Centro de Ciencias Básicas

Departamento de Sistemas de Información

Tesis

“Aplicación de Metaheurísticas Para Identificación de Patrones en Secuencias de Proteínas”

Presenta

Ing. Walbert Rivero Maceo

Para Obtener El Grado De Maestro En Informática Y Tecnologías Computacionales

Comité Tutoral

Mtro. Jorge Eduardo Macías Luévano

Dra. Eunice Esther Ponce de León Sentí

Dra. María Dolores Torres Soto

Aguascalientes, Ags, 29 de noviembre de 2024

Autorizaciones.

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

CARTA DE VOTO APROBATORIO INDIVIDUAL

M. en C. Jorge Martín Alférez Chávez
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

PRESENTE

Por medio del presente como ***TUTOR*** designado del ***WALBERT RIVERO MACEO*** con ID 354273 quien realizó *la tesis* titulada: ***APLICACIÓN DE METAHEURÍSTICAS PARA IDENTIFICACIÓN DE PATRONES EN SECUENCIAS DE PROTEÍNAS***, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirla así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

A T E N T A M E N T E
"Se Lumen Proferre"
Aguascalientes, Ags., a 29 días del mes de noviembre de 2024.

Dra. Eunice Esther Ponce de León Senti
Co-Tutor de *tesis*

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

Elaborado por: Depto. Apoyo al Posgrado.
Revisado por: Depto. Control Escolar/Depto. Gestión de Calidad.
Aprobado por: Depto. Control Escolar/ Depto. Apoyo al Posgrado.

uaa.mx /
Código: DO-SEE-FO-07
Actualización: 01
Emisión: 17/05/19

CARTA DE VOTO APROBATORIO
INDIVIDUAL

M. en C. Jorge Martín Alférez Chávez
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

PRESENTE

Por medio del presente como ***TUTOR*** designado del ***WALBERT RIVERO MACEO*** con ID 354273 quien realizó *la tesis* titulada: ***Aplicación de Metaheurísticas para Identificación de Patrones en Secuencias de Proteínas***, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirla así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 29 días del mes de noviembre de 2024.

M.I.T.C. Jorge Eduardo Macías Luévano
Co-Tutor de *tesis*

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

CARTA DE VOTO APROBATORIO
INDIVIDUAL

M. en C. Jorge Martín Alférez Chávez
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

PRESENTE

Por medio del presente como ***TUTOR*** designado del ***WALBERT RIVERO MACEO*** con ID 354273 quien realizó *la tesis* titulada: ***Aplicación de Metaheurísticas para Identificación de Patrones en Secuencias de Proteínas***, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirla así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 29 días del mes de noviembre de 2024.

Dra. María Dolores Torres Soto
Asesor de *tesis*

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

universidad autónoma de aguascalientes

DICTAMEN DE LIBERACION ACADEMICA PARA INICIAR LOS TRAMITES DEL EXAMEN DE GRADO

Fecha de dictaminación dd/mm/aaaa: 29/11/24

NOMBRE: WALBERT RIVERO MACEO **ID** 354273

PROGRAMA: MAESTRIA EN INFORMATICA Y TECNOLOGIAS COMPUTACIONALES **LGAC (del posgrado):** INGENIERIA DE SISTEMAS DECISIONALES PARA MEJORAR PROCESOS ORGANIZACIONALES

TIPO DE TRABAJO: (X) Tesis ( ) Trabajo Práctico

TITULO: APLICACIÓN DE METAHEURISTICAS PARA IDENTIFICACION DE PATRONES EN SECUENCIAS DE PROTEINAS

IMPACTO SOCIAL (señalar el impacto logrado): PROFUNDIZACION EN EL CONOCIMIENTO DE LA FAMILIA CORONAVIRIDAE Y EL IMPACTO EN LA SALUD GLOBAL

INDICAR SI NO N.A. (NO APLICA) SEGÚN CORRESPONDA:

	Elementos para la revisión académica del trabajo de tesis o trabajo práctico:
SI	El trabajo es congruente con las LGAC del programa de posgrado
SI	La problemática fue abordada desde un enfoque multidisciplinario
SI	Existe coherencia, continuidad y orden lógico del tema central con cada apartado
SI	Los resultados del trabajo dan respuesta a las preguntas de investigación o a la problemática que aborda
SI	Los resultados presentados en el trabajo son de gran relevancia científica, tecnológica o profesional según el área
SI	El trabajo demuestra más de una aportación original al conocimiento de su área
SI	Las aportaciones responden a los problemas prioritarios del país
SI	Generó transferencia del conocimiento o tecnológica
SI	Cumple con la ética para la investigación (reporte de la herramienta antiplagio)
	El egresado cumple con lo siguiente:
SI	Cumple con lo señalado por el Reglamento General de Docencia
SI	Cumple con los requisitos señalados en el plan de estudios (créditos curriculares, optativos, actividades complementarias,estancia, predoctoral, etc)
SI	Cuenta con los votos aprobatorios del comité tutoral, en caso de los posgrados profesionales si tiene solo tutorpodrá liberar solo el tutor
SI	Cuenta con la carta de satisfacción del Usuario
SI	Coincide con el título y objetivo registrado
SI	Tiene congruencia con cuerpos académicos
SI	Tiene el CVU del Conacyt actualizado
N.A.	Tiene el artículo aceptado o publicado y cumple con los requisitos institucionales (en caso que proceda)
	En caso de Tesis por artículos científicos publicados
N.A.	Aceptación o Publicación de los artículos según el nivel del programa
N.A.	El estudiante es el primer autor
N.A.	El autor de correspondencia as el Tutor del Núcleo Académico Básico
N.A.	En los artículos se ven reflejados los objetivos de la tesis, ya que son producto de este trabajo de investigación.
N.A.	Los artículos integran los capítulos de la tesis y se presentan en el idioma en que fueron publicados
N.A.	La aceptación o publicación de los artículos en revistas indexadas de alto impacto

Con base a estos criterios, se autoriza se continúen con los trámites de titulación y programación del examen de grado: Si X / No

FIRMAS

Elaboró:

* NOMBRE Y FIRMA DEL CONSEJERO SEGÚN LA LGAC DE ADSCRIPCION: DRA. MA. DOLORES TORRES SOTO

NOMBRE Y FIRMA DEL SECRETARIO TÉCNICO: DRA. LIZETH TZIQUERY SOLANO ROMO

* En caso de conflicto de intereses, firmará un revisor miembro del NAB de la LGAC correspondiente distinto al tutor o miembro del comité tutoral, asignado por el Decano

Revisó:

NOMBRE Y FIRMA DEL SECRETARIO DE INVESTIGACIÓN Y POSGRADO: DR. ALEJANDRO PADILLA DÍAZ

Autorizó:

NOMBRE Y FIRMA DEL DECANO: M. EN C. JORGE MARTÍN ALFÉREZ CHÁVEZ

Nota: procede el trámite para el Depto. de Apoyo al Posgrado

En cumplimiento con el Art. 105C del Reglamento General de Docencia que a la letra señala entre las funciones del Consejo Académico: ... Cuidar la eficiencia terminal del programa de posgrado y el Art. 105F las funciones del Secretario Técnico, llevar el seguimiento de los alumnos.

Agradecimientos.

Deseo expresar mi más sincero agradecimiento al Consejo Nacional de Ciencia y Tecnología (CONACyT) por la beca otorgada, la cual fue fundamental para la realización de mis estudios de posgrado y el desarrollo de este trabajo académico. Su apoyo ha sido un pilar esencial en mi formación profesional.

Agradezco profundamente a mis maestros y mentores, en especial a la Dra. Eunice Esther Ponce de León Sentí y al Mtro. Jorge Eduardo Macías Luévano, por su invaluable guía, paciencia y dedicación a lo largo de este proceso. Sus conocimientos, consejos y motivación constante han sido una fuente de inspiración que me impulsó a superar desafíos y alcanzar este logro.

A cada uno de ustedes, mi eterna gratitud.

Dedicatorias.

A mi madre, el pilar de mi vida.

Índice General

Ínice de Figuras

Índice de tablas

Resumen

La familia Coronaviridae ha captado la atención mundial debido a la pandemia provocada por el SARS-CoV-2, pero su relevancia científica abarca mucho más que este evento. Aunque la mayoría de los virus de esta familia utilizan animales como huéspedes, especialmente murciélagos, su diversidad y capacidad para adaptarse a nuevos hospedadores representan un desafío para la biología molecular y la salud pública. Este trabajo tiene como objetivo proponer un modelo computacional que clasifique proteínas de la familia Coronaviridae en géneros con base en patrones evolutivos y estructurales, empleando métricas integradoras de similitud y técnicas avanzadas de optimización combinatoria.

El problema se aborda como una tarea multiobjetivo y se utiliza un algoritmo genético (AG) para clasificar las proteínas, seleccionando las estructuras tridimensionales más representativas generadas por AlphaFold. Durante el desarrollo del proyecto, fue necesario construir flujos de trabajo automatizados para la adquisición y el procesamiento de datos estructurales, lo que mejoró la eficiencia de las metodologías aplicadas. Las secuencias y modelos tridimensionales de 46 proteínas Spike fueron obtenidos de repositorios públicos reconocidos, garantizando la calidad y confiabilidad de la información utilizada.

En el primer capítulo se establece el marco teórico, que abarca conceptos clave como optimización, metaheurísticas y aspectos fundamentales de la biología de proteínas. El segundo capítulo describe detalladamente el proceso de obtención, normalización y preparación de los datos, incluyendo el desarrollo de herramientas computacionales específicas para calcular métricas de similitud estructural. En el tercer capítulo se presenta el modelo propuesto, destacando la formulación del problema, los fundamentos del algoritmo genético y las funciones objetivo diseñadas para la clusterización de proteínas.

La sección experimental documenta la implementación del modelo, mostrando resultados que incluyen clasificaciones coherentes y agrupaciones evolutivamente relevantes. Los experimentos realizados evidenciaron el potencial del AG para minimizar distancias estructurales dentro de los géneros y maximizar distancias entre ellos, aunque también se destacaron los desafíos computacionales derivados del gran número de comparaciones requeridas.

Finalmente, los resultados obtenidos fueron validados mediante la construcción de un árbol filogenético utilizando el método Neighbor-Joining (NJ) en el software MEGA. Este análisis reveló una alta concordancia entre las agrupaciones generadas por el AG y las relaciones filogenéticas tradicionales, destacando las fortalezas del enfoque propuesto. Este trabajo contribuye a la comprensión de patrones estructurales y evolutivos en proteínas de la familia Coronaviridae, y sienta las bases para futuras investigaciones que integren biología computacional, optimización y proteómica en el estudio de organismos virales.

Abstract

The Coronaviridae family has garnered global attention due to the SARS-CoV-2 pandemic, but its scientific relevance extends far beyond this event. While most viruses in this family use animals, particularly bats, as hosts, their diversity and ability to adapt to new hosts present significant challenges to molecular biology and public health. This study proposes a computational model to classify proteins from the Coronaviridae family into genera based on evolutionary and structural patterns, employing integrative similarity metrics and advanced combinatorial optimization techniques.

The problem is addressed as a multi-objective task, utilizing a genetic algorithm (GA) to classify proteins and select the most representative three-dimensional structures predicted by AlphaFold. Automated workflows were developed to acquire and process structural data, enhancing the efficiency of applied methodologies. The sequences and 3D models of 46 Spike proteins were obtained from reputable public repositories, ensuring the quality and reliability of the data.

The theoretical framework covers key concepts such as optimization, metaheuristics, and fundamental aspects of protein biology. The data preparation process included obtaining, normalizing, and structuring data while developing computational tools to calculate structural similarity metrics. The proposed model focuses on problem formulation, GA foundations, and objective functions tailored for protein clustering.

Experimental results demonstrate the model's ability to produce consistent classifications and evolutionary-relevant groupings. The GA effectively minimizes structural distances within genera and maximizes distances between them, despite computational challenges due to the extensive number of comparisons required. Validation through a phylogenetic tree constructed using the Neighbor-Joining (NJ) method in MEGA revealed high concordance between the GA-generated clusters and traditional phylogenetic relationships.

This work contributes to understanding structural and evolutionary patterns in Coronaviridae proteins and establishes a foundation for future research integrating computational biology, optimization, and proteomics in the study of viral organisms.

Introducción:

En el ámbito de la biología computacional, la identificación de patrones en secuencias de proteínas representa un desafío multidisciplinario que requiere la integración de técnicas avanzadas de optimización y modelado. La aplicación de algoritmos heurísticos y metaheurísticos ha demostrado ser una estrategia eficaz para abordar problemas complejos, particularmente en escenarios donde los métodos exactos son inviables debido a la dimensión del espacio de búsqueda o a restricciones computacionales. Estas técnicas permiten explorar soluciones aproximadas que equilibran precisión y eficiencia, proporcionando respuestas prácticas a problemas de optimización combinatoria (Martí, 2001).

La presente investigación propone un modelo computacional que aplica metaheurísticas para analizar secuencias de aminoácidos y clasificar proteínas Spike de la familia Coronaviridae en géneros. Este modelo tiene como objetivo optimizar dos aspectos fundamentales: minimizar las distancias estructurales dentro de cada género y maximizar las distancias entre géneros, reflejando patrones evolutivos y estructurales de las proteínas(E. Ponce-de-Leon-Senti et al., 2017). Este enfoque aborda desafíos significativos en la representación tridimensional de las proteínas y el análisis de sus similitudes, aprovechando métricas estructurales generadas por herramientas avanzadas como AlphaFold y ChimeraX.(E. E. Ponce-de-Leon-Senti et al., 2020) En este contexto, el uso de bases de datos públicas como NCBI Protein se vuelve fundamental, ya que ofrece una amplia colección de información genómica y proteica, siendo clave en proyectos de esta naturaleza(NCBI, 2022; NCBI Resource Coordinators, 2016).

El uso de técnicas de optimización combinatoria en este ámbito es especialmente relevante, ya que problemas similares en biología molecular suelen ser NP-hard, lo que implica que no pueden resolverse eficientemente con métodos exactos para instancias grandes (Garey & Johnson, 1979). Algoritmos como los genéticos han sido seleccionados en este trabajo debido a su capacidad para manejar grandes espacios de búsqueda, evitar óptimos locales y adaptarse dinámicamente a las características del problema.(Veiga, 2019) La capacidad de las metaheurísticas para abordar problemas complejos ha sido demostrada en diversos campos,

desde la clasificación de datos hasta la optimización de sensores (Pessoa Ferreira Lima, 2013).

La importancia de esta investigación también se enmarca en el contexto global actual, donde la familia Coronaviridae ha adquirido especial relevancia debido a la pandemia de SARS-CoV-2(Pascual-Iglesias et al., 2021). Estudios previos han demostrado que las mutaciones dentro de esta familia pueden tener implicaciones significativas en la infectividad y patogénesis del virus (Enjuanes Sánchez et al., 2011). En este sentido, las herramientas computacionales modernas, aunque avanzadas, todavía enfrentan limitaciones para manejar grandes volúmenes de datos o problemas intratables, lo que hace imprescindible el uso de técnicas como las metaheurísticas para generar soluciones aproximadas en tiempos razonables(Torres-Jiménez & Pavón, 2014).

Este proyecto no solo se limita a implementar un modelo computacional, sino que también establece un flujo de trabajo riguroso que incluye la preparación de datos, el diseño de métricas personalizadas como la Distancia Normalizada Ponderada (DNP) y la validación del modelo propuesto. Los resultados obtenidos no solo contribuyen a clasificar y agrupar proteínas en géneros, sino que también generan nuevas oportunidades para explorar relaciones evolutivas y estructurales en estudios biológicos. En un contexto donde la comprensión de los patrones evolutivos puede contribuir al desarrollo de terapias y vacunas, cada esfuerzo en esta dirección representa una contribución significativa para la biología molecular y la salud pública global.

Objetivos:

Objetivo general:

Modelar e implementar una herramienta de biología computacional utilizando metaheurísticas para la identificación de patrones en secuencias de proteínas.

Objetivos específicos:

• Identificar la metaheurística idónea para la identificación de patrones en la secuencia de proteína.

• Realizar el preprocesamiento de la información para aplicar la metaheurística elegida.

• Plantear el problema de optimización para la identificación de patrones en secuencias de proteínas.

• Implementar el algoritmo Metaheurístico para el problema de optimización.

• Definir una medida para evaluar la calidad de las soluciones encontradas.

Capítulo 1: Marco Teórico

En este capítulo se presentan los fundamentos teóricos que sustentan esta investigación, abarcando conceptos relacionados con optimización, heurísticas, metaheurísticas y aspectos biológicos como aminoácidos y proteínas. El propósito es establecer un marco conceptual que permita contextualizar el desarrollo del trabajo y la aplicación de las técnicas propuestas.

1.1 Optimización

La optimización es un proceso orientado a mejorar o perfeccionar un sistema, procedimiento o solución, y posee un alcance multidisciplinario al ser aplicable tanto en matemáticas como en ciencias computacionales y otros campos (Insua et al., 2004). Este concepto comprende diversas áreas de estudio, desde la búsqueda de puntos máximos y mínimos de funciones hasta la mejora de procesos de decisión o diseño(Granillo Macias, 2019). Dependiendo del enfoque y las características del problema, la optimización puede clasificarse en combinatoria, multiobjetivo, estocástica, o bajo restricciones de desigualdad (Guerra Álvarez & Crawford Labrín, 2010).

En informática, la optimización abarca temas como la mejora de sistemas de bases de datos, la optimización de software y estrategias como el SEO (Search Engine Optimization)(Alancay et al., 2016b). En todos estos casos, la optimización busca equilibrar eficiencia y precisión, adaptándose a los requerimientos específicos de cada disciplina (Peña, 2022).

Los principios fundamentales de la optimización incluyen:

1. Definición clara de la función objetivo: La función que se busca maximizar o minimizar debe estar bien definida y ser representativa del problema a resolver.
2. Identificación de restricciones: Las restricciones representan limitaciones que las soluciones deben cumplir. Estas pueden ser físicas, económicas, o de otro tipo.
3. Espacio de búsqueda: Es el conjunto de todas las soluciones posibles dentro del dominio del problema.

4. Criterios de optimalidad: Son condiciones matemáticas que determinan si una solución es óptima.

Aplicaciones en Problemas Reales

La optimización tiene aplicaciones en diversos campos como:

- Ingeniería: Diseño estructural, optimización de procesos de manufactura, y logística.
- Economía: Asignación de recursos, maximización de beneficios, y minimización de costos.
- Ciencias computacionales: Diseño de algoritmos eficientes, optimización de redes, y análisis de datos.
- Biología computacional: Clasificación de secuencias genéticas, diseño de medicamentos, y análisis de estructuras proteicas.

Por ejemplo, en biología computacional, la optimización es crucial para resolver problemas relacionados con el plegamiento de proteínas, donde se busca una configuración tridimensional que minimice la energía libre, o para clasificar proteínas en función de similitudes estructurales(Barreto Hernández, 2008).

1.2 Optimización Combinatoria

La optimización combinatoria se enfoca en resolver problemas donde el espacio de soluciones posibles es discreto y, generalmente, de gran tamaño (Granillo Macias, 2019). Este campo, estrechamente vinculado con la teoría de la complejidad computacional, se ocupa de diseñar algoritmos capaces de explorar eficientemente espacios de búsqueda vastos para encontrar soluciones óptimas o cercanas al óptimo (Martí, 2001). Los problemas abordados en este ámbito suelen ser NP-hard, es decir, su resolución exacta no es viable en tiempo razonable para instancias grandes.

Los métodos de optimización combinatoria incluyen estrategias constructivas, de mejora, y técnicas basadas en la división de problemas en subcomponentes manejables (Portillo-Lara et al., 2019). Estas metodologías no siempre garantizan la solución óptima, pero permiten obtener soluciones suficientemente buenas para aplicaciones prácticas, destacando su importancia en problemas como el de la mochila, el viajante de comercio y la programación lineal.

Características de Problemas NP-hard

La optimización combinatoria se centra en problemas en los que el conjunto de soluciones factibles es discreto y, generalmente, de gran tamaño(Blum & Roli, 2003). Este tipo de problemas suele estar asociado a la teoría de la complejidad computacional, y muchos de ellos pertenecen a la categoría de problemas NP-hard (Alancay et al., 2016a). Las características de estos problemas incluyen:

1. Espacio de soluciones exponencialmente grande: El número de posibles soluciones crece de manera exponencial con el tamaño del problema, lo que hace inviable explorar todas las opciones de forma exhaustiva.
2. Dificultad computacional: No existe un algoritmo conocido que pueda resolver problemas NP-hard de manera eficiente en todas las instancias.
3. Uso de aproximaciones: Debido a la complejidad, se utilizan métodos heurísticos y metaheurísticos para encontrar soluciones aproximadas en un tiempo razonable.

Ejemplos clásicos de problemas NP-hard incluyen:

- Problema del viajante (TSP): Encontrar la ruta más corta que permita visitar un conjunto de ciudades y regresar al punto de partida.
- Problema de la mochila: Determinar qué elementos incluir en una mochila de capacidad limitada para maximizar el valor total.
- Optimización de redes: Diseñar rutas de comunicación eficientes en redes de gran escala.

En biología computacional, problemas como el alineamiento múltiple de secuencias y la predicción de estructuras tridimensionales de proteínas también son ejemplos de problemas NP-hard (Portillo-Lara et al., 2019).

Ejemplos en Biología Computacional

En biología computacional, la optimización combinatoria se utiliza para:

- Clasificación de secuencias genéticas: Identificar similitudes entre secuencias de ADN o proteínas para inferir relaciones evolutivas.
- Clusterización de proteínas: Agrupar proteínas en familias basadas en similitudes estructurales o funcionales.
- Diseño de medicamentos: Seleccionar combinaciones de moléculas que interactúen de manera efectiva con un objetivo biológico.

Ecuación Típica

Un problema típico de optimización combinatoria se puede formular como:

$$z = min \sum_{i=1}^{n} c_i x_i \quad sujeto\ a: \quad \sum_{j=1}^{m} a_{ij} x_j \leq b_i$$

Donde:

- z: Es el valor óptimo de la función objetivo.
- c_i: Representa el costo asociado a la solución x_i.
- x_i: Son variables binarias ($0\ o\ 1$).
- a_{ij}: Son los coeficientes de las restricciones.
- b_i: Representa los límites de las restricciones.

Este modelo es común en problemas de programación lineal entera, como el problema de la mochila o la selección de variables óptimas en un análisis biológico(Lange, 2013).

Ejemplo Aplicado a Biología Computacional

En el contexto de este proyecto, la optimización combinatoria se utiliza para asignar proteínas a géneros y seleccionar los rankings más representativos. La función objetivo puede incluir minimizar las distancias estructurales dentro de los géneros mientras se maximizan las distancias entre géneros. Esto puede representarse matemáticamente como un problema multiobjetivo(Pelikan et al., 1999):

$$min\left(\frac{1}{|G|}\sum_{i\in G} Distancia_{intra}(i)\right), max\left(\frac{1}{|G|}\sum_{ij\in G} Distancia_{inter}\,(i,j)\right)$$

Donde:

- $Distancia_{intra}(i)$: Representa la distancia promedio dentro de un género.
- $Distancia_{inter}(i,j)$: Representa la distancia promedio entre géneros.
- G: Conjunto de géneros.

Este enfoque combina técnicas de optimización con métricas estructurales específicas, como el RMSD y el Alignment Score, para lograr una clasificación precisa.

1.3 Heurísticas

El concepto de heurística se refiere a estrategias prácticas para la resolución de problemas que, aunque no aseguran una solución óptima, permiten obtener resultados aceptables en un tiempo razonable (Alancay et al., 2016a). Estas técnicas, inspiradas en la creatividad y el pensamiento lateral, se utilizan cuando los problemas presentan alta complejidad o información incompleta(Blank & Deb, 2020).

En matemáticas, las heurísticas han sido empleadas históricamente para identificar patrones y construir soluciones aproximadas. En psicología, se aplican para modelar procesos de toma de decisiones, mientras que en ingeniería se utilizan para simplificar diseños y planificar sistemas de operación (Kahneman, 2003; Belloví et al., 2004). Aunque son métodos basados

en aproximaciones, su eficiencia y adaptabilidad los convierten en herramientas valiosas para abordar problemas complejos(Wang & Tong, 2009).

1.4 Metaheurísticas

Las metaheurísticas representan un nivel superior de estrategias heurísticas, diseñadas para resolver problemas de optimización general mediante procedimientos genéricos y abstractos (Vélez-Gallego & Montoya-Echeverri, 2007). Estas técnicas, como los algoritmos genéticos, el recocido simulado y la optimización por enjambre de partículas, exploran el espacio de búsqueda de manera iterativa y adaptativa, utilizando mecanismos como la combinación y mutación de soluciones (Alancay et al., 2016b).

figura 1.Problema Heurístico

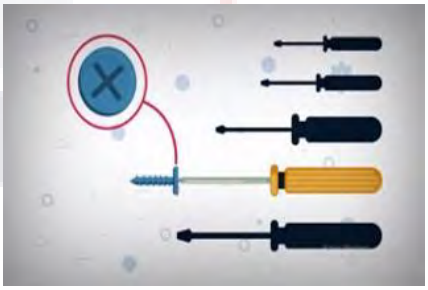


figura 2. Solución Heurística

figura 3.Solución Metaheurística

Su principal ventaja radica en su capacidad para evitar caer en óptimos locales y su aplicabilidad a una amplia gama de problemas combinatorios. Las metaheurísticas no garantizan la solución óptima, pero su diseño eficiente permite alcanzar soluciones de calidad en tiempos computacionales razonables (Navas & Urbaneja, 2013).

