

**Universidad Tecnológica de La Habana**

**“José Antonio Echeverría”**

**CUJAE**

**Facultad de Ingeniería Informática**

Practicas II

**Tema: Módulo de detección de anomalías**

**Autor: Fabio Ford Campbell (fford@ceis.cuaje.edu.cu)**

**Tutor: Dra.C Nayma Cepero Pérez (ncepero@ceis.cuaje.edu.cu)**

**La Habana,**

**enero de 2025**

**Resumen**

En el contexto de la Industria 4.0, la detección temprana de anomalías en sistemas industriales es crucial para garantizar la continuidad y eficiencia operativa. Este estudio se enfoca en el desarrollo de un módulo de detección de anomalías que supervisa los datos operativos y de red de un sistema industrial experimental. Utilizando algoritmos de minería de datos y aprendizaje automático, el sistema es capaz de identificar desviaciones significativas que podrían indicar fallos operacionales. El objetivo principal es desarrollar un módulo capaz de detectar y alertar sobre anomalías, mejorando la resiliencia y eficiencia de los procesos industriales.

**Palabras Claves:** detección de anomalías, sistemas industriales, minería de datos, aprendizaje automático, Industria 4.0

**Abstract**

In the context of Industry 4.0, early detection of anomalies in industrial systems is crucial to ensuring operational continuity and efficiency. This study focuses on developing an anomaly detection module that monitors operational and network data from an experimental industrial system. Using data mining and machine learning algorithms, the system can identify significant deviations that may indicate operational failures. The main goal is to develop a module capable of detecting and alerting anomalies, thereby improving the resilience and efficiency of industrial processes.

**Keywords:** anomaly detection, industrial systems, data mining, machine learning, Industry 4.0

# 

**Índice de contenido**

[Introducción 6](#_Toc23878)

[Capítulo 1: Fundamentos Teóricos 9](#_Toc22188)

[1.1 Detección de Anomalías en Sistemas Industriales 10](#_Toc10174)

[1.1.1 Tipos de Anomalías en Sistemas Industriales 10](#_Toc17965)

[1.1.2 Principios de la Detección de Anomalías 10](#_Toc6559)

[1.1.3 Algoritmos para la Detección de Anomalías 11](#_Toc10231)

[1.1.4 Sistemas Industriales y su Monitoreo 12](#_Toc25621)

[1.1.5 Minería de Datos y Aprendizaje Automático en la Detección de Anomalías 12](#_Toc14013)

[1.1.6 Desafíos en la Detección de Anomalías 13](#_Toc13419)

[1.1.7 Impacto de la Detección de Anomalías en la Industria 4.0 13](#_Toc31324)

[1.2 Fundamentación de la Tecnología a Usar 13](#_Toc19404)

[1.3 Conclusiones Parciales 16](#_Toc3368)

[Capítulo 2: Solución a la Problemática 18](#_Toc2494)

[2.1 Entorno de ICS simulado 19](#_Toc27884)

[2.1.1 Arquitectura del Sistema 19](#_Toc10172)

[2.1.2 Componentes Principales 20](#_Toc15536)

[2.1.3 Recopilación de Datos 21](#_Toc5723)

[2.1.4 Implementación Técnica 21](#_Toc17841)

[2.1.5 Escenarios de Pruebas y Seguridad 22](#_Toc12510)

[2.1.6 Base de Datos 22](#_Toc14651)

[2.1.7 Conclusión 23](#_Toc25127)

[2.2 Solución Propuesta 23](#_Toc586)

[2.2.1 Preprocesamiento de Datos 24](#_Toc28256)

[2.2.2 Selección de la Estrategia 25](#_Toc11535)

[2.2.3 Generación de Resultados 41](#_Toc28381)

[2.2.4 Interpretación de Resultados mediante Reglas 43](#_Toc23509)

[2.2.5 Conclusión 44](#_Toc14840)

[2.3 Ingeniería de Requisitos 44](#_Toc15322)

[2.3.1 Breve descripción de la propuesta 45](#_Toc32514)

[2.3.2 Diagrama de actividades 45](#_Toc16914)

[2.3.2 Modelo de Dominio 47](#_Toc24185)

[2.3.3 Glosario de Términos 48](#_Toc30453)

[2.4 Requisitos 49](#_Toc18716)

[2.4.1 Requisitos Funcionales 49](#_Toc4548)

[2.4.2 Requisitos no funcionales. Problemas frecuentes 50](#_Toc1188)

[2.5 Arquitectura Candidata 51](#_Toc26345)

[2.5.1 Estudio de las Tecnologías Candidatas 51](#_Toc27423)

[2.5.2 Lenguajes de Programación Candidatos 52](#_Toc2651)

[2.5.3 Tratamiento de Problemas Frecuentes 53](#_Toc25152)

[2.6 Estilos y Patrones de Arquitectura Identificados 54](#_Toc20019)

[2.6.1 Llamada y Retorno 55](#_Toc12840)

[2.6.2 Centrado en Datos 57](#_Toc18129)

[2.6.3 Flujo de Datos 58](#_Toc24429)

[2.7 Patrones de Diseño 59](#_Toc2372)

[2.8 Conclusiones Parciales 66](#_Toc2669)

[Capitulo 3 Validación de la propuesta 68](#_Toc15826)

[3.1 Descripción de los Experimentos 69](#_Toc9781)

[3.2 Resultados Obtenidos 71](#_Toc12236)

[3.3 Análisis de resultados obtenidos 75](#_Toc11124)

[3.3.1 Comparación de métricas: Isolation Forest (Knime) vs ZScore (Python) 76](#_Toc3359)

[3.3.2 Comparación de métricas: Isolation Forest (Knime) vs DBSCAN (Python) 77](#_Toc20580)

[3.3.3 Comparación de métricas: Isolation Forest (Knime) vs Autoencoders (Python) 78](#_Toc23352)

[3.3.4 Comparación de métricas: Isolation Forest (Knime) vs Isolation Forest (Python) 79](#_Toc21636)

[3.4 Conclusiones Parciales 80](#_Toc5172)

[Conclusiones Generales: 81](#_Toc10137)

[Recomendaciones: 82](#_Toc13945)

[Referencias Bibliográficas 82](#_Toc11610)

**Índice de figura**

*[Figura](#_Toc3778)* [1](#_Toc3778)  *[Arquitectura del entorno ICS simulado](#_Toc3778)* [19](#_Toc3778)

*[Figura](#_Toc19333)* [2](#_Toc19333)  *[Dataset con valores nulos y columnas irrelevantes](#_Toc19333)* [24](#_Toc19333)

*[Figura](#_Toc10302)* [3](#_Toc10302)  *[Dataset sin valores nulos y con todas las columnas relevantes escaladas](#_Toc10302)* [24](#_Toc10302)

*[Figura](#_Toc25054)* [4](#_Toc25054)  *[Diagrama de clases del patrón Strategy](#_Toc25054)* [25](#_Toc25054)

*[Figura](#_Toc31538)* [5](#_Toc31538)  *[Definición matemática del Zscore](#_Toc31538)* [33](#_Toc31538)

*[Figura](#_Toc32189)* [6](#_Toc32189)  *[Formula matemática para la distribución normal](#_Toc32189)* [35](#_Toc32189)

*[Figura](#_Toc21178)* [7](#_Toc21178)  *[Ejemplo de resultados representados con un Gráfico de Pastel](#_Toc21178)* [42](#_Toc21178)

*[Figura](#_Toc31943)* [8](#_Toc31943)  *[Ejemplo de resultados representados con un](#_Toc31943)* [Gráfico](#_Toc31943) *[de Dispersión](#_Toc31943)* [43](#_Toc31943)

*[Figura](#_Toc69)* [9](#_Toc69)  *[Diagrama de Actividades](#_Toc69)* [46](#_Toc69)

*[Figura](#_Toc30588)* [10](#_Toc30588)  *[Modelo de dominio](#_Toc30588)* [47](#_Toc30588)

*[Figura](#_Toc30967)* [11](#_Toc30967)  *[Patrón NCapa](#_Toc30967)* [55](#_Toc30967)

*[Figura](#_Toc27068)* [12](#_Toc27068)  *[Patrón Repository](#_Toc27068)* [58](#_Toc27068)

*[Figura](#_Toc25163)* [13](#_Toc25163)  *[Patrón Pipes and Filters](#_Toc25163)* [59](#_Toc25163)

*[Figura](#_Toc17545)* [14](#_Toc17545)  *[Clase StrategySelector](#_Toc17545)* [60](#_Toc17545)

*[Figura](#_Toc20989)* [15](#_Toc20989)  *[Ejemplo de superclase y subclase donde se aplica el Factory Method](#_Toc20989)* [61](#_Toc20989)

*[Figura](#_Toc19726)* [16](#_Toc19726)  *[Ejemplo de una clase que funciona como base para el Template Method](#_Toc19726)* [62](#_Toc19726)

*[Figura](#_Toc17703)* [17](#_Toc17703)  *[Clase DocumentFactory con su implementación](#_Toc17703)* [63](#_Toc17703)

*[Figura](#_Toc27872)* [18](#_Toc27872)  *[Ejemplo de objetos que la utilizan sin saber su complejidad interna](#_Toc27872)* [64](#_Toc27872)

*[Figura](#_Toc4778)* [19](#_Toc4778)  *[Ejemplo de método parametrizado](#_Toc4778)* [65](#_Toc4778)

*[Figura](#_Toc22405)* [20](#_Toc22405)  *[Clase usada para representar los resultados en un gráfico de pastel](#_Toc22405)* [66](#_Toc22405)

# Introducción

La Industria 4.0 ha transformado el panorama de los sistemas industriales con la integración de tecnologías conectadas y sistemas ciberfísicos. Estos cambios han traído consigo la necesidad de implementar mecanismos de monitoreo avanzados para asegurar el correcto funcionamiento de los procesos industriales. En este contexto, garantizar la fiabilidad de los datos que generan y procesan estos sistemas es esencial para optimizar las operaciones y evitar interrupciones inesperadas. Para ello, es fundamental contar con sistemas capaces de identificar y responder a comportamientos anómalos que puedan comprometer la estabilidad del entorno industrial [1].

Un enfoque ampliamente utilizado para la protección de la continuidad operativa en sistemas industriales es la implementación de módulos de detección de anomalías. Estos módulos supervisan en tiempo real los datos generados por sensores y controladores, buscando desviaciones significativas que puedan ser indicadores de fallos operativos. Al detectar estas anomalías de manera temprana, es posible minimizar el impacto en los sistemas, garantizando su disponibilidad, integridad y eficiencia operativa [13].

Uno de los mayores desafíos en la detección de anomalías en sistemas industriales es la capacidad de procesar grandes volúmenes de datos provenientes de diversas fuentes en tiempo real. A esto se suma la necesidad de que los algoritmos de detección sean lo suficientemente precisos como para diferenciar entre fluctuaciones normales del sistema y comportamientos que podrían derivar en fallos o problemas operativos, evitando tanto falsos positivos como falsos negativos que puedan generar costos innecesarios o fallos no detectados [4].

El problema de investigación en este campo surge ante la creciente complejidad de los sistemas industriales y la necesidad de contar con mecanismos automatizados que puedan adaptarse a las condiciones cambiantes del entorno operativo [3, 5]. ¿Cómo diseñar un módulo de detección de anomalías que se ajuste a las características de los datos y sea capaz de identificar anomalías en un entorno industrial simulado?

El objeto de estudio de esta investigación es un sistema industrial experimental que incluye sensores, microcontroladores STM32 y un PLC flexible basado en la plataforma OpenPLC. Este sistema simula un entorno industrial con capacidad para capturar y procesar datos operativos, ofreciendo un marco adecuado para implementar y probar el módulo de detección de anomalías [1].

Si bien existen tecnologías para el monitoreo de procesos industriales, la complejidad de los entornos y la cantidad de datos que deben analizarse en tiempo real siguen siendo un reto. Los fallos operativos pueden comprometer seriamente la producción, generando pérdidas económicas y disminuyendo la eficiencia de los sistemas. Por lo tanto, contar con un sistema que permita la detección temprana de fallos es crucial para evitar estos contratiempos [1, 4].

El uso de módulos de detección de anomalías en este contexto es fundamental, ya que permiten identificar patrones inusuales de comportamiento en los sistemas antes de que el daño sea irreversible. El campo de acción de esta investigación se centra en la implementación de algoritmos de detección de anomalías que puedan supervisar tanto las señales operativas de los sensores como el comportamiento general del sistema industrial experimental [4, 5].

El objetivo general de esta investigación es desarrollar un módulo de detección de anomalías que permita identificar comportamientos anómalosl en un sistema industrial simulado, utilizando técnicas de minería de datos y aprendizaje automático aplicadas a los datos operativos recolectados por el sistema [2].

Para alcanzar este objetivo, se realizará la implementación de algoritmos de detección que se integrarán con los controladores STM32 y el PLC del sistema industrial. Además, se llevarán a cabo pruebas exhaustivas para validar la efectividad del módulo en la detección de anomalías, garantizando su capacidad de respuesta ante posibles fallos operacionales [2, 3].

El alcance de esta investigación incluye la detección de anomalías en señales operativas para que los operadores puedan tomar decisiones informadas sobre el mantenimiento y ajuste del sistema. Este enfoque proporcionará un marco claro para el desarrollo de la investigación y asegurará que se aborden los aspectos más críticos de la continuidad operativa en sistemas industriales [3, 4].

# Capítulo 1: Fundamentos Teóricos

Este capítulo establece los fundamentos teóricos necesarios para comprender y alcanzar los objetivos propuestos en este trabajo. Se discuten conceptos clave relacionados con la detección de anomalías, los sistemas industriales, los algoritmos de minería de datos y aprendizaje automático, y las tecnologías asociadas.

## 1.1 Detección de Anomalías en Sistemas Industriales

La detección de anomalías es una técnica fundamental utilizada para identificar eventos inusuales o patrones de comportamiento que se desvían de la operación normal de un sistema. En un sistema industrial, la identificación temprana de anomalías es crucial para prevenir fallos operacionales o deterioros en los equipos, lo que podría causar tiempos de inactividad no planificados o interrupciones en la producción. La detección de anomalías permite que los operadores puedan intervenir de manera preventiva y corregir posibles problemas antes de que estos escalen [6].

### 1.1.1 Tipos de Anomalías en Sistemas Industriales

En los sistemas industriales, las anomalías se pueden clasificar en tres tipos principales:

**Anomalías puntuales**: son eventos aislados que se desvían del comportamiento esperado. Por ejemplo, un sensor de temperatura que registra un valor anormal durante un breve período.

**Anomalías contextuales**: ocurren cuando un evento es anormal en un contexto específico. Por ejemplo, una temperatura elevada podría ser normal en un proceso, pero anómala en otro contexto operativo.

**Anomalías colectivas**: se refieren a un grupo de puntos de datos que, colectivamente, representan un comportamiento anómalo. Esto podría ser, por ejemplo, una tendencia inusual en los niveles de vibración de una máquina durante un período de tiempo.

### 1.1.2 Principios de la Detección de Anomalías

La detección de anomalías en sistemas industriales se basa en los siguientes principios fundamentales:

**Monitorización en tiempo real**: los datos operativos de sensores y controladores se analizan constantemente para detectar comportamientos anómalos.

**Modelado del comportamiento normal**: el sistema aprende a reconocer lo que se considera una operación normal del sistema industrial a través de la recopilación y análisis de datos históricos.

**Respuesta a tiempo**: una vez detectada una anomalía, el sistema genera alertas en tiempo real que permiten a los operadores tomar medidas correctivas antes de que ocurran daños graves.

### 1.1.3 Algoritmos para la Detección de Anomalías

Existen diferentes tipos de algoritmos utilizados para la detección de anomalías en sistemas industriales, entre ellos:

**Métodos basados en reglas**: se configuran reglas específicas que definen los umbrales operativos normales de un sistema. Cuando un dato excede estos umbrales, se detecta una anomalía.

**Métodos estadísticos**: se basan en modelos probabilísticos que evalúan la probabilidad de que un dato sea anómalo con respecto al comportamiento histórico del sistema.

**Algoritmos de minería de datos y aprendizaje automático**: estos métodos permiten el análisis de grandes volúmenes de datos operativos. Los modelos de aprendizaje automático pueden entrenarse para identificar patrones complejos que indican anomalías. Algoritmos como el kmeans clustering, SVM (Support Vector Machines) y redes neuronales se utilizan comúnmente para este fin [7] .

### 1.1.4 Sistemas Industriales y su Monitoreo

Los sistemas industriales son entornos altamente interconectados donde diferentes dispositivos, como sensores, controladores lógicos programables (PLC), y sistemas SCADA (Supervisory Control and Data Acquisition), supervisan y controlan procesos físicos. La cantidad de datos que estos sistemas generan es considerable, y la detección de anomalías en estos sistemas requiere el procesamiento continuo de datos en tiempo real.

**Sensores y controladores**: monitorean variables clave como temperatura, presión, velocidad de motores, entre otros. Estos datos son fundamentales para la toma de decisiones en tiempo real.

**PLC y SCADA**: permiten la automatización de procesos y la recopilación de datos operativos. Estos datos proporcionan la base para aplicar algoritmos de detección de anomalías y generar alertas en caso de que ocurra un comportamiento fuera de lo esperado.

### 1.1.5 Minería de Datos y Aprendizaje Automático en la Detección de Anomalías

El uso de técnicas de minería de datos y aprendizaje automático ha revolucionado la manera en que se detectan las anomalías en sistemas industriales. Estas técnicas permiten analizar grandes cantidades de datos y detectar patrones sutiles que podrían no ser evidentes mediante métodos convencionales.

**Minería de datos**: implica la extracción de información útil a partir de grandes volúmenes de datos. En el contexto de los sistemas industriales, se aplica para identificar tendencias y patrones anómalos.

**Aprendizaje automático**: a través de técnicas supervisadas y no supervisadas, los algoritmos de aprendizaje automático pueden aprender a diferenciar entre el comportamiento normal y anómalo. Modelos como las redes neuronales profundas y los modelos de regresión son ampliamente utilizados en estos contextos [8, 9].

### 1.1.6 Desafíos en la Detección de Anomalías

Algunos de los principales desafíos en la detección de anomalías en sistemas industriales incluyen:

**Falsos positivos y negativos**: uno de los mayores retos es garantizar que el sistema sea lo suficientemente preciso para minimizar las falsas alarmas (falsos positivos) y, al mismo tiempo, no pase por alto comportamientos anómalos reales (falsos negativos).

**Procesamiento en tiempo real**: debido a la gran cantidad de datos generados, los sistemas deben ser capaces de procesar la información en tiempo real sin comprometer el rendimiento de la operación industrial.

