UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ESTUDIO COMPARATIVO DE LAS LIBRERÍAS DE MACHINE LEARNING DE PYTHON Y RUST

TRABAJO FIN DE GRADO

Abril - 2025

AUTOR: Francesc Baeza Terol
DIRECTOR/ES: Jesús Javier Rodríguez Sala

RESUMEN

Este Trabajo de Fin de Grado explora y compara los lenguajes de programación Rust y Python en el contexto de la ciencia de datos y la inteligencia artificial (IA). El lenguaje Rust surgió para abordar los problemas de seguridad de memoria de C y C++, buscando robustez y rendimiento. Por su parte, Python se centró en la facilidad de uso y aprendizaje. Ambos lenguajes han evolucionado significativamente y a día de hoy se utilizan en diversos ámbitos empresariales.

Rust se distingue por su control explícito de la memoria, su seguridad en la gestión de la misma sin hacer uso de un recolector de basura, y también por su alto rendimiento, comparable a C/C++. Python sobresale por ser un lenguaje de alto nivel, interpretado, con tipado dinámico y una vasta biblioteca estándar. Ambos son multiparadigma y multiplataforma. Para el desarrollo, se analizan IDEs como VS Code, IntelliJ IDEA y CLion para Rust, y PyCharm, Spyder y JupyterLab para Python, siendo VS Code el IDE seleccionado para este proyecto por su versatilidad y compatibilidad con ambos lenguajes.

En el ámbito de la ciencia de datos e IA, Python cuenta con librerías consolidadas como Pandas, NumPy, TensorFlow, PyTorch y Scikit-learn. Rust está desarrollando su entorno con librerías como Polars, SmartCore, Burn y Linfa, priorizando el rendimiento y la seguridad. Los experimentos realizados con modelos de clasificación, regresión y clustering han mostrado que Python, con sus librerías más optimizadas como Scikit-Learn, TensorFlow y PyTorch, generalmente ofrece un equilibrio entre precisión y tiempo de ejecución más favorable y una mayor facilidad de uso. Por otra parte, en el desafío 1BRC (1 Billion Row Challenge, el problema del billón de líneas), Rust ha demostrado ser significativamente más eficiente en el procesamiento de grandes volúmenes de datos, superando a Python en velocidad. Este análisis sugiere que Python es ideal para la investigación, el prototipado rápido y el desarrollo de modelos de IA complejos, mientras que Rust se presenta como una alternativa poderosa para aplicaciones que demandan alto rendimiento, seguridad y control de recursos.

AGRADECIMIENTOS

Nunca he sabido cómo agradecer todo el apoyo que me han dado en mi vida y durante la carrera pero los nombraré porque han sido importantes para mi. Para empezar a Jesuja, mi tutor, ya que él me propuso este trabajo y el que me ha ayudado en todo momento a acabarlo y a aumentar mis conocimientos en otro campo que desconocía.

Agradezco a toda mi familia, a mi madre Mª Carmen y a mi padre Vicent (D.E.P.) por el apoyo incondicional que he tenido para que hiciera lo que me gusta y que estudiara aunque empezara más mayor que los demás. A mis hermanos Roger y Andrea por animarme a estudiar y a que fuera una mejor versión de mi mismo.

A mi psicóloga Maria Rivera, antigua profesora de la UMH, ya que me ayudó en momentos muy difíciles que pasé en los que me quedaba nada de carrera y no quería acabarla.

Y por último, a todos mis compañeros de la universidad Santiago Moltó, Enrique Laura, Daniel Rubio y Lluis Martinez por ayudarme durante toda la carrera y por los buenos momentos que hemos vivido, y gracias a ello conservamos la amistad.

ÍNDICE GENERAL

1. Introducción 8
1.1. Lenguajes de programación e inteligencia artificial 8
1.2. Justificación del proyecto 10
1.3. Objetivos 11

2. Antecedentes y estado de la cuestión 13
2.1. Historia y evolución de Rust y Python 13
2.1.1. Origen de Rust 13
2.1.2. Origen de Python 15
2.2. Características de Rust y Python 16
2.2.1. Características de Rust 16
2.2.2. Características de Python 18
2.3. Entornos de desarrollo 20
2.3.1. IDE's para Rust 21
2.3.1.1. VS Code 21
2.3.1.2. IntelliJ IDEA 22
2.3.1.3. CLion 23
2.3.2. IDE's para Python 24
2.3.2.1. PyCharm 24
2.3.2.2. Spyder 25
2.3.2.3. JupyterLab 26
2.3.3. Tablas Comparativas 27
2.4. Librerías y paquetes 28
2.4.1. Librerías de Rust para Ciencia de datos 29
2.4.1.1. Polars 29
2.4.1.2. SmartCore 29
2.4.1.3. Burn 30
2.4.1.4. Linfa 30
2.4.2. Librerías de Python para Ciencia de datos 31
2.4.2.1. Pandas y NumPy 31
2.4.2.2. TensorFlow 32
2.4.2.3. PyTorch 33
2.4.2.4. Scikit-Learn 34
2.4.3. Comparación general 34

3. Hipótesis de trabajo 36
3.1. Tópicos en Ciencia de datos 36
3.1.1. Definición 36

3.1.2. Áreas clave dentro de la Ciencia de datos 37
3.2. Librerías y paquetes de Machine Learning e inteligencia artificial 37
3.3. Librerías para medir rendimiento y velocidad de algoritmos 38
3.4. IDE seleccionado 38
3.4.1. Plugins 39
3.4.1.1. Rust Analyzer 39
3.4.1.2. Python (Plugin) 39
3.4.1.3. Python Extension Pack 40
3.4.1.4. GitHub Copilot 40
3.5. Repositorios 41
3.5.1. GitHub 41
3.5.2. Kaggle 41
3.5.3. UCI Machine Learning Repository 42
3.5.4. El Problema del Billón de Líneas 42

4. Metodología y resultados 44
4.1. Planificación del proyecto 44
4.2. Descripción de datos 49
4.2.1. Housing.csv 49
4.2.2. Bank-full.csv 50
4.3. Resultados 50
4.3.1. Algoritmos de clasificación 51
4.3.2. Algoritmos de Regresión 55
4.3.3. Algoritmos de Clustering 56
4.3.4. Resultados del 1BRC 59

5. Conclusiones y trabajo futuro 63
5.1. Conclusiones 63
5.2. Trabajo futuro 65

6. Bibliografía 66

ÍNDICE DE TABLAS

Tabla 2.1 IDE’s para Rust 28
Tabla 2.2 IDE’s para Python 28
Tabla 2.3 Comparación general 35
Tabla 4.1 Modelos utilizados y librerías correspondientes en Python y Rust 45
Tabla 4.2 Resultados de clasificación 51
Tabla 4.3 Resultados de regresión 55
Tabla 4.4 Resultados de clustering 57
Tabla 4.5 Resultados del problema 1BRC 60

ÍNDICE DE FIGURAS

Figura 2.1 Interfaz Visual Studio Code 22
Figura 2.2 Interfaz IntelliJ IDEA 23
Figura 2.3 Interfaz CLion 24
Figura 2.4 Interfaz PyCharm 25
Figura 2.5 Interfaz Spyder 26
Figura 2.6 Interfaz JupyterLab 27
Figura 3.1 Visual Studio Code 39
Figura 3.2 Plugin Rust Analyzer 39
Figura 3.3 Plugin Python 40
Figura 3.4 Plugin Extension Pack 40
Figura 3.5 GitHub Copilot 40
Figura 3.6 Extracto del fichero 1BRC 43
Figura 4.1 Gráfica de PyTorch 2 variables y 2 clusters (k=2) 57
Figura 4.2 Gráfica de PyTorch 2 variables y 3 clusters (k=3) 58
Figura 4.3 Gráfica de PyTorch 2 variables y 5 clusters (k=5) 58
Figura 4.4 Gráfica de PyTorch 5 variables y 5 clusters (k=5) 59

ÍNDICE DE ALGORITMOS

Algoritmo 4.1 Estructura general de los scripts de clasificación '*cross-validation*' 47
Algoritmo 4.2 Estructura general de los scripts de regresión 48
Algoritmo 4.3 Estructura general de los scripts de Clustering 49
Algoritmo 4.4 Estructura general del algoritmo 1BRC 60
Algoritmo 4.5 Estructura general del algoritmo 1BRC con *chunks* 61

Capítulo 1
Introducción

1.1.- LENGUAJES DE PROGRAMACIÓN E INTELIGENCIA ARTIFICIAL

La Inteligencia Artificial (IA) es un campo amplio que busca desarrollar sistemas inteligentes capaces de llevar a cabo tareas que normalmente requerirían inteligencia humana, como aprender, razonar y resolver problemas [1]. Para materializar estos sistemas, los expertos en IA confían en los lenguajes de programación, que actúan como el nexo entre las ideas abstractas y las máquinas.

Los lenguajes de programación y la IA guardan una estrecha relación, tan obvia que a veces se pasa por alto. Algunas de las razones de esta conexión son [2][3]:

- **Expresión Algorítmica**: La IA se basa en algoritmos complejos que guían a un sistema en el procesamiento de información y la toma de decisiones. Los lenguajes

de programación ayudan a investigadores y desarrolladores a implementar estos algoritmos rigurosos, ya que incluso pequeñas imprecisiones pueden alterar el comportamiento del sistema. Desde algoritmos de clasificación hasta redes neuronales profundas en el aprendizaje automático, los lenguajes de programación permiten expresar estas técnicas de forma clara y ejecutable. Un ejemplo evidente serían las bibliotecas como TensorFlow y PyTorch, que facilitan la implementación y el trabajo con redes neuronales.

- **Manipulación de Datos**: La IA está impulsada por los datos, y la capacidad de manejar grandes volúmenes de datos de forma eficiente incluye cargar, limpiar y transformar los datos en formatos adecuados para el análisis y el uso en modelos de IA. Otra biblioteca popular que permite manipular datos de manera rápida y optimizada es Pandas.

- **Desarrollo de Modelos**: Los modelos de IA son representaciones matemáticas que ayudan a interpretar fenómenos complejos observados en el mundo real. Estos modelos se desarrollan usando lenguajes de programación, lo que permite diseñar, entrenar y ajustar modelos, desde los más simples modelos lineales hasta las redes neuronales profundas. La programación también permite automatizar los procesos de entrenamiento, adaptando los parámetros del modelo para mejorar su rendimiento y optimizar el proceso de aprendizaje.

- **Desarrollo de Aplicaciones**: La IA se implementa en aplicaciones prácticas que interactúan con usuarios o sistemas en el mundo real. La programación facilita la creación de interfaces y aplicaciones que utilizan modelos de IA para ofrecer funcionalidades inteligentes. Este es el último eslabón de la cadena, donde el código permite que la IA funcione en entornos reales, mejorando la forma en que las personas interactúan con esta tecnología.

Elegir el lenguaje de programación idóneo puede marcar la diferencia en el éxito de un proyecto de IA. Algunos aspectos a tener en cuenta son [4][5]:

- **Rendimiento:** El rendimiento en IA es esencial, ya que muchas aplicaciones implican cálculos intensivos, especialmente en redes neuronales y procesamiento de grandes volúmenes de datos. Python, aunque interpretado, cuenta con bibliotecas como NumPy y TensorFlow que optimizan el rendimiento utilizando compilación en C y C++.

- **Facilidad de uso**: La sintaxis del lenguaje tiene que ser simple y legible, lo que va a permitir a los desarrolladores concentrarse más en la lógica de IA, pudiendo abstraerse de otros detalles de implementación de más bajo nivel.

- **Escalabilidad:** Debido a la gran cantidad de datos a procesar y la complejidad de muchos de los modelos se necesita de sistemas que sean escalables. El uso de frameworks que permiten gestionar estos datos ha hecho que se puedan desarrollar nuevas herramientas para Big Data y la computación paralela, que mejoran el rendimiento a la hora de entrenar y ejecutar estos modelos.

- **Comunidad:** Una comunidad de desarrolladores activa y en constante crecimiento puede proporcionar soporte, recursos y herramientas valiosas. Hace que el lenguaje no sé quede estancado y se encuentren nuevos modelos y ayudas.

- **Eficiencia**: Se necesitan lenguajes eficientes en gestión de memoria y en paralelización para ser buenos en tiempo de cómputo y de ejecución.

1.2.- JUSTIFICACIÓN DEL PROYECTO

Seleccionar el mejor lenguaje de programación es fundamental para el éxito en los proyectos de Inteligencia Artificial (IA). A pesar de que Python es ampliamente utilizado en este ámbito, debido al creciente interés en el lenguaje Rust, resulta interesante hacer una comparación detallada que ayude a los desarrolladores a tomar decisiones fundamentadas sobre qué lenguaje puede ser más idóneo en un momento dado.

La realización del presente trabajo está motivada por las siguientes razones:

- **Creciente demanda de soluciones de IA:** La IA está transformando diversos sectores y la demanda de soluciones eficientes y escalables está en constante aumento [5][6].

- **Diversidad de lenguajes:** La existencia de varios lenguajes de programación para el desarrollo de IA nos lleva a la cuestión de cuál es el más conveniente para cada tarea. Python es conocido por su simplicidad y apoyo de bibliotecas mientras que Rust es valorado por su seguridad y rendimiento [7].

- **Limitaciones de Python:** A pesar de ser Python un lenguaje muy versátil y fácil de aprender, a veces aparecen limitaciones en lo que respecta al rendimiento y la seguridad, sobre todo en las aplicaciones de gran escala [8].

- **Crecimiento en popularidad de Rust**: Rust, está ganando terreno como opción para IA, especialmente en sistemas donde la seguridad y la eficiencia son esenciales. A través de su enfoque de seguridad, rendimiento y concurrencia, es una alternativa atractiva para los sistemas de IA [9].

- **Brecha de conocimiento:** A pesar de que aprender algún lenguaje de programación es interesante, la falta de estudios comparativos enfocados en IA representa una oportunidad para realizar investigaciones. Aún se requieren estudios comparativos para explorar sus fortalezas y debilidades dentro del ámbito de la IA [10].

1.3.- OBJETIVOS

El objetivo principal es saber si Rust es una mejor opción que Python para el desarrollo de aplicaciones de inteligencia artificial mediante una comparación detallada de ambos lenguajes en términos de rendimiento, seguridad, facilidad de uso, escalabilidad y soporte comunitario, aunque en lo que más nos vamos a centrar va a ser en el rendimiento que es la característica que más le preocupa a los desarrolladores junto a la seguridad.

Más específicamente, los objetivos se pueden estructurar del siguiente modo:

- **Analizar el rendimiento de Rust y Python en tareas intensivas de computación**: Evaluar el tiempo de ejecución, el consumo de recursos y la capacidad de paralelización en procesos clave como el análisis de grandes conjuntos de datos y algoritmos.

- **Evaluar la seguridad y gestión de memoria en aplicaciones de IA desarrolladas con Rust frente a Python**: Comparar la capacidad de ambos lenguajes para prevenir errores comunes como fugas de memoria, condiciones de carrera y accesos no seguros en aplicaciones de IA de gran escala y alta concurrencia.

- **Comparar la facilidad de desarrollo en ambos lenguajes, incluyendo bibliotecas y frameworks de IA**: Investigar la curva de aprendizaje, el tiempo requerido para implementar una solución desde cero y la disponibilidad de herramientas frente a alternativas emergentes en Rust.

- **Examinar la escalabilidad de aplicaciones creadas con ambos lenguajes en escenarios de alto volumen de datos**: Diseñar y ejecutar pruebas que simulen casos de uso reales, como procesamiento de datos en tiempo real y ejecución en entornos distribuidos, para evaluar la capacidad de ambos lenguajes de manejar grandes volúmenes de datos y usuarios concurrentes.

- **Analizar el nivel de soporte comunitario y la disponibilidad de recursos para desarrolladores en ambos lenguajes**: Investigar la cantidad y calidad de

documentación, foros, contribuciones en GitHub, y la frecuencia de actualizaciones en herramientas y bibliotecas para IA disponibles en ambos ecosistemas.

- **Implementar casos prácticos de aplicaciones de IA utilizando Rust y Python**: Desarrollar prototipos funcionales que incluyan tareas típicas de IA, como clasificación de imágenes, procesamiento de lenguaje natural o sistemas de recomendación, y medir las diferencias en desarrollo, mantenimiento y rendimiento.

- **Proporcionar recomendaciones basadas en los resultados obtenidos**: Crear un conjunto de directrices claras y accionables que identifiquen las áreas donde cada lenguaje sobresale y cuáles son más adecuadas para distintas aplicaciones o escenarios en el ámbito de la Inteligencia Artificial.

Capítulo 2 Antecedentes y estado de la cuestión

2.1.- HISTORIA Y EVOLUCIÓN DE RUST Y PYTHON

2.1.1. Origen de Rust

Rust fue creado en 2006 por Graydon Hoare, programador en Mozilla, como respuesta a los problemas recurrentes de errores de memoria que afectan a lenguajes como C y C + +. Estos errores, comunes en software de alto rendimiento, motivaron a Hoare a diseñar un lenguaje que combinara la velocidad y eficiencia de C con un manejo de memoria más seguro, eliminando la necesidad de un recolector de basura y minimizando vulnerabilidades críticas.

El lenguaje fue nombrado “Rust” en referencia a unas esporas de hongos conocidas por adaptarse a condiciones adversas, simbolizando la robustez y fiabilidad que buscaba lograr.

Rust tuvo su primera versión pública en 2010, y tras un crecimiento inicial lento debido a una comunidad pequeña, ganó popularidad al atraer a desarrolladores interesados en su enfoque en la seguridad y el rendimiento. La primera versión estable, Rust 1.1.0, fue lanzada el 25 de junio de 2015.