1.4.1Clasificación de metaheurísticas

Metaheurísticas Basadas en una Solución Única:

Estas técnicas se centran en la mejora iterativa de una única solución en el espacio de búsqueda, haciendo ajustes incrementales hasta encontrar una solución satisfactoria.(Peralta-Abarca, 2018) Ejemplos destacados incluyen:

Métodos de Recocido Simulado (Simulated Annealing - SA):

- Método de Noising (SA-NM)
- Método de Aceptación por Umbral (SA-TA)
- Recocido Microcanónico (SA-MA)

Búsqueda en Vecindario Grande (Large Neighborhood Search - LNS):

- Búsqueda Tabú (Tabu Search - TS)
- Búsqueda Local Iterada (Iterated Local Search - ILS)
- Búsqueda Local Guiada (Guided Local Search - GLS)
- Búsqueda en Vecindario Variable (Variable Neighborhood Search - VNS)

Otros Métodos de Búsqueda Adaptativa:

- Procedimiento Adaptativo de Búsqueda de Vecindario Grande (Adaptive Large Neighborhood Search - ALNS)
- Búsqueda en Vecindario de Muy Gran Escala (Very Large Scale Neighborhood Search - VLSN)

GRASP (Greedy Randomized Adaptive Search Procedure):

- Procedimientos de búsqueda combinados que utilizan aleatorización y optimización local.
- Método de Electromagnetismo (Electromagnetism-Like Algorithm - EMA)

Ventajas de estos métodos:

- Son relativamente simples de implementar y comprender.
- Funcionan bien en problemas donde las soluciones factibles son rápidas de evaluar.

Desventajas:

- Alta probabilidad de caer en óptimos locales si no se combinan con estrategias adicionales.
- No son tan efectivas en espacios de búsqueda muy amplios.

Metaheurísticas Basadas en Poblaciones:

Estas estrategias trabajan con un conjunto de soluciones que evolucionan a lo largo de iteraciones para explorar de manera más amplia el espacio de búsqueda.(Navas & Urbaneja, 2013) Ejemplos incluyen:

Computación Evolutiva (Evolutionary Computation - EC):

- Algoritmos Genéticos (Genetic Algorithm - GA)
- Programación Genética (Genetic Programming - GP)
- Programación Evolutiva (Evolutionary Programming - EP)

- Estrategia Evolutiva (Evolution Strategy - ES)
- Algoritmos de Estimación de Distribución (Estimation of Distribution Algorithms - EDAs)

Basadas en Inteligencia de Enjambre (Swarm Intelligence - SI):

- Optimización por Colonia de Hormigas (Ant Colony Optimization - ACO)
- Optimización por Enjambre de Partículas (Particle Swarm Optimization - PSO)
- Optimización basada en Biogeografía (Biogeography-Based Optimization - BBO)
- Algoritmos de Inmunidad Artificial (Artificial Immune Systems - AIS)
- Optimización Basada en Colonias de Abejas (Bee Colony Optimization - BCO)
- Optimización por Forrajeo de Bacterias (Bacterial Foraging Optimization Algorithm - BFOA)

Otros Métodos Basados en Poblaciones:

- Algoritmo de Evolución Diferencial (Differential Evolution - DE)
- Algoritmos Coevolutivos (Co-Evolutionary Algorithms - CoEA)
- Algoritmos Culturales (Cultural Algorithms - CA)
- Búsqueda por Dispersión (Scatter Search - SS)
- Enlace de Caminos (Path Relinking - PR)
- Algoritmo de Memética (Memetic Algorithm - MA)
- Algoritmo de Electromagnetismo (EMA)
- Shuffled Frog Leaping Algorithm (SFLA)
- Intelligent Water Drops Algorithm (IWD)
- Búsqueda de Cuco (Cuckoo Search - CS)
- Algoritmo de Luciérnaga (Firefly Algorithm - FA)

Ventajas de estos métodos:

- Exploran el espacio de búsqueda de manera más exhaustiva al trabajar con múltiples soluciones simultáneamente.
- Son capaces de escapar de óptimos locales gracias a la diversidad poblacional.

Desventajas:

- Pueden ser computacionalmente costosos debido al manejo de múltiples soluciones.
- Requieren ajustes cuidadosos de parámetros para garantizar la convergencia.

1.4.2 Aplicación de metaheurísticas en biología computacional

En el ámbito de la biología computacional, las metaheurísticas han demostrado ser herramientas poderosas para resolver problemas complejos como la predicción estructural de proteínas, la clasificación de secuencias y el análisis de redes genómicas. Este proyecto utiliza algoritmos genéticos para clasificar proteínas Spike de la familia Coronaviridae, maximizando la similitud dentro de los géneros y minimizando las distancias entre géneros(Elshaer & Awad, 2020).

1.4.3 Ventajas y desventajas de las metaheurísticas

Ventajas:

- Capacidad para manejar grandes espacios de búsqueda.
- Escapan de óptimos locales, lo que incrementa la calidad de las soluciones.
- Flexibilidad para adaptarse a diferentes problemas.

Desventajas:

- Requieren un ajuste cuidadoso de parámetros, como el tamaño de la población y la probabilidad de mutación.
- Tiempos computacionales elevados en comparación con las heurísticas.

Metaheurística	Tipo	Enfoque	Ventajas	Desventajas
Recocido Simulado (SA)	Single-Solution Based	Explora soluciones gradualmente	Simplicidad, evita óptimos locales	Lento para problemas grandes
Búsqueda Tabú (TS)	Single-Solution Based	Almacena soluciones exploradas	Evita ciclos, explora más áreas del espacio	Complejidad de implementación
Algoritmos Genéticos (GA)	Population-Based	Evolutivo basado en selección natural	Maneja espacios grandes, encuentra soluciones robustas	Requiere parametrización cuidadosa
Programación Evolutiva (EP)	Population-Based	Basado en mutaciones	Flexible y general	Sensible al tamaño de población
Optimización por Enjambre de Partículas (PSO)	Swarm Intelligence	Basado en comportamiento colectivo	Convergencia rápida, fácil implementación	Puede atascarse en óptimos locales
Colonia de Hormigas (ACO)	Swarm Intelligence	Inspirado en comportamiento de hormigas	Buena exploración del espacio de búsqueda	Sensible a parámetros, costoso computacionalmente
Memetic Algorithm (MA)	Population-Based	Híbrido (evolución + búsqueda local)	Combina ventajas de GA y búsquedas locales	Complejidad de implementación
Firefly Algorithm (FA)	Swarm Intelligence	Basado en comportamiento de luciérnagas	Maneja problemas no lineales, flexible	Puede necesitar ajuste de parámetros
Optimización Diferencial (DE)	Population-Based	Evolutivo con operadores específicos	Fácil de implementar, buena convergencia	Requiere muchos cálculos
Optimización Basada en Colonias de Abejas (BCO)	Swarm Intelligence	Basado en búsqueda de alimentos	Eficiente en problemas multiobjetivo	Complejo en configuraciones grandes

Tabla 1.Comparación entre algunas metaheurísticas

Ecuación

La función de fitness en un algoritmo genético evalúa la calidad de las soluciones:

$$F(x) = w_1 \times intra(x) - w_2 \times inter(x)$$

Donde:

- $F(x)$: Representa la calidad de la solución.
- $intra(x)$: Distancia promedio dentro de un grupo.

- $inter(x)$: Distancia promedio entre grupos.
- w_1, w_2: Pesos ajustables.

1.5 Aminoácidos y Proteínas

En el contexto biológico, las proteínas son moléculas esenciales para la estructura y función de los organismos. Están compuestas por aminoácidos, que forman largas cadenas denominadas polímeros (Gómez-Moreno Calerra, 2000). Existen 20 aminoácidos estándar que, al combinarse, determinan las propiedades y funciones de cada proteína. Estas macromoléculas están involucradas en procesos biológicos críticos, como el transporte de moléculas, la catálisis enzimática y la defensa inmunológica(Cech, 2019).

El estudio de las proteínas, conocido como proteómica, se enfoca en comprender su estructura, funciones y relaciones con otros elementos biológicos(Gómez-Moreno Calerra, 2000). Este campo es esencial para identificar patrones evolutivos y diseñar estrategias para abordar problemas biomédicos y biotecnológicos (Roldán Martínez, 2015).

1.6 Estudio de las Proteínas

La investigación sobre proteínas enfrenta desafíos significativos debido a la complejidad de su estructura tridimensional y su constante movimiento(Nieto-Torres et al., 2014). Técnicas como la cristalografía de rayos X y la microscopía electrónica han permitido avances importantes, pero aún existen limitaciones para obtener modelos precisos y completos (Adiyaman & McGuffin, 2019; Feig, 2017).

figura 4.Estructuras de Proteínas

La modelización computacional ha surgido como una herramienta complementaria, destacándose algoritmos como AlphaFold, que predicen estructuras tridimensionales con alta precisión(Galvis Motoa, 2022). Sin embargo, problemas como el plegamiento de proteínas (folding) y la interacción entre diferentes proteínas requieren enfoques más avanzados, como el uso de metaheurísticas, para abordar su complejidad de manera efectiva(Ponce de León Sentí et al., 2022).

1.7 Conclusión del Capítulo

El marco teórico presentado en este capítulo establece los fundamentos conceptuales necesarios para comprender y desarrollar las estrategias propuestas en esta investigación. La optimización, las heurísticas y las metaheurísticas se destacan como herramientas clave para abordar problemas complejos, mientras que los conceptos biológicos proporcionan el contexto necesario para aplicar estas técnicas al análisis y clasificación de proteínas. Este marco interdisciplinario asegura una base sólida para el desarrollo del modelo propuesto en los capítulos siguientes.

Capítulo 2: Preparación de los Datos

En este capítulo se describe el proceso de obtención, organización y preparación de los datos utilizados en el estudio. El objetivo principal es garantizar la calidad, consistencia y adecuación de los datos para su posterior análisis, implementación del algoritmo genético y clasificación de las proteínas seleccionadas. Este trabajo se fundamenta en antecedentes relevantes que incluyen metodologías avanzadas para la construcción de árboles filogenéticos, la clusterización de proteínas y la búsqueda de patrones evolutivos utilizando técnicas computacionales.

2.1 Origen de los Datos

Para este estudio, se utilizaron secuencias de proteínas obtenidas de la base de datos pública NCBI Protein, reconocida por su vasta colección de información biológica. Se seleccionaron 46 proteínas Spike de la familia Coronaviridae por su importancia en procesos esenciales como el reconocimiento del huésped y la infectividad viral. La selección estratégica de estas proteínas buscó garantizar una diversidad significativa en sus secuencias, facilitando así la identificación de patrones evolutivos.

Las estructuras tridimensionales de estas proteínas se generaron utilizando el modelo de inteligencia artificial AlphaFold, que predijo cinco conformaciones diferentes para cada proteína. Estas conformaciones, ordenadas por calidad y etiquetadas como ranked_0 a ranked_4, proporcionaron información estructural detallada. Posteriormente, los modelos se organizaron en carpetas individuales por proteína, almacenados en formato .pdb dentro de un directorio raíz uniformemente estructurada. Este esquema organizativo no solo garantizó la integridad y accesibilidad de los datos, sino que también facilitó su procesamiento en las etapas posteriores de análisis.

figura 5.Origen de los Datos

2.2 Comparación de Estructuras

La comparación de las estructuras tridimensionales de las proteínas Spike se llevó a cabo utilizando ChimeraX y su comando MatchMaker, el cual alinea cadenas principales y analiza propiedades atómicas para determinar similitudes estructurales. Este enfoque se basó en tres métricas clave: la raíz cuadrática media de las desviaciones (RMSD), el puntaje de alineación (Alignment Score) y el número de átomos pareados. Estas métricas han sido previamente utilizadas en investigaciones como la obtención de árboles filogenéticos mediante distancias definidas en los mejores aciertos bidireccionales de organismos (Ponce de León Sentí et al., 2017).

Para evitar redundancias y optimizar el proceso, se automatizó la generación de scripts en formato .cxc diseñados para comparar cada modelo con los modelos restantes. Los resultados de estas comparaciones se registraron en logs detallados, que consolidaron las métricas necesarias para el análisis. Este procedimiento se inspira en estudios previos sobre clusterización de proteínas (Galvis Motoa et al., 2021), permitiendo obtener una base sólida para la clasificación en géneros.

2.3 Métricas Obtenidas

El análisis estructural generó métricas cuantitativas fundamentales para evaluar las similitudes entre los modelos. Estas métricas incluyen:

- RMSD: Mide la distancia promedio entre los átomos equivalentes de los modelos alineados, siendo un indicador directo de similitud estructural, con valores bajos que reflejan mayor concordancia.
- Alignment Score: Calculado utilizando matrices de similitud como BLOSUM-62, evalúa la precisión del ajuste entre las cadenas principales. Valores altos indican una correspondencia significativa.
- Número de Átomos Pareados: Representa la cantidad de átomos correctamente alineados, proporcionando información sobre las regiones estructurales compartidas.

Estas métricas fueron esenciales para sintetizar la similitud estructural en un único valor representativo, conocido como Distancia Normalizada Ponderada (DNP), consolidando así una base cuantitativa para el análisis de patrones evolutivos. Este enfoque se relaciona con metodologías previas de obtención de grupos de proteínas altamente semejantes (Gallegos, 2019; Ponce de León Sentí et al., 2021).

2.4 Procesamiento de los Logs

La información extraída de los logs generados por ChimeraX fue procesada de manera sistemática utilizando Python. Los datos clave, como RMSD, Alignment Score y átomos pareados, se organizaron en un DataFrame, facilitando el acceso y análisis de las métricas consolidadas. Este procesamiento incluyó:

- Eliminación de duplicados.
- Validación de la consistencia de los datos.
- Estructuración en columnas identificables para cada modelo y proteína.

El resultado fue un conjunto de datos accesible y reproducible que sirvió como base para etapas posteriores, como la normalización de métricas y el cálculo del DNP. Este enfoque está alineado con investigaciones previas sobre la automatización de procesos en la clusterización de proteínas (Galvis Motoa et al., 2021).

2.5 Extracción y Normalización de Atributos

Los atributos clave de similitud estructural se extrajeron del DataFrame consolidado y se normalizaron para su análisis posterior. Este proceso incluyó:

- RMSD, Alignment Score y Número de Átomos Pareados transformados mediante una técnica basada en la normalización min-max.
- Eliminación de sesgos para garantizar la comparabilidad entre los atributos.

La normalización facilitó el cálculo del DNP, que sintetizó las métricas normalizadas en un único valor representativo. Este atributo, combinado con técnicas computacionales avanzadas, optimizó la preparación de los datos para la clasificación en géneros, apoyando enfoques similares empleados en la búsqueda de patrones evolutivos mediante metaheurísticas multiobjetivo (Correa Morales et al., 2023).

2.6 Cálculo de DNP

El DNP se diseñó para representar de manera integral la similitud estructural entre los modelos. Utilizando una fórmula lineal ponderada, el DNP combina las métricas normalizadas:

$$DPN = w_1 \times AS_N + w_2 \times RMSD_N + w_3 \times AP_N$$

Donde los pesos asignados ($w_1 = 0.470552, w_2 = 0.282580, w_3 = 0.246868$) reflejan la relevancia de cada métrica en el análisis. Este atributo único se integró como una columna adicional en el DataFrame, sirviendo como la métrica principal para las evaluaciones realizadas por el algoritmo genético.

2.7 Adaptación de los Datos para el Algoritmo Genético

La integración de los datos al modelo genético requirió:

1. Una matriz de distancia basada en los valores de DNP, que permitió calcular eficientemente distancias internas y externas durante las evaluaciones de fitness.
2. El uso del DataFrame consolidado para acceder a información adicional, como el nombre, ranking y género de cada modelo.

Este diseño combinado optimizó tanto la precisión como el rendimiento del modelo, integrando las métricas obtenidas y adaptándolas para su uso en algoritmos evolutivos. Este enfoque está alineado con antecedentes donde se analizaron patrones en imágenes de proteínas y motivos conservados mediante metaheurísticas (Macías Soto et al., 2023; Correa Morales et al., 2023).

2.8 Conclusión del Capítulo

El proceso de preparación de datos se desarrolló de manera sistemática, estableciendo una base sólida para el análisis computacional de patrones evolutivos en proteínas. Desde la obtención de secuencias hasta la integración en el modelo genético, cada etapa se diseñó para garantizar calidad, consistencia y reproducibilidad, apoyándose en técnicas previamente validadas en la literatura científica.

Capítulo 3: Modelo Propuesto

3.1 Metodología

La metodología desarrollada en este proyecto integra técnicas de biología computacional, análisis estructural y optimización mediante metaheurísticas, específicamente algoritmos genéticos (AG), para identificar patrones en secuencias tridimensionales de proteínas. La metodología se centra en automatizar el flujo de trabajo a través de una herramienta implementada en Jupyter Notebook, organizada en 10 pasos sistemáticos que abarcan desde la preparación de datos hasta la optimización y análisis de resultados.

Objetivo de la Metodología

El objetivo principal es proporcionar una herramienta computacional que permita clasificar proteínas en géneros, optimizando la identificación de patrones evolutivos y estructurales. Esto se logra mediante la consolidación de datos estructurales y el uso de metaheurísticas que exploran eficientemente un espacio de búsqueda complejo.

Fases de la Metodología

1. Preparación de Datos

- Cargar y listar archivos: Se identifican y organizan las estructuras de proteínas disponibles en formato .pdb.
- Generar scripts de carga para ChimeraX: Se crean scripts .cxc para cargar automáticamente las estructuras en el software ChimeraX.
- Organización de subdirectorios: Las proteínas y sus estructuras se almacenan en subcarpetas uniformes para facilitar su manejo.

2. Comparación de Estructuras

- Emparejamientos sin repetición: Se generan scripts para realizar comparaciones entre proteínas sin redundancias.
- Ejecución en ChimeraX: Los scripts son procesados en grupos para calcular métricas clave como RMSD, Alignment Score, y Átomos Pareados.
- Consolidación de logs: Los resultados de las comparaciones se almacenan en un único archivo de log.

3. Construcción de un DataFrame

- Se extrae información de los logs consolidados y se organiza en un DataFrame que incluye atributos clave:

- Identificadores y nombres de las proteínas.
- Géneros asociados a las proteínas, obtenidos de un archivo adicional.
- Métricas estructurales de las comparaciones realizadas.

4. Clasificación Supervisada

- Predecir el género: Se utiliza un modelo supervisado (e.g., Random Forest) para evaluar la influencia de los atributos estructurales en la clasificación de proteínas.
- Importancia de características: Se determina la relevancia de las métricas utilizadas (Alignment Score, RMSD, Átomos Pareados) para la predicción del género.

5. Normalización y Cálculo de la Métrica DNP

- Normalización de atributos: Los valores de las métricas se escalan mediante Min-Max Scaling.
- Cálculo de DNP: Se integra una métrica única, Distancia Normalizada Ponderada (DNP), combinando las métricas normalizadas mediante pesos derivados de su importancia relativa.

6. Optimización mediante Algoritmo Genético

- Definición del problema: Se plantea como una tarea de optimización combinatoria para clasificar proteínas en cuatro géneros y seleccionar el ranking más representativo.
- Elementos del AG:
  - Codificación: Cada individuo representa una asignación de proteínas a géneros.
  - Función de fitness: Maximiza la coherencia estructural dentro de géneros y las diferencias entre ellos.
  - Operadores genéticos: Selección por torneo, cruce uniforme, y mutación para explorar el espacio de búsqueda.

Herramienta Computacional

La metodología se implementó en un Jupyter Notebook, que integra código en Python y herramientas externas como ChimeraX. Este notebook automatiza cada paso, asegurando reproducibilidad y eficiencia.

Beneficios de la Herramienta

- Automatización: Minimiza la intervención manual, reduciendo errores.

- Flexibilidad: Puede adaptarse a diferentes conjuntos de datos y problemas relacionados.
- Integración: Combina múltiples etapas del análisis en una sola herramienta.

Resultados Esperados

- Clasificación precisa de las proteínas en géneros.
- Reducción de tiempos computacionales en comparación con métodos manuales.
- Generación de una métrica integradora (DNP) que facilita el análisis estructural y evolutivo.

Esta metodología no solo cumple con los objetivos planteados, sino que también representa un aporte significativo al campo de la biología computacional. La herramienta desarrollada no solo optimiza el análisis de proteínas, sino que establece un marco para futuros estudios evolutivos y estructurales utilizando metaheurísticas.

3.2 Formulación del Problema de Optimizacion

El modelo propuesto tiene como objetivo clasificar 46 proteínas Spike de la familia Coronaviridae en cuatro géneros (genero1, genero2, genero3, genero4) y determinar el ranking más representativo entre las cinco predicciones generadas por AlphaFold (ranked_0 a ranked_4). Esta clasificación busca optimizar dos aspectos fundamentales: minimizar las distancias estructurales dentro de cada género y maximizar las distancias entre géneros, garantizando una agrupación coherente que refleje las similitudes y diferencias evolutivas. El problema se plantea como una tarea de optimización combinatoria en un espacio de búsqueda complejo, donde cada solución representa una posible asignación de rankings y géneros para las proteínas. La evaluación de estas soluciones se basa en el atributo único ponderado DNP, que sintetiza métricas estructurales clave, como el RMSD, el Alignment Score y los Átomos Pareados.

3.3 Fundamentos del Algoritmo Genético

El algoritmo genético utilizado en este modelo se basa en los principios de evolución biológica, empleando una población inicial de soluciones que evoluciona a través de generaciones sucesivas. Cada individuo en la población representa una posible solución al problema, codificada como un vector en el que cada posición corresponde a una proteína e incluye el ranking seleccionado y el género asignado. Para evaluar la calidad de las soluciones, se utiliza una función de fitness que combina dos objetivos: minimizar la distancia promedio dentro de cada género y maximizar la distancia promedio entre géneros. La función de fitness está ponderada por parámetros ajustables (α y β) que permiten controlar la importancia relativa de estos objetivos.

Los operadores genéticos son esenciales en el proceso evolutivo. La selección por torneo asegura que los individuos de mejor calidad tengan mayores probabilidades de ser padres, mientras que el cruce uniforme permite combinar características de dos individuos para generar soluciones potencialmente superiores. La mutación introduce variaciones aleatorias en las soluciones, evitando que el modelo quede atrapado en óptimos locales y promoviendo la diversidad en la población.

3.4 Configuración y Parámetros del Algoritmo

El modelo fue configurado con parámetros ajustados empíricamente para equilibrar la exploración y la explotación del espacio de búsqueda. Se utilizó una población de 100 individuos y se definió un máximo de 200 generaciones para permitir una evolución significativa. La probabilidad de cruce se fijó en 0.8, asegurando que la mayoría de las soluciones se generaran mediante la combinación de características de dos padres, mientras que la probabilidad de mutación se estableció en 0.1 para mantener una tasa de variación controlada. Los pesos asignados a los componentes de la función de fitness ($\alpha = 1.0\ y\ \beta = 1.0$) reflejaron la importancia equivalente de minimizar las distancias dentro de géneros y maximizar las distancias entre géneros.

3.5 Evaluación del Modelo

La evaluación del modelo se centró en analizar su capacidad para generar clasificaciones coherentes y óptimas. Las soluciones fueron valoradas según tres criterios principales: la reducción de distancias estructurales dentro de los géneros, el aumento de las distancias entre géneros y la diversidad en las soluciones generadas durante el proceso evolutivo. Además, se monitoreó la evolución del fitness a lo largo de las generaciones para evaluar la eficacia del modelo en alcanzar el óptimo global. Los resultados confirmaron que el modelo fue capaz de clasificar de manera efectiva las proteínas, asignando rankings y géneros que reflejaron patrones estructurales y evolutivos consistentes.

3.6 Resultados y Visualización

El algoritmo genético produjo una solución óptima que clasifica las 46 proteínas en cuatro géneros y selecciona el ranking más representativo para cada una. Los resultados incluyeron un vector de clasificación que asocia cada proteína con un género y un ranking, así como gráficos que ilustraron la reducción progresiva de las distancias dentro de los géneros y el aumento de las distancias entre géneros. Además, se registró el historial evolutivo de la población, mostrando el progreso del fitness en cada generación. Estos resultados fueron exportados a un archivo CSV para facilitar su análisis y presentación.

3.7 Conclusión del Capítulo

El modelo propuesto, basado en algoritmos genéticos, demostró ser eficaz para resolver el problema de clasificación de proteínas en géneros y selección de rankings óptimos. Su capacidad para integrar datos estructurales, métricas normalizadas y principios evolutivos permitió obtener resultados precisos, reproducibles y consistentes con los objetivos planteados. Este enfoque no solo destaca por su robustez y flexibilidad, sino que también sienta las bases para futuras aplicaciones en el análisis evolutivo y estructural de proteínas en estudios de biología computacional.

Capítulo 4: Implementación del Modelo

Este capítulo presenta el desarrollo e implementación de una metodología novedosa para la identificación de patrones en secuencias tridimensionales de proteínas pertenecientes a la familia Coronaviridae, utilizando una herramienta computacional creada en Jupyter Notebook. Este enfoque integra técnicas avanzadas de procesamiento de datos estructurales, normalización de atributos, y optimización mediante metaheurísticas, específicamente algoritmos genéticos (AG), consolidando los 10 pasos que conforman la metodología planteada en este proyecto.

El principal aporte de este capítulo radica en la creación de una herramienta flexible y reproducible que automatiza cada etapa del proceso, desde la recolección de datos hasta la clasificación final de proteínas en géneros. Además, este capítulo cumple con el objetivo general del proyecto: modelar e implementar una herramienta de biología computacional utilizando metaheurísticas para la identificación de patrones en secuencias de proteínas, brindando una solución robusta, eficiente y adaptable para el análisis estructural de proteínas en biología computacional.

4.1 Paso 1: Cargar y listar todos los archivos .pdb disponibles

El primer paso en el proceso de automatización consiste en identificar y listar todas las estructuras de proteínas en formato .pdb almacenadas en los subdirectorios de la carpeta principal. Este paso es esencial, ya que garantiza que todas las estructuras relevantes estén disponibles y organizadas para los análisis posteriores, como la comparación estructural y la clasificación mediante algoritmos genéticos.

Objetivos del paso

1. Identificar la ruta raíz donde se almacenan las estructuras de proteínas.
2. Explorar todos los subdirectorios de la carpeta principal para localizar los archivos .pdb.
3. Generar una lista consolidada con las rutas completas de cada archivo encontrado.

