ordenar por nombre o número de casos y filtrar los resultados.

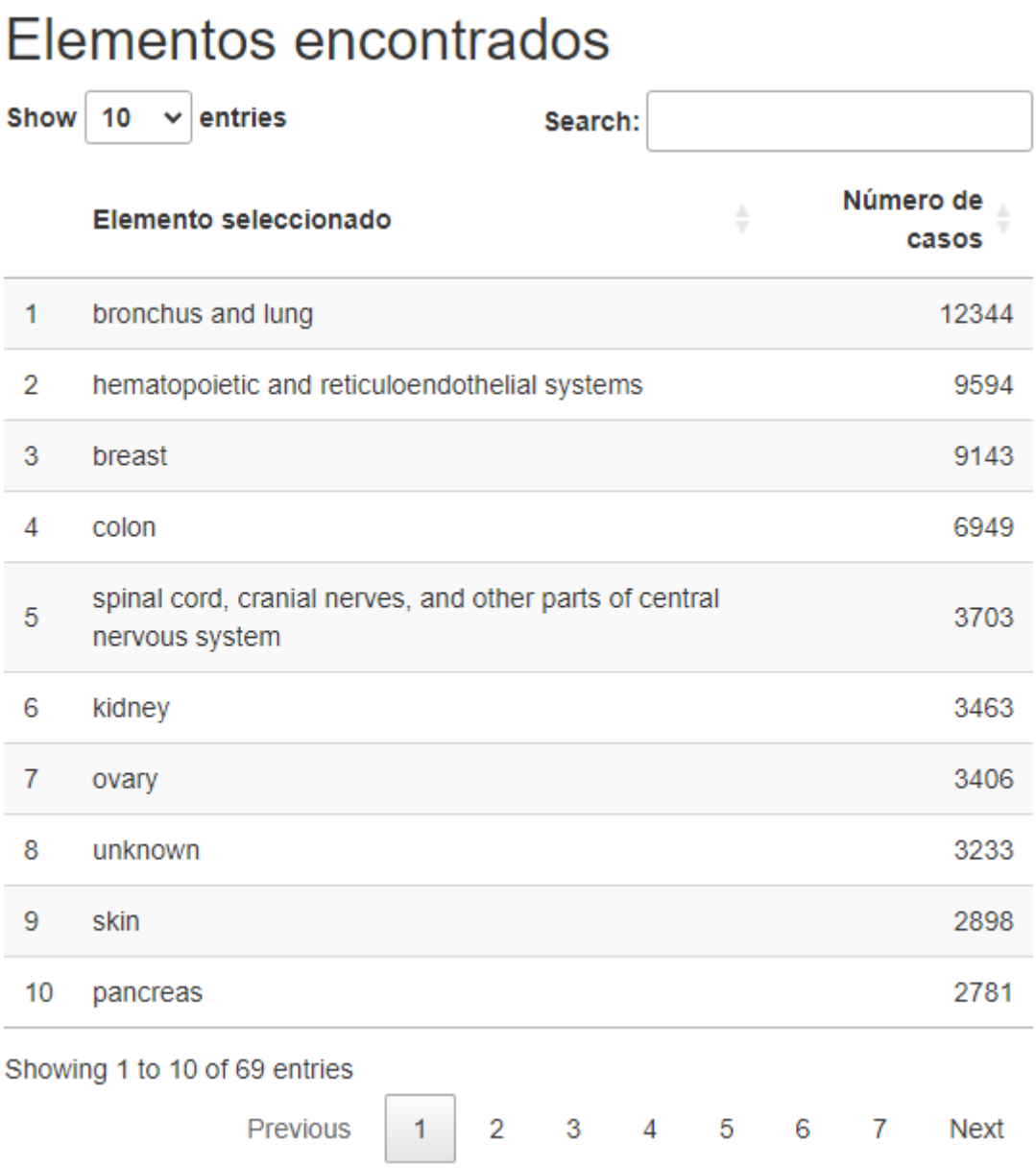

Ilustración 20 - GDC Tutorial - Dashboard 2

Gráficas estadísticas

El apartado “gráficas estadísticas” muestra mediante una serie de gráficos de tipo pastel que se van actualizado cada vez que se modifica la selección, los datos resultantes de filtrar los casos disponibles en base a los metadatos seleccionados en el apartado “valores disponibles”.

Los gráficos mostrados son:

- **Proyecto**: Porcentaje y cantidad de casos por proyecto.
- **Tipo de enfermedad**: Porcentaje y cantidad de casos por tipo de enfermedad.
- **Tipo de muestra**: Porcentaje y cantidad de casos por tipo de muestra.
- **Estado vital**: Porcentaje y cantidad de casos por estado vital actual.
- **Género**: Porcentaje y cantidad de casos por género.
- **Raza**: Porcentaje y cantidad de casos por raza.

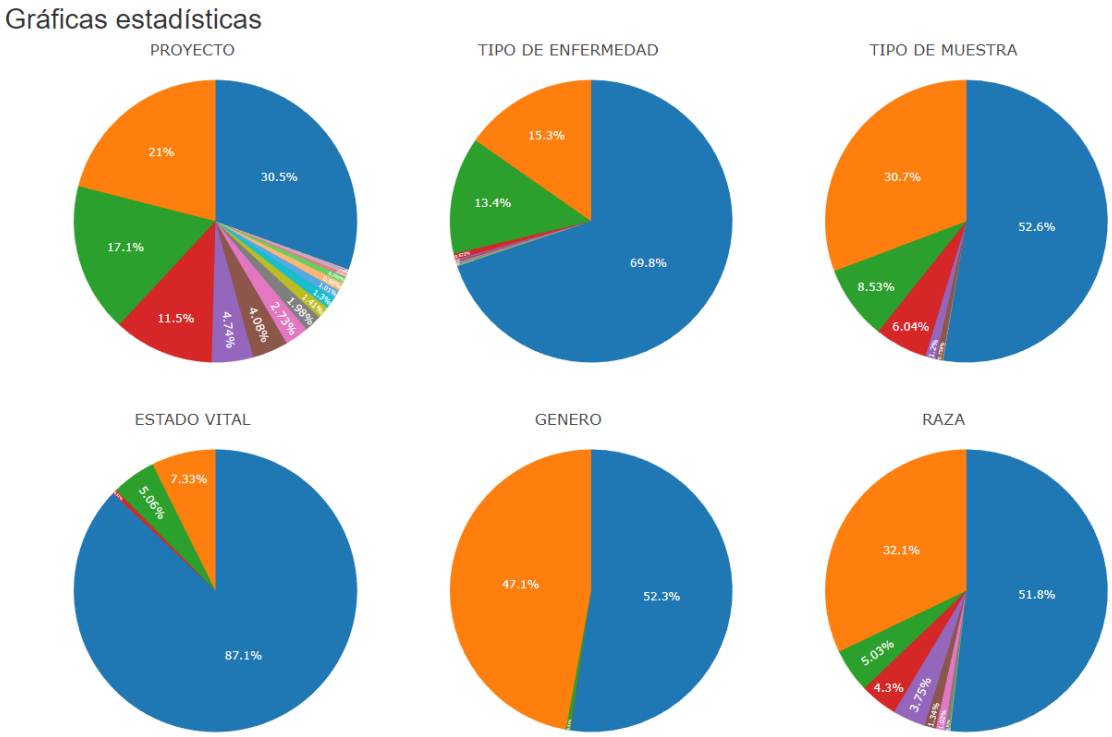

Ilustración 21 - GDC Tutorial - Dashboard 3

Como objetivo principal de la elaboración del Dashboard es poder mostrar al usuario que los datos obtenidos a través de la librería GenomicDataCommons se corresponden con los que se encuentran en la base de datos del GDC, se han intentado proporcionar gráficos fácilmente reproducibles, comparables y visualmente similares a los proporcionados por dicha base de datos.

7. Conclusiones y trabajos futuros

Tal y como se expone en el punto 2.4, durante el estudio del estado del arte del proyecto inicialmente planteado, se llegó a la conclusión que el objetivo principal del trabajo no tenía razón de ser al descubrir que la funcionalidad que se quería implementar ya existía.