Desde entonces, Rust ha adoptado un modelo de desarrollo iterativo y colaborativo, lanzando nuevas versiones aproximadamente cada seis semanas. Este ritmo ha permitido al lenguaje evolucionar rápidamente, adaptándose a las necesidades de los desarrolladores y ganando un lugar destacado en la programación moderna.

Para indicar algunos datos de proyectos que se ejecutan en este lenguaje, varias grandes empresas de tecnología adoptaron Rust, lo que ha contribuido al aumento de su popularidad. Algunas de ellas son:

- **Mozilla:** se utilizó Rust para construir componentes críticos del navegador web Firefox, la barra de GPU de Firefox Servo, un motor de renderizado que ha sido escrito en Rust y ha demostrado ser una vitrina para el rendimiento y la seguridad del lenguaje.

- **Dropbox**: su mejora de backend se realiza a través de Rust, ha sido fundamental en el manejo de la sincronización de archivos y la minimización del tiempo de respuesta de las solicitudes.

- **Amazon Web Service (AWS):** se han utilizado diferentes aplicaciones de Rust en varios de sus servicios principalmente debido a los aspectos de seguridad y rendimiento que les ha permitido ofrecer servicios más seguros y eficientes.

Por lo que se puede ver, el impacto que ha tenido en la industria de la programación ha sido significativo, especialmente en áreas donde la seguridad y el rendimiento son críticos. Algunos de los impactos más notables son:

- **Seguridad**: Ha establecido nuevos estándares en términos de seguridad en la programación. Su enfoque en la prevención de errores de memoria ha llevado a una reducción significativa en vulnerabilidades y errores en el software escrito en Rust.

- **Rendimiento**: Rust ha demostrado ser capaz de igualar y en algunos casos superar el rendimiento de C y C++, gracias a su capacidad de optimización y control de bajo nivel de la memoria.

- **Educación**: Rust también ha tenido un impacto en el ámbito educativo, ya que se ha convertido en una herramienta popular para enseñar conceptos de programación

segura y eficiente. Muchas universidades han comenzado a incluir Rust en sus planes de estudio de ciencias de la computación.

El futuro de Rust parece ser muy prometedor, su comunidad sigue creciendo y la adopción por parte de empresas y proyectos importantes también sigue en aumento. Con cada versión se introducen mejoras y nuevas características que fortalecen aún más su posición como un lenguaje de programación que puede llegar a ser líder en la industria, aunque aún es difícil de llegar a eso ya que la curva de aprendizaje es muy alta [11][12].

2.1.2. Origen de Python

Python surge a finales de los años 80, cuando un programador holandés llamado Guido Van Rossum decidió empezar el proyecto como un pasatiempo para darle continuidad al lenguaje de programación ABC, de cuyo equipo de desarrollo había formado parte en el CWI (Centro de investigación holandés donde actualmente alberga la oficina central del W3C). El nombre "Python" tiene su origen por la afición de Van Rossum al grupo humorístico británico "*Monty Python*". Este lenguaje se enfoca en ser fácil de usar y aprender, manteniendo potencia en su desempeño, pero el hardware de la época hacía difícil su uso y el proyecto fue aparcado.

Van Rossum, el autor principal del proyecto Python, continúa a día de hoy ejerciendo un rol central decidiendo la dirección del lenguaje. En 1991 se lanzó la primera versión de Python (v.0.9.0). A partir de aquí se empezaron a realizar mejoras continuas al lenguaje, poco a poco se le han ido añadiendo diversas actualizaciones, como la programación orientada a objetos, y una amplia gama de librerías. Con el lanzamiento de sucesivas versiones, Python ha ido ganando cada vez más popularidad al ir mejorando sus posibilidades.

En 1994 se liberó la versión Python 1.0, se dijo que era "*un nuevo comienzo*" ya que resultó ser una gran revolución. Estas mejoras sentaron las bases de lo que es el lenguaje ahora mismo, destacando especialmente su estabilidad, lo que hizo que ganará mucha aceptación entre los desarrolladores.

Unos años después salieron 2 versiones de las que se habló mucho. Una fue Python 2 que tuvo muchas mejoras y añadido de características nuevas, pero también produjo una serie de problemas, en especial en la compatibilidad con las nuevas tecnologías, algo que no gustó en la comunidad [20][21][22][23]. Para resolver estos problemas se actualizó a Python 3. Se realizaron cambios en la sintaxis y en el manejo de algunas características, logró convertirse en un lenguaje mucho más eficiente y moderno. Estas dos versiones generaron una división en la comunidad de programadores de este lenguaje sobretodo por dos problemas:

- Incompatibilidad de las apps y biblioteca escritas en Python 2.
- Adopción lenta de la comunidad a Python 3 por su necesidad de adaptar el código.

Algunas de las aplicaciones más conocidas que han sido creadas, al menos en parte, utilizando código Python son:

- **Dropbox**: La empresa que ofrece el almacenamiento en nube perfecto para guardar fotos, documentos, videos y archivos.

- **Spotify**: Si bien la página web de Spotify está construida en WordPress, la app está desarrollada con Python. Esto permite algunos beneficios, como la posibilidad de análisis de datos, o la implementación de algunos servicios de backend.

- **Panda 3D**: Este programa en Python refleja el gran potencial que tiene este lenguaje de programación para el desarrollo de juegos en 3D. Además, muestra de la mejor manera cómo se puede combinar con otros lenguajes de programación como el C++ para ofrecer una funcionalidad más completa.

2.2.- CARACTERÍSTICAS DE RUST Y PYTHON

2.2.1- Características de Rust

El lenguaje Rust ha construido varias características que lo distinguen notablemente de otros lenguajes de programación, particularmente en lo que respecta a la seguridad de la memoria, sin condiciones de carrera y el control explícito de la memoria. Las características principales de Rust, y sus beneficios son [13]:

- **Ejecución dinámica de seguridad (errores y registros)**: Rust tiene funciones avanzadas para detectar y manejar errores en tiempo de ejecución, conocidas como "panic handling". Este sistema permite gestionar los errores sin afectar la estabilidad del programa. Rust también ofrece registros detallados que ayudan en la depuración y mantenimiento del código, permitiendo a los desarrolladores identificar y corregir problemas de manera efectiva.

- **Orientado a objetos (POO)**: Rust soporta conceptos de programación orientada a objetos como estructuras, métodos y rasgos. Las estructuras permiten agrupar datos relacionados en una entidad. Los métodos son funciones asociadas a estas estructuras y facilitan la organización del código. Los rasgos definen

comportamientos compartidos y permiten la reutilización del código. Esto promueve un diseño modular y mantenible [14].

- **Interfaz simple**: La sintaxis de Rust está hecha para ser clara y simple, ayudando a los desarrolladores a escribir código más rápido y con menos errores. Este enfoque minimalista disminuye la complejidad del código y mejora la comprensión, haciendo que Rust sea accesible para principiantes y programadores con experiencia. La simplicidad de su interfaz permite a los desarrolladores enfocarse en resolver problemas en lugar de lidiar con la sintaxis del lenguaje.

- **Gestión automática de guardado**: Rust utiliza un sistema de gestión de memoria llamado "ownership" y "borrowing." Este sistema elimina la necesidad de un recolector de basura, permitiendo a los desarrolladores administrar la memoria de manera segura y eficiente. Los conceptos de "ownership" y "borrowing" aseguran que los recursos se liberen de forma oportuna y segura, previniendo errores comunes como punteros nulos y fugas de memoria. Este control detallado de la memoria mejora el rendimiento y la seguridad del software [15].

- **Inmutable**: Rust fomenta la inmutabilidad, lo que significa que, por defecto, las variables no pueden ser modificadas una vez asignadas. Esta característica promueve la seguridad y la predictibilidad del código, ya que los datos no pueden ser alterados accidentalmente. La inmutabilidad reduce el riesgo de errores y comportamientos impredecibles, facilitando el desarrollo de software fiable y mantenible [16].

- **Compilación nativa y estática**: El compilador de Rust produce ejecutables nativos y estáticos, lo que resulta en aplicaciones altamente eficientes y rápidas. Al no depender de bibliotecas dinámicas, los programas en Rust tienen menos dependencias en tiempo de ejecución, lo que mejora su rendimiento y portabilidad. La compilación estática asegura que todas las dependencias necesarias estén incluidas en el ejecutable final, reduciendo problemas de compatibilidad.

- **Multiplataforma**: Rust es un lenguaje de programación que funciona en Linux, macOS y Windows. Esto ayuda a los desarrolladores a crear aplicaciones aptas para diferentes entornos, facilitando su uso y distribución. La capacidad de Rust de operar en varias plataformas lo hace ideal para proyectos que deben ser compatibles con múltiples sistemas operativos [17].

- **Control de memoria explícita**: Rust proporciona un control detallado de la memoria mediante su sistema de "ownership". Este sistema asegura que cada recurso tenga un propietario único que es responsable de su limpieza, previniendo errores de memoria como punteros colgantes y fugas. El control explícito de la

memoria permite optimizar el rendimiento y garantiza la seguridad en la gestión de recursos. Los desarrolladores pueden escribir código eficiente y seguro sin comprometer el rendimiento [18].

- **Permite cadenas UTF-8**: Soporta cadenas de texto en formato UTF-8 de manera nativa, lo que facilita el manejo de texto multilingüe y la internacionalización de aplicaciones. Este soporte integrado garantiza que las cadenas se gestionen de manera eficiente y segura, evitando problemas de codificación y decodificación. La compatibilidad con UTF-8 permite a los desarrolladores crear aplicaciones que funcionen correctamente con texto en diferentes idiomas y scripts [19].

- **Multiparadigma**: Es un lenguaje multiparadigma que soporta varios estilos de programación, incluidos el procedural, orientado a objetos, funcional y concurrente. Esta flexibilidad permite a los desarrolladores elegir el paradigma que mejor se adapte a sus necesidades y combinar diferentes enfoques para resolver problemas complejos de manera eficiente. La capacidad de trabajar con múltiples paradigmas hace que Rust sea un lenguaje versátil y poderoso.

En resumen, estas características hacen de Rust un lenguaje poderoso y atractivo para el desarrollo de software seguro, eficiente y mantenible. Al combinar rendimiento y seguridad, Rust se posiciona como un lenguaje ideal para una amplia variedad de aplicaciones, desde sistemas operativos hasta servicios web y aplicaciones móviles.

2.2.2- Características de PYTHON

Python es un lenguaje de programación ampliamente conocido por sus características únicas que lo han llevado a convertirse en una de las herramientas más utilizadas en la industria del desarrollo de software. Su diseño se centra en la simplicidad, la legibilidad y la flexibilidad, lo que lo hace ideal tanto para principiantes como para profesionales [20][21][22][23].

- **Lenguaje de alto nivel**: Python es un lenguaje de programación de alto nivel, lo que significa que su sintaxis es cercana al lenguaje humano. Esto facilita que los desarrolladores puedan escribir código de manera más comprensible y mantenible, sin preocuparse demasiado por los detalles del hardware o la gestión de la memoria.

- **Multiparadigma**: Python admite múltiples paradigmas de programación como la programación orientada a objetos (POO), programación funcional, y por supuesto, programación imperativa y procedimental. Este soporte multiparadigma lo convierte en un lenguaje versátil.

- **Interpretado**: Lenguaje interpretado lo que significa que no requiere ser compilado antes de su ejecución. Esto permite a los desarrolladores escribir y probar su código más rápidamente. El intérprete ejecuta el código línea por línea, lo que facilita la depuración y prueba de pequeñas partes del programa.

- **Tipado dinámico**: Python utiliza tipado dinámico, lo que implica que no es necesario declarar explícitamente el tipo de datos de una variable. El intérprete de Python determina automáticamente el tipo de datos en tiempo de ejecución. Esto hace que el desarrollo sea más ágil, aunque puede requerir precauciones adicionales para evitar errores en programas grandes.

- **Extensa biblioteca estándar**: La biblioteca estándar de Python es una de sus mayores fortalezas. Incluye módulos y paquetes para manejar una amplia variedad de tareas, como:
    - Gestión de archivos
    - Procesamiento de texto
    - Desarrollo web
    - Operaciones matemáticas y científicas
    - etc...

- **Multiplataforma**: El código escrito en Python puede ejecutarse en diferentes sistemas operativos como Windows, macOS y Linux sin necesidad de modificaciones importantes, siempre que el intérprete de Python esté disponible en el sistema.

- **Comunidad activa**: Python cuenta con una de las comunidades más activas del mundo de la programación. Esto se traduce en:
    - Miles de librerías y frameworks desarrollados por la comunidad.
    - Abundante documentación y recursos de aprendizaje.
    - Soporte en foros y comunidades como Stack Overflow.

- **Gran versatilidad y usos diversificados**: Python se utiliza en una amplia gama de aplicaciones, incluyendo:
    - Desarrollo web: Frameworks como Django y Flask.
    - Análisis de datos y ciencia de datos: Librerías como Pandas, NumPy o Matplotlib.
    - Aprendizaje automático e IA: TensorFlow, PyTorch y Scikit-learn.
    - Automatización: Scripts para tareas repetitivas.
    - Aplicaciones de escritorio: Uso de PyQt o Tkinter.
    - Desarrollo de videojuegos: Frameworks como Pygame.

- **Programación asíncrona**: Con la introducción de async y await, Python ofrece herramientas nativas para escribir código asíncrono, lo que es ideal para aplicaciones que necesitan manejar múltiples tareas simultáneamente, como servidores web o procesamiento de datos en tiempo real.

- **Excelente compatibilidad**: Python es compatible con diversas herramientas modernas de desarrollo:
    - Entornos de desarrollo integrados (IDEs) como PyCharm, VSCode o Jupyter.
    - Plataformas en la nube como AWS, Azure y Google Cloud.
    - Contenedores y orquestadores como Docker y Kubernetes.

2.3.- ENTORNOS DE DESARROLLO

Un entorno de desarrollo integrado (IDE) es una aplicación de software que ayuda a los programadores a desarrollar código de software de manera eficiente. Este apartado busca identificar y analizar las mejores opciones disponibles para desarrollar con estos lenguajes.

Para estructurar este apartado, se ha optado por un enfoque comparativo basado en criterios clave que se consideran esenciales al evaluar un IDE, como son: la facilidad de uso, integración con herramientas, capacidades de depuración, nivel de configuración requerido y rendimiento general. Este enfoque asegura que la evaluación sea práctica y orientada a las necesidades reales de los desarrolladores [20][24]:

- **Facilidad de uso**: Se mira lo accesible e intuitiva que es una IDE para los desarrolladores, revisando aspectos como la interfaz de usuario, curva de aprendizaje y simplicidad en la configuración inicial.

- **Integración con herramientas:** Este criterio mide la capacidad de la IDE para integrarse con otras herramientas y servicios externos. Algunos aspectos son la compatibilidad con control de versiones, soporte para herramientas como GitLab o GitHub, integración con frameworks y librerías, y soporte para contenedores y entornos virtuales como Docker.

- **Depuración**: Las funcionalidades que ofrece el IDE para detectar y resolver errores en el código. Se ha mirado si tiene depurador integrado, si el IDE detecta errores mientras se escribe código y compatibilidad con depuradores externos.

- **Configuración**: Nivel de personalización y complejidad requerido para configurar el IDE según las necesidades del desarrollador. Incluimos aspectos como la

configuración inicial, el soporte de plugins y extensiones para facilitar las funcionalidades a través de paquetes o extensiones y la gestión de dependencias.

- **Rendimiento**: Se ha buscado que tan rápido y eficiente es el IDE en términos de consumo de recursos y velocidad de respuesta. Se considera la velocidad de inicio al abrirse y estar listo para usar, el uso de recursos del sistema como la CPU y RAM, y por último, reactividad, si la IDE sigue siendo rápida y fluida al manejar proyectos complejos o archivos grandes.

2.3.1.- IDEs para Rust

Los IDEs más conocidos para Rust son los siguientes:

2.3.1.1.- VS Code

Visual Studio Code (VS Code) es uno de los editores de código más populares y ampliamente utilizados en la comunidad de desarrolladores, gracias a su flexibilidad, personalización y amplia gama de extensiones. Es una herramienta gratuita y de código abierto desarrollada por Microsoft que se adapta perfectamente al desarrollo con Rust debido a su compatibilidad con los principales plugins disponibles [25][26]. Sus características principales son:

- **Interfaz de usuario intuitiva:** Una barra lateral que organiza archivos, un terminal integrado y múltiples vistas para trabajar con proyectos grandes.

- **Editor inteligente:** Soporte para resaltado de sintaxis, autocompletado, y refactorización, lo que agiliza la escritura de código.

- **Terminal integrado:** Permite ejecutar comandos y gestionar herramientas como cargo y rustc directamente desde el editor.

Los plugins para trabajar eficientemente con Rust en VS Code son los siguientes:

- **Rust Analyzer**: es la extensión oficial para Rust que ofrece funciones avanzadas como autocompletado, refactorización y ayuda contextual. Tiene integración con cargo para gestionar dependencias, resaltado de errores, sugerencias del editor y herramientas de navegación para explorar proyectos complejos.

- **CodeLLDB**: es una herramienta para depuración paso a paso para lenguajes como Rust, C y C++, con visualización de las variables y compatible con proyectos

basados en cargo. Basada en **LLDB**, ofrece herramientas avanzadas para depurar aplicaciones de sistemas, permitiendo análisis detallados del comportamiento del código en tiempo de ejecución.

- **Crates**: es una extensión fundamental para gestionar y explorar dependencias en el archivo "Cargo.toml", que es el archivo de configuración principal de un proyecto Rust que utiliza Cargo, el gestor de paquetes y herramientas de construcción oficial de Rust. Un crate, que puede ser un ejecutable o una biblioteca reutilizable y compartible entre proyectos, representa la unidad básica de compilación y distribución de código en Rust. En este sentido, un crate es equivalente a un "módulo" o "paquete" en otros lenguajes, y su correcta gestión es esencial para el desarrollo eficiente en Rust.

