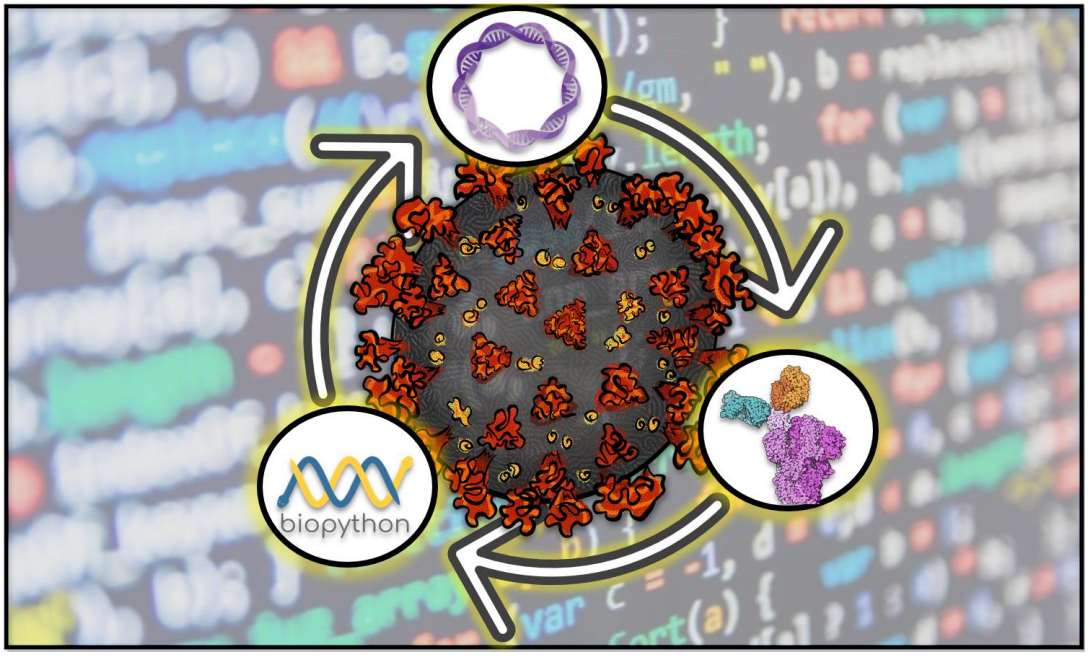
UNIVERSITAT ROVIRA i VIRGILI

DESARROLLO DE HERRAMIENTAS INFORMÁTICAS PARA EL ANÁLISIS DE SARS-COV-2 MEDIANTE PYTHON.

Pedro Manuel López Zarzuela

TRABAJO DE FINAL DE GRADO BIOTECNOLOGÍA



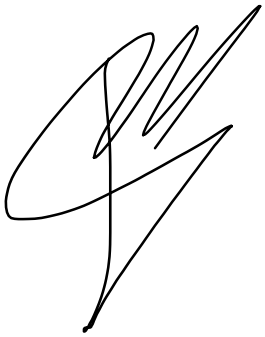
Tutor académico: Dr. Lluis Masip Vernis, Grado de Ingeniería Química, Departamento de Ingeniería Química, lluis.masip@urv.cat.

Fecha de convocatoria: Septiembre 2022

ÍNDICE

Yo, Pedro Manuel López Zarzuela, con DNI 20549193-G, soy conocedor de la guía de prevención del plagio en la URV Prevención, detección y tratamiento del plagio en la docencia: guía para estudiantes (aprobada el julio 2017) (http://www.urv.cat/ca/vidacampus/serveis/crai/que-us-oferim/formacio-competencies-nuclears/plagi/) i afirmo que este TFG no constituye ninguna de las conductas consideradas como plagio por la URV.

Tarragona, 1 de Septiembre de 2022



Abreviaciones

-PAM (del inglés, *Pointed Accepted Mutation*)

-GISAID (del inglés, *Global Initiative on Sharing All Influenza Data*)

-NCBI (por sus siglas en inglés, *National Center for Biotechnology*)

-PDB (del inglés, *Protein Data Bank*)

-MERS (por sus siglas en inglés, *Middle East Respiratory Syndrome*)

-SARS (del inglés, *Severe Acute Respiratory Syndrome*)

-COVID-19 (por sus siglas en inglés, *COronaVIrus Disease 2019*)

-ADN (Ácido Desoxirribonucleico) ARN (Ácido Ribonucleico)

-sgARN (ARN subgenómico)

-ssRNA (del inglés, *single strand RNA*)

-MUSCLE (del inglés, *Multiple Sequence Comparison por Log-Expectation*)

-ACE2 (receptores de la enzima convertidora de angiotensina 2)

-RBD (del inglés, *Receptor Binding Domain*)

-NTD (del inglés, *N-Terminal Domain*)

-ORF (del inglés, *Open Reading Frame*)

-Prody (por sus siglas en inglés, *Protein Dynamics & Sequence Analysis*)

-ExPASy (por sus siglas en inglés, *Expert Protein Analysis System*)

-CIF (por sus siglas en inglés, *Crystallographic Information File*).

-MSA (del inglés, *Multiple Sequence Alignment*)

1 Resumen

Introducción: La bioinformática es una herramienta de aproximación a la comprensión biológica, que hace uso de datos e información multidisciplinares, en los que pueden incluir resultados de ensayos experimentales o clínicos. Este campo comprende retos complejos como el estudio de las enfermedades raras de origen genético, la identificación de las mutaciones asociadas a tumores, la identificación del patógeno causante de un brote infeccioso o el descubrimiento de nuevos virus, como el SARS-CoV-2.

Objetivos: El objetivo principal de este estudio es desarrollar un programa que analice las secuencias genómicas y proteicas de las diferentes variantes del SARS-CoV-2. El programa implementa algoritmos de análisis de secuencias biológicas capaces de descargar y procesar los archivos de las principales bases de datos en línea.

Materiales y Métodos: Para el desarrollo del programa se utilizó Python, por sus características de accesibilidad, legibilidad y modularidad. Como herramientas de desarrollo principal se usaron diferentes módulos y librerías orientadas a la bioinformática o al procesamiento de datos como: Biopython, Prody, Matplotlib o Numpy.

Resultados: Coronalyzer cuenta con funcionalidades para comprender la estructura y funcionamiento del virus SARS-CoV-2. Las funciones de comparación, análisis de secuencia y de análisis filogenético proporcionan información sobre la evolución y origen del virus. Por otro lado, el análisis de mutaciones, junto a las herramientas de alineación y modelado nos permiten monitorear cambios en los diferentes genes de las variantes de SARS-CoV-2 y ver que implicaciones tienen sobre las proteínas principales que conforman el virus.

Conclusión: Las bases de datos en línea en conjunto con los lenguajes de programación y módulos usados durante el trabajo hacen accesibles los estudios en biología molecular, filogenética, modelado de proteínas y secuenciación, facilitando la comprensión de patógenos como el SARS-CoV-2.

Palabras clave: Bioinformática; SARS-CoV-2; Python; análisis de secuencia; análisis filogenético; modelado de proteínas.

2 Introducción

2.1 Bioinformática y SARS-CoV-2

La bioinformática nació a comienzos de 1960 con la aplicación de métodos computarizados al análisis de la secuencia de proteínas. Margaret Dayhoff, pionera en este campo, publica el primero de los *Atlas of Protein Sequences* en 1965, y algunos años más tarde, desarrolló las matrices de sustitución PAM, precursoras de las actuales bases de datos de proteínas. El crecimiento de este campo fue ligado al desarrollo de la biología molecular, el descubrimiento del ADN y a los avances en computación. En la actualidad la bioinformática puede definirse como la aplicación de tecnologías computacionales y la estadística a la gestión y análisis de datos biológicos, siendo indispensable en aquellos campos donde se trabaja con un gran volumen de datos ("ómicas", como la genómica, proteómica o la metabolómica).

Uno de los proyectos de relevancia en este campo fue la secuenciación del genoma humano (2004), que tuvo grandes implicaciones en biomedicina y medicina genética (estudio de las enfermedades raras de origen genético, la identificación de las mutaciones asociadas a tumores…). Además, la bioinformática se ha centrado en la identificación de patógenos causantes de enfermedades infecciosas o el descubrimiento de nuevos virus, como el SARS-CoV-2.

El SARS-CoV-2 surgió a fines de 2019 y se expandió a nivel mundial, lo que resultó en más de 560 millones de casos confirmados a fines de 2022 (Home - Johns Hopkins Coronavirus Resource Center, n.d.). El virus recibió su nombre debido a su similitud con un conocido coronavirus previamente identificado SARS-CoV-1, y desde que fue identificado por primera vez en Wuhan, se han estudiado más de 500 coronavirus diferentes, de los cuales se sabe que 7 causan infecciones en humanos: SARS-CoV-1, SARS-CoV-2, MERS, HKU1, NL63, OC43 y 229E.

Una de las herramientas informáticas utilizadas durante la pandemia para el análisis filogenético del SARS-CoV-2 fue MUSCLE (*Multiple Sequence Comparison por Log-Expectation*), un algoritmo diseñado para el alineamiento múltiple de secuencias. Los resultados del análisis filogenético con la herramienta bioinformática MUSCLE demostraron que el SARS-CoV-2 tiene una identidad de secuencia del 96,3% con los coronavirus que infectan a murciélagos, BatCoV y RaTG13, ofreciendo una mejor perspectiva sobre el origen del virus (Lu et al., 2020). Los estudios con herramientas informáticas como MUSCLE son esenciales para monitorear la evolución del virus,

así como los estudios de epidemiología molecular, que han generado más de 12.028.363 secuencias genómicas virales, compartidas con una velocidad sin precedentes a través de la iniciativa GISAID (GISAID - Contact, n.d.). GISAID es una iniciativa de ciencia que proporciona acceso abiertos a datos genómico del virus influenza y el coronavirus responsable de la pandemia de COVID-19. De esta manera la bioinformática se convirtió en una aliada en la lucha contra la Covid-19, ya que es clave para vigilar las variantes más agresivas: Alpha, Beta, Gamma, Delta y Omicron.

La secuenciación es una herramienta imprescindible para el seguimiento y la lucha contra agentes infecciosos como el SARS-CoV-2 (Ma et al., 2021). El 10 de enero de 2020, estuvieron disponibles las primeras secuencias del genoma completo de SARS-CoV-2 en GISAID, que habilitó respuestas globales a la pandemia, incluyendo el desarrollo de las primeras vacunas y pruebas de diagnóstico para detectar SARS-CoV-2 (Figura 1).

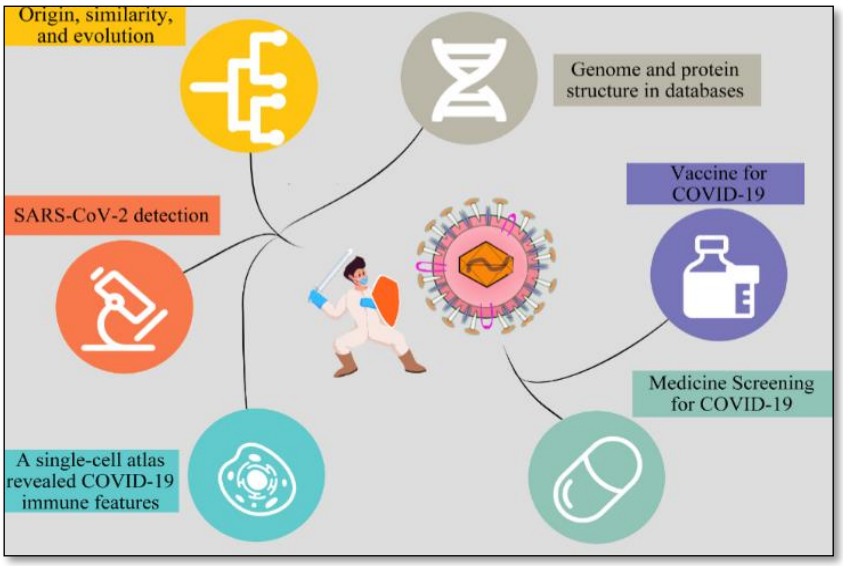


Figura 1. El flujo de trabajo de los métodos y aplicaciones más importantes de las tecnologías bioinformáticas sobre COVID-19. Fuente: Ma et al (2021).

2.2 Estructura del SARS-CoV-2

El SARS-CoV-2 es un virus esférico de 100-160 nm de diámetro, envuelto por una bicapa lipídica y que contienen ARN monocatenario (ssRNA) de polaridad positiva. La espiga S, es la responsable de la entrada viral por endocitosis. El virus ingresa a las células a través de la interacción del dominio de unión al receptor viral (RBD) con los receptores de la enzima convertidora de angiotensina 2 (ACE2), que se encuentran en la superficie de la mucosa nasofaríngea, pulmonar e intestinal humana.

Por este motivo la proteína S es uno de los *targets* a estudiar, para ser neutralizada por los anticuerpos generados por infección o por la vacunación, así como por medicamentos de anticuerpos monoclonales (Yang et al., 2021).

Los esfuerzos de desarrollo de vacunas y medicamentos se dirigen a estructuras y/o proteínas virales específicas. Es muy importante que comprendamos como afectan las mutaciones virales antes de que se decida tal objetivo. Si la estructura de la proteína objetivo sufre una mutación, entonces el medicamento o vacuna puede no ser efectivo en el futuro. Aquí es donde el análisis de mutaciones juega un papel crítico en los esfuerzos para crear un medicamento o vacuna.

El genoma del SARS-CoV-2 tiene un tamaño aproximado de 30kb, contiene 14 marcos de lectura abiertos (ORF) y codifica 29 proteínas virales. Entre los diferentes marcos de lectura encontramos: ORF1ab, S, ORF3a, ORF3b, E, M, ORF6, ORF7a, ORF7b, ORF8, ORF9a, ORF9b, N y ORF10 (Figura 2).

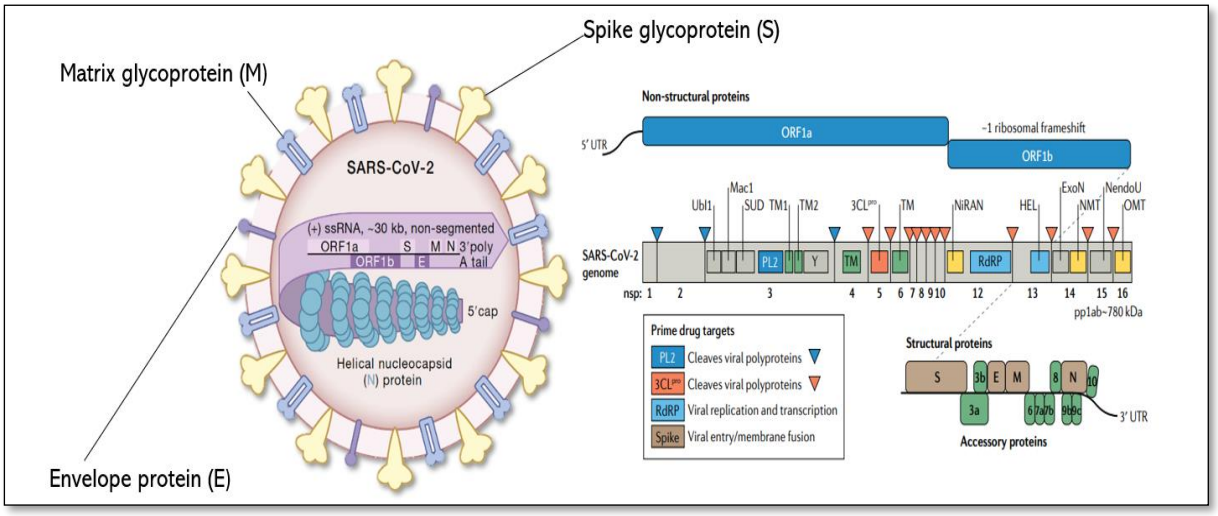


Figura 2. Estructura SARS-CoV-2. Fuente: Adaptación Yang et al. (2021).

El genoma de SARS-CoV-2 se puede dividir en tres tercios. Los dos primeros tercios codifican para el gen de la replicasa viral. Este gen está constituido por ORF1a y ORF1b, los que, al comienzo de la infección, serán traducidos directamente en dos poliproteínas que posteriormente serán procesadas proteolíticamente para generar 16 proteínas no estructurales implicadas en la replicación del genoma viral y en la transcripción de ARNm subgenómicos (sgARNs). El último tercio del genoma, en el extremo 3', codifica los genes de las 4 proteínas estructurales principales (S, M, E, N)

y lo genes de las proteínas accesorias (ORF3a, ORF3b, ORF6, ORF7a, ORF7b, ORF8b, ORF9b y ORF10).

2.3 Seguimiento y evolución del virus

El seguimiento y análisis de las proteínas estructurales del virus ha sido gran interés. La monitorización de las mutaciones es necesaria para confirmar que alteraciones en el genoma del virus repercuten sobre la inmunidad o la infectividad.

Existen varios mecanismos distintos por los cuales las mutaciones pueden alterar las propiedades antigénicas de una glicoproteína como la proteína S:

- **Sustituciones de aminoácidos que alteran el epítopo:** Un cambio en las propiedades biofísicas de un residuo de epítopo disminuye directamente la unión de anticuerpos.
- **Aumento de la avidez de unión al receptor:** Las sustituciones que aumentan individualmente la afinidad de unión al receptor pueden cambiar el equilibrio de unión entre la glicoproteína y los anticuerpos neutralizantes en favor de una interacción de mayor avidez entre la glicoproteína y el receptor celular.
- **Cambios en la glicosilación:** Los glicanos son moléculas de azúcar voluminosas que pueden proteger a los epítopos de la unión de anticuerpos. Una sustitución puede introducir un motivo de glicosilación adicional.
- **Deleciones e inserciones:** La deleción o inserción de residuos tiene el potencial de alterar la conformación del epítopo, disminuyendo la unión de anticuerpos.
- **Efectos estructurales alostéricos:** La sustitución de aminoácidos puede afectar la unión de anticuerpos al cambiar la conformación de la proteína de tal manera que un epítopo se altera o se muestra de manera diferente.

Entre las diferentes mutaciones encontradas en la proteína S, la D614G confiere una mayor infectividad y transmisibilidad. La mutación reduce la compactación del trímero de la proteína y favorece el estado abierto del dominio de unión al receptor. El aminoácido 614 se encuentra fuera del dominio de unión al receptor de S, por lo que es poco probable que esta variante ponga en peligro la inmunidad natural o mediada por anticuerpos derivada de la vacuna (Thomson et al., 2021). En el extremo N-terminal de la proteína S (NTD) la sustitución de aminoácido N501 aumenta la afinidad de unión a ACE2 y los cambios en los enlaces disulfuro reducen la unión de múltiples

anticuerpos monoclonales (Harvey et al., 2021). Podemos visualizar las regiones mencionadas en la figura 3.

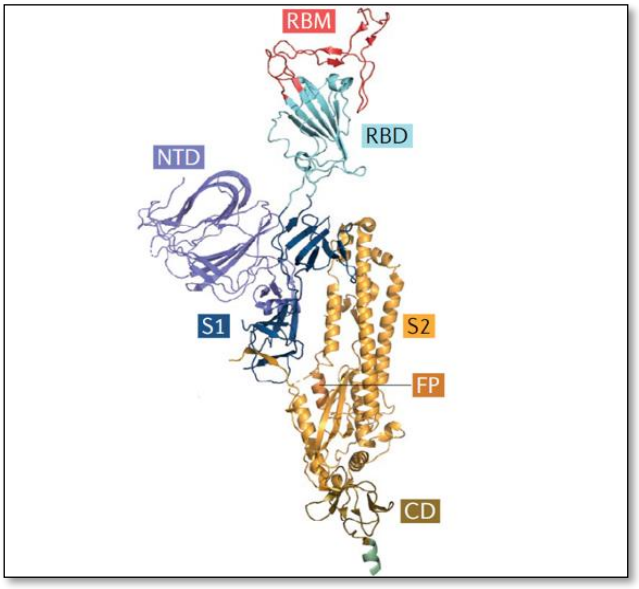


Figura 3. Monómero proteína S. Fuente: Harvey et al. (2021).