Si bien es cierto que desde un principio se plantearon una serie de acciones e hitos con el fin de paliar posibles problemas que pudieran surgir en el desarrollo del trabajo, no se planteó la posibilidad de que prácticamente todo el trabajo realizado durante más de un mes se tuviese que descartar y fuese necesario reenfocarlo todo por completo, que es lo que finalmente sucedió.

Tras el impacto que supuso el imprevisto anterior, se buscó la forma de poder llevar a cabo un trabajo que se mantuviese alineado con el área de desarrollo de aplicaciones y que no se alejase de los elementos que se habían empezado a estudiar, eso llevó a plantear este nuevo enfoque centrado en estudiar y dar a conocer la estructura de la base de datos del GDC, y el uso de la librería GenomicDataCommons mediante una aplicación tutorial.

Se han conseguido alcanzar el objetivo principal del trabajo ajustándonos a la planificación propuesta, desarrollando una herramienta pública disponible en la siguiente dirección https://adocasuoc.shinyapps.io/GDC_Tutorial/ y que permite a los usuarios familiarizarse con el uso de la librería GenomicDataCommons del paquete Bioconductor.

Uno de los problemas encontrados a lo largo del desarrollo, ha sido la imposibilidad de conseguir un usuario para poder acceder a los datos restringidos de la base de datos del GDC, este hecho ha limitado en parte el abanico de los ejemplos a desarrollar.

Con la vista pueda en el futuro, la herramienta desarrollada tiene una gran capacidad de ampliación y mejora en tres aspectos principales:

- Debería ir actualizándose junto con el componente GenomicDataCommons, para ir incorporando las nuevas funcionalidades que se le vayan añadiendo.
- Podrían documentarse con más detalle las funciones secundarias y dotar de más ejemplos los apartados existentes.
- Podría ampliarse la funcionalidad del Dashboard para mostrar datos de otras entidades.

8. Glosario

- **R:** es un lenguaje de programación utilizado principalmente para análisis estadístico, visualización de datos y desarrollo de aplicaciones relacionadas con la ciencia de datos.
- **RStudio:** RStudio es un entorno de desarrollo integrado (IDE) diseñado específicamente para el lenguaje de programación R.
- **Markdown:** Markdown es un lenguaje de marcado ligero que permite escribir texto con formato simple y estructurado utilizando una sintaxis sencilla.
- **Shiny:** Shiny es un paquete de R que permite crear aplicaciones web interactivas directamente desde el lenguaje de programación R
- **GDC:** Son las siglas en ingles de “Genomic Data Commons”, se trata de un repositorio público y centralizado creado por el National Cancer Institute (NCI) de Estados Unidos que almacena datos genómicos y clínicos de alta calidad.
- **Dashboard:** Es una interfaz gráfica que muestra de manera visual información clave y métricas relevantes en tiempo real.
- **API:** Es una pieza de código, en este caso un servicio web, que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidades.

9. Bibliografía

Bos, W. (11 de 05 de 2023). *Mastering Markdown*. Obtenido de https://masteringmarkdown.com/

Cone, M. (13 de 05 de 2023). *Markdown Guide*. Obtenido de https://www.markdownguide.org/

De la Cruz, M. (2019). Como escribir funciones en R. *Ecosistemas 28(3)*, 213-216.

Gentleman, R. C. (2004). Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5(10).

Gomez-Cabrero, D. A. (13 de 03 de 2014). *Data integration in the era of omics: current and future challenges.* doi:https://doi.org/10.1186/1752-0509-8-S2-I1

Gruber, J. (10 de 05 de 2023). *Daring Firewall - Markdown*. Obtenido de https://daringfireball.net/projects/markdown/

Institute, N. N. (05 de 03 de 2023). *Genomic Data Commons*. Obtenido de https://gdc.cancer.gov/

Morgan, M., & Davis, S. (10 de 04 de 2023). *GenomicDataCommons: NIH / NCI Genomic Data Commons Access*. Obtenido de http://bioconductor.github.io/GenomicDataCommons/

ONU. (s.f.). *Objetivos de desarrollo sostenible 2030*. Obtenido de https://www.un.org/sustainabledevelopment/

Team, R. C. (2008). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Core Team.

UOC. (2023). *Guia transversal sobre la CCEG para estudiantado de TFx-EIMT*.

UOC. (s.f.). *Impacto global agenda 2030*. Obtenido de https://www.uoc.edu/portal/es/compromis-social/index.html

Wickham, H. (2015). *R Packages: Organize, Test, Document and Share your Code.* Sebastopol: O'REILLY.

