UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
DIRECCIÓN DE POSGRADO

DIPLOMADO ESTADÍSTICA APLICADA A LA TOMA DE DECISIONES

TERCERA VERSIÓN

OPTIMIZACIÓN Y PREDICCIÓN DEL RENDIMIENTO EN LA GESTIÓN DE TAREAS EN UN PROYECTO DE DESARROLLO ÁGIL

PROYECTO PRESENTADO PARA OBTENER EL GRADO DE LICENCIATURA EN INGENIERÍA EN SISTEMAS
MODALIDAD DOBLE TITULACIÓN

POSTULANTE: ANAHI COSIMA CALLEJAS RAMOS
TUTOR: LIC. RAFAEL ROJAS ALTAMIRANO

Cochabamba - Bolivia
2024

OPTIMIZACIÓN Y PREDICCIÓN DEL RENDIMIENTO EN LA GESTIÓN DE TAREAS EN UN PROYECTO DE DESARROLLO ÁGIL

Por

Anahi Cosima Callejas Ramos

El presente documento, Trabajo de Grado es presentado a la Dirección de Posgrado de la Facultad de Ciencias y Tecnología en cumplimiento parcial de los requisitos para la obtención del grado académico de Licenciatura en Ingeniería en Sistemas, modalidad Doble Titulación, habiendo cursado el Diplomado "Estadística Aplicada a la Toma de Decisiones" propuesta por el Centro de Estadística Aplicada (CESA) en su tercera versión.

ASESOR/TUTOR

Lic. Rafael Rojas Altamirano

COMITÉ DE EVALUACIÓN

Ing. M. Sc. Ronald Edgar Patiño Tito. (Presidente)

Ing. M. Sc. Guillen Salvador Roxana. (Coordinador)

Ing. Por designar

Ing. Por designar

DIRECCIÓN DE POSGRADO
FACULTAD DE CIENCIAS Y TECNOLOGIA

DIRECCIÓN DE POSGRADO, FACULTAD DE CIENCIAS Y TECNOLOGÍA

Cochabamba, Bolivia

Aclaración

Este documento describe el trabajo realizado como parte del programa de estudios de Diplomado "Estadística Aplicada a la Toma de Decisiones" en el Centro de Estadística Aplicada CESA y la Dirección de Posgrado de la Facultad de Ciencias y Tecnología. Todos los puntos de vista y opiniones expresadas en el mismo son responsabilidad exclusiva del autor y no representan necesariamente las de la institución.

Resumen

Dedico el presente proyecto a mi familia, por estar siempre a mi lado, brindándome su amor, cuidado y apoyo incondicional, y por desear siempre lo mejor para mí. A mis compañeros, por inspirarme a crecer profesionalmente, por su colaboración y por ser parte fundamental de este camino.

Y, finalmente, a mí misma, por no rendirme ante los desafíos, por confiar en mis capacidades y por superar cada obstáculo con determinación. Este logro es el fruto de esfuerzo, sacrificio y dedicación constante.

Agradecimientos

A Dios por permitirme culminar este trabajo, cumplir con una de mis metas, bendecirme con el amor, cariño y apoyo de mi familia.

A la empresa en la que trabajo, por el apoyo durante la elaboración de este proyecto.

A la Universidad Mayor de San Simón por acogerme en sus predios y formarme como profesional.

Tabla de Contenidos

Lista de figuras

Lista de tablas

1. Introducción

En los últimos años, la industria del software ha experimentado un cambio hacia enfoques de desarrollo más ágiles, donde la adaptabilidad y la capacidad de respuesta son primordiales. Las metodologías ágiles, descritas en el "Manifesto for Agile Software Development", buscan fomentar la colaboración y la entrega continua de valor mediante ciclos de trabajo iterativos conocidos como sprints. Estas prácticas permiten a los equipos de desarrollo ajustar rápidamente sus estrategias, adaptándose a las necesidades cambiantes del cliente y a las demandas del mercado (Beck et al., 2001).

En este contexto, herramientas como JIRA se han convertido en aliados esenciales para los equipos de desarrollo, ya que permiten gestionar y priorizar incidencias, tareas o "issues" de manera visual y estructurada (Atlassian, 2023). A través de JIRA, los equipos pueden monitorizar el estado de cada tarea, identificar bloqueos, y evaluar el progreso de los sprints. Esto facilita la coordinación en proyectos complejos y mejora la transparencia del flujo de trabajo (Atlassian, 2023).