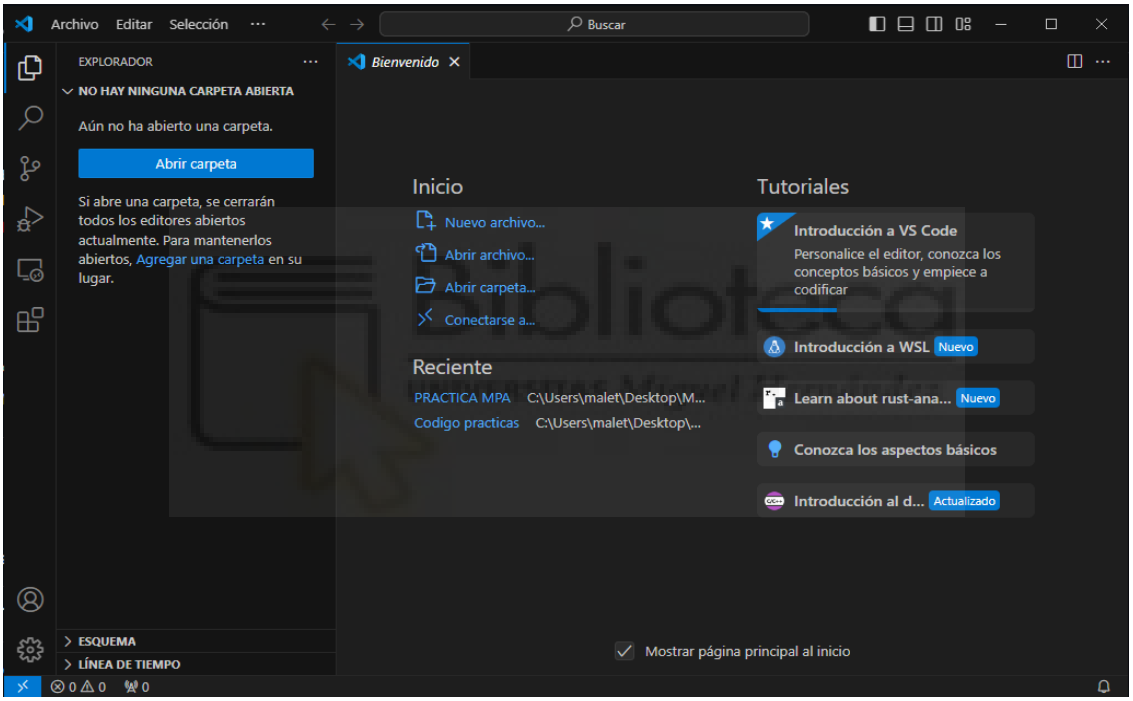

Figura 2.1: Interfaz Visual Studio Code

2.3.1.2.- IntelliJ IDEA

Desarrollado por JetBrains, es un entorno de desarrollo integrado (IDE) ampliamente utilizado por programadores debido a su potencia, flexibilidad y herramientas avanzadas para el desarrollo de software. Aunque originalmente fue diseñado para Java, IntelliJ IDEA admite una amplia variedad de lenguajes mediante plugins, incluido Rust. Gracias a su interfaz intuitiva y características de productividad, se ha convertido en una opción destacada para desarrolladores que buscan trabajar con Rust en proyectos grandes y complejos [27]. Las características principales de este IDE son:

- **Soporte avanzado para Rust**: mediante el plugin “Intellij Rust” que contiene autocompletado de código, detección de errores, navegación eficiente por el código e integración con cargo.

- **Refactorización inteligente**: capacidad para renombrar variables, funciones o estructuras sin romper el código.

- **Depuración integrada**.

- **Terminal incorporado**: que permite ejecutar comandos de Cargo.

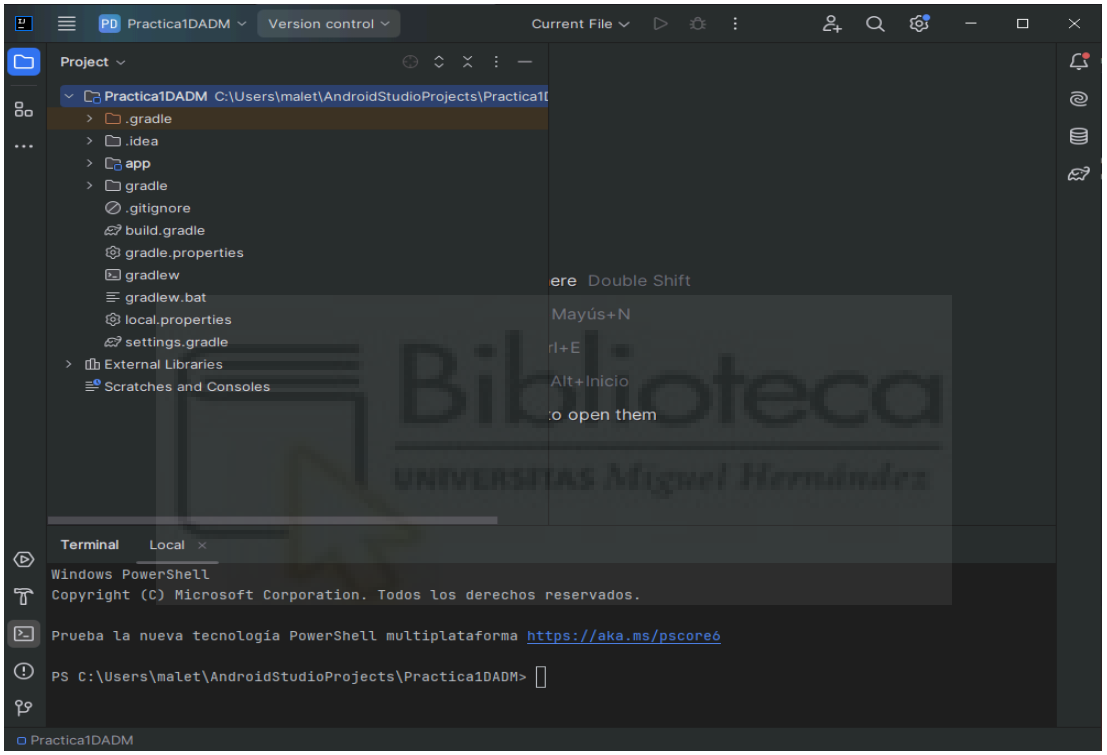

Figura 2.2: Interfaz IntelliJ IDEA

2.3.1.3.- CLion

También fue desarrollado por JetBrains, es un entorno de desarrollo integrado (IDE) especializado en lenguajes como C y C++, aunque también ofrece soporte completo para Rust mediante el plugin oficial "IntelliJ Rust". Diseñado para programadores que trabajan en proyectos de sistemas o de alto rendimiento, CLion se destaca por sus avanzadas herramientas de depuración, navegación de código y análisis estático. Su enfoque en lenguajes de bajo nivel lo convierte en una opción potente para desarrolladores que trabajan con Rust, especialmente en combinación con C o C++ [28]. Algunas de sus características a destacar son:

- Integración nativa con herramientas de Rust como soporte para Cargo y Rustc directamente desde el IDE, ejecución y construcción de proyectos con comandos automáticos.

- Depuración avanzada ya que es compatible con depuradores como GDB Y LLDB, permitiendo análisis detallados del código Rust en ejecución.

- Refactorización y navegación eficiente.

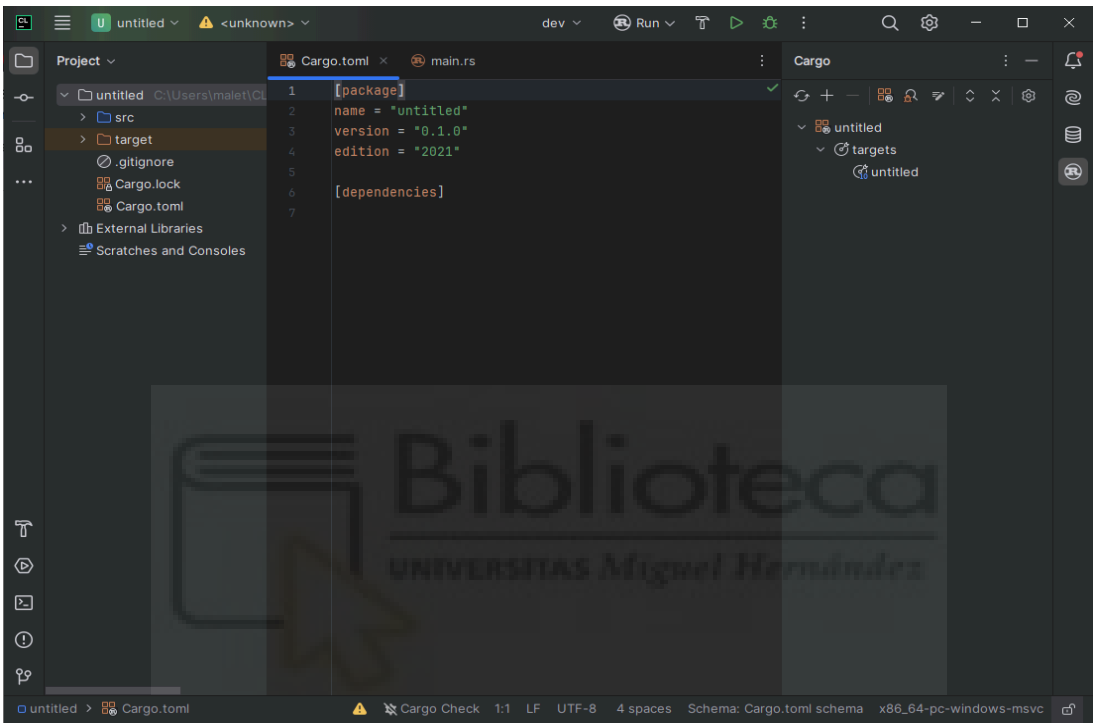

Figura 2.3: Interfaz CLion

2.3.2.- IDE's para Python

Entre los IDEs y entornos de desarrollo para Python destacan los siguientes:

2.3.2.1.- PyCharm

Desarrollado por JetBrains, es uno de los IDEs más potentes y populares para el desarrollo en Python. Diseñado específicamente para este lenguaje, PyCharm ofrece herramientas avanzadas que facilitan la escritura, depuración y gestión de proyectos, ya sea para tareas de desarrollo web, análisis de datos, aprendizaje automático o aplicaciones de escritorio [29][30]. Sus características principales son:

- **Interfaz de usuario intuitiva y familiar**: es igual que la de CLion y IntellijIdea. Organización clara del proyecto con vistas dedicadas a archivos, clases y métodos.

- **Soporte para frameworks y herramientas populares**: integración nativa con Django, Flask y FastAPI.

- **Testing integrado**: con soporte nativo para pruebas unitarias con unittest, pytest y nose.

- **Gestión avanzada de entornos virtuales**: con soporte para herramientas como venv, pipenv y conda

- **Depurador gráfico**: con soporte para breakpoints, inspección de variables y evaluación de expresiones.

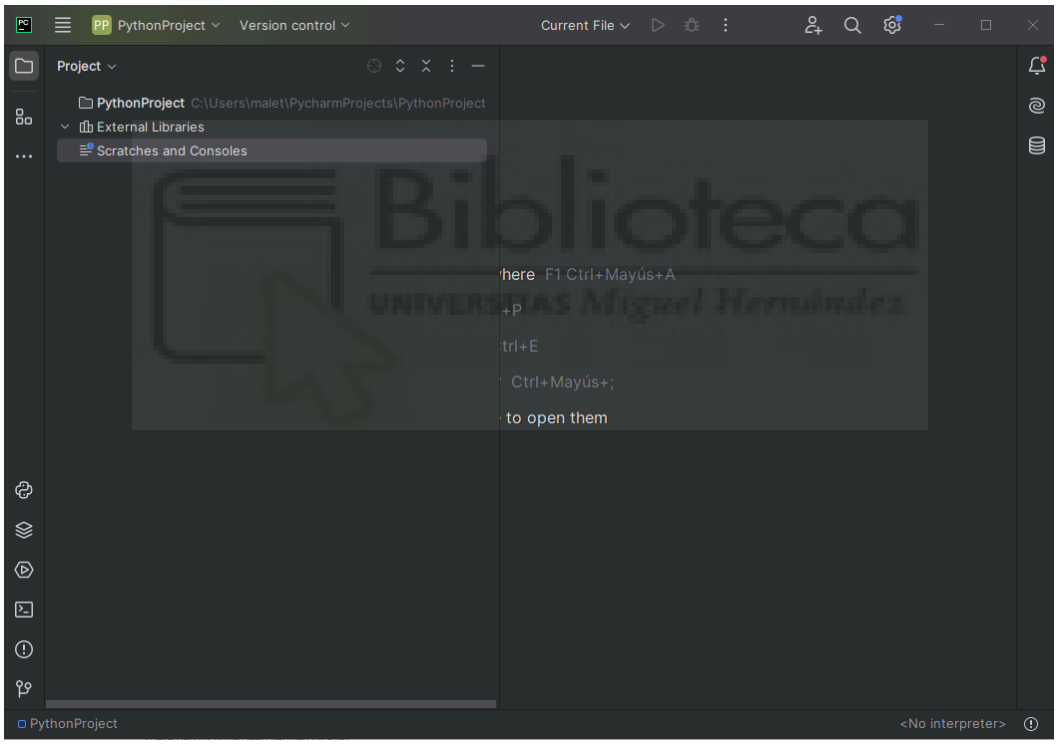

Figura 2.4: Interfaz PyCharm

2.3.2.2.- Spyder

Spyder (Scientific Python Development Environment) es un entorno de desarrollo integrado gratuito y de código abierto diseñado específicamente para científicos de datos, ingenieros y desarrolladores que trabajan en análisis de datos, computación científica y aprendizaje automático. Es una herramienta ligera que combina un editor avanzado con herramientas interactivas para trabajar con datos de manera eficiente, integrándose

perfectamente con las bibliotecas más utilizadas en Python [31][32]. De este IDE, destacan las siguientes características:

- **Interfaz de usuario optimizada**: para análisis de datos con consola interactiva integrada basada en IPython, que permite ejecutar código en tiempo real. Contiene un explorador de variables que muestra datos en formato tabular, ideal para manipular y visualizar matrices, listas o DataFrames.

- **Editor de código avanzado**: con resaltado de la sintaxis y autocompletado inteligente.

- **Soporte para bibliotecas científicas**: optimizado para trabajar con librerías como NumPy, Pandas, Matplotib, SciPy y Scikit-learn, con un modo gráfico interactivo integrado.

- **Integración nativa con Anaconda**: para gestionar entornos y dependencias.

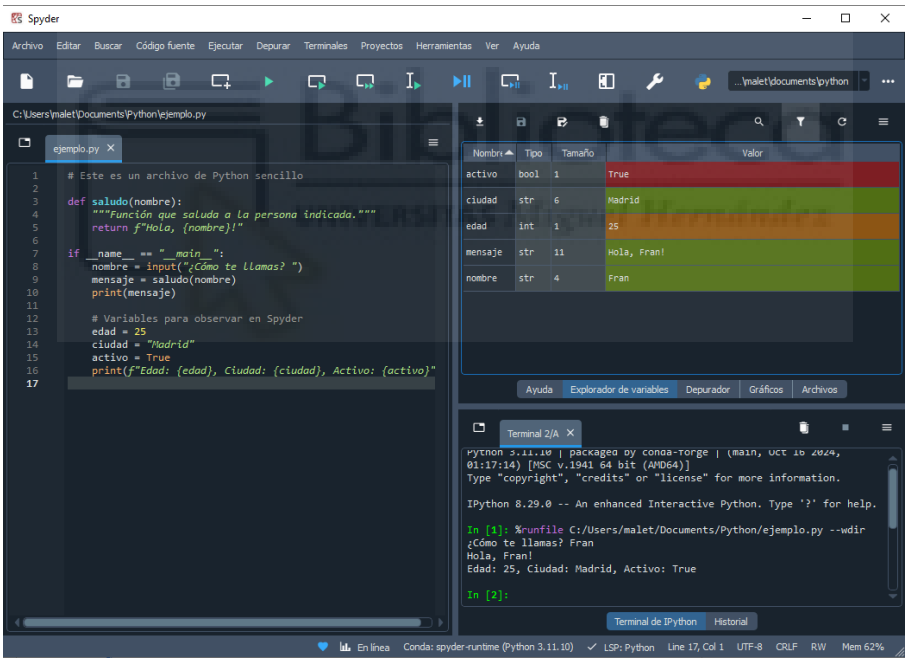

Figura 2.5: Interfaz Spyder

2.3.2.3.- JupyterLab

Es un entorno de desarrollo interactivo y flexible que forma parte del Proyecto Jupyter. Diseñado para mejorar la experiencia de los usuarios que trabajan con notebooks, datos y código, JupyterLab permite combinar múltiples documentos y herramientas en una interfaz unificada y personalizable [33][34]. Sus características son:

- **Interfaz modular y personalizable**: permite organizar múltiples documentos y actividades, como notebooks, editores de texto y terminales, en una disposición y paneles ajustables, facilitando el flujo de trabajo.

- **Soporte para múltiples kernels**: ofrece compatibilidad con diversos lenguajes de programación, incluyendo Python, R y Julia, permitiendo ejecutar código en diferentes entornos.

- **Proporciona consolas interactivas**: actúan como espacios de prueba para ejecutar código de forma dinámica, con soporte completo para resultados enriquecidos.

- **Completo explorador de archivos**: facilita la gestión de archivos y directorios, permitiendo interactuar con ellos.