10. Anexos

10.1. Código fuente de la aplicación GDC_Tutorial

```
================================================
Fichero: \app.R
================================================
library(shiny)
library(ggplot2)
library(markdown)
library(plotly)
library(data.table)



ui <- fluidPage(
  navbarPage(
    title = "GDC Tutorial",
    source(file.path("ui/tab01", "tab01.R"), local=TRUE)$value,
    source(file.path("ui/tab02", "tab02.R"), local=TRUE)$value,
    source(file.path("ui/tab03", "tab03.R"), local=TRUE)$value,
    source(file.path("ui/tab04", "tab04.R"), local=TRUE)$value,
    source(file.path("ui/tab05", "tab05.R"), local=TRUE)$value,
    source(file.path("ui/tab06", "tab06.R"), local=TRUE)$value,
    source(file.path("ui/tab07", "tab07.R"), local=TRUE)$value,
    source(file.path("ui/tab08", "tab08.R"), local=TRUE)$value,
    source(file.path("ui", "dashboard.R"), local=TRUE)$value
  )
)

server <- function(input, output, session) {
  source(file.path("server", "server.R"),  local = TRUE)$value
}

shinyApp(ui = ui, server = server)
```

```
================================================
Fichero: \server\server.R
================================================
library("GenomicDataCommons")

# Common
source(file.path("server", "srv_common.R"), local=TRUE)$value

# TAB 3
source(file.path("server", "srv_tab3.R"), local=TRUE)$value

# TAB 4
source(file.path("server", "srv_tab4.R"), local=TRUE)$value

# TAB 5
source(file.path("server", "srv_tab5.R"), local=TRUE)$value

# TAB 8
source(file.path("server", "srv_tab8.R"), local=TRUE)$value

# DASHBOARD FILE
source(file.path("server", "srv_dashboard.R"), local=TRUE)$value



================================================
Fichero: \server\srv_common.R
================================================
# List with some common query options used more than once.
queryOptions=list("cases"="cases",
                  "files"="files",
                  "projects"="projects",
```

```
                    "annotations"="annotations",
                    "ssms"="ssms",
                    "ssm_occurrences"="ssm_occurrences",
                    "cnvs"="cnvs",
                    "cnv_occurrences"="cnv_occurrences",
                    "genes"="genes")
================================================
Fichero: \server\srv_dashboard.R
================================================
# File to store the code for the Dashboard section

# List with the filter type options to apply in the Dashboard
db_filterTypeOptions = list("Zona primaria"="primary_site",
                            "Proyecto"="project.project_id",
                            "Tipo de enfermedad"="disease_type")

# Function responsible for update de dropdown with the filter type
options
updateSelectInput(session     =     getDefaultReactiveDomain(),
"db_sl_filterType",                choices=db_filterTypeOptions,
selected=head(db_filterTypeOptions, 1))

# Load the data that will be used for filter the results.
db_data_aggregated<-cases() %>%
  facet(c("primary_site",  "project.project_id",  "disease_type",
"samples.sample_type")) %>%
  aggregations()


# Function responsible for showing the selected number of values
to filter.
observe({
  req(input$db_sl_optionsQtyy)
```

```
  req(input$db_sl_filterType)

  db_f_primarySite                                                    =
head(db_data_aggregated[[input$db_sl_filterType]],
input$db_sl_optionsQtyy)
  updateCheckboxGroupInput(session = getDefaultReactiveDomain(),
"db_ck_first_filter", choices = db_f_primarySite$key, selected =
NULL)
})


# Function resposible for filter the data that will be shown in
the Dashboard depending on the filter type selected
get_filtered_data <- reactive({
  req(input$db_sl_filterType)
  req(input$db_ck_first_filter)

  facets<-
c("project.project_id","demographic.gender","demographic.vital_s
tatus",                "demographic.race","samples.sample_type",
"disease_type", "demographic.year_of_birth")

  if (input$db_sl_filterType == "primary_site")
  {
    res = cases() %>%
          filter(primary_site %in% input$db_ck_first_filter) %>%
          facet(facets) %>%
          aggregations()
  }
  else if(input$db_sl_filterType == "project.project_id")
  {
    res = cases() %>%
```

```
      filter(project.project_id  %in%  input$db_ck_first_filter)
%>%
      facet(facets) %>%
      aggregations()
  }
  else if(input$db_sl_filterType == "disease_type")
  {
    res = cases() %>%
      filter(disease_type %in% input$db_ck_first_filter) %>%
      facet(facets) %>%
      aggregations()
  }

  res
})


# Function responsible for render the data table shown in the
Dashboard
output$db_dt_primaryZone <- DT::renderDataTable({
  req(input$db_sl_filterType)

  db_f_primarySite                                          =
db_data_aggregated[[input$db_sl_filterType]]
  setcolorder(db_f_primarySite, c("key", "doc_count"))
  DT::datatable(db_f_primarySite,
                selection = "none",
                colnames = c("Elemento seleccionado", "Número de
casos"))
})

# Function responsible for render the pie chart shown in the
Dashboard with the Project information
```

```
output$db_data_plot_project <- renderPlotly({
  data <- get_filtered_data()

  fig <- plot_ly(data$project.project_id,
                 labels=data$project.project_id$key,
                 values=data$project.project_id$doc_count,
                 type="pie",
                 textposition='inside',
                 textinfo = 'percent',
                 insidetextfont = list(color = '#FFFFFF'),
                 hoverinfo = 'text',
                 text = ~paste(doc_count, ' ', key),
                 showlegend = FALSE) %>%
    layout(title = 'PROYECTO',
           xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
           yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
  fig

})

# Function responsible for render the pie chart shown in the
Dashboard with the Disease Type information
output$db_data_plot_diseaseType <- renderPlotly({
  data <- get_filtered_data()

  fig <- plot_ly(data$disease_type,
                 labels=data$disease_type$key,
                 values=data$disease_type$doc_count, type="pie",
                 textinfo = 'percent',
                 insidetextfont = list(color = '#FFFFFF'),
                 hoverinfo = 'text',
```

```
                textposition='inside',
                text = ~paste(doc_count, ' ', key),
                showlegend = FALSE) %>%
    layout(title = 'TIPO DE ENFERMEDAD',
           xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
           yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
  fig
})

# Function responsible for render the pie chart shown in the
Dashboard with the Sample Type information
output$db_data_plot_sampleType <- renderPlotly({
  data <- get_filtered_data()

  fig <- plot_ly(data$samples.sample_type,
                labels=data$samples.sample_type$key,
                values=data$samples.sample_type$doc_count,
type="pie",
                textinfo = 'percent',
                insidetextfont = list(color = '#FFFFFF'),
                hoverinfo = 'text',
                textposition='inside',
                text = ~paste(doc_count, ' ', key),
                showlegend = FALSE) %>%
    layout(title = 'TIPO DE MUESTRA',
           xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
           yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
  fig
})
```

```
# Function responsible for render the pie chart shown in the
Dashboard with the Vital Status information
output$db_data_plot_vitalStatus <- renderPlotly({
  data <- get_filtered_data()

  fig <- plot_ly(data$demographic.vital_status,
                 labels=data$demographic.vital_status$key,
                 values=data$demographic.vital_status$doc_count,
type="pie",
                 textinfo = 'percent',
                 insidetextfont = list(color = '#FFFFFF'),
                 hoverinfo = 'text',
                 textposition='inside',
                 text = ~paste(doc_count, ' ', key),
                 showlegend = FALSE) %>%
    layout(title = 'ESTADO VITAL',
           xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
           yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
  fig
})

# Function responsible for render the pie chart shown in the
Dashboard with the Gender information
output$db_data_plot_gender <- renderPlotly({
  data <- get_filtered_data()

  fig <- plot_ly(data$demographic.gender,
                 labels=data$demographic.gender$key,
                 values=data$demographic.gender$doc_count,
type="pie",
```

```
                textinfo = 'percent',
                textposition='inside',
                insidetextfont = list(color = '#FFFFFF'),
                hoverinfo = 'text',
                text = ~paste(doc_count, ' ', key),
                showlegend=FALSE) %>%
    layout(title = 'GÉNERO',
           xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
           yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
  fig
})

# Function responsible for render the pie chart shown in the
Dashboard with the Race information
output$db_data_plot_race <- renderPlotly({
  data <- get_filtered_data()

  fig <- plot_ly(data$demographic.race,
                labels=data$demographic.race$key,
                values=data$demographic.race$doc_count,
type="pie",
                textinfo = 'percent',
                textposition='inside',
                insidetextfont = list(color = '#FFFFFF'),
                hoverinfo = 'text',
                text = ~paste(doc_count, ' ', key),
                showlegend=FALSE) %>%
    layout(title = 'RAZA',
           xaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE),
```

```
           yaxis = list(showgrid = FALSE, zeroline = FALSE,
showticklabels = FALSE))
  fig
})