4. Mostrar los resultados al usuario para garantizar que el proceso se ha ejecutado correctamente.

Detalles del proceso

1. Definir la ruta raíz: El script comienza especificando la ubicación de la carpeta que contiene las estructuras de proteínas. Esto se realiza mediante la variable root_dir, que apunta a la ruta principal donde se espera encontrar subdirectorios con archivos .pdb.
2. Explorar subcarpetas: Una vez definida la ruta raíz, el código verifica si la carpeta existe. Si no existe, se imprime un mensaje de error que ayuda al usuario a identificar el problema. Si la carpeta existe, el script itera por cada subdirectorio.
3. Buscar archivos .pdb: Dentro de cada subcarpeta, se busca específicamente archivos con la extensión .pdb. Esto asegura que únicamente se consideren los archivos relevantes para el análisis.
4. Generar una lista completa: Los archivos encontrados se añaden a una lista denominada pdb_files. Esta lista consolida las rutas completas de todos los archivos .pdb encontrados en las subcarpetas.
5. Mostrar el resultado: Si se encuentran archivos .pdb, el script imprime una lista detallada con las rutas completas de cada archivo. Si no se encuentran, se muestra un mensaje de error para informar al usuario.

Ejemplo de salida:

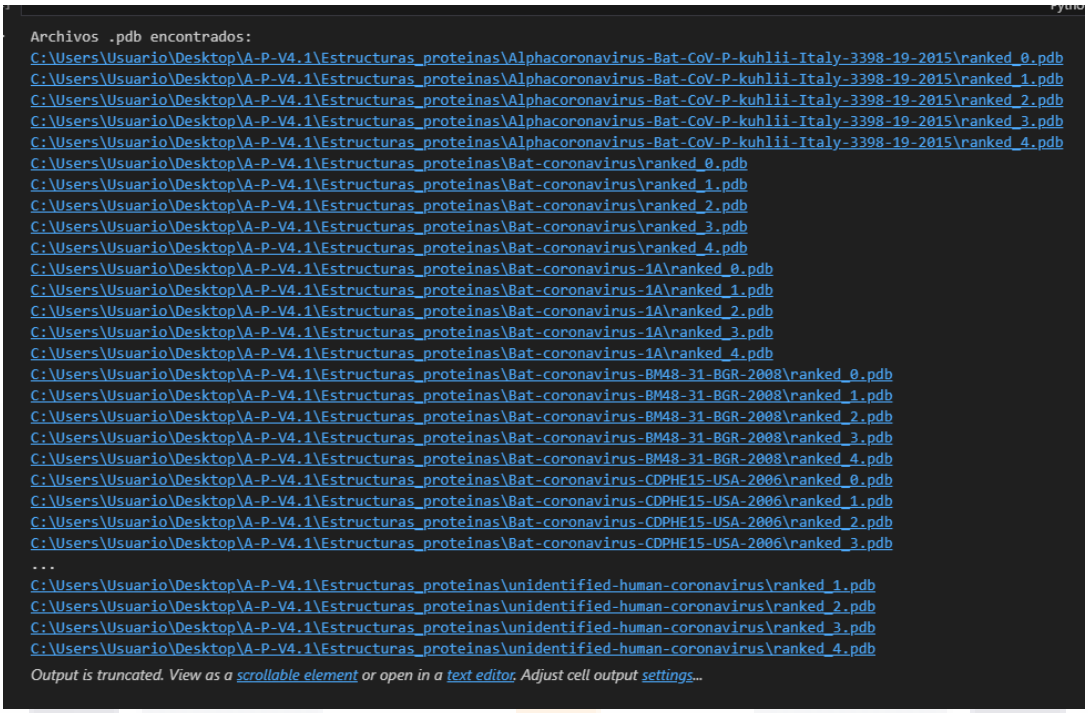


figura 6 Archivos .pdb encontrados

Importancia del paso:

Este paso garantiza que el conjunto de datos esté correctamente cargado y estructurado, permitiendo a los usuarios identificar cualquier problema en la organización de los datos desde el inicio. Además, asegura que todos los archivos necesarios estén disponibles para su análisis posterior, evitando interrupciones en las etapas siguientes del proceso.

Código del paso:

El código completo de este paso se encuentra en los anexos del documento.

4.2 Paso 2: Generar un script .cxc para cargar todos los archivos en ChimeraX

En este paso, se utiliza la lista de rutas de los archivos .pdb generada en el Paso 1 para crear un script .cxc que contiene comandos necesarios para cargar todas las estructuras de proteínas en ChimeraX. Este archivo facilita la automatización del proceso de apertura y análisis de las estructuras en ChimeraX, eliminando la necesidad de cargarlas manualmente. El script

generado puede ser ejecutado directamente en la interfaz gráfica o en el modo de línea de comandos de ChimeraX.

Objetivos del paso:

1. Utilizar la lista pdb_files generada previamente para generar comandos que abran cada archivo .pdb en ChimeraX.
2. Crear un archivo llamado load_proteins.cxc que contenga todos los comandos necesarios.
3. Guardar el archivo en la carpeta principal del proyecto y mostrar su contenido al usuario para validación.

Detalles del proceso:

1. Verificación de la lista pdb_files: Antes de proceder, se verifica que la lista pdb_files, generada en el Paso 1, esté disponible. Si no se encuentra, el script emite un mensaje de error que alerta al usuario de la necesidad de ejecutar el Paso 1 primero.
2. Construcción del contenido del script: Se itera sobre cada archivo en la lista pdb_files, generando un comando del tipo open ruta_del_archivo. Estos comandos se concatenan en una cadena de texto llamada script_content.
3. Creación del archivo load_proteins.cxc: Una vez generado el contenido del script, se crea un archivo llamado load_proteins.cxc en la carpeta principal del proyecto. Este archivo contiene todos los comandos necesarios para abrir las estructuras de proteínas en ChimeraX.
4. Visualización y validación: El contenido del archivo generado se imprime en la consola para que el usuario pueda verificar su precisión antes de ser usado en ChimeraX.
5. Notificación al usuario: El script notifica al usuario la ubicación del archivo generado, asegurando que pueda localizarlo y ejecutarlo sin inconvenientes.

Ejemplo de salida:

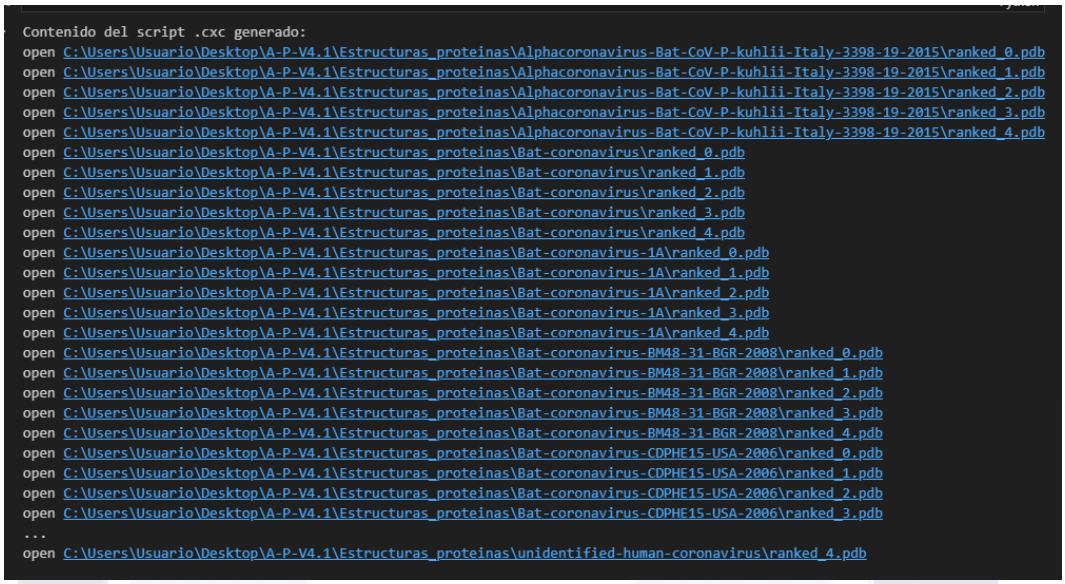


figura 7 Contenido del script .cxc generado

Importancia del paso:

Este paso es crucial para garantizar la automatización del proceso de carga de archivos .pdb en ChimeraX. Al generar un script que puede ser ejecutado directamente, se ahorra tiempo y se evita la posibilidad de errores manuales en la apertura de los archivos. Esto es especialmente útil cuando se trabaja con un gran número de estructuras de proteínas.

Notas adicionales:

- El archivo load_proteins.cxc puede ser ejecutado directamente desde ChimeraX utilizando el comando source load_proteins.cxc.
- Si el usuario necesita modificar o revisar el script, puede abrirlo con cualquier editor de texto.

Código del paso:

El código completo de este paso se encuentra en los anexos del documento.

4.3 Paso 3: Cargar ChimeraX y abrir los scripts generados

En este paso, se utiliza el software ChimeraX para cargar las estructuras de proteínas listadas en el script load_proteins.cxc, generado en el Paso 2. Este paso tiene como único objetivo abrir el programa ChimeraX y cargar los datos de forma automatizada, asegurando que las estructuras de proteínas estén listas para ser procesadas en los pasos posteriores.

Objetivos del paso:

1. Abrir ChimeraX: Iniciar el software ChimeraX, necesario para la visualización y análisis de estructuras tridimensionales de proteínas.
2. Cargar los datos generados: Utilizar el archivo load_proteins.cxc para abrir automáticamente las estructuras de proteínas en ChimeraX.
3. Dejar las estructuras cargadas: Verificar que todas las estructuras listadas en el archivo .cxc estén correctamente cargadas y listas para análisis.

Detalles del proceso:

1. Ubicación del archivo generado: El archivo load_proteins.cxc, creado en el Paso 2, debe estar almacenado en el directorio de salida especificado (por ejemplo, output dir). Este archivo contiene comandos para cargar las proteínas en ChimeraX.
2. Abrir ChimeraX: Se abre el software ChimeraX desde la interfaz gráfica o mediante línea de comandos.
3. Cargar el script en ChimeraX:
    - En la interfaz gráfica de ChimeraX, el usuario puede cargar el archivo .cxc siguiendo estos pasos:
        - Abrir el menú principal de ChimeraX.
        - Seleccionar la opción File > Open.
        - Navegar hasta la ubicación del archivo load_proteins.cxc y seleccionarlo.
    - Alternativamente, si ChimeraX se ejecuta desde la línea de comandos, se puede cargar el archivo utilizando el comando:
    - source path/to/load_proteins.cxc

Reemplazando path/to con la ruta correspondiente al archivo.

4. Verificación de carga: Una vez cargado el archivo, ChimeraX abrirá automáticamente todas las estructuras listadas en el script. El usuario puede verificar que todas las estructuras han sido cargadas correctamente observando el panel lateral de ChimeraX, donde se enumerarán las proteínas abiertas.

Importancia del paso:

Este paso es crucial para garantizar que todas las estructuras de proteínas estén disponibles en el entorno de trabajo de ChimeraX antes de proceder con los análisis. Aunque no se realiza ningún procesamiento en este punto, cargar las estructuras correctamente asegura que los pasos posteriores se puedan ejecutar de manera fluida y sin interrupciones.

Recomendaciones

- Asegúrate de que ChimeraX esté instalado y configurado correctamente antes de realizar este paso.
- Si el archivo load_proteins.cxc no se carga o contiene errores, revisa su contenido y la lista pdb_files generada en el Paso 1.
- Mantén una copia de seguridad del archivo load_proteins.cxc en caso de que necesites recargar los datos.

Resultado esperado

Al finalizar este paso:

- ChimeraX estará abierto y mostrará todas las estructuras de proteínas listadas en el archivo load_proteins.cxc.
- El entorno de trabajo estará listo para proceder con los análisis detallados y la ejecución de comparaciones en los siguientes pasos.

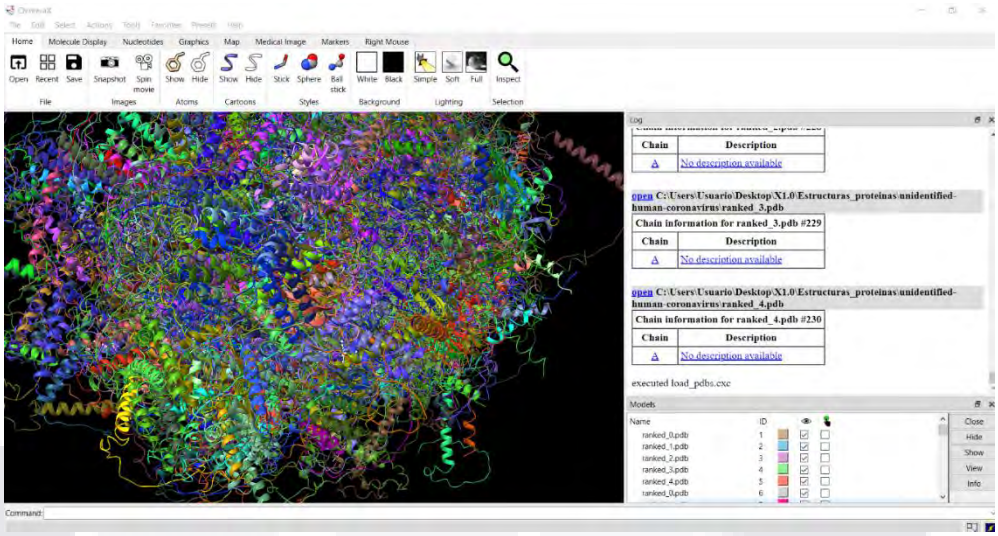


figura 8 Todas las estructuras de proteínas cargadas en ChimeraX

Notas adicionales

Este paso no incluye cálculos ni modificaciones en los datos. Su propósito es puramente preparatorio, asegurando que los datos estén cargados y organizados en ChimeraX para el análisis estructural.

4.4 Paso 4: Generar scripts .cxc para realizar comparaciones sin repeticiones

En este paso, se generan scripts .cxc que contienen las instrucciones necesarias para realizar comparaciones estructurales entre archivos .pdb en ChimeraX, siguiendo un esquema que evita repeticiones innecesarias. Estos scripts serán ejecutados en pasos posteriores para alinear y analizar las similitudes estructurales entre las proteínas cargadas.

Objetivos del paso

1. Alinear proteínas sin repeticiones: Crear scripts .cxc para realizar comparaciones estructurales entre pares únicos de archivos .pdb.

2. Automatizar el proceso: Generar los scripts de manera programada para evitar errores manuales y agilizar el análisis.
3. Guardar los scripts generados: Almacenar cada script .cxc en una carpeta específica (comparison_scripts) para facilitar su acceso y ejecución.

Detalles del proceso

1. Identificación de emparejamientos únicos:
    - Dado un conjunto de archivos .pdb (identificados en el Paso 1), se genera una lista de comparaciones únicas.
    - Por ejemplo:
        - Archivo 1 se compara con Archivos 2, 3, ..., n.
        - Archivo 2 se compara con Archivos 3, 4, ..., n.
        - Este esquema asegura que cada par de archivos se compare una sola vez, evitando duplicaciones.
2. Composición de los scripts .cxc:
    - Cada script .cxc corresponde a un archivo base (por ejemplo, Archivo 1) y contiene comandos matchmaker para alinearlo con los demás archivos.
    - El contenido de un script típico incluye comandos como:
    - open file1.pdb
    - open file2.pdb
    - matchmaker #0 #1
    - close all
    - Donde file1.pdb y file2.pdb representan rutas relativas o absolutas a los archivos .pdb.
3. Creación de múltiples scripts:
    - Se genera un archivo .cxc para cada archivo base en la lista pdb_files, excepto el último archivo (ya que no tiene pares restantes para comparar).
    - Cada archivo se guarda con un nombre único, como comparison_1.cxc, comparison_2.cxc, etc.
4. Carpeta de salida:

- Todos los scripts .cxc generados se almacenan en una carpeta específica llamada comparison_scripts, creada dentro del directorio raíz definido previamente.
- Si la carpeta no existe, el script la crea automáticamente antes de guardar los archivos.

Resultados esperados

- Scripts generados: Por cada archivo base, se genera un script .cxc que realiza las comparaciones únicas necesarias con los demás archivos de la lista.
- Carpeta organizada: Todos los scripts .cxc estarán organizados en la carpeta comparison_scripts, listos para ser ejecutados en ChimeraX.

Importancia del paso

Este paso automatiza la tarea de crear emparejamientos únicos para las comparaciones estructurales, asegurando que no haya redundancias y optimizando el análisis en ChimeraX. Esto no solo reduce los tiempos de configuración manual, sino que también mejora la reproducibilidad y organización del proyecto.

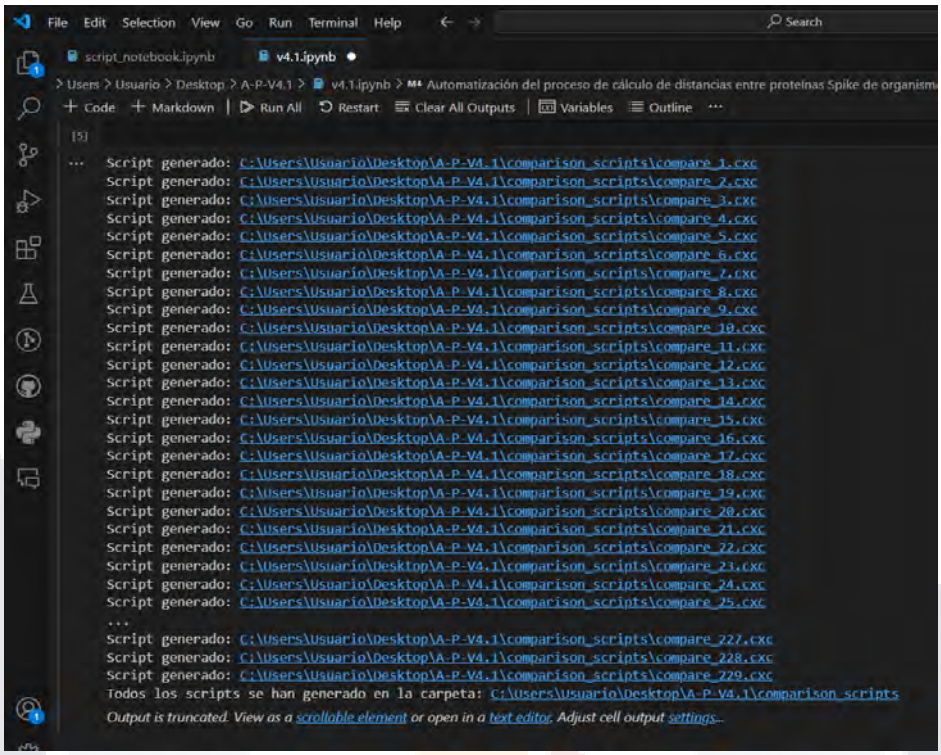


figura 9• Scripts generados por cada archivo base

Recomendaciones

- Antes de ejecutar este paso, asegúrate de que los archivos .pdb hayan sido correctamente identificados en el Paso 1 y cargados en ChimeraX en el Paso 3.
- Revisa los scripts generados para confirmar que las rutas de los archivos .pdb son correctas.

Notas adicionales

- Este esquema de emparejamiento es escalable y puede manejar listas de archivos más grandes.
- La estructura generada facilita la paralelización del análisis en sistemas que soporten múltiples procesos.

4.5 Paso 5: Ejecutar los scripts generados para realizar las comparaciones

En este paso, llevaremos a cabo las comparaciones estructurales de las proteínas cargadas en ChimeraX, utilizando los scripts .cxc generados en el Paso 4. Este proceso es fundamental para calcular las métricas estructurales (como el RMSD, el Alignment Score y los átomos pareados), que serán utilizadas en los pasos posteriores para clasificar y analizar las proteínas.

Objetivos del paso

1. Ejecutar los scripts de comparación: Ejecutar cada script .cxc de forma secuencial o en lotes para realizar las alineaciones estructurales en ChimeraX.
2. Generar archivos de registro: Durante la ejecución de los scripts, ChimeraX producirá logs que contienen los resultados de las comparaciones realizadas.
3. Optimizar el rendimiento: Dividir la ejecución en lotes para evitar sobrecargar los recursos computacionales disponibles.

Instrucciones detalladas

1. Abrir ChimeraX:
    - Inicia el software ChimeraX en tu computadora. Es importante que tengas instalada la versión utilizada durante los pasos anteriores para garantizar la compatibilidad con los scripts generados.
2. Acceder a la carpeta de scripts:
    - Navega hasta la carpeta comparison_scripts que contiene los scripts .cxc generados en el Paso 4. Esta carpeta debería estar ubicada dentro del directorio raíz definido previamente (e.g., C:\Users\Usuario\Desktop\A-P-V4.1\comparison_scripts).
3. Seleccionar scripts para ejecutar:
    - Dado que ejecutar todos los scripts al mismo tiempo podría sobrecargar los recursos de tu computadora, selecciona un número razonable de scripts para ejecutarlos en lotes. Por ejemplo:

    - Si tienes 100 scripts y tu computadora es moderada en rendimiento, puedes ejecutar 5 a 10 scripts al mismo tiempo.
    - Si tu computadora tiene mayores capacidades (procesador y RAM potentes), puedes probar con 20 o más.

4. Cargar y ejecutar scripts en ChimeraX:
    - En la barra de menús de ChimeraX:
        - Ve a File > Open.
        - Selecciona los scripts que deseas ejecutar (por ejemplo, comparison_1.cxc, comparison_2.cxc, ..., comparison_5.cxc).
        - Haz clic en Open para cargarlos y permitir que ChimeraX comience a procesar las comparaciones estructurales.
    - Durante la ejecución, ChimeraX abrirá y comparará los archivos .pdb especificados en cada script, realizando las alineaciones correspondientes.
5. Supervisar el progreso:
    - Observa la ventana de ejecución de ChimeraX para verificar que los scripts se están ejecutando correctamente.
    - A medida que se ejecutan, ChimeraX generará archivos de log en el directorio especificado en los scripts (o mostrará la información en pantalla, dependiendo de la configuración).
6. Finalizar un lote antes de cargar otro:
    - Espera a que todos los scripts seleccionados en el lote actual terminen de ejecutarse antes de proceder con otro grupo.
    - Esto asegura que los recursos de tu sistema no se saturen y permite identificar cualquier error de ejecución en los scripts.
7. Repetir hasta completar todos los scripts:
    - Continúa cargando y ejecutando scripts en lotes hasta completar todos los archivos en la carpeta comparison_scripts.

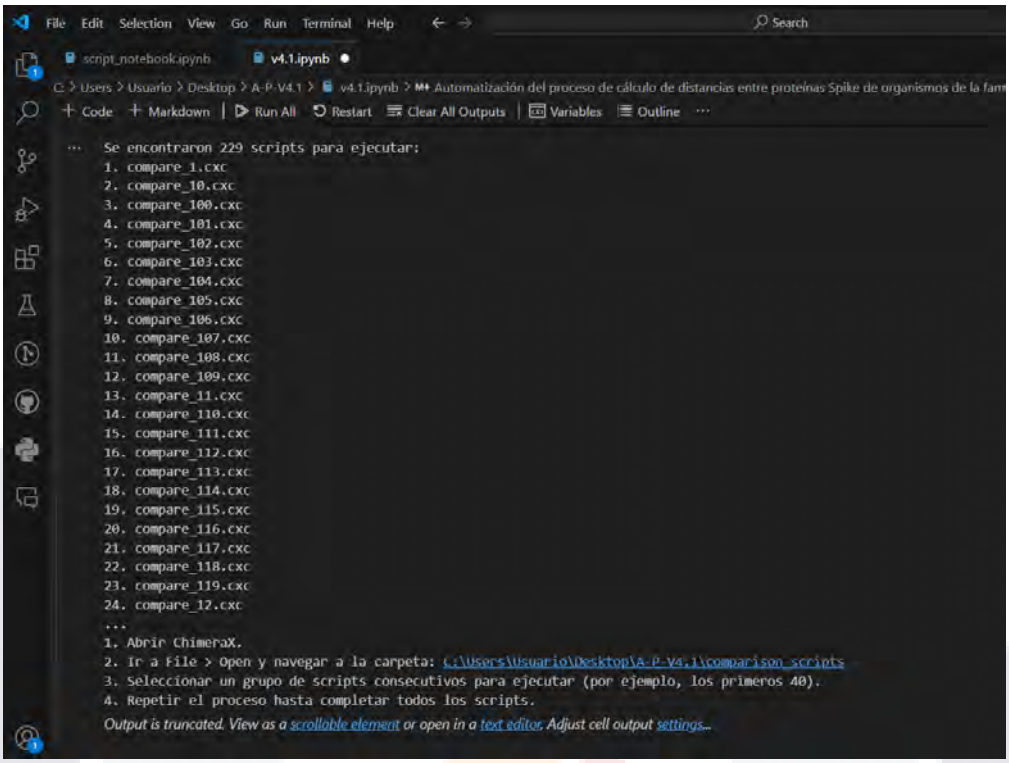


figura 10 Scripts generados para realizar las comparaciones

Notas importantes

- Verifica los resultados intermedios: Después de ejecutar cada lote de scripts, revisa los archivos de log generados para confirmar que las comparaciones se realizaron correctamente.
- Controla el uso de recursos: Si observas que ChimeraX responde de forma lenta o inestable, reduce el número de scripts en el próximo lote.
- Respaldo de logs: Asegúrate de guardar y respaldar los archivos de log generados, ya que serán cruciales para los pasos siguientes.

Resultado esperado

Al finalizar este paso:

1. Todos los scripts .cxc habrán sido ejecutados en ChimeraX.

2. Se habrán generado logs con los resultados de las comparaciones estructurales, que incluirán métricas como el RMSD, el Alignment Score y los átomos pareados para cada par de proteínas comparadas.

Estos resultados serán utilizados en los pasos posteriores para estructurar la información y calcular métricas adicionales, como la Distancia Normalizada Ponderada (DNP).