- **Extensiones**: permite la instalación de extensiones que añaden nuevas funcionalidades, como nuevos temas, editores de archivos y renderizadores para salidas enriquecidas en los notebook.

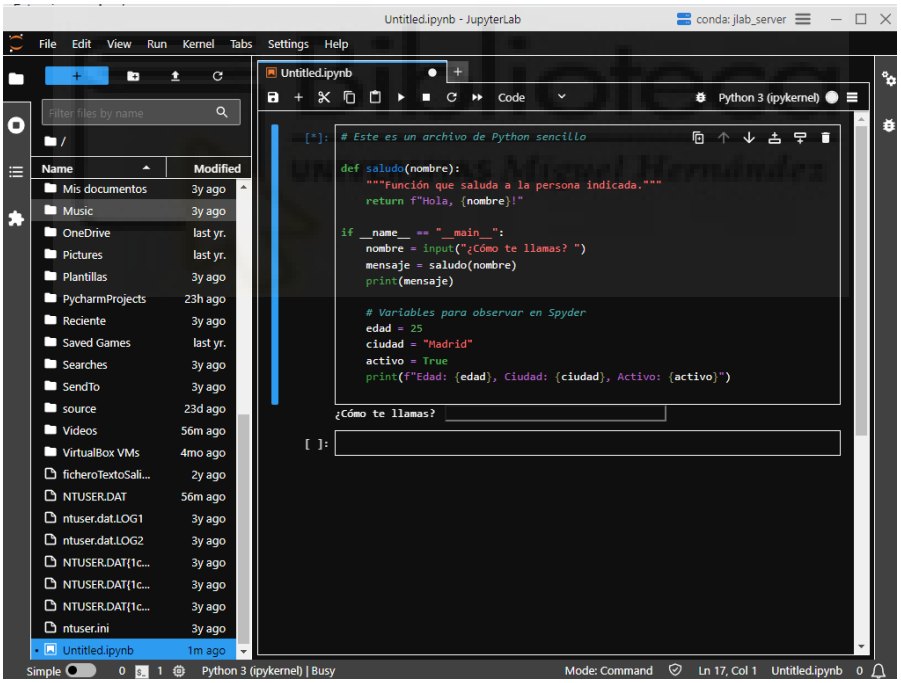

Figura 2.6: Interfaz JupyterLab

2.3.3.- Tablas Comparativas

A modo de resumen, a continuación se muestran dos tablas que permiten comparar fácilmente los diferentes IDEs, tanto para Rust como para Python, atendiendo a varias de sus características.

Tabla 2.1: IDE's para Rust

IDE	Facilidad de uso	Integración con herramientas	Depuración	Configuración	Rendimiento
VS Code	Alta	Excelente	Muy Buena	Moderada	Ligero
IntelliJ IDEA	Media	Muy Buena	Excelente	Baja	Medio
CLion	Media	Muy Buena	Excelente	Baja	Alto

Tabla 2.2: IDE's para Python

IDE	Facilidad de uso	Integración con herramientas	Depuración	Configuración	Rendimiento
PyCharm	Alta	Excelente	Excelente	Moderada	Medio
Spyder	Alta	Muy Buena	Buena	Baja	Ligero
JupyterLab	Alta	Muy Buena	Básica	Muy Baja	Ligero
VS Code	Alta	Excelente	Excelente	Moderada	Medio

2.4.- LIBRERÍAS Y PAQUETES

Las librerías y paquetes son fundamentales para el desarrollo de casi cualquier tipo de aplicación, también las de Inteligencia Artificial (IA). Estas librerías proporcionan herramientas preconstruidas para tareas complejas como el aprendizaje automático, el procesamiento de datos y el desarrollo de modelos de aprendizaje profundo. Tanto Python como Rust cuentan con un conjunto de librerías específicas que facilitan y potencian el trabajo en este ámbito. Además, ambos lenguajes disponen de repositorios oficiales bastante completos:

- **PyPI**: Repositorio oficial de paquetes y librerías para Python. Contiene miles de proyectos desarrollados por la comunidad y es la principal fuente de paquetes para los desarrolladores. Es accesible a través del gestor de paquetes '**pip**', que viene incluido con Python. Contiene paquetes para múltiples propósitos como desarrollo web, análisis de datos o aprendizaje automático [35].

- **Crates.io**: Es el repositorio oficial de librerías y paquetes para Rust, se gestiona mediante '**Cargo**', el gestor de paquetes y herramienta de construcción de Rust. Contiene una amplia variedad de '*crates*' (paquetes) desarrollados por la comunidad y tiene integración con Cargo.toml para gestionar dependencias de manera automática [36].

Python, con su madurez y amplia adopción en la comunidad de IA, ofrece un ecosistema consolidado que abarca desde bibliotecas tradicionales hasta avanzadas herramientas de Deep Learning. Por otro lado, Rust, está ganando terreno gracias a su enfoque en el

rendimiento, la seguridad y la eficiencia. A continuación se describen las principales librerías y paquetes de cada lenguaje, destacando sus características, aplicaciones, y cómo contribuyen al desarrollo de soluciones de Inteligencia Artificial.

2.4.1- Librerías de Rust para Ciencia de Datos

Rust, aunque es más reciente en el ámbito de la Inteligencia Artificial, está ganando terreno gracias a su enfoque en el rendimiento, la seguridad y el manejo eficiente de la memoria. Estas son algunas de las principales librerías y herramientas relacionadas con IA.

2.4.1.1.- Polars

Polars es una biblioteca de procesamiento de datos que se puede utilizar tanto en Rust como en Python y se ha vuelto popular debido a su alto rendimiento y eficiencia, especialmente en el manejo de grandes volúmenes de datos. Diseñada para superar algunas de las limitaciones de pandas, Polars destaca por su arquitectura basada en Apache Arrow, lo que le permite ofrecer procesamiento columnar altamente optimizado y ejecución paralela automática [37].

- **Alto rendimiento:** Está escrito en Rust, lo que lo hace mucho más rápido que pandas en muchas operaciones, especialmente con datos grandes.

- **Ejecución paralela:** Aprovecha el multithreading para realizar operaciones más rápidas en comparación con pandas.

- **Interfaz similar a pandas:** Si vienes de pandas, la curva de aprendizaje es bastante fácil.

- **Lazy execution:** Permite evaluar operaciones sólo cuando son necesarias, optimizando el uso de memoria y el rendimiento.

2.4.1.2.- SmartCore

Librería para aprendizaje automático puro en Rust, ofrece herramientas para el aprendizaje supervisado y no supervisado, así como diversas herramientas para la selección y evaluación de modelos. El conjunto de algoritmos que ofrece son [38]:

- **Aprendizaje supervisado**: Una amplia gama de algoritmos que incluyen modelos lineales, árboles de decisión y SVM.

- **Validación cruzada**: Método iterativo para calcular el rendimiento de modelos de aprendizaje supervisado.

- **Clustering:** Algoritmos que ayudan a descubrir subgrupos o clusters homogéneos inherentes en nuestros datos, como K-means y DBSCAN.

- **Descomposición de matrices**: para resolver sistemas de ecuaciones lineales, calcular determinantes de matrices y matrices inversas.

2.4.1.3.- Burn

Burn es una biblioteca de aprendizaje profundo en Rust que combina el alto rendimiento y la seguridad del lenguaje con una interfaz inspirada en PyTorch, lo que facilita su uso para desarrolladores experimentados. Está diseñada para aplicaciones críticas que requieren eficiencia y fiabilidad, especialmente en sistemas con restricciones de recursos [39][40]. Algunas características de Burn son:

- **Gran rendimiento**: aprovecha el manejo eficiente de memoria de Rust y optimizaciones CPU y GPU, incluyendo soporte para computación paralela con CUDA.

- **Seguridad en recursos**: gracias al sistema de propiedad de Rust, minimiza errores como fugas de memoria, garantizando estabilidad en entornos críticos.

- **Diseño modular:** Permite personalización en componentes clave como grafos computacionales.

- **Rendimiento distribuido**: soporta el uso de múltiples GPUs o nodos, ideal para modelos grandes y procesamiento en paralelo.

- **Familiaridad con PyTorch:** su API está inspirada en PyTorch, facilitando la transición para desarrolladores con experiencia en este framework.

2.4.1.4.- Linfa

Linfa es una biblioteca de aprendizaje automático en Rust diseñada para ofrecer un enfoque simple y modular para el desarrollo de modelos tradicionales de machine learning. Inspirada en Scikit-learn (de Python, ver apartado 2.4.2.3), Linfa proporciona herramientas efectivas para el preprocesamiento de datos, algoritmos clásicos y creación de pipelines,

posicionándose como una opción ideal para proyectos que requieren eficiencia, flexibilidad y control en sistemas críticos [41]. Sus características son diversas y abarcan desde la facilidad de uso hasta la capacidad de personalización:

- **Enfoque inspirado en Scikit-learn**: su diseño se basa en principios similares a Scikit, proporcionando una experiencia familiar.

- **Modularidad:** permite implementar y personalizar componentes específicos de algoritmos y flujos de trabajo, facilitando la adaptación a las necesidades del usuario.

- **Compatibilidad con algoritmos tradicionales:** soporte para métodos clásicos como regresión lineal, clustering (K-means) y análisis de componentes principales (PCA).

- **Optimización y rendimiento:** aprovecha el alto rendimiento nativo de Rust para manejar eficientemente los recursos y ejecutar modelos de machine learning en aplicaciones exigentes.

- **Pipeline flexible**: soporte para construir flujos de trabajo completos, desde el preprocesamiento de datos hasta la evaluación de modelos.

2.4.2- Librerías de Python para Ciencia de Datos

Python es ampliamente reconocido como uno de los lenguajes líderes en el ámbito de la inteligencia artificial (IA) y la ciencia de datos debido a su sintaxis clara, facilidad de uso y extensa comunidad de desarrolladores. Su enfoque de alto nivel permite a los investigadores y desarrolladores concentrarse en la lógica y la experimentación sin preocuparse por detalles complejos de implementación.

2.4.2.1.- Pandas y NumPy

Pandas es una biblioteca de código abierto diseñada específicamente para la manipulación y análisis de datos estructurados en Python. Se basa en NumPy y proporciona estructuras de datos altamente eficientes, como Series y DataFrames, que permiten trabajar con datos tabulares y multidimensionales de manera intuitiva y eficiente. Su nombre proviene del término "Panel Data", utilizado en econometría, y también hace referencia a Python Data Analysis [42].

Uno de los aspectos más destacados de Pandas es su capacidad para procesar grandes volúmenes de datos de manera eficiente, ofreciendo herramientas avanzadas para la manipulación, limpieza y transformación de datos.

- **Estructuras de datos flexibles:** La estructura principal de Pandas es el DataFrame, una tabla bidimensional con etiquetas en los ejes. Esto facilita la manipulación de datos de manera similar a las hojas de cálculo o bases de datos relacionales.

- **Manejo eficiente de datos faltantes:** Pandas ofrece herramientas para detectar, eliminar o reemplazar valores nulos, lo que es esencial para mantener la integridad de los análisis.

- **Operaciones de agrupamiento y agregación:** Permite agrupar datos y realizar operaciones de agregación como sumas, promedios o conteos, facilitando el análisis estadístico.

- **Integración con otras bibliotecas:** Pandas se integra perfectamente con bibliotecas como NumPy para cálculos numéricos, Matplotlib para visualización de datos y SciPy para análisis científicos.

NumPy (Numerical Python) es una biblioteca orientada a la computación científica en Python y se considera la base de muchas otras bibliotecas utilizadas en el análisis de datos, inteligencia artificial y computación numérica. Su principal fortaleza está en su capacidad para manejar grandes volúmenes de datos numéricos de manera eficiente, ofreciendo una mejora significativa en rendimiento en comparación con las listas nativas de Python [43].

- **Vectores multidimensionales (ndarray):** Numpy introduce el objeto ndarray, que permite almacenar y manipular datos de manera eficiente en múltiples dimensiones.

- **Operaciones vectorizadas**: Permite realizar operaciones aritméticas y matemáticas directamente sobre los arreglos sin necesidad de bucles explícitos, lo que mejora el rendimiento y la legibilidad del código.

- **Funciones matemáticas y estadísticas:** Incluye una amplia gama de funciones para operaciones matemáticas, estadísticas y algebraicas, como transformadas de Fourier, álgebra lineal y generación de números aleatorios.

2.4.2.2.- TensorFlow

TensorFlow es una librería de código abierto desarrollada por Google, ampliamente reconocida por su especialización en Machine Learning y el entrenamiento de modelos de

aprendizaje profundo (Deep Learning). Su flexibilidad y escalabilidad la convierten en una de las herramientas más utilizadas tanto en investigación como en producción [44][45]. De esta librería se puede destacar lo siguiente:

- **Soporte para redes neuronales avanzadas:** Permite trabajar con redes neuronales profundas (DNN) y convolucionales (CNN), entre otras arquitecturas.

- **Compatibilidad con hardware avanzado:** Funciona con CPU, GPU y TPU, lo que permite un alto rendimiento en tareas intensivas de cálculo.

- **Gestión de modelos:** Facilita la construcción, entrenamiento y despliegue de modelos en servidores, dispositivos móviles o navegadores web.

- **APIs flexibles:** Proporciona APIs de alto y bajo nivel, lo que permite control detallado o simplicidad según las necesidades del usuario.

- **Visualización con TensorBoard:** Herramienta integrada para monitorizar y depurar modelos de forma gráfica.

2.4.2.3.- PyTorch

PyTorch es una librería de código abierto desarrollada por Facebook AI Research (Meta) que combina el poder de la biblioteca de Machine Learning Torch con una API intuitiva basada en Python. Está diseñada para facilitar la creación, entrenamiento y despliegue de modelos de redes neuronales profundas (aprendizaje profundo o deep learning) [46]. Entre sus características:

- **Amplia variedad de arquitecturas:** Soporta desde modelos simples, como la regresión lineal, hasta complejas redes convolucionales (CNN) y modelos transformadores como GPT para NLP.

- **Gráficos computacionales dinámicos:** A diferencia de TensorFlow, permite construir y modificar gráficos sobre la marcha, lo que simplifica la depuración y el desarrollo iterativo.

- **Compatibilidad con frameworks avanzados:** Admite bibliotecas como Hugging Face y TorchVision para procesamiento de texto y visión por computadora.

- **Optimización de rendimiento:** Funciona eficientemente con CPU y GPU, proporcionando herramientas para entrenamiento distribuido.

- **API Pythonic:** Su diseño intuitivo permite a los desarrolladores centrarse en el diseño lógico sin preocuparse por detalles de implementación.

PyTorch permite crear y ejecutar sofisticadas redes de deep learning, al tiempo que minimiza el tiempo y la mano de obra dedicados al código y la estructura matemática, haciéndolo en tiempo real en lugar de implementar todo el código.

2.4.2.4.- Scikit-Learn

Scikit-learn es una librería gratuita y de código abierto para Python que implementa la mayoría de los algoritmos clásicos de Machine Learning. Es conocida por su simplicidad y efectividad, siendo una herramienta clave para realizar tareas tanto de aprendizaje supervisado, como no supervisado [47]. A continuaciòn se describen sus principales características:

- **Amplia variedad de algoritmos:** Implementa modelos como regresión logística, árboles de decisión, máquinas de vectores soporte (SVM), k-means y reducción de dimensionalidad (PCA).

- **Preprocesamiento de datos:** Ofrece herramientas para escalar, normalizar y transformar datos, facilitando su preparación para modelos.

- **Compatibilidad con el ecosistema Python:** Se integra perfectamente con NumPy, SciPy y Matplotlib, ampliando sus capacidades.

- **Optimización automática de modelos:** Herramientas como GridSearchCV para buscar los mejores hiperparámetros.

- **API sencilla:** Ideal tanto para principiantes como para profesionales que buscan soluciones rápidas.

2.4.3- Comparación General

El proceso de desarrollo de aplicaciones basadas en inteligencia artificial está profundamente influenciado por el lenguaje de programación elegido y las herramientas que forman parte del ecosistema asociado a dicho lenguaje. En este contexto, Python y Rust se destacan como dos opciones interesantes, cada una abordando este campo desde enfoques distintos que reflejan sus características únicas. Python, conocido por su simplicidad y extensa biblioteca de recursos, se ha convertido en una opción preferida para quienes buscan rapidez en la implementación y facilidad de uso. Por otro lado, Rust, con su

enfoque en la seguridad y el rendimiento, ofrece una alternativa robusta para proyectos que requieren un control más detallado y una ejecución eficiente. Ambos lenguajes representan fortalezas específicas y áreas de especialización que los hacen valiosos en diferentes escenarios dentro del ámbito de la inteligencia artificial.