================================================
Fichero: \server\srv_tab3.R
================================================
# Function responsible for update de dropdown with the default
query options
updateSelectInput(session      =      getDefaultReactiveDomain(),
"t3_QueryType", choices=queryOptions, selected="cases")

# Function responsible for print in the screen the result function
output$t3_QueryTypeText <- renderText({
  paste(sprintf("myQuery = query(%s) \ndefault_fields(myQuery)",
input$t3_QueryType))
})

# Function responsible for print in the screen the results of the
function execution
output$t3_MyQuery <- renderPrint({
  paste(sprintf("myQuery = query(%s)", input$t3_QueryType))
  myQuery = query(input$t3_QueryType)
  default_fields(myQuery)
})
================================================
Fichero: \server\srv_tab4.R
================================================
# List with the aggregation options used in this tab
t4_AggregationFields=c('type','data_type','state',
'data_category')
```

```
# Function responsible for update de dropdown with the aggregation
options
updateSelectInput(session      =      getDefaultReactiveDomain(),
"t4_Aggregation", choices=t4_AggregationFields, selected="type")


# Function responsible for print in the screen the results of the
aggregation
observe({
  req(input$t4_Aggregation)

  output$t4_ResultAggregation <- renderPrint({
    library(GenomicDataCommons)

    res   =   files()   %>%   facet(c('type','data_type','state',
'data_category')) %>% aggregations()
    res[[input$t4_Aggregation]]
  })
})

================================================
Fichero: \server\srv_tab5.R
================================================
# Function responsible for update de dropdown with the available
query options
updateSelectInput(session      =      getDefaultReactiveDomain(),
"t5_QueryType", choices=queryOptions, selected="cases")

# Function responsible for print in the screen the available fiels
for the selected type
observe({
  req(input$t5_QueryType)
```

```
  myQuery = query(input$t5_QueryType)
  myFields<-default_fields(myQuery)
  updateSelectInput(session,    "t5_Fields",    choices=myFields,
selected=head(myFields,1))
})

# Function responsible for print in the screen the available
values for the selected field
observe({
  req(input$t5_QueryType)
  req(input$t5_Fields)
  myFieldValues<-available_values(input$t5_QueryType,
input$t5_Fields)
  updateSelectInput(session,                    "t5_FieldValues",
choices=myFieldValues, selected=head(myFieldValues,1))
})

# Function responsible for refreshing the screen data
observe({
  req(input$t5_QueryType)
  req(input$t5_Fields)
  req(input$t5_FieldValues)

  # Function responsible for print in the screen the function that
will be executed
  output$t5_ResultQuery <- renderText({
    paste(sprintf("myQuery     =     query(%s)     \nmyQuery     =
filter(myQuery,~  %s  ==  %s)  \nmyQuery  %%>%%  results_all()",
input$t5_QueryType, input$t5_Fields, input$t5_FieldValues))
  })

  # Function responsible for print in the screen the same function
with other format that will be executed
```

```
  output$t5_ResultQueryConcatenated <- renderText({
    paste(sprintf("query('%s') %%>%% \nfilter(%s == '%s') %%>%%
\nresults_all()",      input$t5_QueryType,      input$t5_Fields,
input$t5_FieldValues))
  })

  # Function responsible for print in the screen the result of
the function executed
  output$t5_SelectedValuesInformation <- renderPrint({
    myQuery = query(input$t5_QueryType)
    myQuery     =     filter(myQuery,~     input$t5_Fields     ==
input$t5_FieldValues)
    myQuery %>% results_all()
  })

})
================================================
Fichero: \server\srv_tab8.R
================================================

# Function responsible for print in the screen the result of the
function executed
output$t8_ProjectsByProgram <- renderPrint({
  res = projects() %>% facet("program.name") %>% aggregations()
  head(res)
})

# Function responsible for print in the screen the result of the
function executed
output$t8_ProjectsInTCGA <- renderPrint({
  res = cases() %>% filter(project.program.name == "TCGA") %>%
facet("project.project_id") %>% aggregations()
  head(res)
```

```
})

# Function responsible for print in the screen the result of the
function executed
output$t8_ProjectTCGAInfo <- renderPrint({
  res = projects() %>% filter(project_id == "TCGA-BRCA") %>%
results_all()
  res
})

# Function responsible for load the data that will be printed as
text and plot
t8_res_SampleTypeInTCGA = cases() %>% filter(project.project_id
==    "TCGA-BRCA")    %>%    facet("samples.sample_type")    %>%
aggregations()

# Function responsible for print in the screen the result of the
function executed
output$t8_SampleTypeInTCGA <- renderPrint({
  t8_res_SampleTypeInTCGA
})

# Function responsible for print in the screen the result of the
function executed as a plot
output$t8_Plot_SampleTypeInTCGA <- renderPlot({
  ggplot(t8_res_SampleTypeInTCGA$samples.sample_type,aes(x     =
key, y = doc_count)) +
    geom_bar(stat='identity', colour="#005FFF", fill="#005FFF") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
})

# Function responsible for load the data that will be printed as
text and plot
```

```
t8_res_DemographicTCGA = cases() %>% filter(project.project_id ==
"TCGA-BRCA")           %>%           facet(c("demographic.race",
"demographic.gender",        "demographic.ethnicity"))        %>%
aggregations()

# Function responsible for print in the screen the result of the
function executed
output$t8_DemographicTCGA <- renderPrint({
  t8_res_DemographicTCGA
})

# Function responsible for print in the screen the result of the
function executed as a plot
output$t8_Plot_DemographycRace <- renderPlot({
  ggplot(t8_res_DemographicTCGA$demographic.race,aes(x = key, y =
doc_count)) +
    geom_bar(stat='identity', colour="#005FFF", fill="#005FFF") +
    labs(title="RACE", x="Race", y="Count") +
    theme(plot.title = element_text(size=16, face="bold.italic",
hjust=0.5),
          axis.title.x=element_blank(),
          axis.title.y=element_blank(),
          axis.text.x = element_text(angle = 45, hjust = 1))
})

# Function responsible for print in the screen the result of the
function executed as a plot
output$t8_Plot_DemographycGender <- renderPlot({
  ggplot(t8_res_DemographicTCGA$demographic.gender,aes(x = key, y
= doc_count)) +
    geom_bar(stat='identity', colour="#005FFF", fill="#005FFF") +
    labs(title="GENDER", x="Race", y="Count") +
```

```
    theme(plot.title = element_text(size=16, face="bold.italic",
hjust=0.5),
          axis.title.x=element_blank(),
          axis.title.y=element_blank(),
          axis.text.x = element_text(angle = 45, hjust = 1))
})

# Function responsible for print in the screen the result of the
function executed as a plot
output$t8_Plot_DemographycEthnicity <- renderPlot({
  ggplot(t8_res_DemographicTCGA$demographic.ethnicity,aes(x    =
key, y = doc_count)) +
    geom_bar(stat='identity', colour="#005FFF", fill="#005FFF") +
    labs(title="ETHNICITY", x="Race", y="Count") +
    theme(plot.title = element_text(size=16, face="bold.italic",
hjust=0.5),
          axis.title.x=element_blank(),
          axis.title.y=element_blank(),
          axis.text.x = element_text(angle = 45, hjust = 1))
})

# Function responsible for print in the screen the result of the
function executed
output$t8_HowManyOtherBreastProjects <- renderPrint({
  res  =  projects()  %>%  filter(primary_site  ==  "Breast")  %>%
count()
  res
})

# Function responsible for print in the screen the result of the
function executed
output$t8_IDsIfOtherBreastProjects <- renderPrint({
```

```
  res = projects() %>% filter(primary_site == "Breast") %>%
results_all()
  res$project_id
})

# Function responsible for print in the screen the result of the
function executed
output$t8_NumberOfCases <- renderPrint({
  res = cases() %>% filter(project.primary_site == "Breast") %>%
facet("project.project_id") %>% aggregations()
  res
})

================================================
Fichero: \ui\dashboard.R
================================================
tabPanel("Dashboard",
  h2("Dashboard - Casos disponibles"),

  sidebarLayout(
    sidebarPanel(
      selectInput("db_sl_filterType",
                  "Filtrar por",
                  choices = list(),
                  selected = 1),
      sliderInput("db_sl_optionsQtyy",
                  "Número de elementos a mostrar",
                  min=15,
                  max=100,
                  value=7),
      checkboxGroupInput("db_ck_first_filter",
                         "Valores disponibles",
                         choices = list(),
```

```
                          selected = 1)
      ),
    mainPanel(
      h2("Elementos encontrados"),
      DT::dataTableOutput("db_dt_primaryZone"),

      h2("Gráficas estadísticas"),
      column(4,plotlyOutput("db_data_plot_project")),
      column(4,plotlyOutput("db_data_plot_diseaseType")),
      column(4,plotlyOutput("db_data_plot_sampleType")),
      column(4,plotlyOutput("db_data_plot_vitalStatus")),
      column(4,plotlyOutput("db_data_plot_gender")),
      column(4,plotlyOutput("db_data_plot_race"))
      )
    )
)

================================================
Fichero: \ui\tab01\tab01.md
================================================
# Instalación del paquete GenomicDataCommons

La instalación del paquete GenomicDataCommons es un paso esencial para poder utilizar sus funcionalidades en el entorno de programación R. Este paquete, que forma parte de Bioconductor, proporciona una interfaz para acceder y analizar datos genómicos de la plataforma Genomic Data Commons (GDC).

Para realizar una instalación correcta, se deben seguir los siguientes pasos:

### a) Verificación de la versión de R:
```