2.4 Software orientado a la bioinformática

Los recursos de secuenciación están evolucionando muy rápidamente y la simple alineación de secuencias de virus u otros organismos consume pocos recursos y se puede hacer utilizando bases de datos en línea. Entre los diferentes lenguajes de programación usados en el sector de la bioinformática Python es un lenguaje de programación de muy alto nivel, con un uso comercial y académico generalizado (Shajii et al., 2021).

Biopython es otra de las herramientas imprescindibles en este campo, con una colección de módulos destinados a los programadores de biología molecular computacional o bioinformática. Su licencia es compatible con la mayoría de licencias de código abierto, de manera que puede ser utilizado para crear una gran variedad de proyectos informáticos. Este módulo permite usar scripts o incorporar código en su propio software. Entre las diversas funciones que se pueden realizar utilizando Biopython encontramos: acceso a base de datos como NCBI, GenBank, Swiss-Prot, ExPasy o PDB, la reconstrucción de árboles filogenéticos, el análisis de secuencias de aminoácidos y nucleótidos, así como el modelado 3D de proteínas, entre otras muchas funciones.

3 Hipótesis y Objetivos

La hipótesis en la que se basa el presente estudio defiende que el uso de herramientas informáticas como Python y Biopython facilitan el análisis y comprensión de agentes patógenos como el SARS-CoV-2. Para validar dicha hipótesis, el objetivo principal del trabajo ha sido explorar las diferentes herramientas informáticas que hay disponibles en el mercado y desarrollar un programa que analice las secuencias genómicas y proteicas de las variantes de SARS-CoV-2. El objetivo específico del estudio es aprender a implementar herramientas informáticas orientadas a la biología molecular. El programa consta con las siguientes características.

- Descarga de archivos de las bases de datos biológicas.
- Análisis de secuencia.
- Comparación de secuencias genómicas.
- Detección de mutaciones de secuencia.
- Comparación de secuencias proteicas.
- Generación de gráficos de proteínas.
- Visualización 3D proteínas.
- Procesamiento de árboles filogenéticos.

Coronalyzer proporcionar una plataforma de análisis bioinformático completo, con una interfaz gráfica sencilla para facilitar un uso generalizado en tareas de secuenciación y modelado.

4 Metodología

Para el desarrollo de un programa informático orientado al tratamiento de secuencias de SARS-CoV-2 se realizó una búsqueda previa de los lenguajes más extendidos y utilizados en este campo. Dadas las características del lenguaje, accesibilidad, legibilidad y modularidad se escogió Python como principal lenguaje de desarrollo. Una vez escogido el lenguaje, entre las distribuciones existentes en el mercado, Anaconda ofrece gran variedad de entornos orientados a la computación científica como Jupyterlab o Spyder, usados en el proyecto. Para la búsqueda bibliográfica se optó por la plataforma Pubmed y su herramienta Advanced search con las palabras clave "SARS-CoV-2" y "Bioinformatics" para todos los campos. Se filtró para hacer una búsqueda de artículos publicados los últimos 5 años. De los 4544 resultados se

escogieron los más adecuados para el proyecto. Durante la búsqueda bibliográfica se encontraron publicaciones que mencionaban Biopython (Adam, 2021), entre otras herramientas orientadas al campo de la bioinformática. La recopilación de datos, artículos y manuales de programación en línea sirvieron de base para iniciar la etapa de desarrollo.

4.1 Bases de Datos

Uno de los pasos a seguir en bioinformática es la extracción de información de las bases de datos biológicas (Tabla 1). Biopython facilita todo este proceso haciendo que algunas bases de datos en línea estén disponibles a partir de scripts de Python.

BASES DE DATOS GENÓMICAS Y PROTEICAS	
Tipo	**Base de datos**
Genómica	Global Initiative on Sharing Avian Influenza Data (GISAID)
Genómica	National Center of Biotechnology Information (NCBI)
Genómica	National Microbiology Data Center (NMDC)
Genómica	China National Genbank y China National Center for Bioinformationn (CNGB y CNCB)
Proteica	Universal Protein (UniProt)
Proteica	Research Collaboratory for Structural Bioinformatics Protein Data Bank Database e (RCSB PDB)

Tabla 1. Bases de datos SARS-CoV-2.

4.2 Módulos y Entorno

Del módulo Biopython utilizaremos las siguientes librerías de funciones:

- **Bio.SeqIO & Bio.AlignIO:** Son dos librerías de funciones para leer y escribir formatos de archivo de secuencia. Estas tiene como objetivo proporcionar una interfaz simple para trabajar con formatos de archivo de secuencia variados de una manera uniforme. Bio.SeqIO trabaja con archivos de una única secuencia, mientras que Bio.AlignIO trabaja con archivos que contienen alineamientos de secuencias.

- **Bio.Entrez:** Hace uso de las Utilidades de Programación de Entrez, un sistema de recuperación de datos que proporciona a los usuarios acceso a las bases de datos de NCBI, como PubMed, GenBank y GEO entre otras.
- **Bio.pairwise2:** Contiene los mismos algoritmos de alineamiento *water* (local) y *needle* (global) de la suite bioinformática EMBOSS, empleada fundamentalmente en biología molecular y genética.
- **Bio.pdb:** Esta librería de Biopython se centra en trabajar con estructuras cristalinas de macromoléculas biológicas. Incluye la función PDBList para descargar archivos de la base de datos el PDB y las funciones PDBParser y MMCIFParser que se usan para acceder a los datos atómicos de los archivos del PDB.
- **Bio.ExPASy:** Es una librería de acceso a la base de datos UniProt, un repositorio online de proteínas que capaz de proporcionar secuencias procesables por el resto de librerías.
- **Bio.Phylo:** Esta librería proporciona una forma común de trabajar con árboles filogenéticos independientemente del formato de datos de origen.

El resto de módulos que completan el programa son:

- **Tkinter**: Es una herramienta para desarrollar aplicaciones de escritorio multiplataforma con una interfaz gráfica para sistemas operativos Windows, Linux, Mac y otros.
- **Prody** (*Protein Dynamics & Sequence Analysis*): Es un módulo de Python gratuito y de código abierto orientado al estudio la estructura de proteínas, la dinámica y el análisis de secuencias (Zhang et al., 2021).
- **Matplotlib**: Este módulo se utiliza para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy.
- **NumPy**: Es un módulo para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales.
- **NGLView**: Es un módulo preinstalado en el entorno Jupyterlab que permite mostrar estructuras moleculares con una vista interactiva 3D.

Como entorno de desarrollo principal se usó Jupyterlab, el último entorno de desarrollo interactivo basado en web. Su interfaz flexible permite a los usuarios

configurar y organizar flujos de trabajo en ciencia de datos, computación científica y aprendizaje automático.

4.3 Código

Para entender la estructura y funcionamiento del programa a continuación se nos muestra la descripción de sus funciones principales (Figura 4).

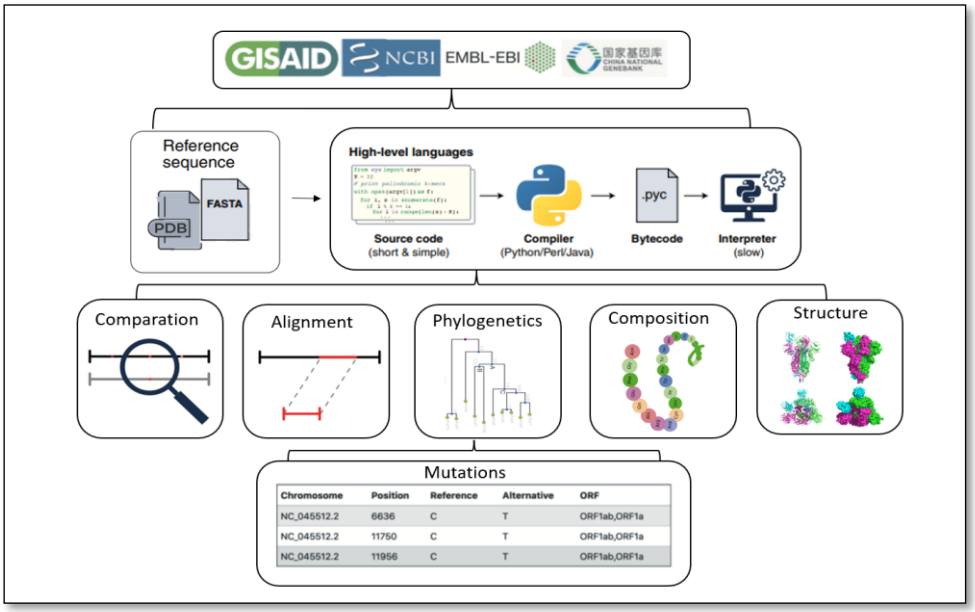


Figura 4. Diagrama de funcionamiento del programa. Fuente: Adaptación.

Función Descargar FASTA y PDB: Descarga los archivos FASTA conjuntamente con los CDS nucleotide FASTA introduciendo el código de registro del GenBank (NCBI). La diferencia entre ambos es que el primero contiene la secuencia completa del genoma sin interrupciones y el otro divide la secuencia en los diferentes genes. Estos archivos son descargados usando la librería Bio.Entrez del módulo Biopython. También descarga archivos del *Protein Data Bank* mediante la función PDBList de la librería Bio.pdb. Se descargan los archivos en su formato PDB y mmCIF (*Crystallographic Information File*).

Función Análisis de secuencia: Analiza la secuencia de un archivo FASTA previamente. A partir de la secuencia se obtiene: Datos de los nucleótidos, la secuencia de ARN, la secuencia proteica, datos de los aminoácidos, los *Open Reading Frames* y las proteínas funcionales. Algunas de las funcionalidades usadas de la librería Bio.SeqIO son: Seq.read, Seq.count, Seq.transcribe, Seq.translate, Seq.reverse_complement y Seq.split. La función Seq.translate hace uso de la tabla de codones correspondiente a bacterias, arqueas y virus procariotas, que es la más

adecuada a la hora de traducir el genoma del SARS-CoV-2 (Figura 5). El módulo Matplotlib se usa para generar gráficas a partir de los datos obtenidos.

TTT F Phe	TCT S Ser	TAT Y Tyr	TGT C Cys
TTC F Phe	TCC S Ser	TAC Y Tyr	TGC C Cys
TTA L Leu	TCA S Ser	TAA * Ter	TGA * Ter
TTG L Leu i	TCG S Ser	TAG * Ter	TGG W Trp
CTT L Leu	CCT P Pro	CAT H His	CGT R Arg
CTC L Leu	CCC P Pro	CAC H His	CGC R Arg
CTA L Leu	CCA P Pro	CAA Q Gln	CGA R Arg
CTG L Leu i	CCG P Pro	CAG Q Gln	CGG R Arg
ATT I Ile i	ACT T Thr	AAT N Asn	AGT S Ser
ATC I Ile i	ACC T Thr	AAC N Asn	AGC S Ser
ATA I Ile i	ACA T Thr	AAA K Lys	AGA R Arg
ATG M Met i	ACG T Thr	AAG K Lys	AGG R Arg
GTT V Val	GCT A Ala	GAT D Asp	GGT G Gly
GTC V Val	GCC A Ala	GAC D Asp	GGC G Gly
GTA V Val	GCA A Ala	GAA E Glu	GGA G Gly
GTG V Val i	GCG A Ala	GAG E Glu	GGG G Gly

Figura 5. Tabla de codones de bacterias y arqueas 11.
Fuente: National Center for Biotechnology Information, n.d.

Función Comparación de secuencias: Esta función puede comparar dos o tres secuencias genómicas de diferentes variantes y calcular el porcentaje de similitud entre ellas. Las secuencias se comparan dos a dos mediante algoritmos de alineamiento por pares. Utiliza las librerías Bio.SeqIO y Bio.Pairwise2 del módulo de Biopython para leer y comparar las secuencias, además del módulo Matplotlib para obtener las diferentes gráficas de comparación. La función se ejecuta mediante un alineamiento global que puntua 1 por cada *match* y no penaliza los *gaps*. La puntuación o score del alineamiento se divide por la longitud total de la secuencia y se múltiplica por cien, obteniendo así el porcentaje de similitud entre ambas secuencias ($\frac{núm.\ total\ coincidencias}{longitud\ secuencia} \times 100$).

Función Análisis de mutaciones: Esta función usa la librería Bio.SeqIO para leer las secuencias, el módulo Numpy para introducir dichas secuencias en sus correspondientes matrices y el módulo Matplotlib para generar mapas de calor. La función lee las secuencias descargadas en el formato CDS nucleotide FASTA y las transforma en un diccionario para poder compararlas.

Un diccionario es un tipo de estructura similar a una matriz, capaz de almacenar variables de todo tipo (caracteres, secuencias, números y otras matrices) junto con un identificador por cada variable o entrada del diccionario. En este caso el diccionario

consta de variables de tipo matriz asociadas a una secuencia y etiquetadas con un identificador con el nombre del gen en cuestión. Por ejemplo, el diccionario de la secuencia NC_045512.2 contiene una variable con el identificador "*Gene S*". La variable con este identificador adquiere la estructura de una matriz de dimensiones 62x62, debido a que el gen S tiene una longitud aproximada de 3844 nucleótidos. Esta estructura se repite con el resto de variables del diccionario. Una vez introducida la secuencia completa, los nucleótidos son transcritos según un código numérico. Finalmente, para comparar las secuencias de dos variantes sus correspondientes diccionarios deben restarse. Si el producto de la resta de diccionarios es diferente de cero en una posición concreta del genoma, existe mutación (Figura 6).

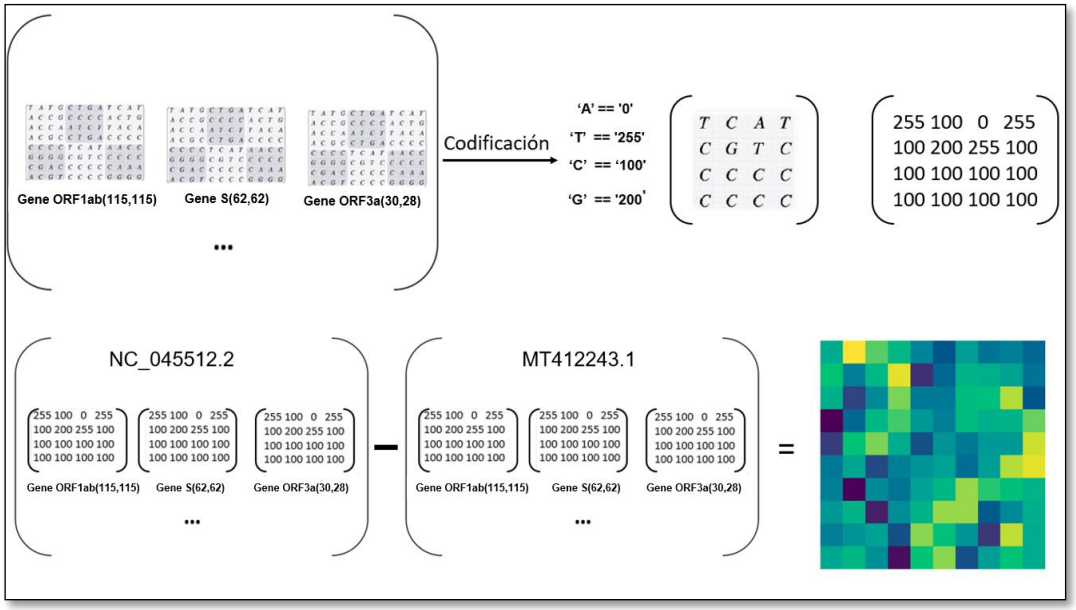


Figura 6. Uso de matrices en la función Análisis de mutaciones. Fuente: Adaptación.

Función Alineamiento proteico: Existen diferentes tipos de modalidades de alineamiento: globales (las secuencias se alinean a lo largo de toda su longitud, intentando alinear secuencias completas) y locales (sólo se alinean la partes más parecidas de la secuencia). Esta función utiliza las librerías Bio.SeqIO y Bio.Pairwise2 del módulo de Biopython para alinear las secuencias. Es posible especificar las puntuaciones por *match* y las penalizaciones por *gap*. En esta ocasión el programa realiza las puntuaciones por *match* mediante una matriz de puntuación BLOSUM62. En cuanto a la puntuación de las penalizaciones, el alineamiento global se penaliza con -10 puntos por la iniciación de un *gap* y -1 por la extensión. Por otro lado el

alineamiento local se penaliza -10 puntos por la iniciación de un *gap* y -0.5 por la extensión. Por lo general, se obtienen mejores alineamientos penalizando los *gaps*.

Función Gráficas de la proteína: A partir de un archivo PDB, y mediante funciones del módulo Prody, genera diferentes gráficas de la proteína: el diagrama de Ramachadran de la proteína, un mapa de contacto de la proteína y una gráfica del factor B o factor de temperatura. Utiliza las funciones ShowContactMap, getBetas, CalcPhi y CalcPsi.

Función MUSCLE: Esta función es capaz de ejecutar la herramienta MUSCLE a partir de su archivo ejecutable. Para ello se descargó la herramienta de su página oficial (MUSCLE, n.d.) y se insertó en la carpeta raíz del programa. Está función además usa las librerías Bio.Align, Bio.AlignIO y StringIO.

Función Árbol filogenético: Esta función es capaz de procesar los archivos obtenidos de la función MUSCLE en su formato de alineamiento PHYLIP para generar y tratar arboles filogenéticos, usando las librerías Bio.AlignIO y Bio.Phylo y sus funciones Phylo.read, Phylo.draw, DistanceCalculator. Para alinear un número elevado de secuencias optaremos por usar la herramienta web MUSCLE de la plataforma Clustal Omega.

Función Visualización 3D: Utiliza la función MMCIFParser de la librería Bio.pdb, entre otras, para imprimir un modelado en tres dimensiones de la proteína a través de su archivo mmCIF. Esta función solo puede ejecutarse en el entorno de desarrollo Jupyterlab, que dispone del módulo NGLView con herramientas gráficas para este tipo de modelado.

Función Menú: Se encarga de ejecutar todas las funciones mencionadas anteriormente a través de una interfaz gráfica creada a partir del módulo Tkinter.

5 Resultados

5.1 Menú Coronalyzer

Este menú se encarga de ejecutar todas las funciones del programa a través de una interfaz gráfica, con ventanas desplegables y botones de acción. En la parte superior izquierda encontramos el logo del programa, así como su nombre y versión, mientras que en la parte superior derecha encontramos los botones de despliegue de ventana. En la parte superior izquierda se localiza también la ventana desplegable de análisis, donde se encuentran los botones de acción de las funciones de análisis del programa: *Sequence, Compare 2 o 3 Seq, Mutations, Protein alignment local o global, Protein Plots, MUSCLE, Pyhlotree* y *3DView*. En la parte central de la interfaz nos encontramos con las casillas de entrada de los archivos. En las primeras tres casillas se pueden introducir los nombres de los archivos a procesar por las funciones de análisis, mientras en las últimas dos se introducen el nombre de los archivos a descargar de su correspondiente base de datos. Los botones de acción de la parte central son los que ejecutan las funciones de descarga (FASTA o PDB) (Figura 7).