Tabla 2.3: Comparación general

Característica	Python	Rust
Madurez de las librerías	Python cuenta con unas librerías de IA consolidadas, con librerías ampliamente adoptadas como TensorFlow, PyTorch y Scikit-learn.	Rust tiene unas librerías más recientes y en crecimiento, con librerías prometedoras como Burn, Linfa y SmartCore.
Facilidad de uso	Python ofrece una sintaxis sencilla y librerías diseñadas para ser accesibles incluso para principiantes.	Rust presenta una curva de aprendizaje más pronunciada debido a su enfoque en seguridad y control explícito.
Rendimiento	Aunque Python es interpretado, puede alcanzar alto rendimiento mediante la optimización en GPU y TPU gracias a frameworks como TensorFlow y PyTorch.	Rust, al ser un lenguaje compilado, ofrece un rendimiento nativo superior, especialmente en aplicaciones críticas o con restricciones de recursos.
Compatibilidad	Python es compatible con una amplia gama de bibliotecas externas, herramientas y plataformas como AWS, Azure y Google Cloud.	Rust tiene compatibilidad limitada con herramientas externas, pero su integración con sistemas críticos y hardware especializado está mejorando.
Seguridad y manejo de memoria	La seguridad depende de las prácticas del desarrollador y las herramientas utilizadas. Existe riesgo de errores como fugas de memoria.	Rust garantiza seguridad en el manejo de memoria gracias a su sistema de propiedad, minimizando errores en entornos críticos.
Flexibilidad de las librerías	Librerías como TensorFlow y PyTorch ofrecen APIs flexibles y adaptables para tareas avanzadas de IA y Deep Learning.	Librerías como Linfa y Burn, con su diseño modular, permiten personalización y control avanzado, aunque con menor soporte para arquitecturas más complejas.
Casos de uso ideales	Python es ideal para investigación, prototipado rápido y desarrollo en producción con modelos avanzados de IA.	Rust es ideal para sistemas críticos, sistemas embebidos y proyectos de alta eficiencia y control en tiempo real.
Soporte de la comunidad	Python cuenta con una de las comunidades más activas y grandes en el ámbito de IA, con abundante documentación y foros.	La comunidad de Rust en IA es menor, pero está creciendo rápidamente con el interés en librerías como Burn y Linfa.
Herramientas de preprocesamiento	Librerías como Pandas y NumPy facilitan el preprocesamiento y análisis de datos de manera sencilla y eficiente.	Rust dispone de menos herramientas para preprocesamiento, pero librerías como Linfa ofrecen funciones básicas.
Escalabilidad	Python, con librerías como TensorFlow, permite escalar proyectos fácilmente en plataformas de nube y hardware distribuido.	Rust, aunque escalable, requiere más trabajo manual para optimizar y adaptar modelos a sistemas distribuidos.

Capítulo 3
Hipótesis de trabajo

3.1.- TÓPICOS EN CIENCIA DE DATOS

3.1.1.- Definición

La ciencia de datos es una disciplina que integra diversas áreas, como la estadística, las matemáticas y la informática, y un conocimiento especializado en el dominio de los datos que se analizan. Su propósito fundamental es el de analizar, interpretar y extraer conclusiones útiles de grandes volúmenes de datos, lo que permite a organizaciones e investigadores tomar decisiones basadas en los propios datos.

Esta área del conocimiento se ha vuelto especialmente importante en el contexto de la inteligencia artificial (IA) y el aprendizaje automático (Machine Learning, ML), ya que proporciona las herramientas necesarias para trabajar con datos complejos y estructurados a gran escala. Gracias a la ciencia de datos, es posible transformar datos crudos en

información valiosa que puede ser utilizada para predecir tendencias, mejorar procesos y generar nuevos conocimientos. Por tanto, la ciencia de datos se centra tanto en la recopilación y el análisis de datos, como en la capacidad de convertir esos datos en información valiosa para el área objeto de estudio [48].

3.1.2.- Áreas clave dentro de la Ciencia de datos

La ciencia de datos abarca diversas áreas fundamentales que permiten transformar datos en información útil [48]. Entre las más relevantes se encuentran las siguientes:

- **Preprocesamiento y la limpieza de datos**: son pasos fundamentales que implican eliminar valores atípicos, manejar datos faltantes y normalizar variables para garantizar la consistencia y utilidad de los datos para su análisis y modelado.

- **Análisis exploratorio de datos**: se centra en explorar y examinar conjuntos de datos para identificar patrones, tendencias, y relaciones clave. Este proceso puede incluir el uso de estadísticas descriptivas y exploratorias.

- **Modelado predictivo**: utiliza algoritmos de aprendizaje automático para generar modelos capaces de predecir resultados futuros basados en datos históricos; aplicaciones comunes incluyen los pronósticos financieros o los sistemas de recomendación.

- **Visualización de datos**: desempeña un papel crucial al representar información mediante gráficos y diagramas, facilitando la interpretación de hallazgos complejos.

3.2.- LIBRERÍAS Y PAQUETES DE MACHINE LEARNING E INTELIGENCIA ARTIFICIAL

En el capítulo 2 se ha abordado extensamente la importancia de las librerías y paquetes en el ámbito de la inteligencia artificial y el aprendizaje automático. Python se destaca como el líder indiscutible en este campo debido a su amplia gama de herramientas y facilidad de uso, mientras que Rust está emergiendo como una alternativa orientada a proyectos donde el rendimiento y la seguridad son cruciales.

Algunas de las principales librerías ya mencionadas incluyen TensorFlow, una poderosa herramienta para redes neuronales profundas desarrollada por Google, y PyTorch, que permite construir gráficos computacionales dinámicos y es ampliamente utilizado en

investigación. El paquete Scikit-learn, por su parte, es una solución versátil y ampliamente extendida para tareas de aprendizaje automático clásico como clasificación, regresión, clustering, entre otras [49][50].

En Rust, librerías como SmartCore y Burn están ganando relevancia, ofreciendo soporte para algoritmos clásicos y aprendizaje profundo con un enfoque en la seguridad y eficiencia del manejo de memoria. Por su parte, Linfa, inspirada en Scikit-learn, se posiciona como una opción sólida para tareas de machine learning tradicionales en proyectos de alto rendimiento [51].

3.3.- LIBRERÍAS PARA MEDIR RENDIMIENTO Y VELOCIDAD DE ALGORITMOS

El análisis del rendimiento y la velocidad de ejecución de los algoritmos es crucial para optimizar aplicaciones de inteligencia artificial y aprendizaje automático. Tanto en Python como en Rust existen herramientas especializadas para esta tarea.

En Python, se destacan librerías como timeit, que permite medir el tiempo de ejecución de pequeños fragmentos de código, y cProfile, una herramienta que analiza el rendimiento general del programa y detectar cuellos de botella. Además, line_profiler ofrece una visión detallada del tiempo que toma cada línea de un script, siendo útil para optimizaciones específicas [52].

Por otro lado, Rust cuenta con herramientas como Criterion.rs, una biblioteca altamente precisa para medir y analizar el rendimiento de funciones y algoritmos. Esta herramienta genera gráficos y reportes que facilitan la identificación de áreas para mejorar. También se utiliza cargo bench, integrado con el sistema de pruebas de Rust, para realizar benchmarks automatizados y comparaciones entre distintas implementaciones [53].

3.4.- IDE SELECCIONADO

Al buscar información sobre Python y Rust, también se analizaron distintos IDE's especializados para cada lenguaje. Para la realización de este proyecto se ha elegido Visual Studio Code como el principal entorno de desarrollo debido a su versatilidad, amplia comunidad y compatibilidad con ambos lenguajes. Al trabajar con Rust y Python, es esencial contar con un IDE que permita una configuración eficiente y una experiencia de desarrollo fluida.

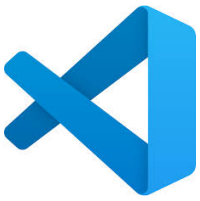

Figura 3.1: Visual Studio Code

Hay varias características de este IDE que han determinado que sea el elegido:

- Es un IDE ligero y altamente configurable.
- Dispone de extensiones oficiales para Rust y Python (ver más adelante).
- Soporta la integración de herramientas de IA como GitHub Copilot, facilitando la generación de código y el autocompletado.
- Tiene una excelente integración con Git y GitHub, permitiendo control de versiones.
- Es compatible con herramientas de depuración.

3.4.1.- Plugins

Para poder trabajar con Rust y Python de manera eficiente se van a utilizar algunos plugins para facilitar el trabajo.

3.4.1.1.- Rust Analyzer

Plugin oficial para el desarrollo en Rust ya que proporciona autocompletado junto con resaltado de sintaxis, refactorización, sugerencias de código, diagnósticos y análisis estático del código [54].

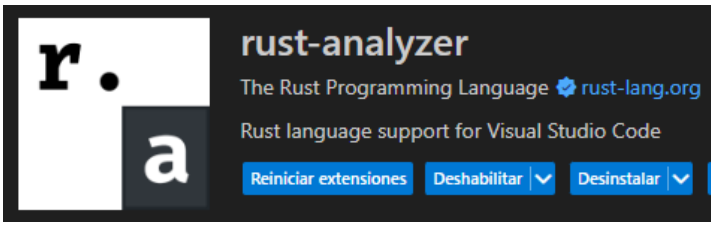

Figura 3.2: Plugin Rust Analyzer

3.4.1.2.- Python (plugin)

Se trata de una extensión específica para el lenguaje Python que proporciona características como el resaltado de sintaxis, soporte para múltiples versiones, integración con entornos

virtuales, herramientas de análisis de código, ejecución y depuración de scripts directamente en el editor [55].

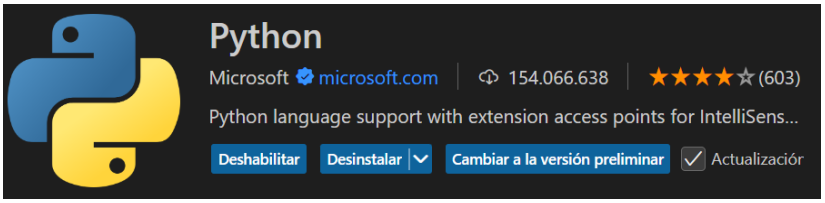

Figura 3.3: Plugin Python

3.4.1.3.- Python Extension Pack

Este es un paquete de plugins para facilitar el desarrollo de código con herramientas como autocompletado de código y depuración, soporte para notebooks de Jupyter y gestión de entornos virtuales y formateo de código. Algunos de los plugins son como el propio de Python, IntelliCode que es una IA que asiste en el desarrollo y Python Indent que ayuda con los autocompletados [56].

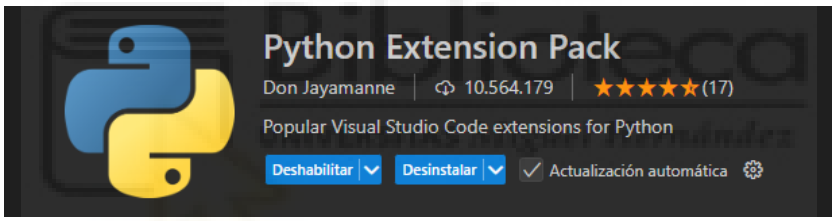

Figura 3.4: Python Extension Pack

3.4.1.4.-GitHub Copilot

Plugin de inteligencia artificial para mejorar la productividad en la programación sugiriendo código en tiempo real, generación de funciones completas a partir de comentarios y del código ya escrito, tiene compatibilidad con Python y Rust (y muchos otros lenguajes de programación) [57].

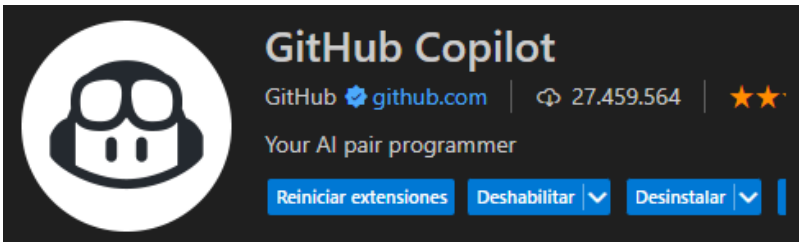

Figura 3.5: GitHub Copilot

3.5.- REPOSITORIOS

En el desarrollo de este proyecto se van a utilizar distintos repositorios para gestionar el código, los datos y la documentación. A continuación se detallan los principales repositorios utilizados.

3.5.1.- GitHub

Github se utilizará como el principal repositorio de código fuente para el proyecto [58]. Se emplea para:

- **Control de versiones**: tanto para Rust como Python, permite realizar un seguimiento de los cambios en el código, facilitando la recuperación de versiones anteriores y la colaboración en equipo.

- **Colaboración**: con el tutor o con otros desarrolladores en caso necesario.

- **Documentación**: Se pueden incluir archivos README para documentar el proyecto, proporcionando una descripción clara de los objetivos y el funcionamiento del código.

- **Gestión del proyecto**: GitHub facilita la gestión de tareas y errores mediante su sistema de issues, ayudando a organizar el desarrollo del proyecto de manera estructurada.

3.5.2.- Kaggle

Kaggle es una plataforma enfocada en el análisis de datos y el desarrollo de modelos de machine learning [59]. En el proyecto se utilizará para:

- **Datos**: Proporciona una amplia colección de datasets públicos que pueden ser utilizados para entrenar y evaluar modelos de inteligencia artificial.

- **Ejecución**: Permite la ejecución de notebooks en servidores de Kaggle, evitando problemas de configuración local y facilitando la experimentación con modelos.

- **Comunidad**: Kaggle cuenta con una comunidad activa de científicos de datos y desarrolladores, donde se pueden encontrar kernels (notebooks) y soluciones a problemas comunes en machine learning.

- **Gamificación**: Ofrece competencias de machine learning donde los participantes pueden comparar el rendimiento de sus modelos con otros, proporcionando una experiencia gamificada.

3.5.3.- UCI Machine Learning Repository

Es una de las bases de datos más antiguas y utilizadas en la investigación de aprendizaje automático e inteligencia artificial. Se ha mantenido como un recurso fundamental para investigadores y desarrolladores en el ámbito del Machine Learning y Data Science [60]. Se utilizará para:

- **Datos**: Proporciona una colección de más de 600 conjuntos de datos utilizados en la investigación y el desarrollo de algoritmos de aprendizaje automático.

- **Accesibilidad**: Es una base de datos gratuita y de acceso abierto, lo que la convierte en un recurso clave para investigadores y estudiantes que buscan conjuntos de datos para pruebas y experimentación sin restricciones.

- **Estandarización**: Los conjuntos de datos disponibles en el repositorio suelen estar bien documentados y estructurados, lo que facilita su uso en estudios comparativos y benchmarking de algoritmos de Machine Learning.

- **Investigación y educación**: Es utilizado en entornos académicos y proyectos de investigación para la evaluación de nuevos modelos y técnicas de Machine Learning.

3.5.4.- El Problema del Billón de Líneas

El Problema del Billón de Líneas, o 1BRC (The One Billion Row Challenge) [61] [62], es un desafío computacional cuyo objetivo es procesar un conjunto de datos masivo con mil millones de filas de la forma más eficiente posible. Este reto de programación fue propuesto por el desarrollador Gunnar Morling en enero de 2024 para tratar de encontrar la forma más rápida de procesar un fichero de gran tamaño utilizando Java como lenguaje de programación, aunque muchos de los participantes en el reto empezaron a hacer pruebas en otros lenguajes.

El fichero CSV proporcionado para este desafío tiene una estructura muy sencilla, cada línea del fichero contiene únicamente 2 valores, una ciudad del mundo y una temperatura. Lo que se pide calcular es, para cada ciudad del fichero cuáles son sus temperaturas mínima, máxima y promedio.

Figura 3.6: Extracto del fichero 1BRC

Por la naturaleza del problema, el 1BRC resulta un ejercicio interesante con el que poder comparar la potencia computacional de diversas implementaciones de un algoritmo que, aunque sencillo, debido al tamaño del fichero de datos, es computacionalmente exigente y, a priori, se puede intuir que requerirá de alguna estrategia para la gestión de memoria.

Capítulo 4
Metodología y resultados

4.1.- PLANIFICACIÓN DEL PROYECTO

El desarrollo del proyecto se ha centrado en diversos modelos de aprendizaje automático para abordar tareas de clasificación, regresión y clustering. Estos modelos se seleccionaron por su relevancia teórica, su aplicabilidad en distintos contextos y su potencial para comparar el rendimiento y la facilidad de uso entre diferentes lenguajes de programación.

Estos modelos constituyen herramientas dentro del campo de la inteligencia artificial, ya que permiten encontrar patrones y realizar predicciones a partir de datos previamente observados. Para la clasificación, se emplearon modelos como el árbol de decisión [63], que estructura la toma de decisiones mediante divisiones jerárquicas simples; el random forest [64], una técnica de ensamblado que mejora la precisión combinando múltiples árboles; las máquinas de vector soporte (SVM) [65], que buscan un hiperplano óptimo para separar las clases; las redes neuronales [66], inspiradas en el funcionamiento del cerebro

humano y capaces de aprender representaciones complejas; y la regresión logística [67], un modelo lineal ampliamente utilizado en clasificación binaria y multiclase. En el caso de la regresión, se utilizó la regresión lineal [68], adecuada para predecir variables continuas bajo el supuesto de una relación lineal entre las variables de entrada y salida. Finalmente, para tareas de aprendizaje no supervisado se aplicó el algoritmo K-Means [69], una técnica de clustering que agrupa los datos en subconjuntos basados en su similitud. La combinación de estos modelos permite analizar distintos enfoques, evaluar su rendimiento y compararlos entre sí en función de su precisión, eficiencia y facilidad de implementación.

Cabe destacar que, aunque este trabajo podría enmarcarse inicialmente dentro del ámbito de la ingeniería informática por el uso de herramientas de programación y desarrollo, en realidad se trata de una investigación orientada a la ciencia de datos. A diferencia de un proyecto de ingeniería del software tradicional, cuyo objetivo principal suele ser el diseño y desarrollo de un sistema o aplicación funcional, este trabajo se centra en el análisis, comparación y evaluación de distintos modelos de aprendizaje automático aplicados sobre datos reales. El enfoque adoptado busca generar conocimiento sobre el comportamiento, la eficiencia y la adecuación de diferentes modelos según el lenguaje de programación utilizado, lo que lo sitúa más cerca del método científico que del desarrollo de software como fin en sí mismo. En este sentido, se ha priorizado la experimentación, la obtención de métricas, la validación cruzada y la interpretación de resultados, características propias de la investigación en ciencia de datos.