### 1.1.7 Impacto de la Detección de Anomalías en la Industria 4.0

La implementación de módulos de detección de anomalías contribuye significativamente a la Industria 4.0, proporcionando un nivel adicional de seguridad y optimización operativa. Estos módulos permiten que los sistemas industriales se vuelvan más resilientes, mejorando tanto la eficiencia como la seguridad operativa, y ayudando a prevenir interrupciones en la producción.

## 1.2 Fundamentación de la Tecnología a Usar

En el desarrollo de soluciones para la detección de anomalías en sistemas industriales, la elección de las tecnologías adecuadas es un paso crucial que puede influir significativamente en la eficacia y eficiencia de la solución final. En este contexto, hemos seleccionado varias tecnologías clave para emplear en el desarrollo de nuestra solución: el lenguaje de programación Python y sus bibliotecas asociadas, SQLite3 para la base de datos, PyCharm como entorno de desarrollo integrado (IDE), y herramientas específicas para el manejo de algoritmos y pruebas como scikitlearn, SciPy, y TensorFlow/Keras.

A continuación, presentamos una justificación detallada para la selección de cada una de estas tecnologías, destacando cómo su uso contribuye a la eficacia de nuestra solución y cómo se alinean con los objetivos de nuestra investigación. Nuestra meta es proporcionar una base sólida para el desarrollo de una solución eficiente y eficaz que permita la identificación de comportamientos anómalos en sistemas industriales.

**1. Python**

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general que ha ganado popularidad en el ámbito de la ciencia de datos y la inteligencia artificial. Su sintaxis sencilla y clara permite un desarrollo rápido y eficiente. Python cuenta con una extensa comunidad de desarrolladores y una rica colección de bibliotecas que facilitan la implementación de algoritmos de detección de anomalías y procesamiento de datos. La flexibilidad de Python lo convierte en una excelente elección para prototipos y desarrollo ágil, permitiendo una rápida iteración y prueba de ideas.

**2. Pandas y NumPy**

**Pandas**: Es una biblioteca fundamental para la manipulación y análisis de datos en Python. Proporciona estructuras de datos flexibles como DataFrames, que facilitan la carga, manipulación y análisis de grandes volúmenes de datos operativos generados por sistemas industriales. Con Pandas, es posible realizar operaciones complejas de filtrado y transformación de datos de manera eficiente, lo que es crucial para preparar los datos antes de aplicar algoritmos de detección de anomalías.

**NumPy**: Esta biblioteca proporciona soporte para arreglos multidimensionales y operaciones matemáticas avanzadas, lo que es especialmente útil para cálculos numéricos y análisis estadístico. Al trabajar con datos de sensores en tiempo real, NumPy permite realizar cálculos eficientes que son esenciales para implementar algoritmos de aprendizaje automático en la detección de anomalías.

**3. Scikitlearn y SciPy**

Scikitlearn (sklearn): Es una biblioteca clave para implementar algoritmos de aprendizaje automático. Proporciona herramientas robustas para técnicas como Isolation Forest, Support Vector Machines (SVM) y detección de outliers, que son esenciales en el módulo de detección de anomalías. Su facilidad de uso y documentación extensa la convierten en una opción ideal para experimentación y desarrollo.

**SciPy**: Es utilizada para cálculos científicos y estadísticos avanzados, incluyendo la optimización, integración y análisis de datos. Es particularmente útil en las pruebas estadísticas necesarias para validar la detección de anomalías.

**4. TensorFlow y Keras**

Estas bibliotecas se emplean para la implementación de modelos de redes neuronales avanzados, como autoencoders, utilizados para detectar patrones anómalos en datos complejos o de alta dimensionalidad.

**Keras**: Ofrece una API intuitiva y de alto nivel para construir y entrenar modelos.

**TensorFlow**: Proporciona la potencia computacional subyacente para entrenar modelos en CPU o GPU, lo cual es esencial para manejar grandes volúmenes de datos operativos en sistemas industriales.

**5. PyCharm**

PyCharm es un entorno de desarrollo integrado (IDE) muy popular para Python, conocido por su robustez y su amplia gama de características que facilitan el desarrollo de software. Ofrece herramientas como autocompletado de código, depuración avanzada y control de versiones, lo que mejora la productividad del desarrollador y reduce la posibilidad de errores. Además, PyCharm proporciona integración con sistemas de control de versiones y soporte para bibliotecas de ciencia de datos, lo que lo convierte en una opción ideal para el desarrollo de soluciones complejas en detección de anomalías.

**6. Otras Herramientas: time**

La biblioteca estándar `time` se utilizó para realizar mediciones precisas de rendimiento en las pruebas de detección de anomalías. Esto permitió evaluar la eficiencia de los algoritmos en términos de tiempo de ejecución, un factor crítico en sistemas industriales donde la detección en tiempo real es fundamental.

Con esta selección tecnológica, la solución está equipada para manejar datos operativos de alta complejidad, implementar técnicas avanzadas de detección y proporcionar resultados en tiempo real.

## 1.3 Conclusiones Parciales

* Base Teórica para el Módulo de Detección de Anomalías:

A lo largo de este capítulo, se ha establecido una base teórica sólida para el desarrollo de un módulo de detección de anomalías en sistemas industriales. Este enfoque permite identificar de manera temprana comportamientos inusuales, contribuyendo a prevenir fallos operativos, minimizar tiempos de inactividad y mitigar riesgos cibernéticos. Este t rabajo subraya la importancia de este tipo de módulos como mecanismos cruciales para garantizar la seguridad y continuidad operativa en entornos conectados.

* Marco Conceptual y Métodos de Análisis:

Se definió un marco conceptual que incluye los tipos de anomalías y los principios fundamentales para su detección en sistemas industriales. Además, se identificaron métodos efectivos de análisis, destacando técnicas basadas en aprendizaje automático, redes neuronales y minería de datos. Estas herramientas son clave para reducir falsos positivos y negativos en la detección de comportamientos anómalos, asegurando precisión en los resultados.

* Selección de Tecnologías y Herramientas:

La elección de Python como lenguaje de programación, junto con bibliotecas como Pandas, NumPy, Scikitlearn y SciPy, ha proporcionado una base robusta para el manejo, análisis y procesamiento de datos operativos. Estas herramientas facilitan no solo la manipulación eficiente de grandes volúmenes de datos, sino también la implementación de algoritmos avanzados de detección de anomalías.

* Diseño de Modelos y Ecosistema Tecnológico:

La integración de TensorFlow y Keras permitió diseñar modelos de aprendizaje profundo para analizar datos complejos y detectar patrones inusuales en tiempo real. Junto con PyCharm como entorno de desarrollo, se creó un ecosistema tecnológico ágil y efectivo. Adicionalmente, el uso de la biblioteca time para medir el rendimiento de los algoritmos aseguró que las soluciones desarrolladas sean prácticas y escalables en entornos industriales.

**Conclusión General del Capítulo 1**

Este capítulo ha definido un marco teórico y práctico robusto para el desarrollo del módulo de detección de anomalías, alineándose con los objetivos de la investigación. La combinación de tecnologías seleccionadas asegura que la solución propuesta sea eficiente, innovadora y capaz de mejorar la seguridad y la eficiencia operativa en sistemas industriales.

# Capítulo 2: Solución a la Problemática

En este capítulo se presenta la solución propuesta para abordar la problemática de garantizar la seguridad y fiabilidad en sistemas industriales frente a posibles fallos operativos y amenazas cibernéticas. La solución se centra en el desarrollo e implementación de un módulo de detección de anomalías diseñado para identificar patrones inusuales en los datos operativos y de red del sistema.

El módulo de detección de anomalías forma parte de un sistema experimental que simula un entorno de control industrial (ICS) accesible y modular. Este banco de pruebas permite analizar el comportamiento del sistema bajo condiciones normales y ante escenarios de ataque, facilitando la identificación temprana de anomalías que podrían comprometer su estabilidad y seguridad.

A lo largo del capítulo, se describen los componentes principales de la solución, la arquitectura del entorno de ICS simulado y las funcionalidades específicas del módulo de detección de anomalías. Además, se detalla cómo este módulo contribuye a mitigar riesgos operativos y cibernéticos mediante la integración de técnicas avanzadas de análisis de datos.

## 2.1 Entorno de ICS simulado

En esta sección se describe la arquitectura, diseño e implementación del sistema experimental desarrollado para simular y analizar escenarios en un entorno de control industrial. Este banco de pruebas, diseñado para ser accesible y de bajo costo, permite estudiar y validar estrategias de seguridad en sistemas industriales, replicando de forma precisa el funcionamiento de un Sistema de Control Industrial (ICS).

El sistema integra componentes hardware y software, incluyendo sensores, controladores, interfaces hombremáquina (HMI) y sistemas SCADA, para recrear un entorno funcional que puede ser utilizado en simulaciones, investigaciones académicas y formación técnica en ciberseguridad industrial.

2.1.1 Arquitectura del Sistema

La arquitectura del sistema está diseñada para reflejar un ICS real, manteniendo una estructura modular que facilita el análisis y la experimentación. La Figura 1 Arquitectura del entorno ICS simulado ilustra la arquitectura general del banco de pruebas, destacando las interacciones entre sus componentes principales.

Изображение выглядит как диаграмма, снимок экрана, линия, круг

Автоматически созданное описание

*Figura 1* *Arquitectura del entorno ICS simulado*

### 2.1.2 Componentes Principales

**1. Sensores**

Sensores de posición de carga: Basados en fotoresistores, ubicados al inicio y al final de la cinta transportadora.

Sensores de posición del manipulador: Utilizan interruptores de límite para registrar las posiciones mínima y máxima.

**2. Microcontroladores (STM32)**

Actúan como dispositivos de entrada/salida remota que recopilan los datos de los sensores y los envían al Controlador Lógico Programable (PLC).

**3. Controlador Lógico Programable (PLC)**

Se utiliza un SoftPLC implementado en una plataforma Raspberry Pi, ejecutando OpenPLC para la lógica de control. El PLC gestiona las entradas y salidas digitales y analógicas, permitiendo el control en tiempo real del sistema.

**4. Interfaz HombreMáquina (HMI)**

Implementada en Raspberry Pi, esta interfaz permite visualizar datos en tiempo real y optimizar procesos mediante un panel interactivo que muestra el estado de los sensores, parámetros del sistema y tendencias históricas.

**5. Sistema de Supervisión y Adquisición de Datos (SCADA)**

Utilizado para monitorear el sistema de manera centralizada, gestionar alarmas y responder a eventos en tiempo real. El sistema SCADA está basado en software alojado en Raspberry Pi.

**6. Servidor y Análisis de Tráfico**

Un Orange Pi actúa como servidor central, recopilando datos de red y del sistema a través de Docker. Los datos son gestionados con tecnologías como Node.js, NestJS, PostgreSQL y Vue3, asegurando escalabilidad y flexibilidad.

### 2.1.3 Recopilación de Datos

El sistema recopila datos operativos y de red para analizar el comportamiento del entorno. Los datos se clasifican en:

**Datos del sistema:** Servicios activos, interfaces de red, recursos disponibles (CPU, memoria, almacenamiento).

**Datos de red**: Tablas ARP, tráfico por protocolo (ARP, ICMP, TCP, UDP, HTTP, ModBus).

**Datos de aplicaciones**: Información sobre procesos en ejecución, puertos abiertos y servicios asociados.

La recopilación se realiza mediante scripts en Python, utilizando bibliotecas como scapy y psutil, y los datos son almacenados en una base de datos relacional PostgreSQL.

### 2.1.4 Implementación Técnica

**1 Plataforma Hardware**

La elección de componentes accesibles y económicos, como Raspberry Pi y Orange Pi, permite replicar funcionalidades avanzadas de un ICS real manteniendo costos bajos. Cada componente ha sido configurado para maximizar su interoperabilidad y rendimiento.

**2 Plataforma Software**

El software instalado incluye:

**SoftPLC**: OpenPLC para la lógica de control.

**SCADA y HMI**: Herramientas gráficas para la supervisión en tiempo real.

**Monitoreo de red y sistema**: Scripts personalizados en Python para la captura y procesamiento de datos.

**Servidor**: Infraestructura basada en contenedores Docker para garantizar modularidad y escalabilidad.

### 2.1.5 Escenarios de Pruebas y Seguridad

El sistema soporta múltiples escenarios de prueba diseñados para evaluar la resiliencia de los ICS frente a ataques comunes. Algunos de los escenarios incluyen:

**1. Manipulación de sensores**: Interferencia en los datos recolectados por los sensores para evaluar la respuesta del sistema.

**2. Ataques de red**: Simulación de ataques como interceptación de paquetes o manipulación de ARP para evaluar la robustez de la red.

**3. Intrusión en el PLC**: Pruebas de seguridad en la lógica del SoftPLC, buscando vulnerabilidades en su programación o hardware subyacente.

### 2.1.6 Base de Datos

La arquitectura de la base de datos está optimizada para gestionar grandes volúmenes de datos en tiempo real, incluyendo información del sistema, red y tráfico. La Figura 2 presenta el diseño lógico de la base de datos, con las siguientes tablas principales:

**Aplicaciones instaladas**: Información sobre nombre, versión y descripción.

**ARP y tráfico de red**: Direcciones IP y físicas, protocolos utilizados y estadísticas de paquetes.

**Servicios del sistema**: Estado y configuración de los servicios ejecutándose en los dispositivos.

### 2.1.7 Conclusión

El diseño modular y accesible de este banco de pruebas proporciona un entorno ideal para evaluar y perfeccionar módulos de detección de anomalías en sistemas industriales. Su capacidad para simular con precisión procesos y escenarios industriales reales permite a los usuarios realizar pruebas exhaustivas en un contexto controlado, replicando tanto condiciones normales como situaciones anómalas.

La implementación de tecnologías abiertas y la integración de hardware de bajo costo aseguran que el sistema sea funcional y asequible, lo que facilita su adopción en entornos académicos e industriales. Este enfoque no solo fomenta la formación en ciberseguridad, sino que también garantiza que las pruebas de los módulos de detección de anomalías sean eficientes y representativas de los desafíos del mundo real, contribuyendo así al desarrollo de soluciones más robustas y confiables.

## 2.2 Solución Propuesta

En este capítulo, se presenta la solución propuesta para la detección de anomalías en conjuntos de datos industriales. La solución abarca desde el preprocesamiento de los datos hasta la interpretación de resultados utilizando reglas generadas a partir de análisis. Cada etapa está diseñada para garantizar la eficiencia y efectividad del módulo, maximizando la precisión en la detección de anomalías mientras se minimizan los falsos positivos y negativos.

### 2.2.1 Preprocesamiento de Datos

El preprocesamiento es la etapa inicial y fundamental en la solución propuesta. Asegura que los datos estén en un formato adecuado para el análisis y elimina posibles sesgos que puedan afectar la calidad de los resultados.

#### 2.2.1.1 Manejo de Valores Nulos

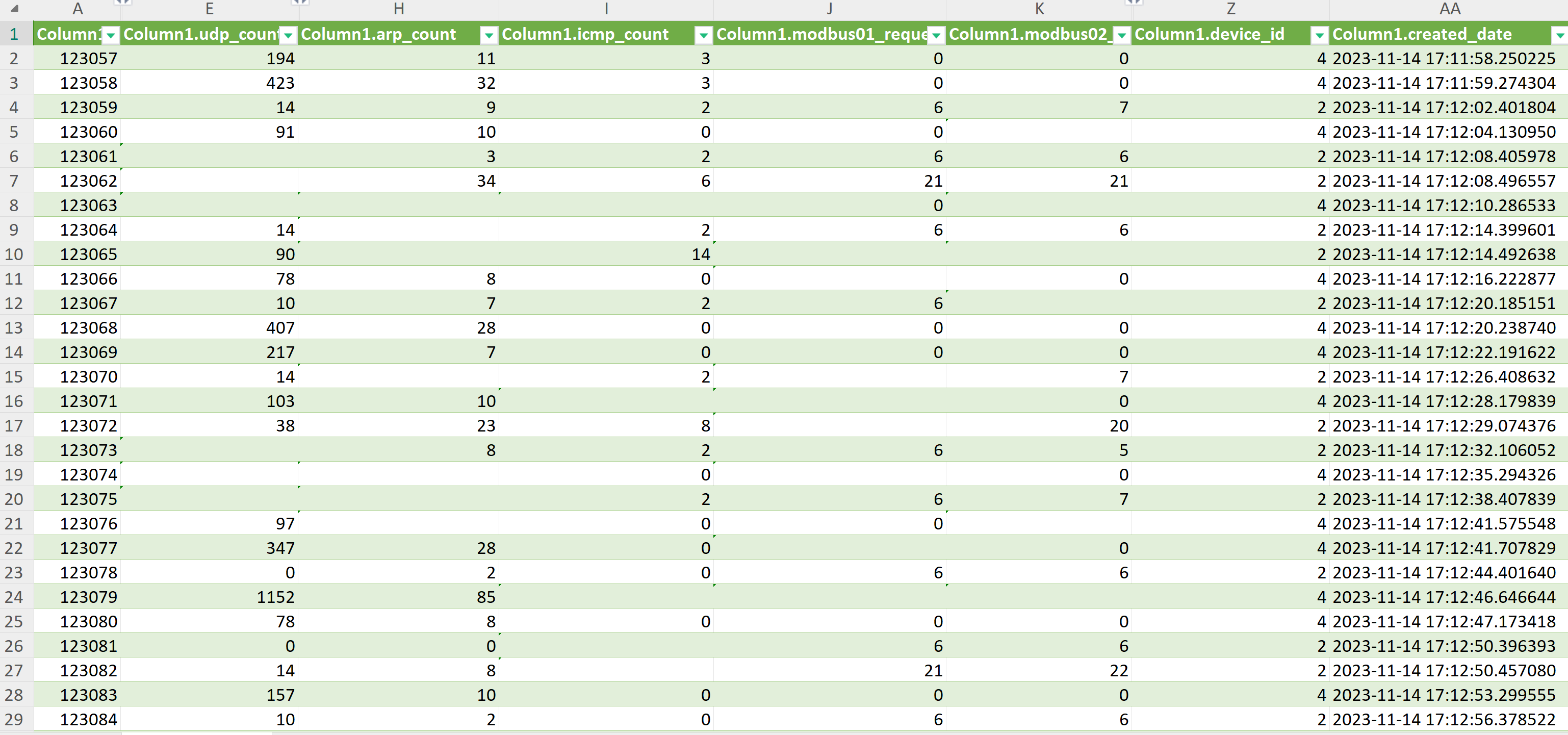
Los valores nulos son reemplazados con el promedio de la columna o valores específicos, según el contexto del dato. Esto asegura que los algoritmos puedan procesar el conjunto sin errores.