4.6 Paso 6: Guardar un único archivo de log después de todas las comparaciones

Este paso consolida toda la información generada durante las comparaciones en un único archivo de log (all_logs.txt). Esto simplifica el manejo y procesamiento de los datos en pasos posteriores y garantiza que toda la información relevante esté en un solo lugar.

Objetivos

1. Centralizar los resultados generados por los scripts .cxc ejecutados en ChimeraX.
2. Facilitar la extracción y análisis de las métricas estructurales (RMSD, Alignment Score, átomos pareados) mediante un único archivo.
3. Garantizar la organización y accesibilidad de los datos en un formato estandarizado.

Pasos detallados

1. Crear la carpeta de logs:
    - Dentro del directorio raíz (root_dir), crea una nueva carpeta llamada logs para organizar los resultados generados. Si ya existe una carpeta con este nombre, asegúrate de que no contenga información previa que pueda interferir.
2. Abrir ChimeraX:

- Asegúrate de que todas las comparaciones hayan sido ejecutadas en ChimeraX y que los resultados estén visibles en la ventana de log del programa.

3. Guardar el contenido del log:
    - Ve a la ventana de Log en ChimeraX (normalmente en la parte inferior o lateral del programa).
    - Accede al menú superior y selecciona:
        - File > Save
    - Navega hasta la carpeta logs que creaste en el directorio raíz.
    - Nombra el archivo como all_logs.txt y guarda.
    - Confirmación: ChimeraX generará un archivo que consolida todas las métricas calculadas durante las comparaciones, incluyendo RMSD, Alignment Score y átomos pareados.
4. Verificar el contenido del archivo:
    - Abre all_logs.txt con un editor de texto o un IDE para asegurarte de que contiene las métricas completas de todas las comparaciones.

Resultados esperados

- Archivo consolidado: Un archivo único llamado all_logs.txt almacenado en la carpeta logs, conteniendo todas las comparaciones realizadas.
- Estructura organizada: Todos los datos estarán disponibles en un formato legible y accesible para los pasos siguientes, como la creación del DataFrame y la normalización de métricas.

Este paso asegura que los resultados generados por ChimeraX estén centralizados y listos para ser procesados, simplificando el flujo de trabajo y facilitando el análisis de las métricas.

4.7 Paso 7: Crear un DataFrame con la información obtenida

Este paso procesa el archivo de logs consolidado (all_logs.txt) generado en el Paso 6 para extraer información clave de las comparaciones realizadas entre proteínas y construir un DataFrame estructurado. Este DataFrame será esencial para los análisis posteriores y contendrá información detallada de cada comparación junto con los géneros asociados a las proteínas.

Objetivo

- Extraer métricas estructurales (RMSD, Alignment Score, átomos pareados) y asociarlas con información adicional sobre los modelos involucrados, como sus nombres y géneros.
- Consolidar los datos en un formato tabular que permita una fácil manipulación y análisis en Python.

Fuentes de datos

1. Logs consolidados:
    - Archivo all_logs.txt, ubicado en la carpeta logs.
    - Contiene los resultados de las comparaciones realizadas en ChimeraX, incluyendo métricas estructurales y nombres de los modelos.
2. Archivo de géneros:
    - Archivo Genero.xlsx, ubicado en root_dir.
    - Contiene una tabla con dos columnas:
        - Modelo: Nombre del modelo de proteína.
        - Género: Género al que pertenece cada modelo.

Estructura del DataFrame

El DataFrame tendrá las siguientes columnas:

1. Comando: El comando utilizado para realizar la comparación (matchmaker).
2. Modelo 1 ID: Identificador del primer modelo (por ejemplo, #1).

3. Modelo 1 Nombre: Nombre completo del primer modelo.
4. Modelo 1 Género: Género del primer modelo (extraído de Genero.xlsx).
5. Modelo 2 ID: Identificador del segundo modelo (por ejemplo, #2).
6. Modelo 2 Nombre: Nombre completo del segundo modelo.
7. Modelo 2 Género: Género del segundo modelo (extraído de Genero.xlsx).
8. Alignment Score: Puntaje de alineación entre los modelos.
9. Átomos Pareados: Número de átomos correctamente alineados.
10. RMSD: Raíz cuadrática media de las desviaciones entre los átomos equivalentes.

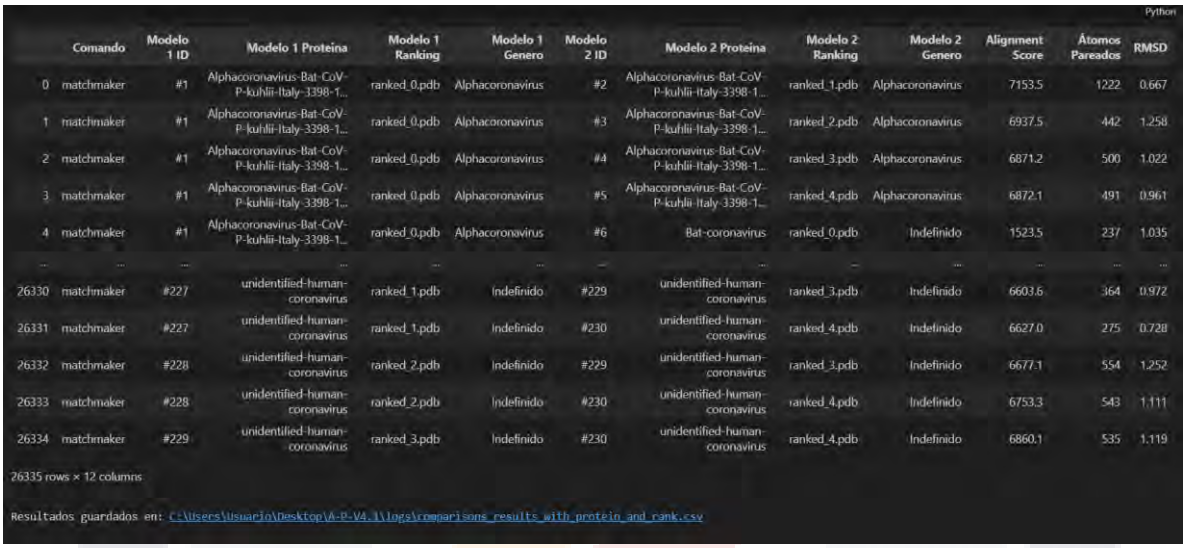
Pasos detallados

1. Cargar las dependencias necesarias:
    - Utiliza las bibliotecas pandas y re para procesar y extraer datos.
    - Lee el archivo Genero.xlsx en un DataFrame para mapear géneros.
2. Leer y procesar el archivo de logs:
    - Lee el contenido de all_logs.txt.
    - Utiliza expresiones regulares para extraer información clave (nombres de modelos, RMSD, Alignment Score, átomos pareados).
3. Construir el DataFrame:
    - Crea un DataFrame vacío con las columnas definidas.
    - Itera sobre los resultados del log y completa cada fila con la información extraída.
4. Mapear géneros:
    - Usa los nombres de los modelos para buscar el género correspondiente en el archivo Genero.xlsx.
    - Completa las columnas Modelo 1 Género y Modelo 2 Género.
5. Guardar el DataFrame:
    - Guarda el DataFrame en un archivo CSV llamado comparisons_results_with_genero.csv, dentro de la carpeta logs.

Resultados:

- DataFrame consolidado: Contiene toda la información relevante para cada comparación, incluyendo géneros, métricas y nombres de los modelos.
- Archivo CSV: Un archivo comparisons_results_with_genero.csv en la carpeta logs para análisis posterior.

Ejemplo de estructura del DataFrame:



Python

	Comando	Modelo 1 ID	Modelo 1 Proteina	Modelo 1 Ranking	Modelo 1 Genero	Modelo 2 ID	Modelo 2 Proteina	Modelo 2 Ranking	Modelo 2 Genero	Alignment Score	Átomos Pareados	RMSD
0	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#2	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_1.pdb	Alphacoronavirus	7153.5	1222	0.667
1	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#3	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_2.pdb	Alphacoronavirus	6937.5	442	1.258
2	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#4	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_3.pdb	Alphacoronavirus	6871.2	500	1.022
3	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#5	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_4.pdb	Alphacoronavirus	6872.1	491	0.961
4	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#6	Bat-coronavirus	ranked_0.pdb	Indefinido	1523.5	237	1.035
...	...	...	...	...	...	...	...	...	...	...	...	...
26330	matchmaker	#227	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	6603.6	364	0.972
26331	matchmaker	#227	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6627.0	275	0.728
26332	matchmaker	#228	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	6677.1	554	1.252
26333	matchmaker	#228	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6753.3	543	1.111
26334	matchmaker	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6860.1	535	1.119

26335 rows × 12 columns

Resultados guardados en: C:\Users\Usuario\Desktop\A-P-V4.1\logs\comparisons_results_with_protein_and_rank.csv

figura 11 Vista del DataFrame "df_logs_global"

Este DataFrame centraliza toda la información para los pasos siguientes, como la normalización de atributos y el cálculo de la métrica DNP.

4.8 Paso 8: Predecir el género utilizando Alignment Score, RMSD, y Átomos Pareados

Este paso tiene como objetivo construir un modelo de clasificación supervisada que permita predecir el género de las proteínas utilizando las métricas obtenidas en las comparaciones. Se emplean Alignment Score, RMSD, y Átomos Pareados como atributos predictivos y un modelo Random Forest como clasificador.

Objetivos del Paso:

1. Identificar qué características tienen mayor influencia en la predicción del género.

2. Construir y evaluar un modelo de clasificación supervisada.
3. Obtener un modelo funcional para predecir géneros basados en atributos de las comparaciones estructurales.

Pasos detallados:

1. Preparar los datos
    - Se utiliza el archivo comparisons_results_with_genero.csv generado en el Paso 7 como fuente de datos.
    - Las columnas seleccionadas como características predictoras son:
        - Alignment Score
        - RMSD
        - Átomos Pareados
    - La variable objetivo es Modelo 1 Genero, que indica el género de la proteína principal en cada comparación.
    - Los datos se dividen en conjuntos de entrenamiento (70%) y prueba (30%) para evaluar el desempeño del modelo.
2. Entrenar el modelo
    - Se utiliza el modelo Random Forest, que es un clasificador basado en árboles de decisión.
    - Este algoritmo es adecuado debido a su capacidad para manejar relaciones no lineales y su robustez frente a ruido en los datos.
    - Durante el entrenamiento, el modelo aprende las relaciones entre los atributos y los géneros a partir del conjunto de entrenamiento.
3. Evaluar el modelo
    - Se evalúa el desempeño del modelo en el conjunto de prueba mediante métricas clave:
        - Precisión: Proporción de predicciones correctas sobre el total de predicciones.
        - Recall: Capacidad del modelo para identificar correctamente los géneros reales.

- F1-score: Promedio ponderado de precisión y recall, útil cuando las clases están desbalanceadas.

- Además, se genera una matriz de confusión para analizar visualmente los aciertos y errores del modelo en las predicciones.

4. Importancia de las características
   - Se calcula la importancia relativa de las características predictivas (Alignment Score, RMSD, Átomos Pareados).
   - Esto permite identificar qué métricas estructurales tienen un mayor impacto en la clasificación de los géneros.
5. Hacer predicciones
   - Una vez entrenado y evaluado, el modelo se utiliza para predecir el género de nuevas proteínas o comparaciones estructurales.

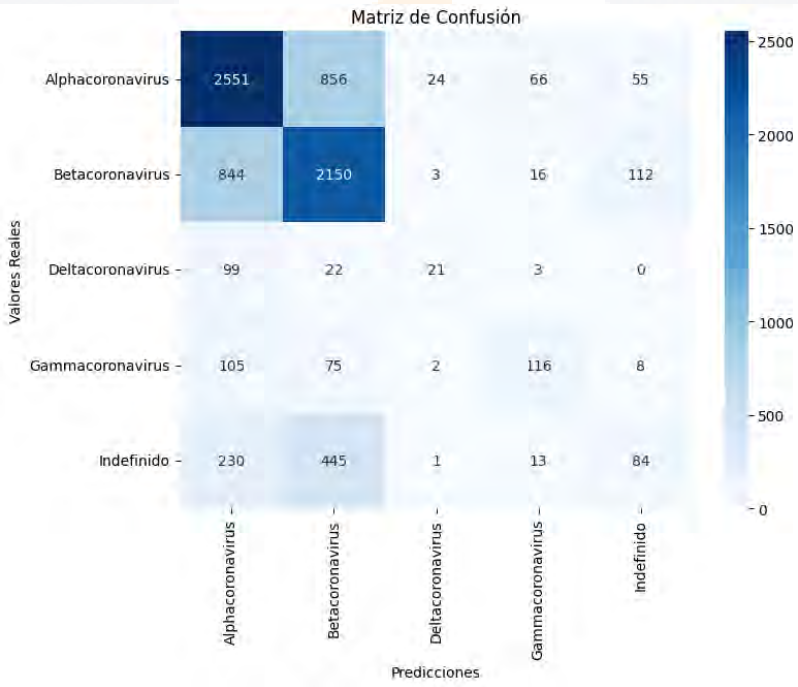


figura 12 Matriz de Confusión

Resultados esperados:

- Un modelo de clasificación supervisada funcional que predice géneros con alta precisión.

- Un análisis de la importancia de las características, que sirve como referencia para la normalización y el cálculo de métricas como la DNP en los pasos siguientes.
- Reportes de desempeño claros y visuales, como la matriz de confusión y gráficos de importancia de características.

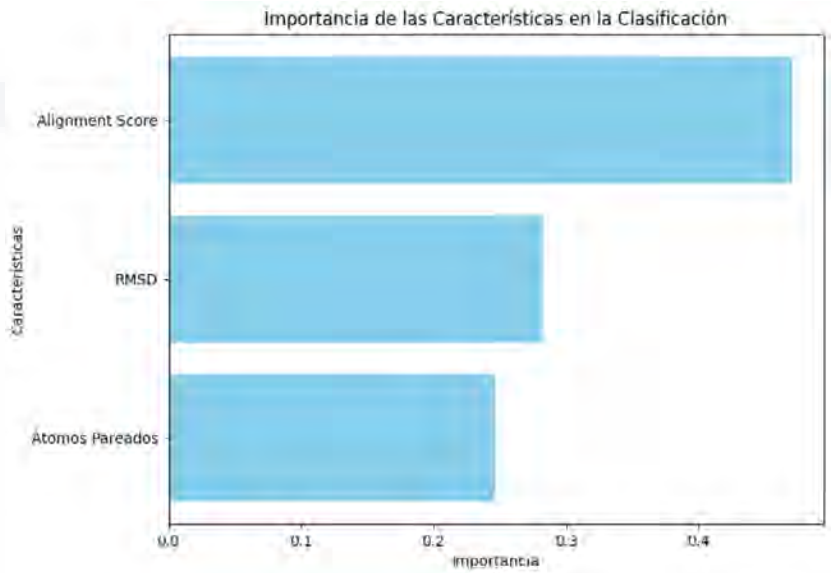


figura 13 Importancia de las características en la clasificación

Este paso no solo permite predecir géneros con un modelo robusto, sino que también proporciona un análisis valioso sobre la importancia relativa de las características. Esto será esencial para los pasos posteriores, como la normalización y el cálculo de la métrica DNP.

4.9 Paso 9: Normalización de atributos y cálculo de la Distancia Normalizada Ponderada (DNP)

Este paso se centra en transformar los valores de las métricas clave (Alignment Score, RMSD, Átomos Pareados) en un rango común y calcular un único atributo, la Distancia

Normalizada Ponderada (DNP). Este atributo sintetiza la información de similitud estructural entre proteínas, facilitando el análisis y la clasificación.

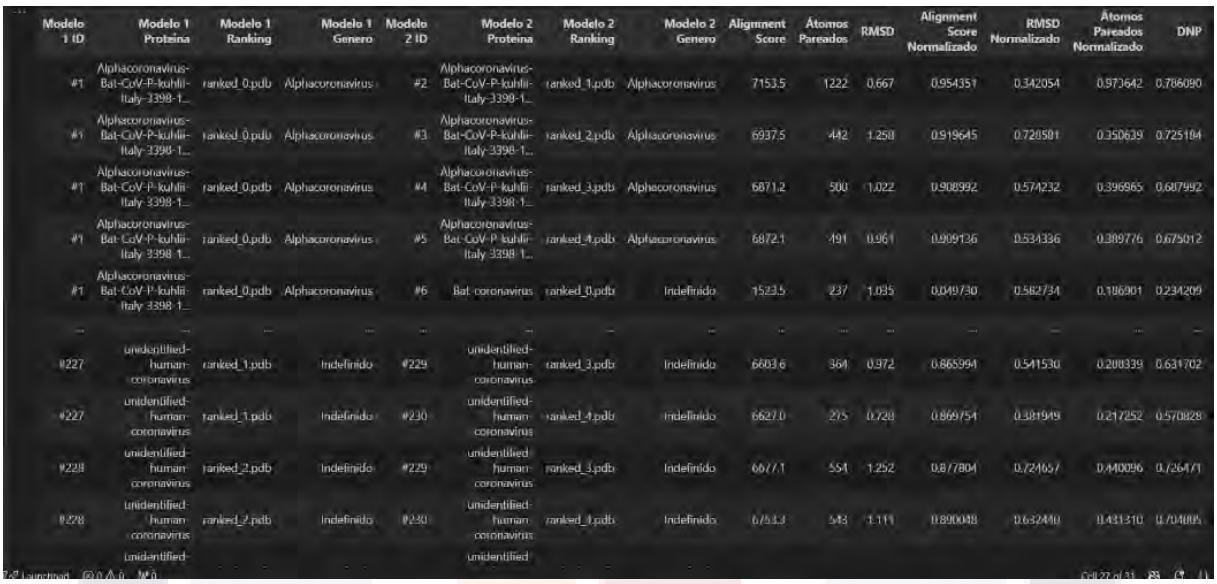
Pasos detallados

1. Normalización de atributos
    - Para garantizar la comparabilidad entre los atributos, se aplica la técnica de Min-Max Scaling, que transforma los valores de cada atributo en un rango entre 0 y 1.
    - Este método elimina sesgos derivados de escalas diferentes y asegura que cada métrica contribuya proporcionalmente al cálculo de la DNP.
    - Fórmula para la normalización:

$$\mathrm{A}_{\mathrm{normalizado}} = \frac{A - A_{min}}{A_{max} - A_{min}}$$

    Donde:
        - A: Valor del atributo original.
        - A_{min}, A_{max}: Mínimo y máximo de la columna correspondiente.
2. Cálculo de la DNP
    - Esta magnitud se calcula en el Capítulo2 y es el atributo que conforma la matriz de distancia.
    - Este cálculo refleja la importancia relativa de cada atributo en la similitud estructural.
3. Agregar columnas al DataFrame
    - Se agregan cuatro nuevas columnas al DataFrame para los valores normalizados y el DNP:
        - Alignment Score Normalizado

- RMSD Normalizado
- Átomos Pareados Normalizado
- DNP



Modelo 1 ID	Modelo 1 Proteína	Modelo 1 Ranking	Modelo 1 Genero	Modelo 2 ID	Modelo 2 Proteína	Modelo 2 Ranking	Modelo 2 Genero	Alignment Score	Átomos Pareados	RMSD	Alignment Score Normalizado	RMSD Normalizado	Átomos Pareados Normalizado	DNP
#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#2	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_1.pdb	Alphacoronavirus	7153.5	1222	0.667	0.954351	0.342054	0.979642	0.786090
#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#3	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_2.pdb	Alphacoronavirus	6937.5	442	1.258	0.919645	0.720581	0.350639	0.725184
#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#4	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_3.pdb	Alphacoronavirus	6871.2	500	1.022	0.908992	0.574232	0.396965	0.687992
#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#5	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_4.pdb	Alphacoronavirus	6872.1	491	0.961	0.909136	0.534336	0.389776	0.675012
#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#6	Bat coronavirus	ranked_0.pdb	Indefinido	1523.5	237	1.035	0.049730	0.582734	0.186901	0.234209
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
#227	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	6603.6	364	0.972	0.865994	0.541530	0.288339	0.631702
#227	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6627.0	275	0.728	0.869754	0.381949	0.217252	0.570828
#228	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	6677.1	554	1.252	0.877804	0.724667	0.440096	0.726471
#228	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6753.3	543	1.111	0.890048	0.632440	0.431310	0.704005
	unidentified-				unidentified									

figura 14 Vista del DataFrame "df_logs_normalized_global"

4. Guardar resultados
    - El DataFrame actualizado se guarda como una variable global llamada df_logs_normalized_global para su uso en pasos posteriores.
    - También se exporta como un archivo CSV llamado df_logs_normalized_with_dnp.csv, ubicado en la carpeta logs dentro de root_dir.

4.10 Paso 10: Implementación del Algoritmo Genético (AG) para la Clasificación de Proteínas

En este paso, el modelo propuesto utiliza Algoritmos Genéticos (AG) para identificar patrones y clasificar las proteínas de la familia Coronaviridae basándose en su similitud estructural tridimensional. Este paso aprovecha la métrica Distancia Normalizada Ponderada (DNP) calculada previamente como criterio central para evaluar las soluciones propuestas. Los AG son herramientas metaheurísticas basadas en los principios de evolución biológica, como la selección natural, la recombinación y la mutación, y son particularmente útiles para resolver problemas de optimización combinatoria complejos.

Estructura del Algoritmo Genético

1. Codificación de soluciones
    - Cada solución se representa como un individuo de la población inicial.
    - En este contexto, un individuo se codifica como un vector donde cada gen corresponde a una proteína, y su valor representa el género asignado (por ejemplo, género1, género2, género3, género4).
2. Población inicial
    - La población inicial contiene 100 individuos, generados aleatoriamente para garantizar diversidad y evitar sesgos iniciales.
    - Cada individuo se evalúa según su calidad (fitness) en función de las distancias calculadas (DNP).
3. Función de fitness
    - La función de fitness evalúa la calidad de las soluciones, basándose en dos objetivos principales:
        1. Minimizar las distancias internas dentro de cada género.
        2. Maximizar las distancias externas entre géneros.

- Fórmula de fitness: F(i)=α·1Promedio Distancias Internas+β·Promedio Distancias ExternasF(i) = \alpha \cdot \frac{1}{\text{Promedio Distancias Internas}} + \beta \cdot \text{Promedio Distancias Externas}F(i)=α·Promedio Distancias Internas1 +β·Promedio Distancias Externas Donde:
    - α\alphaα y β\betaβ: Pesos ajustables para equilibrar la importancia relativa de ambos objetivos.
    - Las distancias se basan en la métrica DNP.

4. Operadores genéticos
    - Selección:
        - Se utiliza selección por torneo, donde se seleccionan individuos compitiendo en pequeños grupos. Esto asegura que los mejores individuos tengan mayor probabilidad de reproducirse.
    - Cruzamiento (recombinación):
        - Se aplica un cruce uniforme, donde los genes de dos padres se combinan aleatoriamente para generar dos hijos. Esto promueve la exploración del espacio de búsqueda.
    - Mutación:
        - Se introduce variación aleatoria en un porcentaje de genes (por ejemplo, cambiando el género asignado a una proteína). Esto evita que el algoritmo quede atrapado en óptimos locales.
    - Elitismo:
        - Los mejores individuos de cada generación se preservan automáticamente para asegurar que las mejores soluciones no se pierdan.
5. Parámetros del algoritmo

- Tamaño de la población: 100 individuos.
- Número máximo de generaciones: 200.
- Probabilidad de cruzamiento: 0.8.
- Probabilidad de mutación: 0.1.
- Criterio de parada: El algoritmo se detiene al alcanzar el número máximo de generaciones o si la mejora en el fitness es menor a un umbral predefinido.

Flujo del Algoritmo

1. Inicialización:
    - Se genera una población inicial aleatoria.
    - Cada individuo se evalúa utilizando la función de fitness.
2. Evolución de generaciones:
    - Se seleccionan individuos para cruzamiento usando selección por torneo.
    - Los hijos generados mediante cruzamiento son sometidos a mutación aleatoria.
    - La nueva generación se evalúa y reemplaza a la anterior, manteniendo los mejores individuos (elitismo).
3. Convergencia:
    - El algoritmo iterativamente mejora la calidad promedio de la población hasta que cumple el criterio de parada.
4. Resultado final:
    - El mejor individuo de la última generación representa la mejor solución encontrada:
        - Clasificación de proteínas en géneros.

- Agrupaciones que maximizan diferencias entre géneros y minimizan distancias internas.

Rol de los elementos de la metaheurística:

1. Selección:
    - Garantiza que los individuos con mejor fitness tengan mayores probabilidades de transmitir sus genes a la próxima generación, acelerando la convergencia hacia soluciones óptimas.
2. Cruzamiento:
    - Combina soluciones existentes para explorar nuevas áreas del espacio de búsqueda, aumentando la probabilidad de encontrar soluciones globales óptimas.
3. Mutación:
    - Introduce diversidad en la población, ayudando a escapar de óptimos locales y promoviendo la exploración de soluciones menos evidentes.
4. Elitismo:
    - Asegura que las mejores soluciones no se pierdan durante la evolución, contribuyendo a la estabilidad del proceso.

Resultados esperados:

- Una clasificación de proteínas en géneros con una alta coherencia estructural y evolutiva.
- Reducción de distancias internas y maximización de las distancias externas.
- Identificación de patrones que reflejan similitudes estructurales clave entre proteínas.