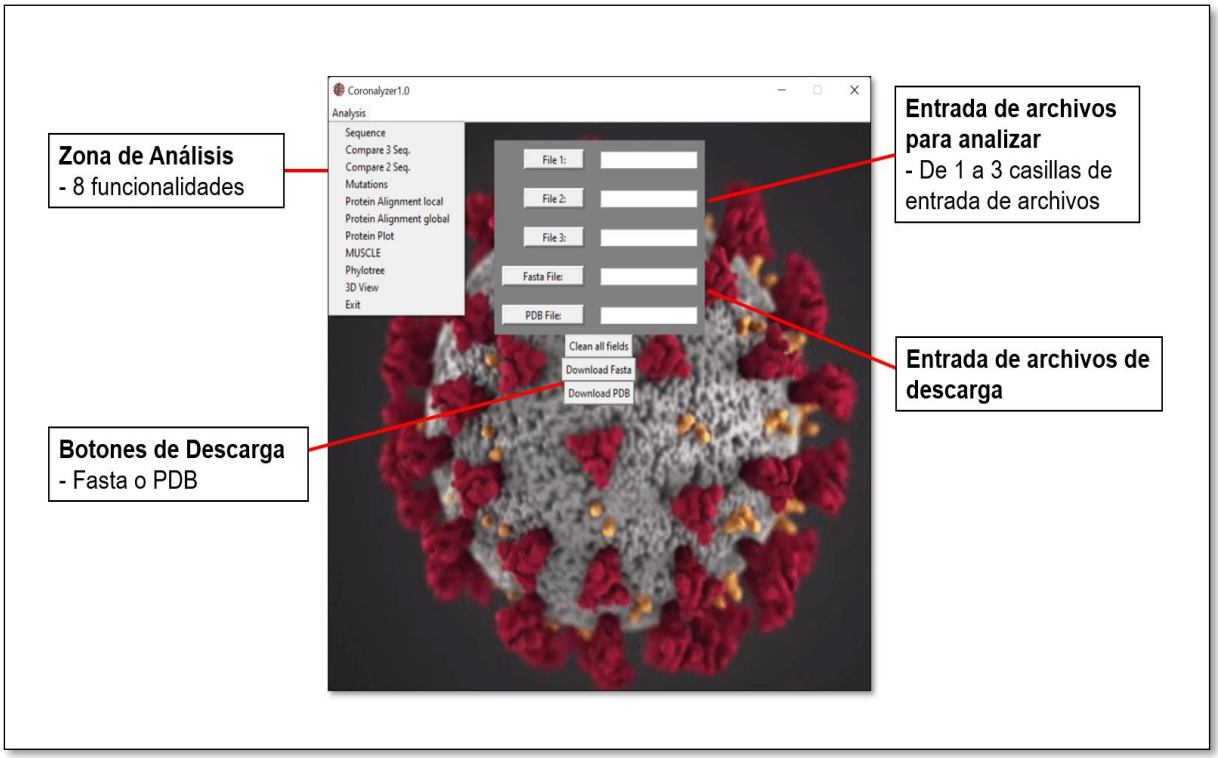


Figura 7. Menú interactivo del programa Coronalyzer. Fuente: Programa.

5.2 Descarga de archivos FASTA y PDB

Las funciones de descarga actúan descargando los archivos de sus correspondientes bases de datos a partir del código de registro. El código de registro en la base de datos GenBank del NCBI consta de dos letras iniciales junto a un código numérico de 7 dígitos, como sucede con la secuencia NC_045512.2, mientras que las proteínas depositadas en el PDB solo constan de un código de 4 dígitos, con una letra inicial y tres números, como sucede en el caso de la proteína 6VXX (Proteína S). Para realizar el proceso de descarga deberemos usar las casillas Archivo FASTA y Archivo PBD y pulsar los botones centrales de descarga de archivo. Una vez descargados los archivos se almacenan en la carpeta raíz del programa para ser procesados posteriormente por las diferentes herramientas de análisis (Figura 8).

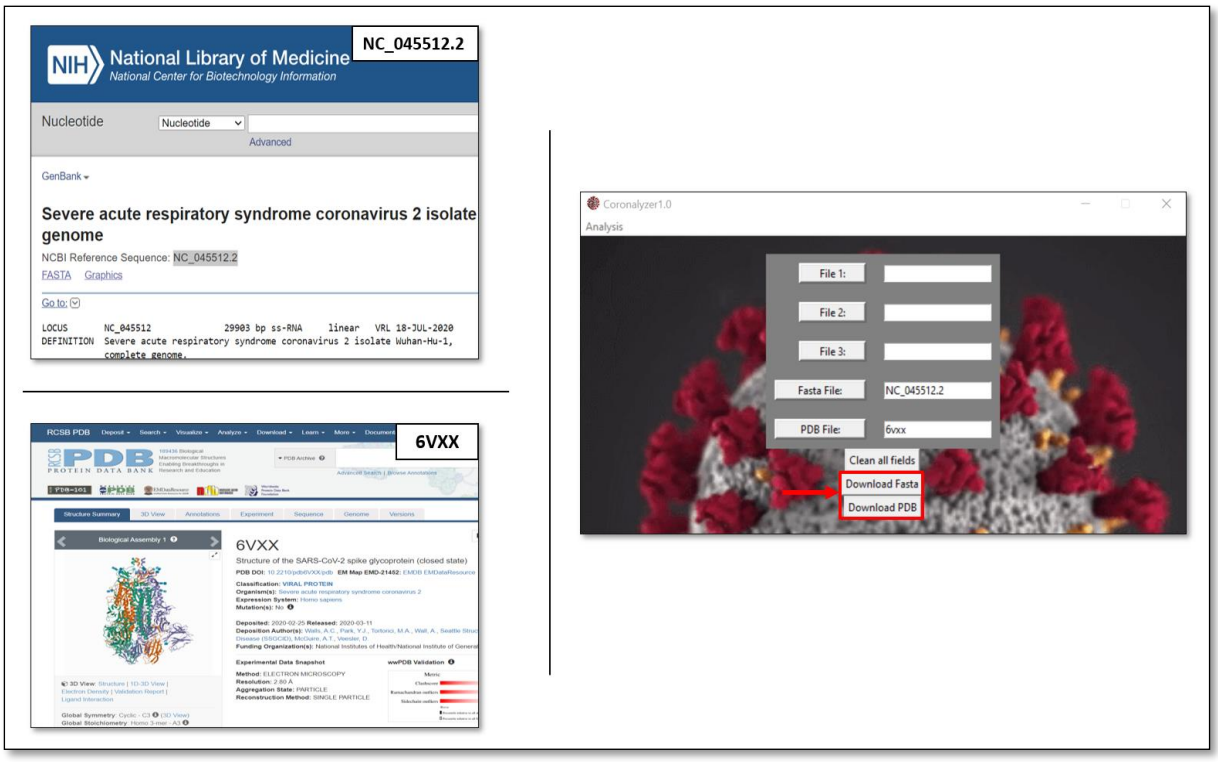


Figura 8. Descarga NC_045512.2 de NCBI y 6VXX de PDB Coronalyzer.

Fuente: National Center for Biotechnology Information, n.d., RCSB PDB: Homepage, n.d. y programa.

5.3 Análisis de secuencia

La estructura de una proteína depende de la organización en el espacio de sus residuos. La proteína adquiere una determinada forma en función de su composición fisicoquímica, que depende directamente de que tipos de aminoácidos la conforman y que posición ocupan en la secuencia. La función de análisis de secuencia utiliza el archivo FASTA almacenado en la carpeta raíz y extrae información de la secuencia que deseemos analizar. Para ello deberemos introducir el nombre del archivo que contiene la secuencia en la casilla Archivo 1 y pulsar en la ventana desplegable de *Analysis* la opción *Sequence* (Figura 9). Esta función es capaz de realizar gráficas sobre el número y tipo de nucleótidos que hay en la secuencia, el número y el tipo de aminoácidos presentes en la secuencia, clasificarlos en grupos, por el tipo de carga y por su posición en estructuras secundarias (Figuras 10 y 11). Todas las gráficas realizadas se almacenarán en la carpeta raíz del programa. Una vez procesada la secuencia, los datos del análisis se muestran a través de la terminal (Figuras 12 y 13).

En este caso para realizar la prueba se analizará la secuencia de la variante de referencia NC_045512.2 aislada de Wuhan el 2019.

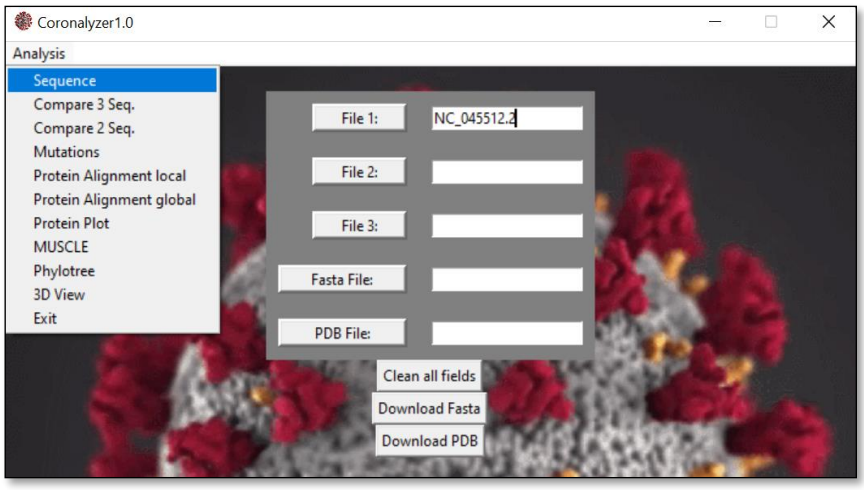


Figura 9**.** Menú Análisis de secuencia Coronalyzer. Fuente: Programa.

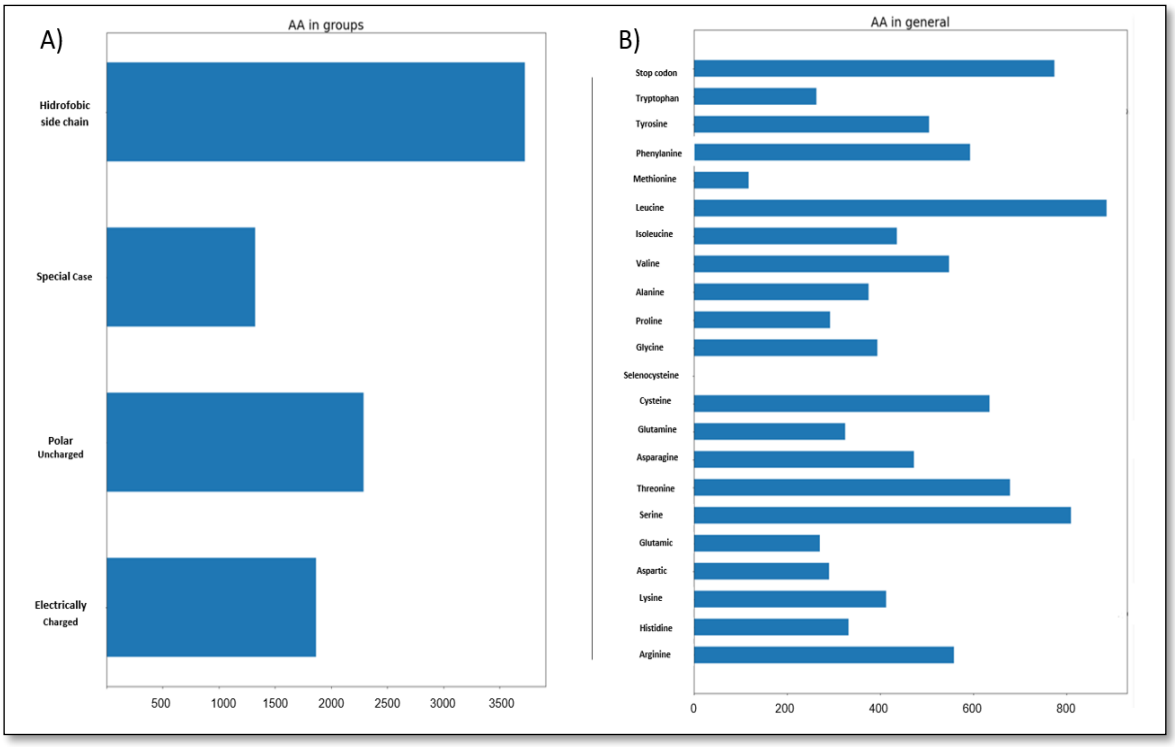


Figura 10. Gráficos de la función Análisis de secuencia NC_045512.2 Coronalyzer: A) Aminoácidos por grupo, B) Tipo y número de aminoácidos. Fuente: Programa.

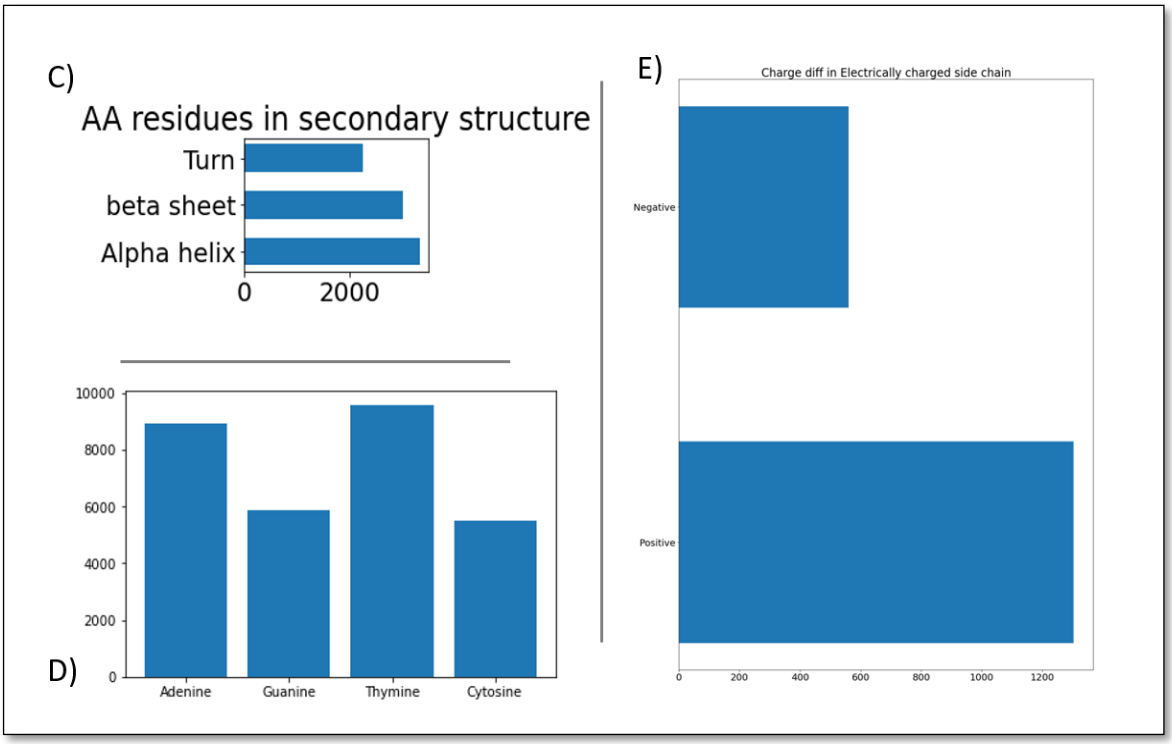


Figura 11. Gráficos de la función Análisis de secuencia NC_045512.2 Coronalyzer: C) Residuos en estructuras secundaria, D) Número de nucleótidos AGTC, E) Cargas en las cadenas laterales hidrofóbicas. Fuente: Programa.

Figura 12. Terminal de la función Análisis de secuencia NC_045512.2 Coronalyzer. Fuente: Programa.

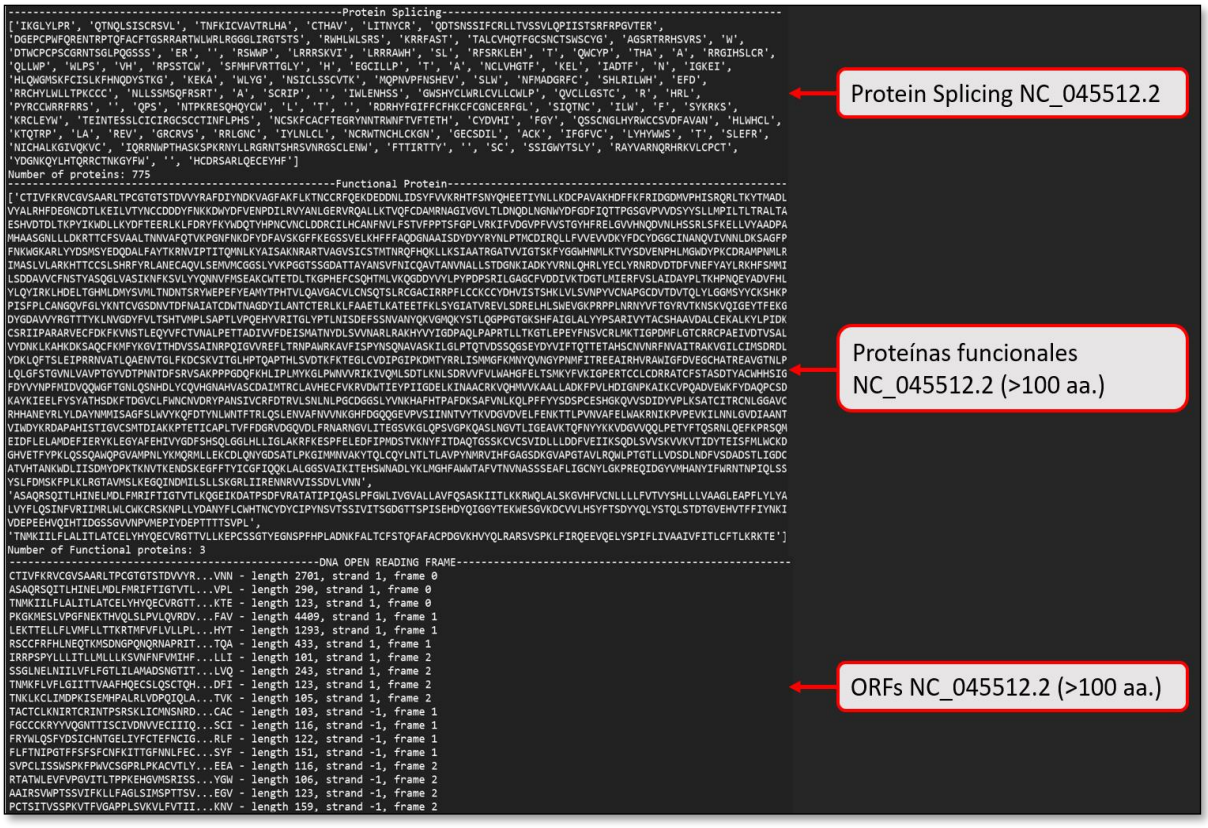


Figura 13. Terminal de la función Análisis de secuencia NC_045512.2 Coronalyzer. Fuente: Programa.

5.4 Comparación de secuencias

En esta función se emplean algoritmos de alineamiento de secuencia por pares. Este tipo de alineamiento es usado para identificar regiones de similitud que pueden indicar relaciones funcionales, estructurales y evolutivas entre dos secuencias. Las secuencias pueden ser comparadas con las opciones *Compare 2* o *Compare 3* de la ventana desplegable de *Analysis*. El nombre de los archivos a comparar ha de ser introducido en las casillas centrales de Archivo 1, 2 y 3. Tras la comparación se obtiene el porcentaje de similitud a través de la terminal del programa, así como una gráfica de similitud que se almacenara en la carpeta raíz. La opción *Compare 2* muestra de manera adicional el alineamiento de las dos secuencias a través de la terminal.