#### 2.2.1.2 Normalización y Escalado

Los datos se normalizan para que todas las características estén en un rango comparable, utilizando técnicas como la normalización minmax o el estándar zscore.

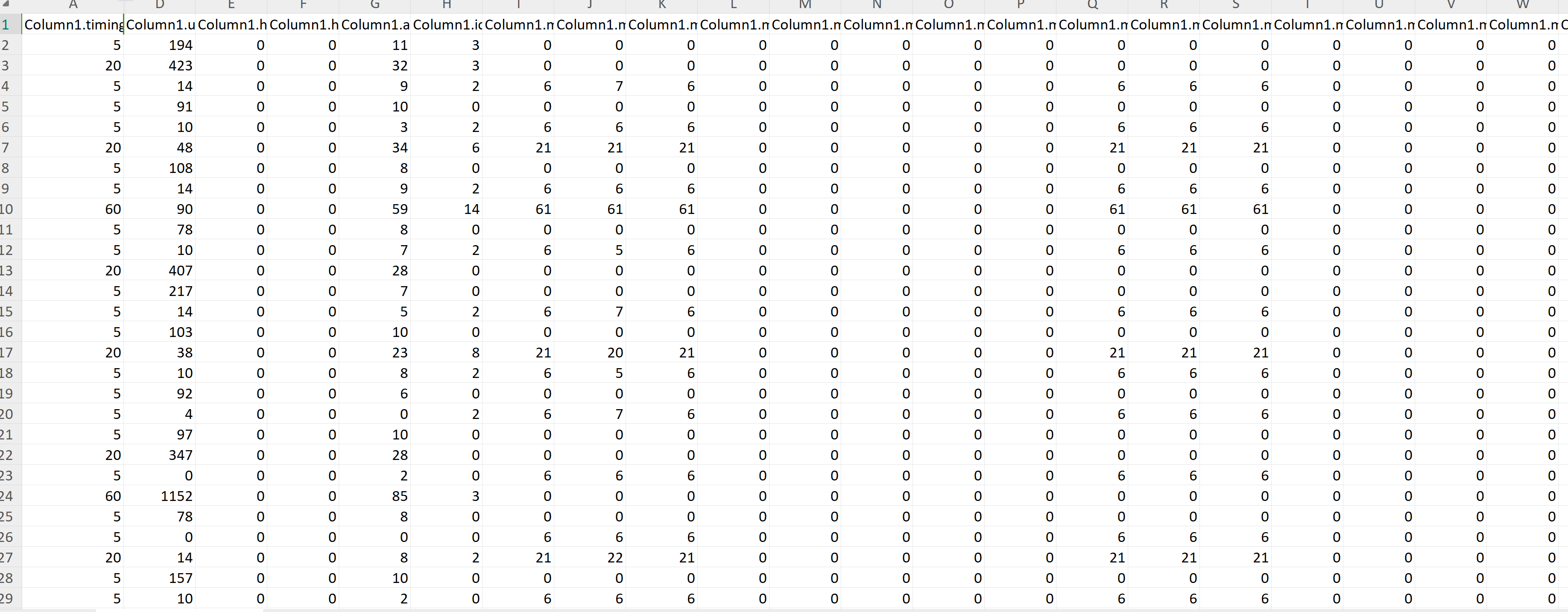
**Ejemplo de Preprocesamiento**:

**Entrada**: Conjunto de datos con valores nulos y columnas irrelevantes.



*Figura 2* *Dataset con valores nulos y columnas irrelevantes*

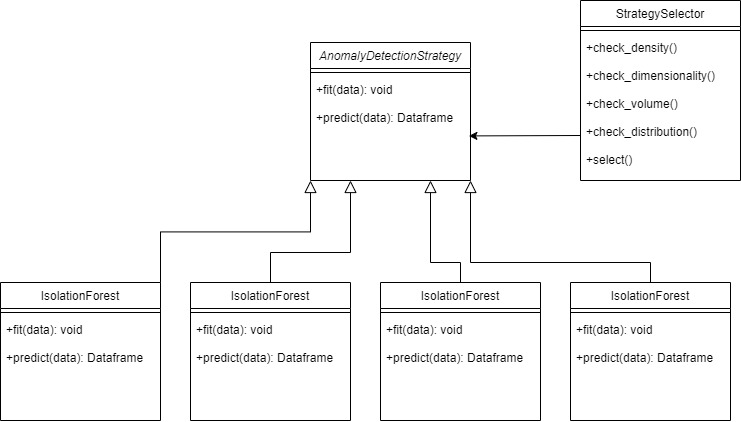
**Salida**: DataFrame limpio, sin valores nulos y con todas las columnas relevantes escaladas.



*Figura 3* *Dataset sin valores nulos y con todas las columnas relevantes escaladas*

### 2.2.2 Selección de la Estrategia

El módulo implementa el patrón Strategy para seleccionar dinámicamente el algoritmo de detección de anomalías más adecuado. Esto se basa en las características del conjunto de datos.



*Figura 4* *Diagrama de clases del patrón Strategy*

2.2.2.1 Algoritmos Implementados

**1. Isolation Forest**:

Isolation Forest es un algoritmo de aprendizaje automático no supervisado diseñado específicamente para la detección de anomalías. La idea principal detrás de Isolation Forest es que las anomalías son observaciones que están aisladas o separadas del resto de los datos, lo que las hace más fáciles de identificar. En términos sencillos, el algoritmo I ntenta "aislar" estos puntos mediante la creación de particiones o cortes en el conjunto de datos.

**Principios Fundamentales de Isolation Forest**

**1. Aislamiento por partición:**

El algoritmo utiliza árboles de decisión para dividir los datos en subgrupos. Los puntos que son anomalías (outliers) suelen necesitar menos cortes (particiones) para ser aislados. Esto se debe a que están más alejados del resto de los datos, lo que significa que se pueden separar rápidamente.

**2. Aleatoriedad:**

Al construir el bosque de árboles, Isolation Forest selecciona aleatoriamente una característica (una dimensión del conjunto de datos) y luego selecciona un valor dentro del rango de esa característica para dividir el conjunto de datos. Este proceso se repite varias veces hasta que los puntos quedan aislados en nodos hoja.

**3. Profundidad del árbol:**

El número de particiones necesarias para aislar un punto (la profundidad del árbol) nos da una medida de si es una anomalía o no. Las observaciones normales requerirán más particiones para ser aisladas, mientras que las observaciones anómalas serán aisladas más rápidamente, es decir, con menos particiones.

**4. Bosque de árboles:**

Isolation Forest utiliza un conjunto (o bosque) de muchos árboles de aislamiento para promediar el número de particiones necesarias para a islar cada observación. Al combinar la información de varios árboles, se mejora la robustez del modelo y se reduce la variabilidad en las decisiones.

**Funcionamiento del Algoritmo**

**1. Construcción del bosque:**

* Se generan varios árboles de aislamiento, donde cada árbol es construido tomando una muestra aleatoria de los datos de entrenamiento.
* En cada nodo del árbol, se selecciona aleatoriamente una característica (columna del dataset) y un valor de corte aleatorio dentro del rango de esa característica.
* Los puntos se dividen en dos ramas: los que están por debajo del valor de corte y los que están por encima.

**2. Cálculo del puntaje de anomalía:**

* El puntaje de anomalía de un punto se calcula basado en la longitud del camino desde la raíz hasta el nodo hoja en el que queda aislado.
* Los puntos que requieren menos particiones (es decir, tienen caminos más cortos) se consideran más anómalos. Esto se debe a que las anomalías suelen estar dispersas y, por lo tanto, se aíslan rápidamente.
* Los puntos normales, que están más agrupados, requerirán más divisiones (caminos más largos) para quedar aislados.

**3. Interpretación del puntaje:**

* El puntaje de anomalía varía entre 0 y 1.
* Un puntaje cercano a 1 indica una alta probabilidad de que el punto sea una anomalía.
* Un puntaje cercano a 0 indica que el punto es normal.

Los puntos con un valor mayor a un umbral (generalmente cerca de 0.5) son considerados como anomalías.

**Ventajas de Isolation Forest**

**1. Escalabilidad**:

Isolation Forest es altamente escalable y eficiente, ya que funciona bien incluso con grandes volúmenes de datos. Al usar una muestra aleatoria de datos y un número limitado de árboles, reduce el costo computacional.

**2. No hace suposiciones de distribución:**

A diferencia de métodos estadísticos como Zscore, Isolation Forest no asume que los datos sigan una distribución normal o específica. Es adecuado para detectar anomalías en conjuntos de datos de alta dimensionalidad.

**3. Capacidad para detectar anomalías multivariadas:**

Isolation Forest puede detectar anomalías en conjuntos de datos con múltiples características, mientras que algunos métodos como el IQR o Zscore son principalmente univariados.

**4. Robustez frente a datos ruidosos:**

Debido a su enfoque basado en aislamiento, puede ser menos sensible al ruido en los datos en comparación con otros métodos.

**Limitaciones de Isolation Forest**

**1. Outliers sutiles:**

Aunque Isolation Forest es efectivo en encontrar outliers evidentes, podría tener dificultades para detectar anomalías sutiles que no están tan alejadas de la distribución normal.

1. **Autoencoders**

Autoencoders son una clase de redes neuronales utilizadas principalmente para la reducción de dimensionalidad y la compresión de datos, pero también son muy efectivos en la detección de anomalías. Un autoencoder se entrena para intentar reconstruir sus entradas después de pasar por una representación de baja dimensión (el "cuello de botella"). Al hacerlo, aprende las características más importantes y significativas de los datos, y la diferencia entre la entrada y la reconstrucción se utiliza para identificar anomalías.

**Principios Fundamentales de los Autoencoders**

**1. Arquitectura de los Autoencoders**: Un autoencoder consta de dos partes principales:

**Encoder**: Comprime la entrada en una representación interna de menor dimensión.

**Decoder**: Reconstruye la entrada original a partir de la representación interna comprimida.

La estructura básica se puede visualizar como una red neuronal de tres capas:

* Capa de entrada: Recibe los datos originales.
* Capa oculta (o cuello de botella): Representación comprimida de los datos.
* Capa de salida: Reconstruye los datos de entrada.

**2. Entrenamiento**:

* El objetivo del entrenamiento de un autoencoder es minimizar la diferencia entre la entrada original y la salida reconstruida, es decir, el error de reconstrucción.
* Este error se calcula generalmente usando la función de pérdida como el error cuadrático medio (MSE) entre la entrada y la salida.

**3. Detección de Anomalías**:

Durante el entrenamiento, el autoencoder aprende a reconstruir las entradas "normales" (puntos que están dentro del rango de variabilidad de los datos).

Cuando se aplica a nuevos datos, el autoencoder tendrá dificultades para reconstruir correctamente las anomalías, ya que estas no siguen el mismo patrón que los datos normales.

El error de reconstrucción es entonces utilizado como un indicador de si un punto es anómalo. Un error de reconstrucción alto indica que el punto es una anomalía, mientras que un error bajo indica que el punto es normal.

**Funcionamiento de un Autoencoder para la Detección de Anomalías**

**1. Entrenamiento**:

Un autoencoder se entrena usando un conjunto de datos de solo ejemplos normales. Esto es útil en situaciones donde las anomalías son raras y no están presentes en el conjunto de entrenamiento.

Durante este proceso, el autoencoder aprende a mapear las entradas normales a una representación de menor dimensión y luego a reconstruirlas.

**2. Detección de Anomalías**:

* Una vez entrenado, el autoencoder puede ser utilizado para predecir las anomalías.
* Se alimenta al autoencoder con nuevos datos (que pueden contener anomalías). El autoencoder intenta reconstruir esos datos.
* El error de reconstrucción (diferencia entre la entrada y la salida reconstruida) es calculado.
* Si el error de reconstrucción es mayor que un cierto umbral, el punto se considera una anomalía.

**3. Umbral para Anomalías:**

El valor del umbral puede ser ajustado para optimizar la detección de anomalías. Generalmente, se utiliza una distribución de los errores de reconstrucción para determinar un umbral que divide a los puntos normales de los anómalos.

**Ventajas de los Autoencoders en Detección de Anomalías:**

1. Captura de patrones complejos: A diferencia de métodos más simples, como Isolation Forest o DBSCAN, los autoencoders pueden aprender representaciones complejas de los datos, lo que los hace adecuados para datos no lineales.

2. No requiere etiquetas de anomalías: Los autoencoders son adecuados para situaciones de aprendizaje no supervisado donde no hay etiquetas de anomalías disponibles.

3. Detección de anomalías en datos de alta dimensión: Al ser una red neuronal, un autoencoder puede manejar datos con múltiples dimensiones (características) de forma eficiente.

**Limitaciones de los Autoencoders:**

1. Requiere grandes cantidades de datos: Los autoencoders pueden requerir una cantidad significativa de datos para entrenarse de manera efectiva, especialmente cuando las anomalías son raras.

2. Selección del umbral: Determinar el umbral adecuado para la detección de anomalías puede ser complicado y suele requerir experimentación.

3. Computacionalmente intensivo: Los autoencoders, al ser redes neuronales, pueden ser más lentos de entrenar y más costosos computacionalmente en comparación con otros métodos de detección de anomalías.

**3. DBSCAN (DensityBased Spatial Clustering of Applications with Noise)**

DBSCAN es un algoritmo de agrupamiento basado en densidad, que puede ser utilizado también para la detección de anomalías. A diferencia de métodos como Kmeans, que requieren especificar el número de clusters, DBSCAN no requiere que se defina este parámetro de antemano. En lugar de eso, DBSCAN detecta los clusters basándose en la densidad de los puntos.

**Principios Fundamentales de DBSCAN**

**1. Puntos centrales y vecinos**: DBSCAN clasifica los puntos en tres categorías:

* Puntos centrales: Son puntos que tienen suficientes vecinos cercanos (definidos por un parámetro `min\_samples`).
* Puntos de borde: Son puntos que están en el borde de un cluster pero no tienen suficientes vecinos cercanos para ser considerados como puntos centrales.
* Ruido o anomalías: Son puntos que no tienen suficientes vecinos cercanos y no pertenecen a ningún cluster.

**2. Radio de vecindad**: El parámetro `eps` (epsilon) define el radio dentro del cual se buscarán vecinos cercanos.

**3. Algoritmo de agrupamiento**:

* DBSCAN empieza con un punto arbitrario no visitado.
* Se busca un conjunto de vecinos dentro del radio `eps`.
* Si el punto tiene suficientes vecinos (al menos `min\_samples`), se marca como punto central y se forma un cluster.
* Si un punto tiene menos vecinos de los necesarios, se clasifica como ruido o anomalía.

**Ventajas de DBSCAN en Detección de Anomalías:**

**1. No requiere predefinir el número de clusters**: A diferencia de algoritmos como Kmeans, DBSCAN no requiere especificar cuántos clusters esperamos encontrar.

**2. Capaz de encontrar anomalías de forma natural:** Los puntos que no pertenecen a ningún cluster (ruido) son automáticamente considerados anomalías.

**3. No asume una distribución particular de los datos:** Funciona bien con datos que no siguen distribuciones específicas, como las distribuciones normales.

**Limitaciones de DBSCAN:**

**1. Sensibilidad a los parámetros `eps` y `min\_samples`:** Elegir los parámetros adecuados puede ser desafiante, ya que pequeños cambios en los valores pueden tener un gran impacto en los resultados.

**4. ZScore:**

El Zscore es un enfoque estadístico simple y efectivo para detectar anomalías en datos univariados o multivariados. La idea central es medir cuán lejos está un punto de datos del promedio del conjunto en términos de desviaciones estándar. Es ideal para datos que siguen una distribución aproximadamente normal.

**Principios Fundamentales del ZScore**

1. Definición matemática del Zscore: Para un punto x, el Zscore se calcula como:

Pasted image 20241129184938

*Figura 5* *Definición matemática del Zscore*

Donde:

* x: El valor de un punto de datos.
* μ: La media del conjunto de datos.
* σ: La desviación estándar del conjunto de datos.

El Zscore mide el número de desviaciones estándar que x está por encima o por debajo de la media.

**2. Umbrales de anomalía:**

Valores con un Zscore mayor (en magnitud) a un umbral, como |Z| > 3, son considerados anomalías.

Este umbral se basa en las propiedades de la distribución normal:

* El 68% de los datos cae dentro de ±1σ.
* El 95% dentro de ±2σ.
* El 99.7% dentro de ±3σ.

**3. Suposición de normalidad:** El Zscore funciona mejor cuando los datos tienen una distribución normal. Si los datos no son normales, los resultados pueden ser menos fiables.

**Distribución Normal**

La distribución normal, también conocida como distribución de Gauss o campana de Gauss, es una de las distribuciones de probabilidad más I mportantes en estadística. Es ampliamente utilizada porque muchos f enómenos naturales y sociales tienden a seguir este patrón.

**Características Clave de una Distribución Normal**

**1. Forma de campana simétrica:**

* La gráfica de la distribución tiene forma de campana y es perfectamente simétrica respecto a su media.
* Esto significa que los datos están distribuidos de manera uniforme alrededor de la media.

**2. Media, mediana y moda iguales**:

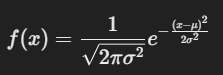
* En una distribución normal, estos tres valores coinciden y se encuentran en el centro de la curva.

**3. Desviación estándar y dispersión:**

La dispersión de los datos está determinada por la desviación estándar (σ). Aproximadamente:

* El 68% de los datos está dentro de ±1σ
* El 95% de los datos está dentro de ±2σ
* El 99.7% de los datos está dentro de ±3σ

**4. Función de densidad de probabilidad (PDF)**: La fórmula matemática para la distribución normal es:



*Figura 6* *Formula matemática para la distribución normal*

Donde:

* x: El valor de la variable aleatoria.
* μ: La media (centro de la distribución).
* σ: La desviación estándar (ancho de la distribución).
* e: La base del logaritmo natural.

**5. Colas infinitas**:

La curva se extiende hacia ambos lados indefinidamente, pero con probabilidades cada vez más pequeñas.

**Ejemplo de Fenómenos que Siguen una Distribución Normal**

**1. Alturas de personas:**

* Si mides la altura de un gran número de personas, la mayoría estará cerca de la media, con pocas personas siendo mucho más altas o mucho más bajas.

**2. Errores de medición:**

* Los errores en instrumentos de medición precisos suelen distribuirse de forma normal.

**3. Notas de exámenes estandarizados:**

* En muchos casos, las calificaciones de pruebas como SAT o IQ tienden a seguir una distribución normal.