Tabla 4.1: Modelos utilizados y librerías correspondientes en Python y Rust

Modelos		Python	Rust
Clasificación	**Arbol de decision**	Scikit-Learn	Smartcore, Linfa
	Random Forest	Scikit-Learn	Smartcore, Linfa
	SVM	Scikit-Learn	Smartcore, Linfa
	Neuronal Network	TensorFlow, PyTorch, Scikit-Learn	---
	Regresión Logística	TensorFlow, PyTorch, Scikit-Learn	Smartcore, Linfa
Regresión		TensorFlow, PyTorch, Scikit-Learn	Smartcore, Linfa
Clustering		TensorFlow, PyTorch, Scikit-Learn	Smartcore, Linfa

En la tabla 4.1 se muestra un resumen con los modelos de aprendizaje automático empleados a lo largo del proyecto, junto con las librerías utilizadas en cada lenguaje de programación. Esta clasificación permite visualizar de forma clara qué modelos se implementan tanto en Python como en Rust, y qué herramientas específicas se usaron en cada caso para su desarrollo y experimentación. La diversidad de librerías empleadas refleja el objetivo del proyecto: comparar no solo los modelos desde el punto de vista del

rendimiento, sino también en términos de facilidad de uso, expresividad del código y soporte en cada ecosistema. Cabe recalcar que se intentó inicialmente utilizar la librería Burn pero se presentaron múltiples dificultades durante su integración, entre ellas problemas de compatibilidad con dependencias (wgpu, bincode) y cambios frecuentes en la API. Además, su documentación aún está en desarrollo, lo cual dificulta considerablemente el aprendizaje y la implementación de modelos funcionales. Por estas razones, se decidió no continuar con Burn en la versión final del proyecto.

En general, el desarrollo con las librerías de Python resultó más accesible gracias a su amplia documentación y comunidad activa. Scikit-Learn ofreció una implementación sencilla y eficiente de los modelos clásicos, permitiendo ejecutar rápidamente experimentos con validación cruzada, aunque no todas las librerías presentaron la misma facilidad. En el caso de TensorFlow, encontramos importantes dificultades a la hora de utilizar modelos como el árbol de decisión, debido a errores relacionados con el archivo “interference.so”(entraba en conflicto la libreria ya que no encontraba este archivo en el ordenador) lo cual imposibilita su uso práctico en esta tarea. Por otro lado, PyTorch, aunque potente para redes neuronales, no proporciona soporte directo para modelos clásicos como árboles de decisión, SVM o Random Forest, lo que limitó su aplicación a tareas de clasificación tradicional. En cuanto a Rust, se trabajó principalmente con las librerías Smartcore y Linfa. Smartcore ofreció un buen rendimiento y soporte para modelos clásicos, mientras que Linfa resultó útil especialmente para clustering. No obstante, ambas presentan una documentación menos extensa y ciertas limitaciones de flexibilidad en comparación con sus equivalentes en Python.

A continuación, se presentan, en pseudocódigo, los algoritmos generales empleados para la implementación de los procesos de clasificación, regresión y clustering, todos ellos estructurados con validación cruzada (excepto clustering), para asegurar una evaluación robusta y coherente de los modelos.

Algoritmo 4.1: Estructura general de los scripts de clasificación con '*cross-validation*'

```
importar librerías necesarias

cargar_datos(archivo)

preprocesar_datos:
      eliminar_nulos()
      codificar_variables_categóricas()
      separar_características_y_etiquetas()
      escalar_características()

inicializar_validación_cruzada(K = número_de_folds)

para cada fold en validación_cruzada:
      dividir_datos_entrenamiento_prueba(fold)

      inicializar_modelo_clasificación()

      tiempo_inicio_entrenamiento = tiempo_actual()
      entrenar_modelo(datos_entrenamiento)
      tiempo_fin_entrenamiento = tiempo_actual()

      tiempo_inicio_predicción = tiempo_actual()
      predicciones = predecir(datos_prueba)
      tiempo_fin_predicción = tiempo_actual()

      calcular_precision(predicciones, etiquetas_reales)
      almacenar_resultados(precisión, tiempo_entrenamiento,
                           tiempo_predicción)

calcular_promedios()
mostrar_resultados_finales()
```

Algoritmo 4.2: Estructura general de los scripts de regresión

```
importar librerías necesarias

cargar_datos(archivo)

preprocesar_datos:
      eliminar_nulos()
      codificar_variables_categóricas()
      separar_características_y_variable_objetivo()
      escalar_características()

inicializar_validación_cruzada(K = número_de_folds)

para cada fold en validación_cruzada:
      dividir_datos_entrenamiento_prueba(fold)

      inicializar_modelo_regresión()

      tiempo_inicio_entrenamiento = tiempo_actual()
      entrenar_modelo(datos_entrenamiento)
      tiempo_fin_entrenamiento = tiempo_actual()

      tiempo_inicio_predicción = tiempo_actual()
      predicciones = predecir(datos_prueba)
      tiempo_fin_predicción = tiempo_actual()

      // MSE, MAE, etc.
      calcular_métricas(predicciones, valores_reales)
      almacenar_resultados(métricas, tiempo_entrenamiento,
                           tiempo_predicción)

calcular_promedios()
mostrar_resultados_finales()
```

Algoritmo 4.3: Estructura general de los scripts de Clustering

```
importar librerías necesarias

cargar_datos(archivo)

preprocesar_datos:
      eliminar_nulos()
      codificar_variables_categóricas()
      escalar_características()

definir_número_de_clústeres(k)

inicializar_modelo_clustering(k)

tiempo_inicio_ajuste = tiempo_actual()
entrenar_modelo(datos)
tiempo_fin_ajuste = tiempo_actual()

mostrar_resultados_finales(métricas, gráficos, tiempo_total)
```

4.2.- DESCRIPCIÓN DE DATOS

En este trabajo se han utilizado dos conjuntos de datos principales, cada uno orientado a diferentes tareas de aprendizaje automático.

4.2.1- Housing.csv

- **Origen:** Este dataset es una versión del conjunto de datos de viviendas de California, disponible originalmente a través de la plataforma Kraggle [70].

- **Tipo de problema:** Apto tanto para regresión como para clustering.

- **Metadatos:**
  - **Número de filas:** 20.640.
  - **Número de columnas:** 10.
  - **Tipo de columnas:**
    - 9 columnas numéricas.
    - 1 columna categórica.
  - **Columnas destacadas:**

        - Variables como total_rooms, median_income, latitude, entre otras, proporcionan datos socioeconómicos y geográficos.
        - Ocean_proximity es una variable categórica que indica la cercanía al océano.
    - **Variable objetivo:** median_house_value, representa el valor medio de la vivienda en la zona

4.2.2- Bank-full.csv

- **Origen:** Este dataset proviene del repositorio para aprendizaje automático de la Universidad de California en Irvine (UCI) [71]. Representa una campaña de marketing telefónico realizada por una entidad bancaria portuguesa.

- **Tipo de problema:** Conjunto de datos para problemas de clasificación binaria, donde el objetivo es predecir si un cliente suscribió (o no) un depósito a plazo.

- **Metadatos:**
    - **Número de filas:** 45.211.
    - **Número de columnas:** 17.
    - **Tipo de columnas:**
        - 7 columnas numéricas.
        - 10 columnas categóricas.
    - **Variable objetivo:** La variable Y toma los valores “yes” o “no”, indicando si el cliente ha aceptado el producto bancario (depósito a plazo).
    - **Características relevantes:** Incluye atributos demográficos, económicos y de interacción.

4.3.- RESULTADOS

Los experimentos fueron realizados en un ordenador personal con las siguientes características técnicas:

- **Sistema operativo**: Windows 11 Home de 64 bits
- **Procesador**: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
    - **Número de núcleos físicos(Cores)**: 4
    - **Hilos(threads)**: 8
- **Memoria RAM**: 16 GB DDR4
- **Almacenamiento**: SSD NVMe de 512 GB
- **GPU integrada**: Intel(R) Iris Xe Graphics

Estas especificaciones deben tenerse en cuenta al comparar los tiempos de entrenamiento y predicción entre los distintos lenguajes y librerías, ya que la capacidad del hardware empleado afecta de forma significativa al rendimiento.

Para garantizar la fiabilidad de los resultados, los experimentos se han ejecutado con el ordenador desconectado de Internet, sin procesos innecesarios en segundo plano y sin ejecutar otras aplicaciones durante las pruebas. Además, cada experimento, se ha repetido 10 veces para minimizar el efecto de posibles fluctuaciones en el rendimiento, y los tiempos reportados corresponden al promedio de dichas repeticiones.

4.3.1- Algoritmos de Clasificación

La tabla 4.2 recoge la comparativa detallada entre distintos algoritmos de clasificación que se han implementado, empleando diversas librerías de aprendizaje automático, tanto en Python como en Rust. Se muestran dos aspectos clave que son el tiempo total de ejecución del algoritmo y la precisión (accuracy) promedio obtenida en las predicciones.

Tabla 4.2: Resultados de clasificación

Modelo / Librería	Tiempo (segundos)	Precisión (%)
Árbol de decisión / Scikit-Learn	**3.0423**	**87.79**
Árbol de decisión / Smartcore	15.8067	87.40
Árbol de decisión / Linfa	38.5215	87.20
Random Forest / Scikit-Learn	**20.9874**	**90.43**
Random Forest / Smartcore	73.3172	90.14
Random Forest / Linfa	45.7543	89.98
SVM / Scikit-Learn	**314.6971**	**90.37**
(*) SVM / Smartcore	1517.8765	90.10
(*) SVM / Linfa	1840.2548	89.15
Redes Neuronales / TensorFlow	19.6358	**90.55**
Redes Neuronales / PyTorch	**2.4250**	86.27
Redes Neuronales / Scikit-Learn	662.2857	88.43
Regresión Logística / TensorFlow	14.6587	89.98
Regresión Logística / PyTorch	12.3574	89.50
Regresión Logística / Scikit-Learn	**0.6137**	**90.13**
Regresión Logística / Smartcore	99.6234	89.48
Regresión Logística / Linfa	6.1031	88.20

En el caso de los árboles de decisión, el modelo implementado con Scikit-Learn alcanza una precisión del 87,79 % con un tiempo de ejecución total muy reducido (3,04 segundos),

lo que lo convierte en una solución altamente eficiente. El código implementa validación cruzada estratificada (StratifiedKFold) con diez particiones, y realiza tanto el entrenamiento como la predicción con gran eficiencia gracias a la robustez de la biblioteca.

En contraste, el modelo equivalente en SmartCore (Rust) obtiene una precisión muy similar (87,40 %) pero con un tiempo de ejecución mucho mayor (15,8067 segundos). Esta diferencia se explica, en parte, por el proceso de codificación categórica manual usando HashMap para transformar texto en valores numéricos sin aprovechar paralelismo o estructuras vectorizadas como en Python. Además, la validación cruzada está implementada manualmente mediante particionado explícito y sin procesamiento multihilo, lo cual impacta negativamente en el rendimiento global. A pesar de que Rust compila a código nativo altamente eficiente, su ecosistema de machine learning aún no está tan optimizado como el de Python para tareas de alto volumen como esta.

Por otro lado, el modelo implementado en Linfa (también en Rust) ofrece una precisión ligeramente inferior (87,20 %) y un tiempo aún mayor (38,5215 segundos). La causa principal es similar: aunque se utiliza la biblioteca ndarray para representar los datos, el proceso de codificación categórica también es manual, y la validación cruzada se ejecuta de forma secuencial. Además, el clasificador DecisionTree de Linfa aún está en desarrollo y no cuenta con técnicas avanzadas de poda o criterios de división más eficientes, lo que podría explicar tanto la leve pérdida de precisión como el tiempo de ejecución superior.

En el caso de los modelos Random Forest, se observa una mejora clara en la precisión respecto a los árboles de decisión simples, alcanzando cifras cercanas o superiores al 90 % en todas las librerías. La implementación en Scikit-Learn destaca por su equilibrio entre rendimiento y eficiencia, logrando un 90,43 % de precisión en apenas 20,98 segundos.

En contraste, la versión desarrollada en SmartCore también alcanza una precisión elevada (90,14 %), pero con un tiempo de ejecución mayor (73,3172 segundos). En este caso, el modelo se configura con 100 árboles y una profundidad máxima de 10, igual que en Scikit-Learn. Sin embargo, la diferencia temporal se explica principalmente por la falta de paralelización y por el proceso manual de validación cruzada, que no está integrado de forma nativa.

Por su parte, la versión en Linfa simula un Random Forest de forma manual. Este enfoque, aunque funcional, es mucho menos eficiente que una implementación nativa. Como resultado, el modelo alcanza una precisión ligeramente inferior (89,98 %) y un tiempo de 45,7543 segundos. El coste computacional se debe a que cada árbol se entrena por separado, sin aprovechar concurrencia, y a que las matrices de datos se reconstruyen en cada iteración. Además, el modelo DecisionTree de Linfa no está optimizado para ser parte de modelos ensamblados, lo que reduce su escalabilidad frente a soluciones más maduras.

En lo relativo a las máquinas de vectores de soporte (SVM), todos los modelos ofrecen una precisión elevada, siendo Scikit-Learn ligeramente superior con un 90,37 %. Sin embargo, incluso en esta librería, el tiempo de ejecución es elevado (314,70 s), debido a que las SVM tienen una complejidad cuadrática o cúbica respecto al número de muestras [65]. En el caso de SmartCore y Linfa, los tiempos son aún mayores (1517 s y 1840 s respectivamente), lo que probablemente se deba a la falta de kernels altamente optimizados o estrategias de reducción de dimensiones que sí están presentes en Scikit-Learn. A esto se suma el hecho de que algunos solucionadores de optimización utilizados en Rust pueden ser más generales y no tan específicos para SVM, lo que aumenta el coste computacional.

Además, cabe destacar que en la tabla comparativa, los resultados obtenidos con SmartCore y Linfa(*) han sido marcados en otro color, ya que en estos casos no se utilizó validación cruzada. Esto se debe a que, tras varios intentos, el coste computacional de aplicar este tipo de validación con los modelos SVM en Rust resultó excesivamente alto, provocando tiempos de ejecución poco manejables dentro del contexto del proyecto. Por ello, se optó por realizar una única partición para obtener resultados representativos, aunque no directamente comparables con el enfoque validado con K-Fold en Python.

En el caso de las máquinas de vectores de soporte (SVM), los tres enfoques obtuvieron una precisión alta, superando el 89 %. La implementación con Scikit-Learn alcanzó un 90,37 % de precisión, aunque con un tiempo de ejecución elevado (314,70 segundos). Esto se debe al tipo de configuración usada, que aunque precisa, es más lenta, especialmente con muchos datos y variables. Aun así, Scikit-Learn gestiona bien estos casos gracias a su optimización interna. También se aplicó un escalado previo de los datos, lo cual es importante para que este modelo funcione correctamente.

La implementación en SmartCore obtiene una precisión muy similar (90,10 %), aunque no realiza validación cruzada ni separación en conjuntos de entrenamiento y prueba, evaluando el modelo directamente sobre todos los datos. El tiempo de ejecución, sin embargo, asciende a 1517,88 segundos, a pesar de usar un kernel lineal, mucho más simple computacionalmente. Esta diferencia sugiere una falta de optimización interna en el solucionador de SmartCore, o una sobrecarga derivada del procesamiento no vectorizado y sin normalización previa de los datos.

La versión en Linfa es inferior a la de Scikit-Learn tanto en precisión (89,15 %) como en tiempo (1840,25 segundos). Esta diferencia se explica por varias razones: primero, la validación cruzada se implementa manualmente, y segundo, la implementación de SVM en Linfa todavía está en desarrollo y carece de optimizaciones como soporte GPU o procesamiento paralelo.

En el caso de las redes neuronales, las tres implementaciones muestran diferencias notables tanto en precisión como en tiempos de ejecución, reflejando el grado de optimización de

cada librería para este tipo de tareas. La implementación con TensorFlow es la que ofrece el mejor equilibrio, alcanzando una precisión del 90,55 % en un tiempo total de tan solo 19,64 segundos. TensorFlow está altamente optimizado para acelerar los cálculos mediante compilación previa y posible uso de GPU, incluso en configuraciones por defecto.

Por otro lado, PyTorch, obtiene un tiempo de ejecución muy bajo (2,43 segundos) pero con una precisión menor (86,27 %). Esta diferencia se debe a varios factores: el código utiliza una red con menos neuronas y una única salida con función sigmoid, adecuada para clasificación binaria, pero más sensible al desbalance de clases.

En contraste, la red neuronal implementada con Scikit-Learn alcanza una precisión intermedia (88,43 %), pero presenta un tiempo total muy superior (662,29 segundos). Esta diferencia temporal se debe a que el modelo MLPClassifier de Scikit-learn, aunque funcional, no está diseñado para grandes volúmenes de datos ni se beneficia de aceleración por hardware como GPU.

La regresión logística, como modelo lineal simple, ofrece muy buenos resultados tanto en precisión como en tiempo de ejecución en todas las implementaciones evaluadas. La versión de Scikit-Learn alcanza una precisión del 90,13 % con un tiempo total de ejecución extremadamente bajo (0,61 segundos), lo que la convierte en la opción más eficiente de toda la tabla. Este rendimiento se debe a que LogisticRegression está altamente optimizado.

La implementación con TensorFlow también alcanza una precisión elevada (89,98 %) y un tiempo bajo (14,66 segundos), lo que demuestra que incluso un modelo lineal puede beneficiarse de las optimizaciones internas de la librería. El modelo utilizado corresponde a una red neuronal con una sola neurona y activación sigmoide, equivalente matemáticamente a una regresión logística binaria.

En PyTorch, la precisión se mantiene en 89,50 %, y el tiempo de ejecución total es también bajo (12,36 segundos). El modelo es idéntico en estructura al de TensorFlow, pero el entrenamiento incluye una estrategia de early stopping manual basada en la evolución de la función de pérdida, lo que impide el sobreentrenamiento innecesario.