No obstante, la gestión efectiva de las incidencias en JIRA representa solo una parte del reto; el valor real se obtiene cuando los equipos pueden utilizar los datos generados a lo largo del proceso para realizar análisis predictivos. La minería de datos y el aprendizaje automático ofrecen herramientas poderosas para identificar patrones en el ciclo de vida de las incidencias y anticipar posibles retrasos, problemas de calidad o bloqueos en los flujos de trabajo (Han, Pei, & Kamber, 2011). De acuerdo con Han et al. (2011), los análisis predictivos aplicados a proyectos de desarrollo ágil pueden proporcionar métricas objetivas que permitan a los equipos mejorar su rendimiento y hacer predicciones precisas sobre el tiempo y esfuerzo requeridos para completar tareas futuras.

En este proyecto, se explorarán datos extraídos de la API de JIRA para analizar y predecir el rendimiento en la gestión de incidencias en un entorno de desarrollo ágil. Se utilizará un enfoque basado en machine learning para detectar patrones de comportamiento y optimizar el flujo de trabajo del equipo de desarrollo. Este análisis contribuirá a la toma de decisiones basada en datos, al proporcionar una base cuantitativa para anticipar los tiempos de resolución y reducir los riesgos asociados con los bloqueos y retrasos en el proyecto.

1.1. Antecedentes

La industria del software ha evolucionado de modelos de desarrollo lineales y secuenciales, como el modelo en cascada, a enfoques más flexibles y adaptativos, como las metodologías ágiles. Este cambio responde a la necesidad de crear productos de software que se ajusten rápidamente a las demandas cambiantes del mercado y a las expectativas de los usuarios (Sutherland y Schwaber, 2013). Las metodologías ágiles se caracterizan por ciclos iterativos, revisiones constantes y un enfoque en la colaboración, permitiendo a los equipos adaptarse y mejorar de forma continua (Beck et al., 2001).

En este contexto, herramientas de gestión de proyectos como JIRA han ganado popularidad, ofreciendo a los equipos de desarrollo una plataforma para organizar, priorizar y realizar un seguimiento efectivo de las incidencias o tareas. JIRA facilita la coordinación de equipos en proyectos complejos, permitiendo una visualización clara del flujo de trabajo a través de tableros de Kanban y sprints de Scrum (Atlassian, 2023). Esta herramienta no solo permite gestionar incidencias, sino también registrar los datos generados durante el proceso de desarrollo, datos que se pueden utilizar para hacer análisis y tomar decisiones informadas (Miller, 2020).

A medida que los datos disponibles en herramientas como JIRA aumentan, el análisis predictivo y el machine learning se vuelven esenciales para optimizar el rendimiento. Según Han, Pei y Kamber (2011), las técnicas de minería de datos permiten identificar patrones en los datos históricos que ayudan a predecir posibles bloqueos y el tiempo de resolución de incidencias, lo cual es fundamental para la mejora continua de los equipos de desarrollo. Esto contribuye a la eficiencia de los proyectos ágiles, ya que permite a los equipos anticiparse a posibles problemas y ajustar sus estrategias de manera proactiva.

En resumen, el uso de metodologías ágiles y herramientas como JIRA, combinado con técnicas de análisis de datos, representa un avance significativo en la gestión de proyectos de desarrollo de software. El presente proyecto pretende aprovechar estas herramientas para analizar y predecir el rendimiento en la gestión de incidencias, brindando una base para la toma de decisiones basada en datos.

1.2. Justificación

En un entorno de desarrollo ágil, la capacidad de los equipos para gestionar eficazmente las incidencias es crucial para el éxito del proyecto y la satisfacción del cliente. Las metodologías ágiles permiten a los equipos de desarrollo adaptarse rápidamente a los cambios en los requisitos y responder a los desafíos a medida que surgen. Sin embargo, la falta de un enfoque predictivo y analítico en la gestión de incidencias limita la capacidad de los equipos para optimizar su rendimiento de manera proactiva y anticiparse a posibles problemas que puedan afectar la eficiencia y los tiempos de entrega (Rigby, Sutherland, & Takeuchi, 2016).