Antes de proceder con la instalación del paquete GenomicDataCommons, es importante asegurarse de tener una versión actualizada de R. Esto es importante debido a que algunas funcionalidades del paquete pueden requerir características específicas de versiones recientes de R.

Para verificar la versión de R, se puede ejecutar el siguiente código en la consola de R:

```{r}
R.Version()$version.string
```

Estos ejemplos se han desarrollado utilizando la versión de R 4.2.3.

b) Instalación de paquetes dependientes:

GenomicDataCommons tiene dependencias adicionales que deben estar instaladas previamente para asegurar su correcto funcionamiento.

Algunos de estos paquetes incluyen httr, jsonlite, curl y digest. Estos paquetes se utilizan para realizar solicitudes HTTP, manipular datos en formato JSON y realizar operaciones criptográficas, respectivamente. Para instalar estos paquetes, se puede utilizar la función install.packages() de la siguiente manera:

```{r}
install.packages(c("httr", "jsonlite", "curl", "digest"))
```

c) Instalación del paquete GenomicDataCommons

Una vez que las dependencias están instaladas, se puede proceder a la instalación del paquete GenomicDataCommons utilizando la función BiocManager::install(), que permite instalar paquetes desde el repositorio oficial de Bioconductor. Si no tienes instalado BiocManager, puedes instalarlo ejecutando el siguiente código:

```{r}
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
```

Después de tener BiocManager instalado, debemos ejecutar la siguiente línea de código para instalar el componente GenomicDataCommons:

```{r}
BiocManager::install("GenomicDataCommons")
```

d) Carga del paquete GenomicDataCommons:

Una vez que la instalación se ha completado con éxito, se puede cargar el paquete GenomicDataCommons utilizando la función library():

```{r}
library(GenomicDataCommons)
```

Con estos pasos, se ha completado la instalación del paquete GenomicDataCommons y se encuentra listo para ser utilizado en R.

Es importante destacar que, durante el proceso de instalación, pueden surgir dificultades como problemas de dependencias faltantes o conflictos de versiones. En caso de enfrentar dificultades, es recomendable consultar la documentación oficial del paquete GenomicDataCommons, así como buscar ayuda en foros y comunidades en línea especializadas en R y Bioconductor. Además, es importante mantener actualizadas tanto R como las dependencias y paquetes utilizados, ya que las nuevas versiones pueden corregir errores y ofrecer mejoras en las funcionalidades.

Ahora que el paquete GenomicDataCommons está correctamente instalado, puedes explorar su documentación y comenzar a utilizar sus funciones para acceder y analizar datos genómicos de la plataforma GDC en R.

```
================================================
Fichero: \ui\tab01\tab01.R
================================================
tabPanel("Paso 1",
  includeMarkdown("ui/tab01/tab01.md")
)
```

```
================================================
Fichero: \ui\tab02\tab02.md
================================================
# Obtención de credenciales de acceso al GDC
```

La Genomic Data Commons (GDC) es una plataforma que proporciona acceso a una amplia variedad de datos genómicos y clínicos.

Mucha de la información proporcionada por la base de datos es de acceso público, no obstante hay una serie de datos a los que únicamente se puede acceder para descargarlos tras obtener una autorización específica.

En el caso de que necesitemos utilizar estos recursos de acceso controlado disponibles en la base de datos del GDC, es necesario obtener ciertas credenciales de acceso que nos permitan autenticarnos correctamente en la plataforma de manera segura y autorizada.

En este apartado del tutorial, se explicará detalladamente el proceso de obtención de credenciales de acceso a la información controlada del GDC y cómo utilizarlas desde la librería GenomicDataCommons de R.

Pasos para obtener credenciales de acceso al GDC

a) Obtención de cuenta eRA Commons Account

El primer paso para obtener acceso a los datos disponibles en GDC es la obtención de una cuenta NIH eRA Commons.

Para obtener una cuenta eRA Commons, lo primero que hay que hacer es navegar a su sitio web https://commons.era.nih.gov/ y acceder como miembro de una organización o institución.

En el caso de que pertenezcamos a una institución no reconocida o federada con este sitio, estos tres enlaces nos proporcionan la información y los requisitos necesarios para poder darla de alta:

* [Preguntas frecuentes de eRA Commons](https://era.nih.gov/faqs.htm).
* [Ayuda y tutoriales de eRA Commons](https://era.nih.gov/help-tutorials).
* [Ayuda y sistema de tickets de soporte de eRA Commons](http://grants.nih.gov/support/index.html).

b) Obtención del acceso dbGaP

Una vez obtenida una cuenta eRA, debemos solicitar el acceso a la base de datos de Genotipos y Fenotipos (dbGaP por sus siglas en inglés).

Para ello, debemos navegar a su sitio web [Página de Login de dbGaP](https://dbgap.ncbi.nlm.nih.gov/aa/wga.cgi?page=login) y seguir las instrucciones que en ella se encuentran para obtener acceso a la plataforma y a la información controlada.

En los siguientes enlaces se nos proporciona la información para poder llevar a cabo este proceso con éxito:

* [Vídeo explicativo sobre el acceso a la información controlada](https://www.youtube.com/watch?annotation_id=annotation_747461745&feature=iv&src_vid=-3tUBeKbP5c&v=m0xp_cCO7kA)
* [Preguntas frecuentes de dbGaP](https://www.ncbi.nlm.nih.gov/books/NBK5295/)
* [Ayuda de la plataforma dbGaP](https://dbgap.ncbi.nlm.nih.gov/aa/wga.cgi?page=email&filter=from&from=login)

c) Acceso a un proyecto de investigación

Una vez tenemos acceso a la información controlada de la plataforma dbGaP como investigadores, se nos muestra una página llamada "Mis proyectos" en la que se nos permiten dos opciones:
* Crear un nuevo proyecto de investigación (siguiendo los pasos requeridos tras hacer clic en la opción "Crear nuevo proyecto de investigación")
* Revisar su proyecto y solicitar acceso a datos controlados (haciendo clic en "Revisar proyecto")

Una vez creado un proyecto de investigación, desde el mismo se puede solicitar el acceso a los distintos datasets existentes en la base de datos, y una vez concedido, ya podemos acceder al GDC para consultarlos.

d) Registro en la plataforma GDC

Una vez conseguido el acceso a los datos restringidos, el siguiente paso es crear una aplicación en GDC para poder obtener un Token que nos permita autenticarnos desde R con la plataforma.

Para crear una aplicación, debemos seguir estos pasos una vez hayamos iniciado sesión en la plataforma:

* Navegamos hasta la sección de "Aplicaciones" o "Aplicaciones de desarrollador" en la plataforma.
* Hacemos clic en el botón "Crear una nueva aplicación".
* Proporcionamos un nombre descriptivo para la aplicación y una breve descripción.
* Se te proporcionará un identificador único de la aplicación y un token para autenticarnos con una duración máxima de 30 días. Estos datos serán los necesarios para la autenticación en el siguiente paso.

e) Autenticación desde R utilizando la librería GenomicDataCommons para acceder a la información controlada

El token proporcionado nos permite acceder a la información controlada para poder descargarla y operar con ella sin restricciones.

Al tratarse de una clave de autenticación secreta y cambiante en el tiempo, la librería GenomicDataCommons contempla que el Token esté informado a nivel de sistema en una variable de entorno llamada *GDC_TOKEN*, en un fichero llamado *GDC_TOKEN_FILE* o en un fichero oculto llamado *.gdc_token*

Con el fin de automatizar el acceso al mismo, la librería nos proporciona la función *gdc_token()*, que nos automatiza la consulta del mismo en los elementos comentados anteriormente.

Una vez definido el correspondiente token a nivel de sistema, podemos comprobar que la función lo devuelve correctamente con una instrucción como la siguiente:

```{r}
token = try(gdc_token(),silent=TRUE)
token
```

Una vez tenemos el token debidamente reconocido, veremos posteriormente como utilizarlo para poder descargar la información de acceso restringido.
==========
Fichero: \ui\tab02\tab02.R
==========

```
tabPanel("Paso 2",
  includeMarkdown("ui/tab02/tab02.md")
)

================================================
Fichero: \ui\tab03\tab03.R
================================================
tabPanel("Paso 3",
  h1("Uso de la función Query"),
  includeMarkdown("ui/tab03/tab03_01.md"),
  sidebarLayout(position = "left",
                sidebarPanel(
                  h2("Entidades"),
                  selectInput("t3_QueryType",
                              "Tipo",
                              choices = list(),
                              selected = 1)
                ),
                mainPanel(
                  includeMarkdown("ui/tab03/tab03_02.md"),
                  verbatimTextOutput("t3_QueryTypeText"),
                  verbatimTextOutput("t3_MyQuery")
                )
  )
)
================================================
Fichero: \ui\tab03\tab03_01.md
================================================
La función principal que utilizaremos para realizar las consultas
de datos en la plataforma GDC es la función query. Toda la
funcionalidad del paquete comienza con la construcción de una
consulta en R gracias al objeto GDCQuery.
```

El objeto GDCQuery contiene los filtros, carácterísticas y otros parámetros que nos permiten definir los resultados devueltos, toda esta información se le puede pasar utilizando una única llamada al constructor o ir especificándola paso a paso.