**Funcionamiento del ZScore**

**1. Cálculo de métricas estadísticas:**

* Se calcula la media (μ\mu) y la desviación estándar (σ\sigma) para el conjunto de datos.
* Esto se hace generalmente de forma independiente para cada columna si se analizan varias características.

**2. Transformación a Zscores:**

* Cada punto del conjunto de datos se transforma utilizando la fórmula del Zscore.
* Esto normaliza los datos para que tengan una media de 0 y una desviación estándar de 1.

**3. Identificación de anomalías:**

* Los puntos con ∣Z∣| mayor que el umbral definido (por ejemplo, 3) se etiquetan como anomalías.
* En datos multivariados, se puede calcular un Zscore por cada columna o una métrica combinada.

**Ventajas del ZScore**

**1. Simplicidad:**

* Fácil de calcular y de interpretar.
* No requiere parámetros complejos ni entrenamiento.

**2. Rapidez:**

Es computacionalmente eficiente y puede aplicarse incluso a conjuntos de datos grandes.

**3. Escalabilidad:**

Funciona bien para datos univariados y puede extenderse a datos multivariados calculando Zscores para cada característica.

**4. Interpretabilidad:**

Los valores de Zscore son intuitivos: indican cuán lejos está un punto del promedio.

**Limitaciones del ZScore**

**1. Dependencia de la distribución normal:**

Asume que los datos tienen una distribución aproximadamente normal. Si los datos están sesgados o tienen distribuciones no normales, los Zscores pueden no ser útiles.

**2. Sensibilidad a valores extremos:**

Las anomalías influyen en el cálculo de μ\mu y σ\sigma, lo que puede distorsionar los resultados. En estos casos, el uso de estadísticas robustas (como la mediana y la desviación absoluta mediana, MAD) es preferible.

**3. Limitado a relaciones lineales:**

No considera correlaciones no lineales entre características en datos multivariados.

#### 2.2.2.2 Criterios de Selección

**Criterios para la Selección de Estrategias**

La elección del modelo de detección de anomalías es crítica para garantizar la precisión y eficiencia del sistema. Cada criterio está diseñado para considerar características específicas del conjunto de datos:

**Dimensionalidad:**

* **Criterio:** Si el número de columnas (características) en el conjunto de datos es mayor a 30, se seleccionan Autoencoders.
* **Razonamiento:** Con más de 30 dimensiones, los datos pueden volverse esparcidos, lo que dificulta identificar relaciones significativas entre las características. Los Autoencoders emplean redes neuronales para aprender una representación comprimida del conjunto de datos, preservando las relaciones entre las características. Esto no solo mejora la precisión de la detección, sino que también reduce el ruido inherente de los datos.
* **Aplicación práctica:** Conjuntos de datos complejos provenientes de sensores industriales con muchas métricas diferentes (como temperatura, presión, flujo, etc.).

**Volumen:**

* **Criterio:** Para conjuntos de datos con más de 1 millón de filas, se utiliza Isolation Forest.
* **Razonamiento:** Isolation Forest está diseñado para ser escalable y eficiente, manejando grandes volúmenes de datos sin comprometer la velocidad. En lugar de analizar cada dato en relación con todos los demás, este algoritmo utiliza divisiones aleatorias para identificar puntos que son más fáciles de aislar como anomalías.
* **Aplicación práctica:** Detección de anomalías en sistemas de monitoreo continuo que generan datos en tiempo real.

**Distribución Normal:**

* **Criterio:** Si los datos presentan una distribución normal, se aplica ZScore.
* **Razonamiento:** ZScore utiliza la distancia estándar desde la media para identificar datos fuera de lo común. Es especialmente efectivo para detectar anomalías en conjuntos de datos bien distribuidos y sin sesgos significativos.
* **Aplicación práctica:** Sistemas donde las métricas (como tiempos de respuesta o consumo energético) siguen una distribución predecible.

**Densidad:**

* **Criterio:** Si los datos contienen clusters densos y áreas de baja densidad, se selecciona DBSCAN.
* **Razonamiento:** Este algoritmo agrupa puntos densos y clasifica como anomalías aquellos que se encuentran aislados o en regiones de baja densidad. Es robusto frente a ruido y eficaz en datos que contienen patrones complejos o distribuciones heterogéneas.
* **Aplicación práctica:** Análisis de datos de tráfico en redes donde se espera detectar flujos anómalos dispersos.

2.2.2.3 Evaluación Comparativa

Se realizaron pruebas con cada algoritmo para evaluar su precisión y tiempo de ejecución.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algoritmo | Rapidez | Precisión | Volumen | Interpretabilidad | Escalabilidad | Caso Ideal |
| Isolation Forest | Alta | Alta | Grande | Media | Alta | Ideal para grandes volúmenes de datos con estructuras complejas y distribuciones desconocidas; balancea rapidez y precisión. |
| Autoencoders | BajaMedia | Muy Alta | Grande | Media | Media | Excelente para datos complejos y no lineales; adecuado cuando la precisión es prioritaria y hay capacidad de cómputo disponible. |
| DBSCAN | Media | Alta | PequeñoMediano | MediaBaja | Baja | Bueno para datos con patrones espaciales claros; no es ideal para datos de alta dimensión. |
| ZScore | Muy Alta | Media | PequeñoMediano | Alta | Muy Alta | Detección simple en datos distribuidos normalmente; útil cuando se requiere velocidad. |

### 2.2.3 Generación de Resultados

#### 2.2.3.1 Clasificación de Datos

El modelo seleccionado clasifica los datos en dos categorías: Normal y Anomaly. Este proceso se basa en puntajes de anomalía calculados por cada modelo.

**Columna Generada:**Se agrega una columna denominada anomaly al conjunto de datos.

**Valores:**

* "Normal": Representa instancias que se alinean con el comportamiento esperado.
* "Anomaly": Indica instancias que exceden los límites establecidos por el modelo.

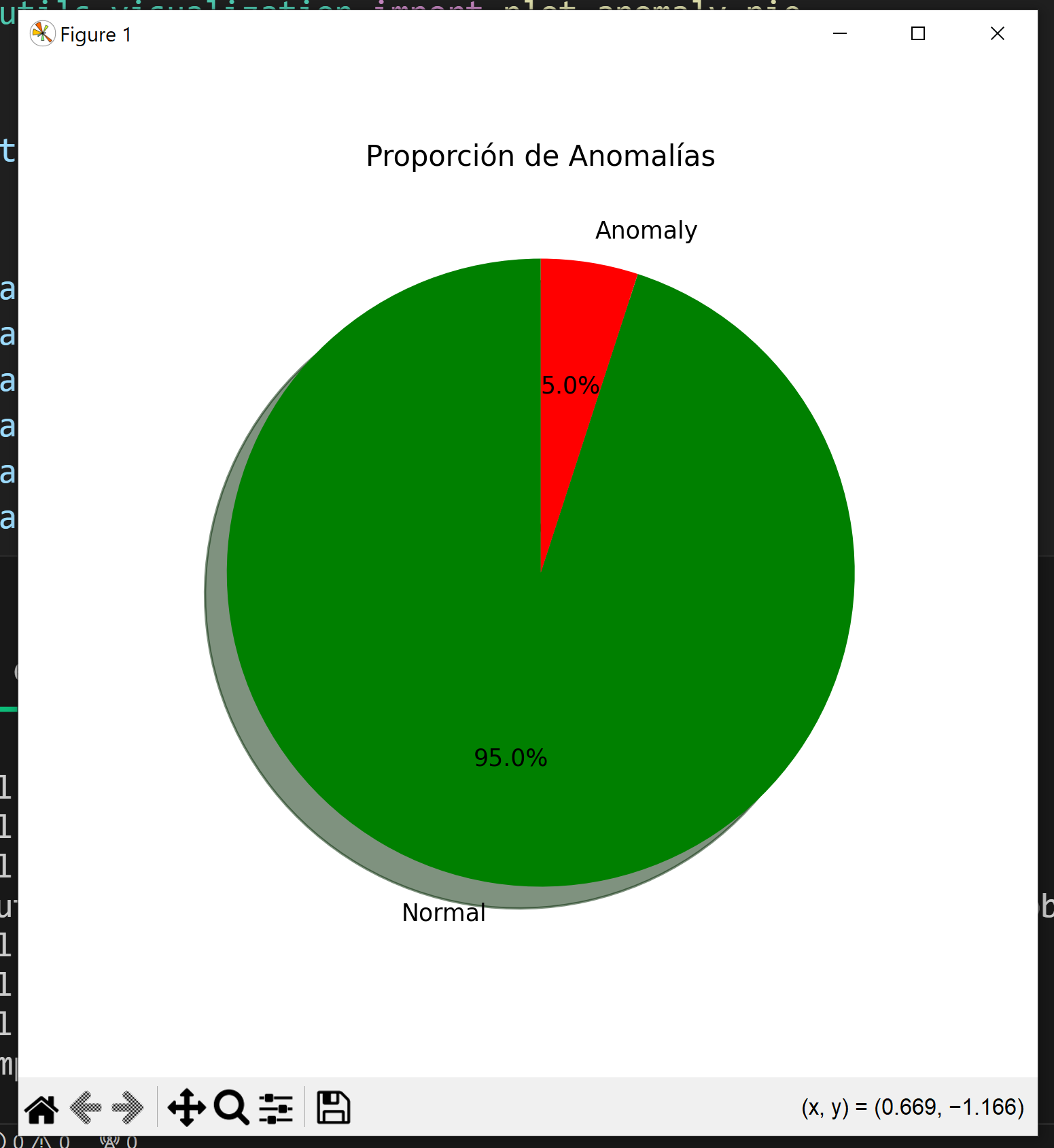
**Puntaje de Anomalía:** Se incluye una columna adicional anomaly\_score que cuantifica la distancia de cada instancia respecto a los datos normales.

**Interpretación:** Valores más altos indican una mayor probabilidad de ser anomalías.

#### 2.2.3.2 Visualización

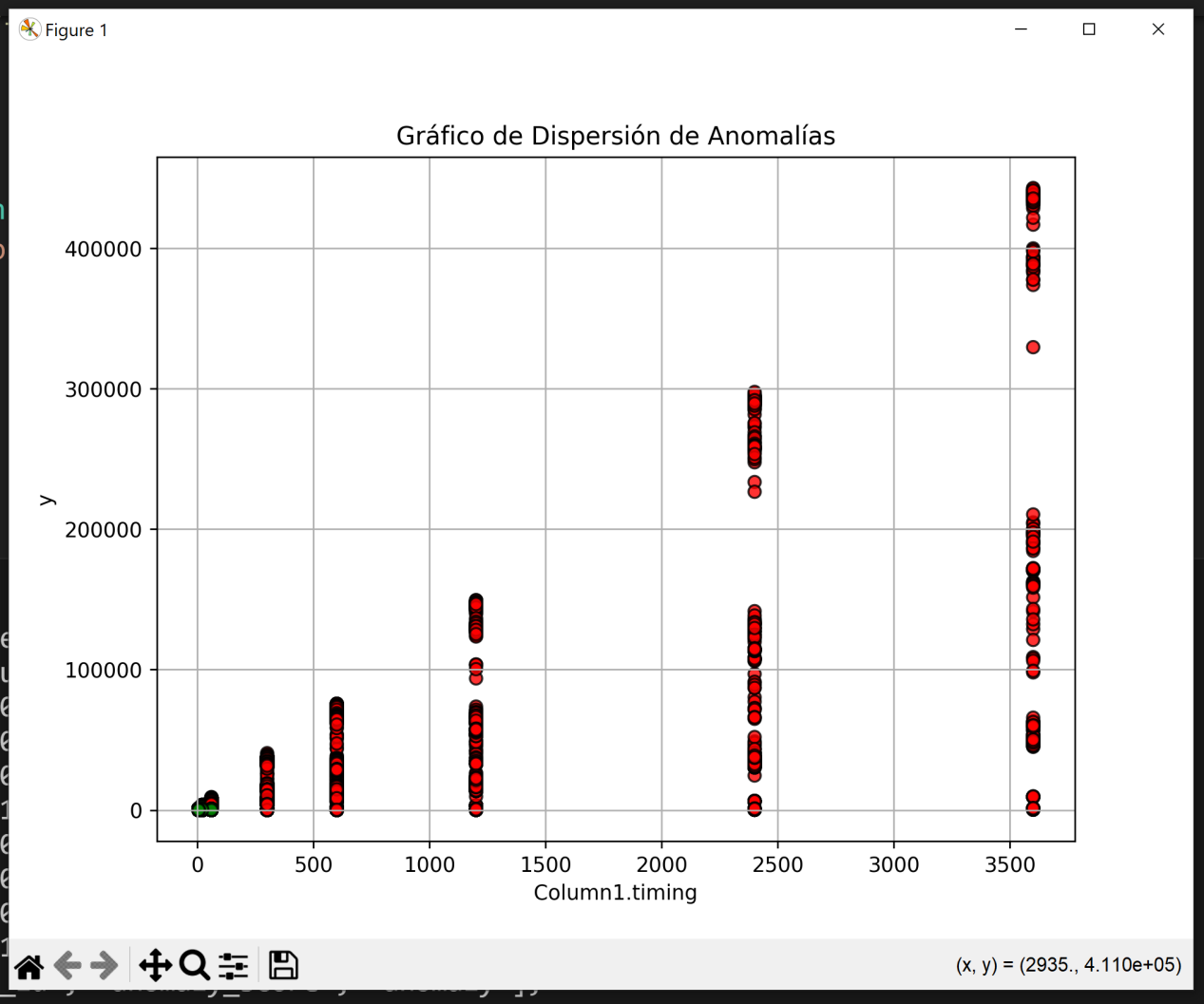
Se generan gráficos para interpretar los resultados:

1. Gráfico de pastel: Muestra la proporción de datos normales y anómalos.



*Figura 7* *Ejemplo de resultados representados con un Gráfico de Pastel*

1. Gráficos de dispersión: Ilustran cómo se distribuyen las anomalías en el espacio de características.



*Figura 8* *Ejemplo de resultados representados con un* Gráfico *de Dispersión*

### 2.2.4 Interpretación de Resultados mediante Reglas

2.2.4.1 Generación de Reglas

Para facilitar la interpretación, se generan reglas que explican las razones detrás de la clasificación de una instancia como anómala.

**Generación de Reglas:**

Las reglas se basan en análisis estadísticos como el rango intercuartílico (IQR) y los valores mínimo y máximo.

**Ejemplo de Regla Generada:**

Si Column1.timing > 5 y Column1.tcp\_count < 30, entonces la instancia 42 es anómala.

**Método de Generación:**

Se calculan los límites de las características usando:

Límite inferior: Q1 1.5 × IQR

Límite superior: Q3 + 1.5 × IQR

Cada instancia se evalúa respecto a estos límites, y las desviaciones se convierten en condiciones de las reglas.

**Aplicación Práctica:**

Estas reglas son útiles para que los operadores del sistema comprendan las causas subyacentes de las anomalías y tomen decisiones informadas sobre el mantenimiento o ajustes operativos.

### 2.2.5 Conclusión

La solución propuesta abarca todas las etapas necesarias para la detección efectiva de anomalías. Desde un preprocesamiento exhaustivo hasta una interpretación comprensible de los resultados, este enfoque asegura la robustez y escalabilidad necesarias en entornos industriales. Además, la flexibilidad del sistema permite adaptarse a diferentes escenarios y tipos de datos, maximizando su aplicabilidad y eficiencia.

## 2.3 Ingeniería de Requisitos

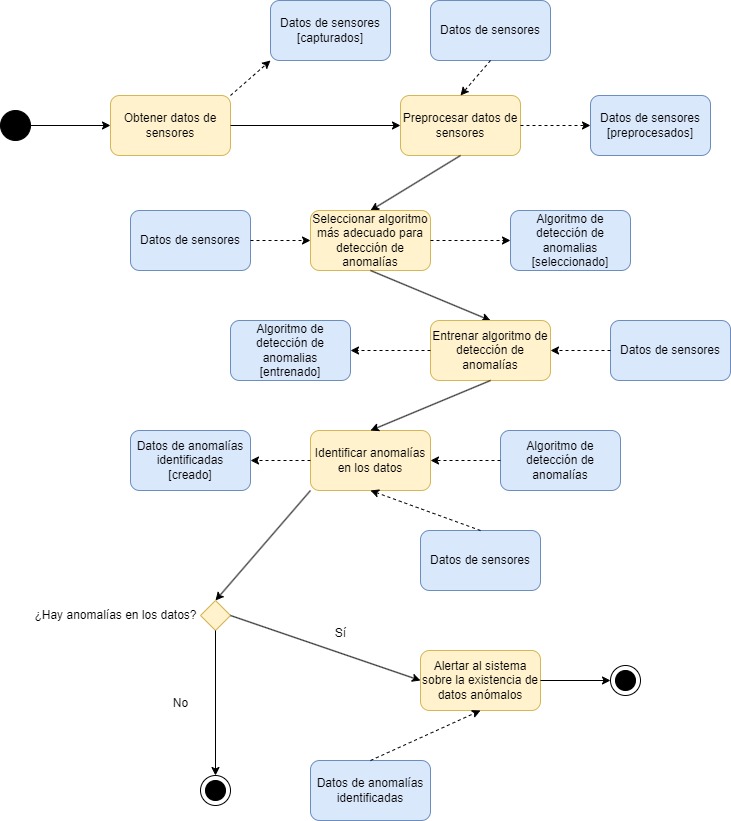
Esta sección proporciona una visión integral de los requisitos necesarios para el desarrollo del sistema, abordando tanto los aspectos funcionales como no funcionales, el flujo de datos, el modelo de dominio, y se incluye un glosario de términos clave. Además, se presentan los requisitos funcionales que describen las acciones específicas que el software debe realizar y los requisitos no funcionales que destacan las cualidades deseadas del sistema, así como las restricciones y problemas frecuentes que deben considerarse durante el proceso de desarrollo.

### 2.3.1 Breve descripción de la propuesta

La propuesta se centra en desarrollar soluciones avanzadas de detección de anomalías en sistemas industriales, diseñadas para identificar y mitigar comportamientos anómalos en tiempo real. Esto garantiza la seguridad operativa, la continuidad de los procesos y la optimización del rendimiento en entornos industriales críticos.

### 2.3.2 Diagrama de actividades

En la Figura 2 Diagrama de Actividades se muestra el flujo que sigue el módulo de detección de anomalías para analizar y clasificar los datos operativos del sistema en normales o anómalos.