Las secuencias de MERS (NC_019843.3), SARS-CoV-1 (NC_004718.3) y SARS-CoV-2 (NC_045512.2) se compararon mediante la función *Compare* 3. Se puede observar un mayor porcentaje de similitud entre SARS-CoV-1 y SARS-CoV-2, con un 83,33% (Figura 14 A). También se compararon las secuencias del gen S de la variante de referencia NC_045512.2 y la variante Delta OM202516.1 mediante la función *Compare 2,* con un porcentaje del 99,24% (Figura 14 B).

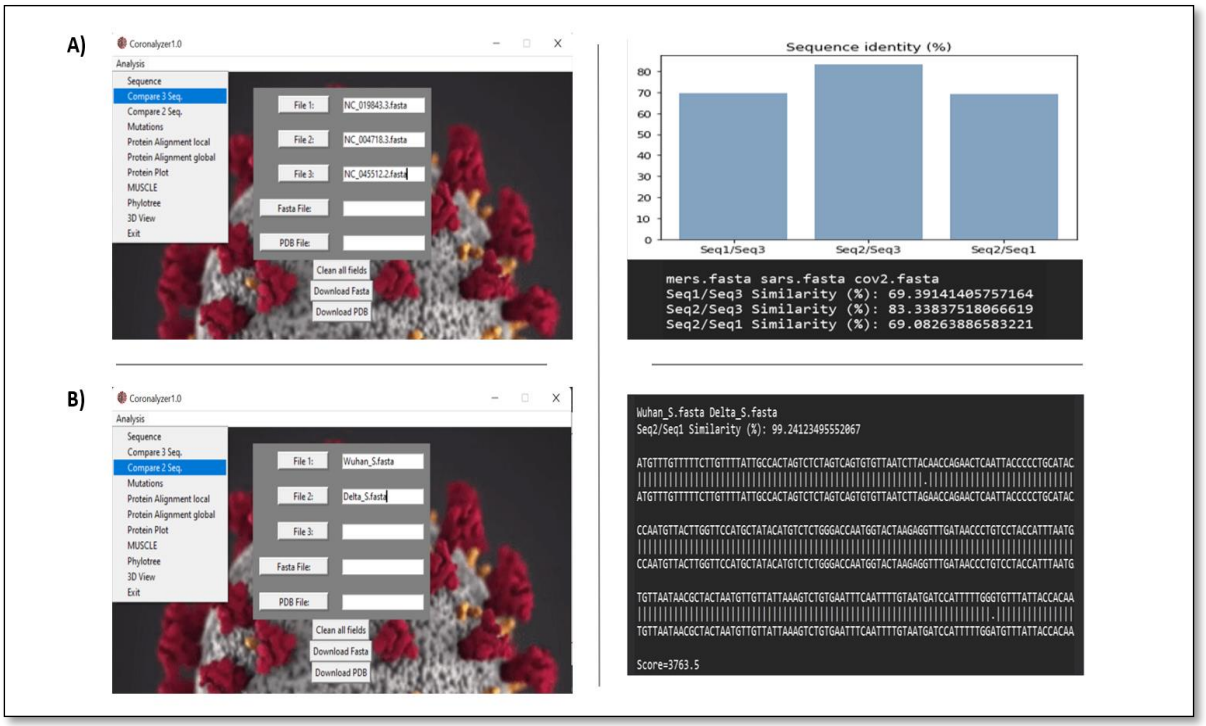


Figura 14**.** Comparación Coronalyzer: A) Compare 3, B) Compare 2. Fuente: Programa

5.5 Analizar mutaciones

El mapa de calor es una técnica de visualización de datos que mide la magnitud de un fenómeno en colores y en dos dimensiones. Esta función usa la estructura característica de los mapas de calor para representar gráficamente los diferentes genes del virus, aplicando diferentes tonos de coloración según el nucleótido de la secuencia. Antes de realizar el análisis es necesario comprobar que no existe desplazamiento entre ambas secuencias y que estas están correctamente alineadas. En caso de que exista desplazamiento, pueden usarse las herramientas del programa para alinear las secuencias. Para realizar el proceso de análisis de mutaciones se introducen los nombres de los archivos (CDS nucleotide FASTA) en las casillas Archivo 1 y Archivo 2, para a continuación pulsar el botón *Mutations* de la ventana desplegable *Analysis* (Figura 15). Todos los mapas de calor generados por el programa se almacenan en la carpeta raíz del programa y las mutaciones encontradas se muestran a través de la terminal.

En este caso para la prueba compararemos la variante de referencia aislada de Wuhan el 2019 (NC_045512.2) y la secuencia de la variante Delta aislada de Italia en el 2022 (OM202516.1) (Figura 16). Las mutaciones encontradas por el programa y su posición (en la matriz, gen y proteína) se muestran a través de la terminal.

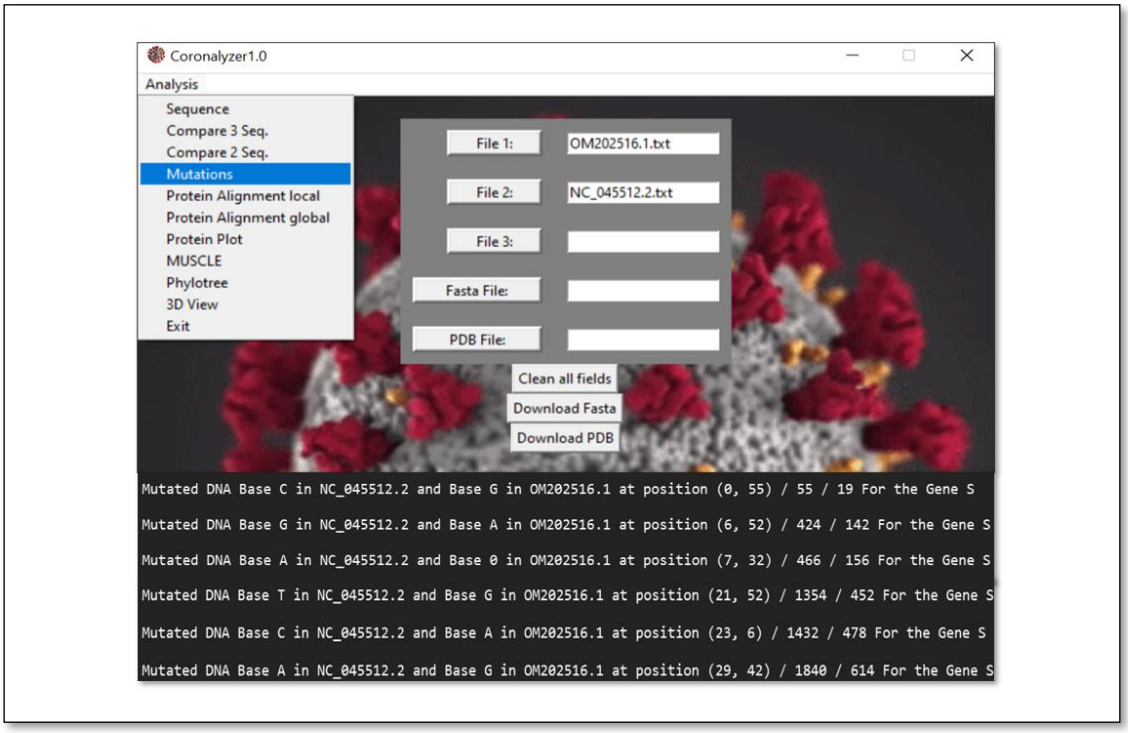


Figura 15. Terminal Análisis de mutaciones Coronalyzer. Fuente: Programa.

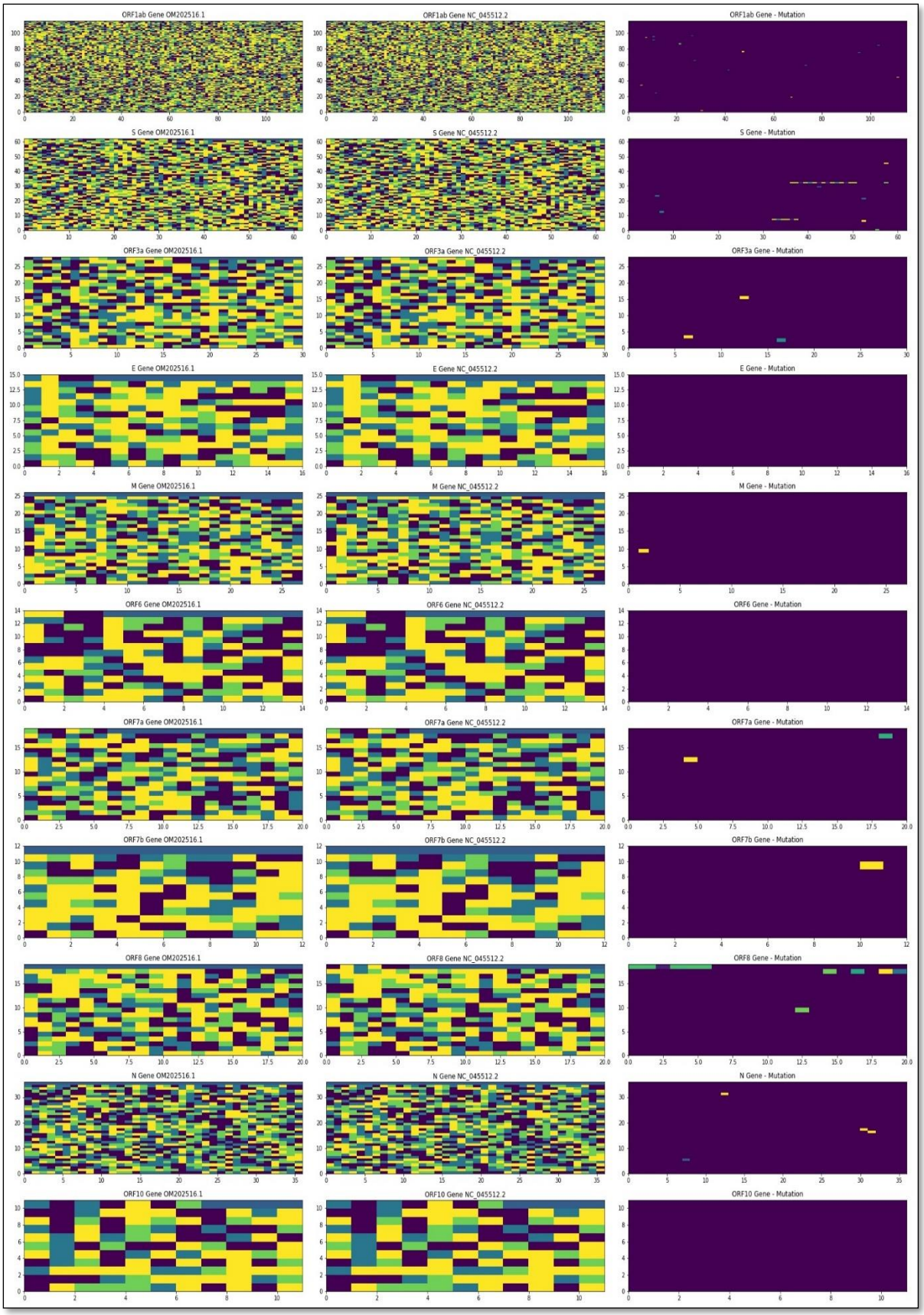


Figura 16. Mapa de calor secuencias Wuhan/ITA (NC_045512.2 y OM202516.1). Fuente: Programa.

El enfoque del mapa de calor nos ofrece la posibilidad de poder visualizar todo el genoma y las mutaciones de dos secuencias diferentes dentro de una sola imagen. En las dos primeras columnas de la figura 16 se observa una representación de los dos genomas de las variantes en forma de matriz coloreada. La coloración hace referencia al nucleótido en cuestión (A= Azul oscuro, T= Amarillo, G= Verde claro , C= Verde Oscuro) y la posición del pixel dentro de los ejes hace referencia a la posición del nucleótido dentro de la secuencia. Por otro lado, en la última columna de la figura se muestran las mutaciones entre ambas secuencias.

Entre las dos variantes se encontraron las siguientes mutaciones:

- **Gen ORF1ab:** (2, 30), (19, 67), (24, 11), (34, 5), (44, 111), (53, 41), (59, 73), (65, 27), (75, 95), (76, 47), (84, 103), (86, 21), (91, 10), (94, 7), (95, 10), (96, 26).
- **Gen S:** (0, 55), (6, 52), (7, 32), (7, 33), (7, 34), (7, 35), (7, 36), (7, 37), (12, 7), (21, 52), (23, 6), (29, 42), (32, 36), (32, 37), (32, 38), (32, 39), (32, 40), (32, 41), (32, 42), (32, 43), (32, 44), (32, 45), (32, 46), (32, 47), (32, 48), (32, 49), (32, 50), (32, 57), (45, 57) .
- **Gen ORF3a:** (2, 16) , (3, 6), (15, 12).
- **Gen M:** (9, 1).
- **Gen ORF7b:** (12, 4), (17, 18), (9, 10).
- **Gen ORF8:** (9, 12), (17, 14), (17, 15), (17, 16), (17, 18), (17,19), (18, 0), (18, 1), (18, 2), (18, 3), (18, 4), (18, 5).
- **Gen N:** (5, 7), (16, 31), (17, 30), (31, 12).

El programa encontró 29 mutaciones en el gen S, resultado que se comprobó con la herramienta Blastn. En la figura 17 podemos observar las coincidencias o *identities* entre ambas variantes, así como la primera mutación del gen S en la posición 55.

Figura 17. Blastn gen S Wuhan/ITA (NC_045512.2 y OM202516.1). Fuente: *Nucleotide BLAST: Search nucleotide databases using a nucleotide query*, n.d.

5.6 Alineamiento de proteínas

El objetivo general de la alineación de secuencias por pares es encontrar el mejor emparejamiento de dos secuencias, de modo que haya una correspondencia máxima entre los residuos. Para lograr este objetivo, una secuencia debe desplazarse en relación con la otra para encontrar la posición donde se encuentran las coincidencias máximas. Esta función usa el alineamiento de secuencia por pares, en esta ocasión para el tratamiento de secuencias proteicas. Para proceder con el alineamiento debemos introducir los nombres de los archivos FASTA de las proteínas que queremos comparar en las casillas Archivo 1 y Archivo 2, y pulsar los botones *Protein Alignment local* o *Protein Alignment global* en la ventana desplegable *Analysis*. Una vez se ejecuta la función podemos observar a través de la terminal el alineamiento de las secuencias con su correspondiente score o puntuación (Figura 18).

La alineación de proteínas de diferentes variantes puede ayudarnos a identificar similitudes y diferencias entre secuencias. En este caso alinearemos la proteína S de MERS (K9N5Q8) y la proteína S de SARS-CoV-2 (PODTC2) mediante un alineamiento local y global.

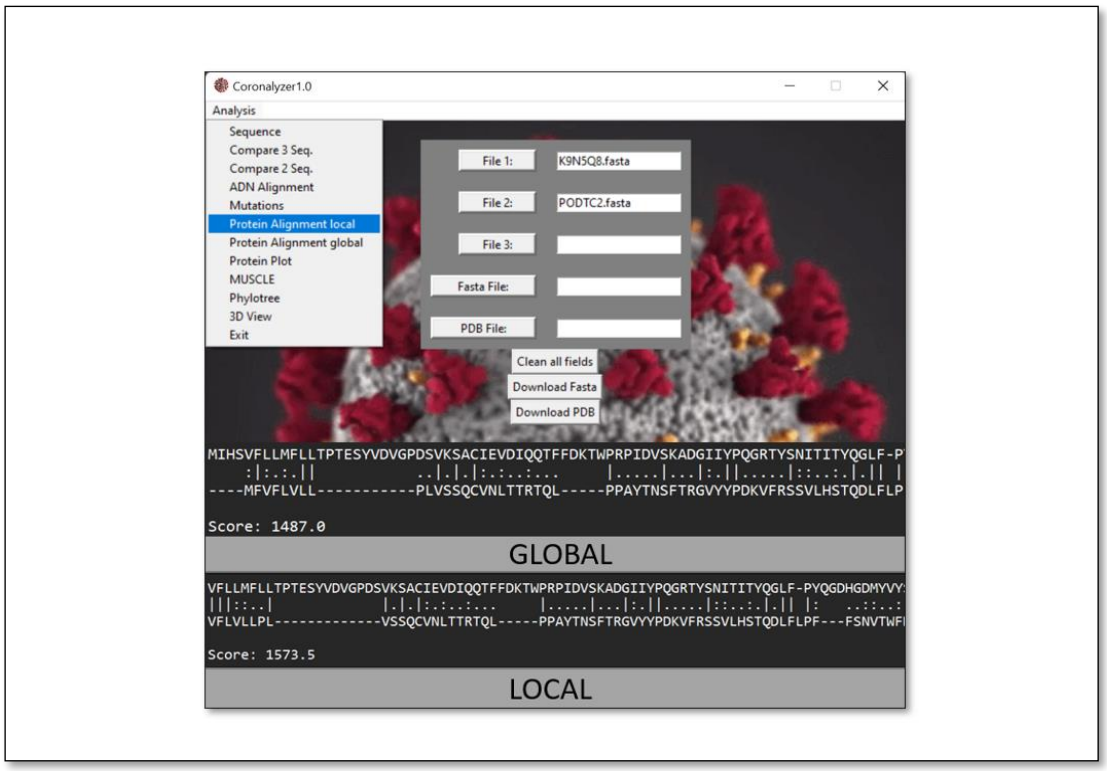


Figura 18**.** Protein Alignment K9N5Q8 y PODTC2 Coronalyzer. Fuente: Programa.

5.7 Gráficas de la proteína

Esta función permite realizar diferentes tipos de gráficas al introducir el código de una proteína descargada anteriormente del PDB en la casilla Archivo 1. Mediante la pulsación del botón *Protein Plots* (Figura 19) de la ventana desplegable *Analysis*, se genera el diagrama de Ramachadran, un mapa de contacto de la proteína y una gráfica del factor B.

En el diagrama de Ramachandran se pueden visualizar todas las combinaciones posibles de ángulos diedros psi y phi en los residuos de un polipéptido, que contribuyen a la conformación de la estructura de las proteínas. Este diagrama es una forma de visualizar las regiones energéticamente permitidas para los ángulos diedros del esqueleto de una proteína. Así, nos permite determinar cuántos residuos se encuentran formando parte de un elemento en una estructura secundaria determinada, y cuántos presentan unos ángulos de torsión forzados de acuerdo con la estructura de la proteína de la que disponemos (Ramakrishnan et al., n.d.). Este gráfico permite, por lo tanto, aproximar a priori cuál será la estructura secundaria del péptido, ya que existen combinaciones de ángulos típicas para cada estructura (hélice α y hoja β). Además es uno de los indicadores que se utilizan para controlar la calidad de las estructuras proteicas determinadas por cristalografía o RMN. En estructuras de alta calidad, más del 90% de los residuos (excluidas las glicinas) se encuentran dentro de las áreas más favorables.