A continuación podemos observar el uso de la clase tal y como se especifica en su documentación:

```{r}
query(entity, filters = NULL, facets = NULL, legacy = FALSE,
  expand = NULL, fields = default_fields(entity))
```

parámetro	descripción
entity	Permite especificar una de las entidades aceptadas por gdc
filters	Permite espeficar un filtro en el constructor, normalmente los resultados se filtran utilizando la función filter que se verá en detalle más adelante
facets	Una lista o vector de facetas para generar acumulados, normalmente se especifican utilizando la función facets que se verá en detalle más adelante
legacy	Un valor booleano que indica si se debe utilizar el archivo "legacy" o no. Consultar https://docs.gdc.cancer.gov/Data_Portal/Users_Guide/Legacy_Archive/ para más información
expand	Una lista o vector de campos a incluir en la información devuelta.
fields	Una lista o vector de campos a devolver. Por norma general no se especifican los campos a devolver, en lugar de ello se utiliza la función select

---

A pesar de que la función query nos genera un objeto GDCQuery adaptado a la entidad pasada por parámetro, el paquete GenominDataCommons nos proporciona constructores específicos para cada una de las entidades contempladas por el mismo, estos son:

* **cases**: Constructor de un objeto GDCQuery para los casos
* **files**: Constructor de un objeto GDCQuery para los ficheros
* **projects**: Constructor de un objeto GDCQuery para los proyectos
* **annotations**: Constructor de un objeto GDCQuery para las anotaciones
* **ssms**: Constructor de un objeto GDCQuery for ssms
* **ssm_occurrences**: Constructor de un objeto GDCQuery for ssm_occurrences
* **cnvs**: Constructor de un objeto GDCQuery for cnvs
* **cnv_occurrences**: Constructor de un objeto GDCQuery for cnv_occurrences
* **genes**: Constructor de un objeto GDCQuery for genes

Eso significa que:

```{r}
qcases = query('cases')
# es equivalente a:
qcases = cases()
```

A lo largo de este tutorial se va a utilizar principalmente el contructor query, debido a que permite parametrizar el tipo de entidad a obtener y eso nos va a permitir una mayor reutilización del código.

A continuación podemos observar como se crea una primera consulta para cada uno de los tipos de entidades y qué campos nos devuelve por defecto cada una de ellas.

```
================================================
Fichero: \ui\tab03\tab03_02.md
================================================
Informamos la entidad seleccionada en el desplegable de la izquierda como parámetro de la función *query* comentada y asignamos su resultado a una variable que contendrá el objeto GDCQuery.

Si el comando se ejecuta con éxito este no proporcionará ningún tipo de respuesta.

Para verificar que el código funciona correctamente, tras la creación de la consulta ejecutaremos la función **default_fields()**, que nos devolverá los campos por defecto de la entidad seleccionada.

El conocimiento de estos campos será necesario en el siguiente apartado para poder entender el filtrado de los resultados obtenidos.
```

```
================================================
Fichero: \ui\tab04\tab04.R
================================================
tabPanel("Paso 4",
  h2("Agrupación de los resultados obtenidos"),
  includeMarkdown("ui/tab04/tab04_01.md"),
```

```
  sidebarLayout(position = "left",
                sidebarPanel(
                  h2("Selección"),
                  selectInput("t4_Aggregation",
                              "Campo",
                              choices = list('type', 'data_type',
'state', 'data_category'),
                              selected = 1),
                ),
                mainPanel(
                  verbatimTextOutput("t4_ResultAggregation"),
                )
  )
)
```

```
================================================
Fichero: \ui\tab04\tab04_01.md
================================================
```

Esta librería nos proporciona dos funciones para realizar consultas avanzadas y obtener resultados agregados en base a diversas características de los datos genómicos.

Éstas son especialmente útiles en el análisis de datos genómicos a gran escala, como los datos disponibles en la base de datos que nos ocupa.

Las funciones proporcionadas son *facets()* y *aggregations()*, y ambas estánpensadas para trabajar en conjunto.

Función Facets

La función *facets* permite especificar facetas sobre las que posteriormente se van a realizar las distintas agregaciones.

Las facetas son atributos o características de los datoss, como pueden ser el tipo de muestra, el estado clínico, el tipo de archivo, etc.

Al utilizar esta función, podemos especificar uno o más campos para obtener el número de registros que coinciden con cada uno de los valores potenciales.

* La función permite devolver varios campos a la vez, pero no existe una función de tabulación cruzada. Eso implica que las distintas agregaciones únicamente están en un campo a la vez.

Función Aggregations

La función encargada de realizar las distintas agrupaciones definidas como facetas, es la función *aggregations*.

La funcionalidad que dan ambas funciones trabajando en conjunto nos permiten obtener una visión general y resumida de los datos.

Ejemplo de agregación

En el siguiente ejemplo vamos a realizar una consulta para conocer la cantidad de ficheros existentes en función de los valores de los campos *type*, *data_type*, *state* y *data_category*:

```{r}
library(GenomicDataCommons)
res = files() %>%
      facet(c('type','data_type','state', 'data_category')) %>%
      aggregations()
```

Podéis utilizar el selector desplegable para seleccionar el campo del que queréis obtener el resultado de la agregación:

```
================================================
Fichero: \ui\tab05\tab05.R
================================================
tabPanel("Paso 5",
         h2("Filtrado de los resultados obtenidos"),
         includeMarkdown("ui/tab05/tab05_01.md"),
         sidebarLayout(position = "left",
                       sidebarPanel(
                         h2("Selección"),
                         selectInput("t5_QueryType",
                                     "Entidad",
                                     choices = list(),
                                     selected = 1),
                         selectInput("t5_Fields",
                                     "Campo",
                                     choices = list(),
                                     selected=1),
                         selectInput("t5_FieldValues",
                                     "Valores",
                                     choices = list(),
                                     selected=1)
                       ),
                       mainPanel(

includeMarkdown("ui/tab05/tab05_02.md"),
                         verbatimTextOutput("t5_ResultQuery"),

verbatimTextOutput("t5_ResultQueryConcatenated"),

includeMarkdown("ui/tab05/tab05_03.md"),
```

```
verbatimTextOutput("t5_SelectedValuesInformation")
                        )
         )
)
================================================
Fichero: \ui\tab05\tab05_01.md
================================================
### Filtrado de datos

Una de las funcionalidades clave de esta librería es la capacidad de filtrar los datos de interés en función de criterios específicos, como pueden ser el tipo de tumor, el tipo de muestra o el proyecto, entre otros.

En este apartado, presentaremos un ejemplo dinámico en el que se puede observar cómo se utiliza la función *filter* para filtrar los datos en la plataforma GDC según varios criterios.

**Conocimiento de los criterios de filtrado**

Es importante destacar que antes de poder utilizar la función indicada con el fin de filtrar los datos, es muy importante tener un conocimiento claro de los elementos disponibles para el filtrado. Como se ha podido observar en el paso anterior, la plataforma GDC proporciona una amplia gama de metadatos asociados a sus datos, lo que nos permite realizar filtros muy precisos.

En este tutorial nos vamos centrar en algunos de los criterios que hemos considerado más comunes, estos son:

* **Tipo de tumor**: Permite filtrar los datos en función del tipo específico de tumor. Por ejemplo, se puede filtrar los datos para
```

obtener solo aquellos relacionados con el cáncer de mama o el cáncer de colon.

* **Tipo de muestra**: Permite filtrar los datos en función del tipo de muestra biológica utilizada. Por ejemplo, se puede filtrar los datos para obtener solo aquellos que correspondan a muestras de tejido tumoral o muestras de sangre.

* **Características clínicas**: Permite filtrar los datos en función de características clínicas específicas, como la etapa del cáncer, el estado de supervivencia, entre otros.

Es importante destacar que los criterios de filtrado disponibles pueden variar según el tipo de datos y proyectos disponibles en la plataforma GDC.