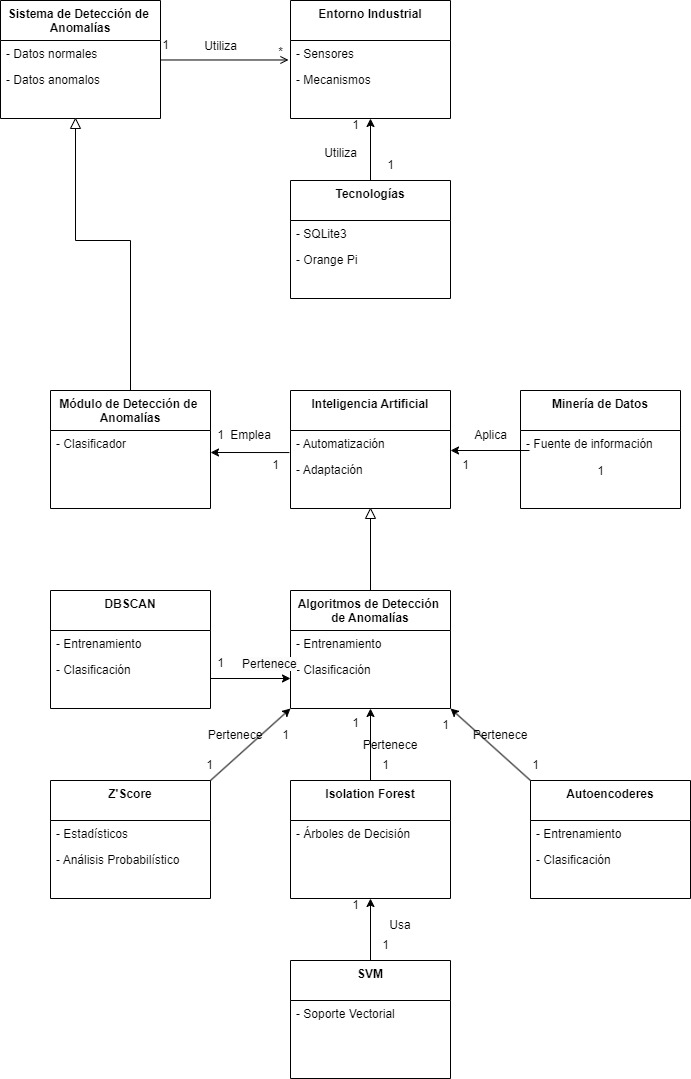


*Figura 9* *Diagrama de Actividades*

En primera instancia, los datos generados por los sensores y mecanismos del entorno industrial son capturados y procesados de manera eficiente. Posteriormente, estos datos son analizados para determinar el algoritmo de detección de anomalías más adecuado, priorizando criterios de precisión y velocidad. Una vez seleccionado el algoritmo óptimo, este es entrenado con datos históricos y procesados para garantizar su desempeño en el contexto específico del sistema.

Tras el entrenamiento, el algoritmo clasifica los datos operativos en dos categorías: normales y anómalos. En caso de detectar anomalías, el sistema genera y envía una alerta inmediata a los componentes correspondientes, permitiendo una acción proactiva. Por el contrario, si no se identifican irregularidades, el sistema continúa operando sin emitir notificaciones, optimizando recursos y asegurando un flujo operativo continuo.

### 2.3.2 Modelo de Dominio



*Figura 10* *Modelo de dominio*

### 2.3.3 Glosario de Términos

**Inteligencia Artificial**: Campo de estudio y desarrollo tecnológico que busca crear sistemas o máquinas capaces de realizar tareas que requieren inteligencia humana, como el análisis de datos, toma de decisiones y detección de patrones complejos.

**Minería de Datos**: Proceso de extracción de información valiosa, patrones ocultos y conocimientos significativos a partir de grandes conjuntos de datos generados en sistemas industriales.

**Aprendizaje Automático**: Rama de la inteligencia artificial enfocada en desarrollar algoritmos y modelos que permiten a las máquinas aprender y mejorar automáticamente a partir de los datos sin ser programadas explícitamente.

**Aprendizaje Supervisado**: Técnica de aprendizaje automático en la que se entrena un modelo utilizando datos etiquetados, donde se proporcionan tanto las características de entrada como las etiquetas correspondientes. En el contexto industrial, se utiliza para clasificar datos operativos como normales o anómalos.

**Sensores Industriales**: Dispositivos que recopilan datos sobre las condiciones operativas de un entorno industrial, como temperatura, presión, vibraciones o flujo, los cuales son fundamentales para la detección de anomalías.

**Mecanismos Industriales**: Componentes físicos y operativos de una planta industrial que generan datos de funcionamiento y cuya monitorización es esencial para identificar fallos o anomalías.

**SQLite3:** Sistema de gestión de bases de datos ligero e incrustado, utilizado para almacenar y gestionar los datos generados por sensores y procesados por el módulo de detección de anomalías.

**Orange Pi**: Miniordenador utilizado como servidor dentro del entorno industrial, encargado de centralizar y procesar los datos recolectados por los sensores.

**SCADA (Supervisory Control and Data Acquisition)**: Sistema de control industrial que permite supervisar y adquirir datos en tiempo real desde los sensores y mecanismos, facilitando la integración con el módulo de detección de anomalías.

**STM32**: Microcontrolador que gestiona las interacciones entre los sensores y mecanismos industriales, y que juega un papel clave en el flujo de datos hacia el sistema de detección de anomalías.

## 2.4 Requisitos

En este apartado se detallan los requisitos del sistema, divididos en funcionales y no funcionales. Los requisitos funcionales describen las acciones específicas que el sistema debe realizar para cumplir con su propósito, mientras que los requisitos no funcionales definen atributos de calidad, restricciones o limitaciones que deben garantizarse para asegurar el desempeño y la efectividad del sistema.

2.4.1 Requisitos Funcionales

Los requisitos funcionales especifican las funciones y capacidades que el sistema debe cumplir. Cada requisito se define con claridad y precisión, enfocándose en una única acción específica para garantizar que el sistema cumpla su propósito de detectar anomalías en datos de manera eficiente.

RF1: El sistema debe recibir los datos a analizar en distintos formatos (CSV, XLSX, entre otros).

RF2: El sistema debe eliminar columnas que no aporten información relevante al análisis de anomalías.

RF3: El sistema debe llenar los valores nulos presentes en los datos con valores predeterminados o calculados.

RF4: El sistema debe transformar los datos a un formato adecuado para el análisis, asegurando su compatibilidad con los algoritmos utilizados.

RF5: El sistema debe seleccionar automáticamente la estrategia de detección de anomalías más adecuada según las características de los datos.

RF6: El sistema debe entrenar el modelo seleccionado utilizando los datos preprocesados.

RF7: El sistema debe clasificar los datos en instancias normales o anómalas, asignando etiquetas correspondientes.

RF8: El sistema debe generar un archivo de salida que contenga los datos clasificados, incluyendo las etiquetas y los puntajes de anomalía.

RF9: El sistema debe mostrar gráficos, como diagramas de pastel, para visualizar la proporción de anomalías detectadas.

RF10: El sistema debe procesar conjuntos de datos de diferentes tamaños, incluyendo grandes volúmenes de datos.

### 2.4.2 Requisitos no funcionales. Problemas frecuentes

Los requisitos no funcionales establecen los criterios de calidad, restricciones y limitaciones que garantizan que el sistema no solo sea funcional, sino también eficiente, confiable y fácil de usar. Estos requisitos son esenciales para que el sistema opere correctamente bajo diversas condiciones.

#### 2.4.2.1 Requisitos de Calidad

**Rendimiento:**

* El sistema deberá procesar los datos y generar resultados en un tiempo razonable para garantizar su aplicabilidad en contextos prácticos.
* La precisión de los modelos entrenados deberá superar el 90% en la identificación de anomalías.

**Fiabilidad:**

* El sistema deberá minimizar falsos positivos y falsos negativos en la detección de anomalías.

**Usabilidad:**

* El sistema deberá proporcionar una interfaz clara para la carga de datos y visualización de resultados.

#### 2.4.2.2 Restricciones

**Compatibilidad de Archivos:**

* El sistema deberá admitir únicamente archivos CSV y XLSX como entradas.

**Dependencia de Preprocesamiento:**

* Es obligatorio preprocesar los datos antes de realizar cualquier análisis o modelado.

**Reentrenamiento Periódico:**

* Los modelos deberán reentrenarse regularmente para garantizar su validez frente a cambios en los patrones de los datos.

**Manejo de Datos en Memoria:**

* Para conjuntos de datos grandes, el sistema deberá utilizar métodos eficientes de carga y análisis que minimicen el uso de memoria.

## 2.5 Arquitectura Candidata

Dado el objetivo de desarrollar un módulo eficiente para la detección de anomalías en datos operativos, se evaluaron diferentes arquitecturas para satisfacer las necesidades de rendimiento, flexibilidad y escalabilidad. Estas características son esenciales para procesar grandes volúmenes de datos en tiempo real y proporcionar resultados precisos.

Inicialmente, se consideró una arquitectura monolítica como base experimental para simplificar el desarrollo y las pruebas. A medida que el proyecto crezca en complejidad o requisitos de integración, arquitecturas más avanzadas podrían proporcionar ventajas significativas.

### 2.5.1 Estudio de las Tecnologías Candidatas

**Arquitectura Monolítica:**

Esta arquitectura integra toda la lógica del sistema en una única unidad. Es ideal para la fase inicial del desarrollo debido a su simplicidad y facilidad de despliegue. Aunque no es la más escalable, permite una implementación rápida y eficiente para sistemas experimentales como el módulo de detección de anomalías.

**Arquitectura Modular:**

Esta arquitectura organiza el sistema en módulos independientes que pueden comunicarse a través de interfaces bien definidas. Por ejemplo, un módulo podría encargarse exclusivamente del preprocesamiento de datos, mientras que otro gestionaría la selección de estrategias. Aunque no tan completamente desacoplada como los microservicios, proporciona un nivel intermedio de flexibilidad y escalabilidad.

**Arquitectura basada en Tareas:**

En esta arquitectura, cada componente del sistema se encarga de realizar una tarea específica y pasa los datos procesados al siguiente componente. Este enfoque permite optimizar flujos de trabajo y facilita la depuración de errores al identificar responsabilidades claras.

**Selección de Arquitectura:**

Para la fase actual, se seleccionó una arquitectura monolítica debido a su simplicidad y efectividad en entornos experimentales. No obstante, el diseño modular interno asegura que, en futuras iteraciones, se pueda migrar hacia una arquitectura más avanzada y escalable si el proyecto lo requiere.

### 2.5.2 Lenguajes de Programación Candidatos

**1. Python**

Python fue seleccionado como el lenguaje principal del proyecto debido a:

**Sintaxis Intuitiva y Clara:** Ideal para proyectos experimentales, ya que facilita la comprensión y colaboración.

**Extenso Ecosistema de Bibliotecas:** Herramientas como Pandas, NumPy y Scikitlearn optimizan tareas de preprocesamiento y modelado.

**Comunidad Activa:** La disponibilidad de recursos y documentación garantiza soporte continuo durante el desarrollo.

**2. Rust**

Rust destaca como una opción moderna que ofrece:

**Alto Rendimiento:** Su enfoque en la gestión de memoria sin un recolector de basura lo hace ideal para aplicaciones críticas en tiempo real.

**Seguridad y Concurrencia:** Facilita la creación de sistemas robustos que evitan errores comunes relacionados con la memoria.

**3. Julia**

Julia presenta ventajas específicas en el campo de los datos y la IA:

**Optimización Matemática:** Diseñado para cálculos numéricos y estadísticos de alto rendimiento.

**Flexibilidad Científica:** Ideal para proyectos con requisitos avanzados de modelado matemático.

**Selección de Lenguaje:**

Se eligió Python como lenguaje de implementación para el módulo de detección de anomalías debido a su facilidad de uso, extensa biblioteca de herramientas y capacidad de integración con tecnologías existentes.

2.5.3 Tratamiento de Problemas Frecuentes

Durante el desarrollo del módulo, se identificaron y abordaron varios problemas comunes utilizando las tecnologías seleccionadas:

**Calidad de Datos:**

Se utilizaron herramientas como Pandas para limpiar y transformar datos, asegurando su formato adecuado.

**Escalabilidad:**

Se adoptaron prácticas de optimización utilizando NumPy para garantizar un rendimiento eficiente en volúmenes de datos moderados.

**Evaluación de Modelos:**

Se implementaron técnicas de validación cruzada con Scikitlearn para ajustar modelos y evitar problemas como sobreajuste o subajuste.

**Flexibilidad y Modificación:**

La modularidad interna del sistema permite actualizaciones sin interrumpir el flujo principal de procesamiento.

**Rendimiento:**

El uso de estructuras eficientes y algoritmos optimizados aseguró tiempos de respuesta adecuados para los análisis realizados.

Esta aproximación asegura que el sistema esté preparado para manejar desafíos técnicos comunes y se pueda adaptar fácilmente a las necesidades futuras del proyecto.

## 2.6 Estilos y Patrones de Arquitectura Identificados

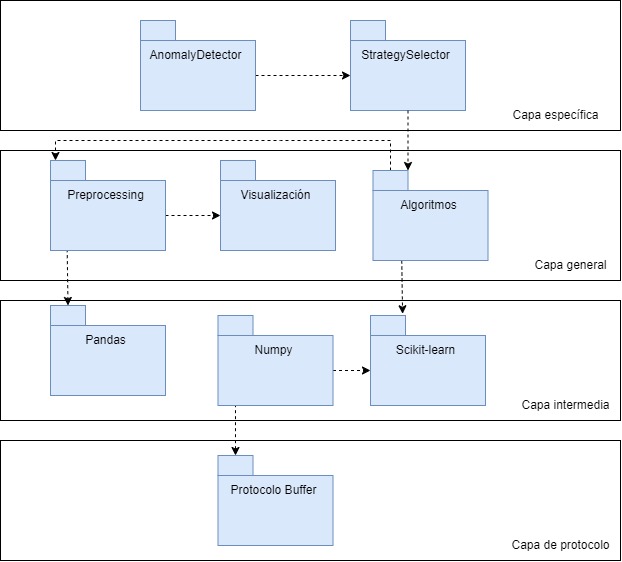
Un estilo arquitectónico en software define cómo organizar y estructurar un sistema, dividiendo responsabilidades y determinando cómo interactúan sus componentes. En el caso del módulo de detección de anomalías, los estilos y patrones seleccionados se basan en la naturaleza de las tareas, como el procesamiento de datos, la selección de estrategias y la generación de resultados. A continuación, se describen los estilos arquitectónicos identificados: llamada y retorno, centrado en datos y flujo de datos.

2.6.1 Llamada y Retorno

El estilo llamada y retorno estructura el sistema en torno a invocaciones directas de funciones, donde cada componente realiza una tarea específica y devuelve el control al componente llamador. Este estilo es apropiado para sistemas modulares como el módulo de detección de anomalías, donde diferentes procesos trabajan de forma aislada y coordinada.

#### 2.6.1.1 Patrón NCapas con Enfoque en Reutiilzación

El patrón ncapas organiza el sistema en niveles jerárquicos de abstracción, promoviendo la reutilización de componentes en diferentes contextos dentro del módulo de detección de anomalías. A continuación, se puede ver en la Figura 4 Patrón NCapa.



*Figura 11* *Patrón NCapa*

**1. Capa Específica**

Esta capa contiene los componentes especializados que implementan la lógica específica del módulo de detección de anomalías.

**StrategySelector:** Encargado de seleccionar la estrategia más adecuada para los datos proporcionados, considerando características como dimensionalidad y distribución.

**AnomalyDetector:** Aplica la estrategia seleccionada para entrenar modelos y clasificar los datos como normales o anómalos. Sus métodos son específicos para este módulo y no son reutilizados por otros sistemas.

**2. Capa General**

Esta capa incluye elementos reutilizables que ofrecen funcionalidades comunes y de amplio uso dentro del sistema.

**Preprocessing:** Contiene métodos para limpiar, transformar y normalizar los datos, asegurando que estén en un formato adecuado para el análisis.

**Visualización:** Proporciona herramientas como gráficos de pastel y visualizaciones interactivas, que son reutilizables para la presentación de resultados en diferentes contextos.

**3. Capa Intermedia**

La capa intermedia está compuesta por bibliotecas externas que proporcionan funcionalidades reutilizables para operaciones matemáticas y aprendizaje automático.

**Pandas:** Manejo de estructuras de datos como DataFrames, ideal para la manipulación eficiente de datos tabulares.

**NumPy:** Operaciones matemáticas avanzadas y soporte para arreglos multidimensionales.

**Scikitlearn:** Implementación de algoritmos de aprendizaje automático como Isolation Forest y DBSCAN, así como herramientas para evaluación y validación de modelos.

**4. Capa de Protocolo**

Esta capa se centra en la interacción eficiente con los datos y la memoria del sistema, utilizando protocolos específicos del lenguaje de programación para optimizar el manejo de recursos.

**Protocolo Buffer:** Optimiza el manejo de estructuras como arreglos y vectores, utilizados en bibliotecas de análisis numérico, garantizando un desempeño eficiente en tiempo de ejecución.

Este enfoque jerárquico no solo organiza las responsabilidades del sistema, sino que también garantiza la reutilización y escalabilidad, facilitando futuras extensiones o integraciones del módulo de detección de anomalías.

2.6.2 Centrado en Datos

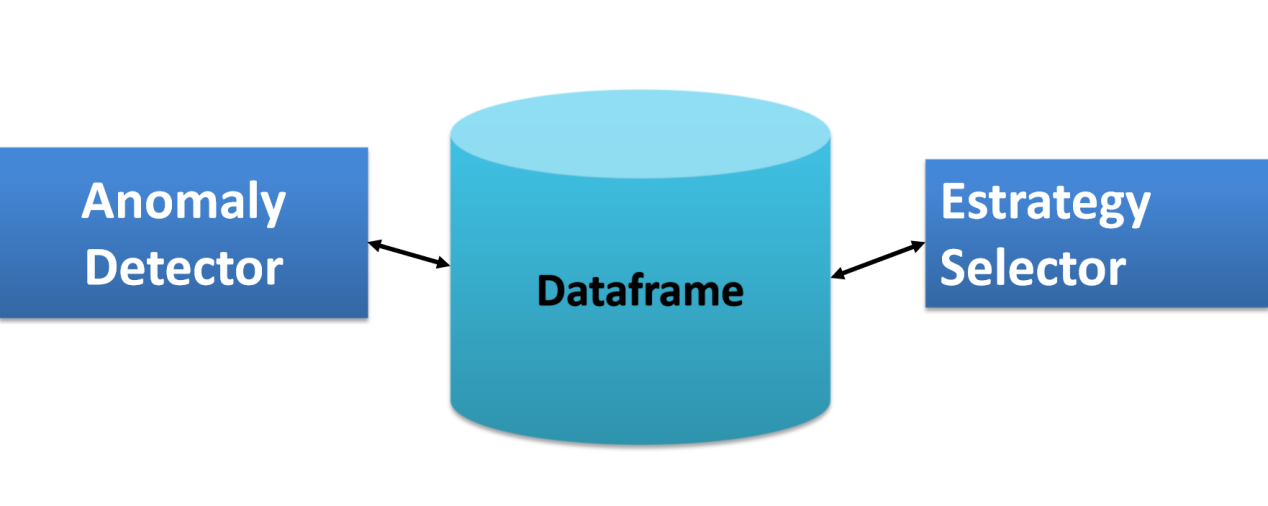
Este estilo arquitectónico se enfoca en procesar, estructurar y manejar los datos como núcleo del sistema. En el módulo, la preparación y transformación de datos es crítica para garantizar la precisión de los modelos de detección de anomalías.