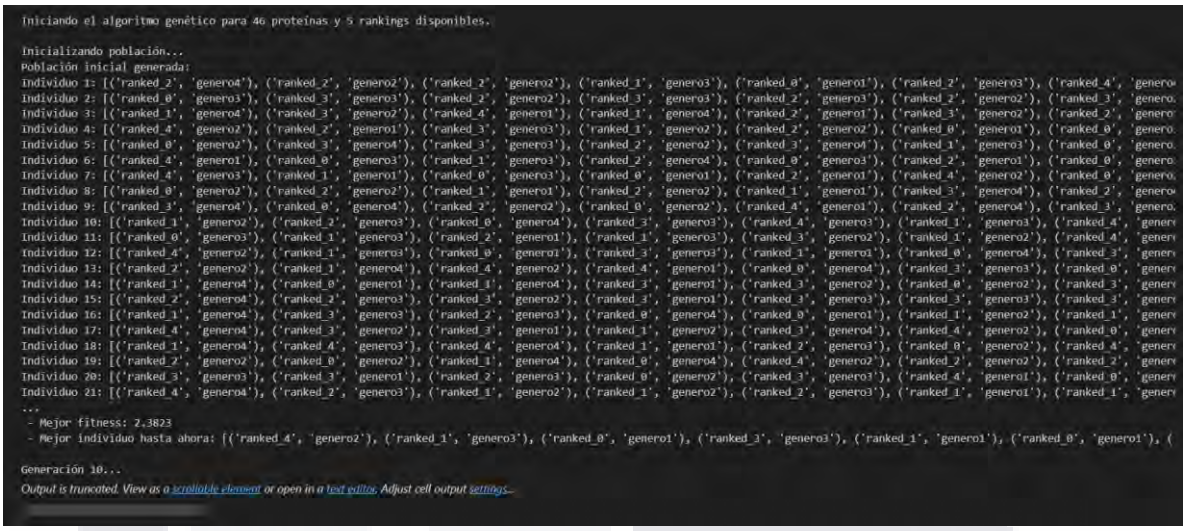


figura 15 Salida del Algoritmo Genético

El Algoritmo Genético utilizado en este modelo es una herramienta poderosa para abordar el problema de clasificación de proteínas en un espacio de búsqueda complejo. Al integrar métricas personalizadas como la DNP y aprovechar los operadores genéticos, este enfoque logra soluciones de alta calidad que reflejan patrones estructurales y evolutivos consistentes en las proteínas de la familia Coronaviridae. Este paso final consolida la contribución del modelo al campo de la biología computacional y al estudio de las proteínas virales.

Capítulo 5: Realización de Experimentos

5.1 Introducción

La realización de los experimentos fue fundamental para validar el modelo propuesto y demostrar su eficacia en la clasificación de proteínas Spike de la familia Coronaviridae. Utilizando los 10 pasos documentados en el notebook de Jupyter, se implementaron procesos sistemáticos para la recolección, procesamiento y análisis de datos, integrando herramientas avanzadas como ChimeraX y algoritmos genéticos. A pesar de los desafíos computacionales, como el elevado número de comparaciones y el tiempo de cálculo requerido, los resultados obtenidos fueron satisfactorios, evidenciando la solidez y aplicabilidad del modelo desarrollado.

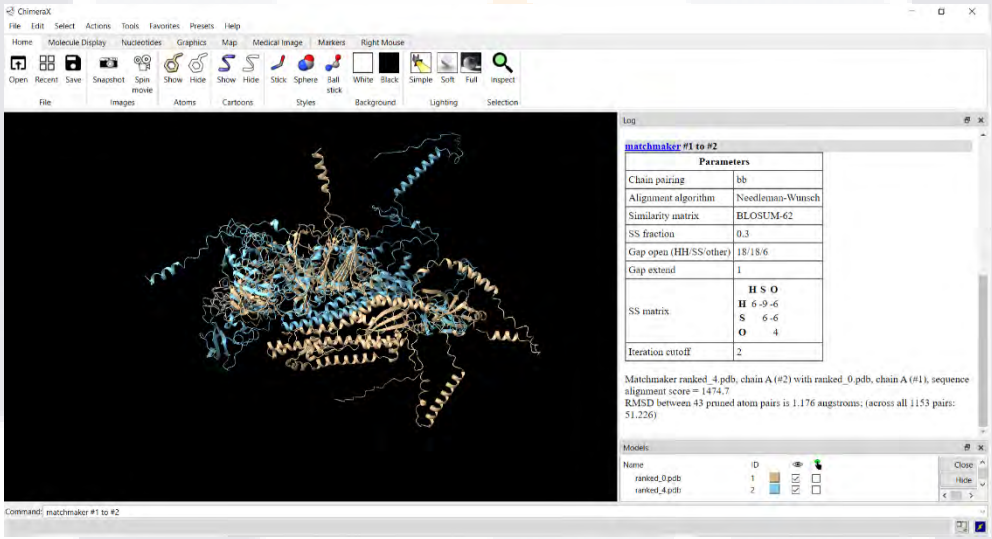


figura 16.ChimeraX

5.2 Preparación de los Datos

El primer paso en la realización de los experimentos fue la preparación y organización de los datos estructurales. Partiendo de las predicciones generadas por AlphaFold, que produjeron cinco modelos tridimensionales por proteína, se consolidó una estructura de datos uniforme. Cada modelo fue almacenado en formato .pdb y organizado en carpetas individuales por

proteína. Este diseño sistemático facilitó el acceso y procesamiento posterior mediante scripts automatizados en Python.

En total, se trabajó con 46 proteínas y 230 modelos asociados, generando un conjunto de datos robusto para las etapas subsecuentes. La preparación de los datos incluyó la extracción de métricas estructurales clave y la normalización de estos valores para su análisis en el algoritmo genético.

5.3 Comparación Estructural

El análisis comparativo de los modelos se realizó utilizando ChimeraX y su comando MatchMaker, que permite alinear estructuras tridimensionales y calcular métricas de similitud, como el RMSD, el Alignment Score y el número de átomos pareados. Con más de 26,000 comparaciones necesarias, el proceso fue automatizado mediante la generación de scripts .cxc que optimizaron las tareas repetitivas.

Aunque el tiempo de cómputo para esta etapa fue significativo debido al gran número de comparaciones, los logs generados proporcionaron un conjunto de datos detallado y estructurado. Estos resultados sirvieron como insumo clave para calcular el atributo único ponderado DNP, una métrica diseñada para representar la similitud estructural de manera integral.

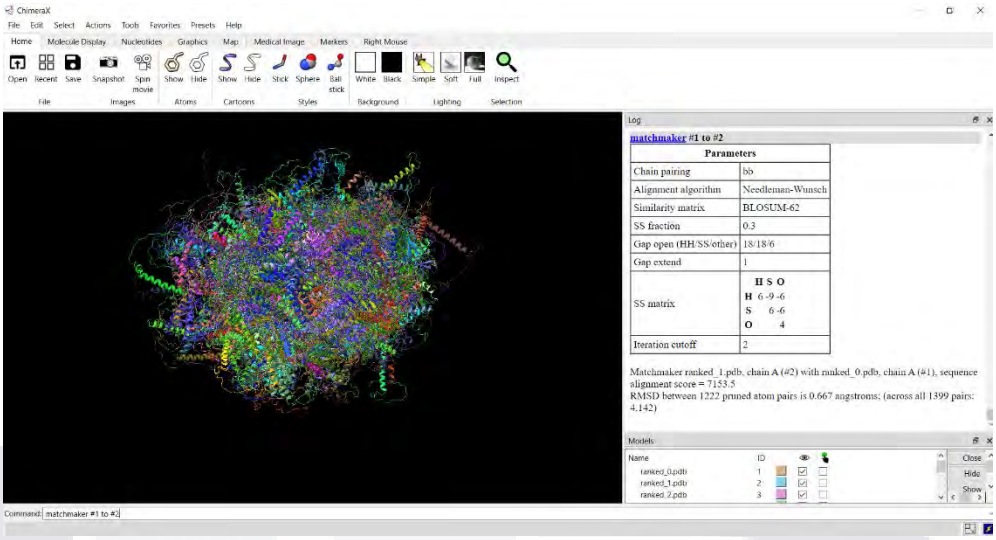


figura 17 Matchmaker en ChimeraX

5.4 Implementación del Algoritmo Genético

El núcleo experimental se centró en la implementación del algoritmo genético, cuyo objetivo fue clasificar las proteínas en cuatro géneros (genero1, genero2, genero3, genero4) y seleccionar el ranking más representativo de los modelos generados por AlphaFold. El algoritmo utilizó una población inicial diversificada, que evolucionó a través de operadores genéticos como selección por torneo, cruce uniforme y mutación.

La evaluación de las soluciones se basó en la métrica DNP, que permitió calcular distancias estructurales dentro y entre géneros. Durante el proceso, se observó que poblaciones grandes incrementaban el tiempo computacional, aunque favorecían una mayor diversidad y calidad en las soluciones. Se realizaron múltiples ejecuciones para ajustar parámetros y garantizar una convergencia estable en el modelo.

5.5 Resultados Obtenidos

Los experimentos generaron resultados satisfactorios, clasificando las 46 proteínas en géneros y seleccionando los rankings óptimos con base en patrones estructurales consistentes. El algoritmo genético demostró su capacidad para minimizar distancias dentro

de los géneros y maximizar distancias entre ellos, logrando agrupaciones coherentes desde una perspectiva evolutiva.

Sin embargo, el tiempo computacional fue elevado, especialmente en las etapas de comparación estructural y en la evolución de generaciones en el algoritmo genético. Este resultado pone de manifiesto la necesidad de optimizar parámetros y explorar técnicas complementarias que reduzcan la complejidad computacional en estudios futuros.

5.6 Análisis del Tiempo Computacional

El alto costo computacional fue uno de los principales desafíos enfrentados durante la realización de los experimentos. La comparación estructural mediante ChimeraX, que involucró más de 26,000 emparejamientos, requirió un tiempo significativo de cálculo. De manera similar, la implementación del algoritmo genético, particularmente con poblaciones grandes, extendió los tiempos de procesamiento, aunque garantizó una mayor diversidad en las soluciones.

Este aspecto subraya la importancia de explorar estrategias de optimización adicionales, como la reducción del espacio de búsqueda mediante técnicas de clustering previo o la paralelización del procesamiento en etapas críticas.

5.7 Conclusión del Capitulo

A pesar de los retos asociados al tiempo computacional, los resultados obtenidos validaron la eficacia del modelo propuesto en la clasificación de proteínas Spike. Los experimentos demostraron que el uso de algoritmos genéticos, en combinación con métricas como el DNP, es una estrategia robusta y flexible para identificar patrones estructurales en proteínas. Este enfoque no solo ofrece una base sólida para investigaciones futuras, sino que también destaca la necesidad de optimizar el procesamiento computacional para manejar eficientemente conjuntos de datos grandes y complejos.

Capítulo 6: Evaluación de los Resultados

6.1 Introducción

La evaluación de los resultados se llevó a cabo mediante la comparación entre la clasificación generada por el modelo basado en algoritmos genéticos (AG) y el árbol filogenético construido utilizando el método Neighbor-Joining (NJ) en el software MEGA. El objetivo de esta evaluación fue validar la precisión del modelo propuesto y analizar posibles discrepancias en la clasificación de géneros, especialmente en los casos en que las proteínas se agrupan en géneros no definidos o poco establecidos por la comunidad científica. Los resultados evidencian una alta concordancia entre las dos aproximaciones, destacando las fortalezas y limitaciones de cada enfoque.

6.2 Construcción del Árbol Filogenético en MEGA

El método Neighbor-Joining (NJ), empleado en MEGA, es una técnica robusta basada en distancias que genera árboles filogenéticos eficientes al minimizar las longitudes totales de las ramas. Para este análisis, se utilizó la matriz de distancias basada en el atributo único ponderado DNP, derivado de las métricas estructurales (RMSD, Alignment Score, y átomos pareados).

El árbol resultante agrupó las 46 proteínas Spike en clados coherentes con las distancias calculadas, reflejando relaciones evolutivas consistentes. Aunque MEGA no asigna géneros explícitamente, se etiquetaron los clados según los géneros tradicionales reconocidos por la comunidad científica (e.g., alpha, beta), facilitando la visualización y el análisis comparativo.

6.3 Concordancia entre MEGA y AG

El modelo AG y el árbol NJ mostraron un alto grado de concordancia en la agrupación de proteínas en géneros. Ambas aproximaciones identificaron correctamente las relaciones

estructurales más estrechas dentro de los géneros predefinidos, destacando la robustez del DNP como métrica integradora de similitud estructural.

Sin embargo, se identificaron diferencias específicas relacionadas con la capacidad del modelo AG para predecir géneros:

- Capacidad predictiva del AG: A diferencia de MEGA, que clasifica únicamente en función de distancias, el AG asigna géneros incluso en casos de géneros indefinidos. Esto permite una categorización más exhaustiva, pero introduce el riesgo de errores interpretativos cuando individuos de géneros no definidos son asignados a un género existente.
- Errores de asignación: En algunos casos, el AG agrupó proteínas de géneros indefinidos dentro de géneros existentes debido a su proximidad estructural. Esto podría generar malinterpretaciones, especialmente si la asignación no está respaldada por evidencia taxonómica adicional.

A pesar de estas limitaciones, el AG clasificó correctamente la gran mayoría de las proteínas, evidenciando su capacidad para identificar patrones consistentes y realizar agrupaciones estructuralmente coherentes.

6.4 Ventajas y Limitaciones de Ambos Métodos

Método Neighbor-Joining en MEGA

Ventajas:

- Genera árboles basados exclusivamente en distancias, lo que elimina la necesidad de predicción directa de géneros.
- Proporciona una representación visual clara de las relaciones filogenéticas.
- Es ampliamente aceptado por la comunidad científica y se puede contrastar con clasificaciones taxonómicas existentes.

Limitaciones:

- No asigna géneros directamente, lo que limita su uso para tareas de clasificación predictiva.
- Depende únicamente de distancias, sin integrar directamente información contextual sobre los géneros.

Modelo AG

Ventajas:

- Integra información adicional para predecir géneros, incluso en casos donde las relaciones no son claramente definidas.
- Es adaptable a distintos problemas de clasificación combinatoria, lo que permite ajustar sus parámetros para mejorar la precisión.
- Utiliza métricas estructurales ponderadas que reflejan con mayor precisión las similitudes funcionales entre proteínas.

Limitaciones:

- La predicción de géneros puede generar errores en casos donde las distancias son pequeñas pero el género real no está definido o reconocido.
- El tiempo computacional requerido para ejecutar el algoritmo es considerablemente alto debido al gran número de comparaciones y soluciones evaluadas.

6.5 Conclusiones de la Evaluación

Los resultados obtenidos demuestran la eficacia del modelo AG en la clasificación de proteínas, logrando una alta concordancia con el árbol filogenético generado por MEGA. Si bien el AG presenta limitaciones relacionadas con la predicción de géneros no definidos, su capacidad para identificar patrones estructurales y realizar agrupaciones coherentes es notable.

Por su parte, MEGA, al centrarse únicamente en las distancias, evita errores asociados a la asignación de géneros, pero carece de una capacidad predictiva directa. Esta diferencia en

enfoques complementa los hallazgos y refuerza la importancia de utilizar ambos métodos en conjunto para lograr una clasificación más robusta y fiable.

En resumen, el modelo AG clasifica correctamente en la mayoría de los casos y ofrece una herramienta poderosa para predecir géneros, mientras que MEGA proporciona una validación independiente basada en distancias. La combinación de estas metodologías resalta el potencial de integrar herramientas computacionales en el análisis filogenético y evolutivo de proteínas.

Conclusiones Generales

El presente estudio ha demostrado la eficacia del modelo propuesto basado en algoritmos genéticos (AG) para la clasificación de proteínas Spike de la familia Coronaviridae, destacándose como una herramienta robusta y flexible para identificar patrones estructurales y evolutivos. A través de la integración de métricas personalizadas como la Distancia Normalizada Ponderada (DNP) y el análisis computacional avanzado, se logró una clasificación coherente que mostró alta concordancia con los árboles filogenéticos generados mediante el método Neighbor-Joining (NJ) en MEGA. Si bien MEGA ofrece una representación precisa de distancias estructurales, su limitación para predecir géneros específicos resalta el valor añadido del AG, que permite una categorización más exhaustiva incluso en casos de géneros indefinidos. Sin embargo, este enfoque predictivo introduce desafíos, como la posible asignación errónea en contextos de géneros no definidos por la comunidad científica, subrayando la necesidad de evidencia taxonómica complementaria. Además, los resultados ponen de manifiesto la importancia de optimizar el tiempo computacional, ya que el análisis de grandes volúmenes de datos, como las más de 26,000 comparaciones estructurales, representa un desafío significativo en términos de recursos computacionales. En síntesis, el modelo propuesto no solo proporciona una herramienta valiosa para clasificar proteínas, sino que también establece un enfoque metodológico integral que puede aplicarse a futuros estudios filogenéticos y estructurales, contribuyendo al avance del conocimiento en biología molecular y computacional.

Referencias

Adiyaman, R., & McGuffin, L. J. (2019). Methods for the refinement of protein structure 3D models. *International Journal of Molecular Sciences*, *20*(9). https://doi.org/10.3390/ijms20092301

Alancay, N., Villagra, S. M., & Villagra, N. A. (2016a). Metaheurísticas de trayectoria y poblacional aplicadas a problemas de optimización combinatoria. *Informes Científicos Técnicos - UNPA*, *8*(1). https://doi.org/10.22305/ict-unpa.v8i1.157

Alancay, N., Villagra, S., & Villagra, N. A. (2016b). Algoritmos metaheurísticos trayectoriales para optimizar problemas combinatorios. *Informes Científicos Técnicos - UNPA*, *8*(3). https://doi.org/10.22305/ict-unpa.v8i3.222

Barreto Hernández, E. (2008). Bioinformática: una oportunidad y un desafío. *Revista Colombiana de Biotecnología*, *X*(1), 132–138.

Blank, J., & Deb, K. (2020). Pymoo: Multi-Objective Optimization in Python. *IEEE Access*, *8*, 89497–89509. https://doi.org/10.1109/ACCESS.2020.2990567

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, *35*(3), 268–308. https://doi.org/10.1145/937503.937505

Cech, J. (2019). Síntesis De Proteínas. In *Universidad Central de Mexico*.

Elshaer, R., & Awad, H. (2020). A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers and Industrial Engineering*, *140*. https://doi.org/10.1016/j.cie.2019.106242

Feig, M. (2017). Computational protein structure refinement: almost there, yet still so far to go. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, *7*(3). https://doi.org/10.1002/wcms.1307

Galvis Motoa, S. I. (2022). Clusterización de proteínas mediante metaheurísticas multiobjetivo en la familia coronaviridae. In *Universidad autónoma de Aguascalientes*.

Gómez-Moreno Calerra, C. (2000). *Estructura de proteínas*. Editorial Ariel.

Granillo Macias, R. (2019). Optimización- Algoritmos programados con Matlab. *Ingenio y Conciencia Boletín Científico de La Escuela Superior Ciudad Sahagún*, *6*(11). https://doi.org/10.29057/ess.v6i11.3750

Guerra Álvarez, N., & Crawford Labrín, B. (2010). Optimización de funciones a través de Optimización por Enjambre de Partículas y Algoritmos Genéticos. *Proceedings of 32a Conferencia Latinoamericana de Informática*.

Lange, K. (2013). Optimization. In *Natural Computing Series* (Vol. 95). Springer New York. https://doi.org/10.1007/978-1-4614-5838-8

Martí, R. (2001). Procedimientos Metaheurísticos en Optimización Combinatoria. *Departament d'Estadística i Investigació Operativa*.

Navas, M. M., & Urbaneja, A. J. N. (2013). Metaheurísticas multiobjetivo adaptativas. *Computacion y Sistemas*, *17*(1).

NCBI. (2022). *Genome List - NCBI*. Database Resources of the National Center for Biotechnology Information.

NCBI Resource Coordinators. (2016). *Genome List*. Database Resources of the National Center for Biotechnology Information. Nucleic Acids Research, 44(Database Issue), D7–D19. Http://Doi.Org/10.1093/Nar/Gkv1290.

Nieto-Torres, J. L., Dediego, M. L., Verdiá-Báguena, C., Jimenez-Guardeñ O, J. M., & Regla-Nava, J. A. (2014). Severe Acute Respiratory Syndrome Coronavirus Envelope Protein Ion Channel Activity Promotes Virus Fitness and Pathogenesis. *PLoS Pathog*, *10*(5), 1004077. https://doi.org/10.1371/journal.ppat.1004077

Pascual-Iglesias, A., Canton, J., Ortega-Prieto, A. M., Jimenez-Guardeño, J. M., & Regla-Nava, J. A. (2021). *An Overview of Vaccines against SARS-CoV-2 in the COVID-19 Pandemic Era*. https://doi.org/10.3390/pathogens10081030

Pelikan, M., Goldberg, D. E., & Cantú-Paz, E. (1999). *BOA: The Bayesian Optimization Algorithm*.

Peña, R. P. (2022). Introducción a los modelos de optimización. In *Introducción a los modelos de optimización*. https://doi.org/10.2307/j.ctv2cw0t8k

Peralta-Abarca, J. del C. (2018). Metaheurísticas. *Inventio*, *14*(34). https://doi.org/10.30973/inventio/2018.14.34/3

Pessoa Ferreira Lima, T. de. (2013). *AN AUTOMATIC METHOD FOR CONSTRUCTION OF MULTICLASSIFIER SYSTEMS BASED ON THE COMBINATION OF SELECTION AND FUSION*. https://repositorio.ufpe.br/handle/123456789/12457

Ponce de León Sentí, E. E., Reyes Gallegos, J. E., Cuéllar Garrido, L. D., Martin Álvarez Tostado, E. M., Díaz Díaz, E., Torres Soto, A., Torres Soto, M. D., & Martínez

Guerra, J. J. (2022). Metodología eficiente para obtener cliques de proteínas mediante los mejores aciertos bidireccionales+. In 1st (Ed.), *UN ACERCAMIENTO A LA INVESTIGACIÓN MULTIDISCIPLINAR EN LA UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES* (Vol. 1, pp. 73–82). Universidad Autónoma de Aguascalientes. https://doi.org/10.33064/UAA/978-607-8782-60-4

Ponce-de-Leon-Senti, E., Diaz, E., Guardado-Muro, H., Cuellar-Garrido, D., Martinez-Guerra, J. J., Torres-Soto, A., Torres-Soto, D., & Hernandez-Aguirre, A. (2017). A Distance Measure for Building Phylogenetic Trees: A First Approach. In *Research in Computing Science* (Vol. 139, Issue 1). https://doi.org/10.13053/rcs-139-1-12

Ponce-de-Leon-Senti, E. E., Reyes-Gallegos, J. E., Cuellar-Garrido, L. D., Martin, E. M., Díaz Díaz, E., Torres-Soto, A., Torres-Soto, M. D., & Martinez-Guerra, J. J. (2020). METODOLOGÍA EFICIENTE PARA OBTENER CLIQUES DE PROTEÍNAS MEDIANTE LOS MEJORES ACIERTOS BIDIRECCIONALES. *Memorias Del 21 Seminario de Investigación de La Universidad Autónoma de Aguascalientes*, *in-Press*, 13.

Portillo-Lara, H. J., Ávila-Sandoval, M. S., Cruz-Quiñones, M. de los Á., & López-Ruvalcaba, C. (2019). Geogebra y Problemas de Optimización. *Cultura Científica y Tecnológica*, *16*(1). https://doi.org/10.20983/culcyt.2019.1.2.1

Roldán Martínez, D. (2015). *Bioinformática : el ADN a un solo clic*. Ra-Ma.