Ejemplo de filtrado

Aquí tenemos un ejemplo interactivo con un filtro muy sencillo:

==========
Fichero: \ui\tab05\tab05_02.md
==========
Cómo funciona:

* Tras seleccionar uno de los tipos de entidad disponibles, se carga un desplegable con principales campos del tipo de entidad.
* Una vez seleccionado el campo, se carga un nuevo desplegable con todos los valores disponibles para ese campo.
* Si seleccionamos uno de los valores de este último campo, se nos muestra la información de la entidad resultante.

A continuación se muestra el código ejecutado en función de la selección realizada:

```
================================================
Fichero: \ui\tab05\tab05_03.md
================================================
* Los dos fragmentos de código anteriores tienen el mismo resultado, el propósito principal del segundo es mostra la utilización de la función *filter* encadenando funciones, en lugar de llamarla de forma directa.

Un dato muy importante que no podemos olvidar, es que debido a la naturaleza del componente no se pueden concatenar las llamadas a la función query, por lo que en el caso de necesitar añadir más de una condición en el filtro, éstas deben concatenarse mediante las respectivas condiciones OR (*||*) o AND (*&*).

De los dos ejemplos mostrados a continuación, únicamente podemos confiar en los datos devueltos por la primera instrucción.

**Resultado correcto**
```{r}
result = files() %>%
         filter(cases.project.project_id == 'TCGA-OV' &
                access == "open")
```
Resultado incorrecto
```{r}
result = files() %>%
         filter(cases.project.project_id == 'TCGA-OV') %>%
         filter(access == "open")
```
```

Como se puede observar, todo empieza con la creación de una consulta con la función *query* que se ha explicado en el punto anterior.
Una vez creada la consulta, filtramos los datos resultantes mediante la función filter a la cual le añadimos la condición esperada como parámetro.
Para poder mostrar los resultados obtenidos, le enviamos el objeto *myQuery* a la función *results_all()* que se encarga de obtener todos los valores resultantes.

Hay que tener en cuenta que la obtención de todos los valores puede tener un coste muy elevado, tanto a nivel de proceso como de almacenamiento, para ello, el módulo también nos proporciona la función *results()* a la que se le puede pasar como parámetro la una cantidad máxima de elementos a devolver.

A continuación se muestran los resultados de la ejecución del código anterior generado en base a los elementos seleccionados:

```
================================================
Fichero: \ui\tab06\tab06.R
================================================
tabPanel("Paso 6",
         h2("Descarga de datos"),
         includeMarkdown("ui/tab06/tab06_01.md")
)
================================================
Fichero: \ui\tab06\tab06_01.md
================================================
#### Posibles intereses para descargar los datos
```

Descargar los datos de GDC en lugar de trabajar en línea con ellos puede tener varios intereses académicos significativos:

* **Acceso a datos completos:** Es posible que las plataformas de análisis de datos tengan restricciones que limiten la cantidad de datos que se pueden visualizar o descargar a la vez. Al descargarlos por partes, los investigadores pueden tener un acceso completo y sin restricciones a todos los datos relevantes para sus investigaciones.
* **Flexibilidad en el análisis:** Trabajar con datos descargados de GDC brinda a los investigadores una mayor flexibilidad de análisis y procesamiento de los mismos. Éstos pueden utilizar herramientas y paquetes de análisis de datos de su elección, lo que les permite personalizar y adaptar sus análisis de acuerdo a necesidades específicas. Además, pueden aprovechar el poder de procesamiento de sus propias máquinas para realizar análisis intensivos en computación.
* **Seguridad y privacidad de los datos:** Descargar los datos de GDC y trabajar con ellos en un entorno local puede proporcionar un mayor nivel de seguridad y privacidad de los datos. Los investigadores pueden implementar medidas de seguridad adicionales para proteger los datos confidenciales y garantizar el cumplimiento de las regulaciones y políticas de privacidad de datos en sus instituciones académicas. Esto es particularmente relevante cuando se trabajan con datos clínicos o genómicos sensibles.
* **Reproducibilidad y reutilización:** Al descargar los datos de GDC, los investigadores pueden guardar y preservar los conjuntos de datos originales utilizados en sus investigaciones. Esto facilita la reproducción de los resultados y la reutilización esos mismos datos en investigaciones futuras o para colaboraciones con otros investigadores.
* **Exploración y experimentación sin conexión:** Trabajar con datos descargados de GDC permite a los investigadores explorar y experimentar con los datos sin necesidad de estar en línea. Esto

puede resultar especialmente beneficioso en situaciones donde la conectividad a Internet es limitada o inestable.

Función para la descarga de datos

La descarga de ficheros desde la base de datos del GDC se realiza mediante el identificador único *UUID* de cada uno de ellos y posteriormente se renombran una vez descargados.

Es importante tener en cuenta que los ficheros se descarán en el directorio indicado por la función *gdc_cache()* no pueden existir en el destino un fichero con el mismo *UUID* ni el posterior nombre que se le asociará.

La funcionalidad de descarga viene proporcionada por la función *gdcdata* que viene definida en la documentación de la siguiente manera:

```{r}
gdcdata(uuids, use_cached = TRUE, progress = interactive(),
  token = NULL, access_method = "api", transfer_args = character())
```

parámetro	descripción
uuids	Una lista con los identificadores únicos de los ficheros a descargar
use_cached	Indica que si el fichero ya se ha descargado previamente, no se vuelva a descargar
progress	Indica si se debe mostrar el progreso de la descarga

token	(Opcional) permite especificar el token de conexión para acceder a ficheros restringidos
access_method	Indica el método de conexión a la base de datos, '*api*' o '*client*'
transfer_args	Permite especificar otros argumentos opcionales que puede aceptar la función

---

Antes de descargar los datos de GDC, es común realizar filtrados para obtener un subconjunto específico de datos que sean de interés para nuestro análisis. Como hemos podido comprobar en los pasos anteriores, el módulo de conexión a GDC nos ofrece una amplia gama de filtros basados en una gran cantidad de criterios.

Ejemplo de descarga

Supongamos que deseamos descargar los datos de expresión génica de tumores de cáncer de mama, a continuación presentamos un ejemplo paso a paso de utilización de la función:

```{r}
library(GenomicDataCommons)

# Definimos la consulta
myQuery <- query('files') %>%
            # Empezamos cogiendo aquellos archivos de acceso libre
            filter(access == 'open') %>%
            # Nos ceñimos a los archivos del proyecto TCGA-BRCA que está enfocado en este tipo de enfermedad
            filter(cases.project.project_id == "TCGA-BRCA" &
                   # Seleccionamos la categoría de datos que queremos descargar
```

```
                  data_category == "Transcriptome Profiling" &
                  # Filtramos por el tipo de experimentación
utilizada
                  experimental_strategy == "RNA-Seq" &
                  # Filtramos por la zona primaria de la
enfermedad
                  cases.project.primary_site == "Breast" &
                  # Filtramos por la enfermedad concreta
                  cases.project.disease_type == "Breast Invasive
Carcinoma" &
                  # Seleccionamos dos de los posibles tipos de
muestras
                  cases.samples.sample_type    ==    c("Primary
Tumor", "Solid Tissue Normal")) %>%
            # Aplicamos un límite de 3 archivos a descargar
            results(size=3) %>%
            # Seleccionamos los IDs de los archivos a descargar.
            ids()

# Una vez realizada la consulta, solicitamos su descarga,
indicando que si ya se han descargado
# se utilicen los existentes.
myData <- gdcdata(myQuery, use_cached = TRUE)

# Ejecutamos la función print para mostrar los ficheros
descargados.
print(myData)
```

En el ejemplo anterior, hemos definido el filtro de proyecto utilizando el código del proyecto "*TCGA-BRCA*" que se refiere al estudio del cáncer de mama en el proyecto del Atlas del Genoma del Cáncer (TCGA).

Además, hemos aplicado filtros adicionales para seleccionar únicamente las muestras de tumores primarios y tejido normal.

```
================================================
Fichero: \ui\tab07\tab07.R
================================================
tabPanel("Paso 7",
         h2("Análisis integrado de datos"),
         includeMarkdown("ui/tab07/tab07_01.md")
)
================================================
Fichero: \ui\tab07\tab07_01.md
================================================
```

En este punto vamos a tratar la integración los datos de GDC con otros tipos de datos, como datos clínicos o de expresión génica, para obtener una imagen más completa del conjunto de datos

Sabemos que el GDC es una plataforma centralizada que almacena y distribuye datos genómicos y clínicos recopilados de estudios a gran escala, no obstante, en muchas ocasiones puede ser necesario llevar a cabo análisis integrados, donde se combinen los datos genómicos con otros tipos de información relevante, como datos clínicos y de expresión genética.