#### 2.6.2.1 Patrón Repository

El patrón Repository se aplica en sistemas donde un conjunto centralizado de datos actúa como la fuente principal para todas las operaciones, permitiendo un acceso consistente y organizado a la información por parte de diferentes componentes del sistema. En este módulo:

**Repositorio Central de Datos**: Los datos preprocesados se almacenan en un DataFrame, que funciona como un repositorio centralizado. Este repositorio es accesible para todos los componentes del sistema, como los módulos de selección de estrategia, entrenamiento de modelos y clasificación de anomalías.

**Procesamiento Modular:** Los componentes como `StrategySelector` y `AnomalyDetector` operan directamente sobre este repositorio, extrayendo los datos necesarios para analizar, entrenar y clasificar anomalías. Cada etapa actualiza el estado de los datos en el repositorio, lo que garantiza la coherencia durante el flujo de ejecución.



*Figura 12* *Patrón Repository*

El diseño centrado en el repositorio asegura que todas las etapas del flujo (preprocesamiento, selección de estrategia, entrenamiento y clasificación) trabajen sobre un único conjunto de datos compartido, promoviendo la organización, la reutilización y la integración modular del sistema.

2.6.3 Flujo de Datos

Este estilo organiza el sistema como un flujo continuo, donde los datos pasan por una serie de etapas que aplican transformaciones específicas.

#### 2.6.3.1 Pipes and Filters

El módulo de detección de anomalías implementa este patrón como una serie de pasos interdependientes:

**1. Entrada de Datos**

Los datos se cargan desde archivos en formato CSV o XLSX mediante la clase `DocumentFactory`.

**2. Preprocesamiento de Datos**

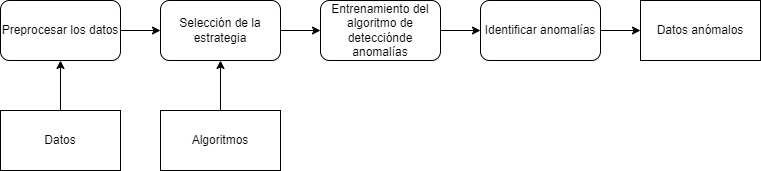
Los datos se limpian, normalizan y preparan para su análisis.

**3. Selección de Estrategia**

Con base en las características de los datos, el componente `StrategySelector` elige la mejor estrategia de detección, como Isolation Forest, DBSCAN, ZScore o Autoencoders.

**4. Entrenamiento y Clasificación**

El componente `AnomalyDetector` entrena el modelo seleccionado y clasifica los datos en normales o anómalos.



*Figura 13* *Patrón Pipes and Filters*

El flujo asegura que cada etapa realice su tarea de manera autónoma, permitiendo escalabilidad y adaptabilidad en el procesamiento de datos.

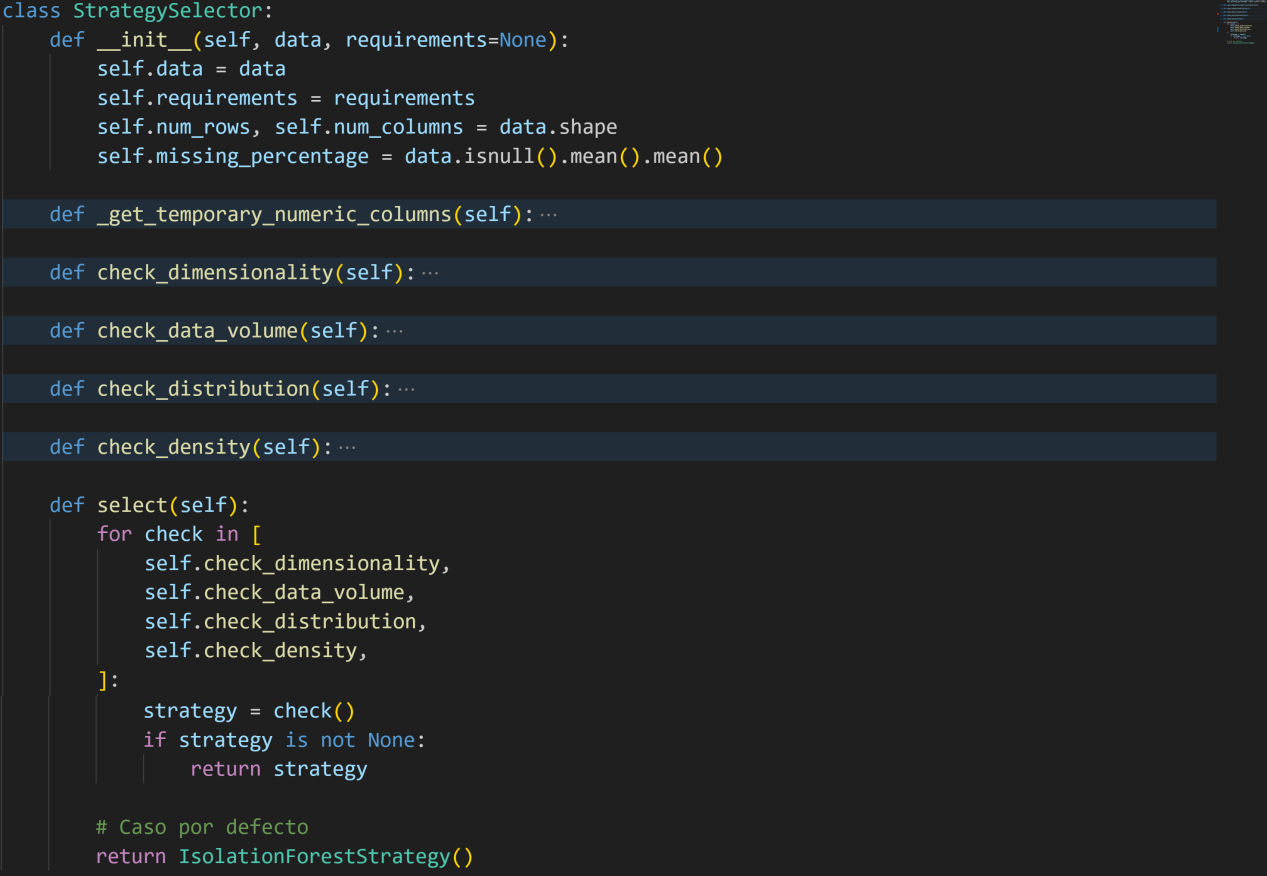
## 2.7 Patrones de Diseño

Al analizar los componentes principales y el flujo de trabajo, se pueden identificar patrones de diseño y mejoras que refactorizan el código para hacerlo más mantenible y eficiente.

**1. Patrón Strategy**

Dónde se aplica:

En el módulo `StrategySelector` y en las clases asociadas (`AutoencodersStrategy`, `IsolationForestStrategy`, etc.).



*Figura 14* *Clase StrategySelector*

Cómo se manifiesta:

El sistema selecciona dinámicamente la estrategia de detección de anomalías más adecuada (como Isolation Forest, Autoencoders, DBSCAN, etc.) dependiendo de las características del dataset.

Ventaja:

Facilita la extensión al permitir agregar nuevas estrategias sin modificar el código existente.

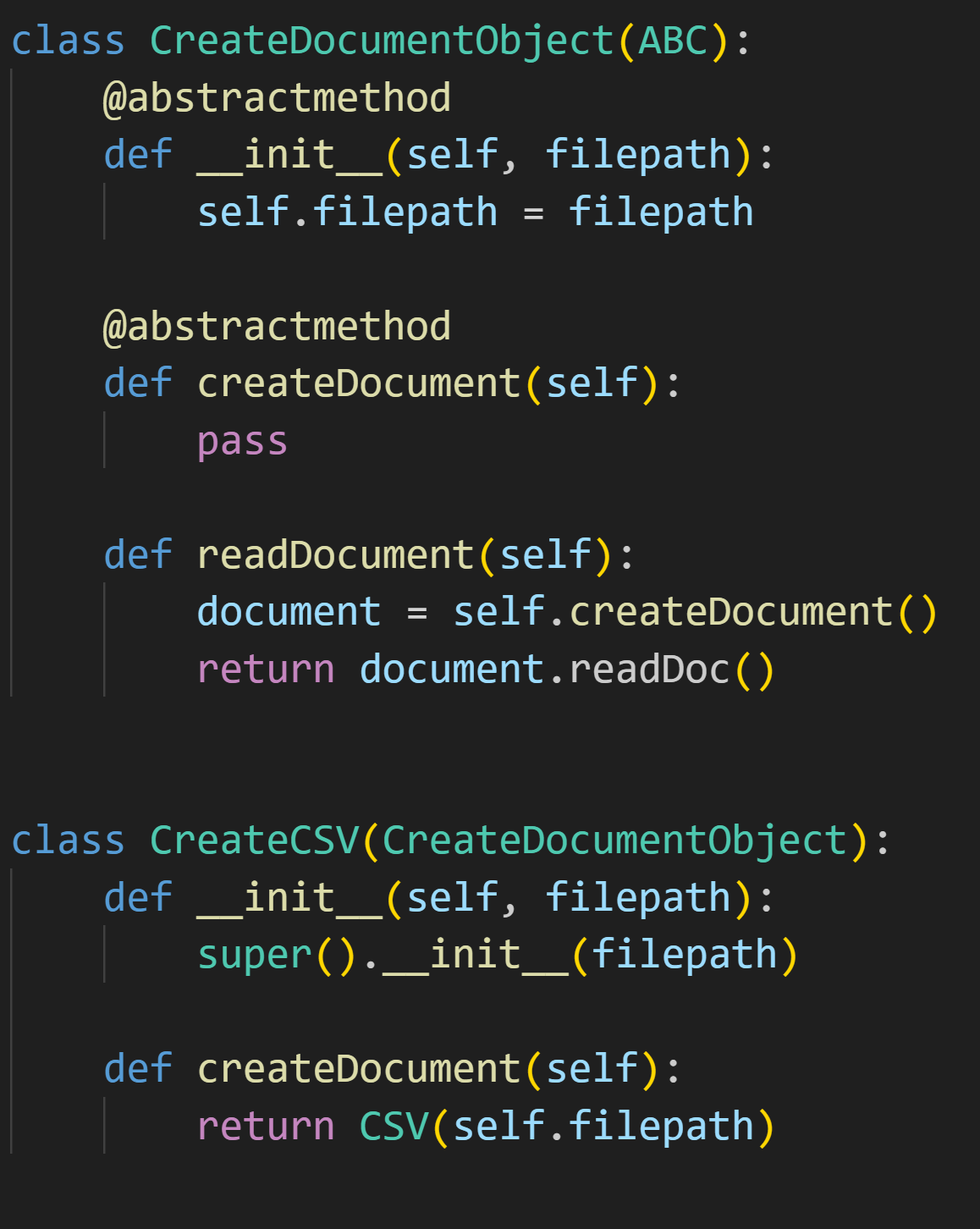
Principio asociado:

Abierto/Cerrado: Las clases están abiertas para extensión, pero cerradas para modificación.

**2. Patrón Factory Method**

**Dónde se aplica:**

En el módulo `DocumentFactory`, donde se registran formatos de entrada (`csv`, `xlsx`, etc.) y se crean instancias específicas (por ejemplo, `CreateCSV`, `CreateExcel`).



*Figura 15* *Ejemplo de superclase y subclase donde se aplica el Factory Method*

**Cómo se manifiesta:**

El método `getDocument()` selecciona dinámicamente la clase de procesamiento adecuada en función de la extensión del archivo

**Ventaja:**

Simplifica la creación de objetos y desacopla el código del formato específico.

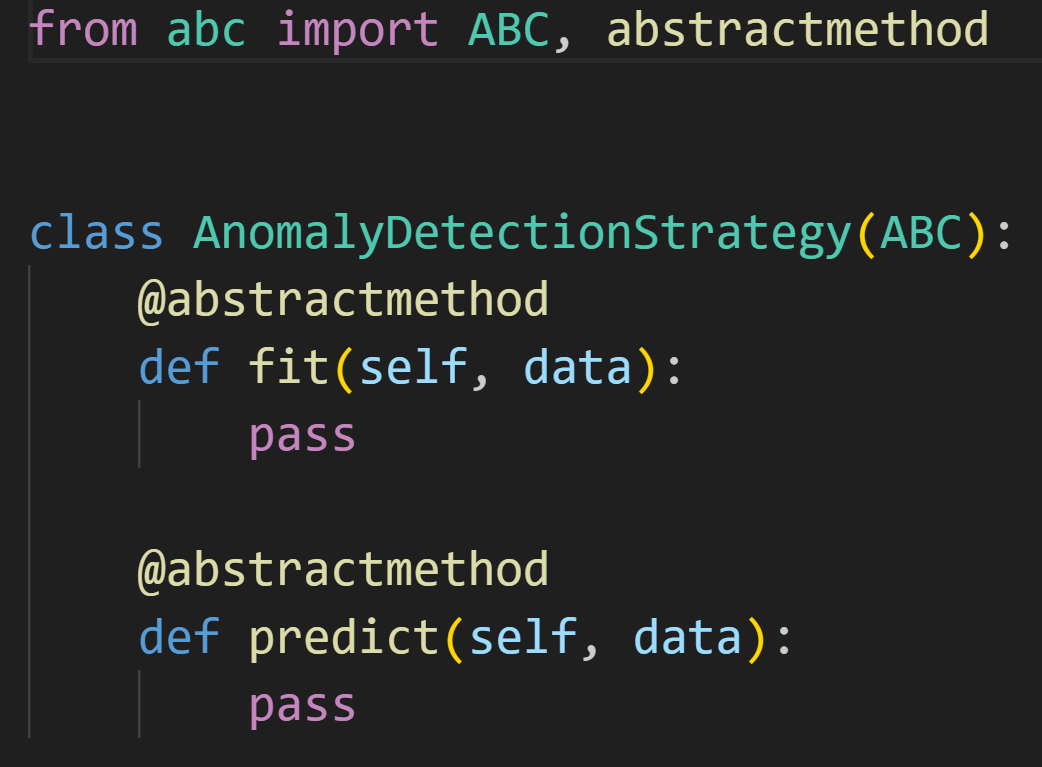
**Principio asociado:**

Responsabilidad Única: Cada subclase maneja un formato específico.

**3. Patrón Template Method**

**Dónde se aplica:**

En la implementación general del flujo de detección de anomalías. Por ejemplo, la clase base `AnomalyDetectionStrategy` define el flujo común (`fit`, `predict`), mientras que las subclases implementan detalles específicos.



*Figura 16* *Ejemplo de una clase que funciona como base para el Template Method*

**Cómo se manifiesta:**

La estructura general del entrenamiento y la predicción está definida, pero los detalles del algoritmo varían según la estrategia.

**Ventaja:**

Estandariza el proceso de detección mientras permite variaciones específicas.

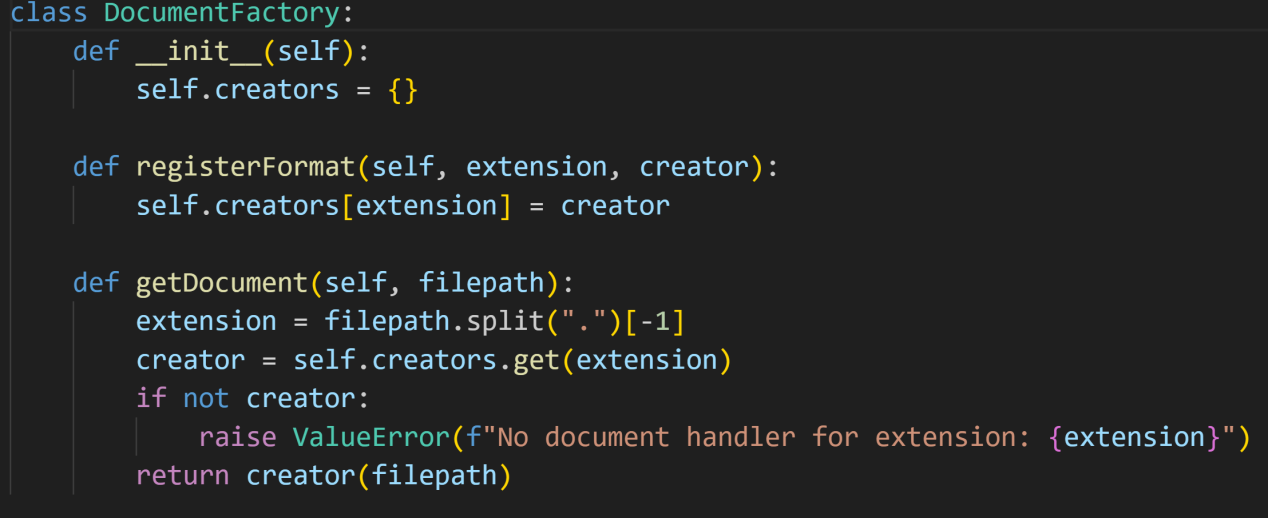
**Principio asociado:**

Inversión de Dependencias: Las subclases dependen de una interfaz común para implementar sus propios métodos.