Por otro lado, la versión desarrollada en SmartCore también es funcional y consigue una precisión competitiva (89,48 %), pero con un tiempo total significativamente mayor (99,6234 segundos). A pesar de que se utiliza un modelo lineal sin regularización adicional, la validación cruzada está implementada de forma manual, sin paralelización ni estructuras vectorizadas, y el preprocesamiento es mucho más básico. La codificación de variables se hace manualmente usando HashMap y los datos se procesan secuencialmente en cada fold, lo que introduce un gran coste computacional incluso para un modelo simple.

Estas limitaciones técnicas explican la gran diferencia temporal frente a las alternativas en Python.

Finalmente, la implementación en Linfa ofrece una precisión algo menor (88,20 %) con un tiempo intermedio (6,1031 segundos). Al igual que en SmartCore, la validación cruzada se implementa de forma manual y no hay codificación one-hot, lo que puede afectar tanto la calidad del modelo como la velocidad de entrenamiento.

4.3.2- Algoritmos de Regresión

En la tarea de regresión, se observan diferencias notables en los errores de predicción y tiempos de ejecución entre las distintas librerías. La implementación en Scikit-Learn destaca como la más eficiente, alcanzando un Mean Squared Error (MSE) de 4.741×10^9 y un Mean Absolute Error (MAE) de 49.818, con un tiempo de ejecución extremadamente bajo (0.057 s). Esta eficiencia se debe a que la regresión lineal en Scikit-Learn está altamente optimizada, usando operaciones vectorizadas sobre arrays numpy. Además, el modelo no requiere entrenamiento iterativo como ocurre en redes neuronales, por lo que su tiempo es prácticamente instantáneo.

Tabla 4.3: Resultados de regresión

Librería	Tiempo (seg.)	MAE	MSE
TensorFlow	10.6322	62,196.66	7,256,782,387.20
PyTorch	18.6831	51,283.43	5,751,248,384.00
Scikit-Learn	**0.0574**	**49,817.91**	**4,741,574,100.86**
SmartCore	0.1710	52,357.70	5,093,998,819.96
Linfa	0.2062	52,143.10	5,143,934,219.29

En cuanto a PyTorch, se logra un rendimiento notable en precisión (MAE: 51.283, MSE: 5.751×10^9), aunque el tiempo total de ejecución se eleva a 18,68 segundos. El modelo utilizado es una red neuronal multicapa (MLP) con dos capas ocultas (64 y 32 neuronas), entrenada durante 1000 épocas con el optimizador Adam. Aunque esto permite capturar relaciones no lineales, el coste computacional es mayor debido al entrenamiento por mini-lotes y la naturaleza iterativa del proceso. A pesar de ello, el modelo alcanza resultados comparables a la regresión lineal clásica, demostrando su capacidad incluso sin ajuste de hiperparámetros.

TensorFlow, por su parte, ofrece un modelo similar en arquitectura al de PyTorch, también con 1000 épocas y dos capas ocultas, alcanzando un MAE de 62.196 y MSE de 7.257×10^9, con un tiempo algo menor (10,63 segundos). La diferencia en error puede deberse al orden en que se presentan los datos o al control interno del proceso de entrenamiento (por ejemplo, diferente inicialización de pesos o falta de early stopping). No obstante,

TensorFlow demuestra un excelente rendimiento para tratarse de un modelo no lineal con un dataset tabular, y su optimización interna permite un tiempo razonable considerando la cantidad de parámetros entrenables.

Las implementaciones en Rust, si bien funcionales y competitivas, muestran un mayor tiempo de ejecución con ligeras diferencias en precisión. En el caso de SmartCore, se obtiene un MAE de 52.358 y un MSE de 5.094×10^9, con un tiempo total de 0,17 segundos. Este resultado es notable teniendo en cuenta que la implementación usa regresión lineal clásica sin optimizaciones como normalización o one-hot encoding, y que la validación cruzada se ejecuta de forma manual, sin paralelismo ni aceleración por hardware. A pesar de ello, el rendimiento es superior al esperado para una librería más joven, lo cual la convierte en una opción prometedora.

Finalmente, la versión en Linfa también emplea regresión lineal y alcanza un MAE de 52.143 y un MSE de 5.144×10^9, con un tiempo reducido (0,21 segundos), lo cual es destacable. En este caso, el preprocesamiento se maneja directamente sobre arrays ndarray, y la validación cruzada se implementa manualmente con concatenaciones entre subconjuntos. Aunque el modelo carece de mecanismos de regularización o normalización automática, los resultados son aceptables y muestran el potencial de Linfa para tareas de regresión simple, siempre que se optimicen las fases de entrada y validación.

4.3.3- Algoritmos de Clustering

El clustering es un tipo de procesamiento descriptivo, en este caso, al no haber predicción ni datos etiquetados, no tiene sentido aplicar fase de entrenamiento, fase de test, ni tampoco validación cruzada. En este tipo de problemas simplemente se ejecuta el conocido algoritmo K-Means disponible en cada una de las librerías estudiadas, o bien una implementación de dicho algoritmo usando algunas funciones y métodos de dichas librerías.

Tal y como se observa en la Tabla 4.4, existen diferencias notables en los tiempos de ejecución entre las distintas librerías. En general, PyTorch es la más rápida en todos los escenarios evaluados, seguida por Linfa y TensorFlow. En cambio, las implementaciones en Scikit-Learn y SmartCore, presentan tiempos significativamente más elevados, lo que puede deberse a la falta de optimización en los algoritmos internos o a la gestión manual de ciertas operaciones como la inicialización de centroides o la convergencia. En principio se pensaba que Scikit-Learn sería la que nos daría los mejores resultados.

Además, se observa que a medida que se incrementa el número de variables y de clústeres, el tiempo de ejecución aumenta en todas las librerías, siendo esta tendencia especialmente

pronunciada en SmartCore, donde el coste computacional se dispara al aumentar la dimensionalidad.

Tabla 4.4: Resultados de clustering

		Tiempos (seg.)				
Nº var	**K**	**TensorFlow**	**PyTorch**	**Scikit-Learn**	**SmartCore**	**Linfa**
2	2	0.0865	**0.0197**	0.2183	0.4492	0.0238
2	3	0.0935	**0.0221**	0.2461	0.4931	0.0287
2	5	0.0987	**0.0243**	0.2641	0.5715	0.0312
5	2	0.1023	**0.0232**	0.2576	0.6518	0.0387
5	3	0.1157	**0.0285**	0.2825	0.7037	0.0498
5	5	0.1276	**0.0311**	0.3631	0.8371	0.0685
7	2	0.1568	**0.0293**	0.2870	0.7908	0.0784
7	3	0.1583	**0.0384**	0.3154	0.8583	0.0846
7	5	0.1623	**0.0435**	0.3865	1.3253	0.1856

Para facilitar la interpretación de los resultados, se han generado visualizaciones en dos dimensiones a partir de los resultados del algoritmo K-Means con distintos valores de k. En estas gráficas, los datos se han reducido a dos variables normalizadas, y se representan los puntos coloreados según el clúster al que fueron asignados.

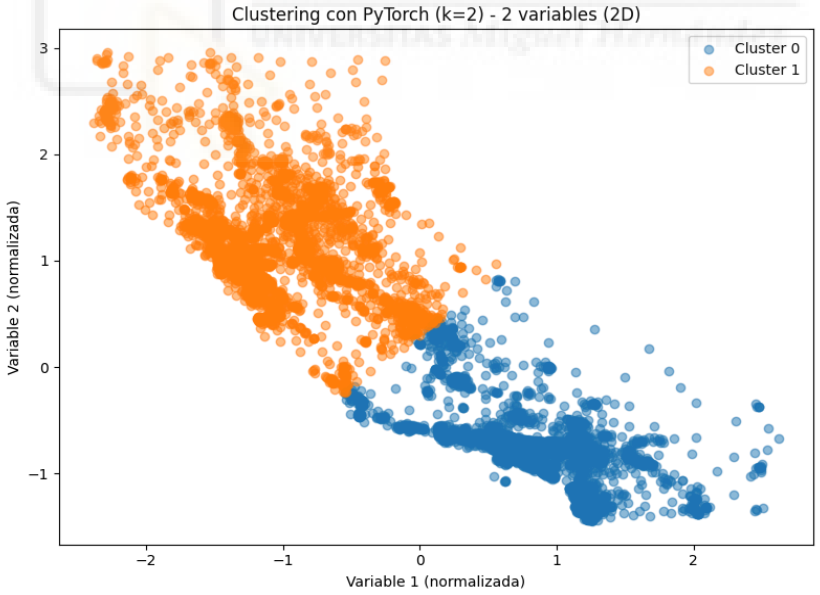

Figura 4.1: Gráfica Pytorch 2 variables y 2 clusters (k=2).

En esta primera visualización (figura 4.1) se puede observar que los datos se dividen claramente en dos grupos bien definidos. La separación entre los clústeres es nítida, lo que sugiere que existe una estructura natural binaria en el conjunto de datos. Este resultado es coherente con situaciones donde las variables muestran una fuerte correlación o cuando hay una tendencia dominante que permite agrupar los puntos con facilidad. Es un ejemplo

representativo del buen rendimiento de K-Means en contextos donde la forma de los datos es aproximadamente lineal y compacta.

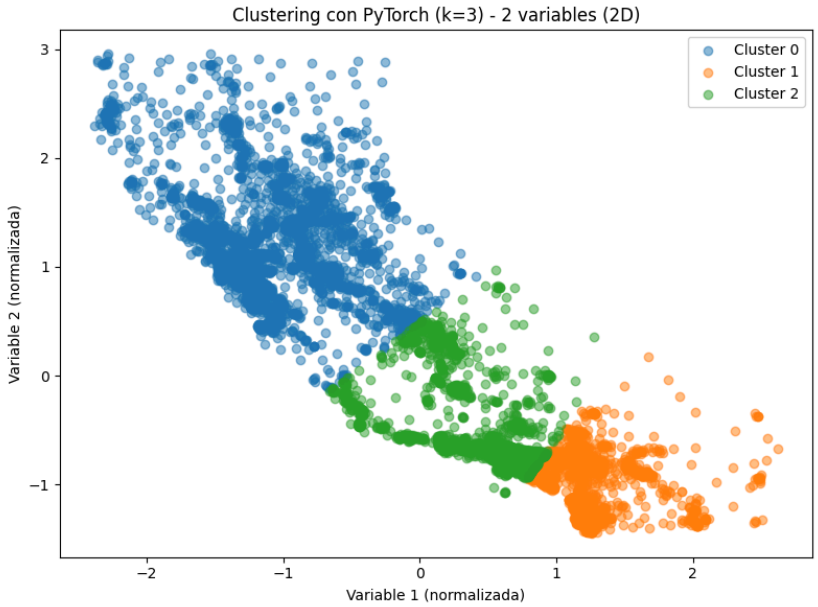

Figura 4.2: Gráfica Pytorch 2 variables y 3 clusters (k=3).

Para k=3 (figura 4.2), el algoritmo refina la segmentación, introduciendo un grupo intermedio. Se distingue mejor la variabilidad del conjunto de datos. Sin embargo, se observa solapamiento en las fronteras de los clústeres, lo que indica que el modelo podría forzar divisiones no naturales.

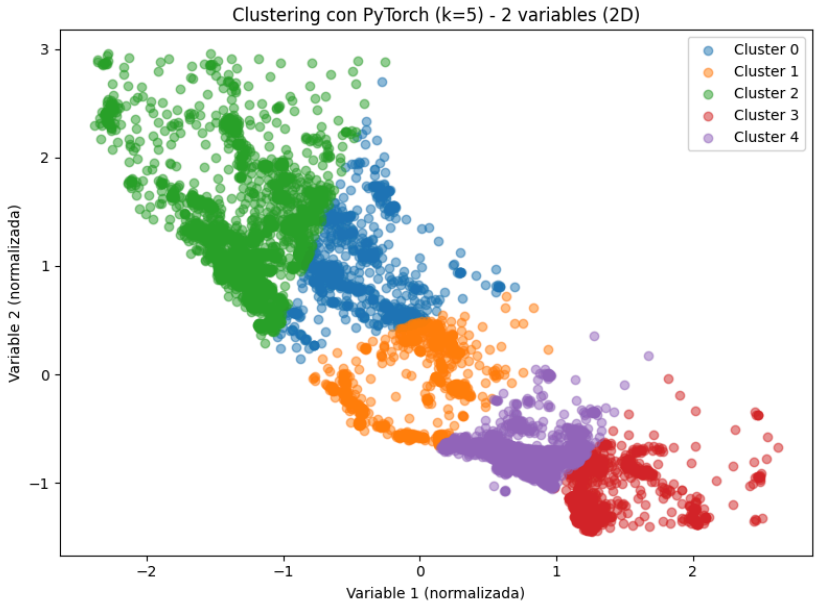

Figura 4.3: Gráfica Pytorch 2 variables y 5 clusters (k=5).

Con cinco clústeres, la segmentación se vuelve más granular (figura 4.3). La visualización muestra cómo el algoritmo reparte los datos en subconjuntos más pequeños, algunos de ellos claramente definidos, mientras que otros aparecen parcialmente solapados. Esta configuración puede ser útil si se busca una clasificación más específica, aunque también conlleva el riesgo de fragmentar grupos naturales, especialmente si no hay una justificación fuerte para utilizar un valor de *k* tan alto. Este caso permite observar cómo la complejidad del modelo puede impactar en la interpretación de los resultados.

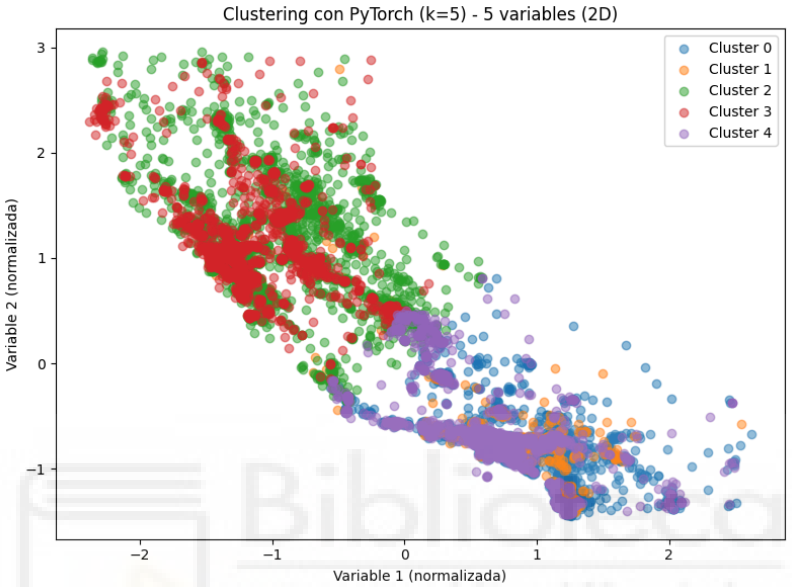

Figura 4.4: Gráfica Pytorch 5 variables y 5 clusters (k=5).

En la última visualización (figura 4.4) se mantuvo el mismo número de clústeres que en el caso anterior (k=5), pero se aumentó el número de variables consideradas. Aunque la proyección es en dos dimensiones, los datos se agruparon utilizando cinco variables del dataset original. Se puede observar una mayor dispersión y solapamiento entre los puntos, lo cual es esperable cuando se reduce información de un espacio de mayor dimensión a solo dos componentes. Esta representación refleja los desafíos de interpretar clustering en espacios de alta dimensionalidad y la importancia de aplicar técnicas de reducción o visualización adecuadas.

4.3.4- Resultados del 1BRC

Para abordar el reto se implementaron dos versiones del mismo algoritmo: una en Python y otra en Rust. En ambos casos se procesó el archivo línea a línea para evitar cargar todo el contenido en memoria, y se utilizó una estructura clave-valor para almacenar los resultados acumulados por estación

Algoritmo 4.4: Estructura general del algoritmo 1BRC

```
inicializar estructura resultado vacía

abrir archivo de texto en modo lectura

para cada línea en el archivo:
      dividir línea en estación y temperatura
      si la estación ya está en la estructura:
      actualizar mínimo si corresponde
      actualizar máximo si corresponde
      sumar temperatura al total
      incrementar contador
      si no está:
      crear nuevo registro con min, max, suma y contador

para cada estación:
      calcular promedio = suma / contador
```

Este pseudocódigo fue implementado siguiendo la misma lógica tanto en Python como en Rust. En ambos lenguajes se utilizó el mismo archivo de entrada con mil millones de líneas (<ciudad ; temperatura>). Para obtener resultados consistentes, se realizaron diez ejecuciones por lenguaje y se calculó la media del tiempo total de ejecución.

Tabla 4.5: Resultados del problema 1BRC

Lenguaje	Tiempo (segundos)
Python	1110.5289
Rust	**153.8386**

Estos resultados reflejan una clara ventaja de Rust en cuanto a eficiencia, reduciendo el tiempo total en aproximadamente un 83 % respecto a Python. Esto se debe en parte a la mayor velocidad de ejecución nativa del lenguaje, así como a la gestión eficiente de memoria y a la ausencia de un recolector de basura.

A pesar de utilizar estructuras conceptualmente similares, la diferencia se hace evidente en tareas de este tamaño. Rust también ofrece mayor control sobre las operaciones de bajo nivel, como el manejo del buffer de lectura, lo que repercute directamente en el rendimiento global del programa.

Este comportamiento sugiere que, a medida que se incrementa el tamaño del conjunto de datos, la ventaja de Rust podría volverse aún más significativa. Su compilación a código máquina, junto con una gestión explícita de la memoria y estructuras eficientes lo convierten en una opción robusta para aplicaciones que requieren alto rendimiento en escenarios de Big Data.