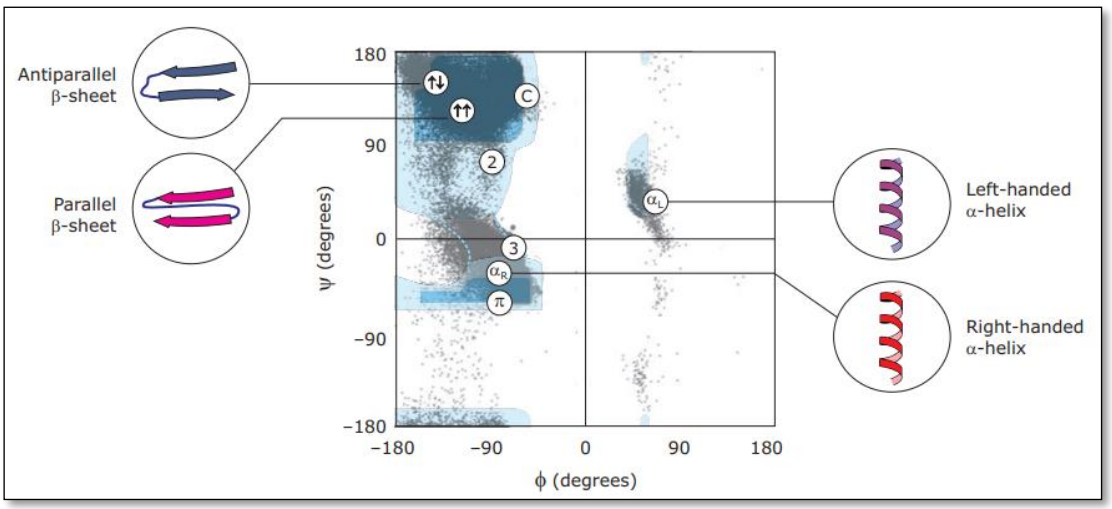


Figura 19**.** Diagrama de Ramachadran. Fuente: Ramakrishnan et al., n.d.

Un mapa de contacto representa la distancia entre todos los posibles pares de residuos de la estructura de una proteína tridimensional utilizando una matriz

bidimensional binaria. Para dos residuos, el elemento de la matriz es 1 si los dos residuos están más cerca de un umbral predeterminado y 0 en caso contrario. Los mapas de contacto también se utilizan para la superposición de proteínas y para describir la similitud entre las estructuras de las proteínas (Cueno et al., 2021).

El factor B o factor de temperatura describe la atenuación de la dispersión de rayos X o la dispersión de neutrones coherente causada por el movimiento térmico. Los factores B se refieren a una disminución de la intensidad en la difracción como resultado de dos fenómenos diferentes, el trastorno dinámico causado por la vibración dependiente de la temperatura de los átomos y el trastorno estático. Los factores B se pueden usar para identificar la flexibilidad en las proteínas, proponiendo que los factores B altos indican una flexibilidad más alta que el promedio en comparación con los factores B bajos, en posiciones más rígidas. Los datos de cada estructura de rayos X depositada en el PDB incluyen un factor B para todos los átomos excepto para el hidrógeno (Sun et al., 2019).

En esta ocasión para realizar la prueba se generaron las gráficas de las proteínas 6VXX y 7TEY, correspondientes a la proteína S de la variante de referencia y la variante Delta. Como podemos ver, en las gráficas del factor B encontramos un factor de temperatura más alto en 7TEY que en 6VXX. Los picos más altos en estas gráficas los encontramos en los 500 primeros residuos aproximadamente (Figura 21 A y B). Estos residuos corresponden a los dominio NTD y RBD de la proteína, zona con una alta flexibilidad (Figura 21 C). Cabe destacar un aumento del factor B en la proteína 7TEY respecto a 6VXX. En los mapas de contacto se aprecian diferencias en la estructura entre los residuos 300 y 650 de ambas proteínas. En cuanto al diagrama de Ramachadran, encontramos una mayor cantidad aminoácidos formando parte de láminas beta que de hélices alfa (Figura 21 A y B), como se puede ver en la representación molecular en la (Figura 21 C). También hay una mayor concentración de residuos en zonas favorables, lo que refleja calidad en el proceso de cristalización de ambas proteínas. Las diferencias encontradas entre los diferentes gráficos pueden deberse a las mutaciones encontradas en 7TEY o al diferente estado en el que fueron cristalizadas las proteínas.

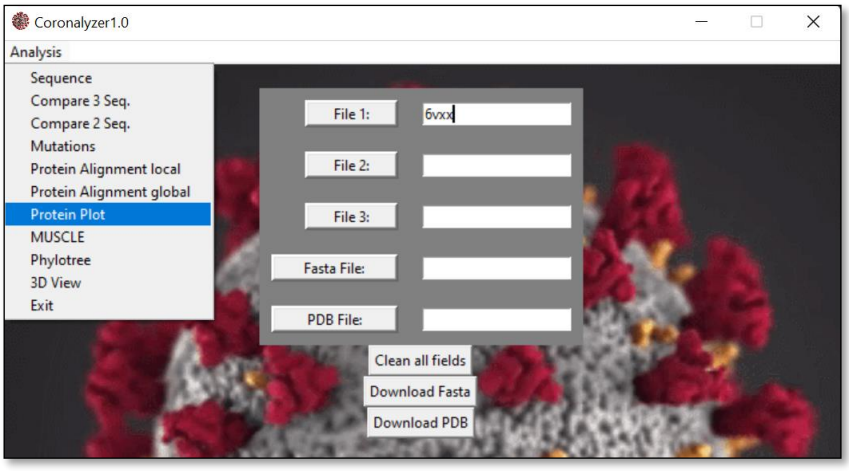


Figura 20**.** Menú Protein Plots Coronalyzer. Fuente: Programa

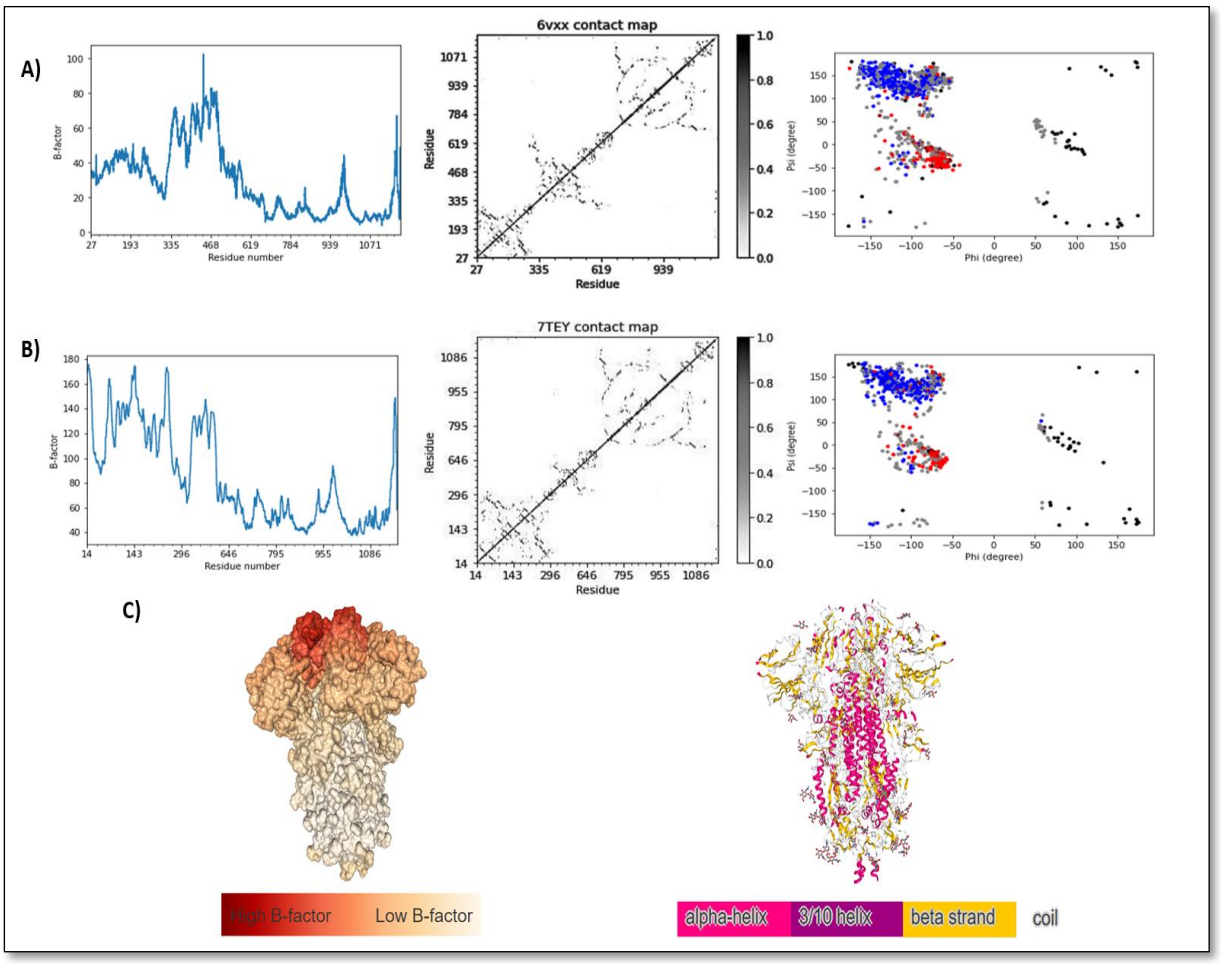


Figura 21. A) Diagrama de Ramachadran, mapa de contacto y factor B de 6VXX y 7TEY. En el diagrama de Ramachadran podemos visualizar GLY en negro, en rojo los residuos en hélices alfa, en granate residuos en hélices 3/10 y en azul los residuos en láminas beta. B) Visualización 3D de la proteína S con el esquema de color bfactor y sstructure. Fuente: Programa.

5.8 MUSCLE

Los métodos de alineamiento múltiple de secuencias (MSA) se refieren a una serie de soluciones algorítmicas para la alineación de secuencias relacionadas evolutivamente, teniendo en cuenta eventos evolutivos y cambios como las inserciones, deleciones y reordenamientos bajo ciertas condiciones. Estos métodos se pueden aplicar a secuencias de ADN, ARN o proteínas. El MSA tiene muchas aplicaciones, incluida la estimación de árboles filogenéticos, la predicción de estructuras y la identificación de residuos críticos.

El algoritmo MUSCLE es un *software* de dominio público implementado en el programa para el alineamiento múltiple de secuencias. Para el alineamiento múltiple de secuencias necesario un archivo FASTA que recoja todas las secuencias que deseamos alinear. Para ejecutar la función el nombre del archivo debe ser introducido en la casilla Archivo 1 y pulsar el botón *MUSCLE* de la ventana desplegable *Analysis*. Una vez realizados estos pasos se nos mostrará parte de la secuencia alineada a través de la terminal. La secuencia completa alineada se almacena en un archivo de formato PHYLIP en la carpeta raíz del programa (variantes.aln).

En este caso para realizar la prueba se introdujeron las secuencias genómicas de MERS (NC_019843.3), Sars-CoV-1 (NC_004718.3) y Sars-CoV-2 (NC_045512.2) en el archivo variantes.fasta.

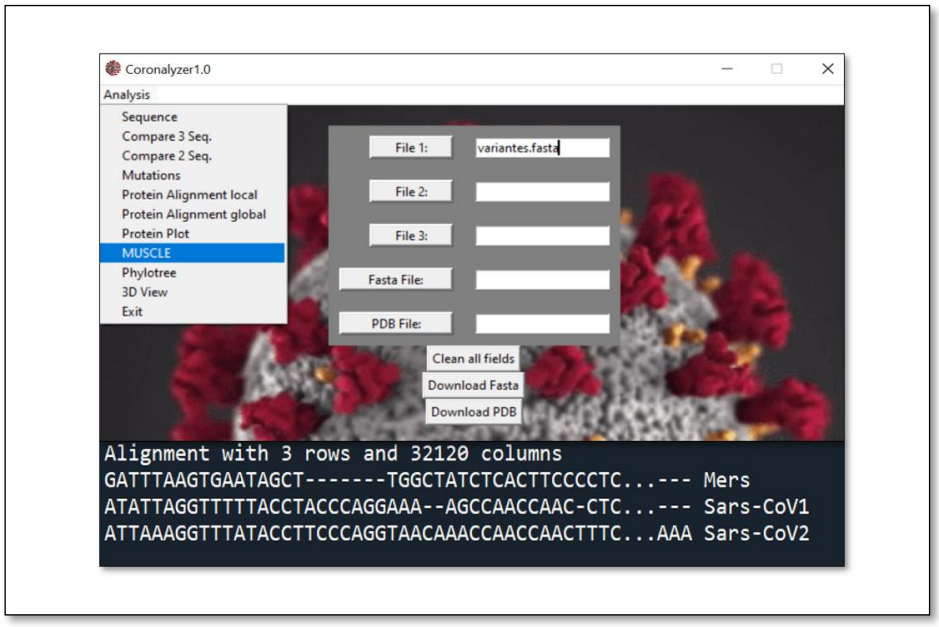


Figura 22**.** Multiple Sequence Alignment MERS, SARS-CoV-1, SARS-CoV-2 Coronalyzer.

Fuente: Programa.

5.9 Árbol filogenético

La filogenética es una disciplina de la biología evolutiva que se ocupa de comprender las relaciones históricas entre diferentes grupos de organismos a partir de la distribución en un árbol o cladograma dicotómico de los caracteres derivados de un antecesor común. Usando herramientas como MUSCLE, que utilizan el alineamiento múltiple de secuencias, se puede inferir la homología y estudiar la relación evolutiva entre las secuencias. El alineamiento múltiple de secuencias a menudo se utiliza para evaluar la conservación de los dominios proteicos, las estructuras terciarias y secundarias, e incluso aminoácidos o nucleótidos individuales.

Coronalyzer, a través de MUSCLE, es capaz de procesar un archivo FASTA que contiene diferentes secuencias y proporcionar uno nuevo con todas las secuencias alineadas en formato PHYLIP. Este formato permite que el programa pueda generar y tratar arboles filogenéticos, representándolos en diferentes formatos (Figura 23). Para ello introduciremos el archivo variantes.aln en la casilla archivo 1 pulsaremos el botón *Pyhlotree* de la ventana desplegable *Analysis.*

Para poner a prueba esta función se unieron las secuencias de diferentes coronavirus: 229E (NC_002645.1), HKU1 (NC_006577.2), MERS (NC_019843.3), SARS-CoV-1 (NC_004718.3), SARS-CoV-2 (NC_045512.2), Delta (OM202516.1), Gamma (MZ427312.1) y Beta (MZ433432.1). En el árbol se evidencia el relación filogenética de SARS-CoV-2 con otros coronavirus humanos previamente estudiados como 229E, HKU1 y MERS.

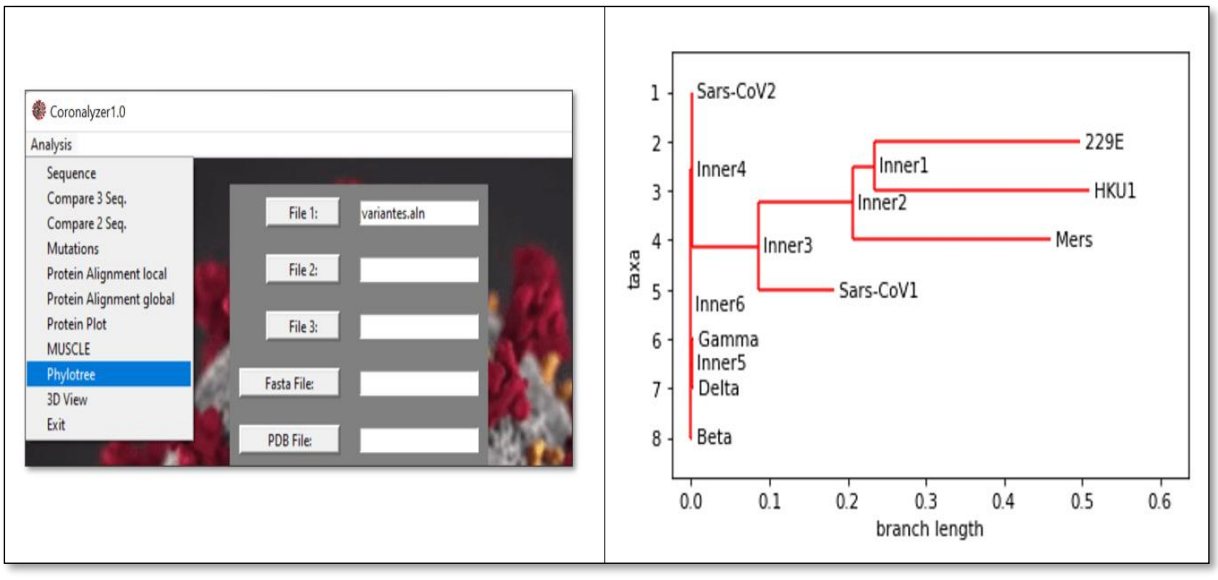


Figura 23**.** Árbol filogenético Coronalyzer. Fuente: Programa.

5.10 Visualización 3D

El objetivo de esta función facilitar la comprensión de la estructura de las proteínas y su implicación en distintos procesos biológicos y biotecnológicos. Es posible manipular modelos moleculares para visualizarlos tridimensionalmente. Los modos de visualización son muy parecidos a los que podemos encontrar en otros programas de visualización molecular, permitiéndonos observar diferentes características de la proteína: enlaces, dominios, cadenas, tipos de átomos (C, N, H, O), la distancia entre átomos, ángulos, la posición de un aminoácido o residuo dentro de la cadena, entre otras características de la molécula. Para visualizar la estructura 3D de la proteína es necesario introducir el nombre del archivo PDB previamente descargado en la casilla Archivo 1 y pulsar el botón *3D View* de la ventana desplegable *Analysis*.

En esta ocasión se realizó la prueba a partir del archivo 6VXX. El formato *Cartoon* se muestra con un esquema de color *residueindex* y el formato *Surface* con un esquema *Chainindex*. La primera configuración asigna un color único a cada residuo, lo que permite una fácil identificación del tipo de residuo y su distribución en la proteína, mientras que la segunda asigna un color único a cada cadena. Gracias a ello podemos observar la estructura de trimérica de la proteína y su distribución de cargas. Por otro lado, el formato *Ball+Stick* nos facilita la visualización de las moléculas y sus interacciones y enlaces (Figura 24).

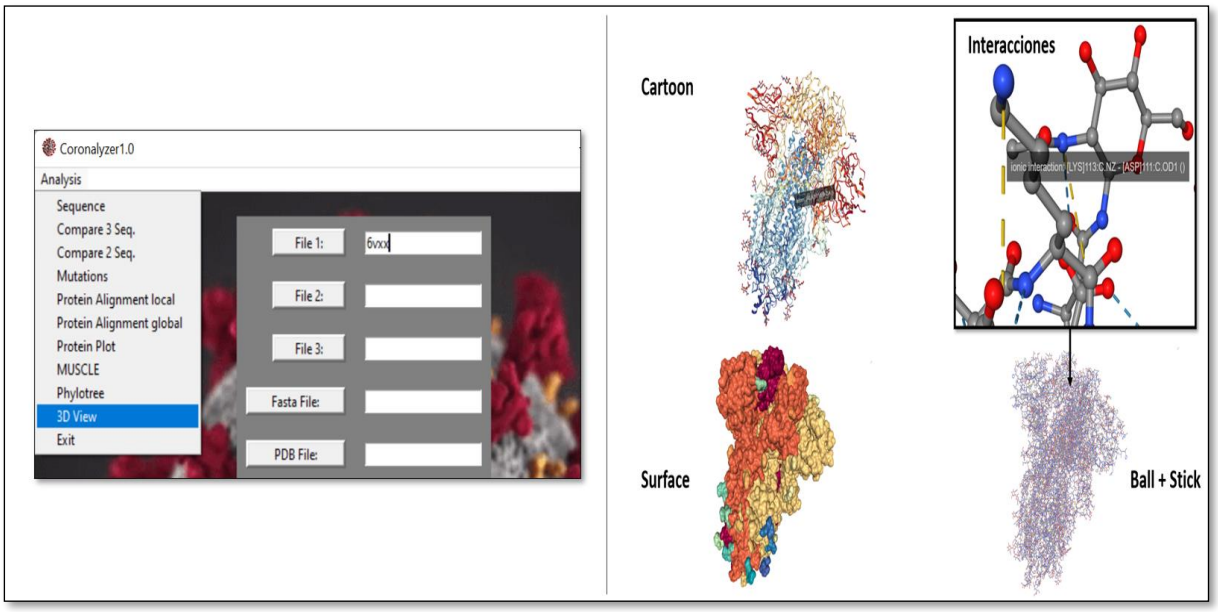


Figura 24**.** Visualización 3D de la proteína 6VXX (Proteína S). Fuente: Programa.