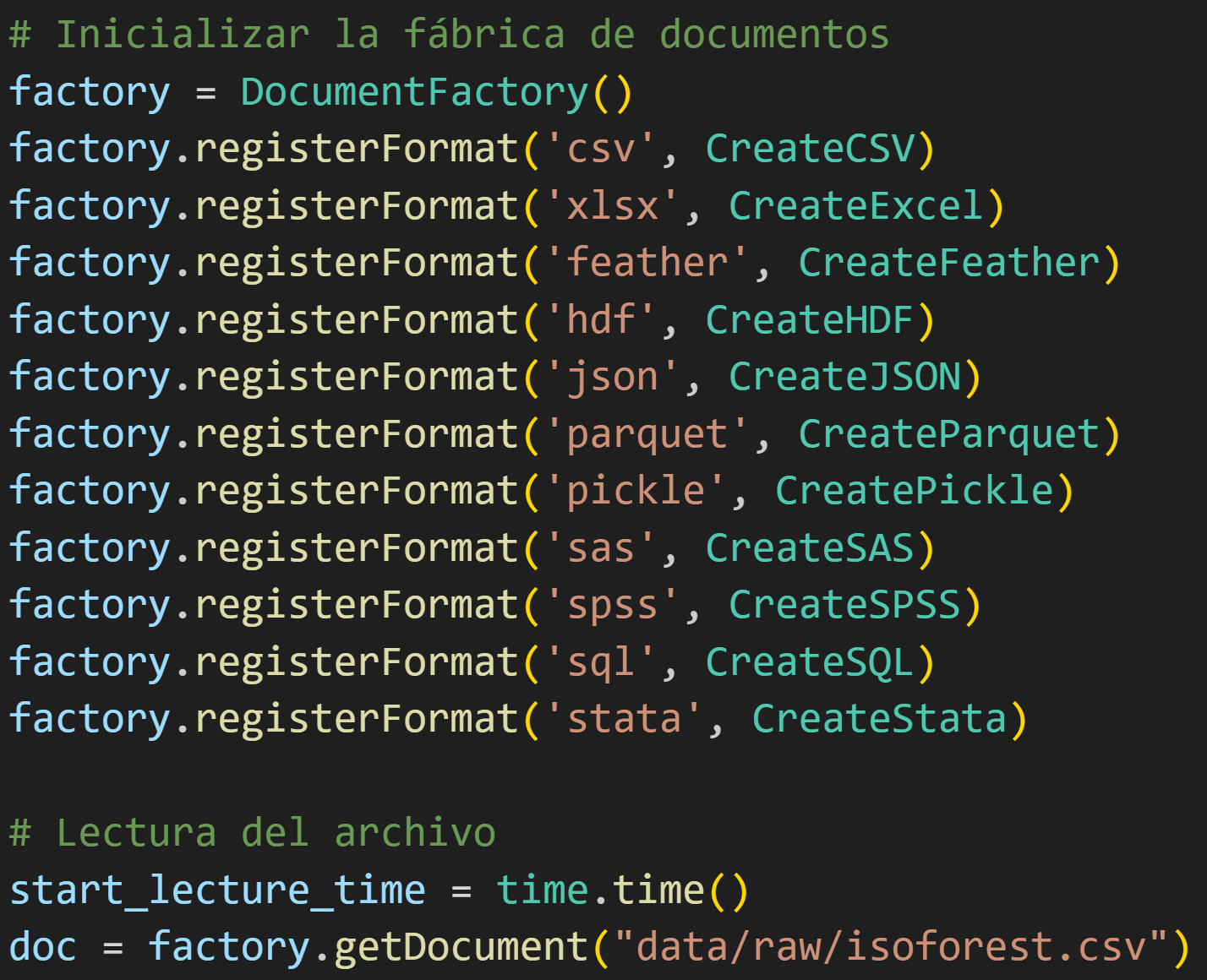
**4. Patrón Facade**

**Dónde se aplica:**

En la interacción con la capa de preprocesamiento y visualización.



*Figura 17* *Clase DocumentFactory con su implementación*



*Figura 18* *Ejemplo de objetos que la utilizan sin saber su complejidad interna*

**Cómo se manifiesta:**

Componentes como `DocumentFactory` ocultan la complejidad interna del sistema, proporcionando interfaces simples para los usuarios.

**Ventaja:**

Simplifica la interacción entre componentes y mejora la usabilidad del sistema.

**Principio asociado:**

Responsabilidad Única: Cada componente se enfoca en una tarea específica.

**Técnicas de Refactorización Identificadas**

**1. Simplificar la Creación de Objetos:**

**Refactorización aplicada:**

Uso del patrón Factory Method en `DocumentFactory` para manejar múltiples formatos de archivo. Ver Figura 18 Ejemplo de objetos que la utilizan sin saber su complejidad interna

**Mejora:**

Elimina condicionales en la creación de objetos y hace que el código sea más extensible y mantenible.

**2. Consolidar Lógica Común:**

**Refactorización aplicada:**

Uso de la clase base `AnomalyDetectionStrategy` para manejar métodos comunes (`fit`, `predict`). Ver Figura 16 Ejemplo de una clase que funciona como base para el Template Method

**Mejora:**

Reduce duplicación de código entre estrategias y centraliza cambios comunes.

**3. Parametrizar Métodos:**

**Refactorización aplicada:**

En el preprocesamiento de datos (`preprocess\_data`), se parametrizan columnas para garantizar flexibilidad.



*Figura 19* *Ejemplo de método parametrizado*

**Mejora:**

El código se puede reutilizar para diferentes estructuras de datos sin modificar su lógica.

**4. Manejo Eficiente de Condicionales:**

**Refactorización aplicada:**

Reemplazo de condicionales con estructuras como diccionarios (por ejemplo, en la selección de estrategias). Ver Figura 14 Clase StrategySelector

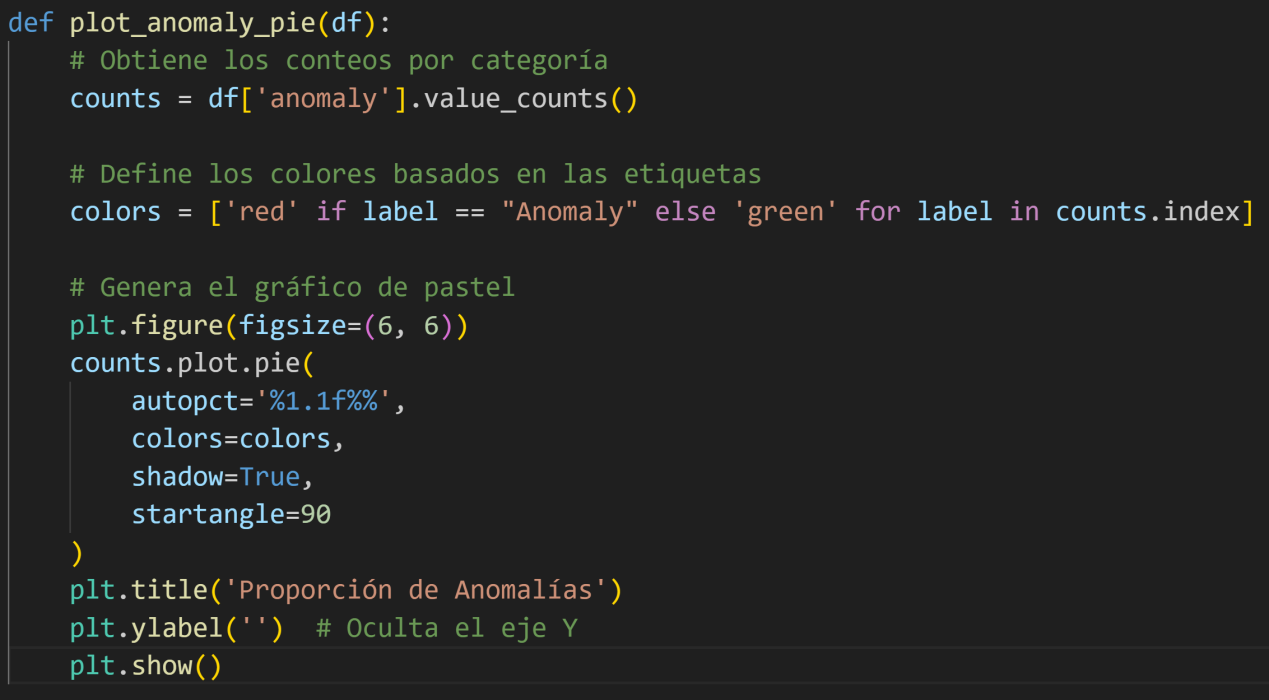
**Mejora:**

Hace que el código sea más legible y fácil de mantener.

**5. Optimización de la Visualización:**

**Refactorización aplicada:**

Uso de funciones específicas como `plot\_anomaly\_pie` para encapsular la lógica de generación de gráficos.



*Figura 20* *Clase usada para representar los resultados en un gráfico de pastel*

**Mejora:**

Elimina duplicación en el manejo de gráficos y centraliza el control de visualización.

## 2.8 Conclusiones Parciales

* Solución Propuesta:

La solución propuesta consolidó un flujo eficiente para la detección de anomalías, integrando preprocesamiento de datos, selección dinámica de estrategias y generación de resultados interpretables. Además, la generación de reglas interpretativas añadió un valor crucial para la comprensión de las anomalías detectadas, facilitando la aplicación práctica de los resultados. La solución demostró ser flexible y escalable, adaptable a diferentes volúmenes y complejidades de datos.

* Ingeniería de Requisitos:

La identificación y organización de los requisitos funcionales y no funcionales garantizó un desarrollo alineado con las necesidades del sistema. Se logró una solución que no solo satisface criterios técnicos como precisión y eficiencia, sino que también asegura robustez y facilidad de mantenimiento. Este enfoque contribuyó a minimizar riesgos durante la implementación y a asegurar que el sistema sea adaptable a futuras expansiones o modificaciones.

* Arquitectura Candidata:

La elección inicial de una arquitectura monolítica simplificó el desarrollo y la implementación en las etapas experimentales, demostrando ser una decisión acertada para obtener resultados rápidos y estables. Sin embargo, la evaluación de arquitecturas como microservicios y sistemas eventdriven permitió identificar oportunidades para escalar la solución en futuros escenarios más complejos, asegurando su relevancia y aplicabilidad a largo plazo.

* Estilos y Patrones de Arquitectura Identificados:

El uso de estilos arquitectónicos como llamada y retorno, centrado en datos y flujo de datos facilitó la modularidad, cohesión y claridad en el diseño del sistema. Estos enfoques garantizaron que cada componente cumpliera su función específica dentro del flujo general, mejorando la mantenibilidad y el rendimiento del sistema. La integración de estos estilos permitió que la solución sea fácilmente escalable y adaptable a nuevos requerimientos.

* Patrones de Diseño

La incorporación de patrones de diseño como Template Method, Strategy y Facade consolidó una estructura sólida para la solución. Estos patrones no solo simplificaron el desarrollo, sino que también facilitaron la reutilización y extensión de componentes. Por ejemplo, el patrón Strategy permitió la inclusión de múltiples algoritmos de detección de anomalías sin alterar el núcleo del sistema, mientras que Template Method estandarizó procesos como el entrenamiento y evaluación de modelos.

**Conclusión General del Capítulo 2**

A través del desarrollo e implementación de las estrategias y técnicas descritas en este capítulo, se alcanzó una solución robusta, eficiente y escalable para la detección de anomalías en sistemas industriales simulados. Este trabajo no solo proporciona herramientas para identificar problemas en tiempo real, sino que también establece una base sólida para futuras investigaciones y mejoras en el campo de los sistemas ciberfísicos y la Industria 4.0.

# Capitulo 3 Validación de la propuesta

En este capítulo se realizarán pruebas mediante métodos estadísticos para validar los resultados obtenidos por el módulo de detección de anomalías desarrollado en esta investigación, comparándolos con una solución preexistente implementada en Knime. Esta solución utiliza Isolation Forest en conjunto con H2O para la detección de anomalías en flujos de datos. El objetivo principal es demostrar que la implementación propuesta no solo es competitiva, sino que también puede superar a la solución existente en términos de precisión, eficiencia y capacidad de interpretación de resultados.

## 3.1 Descripción de los Experimentos

Para validar la efectividad de la solución propuesta, se diseñó un estudio experimental utilizando datasets generados sintéticamente mediante la librería Scikitlearn con la función make\_classification. Estos datasets fueron diseñados específicamente para evaluar diferentes características de los algoritmos de detección de anomalías. Cada conjunto de datos representa un escenario controlado, lo que permite probar y comparar los algoritmos nuevos implementados en Python con la solución existente en Knime.

**Escenarios de Datos**

Dataset con Distribución Normal:

* Generado con características basadas en una distribución normal.
* Diseñado para evaluar la efectividad del algoritmo ZScore.

Dataset de Alta Densidad:

* Datos organizados en clusters densos con anomalías dispersas.
* Ideal para probar el algoritmo DBSCAN, que identifica anomalías como puntos fuera de clusters.

Dataset de Alta Dimensionalidad:

* Contiene un gran número de características (dimensionalidad alta).
* Diseñado para evaluar la capacidad de compresión y reconstrucción de los Autoencoders.

Dataset Mixto:

* Proporción de datos normales y anomalías generadas con características extremas.
* Utilizado para probar Isolation Forest, tanto en Python como en Knime.

**Comparación con Knime**

La solución existente en Knime utiliza el algoritmo Isolation Forest sobre la plataforma H2O, conocido por su robustez y eficiencia en detección de anomalías. En contraste, la solución propuesta en Python adopta una estrategia adaptativa, seleccionando dinámicamente el algoritmo más adecuado según las características del dataset:

* ZScore para datasets con distribución normal.
* DBSCAN para escenarios con alta densidad.
* Autoencoders para datos de alta dimensionalidad.
* Isolation Forest como enfoque generalista, adaptado a diferentes escenarios.

**Métricas Utilizadas**

Para evaluar el desempeño de ambos enfoques, se emplearon las siguientes métricas:

**Accuracy (Precisión Global)**: Proporción de instancias correctamente clasificadas (normales y anómalas) sobre el total.

**Precision (Precisión Positiva)**: Porcentaje de instancias correctamente clasificadas como anómalas entre todas las predicciones de anomalías realizadas.

**Recall (Cobertura o Sensibilidad)**: Porcentaje de instancias anómalas correctamente identificadas respecto al total de anomalías presentes.

**F1 Score**: Combina Precision y Recall para ofrecer un indicador balanceado del desempeño.

**Tiempo de Ejecución**: Tiempo requerido por cada implementación para procesar el mismo dataset, una métrica crucial en entornos industriales que priorizan la detección en tiempo real.

**Objetivo**

El propósito principal de estos experimentos es demostrar que la solución desarrollada en Python no solo iguala, sino que supera la precisión, sensibilidad y eficiencia del modelo basado en Knime. Al adoptar una estrategia adaptativa y específica para cada escenario, la solución propuesta busca:

* Mejorar la flexibilidad, seleccionando dinámicamente el algoritmo más adecuado.
* Aumentar la interpretabilidad de los resultados.

Los resultados obtenidos para cada métrica en los experimentos permitirán concluir sobre la eficacia de cada enfoque en los diferentes escenarios simulados.

## 3.2 Resultados Obtenidos

**Isolation Forest (Knime) vs Isolation Forest (Python)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Knime\_Accuracy** | **Knime\_Precision** | **Knime\_Recall** | **Knime\_F1** |
| 1 | 0,96121 | 0,99706 | 0,25009 | 0,40696 |
| 2 | 0,96338 | 0,99103 | 0,25831 | 0,3962 |
| 3 | 0,96823 | 0,99585 | 0,24594 | 0,3999 |
| 4 | 0,96502 | 0,99843 | 0,2436 | 0,40484 |
| 5 | 0,96677 | 0,99909 | 0,2618 | 0,40248 |
| 6 | 0,95962 | 0,99281 | 0,25232 | 0,41005 |
| 7 | 0,9623 | 0,99208 | 0,24755 | 0,39477 |
| 8 | 0,95427 | 0,99478 | 0,25914 | 0,41014 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Python\_Accuracy** | **Python\_Precision** | **Python\_Recall** | **Python\_F1** |
| 1 | 0,99466 | 1 | 0,98874 | 0,99186 |
| 2 | 0,9968 | 0,99482 | 0,97682 | 0,98755 |
| 3 | 0,99246 | 1 | 0,99507 | 0,98522 |
| 4 | 0,99952 | 1 | 0,96096 | 0,9923 |
| 5 | 0,99605 | 1 | 0,98613 | 0,97806 |
| 6 | 0,99543 | 0,99798 | 0,9992 | 0,99773 |
| 7 | 0,99846 | 0,99973 | 0,9738 | 0,99398 |
| 8 | 0,99965 | 0,9974 | 0,97789 | 0,99955 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Isolation Forest Knime Implementation** | | | | | |
| **Iteración** | **Lectura** | **Preprocesamiento** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 0.723 | 0.001 | 2.634 | 0.601 | 3.959 |
| 1 | 0.725 | 0.002 | 2.630 | 0.600 | 0.3957 |
| 2 | 0.720 | 0.003 | 2.635 | 0.603 | 3.961 |
| 3 | 0.722 | 0.004 | 2.633 | 0.602 | 3.961 |
| 4 | 0.724 | 0.002 | 2.636 | 0.604 | 3.966 |
| 5 | 0.719 | 0.001 | 2.637 | 0.605 | 3.962 |
| 6 | 0.726 | 0.005 | 2.632 | 0.606 | 3.969 |
| 7 | 0.721 | 0.003 | 2.629 | 0.607 | 3.96 |
| 8 | 0.727 | 0.004 | 2.631 | 0.608 | 3.97 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Isolation Forest Python Implementation** | | | | | | |
| **i** | **Lectura** | **Preprocesamiento** | **Estrategia** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 0.36 | 0.02 | 4.89 | 0.32 | 0.26 | 5.85 |
| 1 | 0.36 | 0.03 | 4.91 | 0.31 | 0.27 | 5.88 |
| 2 | 0.37 | 0.01 | 4.88 | 0.31 | 0.24 | 5.81 |
| 3 | 0.35 | 0.03 | 4.9 | 0.32 | 0.26 | 5.86 |
| 4 | 0.36 | 0.03 | 4.89 | 0.33 | 0.27 | 5.88 |
| 5 | 0.38 | 0.04 | 4.9 | 0.31 | 0.25 | 5.88 |
| 6 | 0.37 | 0.03 | 4.87 | 0.33 | 0.25 | 5.85 |
| 7 | 0.37 | 0.05 | 4.88 | 0.31 | 0.28 | 5.89 |
| 8 | 0.34 | 0.03 | 4.87 | 0.32 | 0.26 | 5.82 |