Dado el tamaño extremadamente grande del archivo utilizado en el reto 1BRC (mil millones de líneas), se planteó en Python una versión del algoritmo basada en el procesamiento por bloques (*chunks*), en lugar de línea a línea o cargando el archivo completo en memoria. Esta decisión surgió por motivos prácticos: al intentar cargar un volumen de datos tan grande directamente en un DataFrame o en estructuras completas de Python, el sistema comenzaba a ralentizarse notablemente o incluso a agotar la memoria disponible. En cambio, al leer y procesar los datos en bloques de líneas de tamaño fijo, se mantuvo un uso de memoria más estable y se logró controlar mejor el avance del procesamiento. Esta técnica no se aplicó en Rust, ya que allí el procesamiento ya se realizaba línea a línea de forma eficiente, y el uso de memoria se mantuvo controlado gracias a la optimización nativa del lenguaje. Sin embargo, la versión por chunks en Python resultó especialmente útil para poder trabajar con datasets masivos en un entorno con recursos limitados, permitiendo dividir el problema en partes más pequeñas y manejables sin comprometer la precisión de los resultados.

Algoritmo 4.5: Estructura general del algoritmo 1brc con *chunks*

```
inicializar diccionario resultado vacío
definir tamaño del chunk (número de líneas por bloque)
inicializar contador de chunks

abrir archivo de texto en modo lectura

para cada línea en el archivo:
      añadir línea al chunk actual
      si el chunk alcanza el tamaño definido:
      incrementar contador de chunk
      mostrar mensaje del chunk que se está procesando

      para cada línea en el chunk:
            dividir línea en estación y temperatura
            si estación ya está en resultado:
                  actualizar mínimo, máximo, suma y contador
            si no está:
                  crear nuevo registro con min, max, suma y
contador
      vaciar el chunk
si queda un chunk incompleto al final:
      procesarlo de la misma forma

para cada estación en resultado:
      calcular promedio = suma / contador

```

Aunque la hipótesis del procesamiento por bloques resultaba prometedora, especialmente para evitar sobrecargas de memoria, los resultados obtenidos no fueron los esperados. Tras

diez ejecuciones, el tiempo medio de la versión en Python que procesaba el archivo por *chunks* fue de 1399,21 segundos, lo cual representa un incremento significativo respecto a la versión estándar (1110,52 s). Este resultado sugiere que, si bien el procesamiento por bloques facilita la organización del código y permite controlar el uso de memoria, también introduce cierta sobrecarga computacional al manejar estructuras intermedias, especialmente en un lenguaje interpretado como Python. Por tanto, aunque útil en situaciones con memoria limitada, esta estrategia no logró mejorar el rendimiento global en términos de tiempo de ejecución.

Capítulo 5
Conclusiones y trabajo futuro

5.1.- CONCLUSIONES

A lo largo de este trabajo se ha llevado a cabo una comparativa entre los lenguajes de programación Python y Rust en el contexto del desarrollo de modelos de inteligencia artificial, abarcando tareas de clasificación, regresión y clustering. El objetivo ha sido evaluar cuál de los dos lenguajes ofrece un mejor equilibrio entre rendimiento, facilidad de uso y soporte de librerías, aportando así una visión crítica y actualizada sobre su aplicabilidad real en el ámbito del aprendizaje automático.

Los resultados muestran que Python sigue siendo la opción preferida para la mayoría de proyectos de IA, gracias a su sintaxis sencilla, su gran comunidad de usuarios y un ecosistema muy consolidado de librerías como Scikit-learn. Estas herramientas no solo facilitan el desarrollo, sino que también proporcionan un alto grado de fiabilidad y precisión en los resultados.

En cambio, Rust ha demostrado tener un enorme potencial como lenguaje para Machine Learning, especialmente en términos de rendimiento, eficiencia y seguridad en la gestión de memoria. Sin embargo, las librerías actuales para aprendizaje automático en Rust aún están verdes y presentan claras limitaciones. Muchas de ellas parecen estar en fases tempranas de desarrollo, con APIs poco estables, documentación escasa y una comunidad aún pequeña, lo que dificulta la implementación de modelos complejos y reduce la productividad.

A esto se suma que la curva de aprendizaje de Rust es muy pronunciada, lo que representa una barrera adicional tanto para nuevos usuarios como para desarrolladores acostumbrados a lenguajes más accesibles como Python. Esta dificultad se ha hecho especialmente evidente durante la implementación práctica, donde la falta de recursos y ejemplos ha ralentizado el avance del trabajo en comparación con Python.

No obstante, Rust ha demostrado ser especialmente útil en contextos donde se manejan volúmenes masivos de datos, como ha quedado evidenciado en la implementación del reto *1 Billion Row Challenge (1BRC)*. En este tipo de tareas, donde la eficiencia y el rendimiento marcan una diferencia significativa, Rust ha ofrecido tiempos de ejecución mucho menores que Python, evidenciando su capacidad para exprimir al máximo los recursos del sistema y procesar grandes archivos CSV de forma rápida y segura. Este tipo de escenarios podrían ser clave para su adopción futura en proyectos de Machine Learning a gran escala.

En resumen, puede afirmarse que Python sigue siendo la mejor opción para quienes buscan resultados rápidos, facilidad de uso y una gran variedad de recursos listos para su uso en producción. No obstante, Rust podría convertirse en una opción muy potente en el futuro, especialmente en entornos donde el rendimiento y la seguridad sean prioritarios, siempre y cuando su ecosistema de librerías continúe madurando y se reduzca su complejidad.

Para facilitar la consulta y la reproducción de los experimentos realizados en este Trabajo de Fin de Grado, he habilitado un repositorio público en GitHub con todo el código fuente utilizado. El repositorio está disponible en la siguiente dirección:

https://github.com/FrancescBaezaT/tfg

En él se incluyen los scripts organizados por lenguaje y tipo de tarea (clasificación, regresión, clustering), así como un archivo README.me con instrucciones detalladas para ejecutar cada experimento. También se proporciona información adicional sobre cómo descargar el archivo utilizado en el reto *1BRC* (1 Billion Row Challenge), incluyendo el enlace y los pasos necesarios para procesarlo correctamente.

5.2.- TRABAJO FUTURO

Como posibles líneas futuras de trabajo, se proponen varias direcciones que permitirían enriquecer y ampliar los resultados obtenidos. En primer lugar, sería interesante realizar comparativas similares con otros lenguajes de programación que también cuenten con librerías para aprendizaje automático, como Julia, R, o incluso Go. Esto permitiría tener una visión más amplia sobre el estado actual del desarrollo de IA en distintos entornos tecnológicos.

Por otro lado, podría explorarse el desarrollo de modelos de machine learning en Rust más optimizados, buscando técnicas de paralelización, uso de GPUs o integración con librerías en C/C++ para acelerar los cálculos, con el objetivo de reducir la brecha de rendimiento frente a las soluciones consolidadas en Python. También se podrían construir wrappers o bindings más amigables sobre las librerías existentes para simplificar su adopción por parte de nuevos usuarios.

Otra línea prometedora sería explorar arquitecturas de deep learning más complejas utilizando Rust, en especial cuando librerías como Burn alcancen mayor madurez. Esta experimentación podría extenderse a tareas como visión por computador, procesamiento de lenguaje natural o series temporales, donde la eficiencia y el control que ofrece Rust podrían marcar una diferencia significativa.

Además, se podría estudiar el impacto del uso combinado de Rust y Python mediante bindings, permitiendo aprovechar la rapidez de Rust en las partes críticas del código mientras se mantiene la facilidad de uso de Python para la lógica general y el preprocesamiento. Esta estrategia híbrida podría ofrecer una solución realista para entornos donde rendimiento y productividad deben convivir.

Finalmente, se propone como trabajo complementario la creación de entornos Docker o notebooks preconfigurados que permitan a otros investigadores o estudiantes ejecutar fácilmente las pruebas y comparar sus resultados, fomentando así la reproducibilidad científica y la colaboración abierta en el estudio del rendimiento en Machine Learning.

Bibliografía

[1] ¿Qué es la inteligencia artificial (IA)?
https://planderecuperacion.gob.es/noticias/que-es-inteligencia-artificial-ia-prtr
Fecha de consulta 9 Noviembre 2024

[2] Los mejores lenguajes de programación para inteligencia artificial
https://lovtechnology.com/los-mejores-lenguajes-de-programacion-para-inteligencia-artificial/
12 Noviembre 2024

[3] Los 10 mejores lenguajes de programación de IA: guia para iniciación de principiantes
https://www.datacamp.com/es/blog/ai-programming-languages
12 Noviembre 2024

[4] Python vs Julia: which is the best language for Data Science
https://datascientest.com/en/python-vs-julia-which-is-the-best-language-for-data-science
13 Noviembre 2024

[5] El estado de la IA en 2022 y el balance de media década.
https://www.mckinsey.com/featured-insights/destacados/el-estado-de-la-ia-en-2022-y-el-balance-de-media-decada/es
13 Noviembre 2024

[6] La demanda de software para plataformas de IA crecerá un 40% anual durante los próximos cuatro años.
https://www.computerworld.es/article/3479406/la-demanda-de-software-para-plataformas-de-ia-crecera-un-40-anual-durante-los-proximos-cuatro-anos.html
13 Noviembre 2024

[7] 6 Useful Programming Languages for Data Science you should learn
https://www.analyticsvidhya.com/blog/2019/06/6-useful-programming-languages-data-science-r-python/
13 Noviembre 2024

[8] The Dark Side of Python: Why it's not the Silver Bullet of programming
https://python.plainenglish.io/the-dark-side-of-python-why-its-not-the-silver-bullet-of-programming-d58889048b52
13 Noviembre 2024

[9] Why Rust is the most admired language among developers
https://github.blog/developer-skills/programming-languages-and-frameworks/why-rust-is-the-most-admired-language-among-developers/
13 Noviembre 2024

[10] Rust vs Python: A comparative guide for Data Scientist
https://exaloop.io/blog/rust-vs-python/
13 Noviembre 2024

[11] Breve historia de Rust, el lenguaje de programación que ha destronado a C
https://www.technologyreview.es//s/15106/breve-historia-de-rust-el-lenguaje-de-programacion-que-ha-destronado-c
11 Noviembre 2024

[12] ¿Qué es Rust?
https://learn.microsoft.com/es-es/training/modules/rust-introduction/2-rust-overview
11 Noviembre 2024

[13] ¿Qué es Rust?
https://openwebinars.net/blog/que-es-rust/
11 Noviembre 2024

[14] Rust como un Lenguaje de Programación Orientado a Objetos
https://book.rustlang-es.org/ch17-00-oop
11 Noviembre 2024

[15] Métodos de Gestión de Memoria
https://google.github.io/comprehensive-rust/es/memory-management/approaches.html#speaker-notes
11 Noviembre 2024

[16] Variable and mutability
https://doc.rust-lang.org/book/ch03-01-variables-and-mutability.html
11 Noviembre 2024

[17] Instalación
https://book.rustlang-es.org/ch01-01-installation
11 Noviembre 2024

[18] Rust: Introducción a la gestión de memoría
https://danigm.net/rust-borrow.html
11 Noviembre 2024

[19] Almacenando texto codificado en UTF-8 con Strings
https://book.rustlang-es.org/ch08-02-strings
11 Noviembre 2024

[20] ¿Qué es Python?
https://aws.amazon.com/es/what-is/python/
22 Noviembre 2024

[21] Historia de Python: Desde sus orígenes hasta su impacto actual
https://codigoespinoza.com/blog/python_historia.html?utm_source=chatgpt.com
22 Noviembre 2024

[22] Historia de Python
https://www.wikiwand.com/es/articles/Historia_de_Python
22 Noviembre 2024

[23] Historia de Python
https://www.campustraining.es/curso-programacion-python/historia/?utm_source=chatgpt.com
22 Noviembre 2024

[24] What IDE to use for Python?
https://stackoverflow.com/questions/81584/what-ide-to-use-for-python
25 Noviembre 2024

[25] Rust in Visual Studio Code
https://code.visualstudio.com/docs/languages/rust
6 Diciembre 2024

[26] 5 Best VS Code extensions for Rust development
https://srodrigoroyo.com/best-vs-code-extensions-for-rust-development/
8 Diciembre 2024

[27] Rust - IntelliJ Idea
https://www.jetbrains.com/help/idea/2024.3/rust-plugin.html
10 Diciembre 2024

[28] Rust - CLion 2024.3
https://www.jetbrains.com/help/clion/2024.3/rust-plugin.html
10 Diciembre 2024

[29] Tutorial de PyCharm: Características, instalación y usos
https://platzi.com/blog/pycharm/
10 Diciembre 2024

[30] PyCharm: Definición, usos y características
https://www.inabaweb.com/pycharm-definicion-usos-y-caracteristicas
10 Diciembre 2024

[31] Bienvenidos a la documentación de Spyder
https://docs.spyder-ide.org/5/es/index.html
10 Diciembre 2024

[32] ¿Qué es Python Spyder IDE y cómo usarlo?
https://www.devzv.com/es/python-spyder-ide.html
10 Diciembre 2024

[33] JupyterLab Docs
https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html
10 Diciembre 2024

[34] JupyterLab: La evolución de Jupyter Notebook
https://doc.hpc.iter.es/2023.03/software/how_to_jupyter_lab
10 Diciembre 2024

[35] Librería oficial de Python
https://pypi.org
10 Diciembre 2024

[36] Librería oficial de Rust
https://crates.io/
10 Diciembre 2024

[37] Blazingly Fast DataFrame Library
https://docs.pola.rs
2 Febrero 2025

[38] Machine Learning in Rust, Smarcore
https://medium.com/swlh/machine-learning-in-rust-smartcore-2f472d1ce83
29 Noviembre 2024

[39] Burn
https://burn.dev/docs/burn/
29 Noviembre 2024

[40] Candle vs Burn: Comparing Rust Machine Learning frameworks
https://medium.com/@athan.seal/candle-vs-burn-comparing-rust-machine-learning-frameworks-4dbd59c332a1
29 Noviembre 2024

[41] Machine learning in Rust using Linfa
https://blog.logrocket.com/machine-learning-in-rust-using-linfa/?utm_source=chatgpt.com
30 Noviembre 2024

[42] Pandas User Guide
https://pandas.pydata.org/docs/user_guide/index.html
2 Febrero 2025

[43] What is NumPy?
https://numpy.org/doc/2.2/user/whatisnumpy.html
2 Febrero 2025

[44] ¿Qué es TensorFlow?
https://www.incentro.com/es-ES/blog/que-es-tensorflow
29 Noviembre 2024

[45] Página oficial de TensorFlow
https://www.tensorflow.org/?hl=es
29 Noviembre 2024

[46] ¿Qué es PyTorch?
https://www.ibm.com/es-es/topics/pytorch
29 Noviembre 2024

[47] Scikit-learn, herramienta básica para el data science en python
https://www.master-data-scientist.com/scikit-learn-data-science/
29 Noviembre 2024

[48] ¿Qué es la ciencia de datos?
https://www.ibm.com/es-es/topics/data-science
26 Diciembre 2024

[49] Las mejores bibliotecas de Python para el aprendizaje automático
https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/
4 Febrero 2025

[50] 9 Best Python Libraries for Machine Learning
https://www.coursera.org/articles/python-machine-learning-library
4 Febrero 2025

[51] Awesome-Rust-MachineLearning
https://github.com/vaaaaanquish/Awesome-Rust-MachineLearning
4 Febrero 2025

[52] timeit – Measure execution time of small code
https://docs.python.org/3/library/timeit.html#module-timeit
4 Febrero 2025

[53] The Python Profilers
https://docs.python.org/3/library/profile.html
4 Febrero 2025

[54] Rust-analyzer
https://marketplace.visualstudio.com/items?itemName=rust-lang.rust-analyzer
4 Febrero 2025

[55] Python
https://marketplace.visualstudio.com/items?itemName=ms-python.python
4 Febrero 2025

[56] Python Extension Pack
https://marketplace.visualstudio.com/items?itemName=donjayamanne.python-extension-pack
4 Febrero 2025

[57] GitHub Copilot
https://marketplace.visualstudio.com/items?itemName=GitHub.copilot
4 Febrero 2025

[58] Acerca de Git
https://docs.github.com/es/get-started/using-git/about-git
4 Febrero 2025

[59] Kaggle - Todo lo que tienes que saber
https://www.youtube.com/watch?v=Qt-pN9CqJeo
4 Febrero 2025

[60] Espacio de recursos de ciencia de datos
https://datascience.recursos.uoc.edu/es/repositorio-uci-ml/
4 Febrero 2025

[61] The One Billion Row Challenge
https://www.morling.dev/blog/one-billion-row-challenge/
4 Febrero 2025

[62] Repositorio 1BRC en GitHub (one-billion-row-challenge)
https://github.com/gunnarmorling/1brc
4 Febrero 2025

[63] Decision Trees
https://scikit-learn.org/stable/modules/tree.html
30 Marzo 2025

[64] RandomForestClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
30 Marzo 2025

[65] Support Vector Machines
https://scikit-learn.org/stable/modules/svm.html
30 Marzo 2025

[66] ¿Qué son las redes neuronales?
https://www.ibm.com/es-es/topics/neural-networks
30 Marzo 2025

[67] ¿Qué es la regresión logística?
https://www.ibm.com/mx-es/topics/logistic-regression
30 Marzo 2025

[68] ¿Qué es la regresión lineal?
https://www.ibm.com/es-es/topics/linear-regression
30 Marzo 2025

[69] What is clustering?
https://www.ibm.com/think/topics/clustering
30 Marzo 2025

[70] California Housing Price
https://www.kaggle.com/datasets/camnugent/california-housing-prices
30 Marzo 2025

[71] Bank Marketing
https://archive.ics.uci.edu/dataset/222/bank+marketing
30 Marzo 2025