5.11 Aplicaciones

Para poner a prueba de manera conjunta las aplicaciones del programa se analizaron los genomas de las variantes NC_045512.2 y OM202516.1. Las mutaciones encontradas en el gen S se verificaron mediante la herramienta de alineamiento de proteínas, alineando las secuencias de la proteína S de PODTC2 (Wuhan) y 7TEY (Delta) (Figura 25 C). En el análisis genómico y proteico se encontraron 29 mutaciones, que se contrastaron con la información bibliográfica correspondiente (Tian et al.).

Las mutaciones más relevantes encontradas por el programa fueron (Figura 25):

- **T19R** (C-G 55), **G142D** (G-A 424), **R158G** (C-0), **L452R** (T-G 1354), **T478K** (C-A 1432), **D614G** (A-G 1840), **P681R** (C-G 2041), **D950N** (G-A 2847).

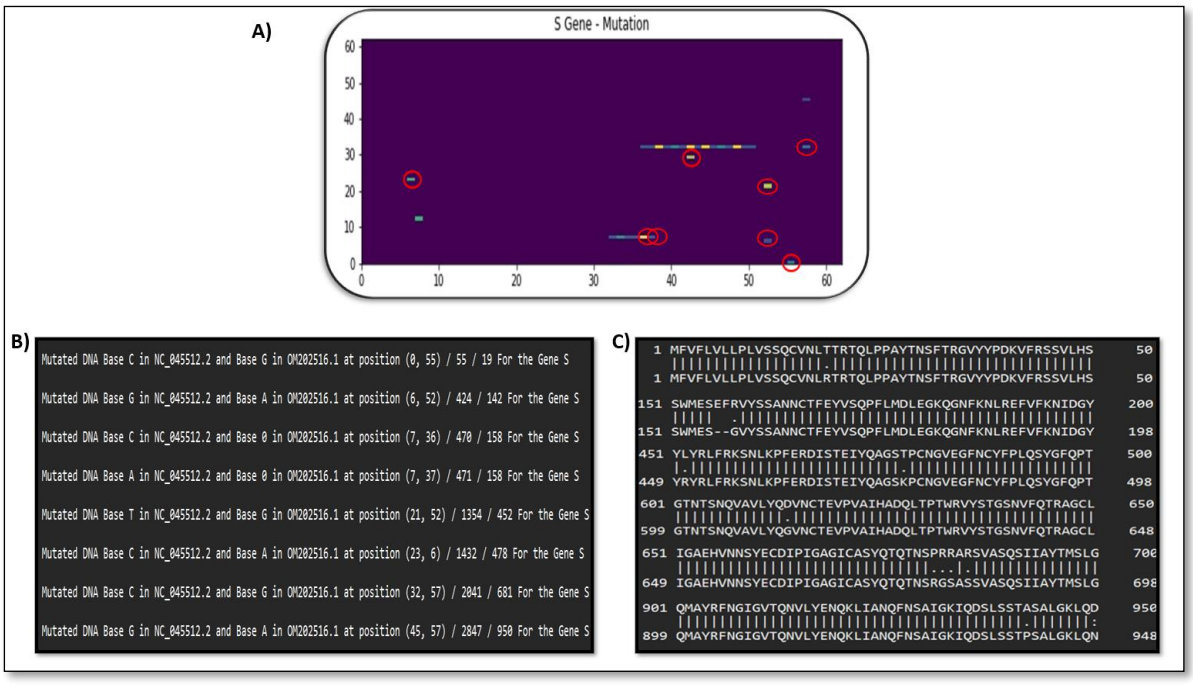


Figura 25. A) Análisis de mutaciones Heatmap. B) Terminal Análisis de mutaciones. C) Terminal Alineamiento global por pares de PODTC2 y 7TEY. Fuente: Programa.

Las mutaciones T19R, G142D y R158G afectan al dominio NTD, lo que sugiere la posibilidad de que la unión de algunos anticuerpos neutralizantes anti-NTD puede haberse reducido. La mutación L452R se localiza en la región del motivo de unión al receptor (RBM) en la región RBD, conteniendo residuos que se unen a ACE2. Esta mutación puede causar cambios estructurales en esta región que estabilizan la interacción entre la proteína espiga y el receptor ACE2 de la célula huésped, lo que lleva a un aumento de la infectividad. La mutación T478K, sustituyendo un aminoácido

no cargado (treonina) por uno positivo (lisina), ha aumentado notablemente el potencial electrostático de superficie del RBD de la proteína. Por otro lado, la mutación D614G reduce la compactación del trímero de proteína S y favorece el estado abierto del dominio RBD, aumentando considerablemente le infectividad del virus. Curiosamente, la mutación P681R es única y está ubicada en el sitio de escisión mediado por furina, región clave para la entrada en la célula huésped. La mutación P681R afecta la dinámica de replicación viral, acelera la fusión y promueve la infección célula-célula. Estas mutaciones afectan el comportamiento biológico de la variante Delta, incluido el aumento de la transmisibilidad y la evasión inmune (Figura 26).

Entre los diferentes efectos de las mutaciones que se vieron reflejados en la investigación epidemiológica, se mostró que el período de incubación después de la infección con la variante Delta fue de 2-3 días, período más corto que el de la cepa de tipo salvaje, de 3-7 días. El número reproductivo básico de la variante Delta (R0: 4.04 ~ 5.0) fue más alto que el de la cepa de tipo salvaje (R0: 2.2 ~ 3.77), y el tiempo de generación (el intervalo entre la infección del caso primario y los casos secundarios) fue de 2,9 días, que fue mucho más corto que la cepa de tipo salvaje, de 5,7 días. Además se ha informado que la actividad de neutralización de mAbs contra la variante Delta se redujo más de 5 veces en comparación con la de la cepa de tipo salvaje (Tian et al.).

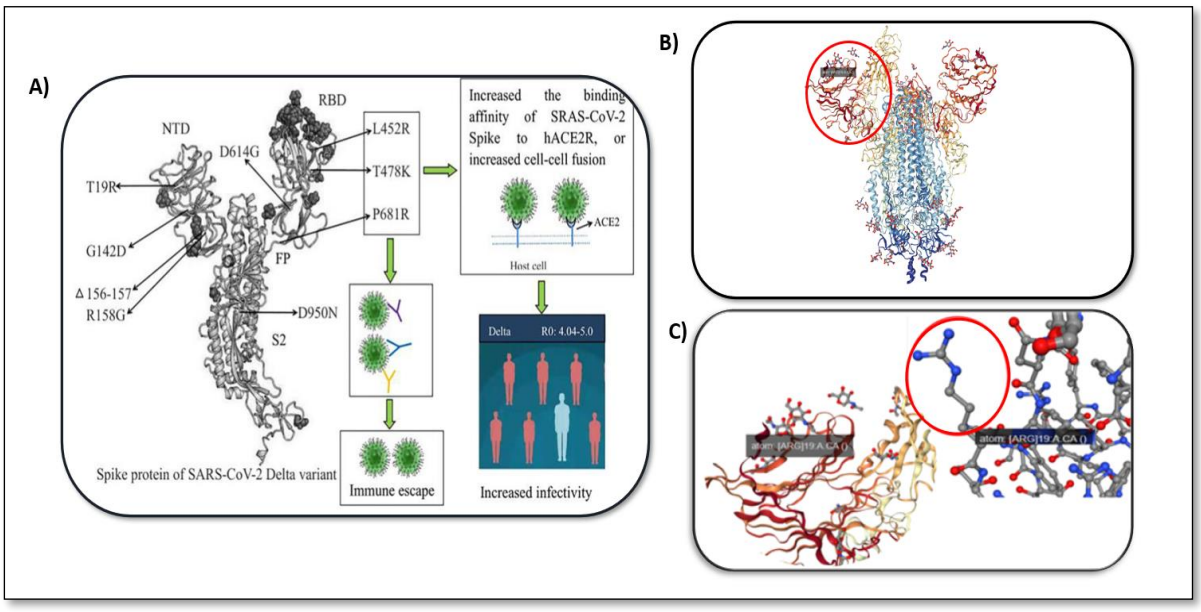


Figura 26. A) Mutaciones clave variante Delta. D) Región NTD proteína 7TEY 3D View. E) Arginina 19 de la proteína 7TEY 3DView. Fuente: Adaptación programa y Tian et al.

6 Discusión

En la actualidad, la bioinformática basada en la computación de alto rendimiento es el estándar de las herramientas de investigación científica y sin duda se utilizará para acelerar el desarrollo de medicamentos específicos. En el caso del SARS-CoV-2, la bioinformática ofrece herramientas para comprender la estructura del virus e identificar posibles *targets*, todo para poder desarrollar moléculas que puedan unirse e inactivar el virus.

Coronalyzer actúa como un programa de interfaz gráfica simple e implementación sencilla que resulta útil a la hora de comprender la estructura y funcionamiento del virus. El menú, así como sus diferentes opciones de análisis, hacen que el estudio de variantes de SARS-CoV-2 sea rápido y sencillo gracias a las bases de datos en línea.. Además, al tratarse de un programa de código abierto, Coronalyzer puede adaptarse a las necesidades del usuario, con la posibilidad de modificar el código así como de añadir nuevas funcionalidades.

En cuanto a las aplicaciones, Coronalyzer puede usarse para:

- **Análisis de secuencia**: Ofrecen información sobre el genoma del virus y sus variantes que puede ser usada para desarrollar diferentes líneas de investigación. Si se realiza una búsqueda bibliográfica de las variantes y sus características e impacto en la pandemia, podemos relacionar su comportamiento y fenotipo con las características de secuencia encontradas con el programa.
- **Análisis proteico**: es necesario para entender cómo funcionan las proteínas *target* que hacen infeccioso al virus, cómo evolucionan con el paso del tiempo y como les afecta a esta evolución factores como la presión por selección (medicamentos, vacunas, mAb). El alineamiento proteico y el modelado 3D sirven para encontrar mutaciones como las sustituciones, deleciones o inserciones y ver sus implicaciones sobre la función biológica (sobre la interacción con los receptores celulares, la fusión con las membranas celulares o la antigenicidad).
- **Análisis filogenético:** Es posible obtener pistas y formular hipótesis sobre las relaciones entre las diferentes especies de coronavirus utilizando técnicas como el alineamiento múltiple de secuencias (MSA). Este tipo de análisis

puede usarse para realizar estudios taxonómicos, registrando las características de secuencia encontradas por el programa junto a las variantes correspondientes. Esto permite identificar y registrar variantes de aminoácidos en la proteína S que están aumentando con frecuencia en muchas regiones del mundo, permitiéndonos realizar estudios geográficos y de distribución de las variantes (Korber et al., 2020).

El uso de este tipo de herramientas y las nuevas tecnologías en la producción de fármacos han demostrado ser claves a la hora de combatir el virus. Un ejemplo es la vacunología inversa, que se basa en métodos bioinformáticos para identificar antígenos objetivo de acuerdo con la información genómica. Otro ejemplo es el enfoque inmunoinformático, por el cual después de obtener el genoma del SARS-CoV-2 y la estructura de la proteína 3D, se predijeron los sitios de unión a poliproteínas e inhibidores, que pueden generar una respuesta inmune específica (Smith, C. C., Olsen, K. S 2021).

7 Conclusión

El uso de herramientas informáticas en la lucha contra el SARS-CoV-2, en tareas como la secuenciación y el seguimiento de la evolución del virus, suponen un gran avance a la hora de combatir escenarios como los ocurridos en 2019. En comparación con la mayoría de los otros virus de ADN con genomas relativamente más estables, el nuevo coronavirus es más propenso a mutaciones genéticas, por este motivo es probable que cambie la estructura fisicoquímica del virus y cause reducción en la efectividad de las vacunas.

Los recursos de secuenciación del SARS-CoV-2 están evolucionando muy rápidamente, la simple alineación de secuencias del virus consume pocos recursos y se puede hacer utilizando bases de datos en línea, como hemos visto en el desarrollo del trabajo. Además, el uso de módulos como Biopython, pensados para cuantificar y hacer cálculos con datos biológicos, hacen accesibles los algoritmos más usados en el sector bioinformático. La disponibilidad de este módulo y otros han hecho posible la creación programas propios como Coronalyzer. El programa reúne las características de diferentes módulos para facilitar y automatizar tareas análisis, acelerando la comprensión del virus. En concreto, Coronalyzer nos ofrece la posibilidad de obtener información sobre la composición aminoacídica y nucleotídica

de las diferentes variantes de SARS-CoV-2, así como los diferentes betacoronavirus con un linaje próximo como el SARS-CoV-1 y MERS, aportando información directa o gráfica de los mismos.

En conclusión, la unión de la epidemiología, bioestadística e informática es posible gracias a herramientas y bases de datos como las comentadas a lo largo del trabajo. Iniciativas como GISAID o bases de datos como el NCBI o el PDB, junto a lenguajes de programación y módulos usados, hacen accesibles los estudios en biología molecular, filogenética, modelado de proteínas y secuenciación. Gracias a este tipo de herramientas se han conseguido acelerar muchos de los estudios realizados sobre el SARS-CoV-2, motivo por el cuál la bioinformática tuvo un papel fundamental en la lucha contra la pandemia iniciada en 2019.

8 Bibliografía y Webgrafía

1. *Home - Johns Hopkins Coronavirus Resource Center*. (n.d.). Retrieved July 1, 2022, from https://coronavirus.jhu.edu/

2. Lu, R., Zhao, X., Li, J., Niu, P., Yang, B., Wu, H., Wang, W., Song, H., Huang, B., Zhu, N., Bi, Y., Ma, X., Zhan, F., Wang, L., Hu, T., Zhou, H., Hu, Z., Zhou, W., Zhao, L., … Tan, W. (2020). Genomic characterisation and epidemiology of 2019 novel coronavirus: implications for virus origins and receptor binding. *The Lancet*.

3. *GISAID - Contact*. (n.d.). Retrieved June 23, 2022, from https://gisaid.org/help/contact/

4. Ma, L., Li, H., Lan, J., Hao, X., Liu, H., Wang, X., & Huang, Y. (2021). Comprehensive analyses of bioinformatics applications in the fight against COVID-19 pandemic. In *Computational Biology and Chemistry* (Vol. 95). Elsevier Ltd.

5. Yang, H., & Rao, Z. (2021). Structural biology of SARS-CoV-2 and implications for therapeutic development. In *Nature Reviews Microbiology* (Vol. 19, Issue 11, pp. 685–700). Nature Research.

6. Thomson, E. C., Rosen, L. E., Shepherd, J. G., Spreafico, R., da Silva Filipe, A., Wojcechowskyj, J. A., Davis, C., Piccoli, L., Pascall, D. J., Dillen, J., Lytras, S., Czudnochowski, N., Shah, R., Meury, M., Jesudason, N., de Marco, A., Li, K., Bassi, J., O'Toole, A., … Snell, G. (2021). Circulating SARS-CoV-2 spike N439K variants maintain fitness while evading antibody-mediated immunity. *Cell*, *184*(5), 1171-1187.e20.

7. Harvey, W. T., Carabelli, A. M., Jackson, B., Gupta, R. K., Thomson, E. C., Harrison, E. M., Ludden, C., Reeve, R., Rambaut, A., Peacock, S. J., & Robertson, D. L. (2021). SARS-CoV-2 variants, spike mutations and immune

escape. In *Nature Reviews Microbiology* (Vol. 19, Issue 7, pp. 409–424). Nature Research.

8. Shajii, A., Numanagić, I., Leighton, A. T., Greenyer, H., Amarasinghe, S., & Berger, B. (2021). A Python-based programming language for high-performance computational genomics. In *Nature Biotechnology* (Vol. 39, Issue 9, pp. 1062–1064). Nature Research.

9. Adam, C. (2021). Biopython: Comparing the DNA Polymerase I (polA) Gene of Thermophilic, Mesophilic, and Psychrophilic Bacteria. *BiosciED: Journal of Biological Science and Education*, *2*(1), 10–20.

10. Zhang, X., Beinke, B., Kindhi, B. al, & Wiering, M. (2020). Comparing Machine Learning Algorithms with or without Feature Extraction for DNA Classification.

11. *National Center for Biotechnology Information*. (n.d.). Retrieved June 21, 2022, from https://www.ncbi.nlm.nih.gov/

12. *MUSCLE*. (n.d.). Retrieved July 15, 2022, from https://www.drive5.com/muscle/

13. *RCSB PDB: Homepage*. (n.d.). Retrieved July 5, 2022, from https://www.rcsb.org/

14. *Nucleotide BLAST: Search nucleotide databases using a nucleotide query*. (n.d.). Retrieved June 29, 2022, from https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastn&BLAST_SPEC=GeoBlast&PAGE_TYPE=BlastSearch

15. Ramakrishnan, C., & Sasisekharan Ramachandran, V. (n.d.). *H N 2*.

16. Cueno M, & Imai (2021). Structural comparison of the SARS COV 2 spike protein relative to other human-infecting coronaviruses. *Frontiers in medicine*. U.S. National Library of Medicine.

17. Sun, Z., Liu, Q., Qu, G., Feng, Y., & Reetz, M. T. (2019). Utility of B-Factors in Protein Science: Interpreting Rigidity, Flexibility, and Internal Motion and Engineering Thermostability. *Chemical Reviews*.

18. Tian, D., Sun, Y., Zhou, J., & Ye, Q. (2021). The Global Epidemic of the SARS-CoV-2 Delta Variant, Key Spike Mutations and Immune Escape. In *Frontiers in Immunology* (Vol. 12). Frontiers Media S.A.

19. Korber, B., Fischer, W. M., Gnanakaran, S., Yoon, H., Theiler, J., Abfalterer, W., Hengartner, N., Giorgi, E. E., Bhattacharya, T., Foley, B., Hastie, K. M., Parker, M. D., Partridge, D. G., Evans, C. M., Freeman, T. M., de Silva, T. I., Angyal, A., Brown, R. L., Carrilero, L., … Montefiori, D. C. (2020). Tracking Changes in SARS-CoV-2 Spike: Evidence that D614G Increases Infectivity of the COVID-19 Virus. *Cell*, *182*(4), 812-827.e19.

20. Smith, C. C., Olsen, K. S., Gentry, K. M., Sambade, M., Beck, W., Garness, J., Entwistle, S., Willis, C., Vensko, S., Woods, A., Fini, M., Carpenter, B., Routh, E., Kodysh, J., O'Donnell, T., Haber, C., Heiss, K., Stadler, V., Garrison, E., … Rubinsteyn, A. (2021). Landscape and selection of vaccine epitopes in SARS-CoV-2. *Genome Medicine*, *13*(1).