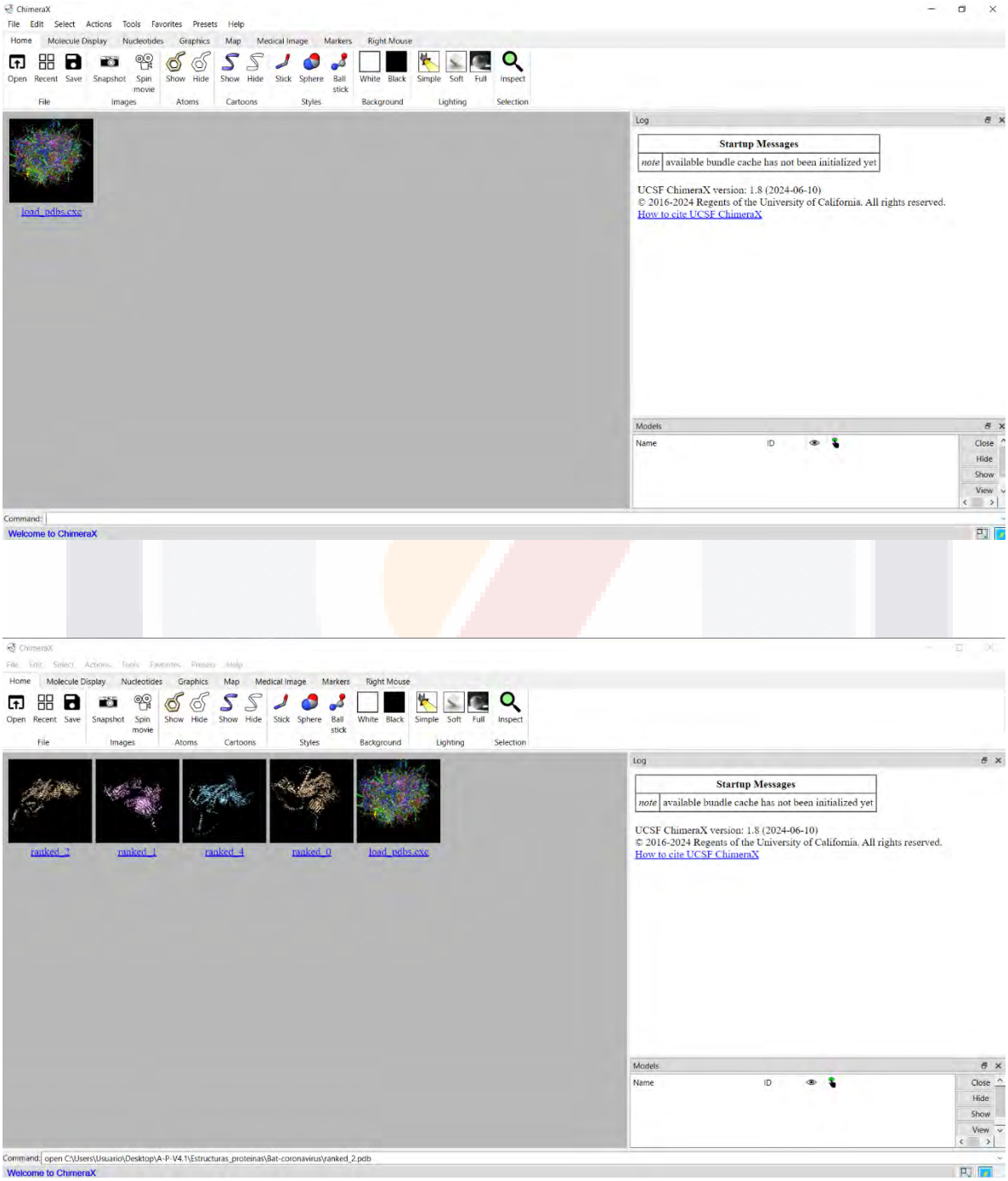
Torres-Jiménez, J., & Pavón, J. (2014). Applications of metaheuristics in real-life problems. *Prog Artif Intell*, *2*, 175–176. https://doi.org/10.1007/s13748-014-0051-8

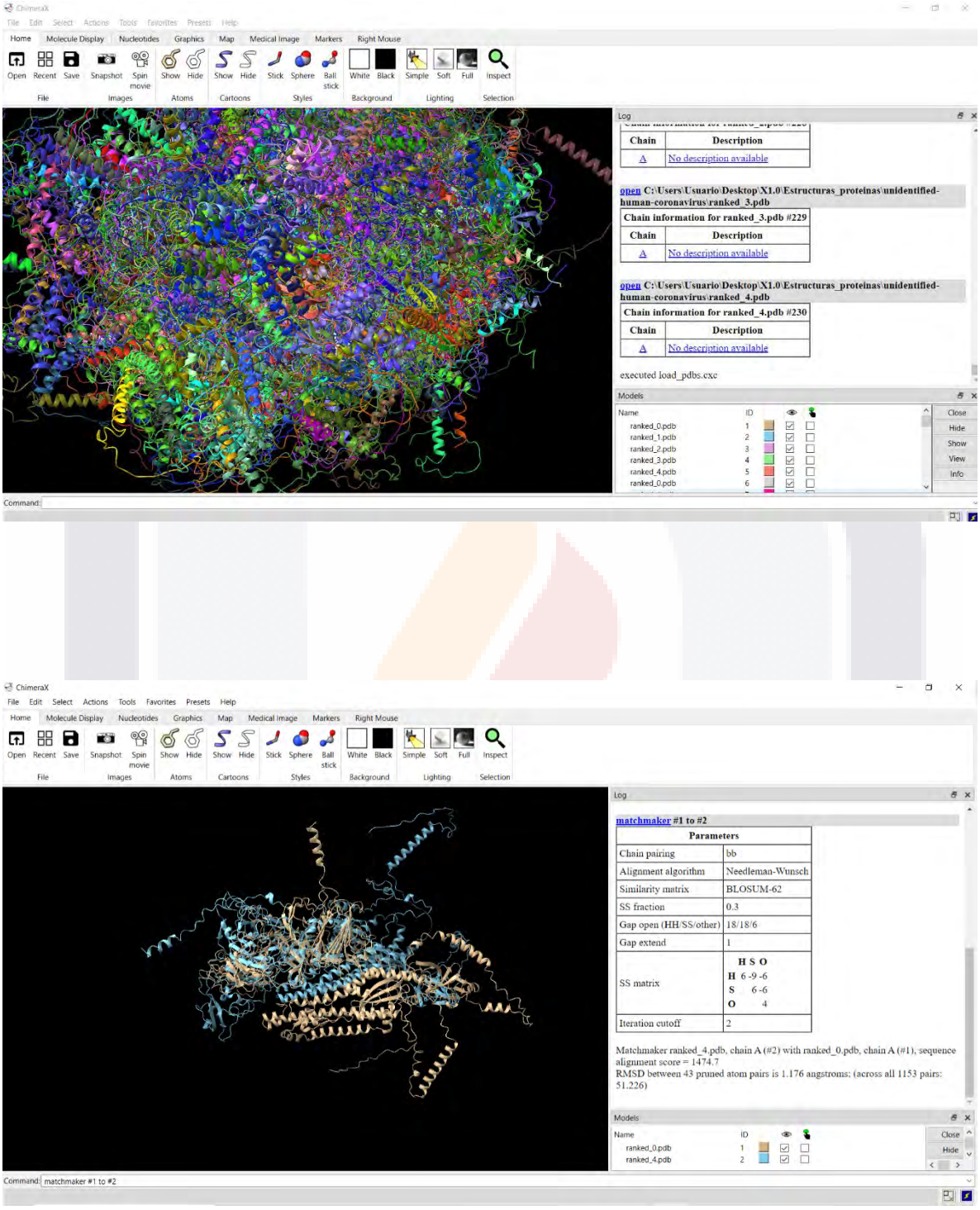
Veiga, A. (2019). Western Blot: Introducción y Optimización. *Abcam*.

Wang, J. P., & Tong, Q. (2009). Urban planning decision using multi-objective optimization algorithm. *2009 Second ISECS International Colloquium on Computing, Communication, Control, and Management*, *4*, 392–394. https://doi.org/10.1109/CCCM.2009.5267600

Anexos

Gropo 1: Software ChimeraX





Grupo 2: Jupyter Notebook

Automatización del proceso de cálculo de distancias entre proteínas Spike de organismos de la familia coronaviridae

Presentación:

Este notebook tiene como objetivo automatizar el proceso de carga y emparejamiento de archivos de estructuras de proteínas en el software ChimeraX. Para este fin, se generarán scripts .cxc que permitirán comparar todas las estructuras de proteínas utilizando el comando matchmaker de ChimeraX.

En el proceso, se siguen los siguientes pasos:

1. Cargar y listar los archivos de proteínas disponibles.
2. Generar un script .cxc que cargue todos los archivos en ChimeraX.
3. Crear scripts .cxc individuales para realizar comparaciones sin repeticiones de emparejamientos entre las estructuras.
4. Generar scripts `.cxc` para realizar comparaciones sin repeticiones
5. Ejecutar los scripts generados para realizar las comparaciones
6. Guardar un único archivo de log después de todas las comparaciones
7. Crear un DataFrame con la información obtenida
8. Predecir el género utilizando `Alignment Score`, `RMSD` y `Átomos Pareados`
9. Normalización de atributos y cálculo de la Distancia Normalizada Ponderada (DNP)
10. Implementación del Algoritmo Genético para Clasificación de Proteínas

Paso 1: Cargar y listar todos los archivos `.pdb` disponibles

Este paso tiene como objetivo identificar y listar todas las estructuras de proteínas en formato `.pdb` almacenadas en subdirectorios dentro de la carpeta principal `Estructuras_proteinas`. Para lograr esto, el script:

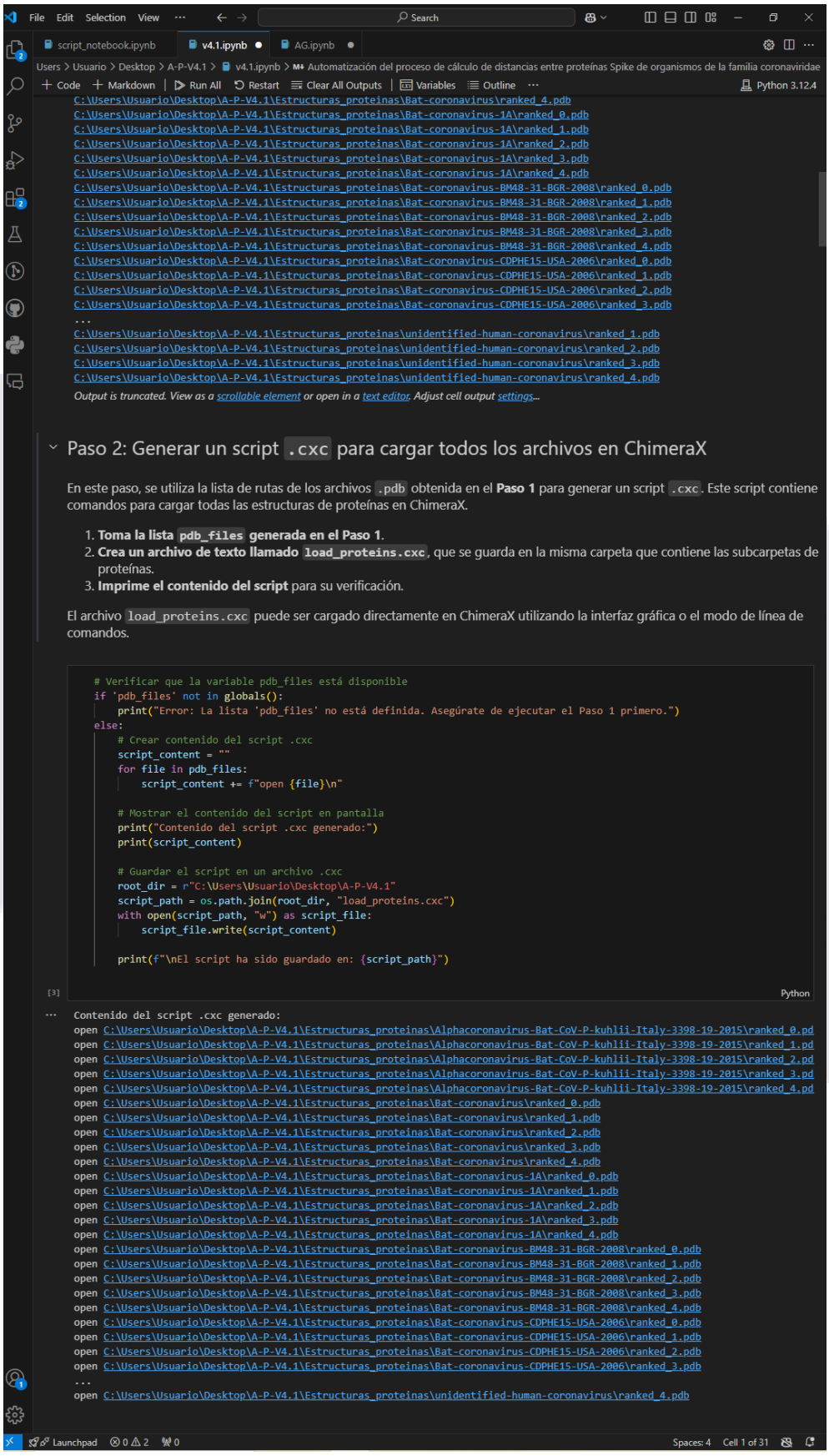
1. **Define la ruta raíz:** Se especifica la ubicación de la carpeta que contiene las proteínas.
2. **Explora subcarpetas:** Se verifica cada subdirectorio dentro de la carpeta principal.
3. **Busca archivos `.pdb`:** En cada subcarpeta, identifica los archivos con la extensión `.pdb`.
4. **Genera una lista completa:** Todas las rutas de los archivos `.pdb` encontrados se almacenan en una lista.
5. **Muestra el resultado:** Si se encuentran archivos, imprime las rutas completas de cada archivo. Si no se encuentran, muestra un mensaje de error indicando el problema.

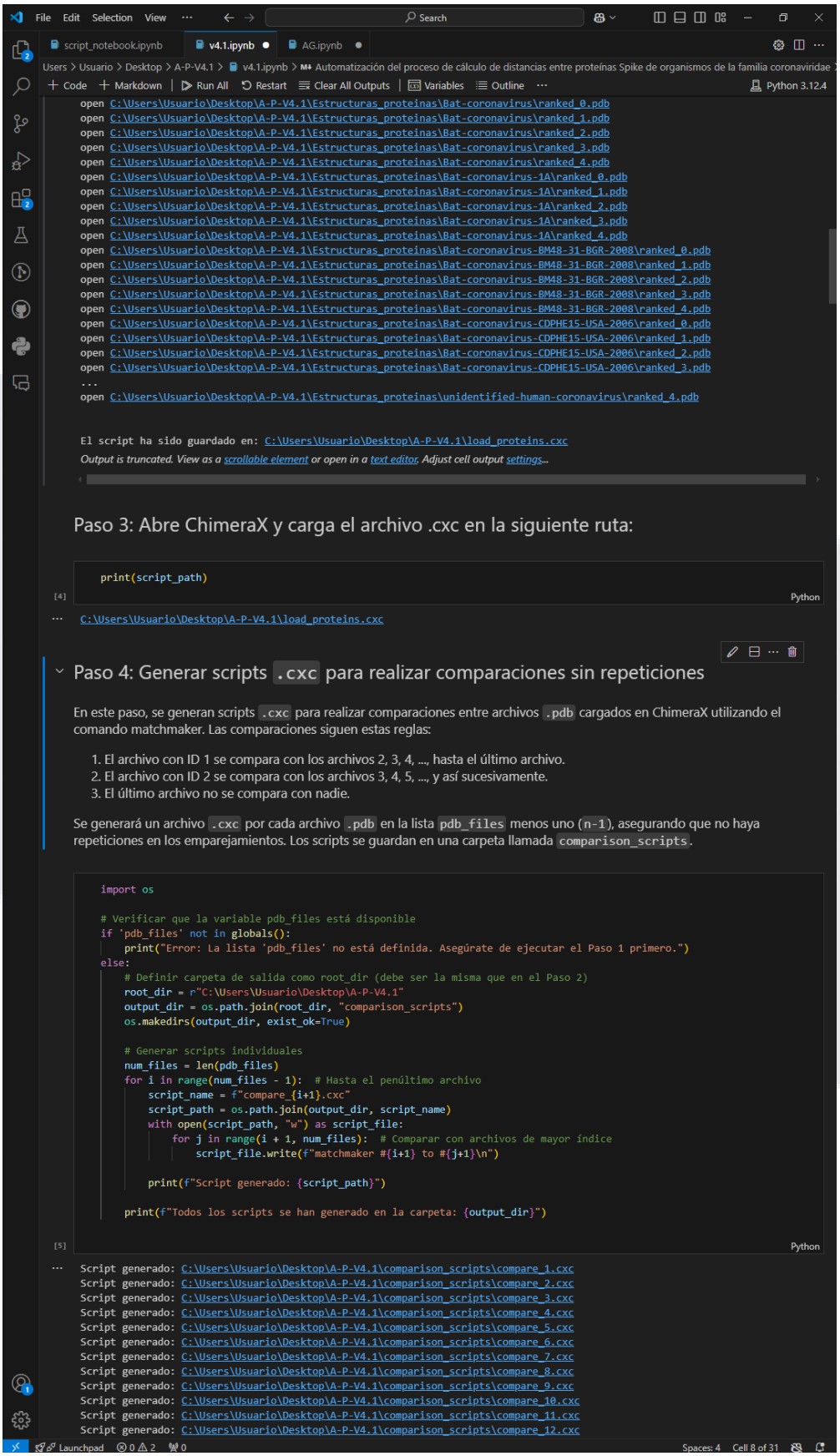
Este listado es fundamental para los pasos siguientes, ya que permite cargar y procesar las estructuras de proteínas de manera automatizada.

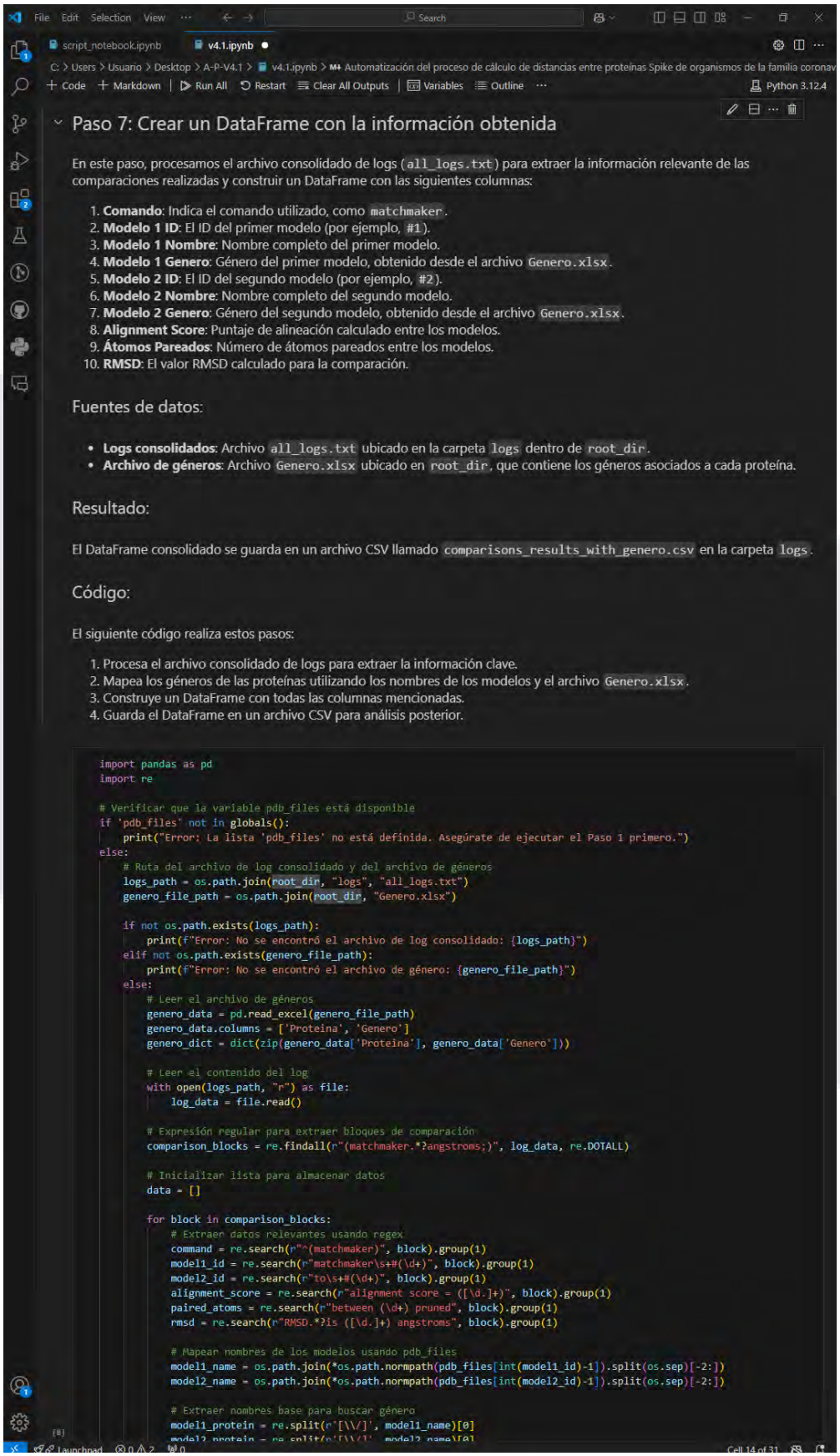
Nota: Si la carpeta principal no existe o no contiene archivos `.pdb`, el código emitirá un mensaje de error para ayudar a depurar el problema.

```python
import os

# Ruta de la carpeta raíz
root_dir = r"C:\Users\Usuario\Desktop\A-P-V4.1\Estructuras_proteinas"

# Lista para almacenar las rutas de los archivos .pdb encontrados
pdb_files = []

# Verificar si la carpeta raíz existe
if not os.path.exists(root_dir):
    print(f"Error: La carpeta '{root_dir}' no existe.")
else:
    # Buscar archivos .pdb en las subcarpetas
    for folder in os.listdir(root_dir):
        folder_path = os.path.join(root_dir, folder)
        if os.path.isdir(folder_path):
            pdb_files.extend([os.path.join(folder_path, f) for f in os.listdir(folder_path) if f.endswith(".pdb")])

    # Verificar si se encontraron archivos .pdb
    if not pdb_files:
        print("Error: No se encontraron archivos .pdb en las subcarpetas.")
    else:
        # Mostrar la lista completa de archivos .pdb encontrados
        print("Archivos .pdb encontrados:")
        for file in pdb_files:
            print(file)
```

```
Archivos .pdb encontrados:
C:\Users\Usuario\Desktop\A-P-V4.1\Estructuras_proteinas\Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-19-2015\ranked_0.pdb
C:\Users\Usuario\Desktop\A-P-V4.1\Estructuras_proteinas\Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-19-2015\ranked_1.pdb
C:\Users\Usuario\Desktop\A-P-V4.1\Estructuras_proteinas\Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-19-2015\ranked_2.pdb
C:\Users\Usuario\Desktop\A-P-V4.1\Estructuras_proteinas\Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-19-2015\ranked_3.pdb
C:\Users\Usuario\Desktop\A-P-V4.1\Estructuras_proteinas\Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-19-2015\ranked_4.pdb
C:\Users\Usuario\Desktop\A-P-V4.1\Estructuras_proteinas\Bat-coronavirus\ranked_0.pdb
```



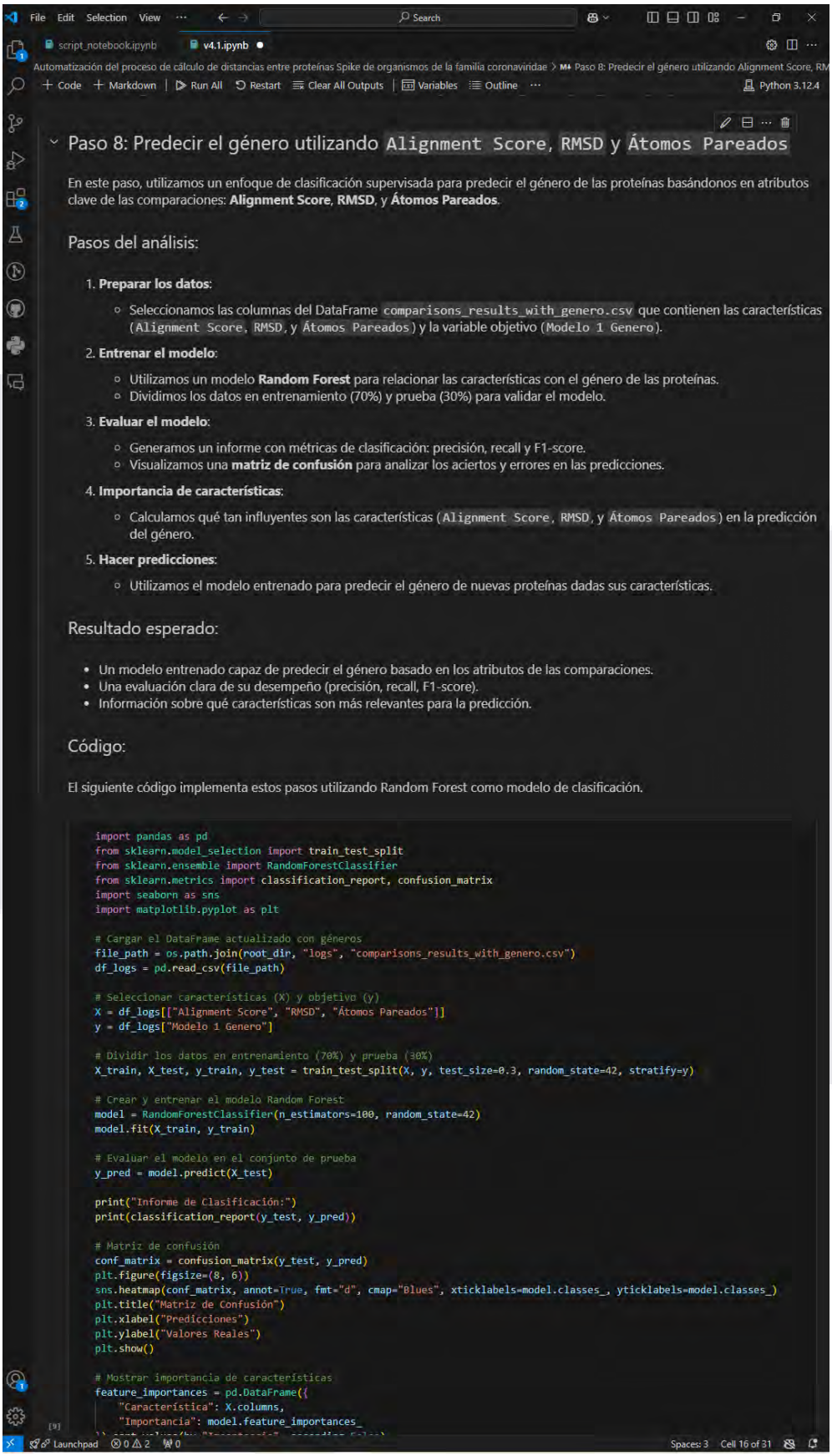




```
                paired_atoms = re.search(r"between (\d+) pruned", block).group(1)
                rmsd = re.search(r"RMSD.*?is ([\d.]+) angstroms", block).group(1)

                # Mapear nombres de los modelos usando pdb_files
                model1_name = os.path.join(*os.path.normpath(pdb_files[int(model1_id)-1]).split(os.sep)[-2:])
                model2_name = os.path.join(*os.path.normpath(pdb_files[int(model2_id)-1]).split(os.sep)[-2:])

                # Extraer nombres base para buscar género
                model1_protein = re.split(r'[\\/]', model1_name)[0]
                model2_protein = re.split(r'[\\/]', model2_name)[0]

                # Extraer ranking del modelo
                model1_rank = re.split(r'[\\/]', model1_name)[1]
                model2_rank = re.split(r'[\\/]', model2_name)[1]

                # Obtener género desde el archivo de género
                model1_genero = genero_dict.get(model1_protein, "Desconocido")
                model2_genero = genero_dict.get(model2_protein, "Desconocido")

                # Agregar fila a los datos
                data.append([command, f"#{model1_id}", model1_protein, model1_rank, model1_genero,
                             f"#{model2_id}", model2_protein, model2_rank, model2_genero,
                             float(alignment_score), int(paired_atoms), float(rmsd)])

        # Crear el DataFrame
        columns = ["Comando", "Modelo 1 ID", "Modelo 1 Proteina", "Modelo 1 Ranking", "Modelo 1 Genero",
                   "Modelo 2 ID", "Modelo 2 Proteina", "Modelo 2 Ranking", "Modelo 2 Genero",
                   "Alignment Score", "Átomos Pareados", "RMSD"]
        df_logs = pd.DataFrame(data, columns=columns)

        # Guardar el DataFrame en una variable global
        global df_logs_global
        df_logs_global = df_logs

        # Mostrar el DataFrame
        from IPython.display import display
        display(df_logs)

        # Guardar el DataFrame en un archivo CSV (opcional)
        output_csv_path = os.path.join(root_dir, "logs", "comparisons_results_with_protein_and_rank.csv")
        df_logs.to_csv(output_csv_path, index=False)
        print(f"Resultados guardados en: {output_csv_path}")
```