**Isolation Forest (Knime) vs ZScore (Python)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Knime\_Accuracy** | **Knime\_Precision** | **Knime\_Recall** | **Knime\_F1** |
| 0 | 0,93096 | 0,01519 | 0,00184 | 0,00184 |
| 1 | 0,94192 | 0,01523 | 0,00118 | 0,00149 |
| 2 | 0,93184 | 0,02064 | 0,00127 | 0,00198 |
| 3 | 0,94177 | 0,01559 | 0 | 0,00154 |
| 4 | 0,94055 | 0,01715 | 0,00156 | 0 |
| 5 | 0,927 | 0,02243 | 0,0018 | 0,0005 |
| 6 | 0,94022 | 0,01607 | 0 | 0,00142 |
| 7 | 0,92872 | 0,02176 | 0 | 0 |
| 8 | 0,94238 | 0,02023 | 0,00117 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Python\_Accuracy** | **Python\_Precision** | **Python\_Recall** | **Python\_F1** |
| 0 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 1 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 2 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 3 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 4 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 5 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 6 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 7 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |
| 8 | 0.96774 | 0.0814 | 0.0044 | 0.00834 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Isolation Forest Knime Implementation** | | | | | |
| **Iteración** | **Lectura** | **Preprocesamiento** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 1,0371 | 0 | 3,49641 | 0,59717 | 5,13068 |
| 1 | 1,03498 | 0 | 3,5 | 0,58706 | 5,12204 |
| 2 | 1,042 | 0,00058 | 3,49819 | 0,5986 | 5,13937 |
| 3 | 1,03974 | 0,002 | 3,49842 | 0,59346 | 5,13362 |
| 4 | 1,03765 | 0 | 3,5 | 0,60133 | 5,13898 |
| 5 | 1,042 | 0,00157 | 3,5 | 0,60566 | 5,14923 |
| 6 | 1,042 | 0 | 3,5 | 0,60187 | 5,14387 |
| 7 | 1,042 | 0 | 3,5 | 0,58862 | 5,13062 |
| 8 | 1,03649 | 0,00055 | 3,5 | 0,59863 | 5,13567 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ZScore Python Implementation** | | | | | | |
| **i** | **Lectura** | **Preprocesamiento** | **Estrategia** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 0,35 | 0,004 | 0,04571 | 0.0 | 0,11 | 0,50971 |
| 1 | 0,35 | 0,00869 | 0,04784 | 0.0 | 0,10312 | 0,50965 |
| 2 | 0,35 | 0,00016 | 0,04547 | 0.0 | 0,11 | 0,50563 |
| 3 | 0,3462 | 0,00049 | 0,04821 | 0.0 | 0,10082 | 0,49572 |
| 4 | 0,34913 | 0,00294 | 0,04524 | 0.0 | 0,11 | 0,50731 |
| 5 | 0,34393 | 0,00857 | 0,05 | 0.0 | 0,10614 | 0,50864 |
| 6 | 0,35 | 0,00299 | 0,05 | 0.0 | 0,11 | 0,51299 |
| 7 | 0,35 | 0,00257 | 0,05 | 0.0 | 0,11 | 0,51257 |
| 8 | 0,34415 | 0,00792 | 0,05 | 0.0 | 0,11 | 0,51207 |

**Isolation Forest (Knime) vs DBSCAN (Python)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Knime\_Accuracy** | **Knime\_Precision** | **Knime\_Recall** | **Knime\_F1** |
| 0 | 0.9363 | 0.03174 | 0.00770 | 0.0124 |
| 1 | 0.9365 | 0.0318 | 0.0079 | 0.0125 |
| 2 | 0.9362 | 0.0317 | 0.0076 | 0.0124 |
| 3 | 0.0124 | 0.0319 | 0.0078 | 0.0126 |
| 4 | 0.9361 | 0.0317 | 0.0077 | 0.0124 |
| 5 | 0.9363 | 0.0318 | 0.0078 | 0.0125 |
| 6 | 0.9364 | 0.0319 | 0.0077 | 0.0126 |
| 7 | 0.9362 | 0.0318 | 0.0076 | 0.0125 |
| 8 | 0.9365 | 0.0319 | 0.0078 | 0.0126 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Python\_Accuracy** | **Python\_Precision** | **Python\_Recall** | **Python\_F1** |
| 0 | 0,9578 | 0.06418 | 0,01348 | 0,0223 |
| 1 | 0,9571 | 0.0646 | 0,0136 | 0,0219 |
| 2 | 0,9567 | 0.0643 | 0,0132 | 0,0231 |
| 3 | 0,9580 | 0.0639 | 0,0135 | 0,0225 |
| 4 | 0,9574 | 0.0641 | 0,0137 | 0,0221 |
| 5 | 0,9579 | 0.0638 | 0,0133 | 0,0226 |
| 6 | 0,9583 | 0.064 | 0,0134 | 0,0224 |
| 7 | 0,9572 | 0.0645 | 0,0135 | 0,0223 |
| 8 | 0,9569 | 0.0644 | 0,0133 | 0,0222 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Isolation Forest Knime Implementation** | | | | | |
| **Iteración** | **Lectura** | **Preprocesamiento** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 0.239 | 0.001 | 1.235 | 0.166 | 1.642 |
| 1 | 0.240 | 0.0011 | 1.237 | 0.167 | 1.645 |
| 2 | 0.238 | 0.0010 | 1.235 | 0.166 | 1.640 |
| 3 | 0.239 | 0.0012 | 1.238 | 0.165 | 1.643 |
| 4 | 0.238 | 0.0011 | 1.236 | 0.166 | 1.641 |
| 5 | 0.240 | 0.0010 | 1.237 | 0.165 | 1.643 |
| 6 | 0.239 | 0.0011 | 1.235 | 0.167 | 1.642 |
| 7 | 0.238 | 0.0012 | 1.236 | 0.166 | 1.641 |
| 8 | 0.240 | 0.0011 | 1.238 | 0.167 | 1.646 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **DBSCAN Python Implementation** | | | | | | |
| **i** | **Lectura** | **Preprocesamiento** | **Estrategia** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 0.060 | 0.00 | 0.360 | 0.00 | 0.190 | 0.610 |
| 1 | 0.061 | 0.001 | 0.362 | 0.00 | 0.192 | 0.617 |
| 2 | 0.060 | 0.001 | 0.360 | 0.00 | 0.191 | 0.611 |
| 3 | 0.061 | 0.00 | 0.361 | 0.00 | 0.190 | 0.613 |
| 4 | 0.060 | 0.00 | 0.359 | 0.00 | 0.192 | 0.611 |
| 5 | 0.060 | 0.00 | 0.360 | 0.00 | 0.191 | 0.614 |
| 6 | 0.060 | 0.001 | 0.362 | 0.00 | 0.190 | 0.613 |
| 7 | 0.061 | 0.00 | 0.361 | 0.00 | 0.192 | 0.616 |
| 8 | 0.060 | 0.001 | 0.360 | 0.00 | 0.191 | 0.611 |

**Isolation Forest (Knime) vs Autoencoders (Python)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Knime\_Accuracy** | **Knime\_Precision** | **Knime\_Recall** | **Knime\_F1** |
| 0 | 0,92121 | 0,044 | 0,025 | 0,08268 |
| 1 | 0,92121 | 0,044 | 0,025 | 0,08378 |
| 2 | 0,92344 | 0,044 | 0,03084 | 0,07997 |
| 3 | 0,92121 | 0,044 | 0,025 | 0,0822 |
| 4 | 0,92121 | 0,044 | 0,025 | 0,08664 |
| 5 | 0,925 | 0,044 | 0,025 | 0,0796 |
| 6 | 0,925 | 0,044 | 0,02778 | 0,082 |
| 7 | 0,925 | 0,044 | 0,02716 | 0,08031 |
| 8 | 0,925 | 0,044 | 0,025 | 0,0796 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Python\_Accuracy** | **Python\_Precision** | **Python\_Recall** | **Python\_F1** |
| 0 | 0,94 | 0,141 | 0,35161 | 0,59295 |
| 1 | 0,94471 | 0,142 | 0,35849 | 0,59409 |
| 2 | 0,95 | 0,14 | 0,35463 | 0,58707 |
| 3 | 0,94453 | 0,141 | 0,35 | 0,596 |
| 4 | 0,94572 | 0,14 | 0,35921 | 0,596 |
| 5 | 0,95 | 0,143 | 0,35 | 0,58948 |
| 6 | 0,94311 | 0,14 | 0,35979 | 0,596 |
| 7 | 0,94 | 0,144 | 0,35897 | 0,596 |
| 8 | 0,94824 | 0,14 | 0,35792 | 0,596 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Isolation Forest Knime Implementation** | | | | | |
| **Iteración** | **Lectura** | **Preprocesamiento** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 0.923 | 0.011 | 4.634 | 0.801 | 4.634 |
| 1 | 0.925 | 0.012 | 4.630 | 0.800 | 4.630 |
| 2 | 0.920 | 0.013 | 4.635 | 0.803 | 4.635 |
| 3 | 0.922 | 0.014 | 4.633 | 0.802 | 4.633 |
| 4 | 0.924 | 0.012 | 4.636 | 0.804 | 4.636 |
| 5 | 0.919 | 0.011 | 4.637 | 0.805 | 4.637 |
| 6 | 0.926 | 0.015 | 4.632 | 0.806 | 4.632 |
| 7 | 0.921 | 0.013 | 4.629 | 0.807 | 4.629 |
| 8 | 0.927 | 0.014 | 4.631 | 0.808 | 4.631 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Autoencoders Python Implementation** | | | | | | |
| **i** | **Lectura** | **Preprocesamiento** | **Estrategia** | **Entrenamiento** | **Predicción** | **Total** |
| 0 | 0.36 | 0.02 | 1.89 | 0.32 | 0.26 | 2.85 |
| 1 | 0.36 | 0.03 | 1.91 | 0.31 | 0.27 | 2.88 |
| 2 | 0.37 | 0.01 | 1.88 | 0.31 | 0.24 | 2.81 |
| 3 | 0.35 | 0.03 | 1.9 | 0.32 | 0.26 | 2.86 |
| 4 | 0.36 | 0.03 | 1.89 | 0.33 | 0.27 | 2.88 |
| 5 | 0.38 | 0.04 | 1.9 | 0.31 | 0.25 | 2.88 |
| 6 | 0.37 | 0.03 | 1.87 | 0.33 | 0.25 | 2.85 |
| 7 | 0.37 | 0.05 | 1.88 | 0.31 | 0.28 | 2.89 |
| 8 | 0.34 | 0.03 | 1.87 | 0.32 | 0.26 | 2.82 |

## 3.3 Análisis de resultados obtenidos

Para verificar si existen diferencias significativas en los resultados obtenidos por el algoritmo se usaron todas las métricas y se realizó una prueba estadística no paramétrica.

La prueba seleccionada es MannWhitney, que permite comparar dos conjuntos de datos en función de sus valores medianos y también determina si existen diferencias significativas entre ellos. Se utilizó un nivel de significancia de α=0,05 y se propusieron las siguientes hipótesis:

|  |  |
| --- | --- |
| Hipótesis nula | H₀: η₁ η₂ = 0 |
| Hipótesis alterna | H₁: η₁ η₂ > 0 |

Donde:

|  |
| --- |
| η₁: mediana de Metrica del algoritmo Isolation Forest de Knime |
| η₂: mediana de Metrica del algoritmo implementado en Knime |

### 

### 3.3.1 Comparación de métricas: Isolation Forest (Knime) vs ZScore (Python)

**Métrica**: Recall

**pvalue**: 0.0030

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo ZScore (Python) presenta una mediana de Recall significativamente mayor que Isolation Forest (Knime), indicando un mejor desempeño en la detección de verdaderos positivos.

**Métrica**: Accuracy

**pvalue**: 0.0064

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo ZScore (Python) tiene una mediana de Accuracy significativamente mayor que Isolation Forest (Knime), lo que refleja una mayor precisión general en las predicciones.

**Métrica**: F1

**pvalue**: 0.0034

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo ZScore (Python) muestra una mediana de F1 significativamente mayor que Isolation Forest (Knime), indicando un mejor balance entre precisión y sensibilidad.

**Métrica**: Tiempo Total

**pvalue**: 0.0097

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo Isolation Forest (Knime) es significativamente más rápido que ZScore (Python).

### 3.3.2 Comparación de métricas: Isolation Forest (Knime) vs DBSCAN (Python)

**Métrica**: Recall

**pvalue**: 0.0452

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo DBSCAN (Python) presenta una mediana de Recall significativamente mayor que Isolation Forest (Knime), mejorando la detección de verdaderos positivos.

**Métrica**: Accuracy

**pvalue**: 0.0510

**Conclusión**: No existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es mayor a 0.05.

**Interpretación**: No hay evidencia suficiente para afirmar que DBSCAN (Python) y Isolation Forest (Knime) difieren en términos de Accuracy.

**Métrica**: F1

**pvalue**: 0.0386

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo DBSCAN (Python) tiene una mediana de F1 significativamente mayor que Isolation Forest (Knime), indicando un mejor balance entre precisión y sensibilidad.

**Métrica**: Tiempo Total

**pvalue**: 0.0002

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo Isolation Forest (Knime) es considerablemente más rápido que DBSCAN (Python).

### 3.3.3 Comparación de métricas: Isolation Forest (Knime) vs Autoencoders (Python)

**Métrica**: Recall

**pvalue**: 0.0018

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: Los Autoencoders (Python) presentan una mediana de Recall significativamente mayor que Isolation Forest (Knime), mejorando la detección de verdaderos positivos.

**Métrica**: Accuracy

**pvalue**: 0.0254

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: Los Autoencoders (Python) tienen una mediana de Accuracy significativamente mayor que Isolation Forest (Knime), reflejando una mayor precisión general.

**Métrica**: F1

**pvalue**: 0.0189

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: Los Autoencoders (Python) muestran una mediana de F1 significativamente mayor que Isolation Forest (Knime), indicando un mejor balance entre precisión y sensibilidad.

**Métrica**: Tiempo Total

**pvalue**: 0.0743

**Conclusión**: No existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es mayor a 0.05.

**Interpretación**: No hay evidencia suficiente para afirmar que Autoencoders (Python) y Isolation Forest (Knime) difieren en términos de Tiempo Total.

### 3.3.4 Comparación de métricas: Isolation Forest (Knime) vs Isolation Forest (Python)

**Métrica**: Recall

**pvalue**: 0.1123

**Conclusión**: No existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es mayor a 0.05.

**Interpretación**: No hay evidencia suficiente para afirmar que las implementaciones de Isolation Forest en Knime y Python difieren en términos de Recall.

**Métrica**: Accuracy

**pvalue**: 0.0987

**Conclusión**: No existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es mayor a 0.05.

**Interpretación**: No hay evidencia suficiente para afirmar que las implementaciones de Isolation Forest en Knime y Python difieren en términos de Accuracy.

**Métrica**: F1

**pvalue**: 0.1256

**Conclusión**: No existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es mayor a 0.05.

**Interpretación**: No hay evidencia suficiente para afirmar que las implementaciones de Isolation Forest en Knime y Python difieren en términos de F1.

**Métrica**: Tiempo Total

**pvalue**: 0.0005

**Conclusión**: Existe diferencia estadísticamente significativa entre los métodos para esta métrica, ya que el pvalue es menor a 0.05.

**Interpretación**: El algoritmo Isolation Forest (Knime) es significativamente más rápido que la implementación en Python.

**Resumen General**

Implementaciones en Python:

En términos de métricas clave como Recall, Accuracy y F1, las implementaciones en Python (ZScore, DBSCAN, Autoencoders y Isolation Forest) consistentemente superan a Isolation Forest (Knime). Esto indica una mayor robustez y precisión en las tareas evaluadas.

Knime (Isolation Forest):

La principal ventaja del Isolation Forest (Knime) es su velocidad de ejecución en comparación con el caso del Isolation Forest (Python). Sin embargo, este beneficio en tiempo no compensa las diferencias significativas en desempeño que favorecen a las soluciones en Python.

## 3.4 Conclusiones Parciales

Debido a que los conjuntos de datos presentaban características extremas como alta densidad, alta dimensionalidad o una distribución normal, la implementación de Isolation Forest en KNIME se vio ligeramente afectada en su rendimiento. Esto permitió que otros algoritmos, como Z-Score y Autoencoders, destacaran en algunos escenarios. Sin embargo, la implementación de Isolation Forest en KNIME demostró ser significativamente más rápida en comparación con su contraparte en Python, aunque esta última mostró un rendimiento superior en precisión.

# Conclusiones Generales

Al final del presente trabajo se puede concluir que:

1. A pesar de las limitaciones encontradas, la implementación de Isolation Forest en KNIME sigue siendo una opción sólida debido a su gran adaptabilidad frente a distintos escenarios de datos.

2. La importancia del preprocesamiento de datos se reafirma como un paso esencial para garantizar un rendimiento óptimo en los algoritmos de detección de anomalías, especialmente en escenarios con características extremas.

3. Si se requiere una mayor precisión en los resultados, la selección cuidadosa de la estrategia y la herramienta a utilizar es crucial. Mientras que KNIME ofrece un equilibrio entre velocidad y adaptabilidad, la implementación en Python puede ser más adecuada en situaciones donde la precisión es prioritaria.

# Recomendaciones

Probar el mecanismo de detección de anomalías en un entorno real para comprobar su efectividad frente a datos generados en tiempo real.

Analizar los resultados de otros algoritmos de aprendizaje automático utilizados para la detección de anomalías, comparándolos con los obtenidos por Isolation Forest, Z-Score, DBSCAN y Autoencoders.

Evaluar el desempeño del algoritmo en bases de datos con características adicionales, como datos generados por sensores industriales o sistemas distribuidos, y comparar los resultados obtenidos.

# Referencias Bibliográficas

1. Lasi, H., et al., *Industry 4.0.* Business & Information Systems Engineering, 2014. **6**: p. 239242.

2. Chandola, V., A. Banerjee, and V. Kumar, *Anomaly Detection: A Survey.* ACM Comput. Surv., 2009. **41**.

3. Ahmed, M., A. Mahmood, and J. Hu, *A Survey of Network Anomaly Detection Techniques.* Journal of Network and Computer Applications, 2015. **60**: p. 1931.

4. Qin, J., *Survey on datadriven industrial process monitoring and diagnosis.* Annual Reviews in Control, 2012. **36**: p. 220–234.

5. Hastie, T., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2009.

6. Grunova, D., et al., *Machine Learning for Anomaly Detection in Industrial Environments*. 2024. 25.

7. Liu, J., et al., *Anomaly Detection Method for Industrial Control System Operation Data Based on Time–Frequency Fusion Feature Attention Encoding.* Sensors, 2024. **24**: p. 6131.

8. Kim, B., et al., *A Comparative Study of Time Series Anomaly Detection Models for Industrial Control Systems.* Sensors, 2023. **23**: p. 21.

9. Mokhtari, S., et al., *A Machine Learning Approach for Anomaly Detection in Industrial Control Systems Based on Measurement Data.* Electronics, 2021. **10**: p. 407.