9 Autoevaluación

Durante realización del TFG he podido exponer con argumentos el valor de un campo de tanta relevancia hoy día como la Bioinformática. He tenido la oportunidad de profundizar en campos como la Virología, la Genómica y la Proteómica y ver sus implicaciones lucha contra la pandemia iniciada el 2019. Por otra parte, el trabajo refleja la importancia de la accesibilidad a bases de datos biológicas como el NCBI o el PDB, entre otras, así como del software orientado al estudio de la estructura y función de las biomoléculas.

A nivel teórico he tratado temas sobre la estructura y dinámica del virus, sobre sus propiedades moleculares y fisicoquímicas así como de sus diferentes variantes y mutaciones. A nivel práctico he podido desarrollar un programa con los conocimientos adquiridos en la carrera, un programa sencillo pero interesante que nos ofrece información relevante sobre las características de secuencia del SARS-CoV-2.

En general puedo decir que mi aprendizaje realizando este trabajo ha sido bastante satisfactorio, dado que he podido ampliar mis conocimientos en programación y aplicarlos a lo aprendido durante el transcurso del grado.

10 Anexos

```
#\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\AQUÍ EMPIEZ EL LAUNCHER TKINTER/////////////////////////////////////////////
#\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\CORONALYZER 1.1.0//////////////////////////////////////////////////
root =tk.Tk()
root.iconbitmap("icon.ico")
root.title("Coronalyzer1.0")
root.geometry("700x650")
root.resizable(False,False)

menubar = tk.Menu(root)
root.config(bg="#0059b3", menu=menubar, width=300, height=300)
imagen = PhotoImage(file="corona.gif")
my_label=tk.Label(root, image=imagen)
my_label.place(x=0, y=0, relwidth=1, relheight=1,)
filemenu=tk.Menu(menubar, tearoff=0)
filemenu1=tk.Menu(menubar, tearoff=0)
filemenu.add_command(label="Sequence",command=Analizar_Seq)
filemenu.add_command(label="Compare 3 Seq.",command=Analizar_Mult)
filemenu.add_command(label="Compare 2 Seq.",command=Analizar_Sim)
filemenu.add_command(label="Mutations",command=Analizar_Mut)
filemenu.add_command(label="Protein Alignment local",command=protein_aln_local)
filemenu.add_command(label="Protein Alignment global",command=protein_aln_global)
filemenu.add_command(label="Protein Plot",command=protein_plots)
filemenu.add_command(label="MUSCLE",command=MUSCLE)
filemenu.add_command(label="Phylotree",command=phylotree)
filemenu.add_command(label="3D View",command=protein_view)
filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="Analysis", menu=filemenu)


miFrame=tk.Frame(root)
miFrame.pack()
miFrame.config(bg="grey")

miNombre1= tk.StringVar()
miNombre2= tk.StringVar()
miNombre3= tk.StringVar()
miNombre4=tk.StringVar()
miNombre5= tk.StringVar()

nombreID1=tk.Entry(miFrame, textvariable=miNombre1)
nombreID1.grid(row=1, column=1, padx=10, pady=10)

nombreID2=tk.Entry(miFrame, textvariable=miNombre2)
nombreID2.grid(row=2, column=1, padx=10, pady=10)

nombreID3=tk.Entry(miFrame, textvariable=miNombre3)
nombreID3.grid(row=3, column=1, padx=10, pady=10)

nombreID4=tk.Entry(miFrame, textvariable=miNombre4)
nombreID4.grid(row=4, column=1, padx=10, pady=10)

nombreID5=tk.Entry(miFrame, textvariable=miNombre5)
nombreID5.grid(row=5, column=1, padx=10, pady=10)
idLabel=tk.Label(miFrame, text="File 1:", width=10, height=1, borderwidth=3, relief="raised")
idLabel.grid(row=1, column=0, sticky="e", padx=10, pady=10)

idLabel=tk.Label(miFrame, text="File 2:", width=10, height=1, borderwidth=3, relief="raised")
idLabel.grid(row=2, column=0, sticky="e", padx=10, pady=10)

idLabel=tk.Label(miFrame, text="File 3:", width=10, height=1, borderwidth=3, relief="raised")
idLabel.grid(row=3, column=0, sticky="e", padx=10, pady=10)

idLabel=tk.Label(miFrame, text="Fasta File:", width=14, height=1, borderwidth=3, relief="raised")
idLabel.grid(row=4, column=0, sticky="e", padx=10, pady=10)

idLabel=tk.Label(miFrame, text="PDB File:", width=14, height=1, borderwidth=3, relief="raised")
idLabel.grid(row=5, column=0, sticky="e", padx=10, pady=10)

boton4=tk.Button(root, text="Clean all fields", command=limpiarCampos).pack()
boton3=tk.Button(root, text="Download Fasta", command=descargar_fasta).pack()
boton3=tk.Button(root, text="Download PDB", command=descargar_PDB).pack()


root.config(menu=menubar)

root.mainloop()
```

Figura 1. Código de la fusión Menú, Coronalyzer. Esta función contiene el código que ejecuta las diferentes funciones del programa a partir de la interfaz Tkinter, con botones de análisis, descarga, casillas de archivos y ventanas desplegables. Fuente: Programa.

```
'''Función descargar_fasta(), como su propio nombre indica nos descarga los fastas introduciendo el codigo de del NCBI
descarga los fasta y los fasta_cds'''

def descargar_fasta():
    global text4
    text4=nombreID4.get()
    print(text4)

    Entrez.email = "8tato8@gmail.com"
    hd1 = Entrez.efetch(db="nucleotide",id=[text4],rettype='fasta')
    seq = SeqIO.read(hd1,'fasta')
    fw = open(text4,'w')
    SeqIO.write(seq,fw,'fasta')


    handle = Entrez.efetch(db="nucleotide", id=[text4], rettype="fasta_cds_na", retmode="text")
    seq=handle.read()
    text_file = open(text4+'.txt', "w")
    n = text_file.write(seq)

    text_file.close()

    fw.close()
    os.getcwd()

    return(text4)

'''Función descargar_PDB, como su propio nombre indica descarga archivos del PDB introduciendo el codigo de 4 letras'''

def descargar_PDB():
    global text5
    text5=nombreID5.get()
    pdbl=PDBList()
    pdbl1=[text5]
    for i in pdbl1:
        pdbl.download_pdb_files(pdbl1, pdir='PDB', file_format='mmCif')

    for i in pdbl1:
        pdbl.download_pdb_files(pdbl1, pdir='PDB', file_format='pdb')

    return(text5)
```

Figura 2. Código de las funciones de descarga de archivos, Coronalyzer. Descarga archivos Fasta, CDS nucleotide Fasta, PDB y MMCIF. Estos archivos son descargados usando la librería Bio.Entrez y mediante la función PDBList de la librería Bio.pdb. Fuente: Programa.

```
def protein_plots():

    global text1

    text1=nombreID1.get()

    confProDy(auto_show=False)
    confProDy(auto_secondary=True)
    p38 = parsePDB(text1,compressed=False)

    showContactMap(p38.chain_A)

    betas = p38.chain_A.getBetas()
    plot(betas);
    xlabel('Residue index')
    ylabel('B-factor')

    chain = p38['A']
    Phi = []; Psi = []; c = []
    for res in chain.iterResidues():
        try:
            phi = calcPhi(res)
            psi = calcPsi(res)
        except:
            continue
        else:
            Phi.append(phi)
            Psi.append(psi)
            if res.getResname() == 'GLY':
                c.append('black')
            else:
                secstr = res.getSecstrs()[0]
                if secstr == 'H':
                    c.append('red')
                elif secstr == 'G':
                    c.append('darkred')
                elif secstr == 'E':
                    c.append('blue')
                else:
                    c.append('grey')

    return(text1)
```

Figura 3. Código de la función Gráficas de proteínas, Coronalyzer. Esta función genera el mapa de contacto, la gráfica del factor B y el diagrama de ramachadran de la proteína usando el módulo Prody. Fuente: Programa.

```
def Analizar_Seq():
    global text1

    text1=nombreID1.get()

    print(text1)

    coronavirus = open(text1, "r")

    print('HEADER:',coronavirus.readline())
    coronavirus = coronavirus.readlines()
    COVID_seq = ''
    for line in coronavirus:
        line = line.strip()
        COVID_seq += line
    print(COVID_seq[0:1000])

    def basic_properties(DNAseq):
        total_base = len(DNAseq)
        num_Adenine = DNAseq.count('A')
        num_Guanine = DNAseq.count('G')
        num_Thymine = DNAseq.count('T')
        num_Cytosine = DNAseq.count('C')

        if total_base != num_Adenine + num_Guanine + num_Thymine + num_Cytosine:
            print('Something is not right')
        else : pass

        A_percent = num_Adenine / total_base
        G_percent = num_Guanine / total_base
        T_percent = num_Thymine / total_base
        C_percent = num_Cytosine / total_base

        #visualization
        x = np.arange(4)
        bases = ['Adenine', 'Guanine', 'Thymine' ,'Cytosine']
        values = [num_Adenine, num_Guanine, num_Thymine, num_Cytosine]
        plt.bar(x,values)
        plt.xticks(x, bases)
        plt.show()
        table = [['total base',total_base,'Percentage',str('100%')],
                 ['Adenine:',num_Adenine, 'Percentage:',str(round(A_percent*100,2))+'%'],
                ['Guanine:',num_Guanine, 'Percentage:',str(round(G_percent*100,2))+'%'],
                 ['Thynime:',num_Thymine, 'Percentage:',str(round(T_percent*100,2))+'%'],
                 ['Cytosine:',num_Cytosine, 'Percentage:',str(round(C_percent*100,2))+'%']]
        print(tabulate(table))
        print('GC content:', round((((num_Guanine + num_Cytosine) / total_base)*100),2),'%')
    def transcription(DNAseq):
        mRNAseq = DNAseq.replace('T','U')
        print(len(mRNAseq))
        return mRNAseq
    COVID_mRNA = transcription(COVID_seq)
    print("-----------------------------------------------------mRNA Sequence-----------------------------------------------------")
    print(COVID_mRNA[0:1000])
    print("-----------------------------------------------------Protein Sequence-----------------------------------------------------")

    def translate(seq):
        table = {
            'ATA':'I', 'ATC':'I', 'ATT':'I', 'ATG':'M',
            'ACA':'T', 'ACC':'T', 'ACG':'T', 'ACT':'T',
            'AAC':'N', 'AAT':'N', 'AAA':'K', 'AAG':'K',
            'AGC':'S', 'AGT':'S', 'AGA':'R', 'AGG':'R',
            'CTA':'L', 'CTC':'L', 'CTG':'L', 'CTT':'L',
            'CCA':'P', 'CCC':'P', 'CCG':'P', 'CCT':'P',
            'CAC':'H', 'CAT':'H', 'CAA':'Q', 'CAG':'Q',
            'CGA':'R', 'CGC':'R', 'CGG':'R', 'CGT':'R',
            'GTA':'V', 'GTC':'V', 'GTG':'V', 'GTT':'V',
            'GCA':'A', 'GCC':'A', 'GCG':'A', 'GCT':'A',
            'GAC':'D', 'GAT':'D', 'GAA':'E', 'GAG':'E',
            'GGA':'G', 'GGC':'G', 'GGG':'G', 'GGT':'G',
            'TCA':'S', 'TCC':'S', 'TCG':'S', 'TCT':'S',
            'TTC':'F', 'TTT':'F', 'TTA':'L', 'TTG':'L',
            'TAC':'Y', 'TAT':'Y', 'TAA':'_', 'TAG':'_',
            'TGC':'C', 'TGT':'C', 'TGA':'_', 'TGG':'W',}
        protein =""
        for i in range(0, len(seq), 3):
            codon = seq[i:i + 3]
            if codon in table:
                protein+= table[codon]
        return protein

    protein_seq = translate(COVID_seq)
    print(protein_seq[0:1000])

    protein_seq = translate(COVID_seq)
    protein_seq[0:1000]
```

Figura 4. Código de la función Análisis de secuencia, Coronalyzer parte 1. Cálculo de propiedades básicas, transcripción y traducción mediante Biopython. Fuente: Programa.

```
def visualization(protein_seq):
    # composición de los Aminoácidos
    plt.figure(figsize=(35,50))
    x = np.arange(22)
    AA = ['Arginine', 'Histidie','Lysine',
          'Aspartic Acid','Glutamic Acid',
          'Serine','Threonine','Asparagine','Glutamine',
          'Cysteine','Selenocysteine','Glycine','Prolien',
          'Alanine','Valine','Isoleucine','Leucine',
          'Methionine','Phenylalanine','Tyrosine','Tryptophan',
          'Stop Codon']
    values = [protein_seq.count('R'),protein_seq.count('H'),protein_seq.count('K'),
              protein_seq.count('D'),protein_seq.count('E'),
              protein_seq.count('S'),protein_seq.count('T'),protein_seq.count('N'),protein_seq.count('Q'),
              protein_seq.count('C'),protein_seq.count('U'),protein_seq.count('G'),protein_seq.count('P'),
              protein_seq.count('A'),protein_seq.count('V'),protein_seq.count('I'),protein_seq.count('L'),
              protein_seq.count('M'),protein_seq.count('F'),protein_seq.count('Y'),protein_seq.count('W'),
              protein_seq.count('_')]
    plt.subplot(2,2,1)
    plt.rc('font',size = 20)
    plt.barh(AA,values,height=0.6)
    plt.title('AA in general')

    # Visualización de grupos segun su carga
    x = np.arange(4)
    Electric =  protein_seq.count('R')+protein_seq.count('H')+protein_seq.count('K')+protein_seq.count('D')+protein_seq.coun
    Uncharged = protein_seq.count('S')+protein_seq.count('T')+protein_seq.count('N')+protein_seq.count('Q')
    Special =   protein_seq.count('C')+protein_seq.count('U')+protein_seq.count('G')+protein_seq.count('P')
    Hydrophobic = protein_seq.count('A')+protein_seq.count('V')+protein_seq.count('I')+protein_seq.count('L')+protein_seq.co

    plt.subplot(2,2,2)
    types = ['Elecrically charged','Polar uncharged',
             'Special case', 'Hydrophobic Side Chain']
    values = [Electric,Uncharged,Special,Hydrophobic]
    plt.barh(types, values, height = 0.6)
    plt.title('AA in groups')

    # Visualización de los aminoacidos positivos y negativos en grupos de cadena laterales hidrofóbicas
    plt.subplot(2,2,3)
    x = np.arange(2)
    positive = protein_seq.count('R')+protein_seq.count('H')+protein_seq.count('K')
    negative = protein_seq.count('D')+protein_seq.count('E')
    types = ['Positive','Negative']
    values = [positive, negative]
    plt.barh(types,values, height=0.6)
    plt.title('Charge diff in Electrically charged side chain')
    plt.show()

    # Visualizacion de la abundancia de Aminoacidos segun su estructura secundaria en la cadena proteica
    alpha_helix = protein_seq.count('A')+protein_seq.count('C')+protein_seq.count('L')+protein_seq.count('M')+protein_seq.co
    beta_sheet =protein_seq.count('V')+protein_seq.count('I')+protein_seq.count('F')+protein_seq.count('Y')+protein_seq.coun
    turn = protein_seq.count('G')+protein_seq.count('S')+protein_seq.count('D')+protein_seq.count('N')+protein_seq.count('P'
    x = np.arange(3)
    plt.subplot(2,2,4)
    types = ['Alpha helix','beta sheet','Turn']
    values = [alpha_helix, beta_sheet, turn]
    plt.barh(types,values, height=0.6)
    plt.title('AA residues in secondary structure')
    plt.show()
print("-----------------------------------------------------Protein Splicing-----------------------------------------------
def proteinseq_splic(protein_sequence):
    protein_group = protein_sequence.split('_')
    return protein_group
protein_list = proteinseq_splic(protein_seq)
print(proteinseq_splic(protein_seq)[0:100])
print('Number of proteins:',len(protein_list))

print("-----------------------------------------------------Functional Protein-----------------------------------------------
def Functional_protein(protein_list):
    funct_group = []
    for protein in protein_list:
        if len(protein) > 100:
            funct_group.append(protein)
    return funct_group
function_protein = Functional_protein(protein_list)
print(function_protein[0:52])
print('Number of Functional proteins:',len(function_protein))

print("-----------------------------------------------------DNA OPEN READING FRAME-----------------------------------------------

record = SeqIO.read(text1, "fasta")
table = 11
min_pro_len = 100
for strand, nuc in [(+1, record.seq), (-1, record.seq.reverse_complement())]:
     for frame in range(3):
         length = 3 * ((len(record)-frame) // 3) #Multiple of three
         for pro in nuc[frame:frame+length].translate(table).split("*"):
             if len(pro) >= min_pro_len:
                 print("%s...%s - length %i, strand %i, frame %i" \
                       % (pro[:30], pro[-3:], len(pro), strand, frame))
```

Figura 5. Código de la función Análisis de secuencia, Coronalyzer parte 2. Visualización de propiedades básicas, splicing, proteína funcional y ORF mediante Biopython. Fuente: Programa.

```
class dna:

    # metodo cosntructor
    def __init__(self,dna_seq):
        dna_seq = dna_seq.upper() # Convirte bases en mayusculas
        for seq in dna_seq:
            # Valida las bases. Si la secuencia no es valida salta un error
            if seq not in ['A','T','G','C',' ','N']:
                error = 'Wrong DNA Sequence {}!!'.format(seq)
                raise ValueError(error)
        # Quita todos los caracteres vacios de la secuencia.
        dna_seq = dna_seq.replace(' ','')
        self.dir_3_5=dna_seq

    def __repr__(self):
        return "DNA has {} nucleotide and they are {} :".format(self.nucl_len,self.dir_3_5)

    def __eq__(self, other):
        if other is None:
            return False
        return self.seq == other.seq

    def numpfy(self):
        # Este metodo comvierte el Dna en un numpy array.
        # Cada nucleotido de la secuencia se traduce a un numero que vemos a continuación
        # Esto puede ser usado para analisis o comparación, como el One-Hot-Encoding.
        arr = ''
        for i in self.dir_3_5:
            if i == 'A':
                arr += '0 '
            if i == 'T':
                arr += '255 '
            if i == 'C':
                arr += '100 '
            if i == 'G':
                arr += '200 '
            if i == 'N':
                arr += '75 '
        arr_np = np.fromstring(arr,dtype=np.uint8,sep=' ')
        self.num_array = arr_np
        return self.num_array
def read_dna_seq(file_name):
    global fil
    # Este metodo lee la secuencia fasta descargada del NCBI y crea un diccionario con ella.
    try:
        fil = open(file_name,'r')
        fil_list = fil.readlines()
        fil.close
    except: FileNotFoundError()

    genome = {}
    gene_name = ''
    protein_name = ''
    gene_seq = ''
    for i in fil_list:
        if i[0] == '>':
            # Leemos cada linea del archivo y creamos un diccionario con la siguiente información
            # gene. {<'gene_name-1'>:[<protein_name>,nucleotide sequence],
            #        <'gene_name-2'>:[<protein_name>,nucleotide sequence],
            #        <'gene_name-2'>:[<protein_name>,nucleotide sequence]}
            if list(genome.keys()) != []:
                gene_seq = gene_seq.replace('\n','')
                genome[gene_name].append(gene_seq)
            gene_seq = ''
            g_st = i.find('[gene=')
            g_end = i[g_st:].find(']')
            p_st = i.find('[protein=')
            p_end = i[p_st:].find(']')

            if g_st > 0 and g_end > 0:
                gene_name = i[g_st+1:g_st+g_end]
                genome[gene_name] = []

            if p_st > 0 and p_end > 0:
                protein_name = i[p_st+1:p_st+p_end]
                genome[gene_name].append(protein_name)
        else:
            gene_seq += i
    gene_seq = gene_seq.replace('\n','')
    genome[gene_name].append(gene_seq)
    return genome

def gene_mod(genome):
    # Dummy nucleotide 'N' para compactar la numpy array.
    genome_keys = list(genome.keys())
    for k in genome_keys:
        if len(numpy_image_dict[k]) > 1:
            N = numpy_image_dict[k][1]
            seq = add_N(N,genome[k][1])
            genome[k][1] = seq
    return genome

def add_N(n,seq):
    # Esta funcion la usa gene_mod para añadir dummy 'N'.
    for i in range(0,n):
        seq += 'N'
    return seq
```