	Modelo 1 Proteina	Modelo 1 Ranking	Modelo 1 Genero	Modelo 2 ID	Modelo 2 Proteina	Modelo 2 Ranking	Modelo 2 Genero	Alignment Score	Átomos Pareados	RMSD
	lphacoronavirus-3at-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#2	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_1.pdb	Alphacoronavirus	7153.5	1222	0.667
	lphacoronavirus-3at-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#3	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_2.pdb	Alphacoronavirus	6937.5	442	1.258
	lphacoronavirus-3at-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#4	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_3.pdb	Alphacoronavirus	6871.2	500	1.022
	lphacoronavirus-3at-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#5	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_4.pdb	Alphacoronavirus	6872.1	491	0.961
	lphacoronavirus-3at-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#6	Bat-coronavirus	ranked_0.pdb	Indefinido	1523.5	237	1.035
	...	...	...	...	...	...	...	...	...	...
	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	6603.6	364	0.972
	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6627.0	275	0.728
	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	6677.1	554	1.252
	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6753.3	543	1.111
	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinido	6860.1	535	1.119

```
Resultados guardados en: C:\Users\Usuario\Desktop\A-P-V4.1\logs\comparisons_results_with_protein_and_rank.csv
```



File Edit Selection View ··· Search

script_notebook.ipynb v4.1.ipynb ●

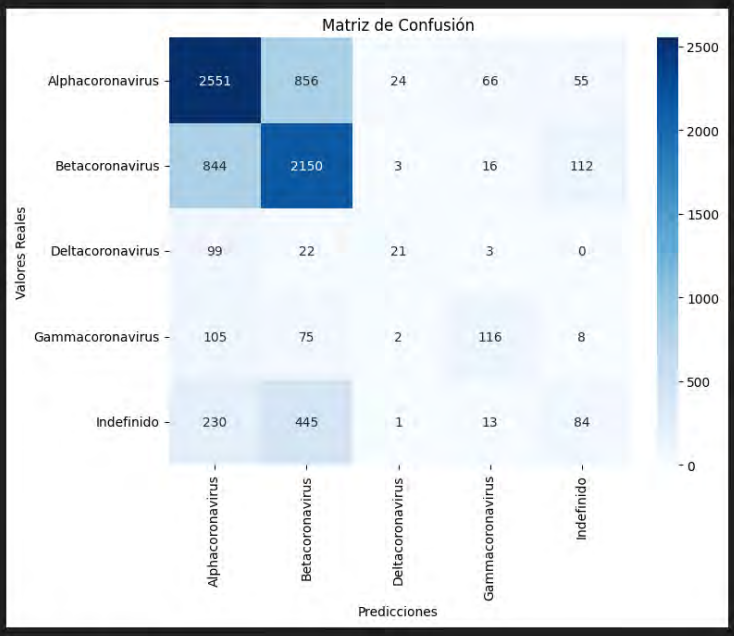
Automatización del proceso de cálculo de distancias entre proteínas Spike de organismos de la familia coronaviridae > M↓ Paso 8: Predecir el género utilizando Alignment Score, RM

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· Python 3.12.4

```
print("Informe de Clasificación:")
print(classification_report(y_test, y_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=model.classes_, yticklabels=model.classes_)
plt.title("Matriz de Confusión")
plt.xlabel("Predicciones")
plt.ylabel("Valores Reales")
plt.show()

# Mostrar importancia de características
feature_importances = pd.DataFrame({
    "Característica": X.columns,
    "Importancia": model.feature_importances_
}).sort_values(by="Importancia", ascending=False)

print("Importancia de Características:")
print(feature_importances)

# Opcional: Hacer predicciones en nuevos datos
new_data = pd.DataFrame({
    "Alignment Score": [7000],
    "RMSD": [0.5],
    "Átomos Pareados": [1200]
})
predicted_genero = model.predict(new_data)
print(f"Predicción para los nuevos datos: {predicted_genero[0]}")
```

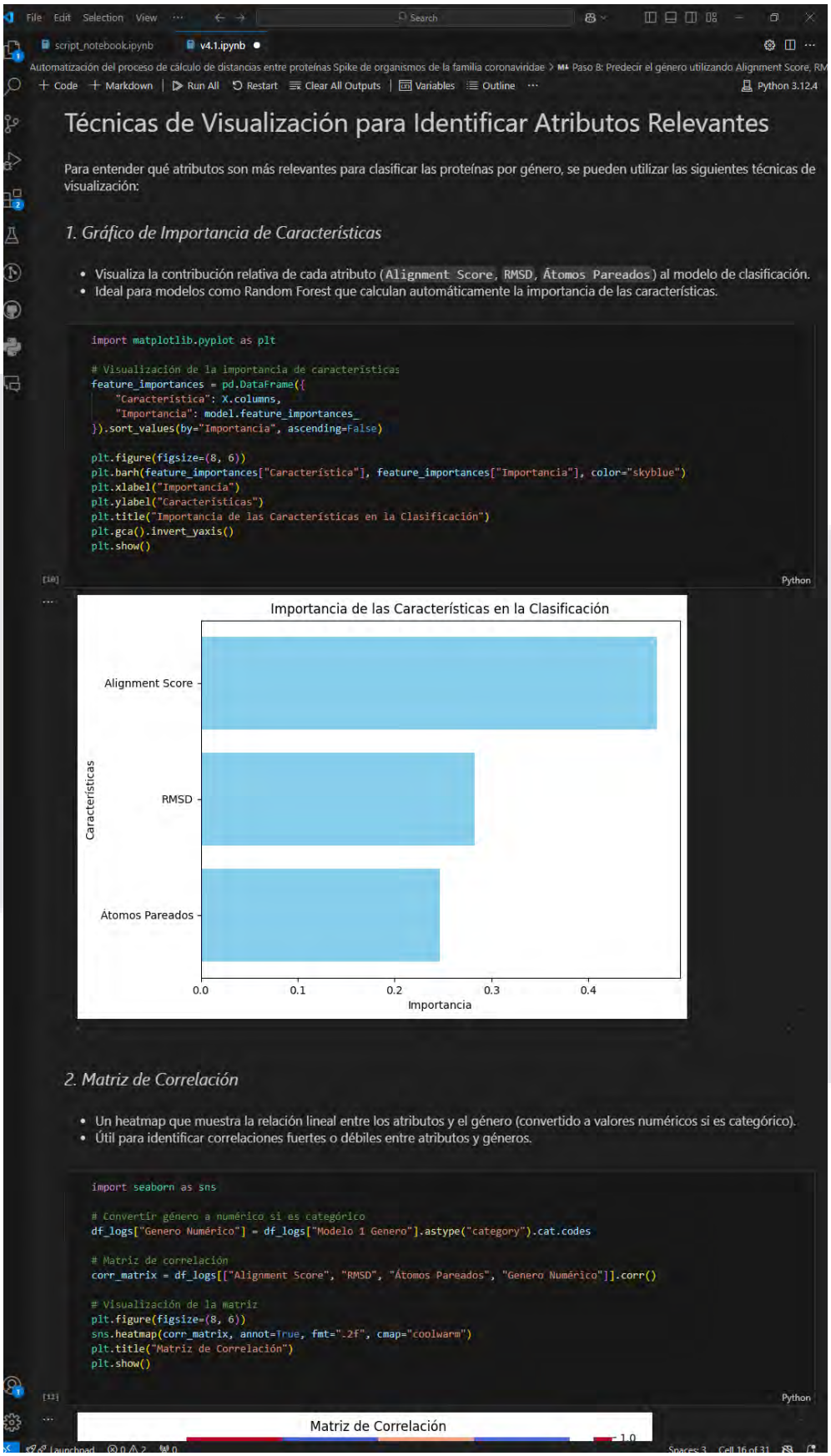
[9] Python

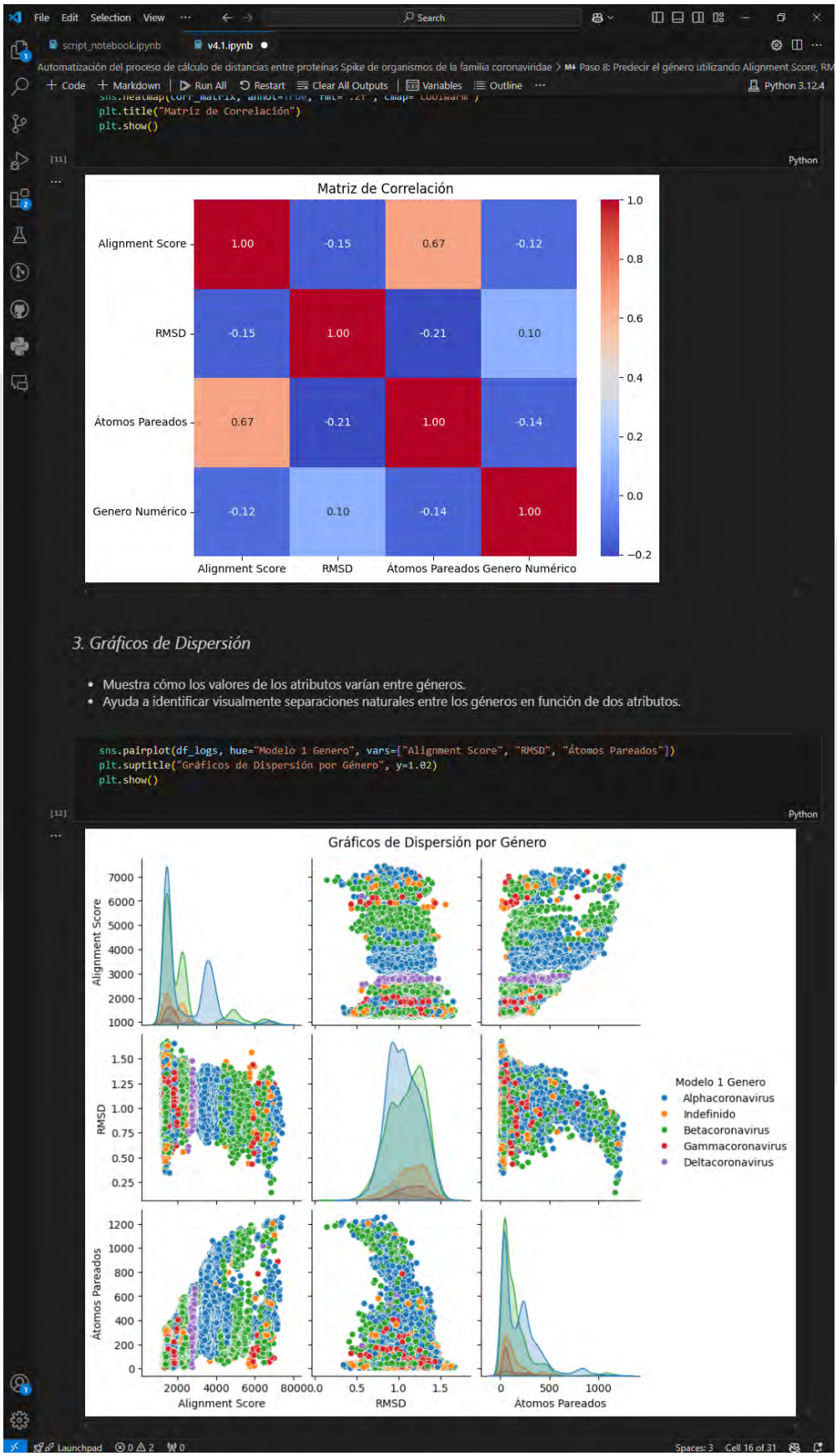
```
Informe de Clasificación:
                   precision    recall  f1-score   support

  Alphacoronavirus       0.67      0.72      0.69      3552
   Betacoronavirus       0.61      0.69      0.64      3125
  Deltacoronavirus       0.41      0.14      0.21       145
  Gammacoronavirus       0.54      0.38      0.45       306
       Indefinido        0.32      0.11      0.16       773

          accuracy                           0.62      7901
         macro avg       0.51      0.41      0.43      7901
      weighted avg       0.60      0.62      0.60      7901
```

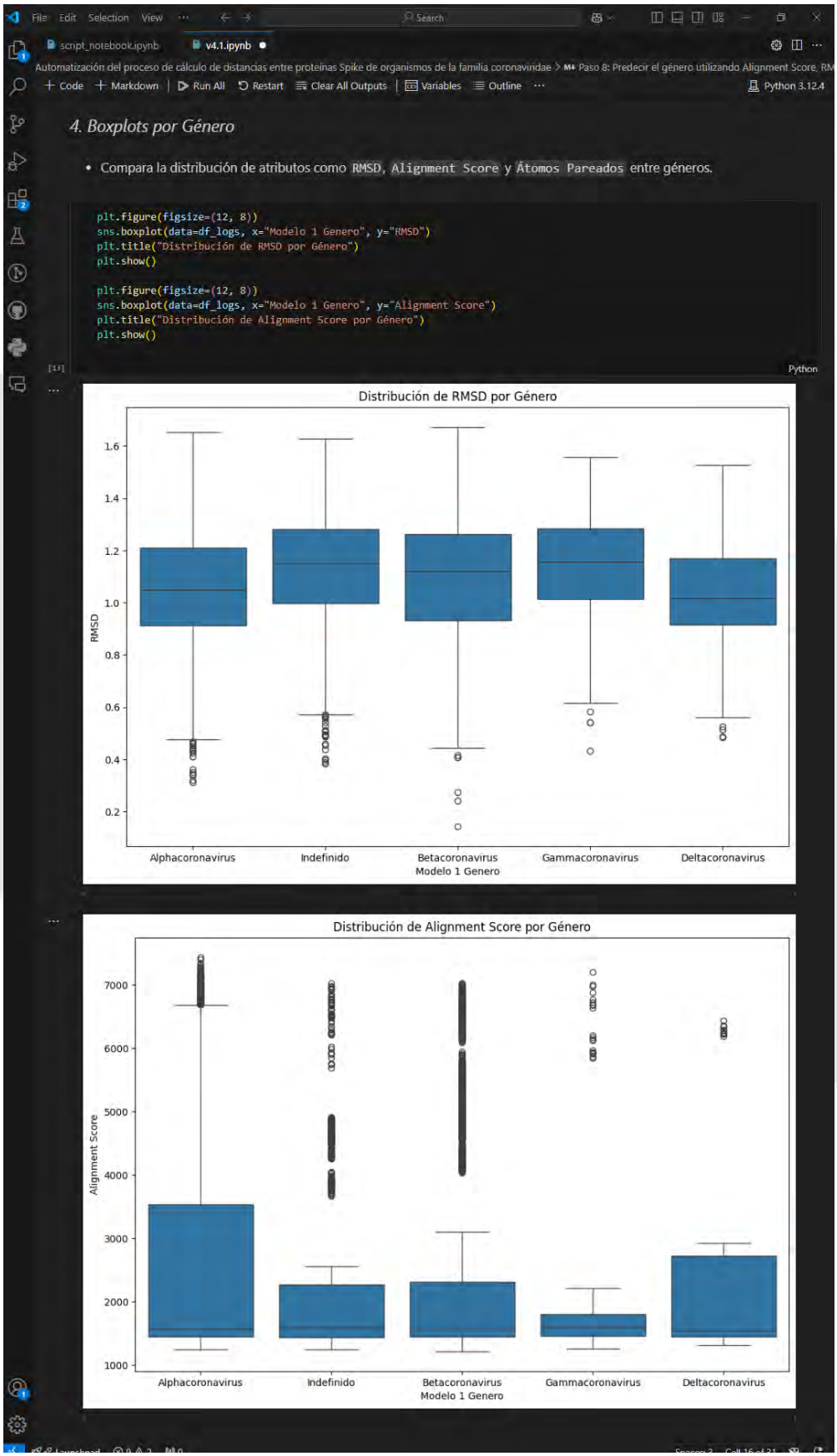


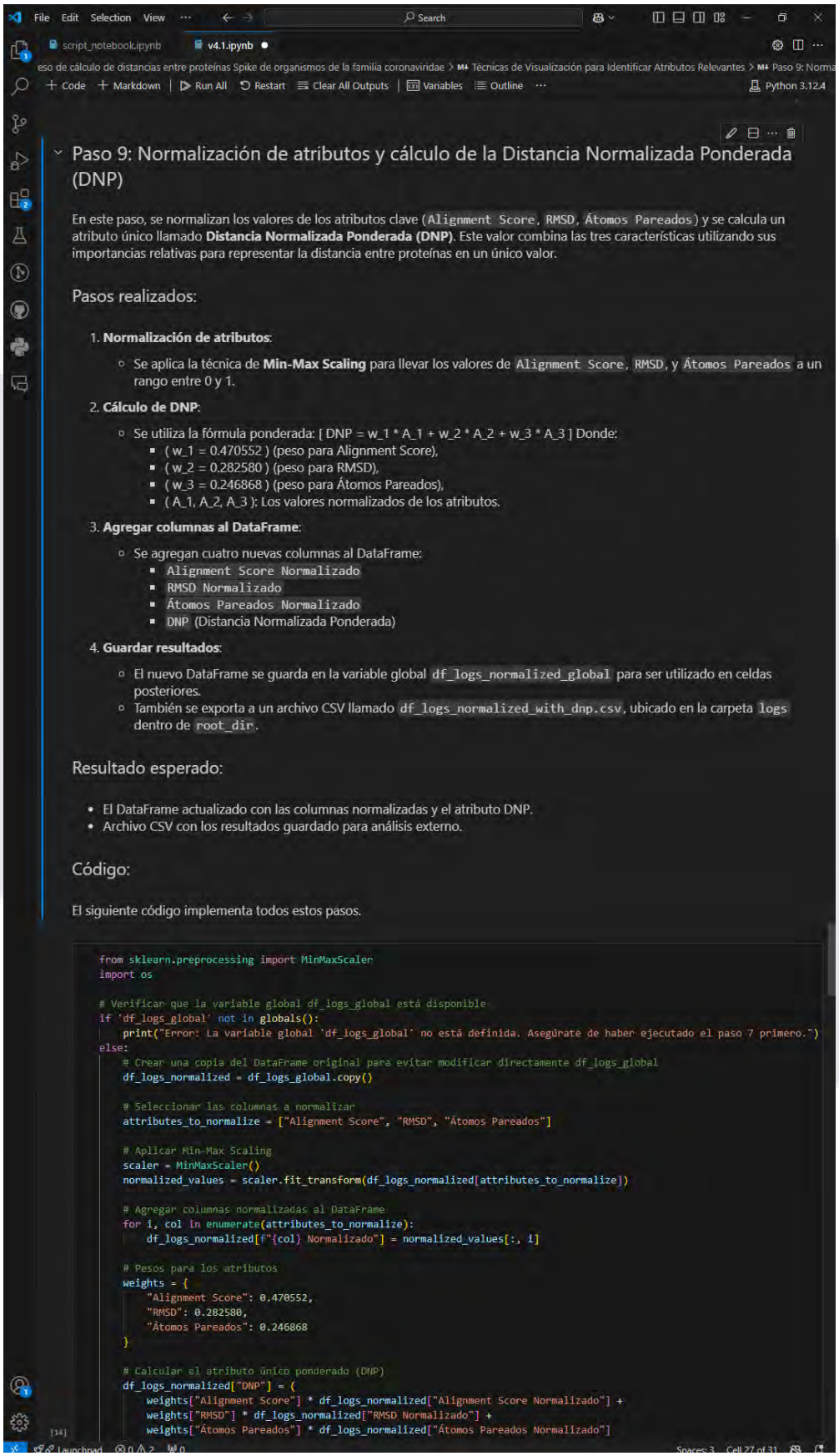

```
Importancia de Características:
       Característica  Importancia
0  Alignment Score     0.470552
1            RMSD      0.282580
2  Átomos Pareados     0.246868
Predicción para los nuevos datos: Alphacoronavirus
```

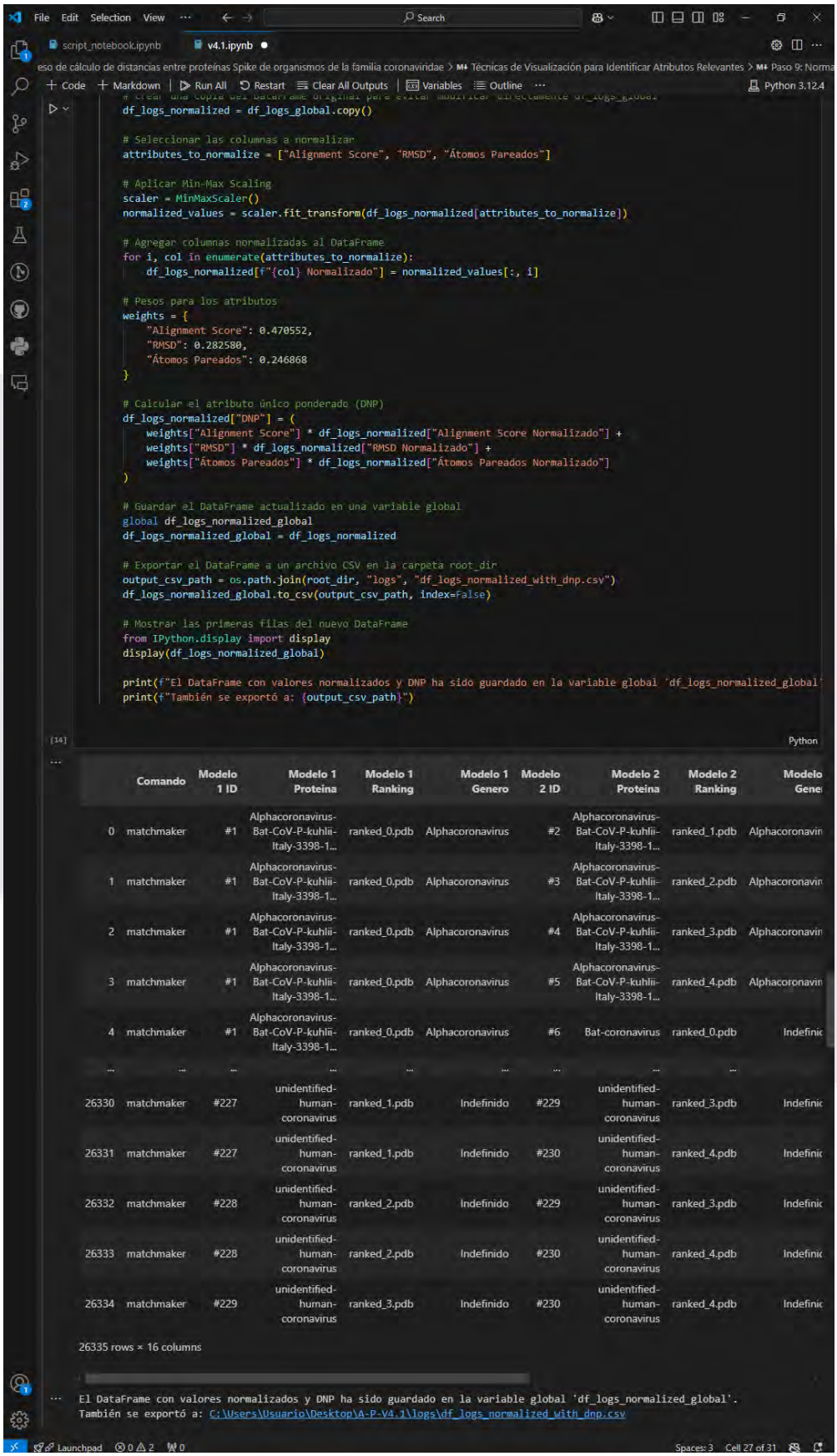
Launchpad ⊗ 0 ⚠ 2 0 Spaces: 3 Cell 16 of 31