Este punto tiene como objetivo proporcionar una guía detallada sobre cómo realizar este análisis integrado utilizando la librería GenomicDataCommons:

a) Preparación de los datos

Antes de iniciar el análisis integrado, es necesario preparar los datos de manera adecuada. En primer lugar, debemos obtener los datos genómicos de GDC utilizando las funciones de la librería

GenomicDataCommons que se han visto en detalle en los puntos anteriores, especificando los correspondientes criterios de búsqueda, como el tipo de datos, el proyecto, etc. Una vez descargados los datos genómicos, se procede a obtener los datos clínicos correspondientes, que pueden estar disponibles en la misma base de datos o en otras fuentes.

b) Exploración y limpieza de datos

Una vez que se han obtenido los datos genómicos y clínicos, es esencial llevar a cabo una exploración exhaustiva para comprender su estructura y contenido.
Esto implica los siguentes puntos:
* Examinar las variables disponibles
* Identificar posibles valores atípicos o faltantes
* Comprender las relaciones entre las diferentes variables

Además de lo anterior, se deben realizar las correspondientes tareas de limpieza y estandarización de datos.

c) Integración de datos

La integración de datos es el proceso de combinar los datos genómicos y clínicos en un conjunto de datos coherente.

Para lograr esto, es necesario identificar una variable común que actúe como clave de unión entre los dos conjuntos de datos, esta puede ser el identificador de muestra, un paciente único, etc.

A través de esta variable común, se pueden combinar los datos utilizando técnicas de fusión de datos disponibles en R, como la función *merge()*.

d) Análisis integrado

Una vez que los datos genómicos y clínicos están integrados en un solo conjunto de datos, es posible realizar un análisis integrado más exhaustivo.

Esto implica explorar las relaciones entre las variables genómicas y clínicas para obtener una comprensión más profunda de los mecanismos moleculares subyacentes a fenotipos clínicos específicos.

Un par de ejemplos de ello sería llevar a cabo análisis de asociación para identificar genes candidatos asociados a una enfermedad en particular o correlaciones entre características clínicas y expresión génica.

e) Visualización de resultados

Una parte crucial del análisis integrado es la visualización de los resultados obtenidos.

La visualización efectiva permite comunicar de manera clara y concisa los hallazgos del análisis a través de gráficos, tablas y otros elementos visuales.

En R, existen numerosas librerías, como ggplot2, que facilitan la creación de visualizaciones de calidad para resaltar las relaciones y patrones identificados en los datos integrados.

```
================================================
Fichero: \ui\tab08\tab08.R
================================================
tabPanel("Paso 8",
         h2("Análisis exploratorio de datos"),
```

```
            includeMarkdown("ui/tab08/tab08_01.md"),
            verbatimTextOutput("t8_ProjectsByProgram"),
            includeMarkdown("ui/tab08/tab08_02.md"),
            verbatimTextOutput("t8_ProjectsInTCGA"),
            includeMarkdown("ui/tab08/tab08_03.md"),
            verbatimTextOutput("t8_ProjectTCGAInfo"),
            includeMarkdown("ui/tab08/tab08_04.md"),
            column(4,
              verbatimTextOutput("t8_SampleTypeInTCGA")
            ),
            column(8,
               plotOutput("t8_Plot_SampleTypeInTCGA")
            ),
            includeMarkdown("ui/tab08/tab08_05.md"),
            column(3,
               verbatimTextOutput("t8_DemographicTCGA")
            ),
            column(3,
               plotOutput("t8_Plot_DemographycRace")
            ),
            column(3,
               plotOutput("t8_Plot_DemographycGender")
            ),
            column(3,
               plotOutput("t8_Plot_DemographycEthnicity")
            ),
            includeMarkdown("ui/tab08/tab08_06.md"),
            verbatimTextOutput("t8_HowManyOtherBreastProjects"),
            includeMarkdown("ui/tab08/tab08_07.md"),
            verbatimTextOutput("t8_IDsIfOtherBreastProjects"),
            includeMarkdown("ui/tab08/tab08_08.md"),
            verbatimTextOutput("t8_NumberOfCases")
)
```

```
================================================
Fichero: \ui\tab08\tab08_01.md
================================================
```
El análisis exploratorio de datos es una etapa fundamental en la investigación genómica, ya que nos permite comprender la estructura y características de los datos antes de realizar análisis más avanzados.

En este apartado, vamos a aplicar los conceptos detallados en los puntos anteriores para obtener respuestas a algunas preguntas de ejemplo con el fin de obtener información relevante de los datos contenidos en la base de datos del GDC y ver como hemos llegado hasta ellos.

1) ¿Cuántos proyectos tiene cada uno de los programas existentes en GDC?

```{r}
res = projects() %>%
      facet("program.name") %>%
      aggregations()
head(res)
```

```
================================================
Fichero: \ui\tab08\tab08_02.md
================================================
```
Podemos observar que el programa TCGA tiene muchos más proyectos que el resto, por lo que vamos a centrar nuestra investigación en él.

2) ¿Cuántos casos tiene registrado cada uno de los proyectos del programa TCGA?

```{r}
res = cases() %>%
      filter(project.program.name == "TCGA") %>%
      facet("project.project_id") %>%
      aggregations()

head(res)
```

==========
Fichero: \ui\tab08\tab08_03.md
==========

Podemos observar que el proyecto TCGA-BRCA tiene muchos más casos que el resto, así que nos vamos a centrar en él.

3) ¿De qué trata exactamente el proyecto TCGA?

```{r}
res = projects() %>%
      filter(project_id == "TCGA-BRCA") %>%
      results_all()
res
```

==========
Fichero: \ui\tab08\tab08_04.md
==========

Podemos observar que el proyecto se dedica principalmente a la investigación del cáncer de pecho.

4) ¿De dónde proceden las distintas muestras asociadas al proyecto?

```{r}
res = cases() %>%
      filter(project.project_id == "TCGA-BRCA") %>%
      facet("samples.sample_type") %>%
      aggregations()
res
```

==========
Fichero: \ui\tab08\tab08_05.md
==========

5) ¿A qué género, raza y etnia pertenecen las muestras del proyecto?

```{r}
res = cases() %>%
      filter(project.project_id == "TCGA-BRCA") %>%
      facet(c("demographic.race",          "demographic.gender",
"demographic.ethnicity")) %>%
      aggregations()
res
```

==========
Fichero: \ui\tab08\tab08_06.md
==========

\
\

6) ¿Cuántos proyectos hay con muestras de cáncer de pecho, además del TCGA-BRCA?

```{r}
res = projects() %>%
```

```
      filter(primary_site == "Breast") %>%
      count()
res
```

```
================================================
Fichero: \ui\tab08\tab08_07.md
================================================
### 7) ¿Cuáles serían los códigos de estos proyectos?

```{r}
res = projects() %>%
      filter(primary_site == "Breast") %>%
      results_all()
res$project_id
```

```
================================================
Fichero: \ui\tab08\tab08_08.md
================================================
### 8) ¿Cuántos casos relacionados tiene cada uno de ellos?

```{r}
res = cases() %>%
      filter(project.primary_site == "Breast") %>%
      facet("project.project_id") %>%
      aggregations()
res
```

```
================================================
Fichero: \ui\tab_08\tab08.R
================================================
tabPanel("Paso 8",
         h2("Análisis de expresión génica")
```

```
)
================================================
Fichero: \ui\tab_09\tab09.R
================================================
tabPanel("Paso 9",
         h2("Análisis de mutaciones")

)
================================================
Fichero: \ui\tab_10\tab10.R
================================================
tabPanel("Paso 9",
         h2("Análisis de datos de metilación")

)
```