Figura 6. Clase y funciones auxiliares de la función Análisis de mutaciones, Coronalyzer. Clase DNA, método por el cual se construyen las matrices Numpy para procesa las secuencias de ADN. Función read_dna_seq, se encarga de leer los archivos CDS nucleotide Fasta y construir la clase DNA. Las funciones gene_mod y add_N se encargan de ajustar los diccionarios. Fuente: Programa.

```
def Analizar_Mut():
    global text1
    global text2
    text1=nombreID1.get()
    text2=nombreID2.get()
    print(text1,text2)

    # Este diccionario se usa para establecer el tamaño de la numpy array por cada uno de los genes.
    # La numpy array del gen  ORF1ab se establecera con rows=115 y cols=115.*
    numpy_image_dict = {'gene=ORF1ab':[(115,115),7],
                        'gene=S':[(62,62),22],
                        'gene=ORF3a':[(28,30),12],
                        'gene=E':[(15,16),12],
                        'gene=M':[(26,27),33],
                        'gene=ORF6':[(14,14),10],
                        'gene=ORF7a':[(19,20),14],
                        'gene=ORF7b':[(12,12),12],
                        'gene=ORF8':[(19,20),14],
                        'gene=N':[(36,36),36],
                        'gene=ORF10':[(11,11),4]}

    # Lee la secuencia 1 de DNA descargada anteriormente del NCBI.
    dict_seq_1 = read_dna_seq(text1+'.txt')
    # Modifica la secuencia con dummy 'N' nucleotide.
    dict_seq_1 = gene_mod(dict_seq_1)

    # Lee la secuencia 2 de DNA descargada anteriormente del NCBI.
    dict_seq_2 = read_dna_seq(text2+'.txt')
    # Modifica la secuencia con dummy 'N' nucleotide.
    dict_seq_2 = gene_mod(dict_seq_2)

    # Creamos plots con Matplotlib para cada gen.
    f,ax = plt.subplots(nrows=11,ncols=3,figsize=(25,30))
    gene_name = list(numpy_image_dict.keys())
    row = 0
    col = 0
    mut_dict={}
    for i in gene_name:
        G = i[5:]
        # Loopeamos a traves de cada gen de la secuencia de Coronavirus.
        gene_us = dna(dict_seq_1['gene='+G][1])
        # Llamamos al metodo que convierte gene la secuencia de genes en un numpy array.
        numpfy_usa = gene_us.numpfy()
        # Reestablecemos las dimensiones del numpy array.
        numpfy_usa = numpfy_usa.reshape(numpy_image_dict['gene='+G][0])
        # Hacemos plot del array.
        ax[row][col].pcolor(numpfy_usa)
        ax[row][col].set_title(G+' Gene'+text1)
        col+=1
        gene_china = dna(dict_seq_2['gene='+G][1])
        # Llamamos el metodo que convierte gene la secuencia de genes en un numpy array.
        numpfy_china = gene_china.numpfy()
        # Reestablecemos las dimensiones del numpy array.
        numpfy_china = numpfy_china.reshape(numpy_image_dict['gene='+G][0])
        # Hacemos el plot de la array
        ax[row][col].pcolor(numpfy_china)
        ax[row][col].set_title(G+' Gene'+text2)
        col+=1
```

Figura 7. Código de la función Análisis de mutaciones, Coronalyzer parte 1. Esta parte del código se encarga de leer las secuencias y ajustarlas a los diccionarios Numpy con las funciones auxiliares. Una vez leídas las secuencias se genera un plot a través de cada diccionario. Fuente: Programa.

```
        # Para encontrar las mutaciones lo que hacemos es restar la nueva secuencia a la  secuencia base.
        mut = numpfy_china - numpfy_usa
        if mut.any():
            alue_of_newer_seq>, <value_in_mutated_numpy>, (x_value,y_value)]]}
            mut_nec = np.nonzero(mut)
            x=mut_nec[0]
            y=mut_nec[1]
            l=0
            mut_dict[G]=[]
            for i in x:
                us_base = numpfy_usa[i][y[l]]
                ch_base = numpfy_china[i][y[l]]
                mut_base = mut[i][y[l]]
                info_list = [ch_base,us_base,mut_base,(i,y[l])]
                mut_dict[G].append(info_list)
                cbase = 0
                usbase = 0
                dims = 0
                pos = 0
                posp = 0
                posh = 0

                if ch_base == 0:
                    cbase = 'A'
                elif ch_base == 255:
                    cbase = 'T'
                elif ch_base == 100:
                    cbase = 'C'
                elif ch_base == 200:
                    cbase = 'G'

                if us_base == 0:
                    usbase = 'A'
                elif us_base == 255:
                    usbase = 'T'
                elif us_base == 100:
                    usbase = 'C'
                elif us_base == 200:
                    usbase = 'G'


                if G == 'ORF1ab':
                    dims == 115
                elif G == 'S':
                    dims = 62
                elif G == 'ORF3a':
                    dims = 28
                elif G == 'E':
                    dims = 15
                elif G == 'M':
                    dims = 26
                elif G == 'ORF6':
                    dims = 14
                elif G == 'ORF7a':
                    dims = 19
                elif G == 'ORF7b':
                    dims = 12
                elif G == 'ORF8':
                    dims = 19
                elif G == 'N':
                    dims = 36
                elif G == 'ORF10':
                    dims = 11

                pos = dims*i+y[l]
                posh = pos/3+1
                posp = round(posh)

                print("Mutated DNA Base {} in {} and Base {} in {} at position {} / {} / {} For the Gene {} \n".format(cbase,tex
                l+= 1

        # Poniendole título al plot
        ax[row][col].pcolor(mut)
        ax[row][col].set_title(G+' Gene - Mutation')
        row+= 1
        col=0

    f.tight_layout()
    # Guardamos el matplotlib subplot como un jpg.
    f.savefig('Sars_Cov-2_Gene_Mutation.jpg')
    img = mpimg.imread('Sars_Cov-2_Gene_Mutation.jpg')
    imgplot = plt.imshow(img)
    #plt.show()

    return (text1,text2,text3)
```

Figura 8. Código de la función Análisis de mutaciones, Coronalyzer Parte 2. Esta parte del código resta los diccionarios Numpy y busca valores diferentes a 0 mediante un bucle for. También se encarga de mostrar los datos y los plots a través de la terminal. Uso del módulo Biopython, Numpy y Matplotlib. Fuente: Programa.

```
def protein_aln_local():

    global text1
    global text2

    text1=nombreID1.get()
    text2=nombreID2.get()
    print(text1,text2)

    seq1 = SeqIO.read(text1, 'fasta')
    seq2 = SeqIO.read(text2, 'fasta')


    print("------------------------------------------------------------PROTEIN ALIGNMENT-----------------------------------------

    blosum62 = substitution_matrices.load("BLOSUM62")
    alignments = pairwise2.align.localds(seq1.seq, seq2.seq, blosum62, -10, -1)
    print(pairwise2.format_alignment(*alignments[0]))

    return(text1,text2)

def protein_aln_global():

    global text1
    global text2

    text1=nombreID1.get()
    text2=nombreID2.get()
    print(text1,text2)

    seq1 = SeqIO.read(text1, 'fasta')
    seq2 = SeqIO.read(text2, 'fasta')

    print("------------------------------------------------------------PROTEIN ALIGNMENT-----------------------------------------

    blosum62 = substitution_matrices.load("BLOSUM62")
    alignments = pairwise2.align.globalds(seq1.seq, seq2.seq, blosum62, -10, -0.5)
    print(pairwise2.format_alignment(*alignments[0]))


    return(text1,text2)
```

Figura 9. Código de la función Alineamiento de proteínas, Coronalyzer. En estas funciones se usa la función pairwise y una matriz Blosum 62. Las diferencias se encuentra entre la modalidad de alineación y los parámetros de puntuación. Fuente: Programa.

```
def Analizar_Sim():
    global text1
    global text2

    text1=nombreID1.get()
    text2=nombreID2.get()
    print(text1,text2)

    Seq1 = SeqIO.read(text1, 'fasta')
    Seq2 = SeqIO.read(text2, 'fasta')

    Seq2_Seq1 = pairwise2.align.globalxx(Seq2.seq, Seq1.seq, one_alignment_only=True, score_only=True)
    print('Seq2/Seq1 Similarity (%):', Seq2_Seq1 / len(Seq1.seq) * 100)


    alignments = pairwise2.align.localms(Seq1.seq, Seq2.seq, 1, -2, -2, -0.5)
    print(pairwise2.format_alignment(*alignments[0]))

      # Printeamos los plots con la data
    X = ['Seq2/Seq1']
    Y = [Seq2_Seq1/len(Seq1.seq)*100]
    plt.title('Sequence identity (%)')
    plt.bar(X,Y,color=(0.2, 0.4, 0.6, 0.6))
    plt.show()
    return (text1,text2)

def Analizar_Mult():
    global text1
    global text2
    global text3

    text1=nombreID1.get()
    text2=nombreID2.get()
    text3=nombreID3.get()
    print(text1,text2,text3)

    Seq1 = SeqIO.read(text1, 'fasta')
    Seq2 = SeqIO.read(text2, 'fasta')
    Seq3 = SeqIO.read(text3, 'fasta')

    Seq1_Seq3 = pairwise2.align.globalxx(Seq1.seq, Seq3.seq, one_alignment_only=True, score_only=True)
    print('Seq1/Seq3 Similarity (%):', Seq1_Seq3 / len(Seq1.seq) * 100)

    Seq2_Seq3 = pairwise2.align.globalxx(Seq2.seq, Seq3.seq, one_alignment_only=True, score_only=True)
    print('Seq2/Seq3 Similarity (%):', Seq2_Seq3 / len(Seq2.seq) * 100)

    Seq2_Seq1 = pairwise2.align.globalxx(Seq2.seq, Seq1.seq, one_alignment_only=True, score_only=True)
    print('Seq2/Seq1 Similarity (%):', Seq2_Seq1 / len(Seq1.seq) * 100)

    # Printeamos la data
    X = ['Seq1/Seq3', 'Seq2/Seq3', 'Seq2/Seq1']
    Y = [Seq1_Seq3/ len(Seq1.seq) * 100, Seq2_Seq3/ len(Seq2.seq)*100, Seq2_Seq1/len(Seq1.seq)*100]
    plt.title('Sequence identity (%)')
    plt.bar(X,Y,color=(0.2, 0.4, 0.6, 0.6))
    plt.show()

    return (text1,text2,text3)
```

Figura 10. Código de la función Comparación de secuencias, Coronalyzer. En estas funciones se usa la función pairwise para alinear secuencias genómicas y generar gráficas. Fuente: Programa.

```
def MUSCLE():

    global text1
    text1=nombreID1.get()

    in_file = "C:/Users/hades/Coronalyzer/"+text1
    out_file = "C:/Users/hades/Coronalyzer/aligned.fasta"
    muscle_exe = "C:/Users/hades/Coronalyzer/muscle3.8.31_i86win32.exe"
    cline = MuscleCommandline(muscle_exe, input=in_file, phyiout=out_file)
    stdout, stderr = cline(in_file)
    align = AlignIO.read(out_file, "phylip")
    print(align)
```

Figura 11. Código Función Muscle. Ejecuta la herramienta MUSCLE descargada en la carpeta raíz del programa. Fuente: Programa.

```python
def visualizar_proteina():
    global structure
    global text1

    text1=nombreID1.get()

    #Encuentra el archivo PDB
    parser = MMCIFParser()
    structure = parser.get_structure(text1, text1+'.cif')
    structure
    structure[0]
    #identifica el numero d cadenas
    for chain in structure[0]:
        print(f'chain ID: {chain.id}')
    nv.demo()
    view = nv.show_biopython(structure)
    view.render_image()
    return (view, structure)

visualizar_proteina()

#GUI
nv.show_biopython(structure, gui=True)
```

Figura 12. Código de la función visualización 3D, Coronalyzer. Estructura 3D a partir de un archivo MMCIF. Fuente: Programa.

```python
def phylotree():

    global text1
    text1=nombreID1.get()

    # Cargue el nuevo archivo de alineación en su carpeta o directorio de trabajo
    # Abra el archivo de alineación como un objeto MultipleSeqAlignment
    with open(text1,"r") as aln:
        alignment = AlignIO.read(aln,"clustal")
    print(type(alignment))

    # Abra e inicie la Calculadora de distancia utilizando el modelo de identidad
    from Bio.Phylo.TreeConstruction import DistanceCalculator
    calculator = DistanceCalculator('identity')
    # Escribe la matriz de distancias
    distance_matrix = calculator.get_distance(alignment)
    print(distance_matrix)

    # Abrir e iniciar el Constructor de árboles
    from Bio.Phylo.TreeConstruction import DistanceTreeConstructor
    constructor = DistanceTreeConstructor(calculator)
    # Build the tree
    variant_tree = constructor.build_tree(alignment)
    variant_tree.rooted = True
    print(variant_tree)

    # Guardar el árbol en un nuevo archivo
    Phylo.write(variant_tree, "Cov_tree.xml", "phyloxml")

    # Importar matplotlib y crear un árbol básico
    import matplotlib
    import matplotlib.pyplot as plt
    variant_tree.clade[0, 0].color = "blue"
    variant_tree.root.color = "red"
    fig = Phylo.draw(variant_tree)

    # Importar matplotlib y crear un árbol básico
    fig = plt.figure(figsize=(13, 5), dpi=100)
    matplotlib.rc('font', size=12)
    matplotlib.rc('xtick', labelsize=10)
    matplotlib.rc('ytick', labelsize=10)
    axes = fig.add_subplot(1, 1, 1)
    Phylo.draw(variant_tree, axes=axes)
    fig.savefig("Covid_cladogram")
```

Figura 13. Código dela función Phylotree, Coronalyzer. Usando Bio.Phylo del módulo de Biopython se cálculan los parámetros necesarios para generar una gráfica filogenética a partir un archivo MSA. Fuente: Programa.

Figura 14. Menú y parámetros visualización 3D, Coronalyzer. Esta imagen se tomó del entorno de desarrollo Jupyterlab. Fuente: Programa.

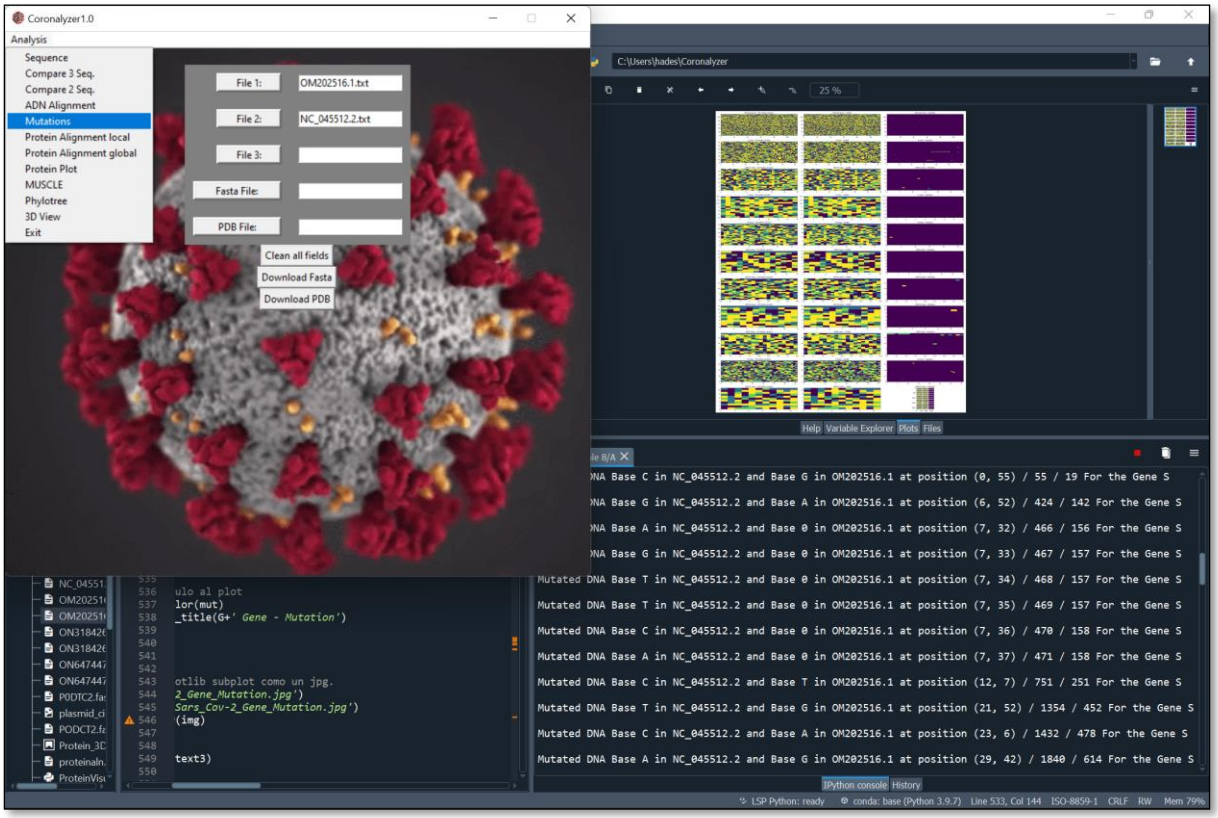


Figura 15. Ejecución del programa Coronalyer en el entorno Spyder. En este entorno podemos ejecutar el programa y visualizar gráficas y el terminal en las ventanas de ejecución. Fuente: Programa.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import math
from tabulate import tabulate
import warnings
warnings.filterwarnings("ignore")
pd.plotting.register_matplotlib_converters()
import seaborn as sns
import Bio
from Bio import pairwise2
from Bio import SeqIO
from Bio import AlignIO
from Bio import Entrez
from Bio import ExPASy
from Bio.pairwise2 import format_alignment
from Bio.PDB import MMCIFParser
#from Bio import Phylo
from tkinter import *
import tkinter as tk
from prody import *
from numpy import *
from matplotlib.pyplot import *
import matplotlib.image as mpimg
import pylab
import pandas as pd
import nglview as nv
```

Figura 16. Listado de librerías y módulos usados en el programa. Fuente: Programa.

ANNEX 2

FITXA DE SEGUIMENT DEL TUTOR/A del TFG

Nom i Cognoms de l'Alumne/a: Pedro Manuel López Zarzuela

Nom i Cognoms del Tutor/a: Lluis Masip Vernis

Data de la entrevista amb l'alumne: 02-03-2022

Recomanacions durant el seguiment:

- Discussió sobre la planificació i terminis per les diferents parts del treball de fi de grau.
- Discussió i recomanacions sobre els continguts i l'extensió del treball de fi de grau.
- Discussió sobre com procedir amb la revisió de la memòria escrita.
- Discussió específica sobre la memòria i els resultats obtinguts.

Observacions:

- El tutor considera que la memòria escrita reflexa un treball original desenvolupat per l'alumne.
- L'alumne sempre ha tingut una actitud adequada i ha complert els terminis establerts.

Observacions Darrera revisió:

- Comentaris sobre els continguts de l'última versió de la memòria escrita.

Signatura del Tutor/a

Signatura del Alumne/a

Tarragona a 25 de maig 2022