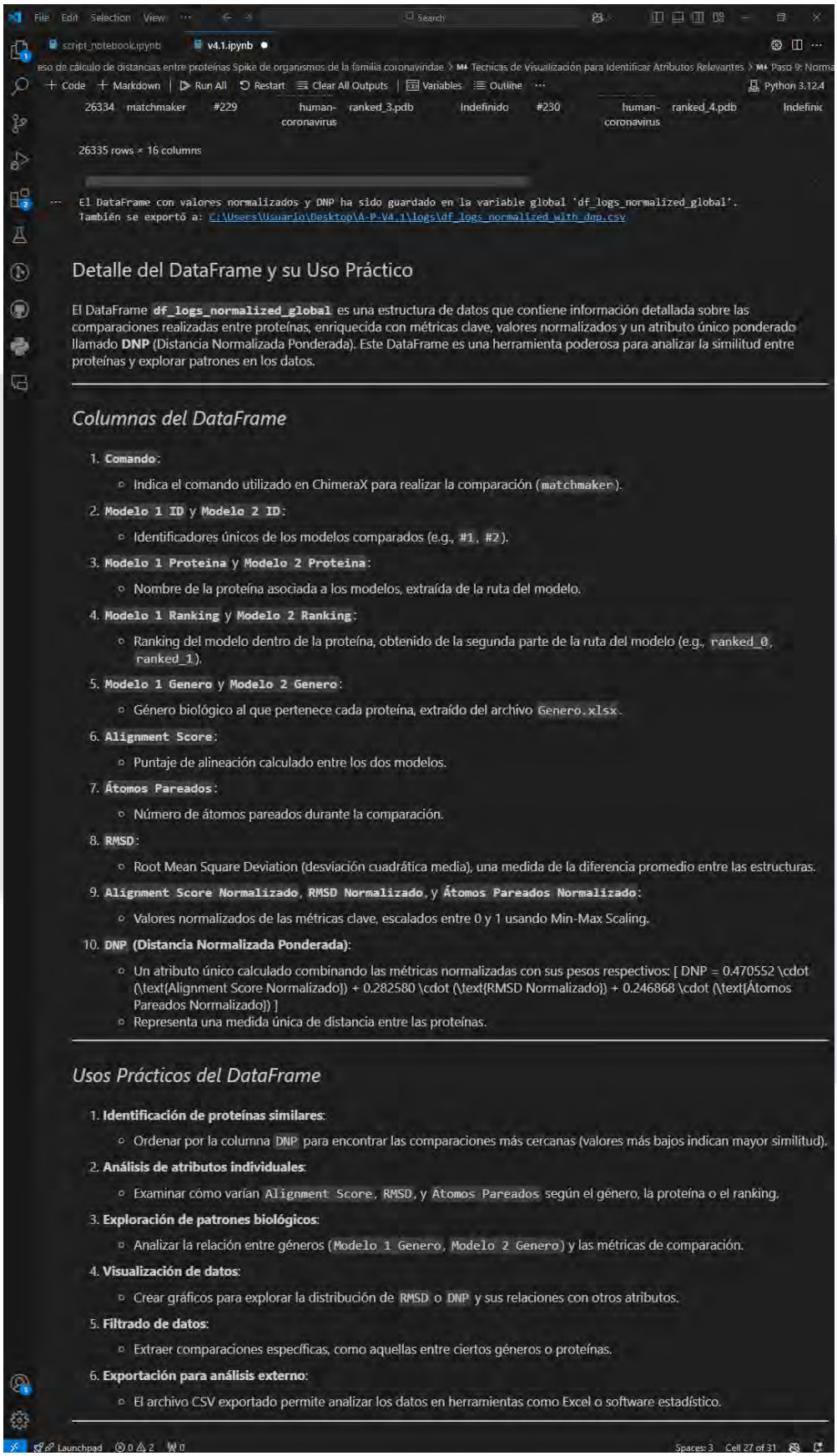
```python
df_logs_normalized = df_logs_global.copy()

# Seleccionar las columnas a normalizar
attributes_to_normalize = ["Alignment Score", "RMSD", "Átomos Pareados"]

# Aplicar Min-Max Scaling
scaler = MinMaxScaler()
normalized_values = scaler.fit_transform(df_logs_normalized[attributes_to_normalize])

# Agregar columnas normalizadas al DataFrame
for i, col in enumerate(attributes_to_normalize):
    df_logs_normalized[f"{col} Normalizado"] = normalized_values[:, i]

# Pesos para los atributos
weights = {
    "Alignment Score": 0.470552,
    "RMSD": 0.282580,
    "Átomos Pareados": 0.246868
}

# Calcular el atributo único ponderado (DNP)
df_logs_normalized["DNP"] = (
    weights["Alignment Score"] * df_logs_normalized["Alignment Score Normalizado"] +
    weights["RMSD"] * df_logs_normalized["RMSD Normalizado"] +
    weights["Átomos Pareados"] * df_logs_normalized["Átomos Pareados Normalizado"]
)

# Guardar el DataFrame actualizado en una variable global
global df_logs_normalized_global
df_logs_normalized_global = df_logs_normalized

# Exportar el DataFrame a un archivo CSV en la carpeta root_dir
output_csv_path = os.path.join(root_dir, "logs", "df_logs_normalized_with_dnp.csv")
df_logs_normalized_global.to_csv(output_csv_path, index=False)

# Mostrar las primeras filas del nuevo DataFrame
from IPython.display import display
display(df_logs_normalized_global)

print(f"El DataFrame con valores normalizados y DNP ha sido guardado en la variable global 'df_logs_normalized_global'")
print(f"También se exportó a: {output_csv_path}")
```

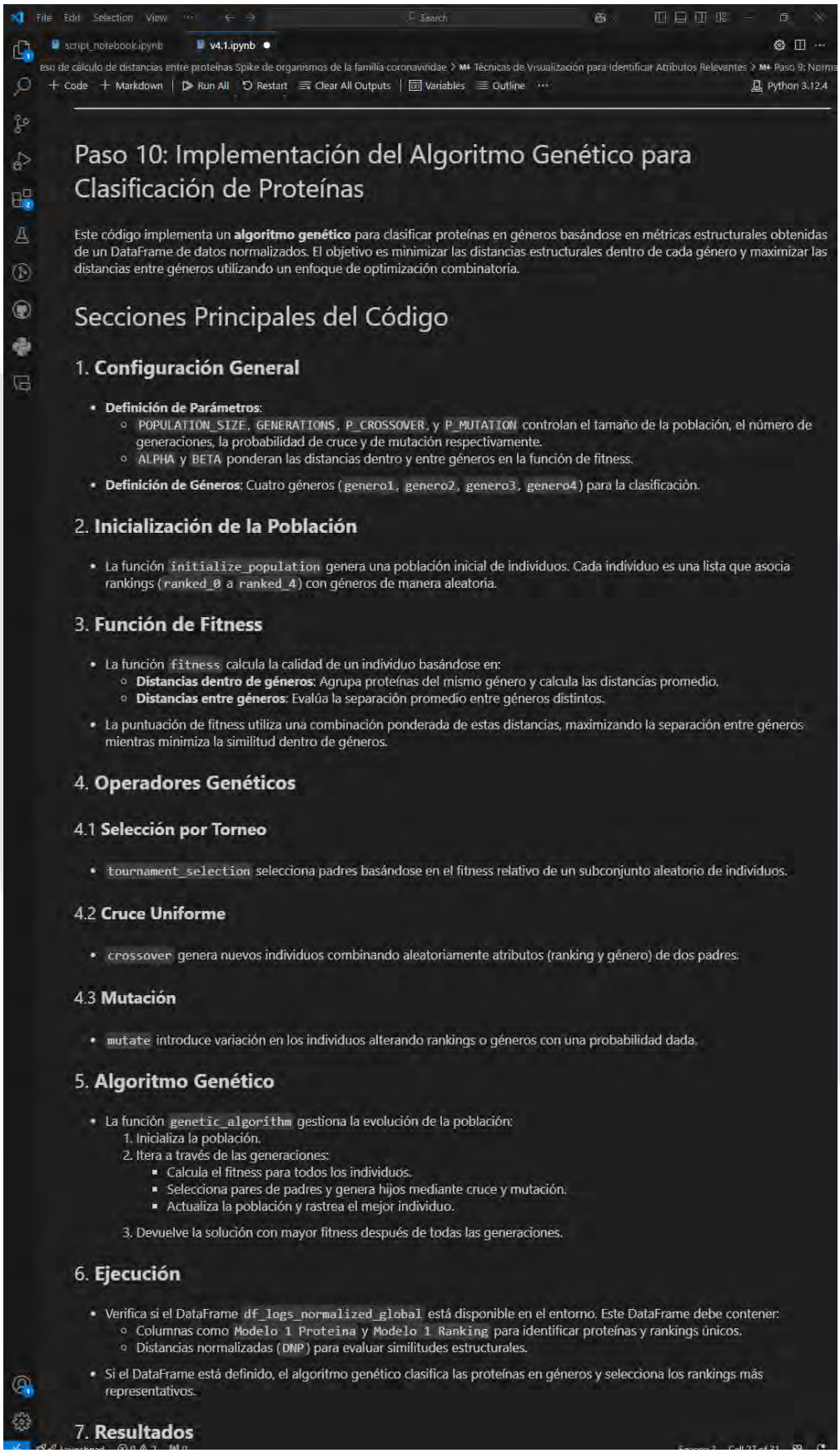
[14] Python

	Comando	Modelo 1 ID	Modelo 1 Proteina	Modelo 1 Ranking	Modelo 1 Genero	Modelo 2 ID	Modelo 2 Proteina	Modelo 2 Ranking	Modelo Gene
0	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#2	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_1.pdb	Alphacoronavir
1	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#3	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_2.pdb	Alphacoronavir
2	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#4	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_3.pdb	Alphacoronavir
3	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#5	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_4.pdb	Alphacoronavir
4	matchmaker	#1	Alphacoronavirus-Bat-CoV-P-kuhlii-Italy-3398-1...	ranked_0.pdb	Alphacoronavirus	#6	Bat-coronavirus	ranked_0.pdb	Indefinic
...	...	...	...	...	...	...	...	...	
26330	matchmaker	#227	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinic
26331	matchmaker	#227	unidentified-human-coronavirus	ranked_1.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinic
26332	matchmaker	#228	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinic
26333	matchmaker	#228	unidentified-human-coronavirus	ranked_2.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinic
26334	matchmaker	#229	unidentified-human-coronavirus	ranked_3.pdb	Indefinido	#230	unidentified-human-coronavirus	ranked_4.pdb	Indefinic

26335 rows × 16 columns

```
El DataFrame con valores normalizados y DNP ha sido guardado en la variable global 'df_logs_normalized_global'.
También se exportó a: C:\Users\Usuario\Desktop\A-P-V4.1\logs\df_logs_normalized_with_dnp.csv
```





Paso 10: Implementación del Algoritmo Genético para Clasificación de Proteínas

Este código implementa un **algoritmo genético** para clasificar proteínas en géneros basándose en métricas estructurales obtenidas de un DataFrame de datos normalizados. El objetivo es minimizar las distancias estructurales dentro de cada género y maximizar las distancias entre géneros utilizando un enfoque de optimización combinatoria.

Secciones Principales del Código

1. **Configuración General**

- **Definición de Parámetros**:
  - `POPULATION_SIZE`, `GENERATIONS`, `P_CROSSOVER`, y `P_MUTATION` controlan el tamaño de la población, el número de generaciones, la probabilidad de cruce y de mutación respectivamente.
  - `ALPHA` y `BETA` ponderan las distancias dentro y entre géneros en la función de fitness.
- **Definición de Géneros**: Cuatro géneros (`genero1`, `genero2`, `genero3`, `genero4`) para la clasificación.

2. **Inicialización de la Población**

- La función `initialize_population` genera una población inicial de individuos. Cada individuo es una lista que asocia rankings (`ranked_0` a `ranked_4`) con géneros de manera aleatoria.

3. **Función de Fitness**

- La función `fitness` calcula la calidad de un individuo basándose en:
  - **Distancias dentro de géneros**: Agrupa proteínas del mismo género y calcula las distancias promedio.
  - **Distancias entre géneros**: Evalúa la separación promedio entre géneros distintos.
- La puntuación de fitness utiliza una combinación ponderada de estas distancias, maximizando la separación entre géneros mientras minimiza la similitud dentro de géneros.

4. **Operadores Genéticos**

4.1 **Selección por Torneo**

- `tournament_selection` selecciona padres basándose en el fitness relativo de un subconjunto aleatorio de individuos.

4.2 **Cruce Uniforme**

- `crossover` genera nuevos individuos combinando aleatoriamente atributos (ranking y género) de dos padres.

4.3 **Mutación**

- `mutate` introduce variación en los individuos alterando rankings o géneros con una probabilidad dada.

5. **Algoritmo Genético**

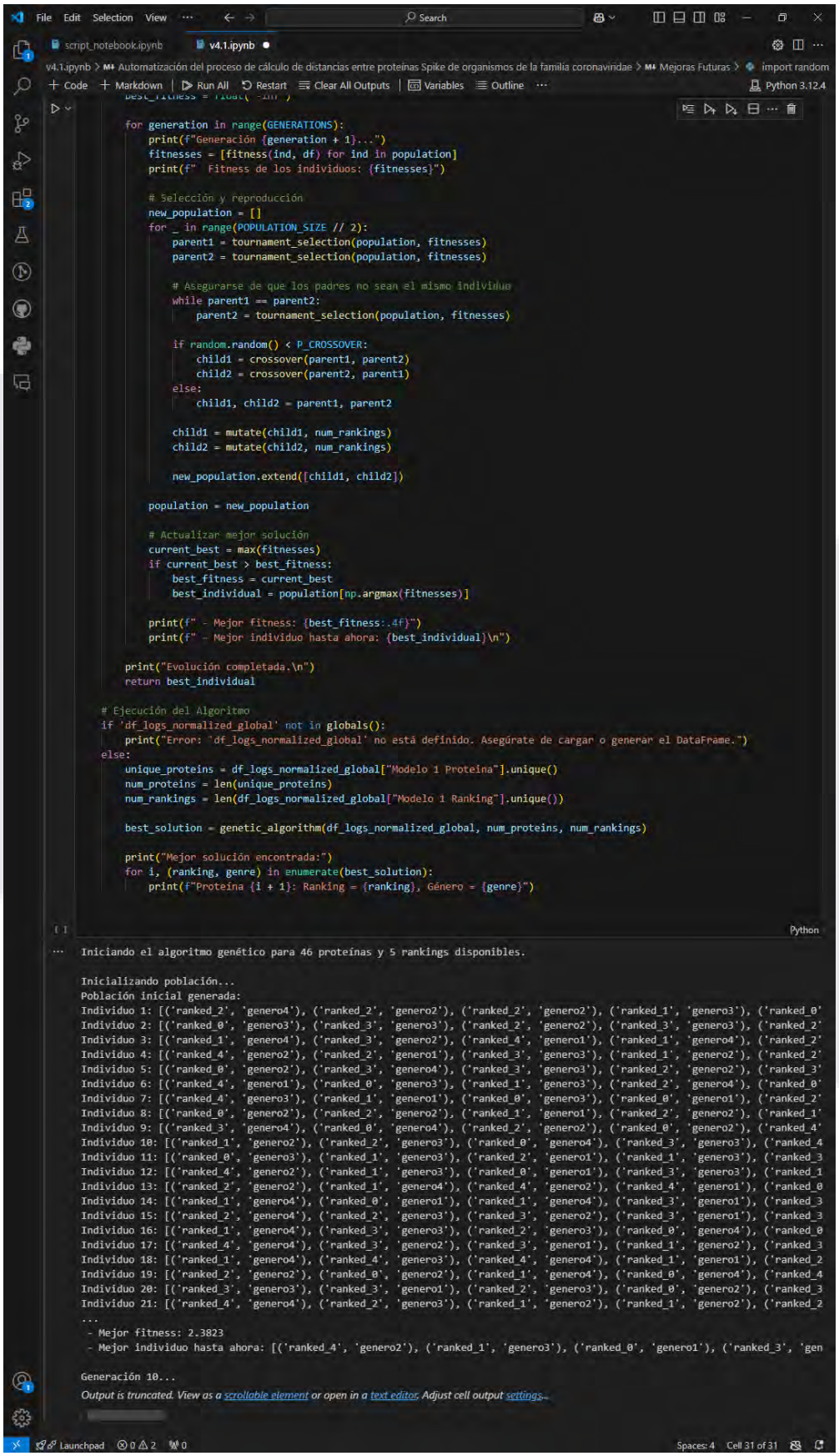
- La función `genetic_algorithm` gestiona la evolución de la población:
  1. Inicializa la población.
  2. Itera a través de las generaciones:
     - Calcula el fitness para todos los individuos.
     - Selecciona pares de padres y genera hijos mediante cruce y mutación.
     - Actualiza la población y rastrea el mejor individuo.
  3. Devuelve la solución con mayor fitness después de todas las generaciones.

6. **Ejecución**

- Verifica si el DataFrame `df_logs_normalized_global` está disponible en el entorno. Este DataFrame debe contener:
  - Columnas como `Modelo 1 Proteina` y `Modelo 1 Ranking` para identificar proteínas y rankings únicos.
  - Distancias normalizadas (`DNP`) para evaluar similitudes estructurales.
- Si el DataFrame está definido, el algoritmo genético clasifica las proteínas en géneros y selecciona los rankings más representativos.

7. **Resultados**

```
                        distances_within.append(df.loc[indices[i], "DNP"] + df.loc[indices[j], "DNP"])

        # Calcular distancias entre géneros
        genres = list(genre_groups.keys())
        for i in range(len(genres)):
            for j in range(i + 1, len(genres)):
                genre1 = genre_groups[genres[i]]
                genre2 = genre_groups[genres[j]]
                for g1 in genre1:
                    for g2 in genre2:
                        distances_between.append(df.loc[g1, "DNP"] + df.loc[g2, "DNP"])

        avg_within = np.mean(distances_within) if distances_within else float("inf")
        avg_between = np.mean(distances_between) if distances_between else 0

        # Fitness
        return ALPHA / avg_within + BETA * avg_between

    # Selección por torneo
    def tournament_selection(population, fitnesses, k=3):
        selected = random.sample(range(len(population)), k)
        best = min(selected, key=lambda i: fitnesses[i])
        return population[best]

    # Cruce uniforme
    def crossover(parent1, parent2):
        print(f"  Padres seleccionados para cruce:\n    Padre 1: {parent1}\n    Padre 2: {parent2}")
        child = []
        for p1, p2 in zip(parent1, parent2):
            if random.random() < 0.5:
                child.append(p1)
            else:
                child.append(p2)
        print(f"  Hijo generado: {child}")
        return child

    # Mutación
    def mutate(individual, num_rankings):
        print(f"  Antes de mutación: {individual}")
        for i in range(len(individual)):
            if random.random() < P_MUTATION:
                if random.random() < 0.5:
                    # Cambiar ranking
                    individual[i] = (f"ranked_{random.randint(0, num_rankings - 1)}", individual[i][1])
                else:
                    # Cambiar género
                    individual[i] = (individual[i][0], random.choice(GENRES))
        print(f"  Después de mutación: {individual}")
        return individual

    # Algoritmo Genético
    def genetic_algorithm(df, num_proteins, num_rankings):
        print(f"Iniciando el algoritmo genético para {num_proteins} proteínas y {num_rankings} rankings disponibles.\n")
        population = initialize_population(num_proteins, num_rankings)
        best_individual = None
        best_fitness = float("-inf")

        for generation in range(GENERATIONS):
            print(f"Generación {generation + 1}...")
            fitnesses = [fitness(ind, df) for ind in population]
            print(f"  Fitness de los individuos: {fitnesses}")

            # Selección y reproducción
            new_population = []
            for _ in range(POPULATION_SIZE // 2):
                parent1 = tournament_selection(population, fitnesses)
                parent2 = tournament_selection(population, fitnesses)

                # Asegurarse de que los padres no sean el mismo individuo
                while parent1 == parent2:
                    parent2 = tournament_selection(population, fitnesses)

                if random.random() < P_CROSSOVER:
                    child1 = crossover(parent1, parent2)
                    child2 = crossover(parent2, parent1)
                else:
                    child1, child2 = parent1, parent2

                child1 = mutate(child1, num_rankings)
                child2 = mutate(child2, num_rankings)

                new_population.extend([child1, child2])

            population = new_population

            # Actualizar mejor solución
            current_best = max(fitnesses)
            if current_best > best_fitness:
                best_fitness = current_best
                best_individual = population[np.argmax(fitnesses)]

            print(f" - Mejor fitness: {best_fitness:.4f}")
```



```python
        for generation in range(GENERATIONS):
            print(f"Generación {generation + 1}...")
            fitnesses = [fitness(ind, df) for ind in population]
            print(f"  Fitness de los individuos: {fitnesses}")

            # Selección y reproducción
            new_population = []
            for _ in range(POPULATION_SIZE // 2):
                parent1 = tournament_selection(population, fitnesses)
                parent2 = tournament_selection(population, fitnesses)

                # Asegurarse de que los padres no sean el mismo individuo
                while parent1 == parent2:
                    parent2 = tournament_selection(population, fitnesses)

                if random.random() < P_CROSSOVER:
                    child1 = crossover(parent1, parent2)
                    child2 = crossover(parent2, parent1)
                else:
                    child1, child2 = parent1, parent2

                child1 = mutate(child1, num_rankings)
                child2 = mutate(child2, num_rankings)

                new_population.extend([child1, child2])

            population = new_population

            # Actualizar mejor solución
            current_best = max(fitnesses)
            if current_best > best_fitness:
                best_fitness = current_best
                best_individual = population[np.argmax(fitnesses)]

            print(f"  - Mejor fitness: {best_fitness:.4f}")
            print(f"  - Mejor individuo hasta ahora: {best_individual}\n")

        print("Evolución completada.\n")
        return best_individual

    # Ejecución del Algoritmo
    if 'df_logs_normalized_global' not in globals():
        print("Error: 'df_logs_normalized_global' no está definido. Asegúrate de cargar o generar el DataFrame.")
    else:
        unique_proteins = df_logs_normalized_global["Modelo 1 Proteína"].unique()
        num_proteins = len(unique_proteins)
        num_rankings = len(df_logs_normalized_global["Modelo 1 Ranking"].unique())

        best_solution = genetic_algorithm(df_logs_normalized_global, num_proteins, num_rankings)

        print("Mejor solución encontrada:")
        for i, (ranking, genre) in enumerate(best_solution):
            print(f"Proteína {i + 1}: Ranking = {ranking}, Género = {genre}")
```

```
Iniciando el algoritmo genético para 46 proteínas y 5 rankings disponibles.

Inicializando población...
Población inicial generada:
Individuo 1: [('ranked_2', 'genero4'), ('ranked_2', 'genero2'), ('ranked_2', 'genero2'), ('ranked_1', 'genero3'), ('ranked_0'
Individuo 2: [('ranked_0', 'genero3'), ('ranked_3', 'genero3'), ('ranked_2', 'genero2'), ('ranked_3', 'genero3'), ('ranked_2'
Individuo 3: [('ranked_1', 'genero4'), ('ranked_3', 'genero2'), ('ranked_4', 'genero1'), ('ranked_1', 'genero4'), ('ranked_2'
Individuo 4: [('ranked_4', 'genero2'), ('ranked_2', 'genero1'), ('ranked_3', 'genero3'), ('ranked_1', 'genero2'), ('ranked_2'
Individuo 5: [('ranked_0', 'genero2'), ('ranked_3', 'genero4'), ('ranked_3', 'genero3'), ('ranked_2', 'genero2'), ('ranked_3'
Individuo 6: [('ranked_4', 'genero1'), ('ranked_0', 'genero3'), ('ranked_1', 'genero3'), ('ranked_2', 'genero4'), ('ranked_0'
Individuo 7: [('ranked_4', 'genero3'), ('ranked_1', 'genero1'), ('ranked_0', 'genero3'), ('ranked_0', 'genero1'), ('ranked_2'
Individuo 8: [('ranked_0', 'genero2'), ('ranked_2', 'genero2'), ('ranked_1', 'genero1'), ('ranked_2', 'genero2'), ('ranked_1'
Individuo 9: [('ranked_3', 'genero4'), ('ranked_0', 'genero4'), ('ranked_2', 'genero2'), ('ranked_0', 'genero2'), ('ranked_4'
Individuo 10: [('ranked_1', 'genero2'), ('ranked_2', 'genero3'), ('ranked_0', 'genero4'), ('ranked_3', 'genero3'), ('ranked_4
Individuo 11: [('ranked_0', 'genero3'), ('ranked_1', 'genero3'), ('ranked_2', 'genero1'), ('ranked_1', 'genero3'), ('ranked_3
Individuo 12: [('ranked_4', 'genero2'), ('ranked_1', 'genero3'), ('ranked_0', 'genero1'), ('ranked_3', 'genero3'), ('ranked_1
Individuo 13: [('ranked_2', 'genero2'), ('ranked_1', 'genero4'), ('ranked_4', 'genero2'), ('ranked_4', 'genero1'), ('ranked_0
Individuo 14: [('ranked_1', 'genero4'), ('ranked_0', 'genero1'), ('ranked_1', 'genero4'), ('ranked_3', 'genero1'), ('ranked_3
Individuo 15: [('ranked_2', 'genero4'), ('ranked_2', 'genero3'), ('ranked_3', 'genero2'), ('ranked_3', 'genero1'), ('ranked_3
Individuo 16: [('ranked_1', 'genero4'), ('ranked_3', 'genero3'), ('ranked_2', 'genero3'), ('ranked_0', 'genero4'), ('ranked_0
Individuo 17: [('ranked_4', 'genero4'), ('ranked_3', 'genero2'), ('ranked_3', 'genero1'), ('ranked_1', 'genero2'), ('ranked_3
Individuo 18: [('ranked_1', 'genero4'), ('ranked_4', 'genero3'), ('ranked_4', 'genero4'), ('ranked_1', 'genero1'), ('ranked_2
Individuo 19: [('ranked_2', 'genero2'), ('ranked_0', 'genero2'), ('ranked_1', 'genero4'), ('ranked_0', 'genero4'), ('ranked_4
Individuo 20: [('ranked_3', 'genero3'), ('ranked_3', 'genero1'), ('ranked_2', 'genero3'), ('ranked_0', 'genero2'), ('ranked_3
Individuo 21: [('ranked_4', 'genero4'), ('ranked_2', 'genero3'), ('ranked_1', 'genero2'), ('ranked_1', 'genero2'), ('ranked_2
...
  - Mejor fitness: 2.3823
  - Mejor individuo hasta ahora: [('ranked_4', 'genero2'), ('ranked_1', 'genero3'), ('ranked_0', 'genero1'), ('ranked_3', 'gen

Generación 10...
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Constancia

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

EL CENTRO DE CIENCIAS BÁSICAS OTORGA LA PRESENTE

CONSTANCIA

al profesor de apoyo

Walbert Rivero Macee

como ponente en el evento de Presentación de Resultados Finales de los Miniproyectos con el trabajo:

MP 24 - 112

"Automatización del proceso de cálculo de distancias entre proteínas Spike de organismos de la familia coronaviridae"

Evento realizado el día 12 y 13 de noviembre de 2024

"SE LUMEN PROFERRE"

Aguascalientes, Ags.

M. en C. Jorge Martín Alférez Chávez.
Decano del Centro de Ciencias Básicas

Dr. Alejandro Padilla Díaz
Secretario de Investigación y Posgrado
Centro de Ciencias Básicas