El análisis predictivo en la gestión de incidencias permite identificar patrones históricos y prever problemas potenciales antes de que impacten negativamente en el flujo de trabajo. De acuerdo con Provost y Fawcett (2013), el uso de técnicas de machine learning en el análisis de datos empresariales proporciona un marco para realizar predicciones basadas en patrones de datos pasados, lo que permite a las organizaciones actuar con anticipación en lugar de reaccionar ante problemas inesperados. En el contexto de los equipos de desarrollo de software, esto significa que se pueden prever bloqueos, ajustar la carga de trabajo y optimizar la asignación de recursos, contribuyendo directamente a la eficiencia y efectividad del equipo (Provost & Fawcett, 2013).

Este proyecto es relevante para organizaciones que buscan aprovechar los datos generados por herramientas de gestión como JIRA para no solo supervisar sus operaciones en tiempo real, sino también utilizar estos datos para mejorar continuamente su rendimiento. El desarrollo de un modelo predictivo para la gestión de incidencias permitirá a los equipos de desarrollo adoptar un enfoque proactivo, optimizando sus procesos de resolución de incidencias y mejorando su capacidad de respuesta. Además, al anticipar los tiempos de resolución de incidencias y predecir posibles bloqueos, este proyecto proporcionará a los líderes de equipo y a los Project Managers información clave para la toma de decisiones y la planificación estratégica (Han, Pei, y Kamber, 2011).

Por lo tanto, este proyecto no solo contribuye al campo del desarrollo de software mediante el uso de técnicas de análisis predictivo, sino que también ofrece un valor práctico al mejorar la eficiencia de los equipos y reducir los tiempos de respuesta ante problemas comunes. Esto es especialmente importante en un contexto de alta competitividad, donde la capacidad de entrega rápida y eficiente representa una ventaja competitiva significativa (Meyer, 2014).

1.3. Planteamiento del problema

En los proyectos de desarrollo de software, la gestión eficiente de incidencias (issues o tareas) es un factor crítico para el éxito. A medida que los sistemas de software se vuelven más complejos, y las demandas de entrega rápida aumentan, los equipos de desarrollo enfrentan la presión de resolver de manera ágil una gran variedad de incidencias, desde problemas técnicos hasta nuevas funcionalidades. Sin embargo, debido a la naturaleza dinámica de estos proyectos, los equipos suelen encontrarse con bloqueos, cambios en las prioridades y demoras en la resolución de incidencias, lo cual afecta tanto la calidad del producto final como los tiempos de entrega (Meyer, 2014).

Las metodologías ágiles y herramientas como JIRA han surgido precisamente para abordar estos retos, proporcionando una plataforma que permite organizar y visualizar el flujo de trabajo. A través de la gestión visual y estructurada de incidencias, JIRA facilita la planificación en ciclos cortos (sprints) y permite la adaptabilidad ante los cambios constantes del proyecto (Rigby, Sutherland y Takeuchi, 2016). No obstante, la simple supervisión en tiempo real de las incidencias no es suficiente para prever los problemas que puedan impactar en el rendimiento futuro del equipo, especialmente cuando estos problemas se relacionan con bloqueos repetitivos o demoras en la resolución de tareas (Drury-Grogan, 2014).

A pesar de los avances en la gestión ágil, los equipos de desarrollo suelen subestimar o sobreestimar los tiempos necesarios para resolver incidencias, especialmente en tareas complejas. Esto genera acumulación de trabajo, retrasos en la entrega de funcionalidades críticas y conflictos en la asignación de recursos

El problema central es la falta de un enfoque basado en datos que permita a los equipos de desarrollo prever y resolver proactivamente los problemas recurrentes que afectan la eficiencia en la gestión de incidencias. El presente proyecto aborda esta necesidad mediante el desarrollo de un modelo predictivo para el rendimiento en la gestión de incidencias, que facilitará la identificación de patrones y la previsión de tiempos de resolución, permitiendo optimizar el flujo de trabajo y reducir los riesgos de bloqueos (Meyer, 2014).

1.4. Objetivo general

Desarrollar un modelo predictivo basado en aprendizaje automático que permita estimar con precisión los tiempos de resolución de incidencias, reduciendo el error absoluto medio (MAE) en al menos un 50% respecto a las estimaciones manuales.

1.4.1. Objetivos específicos

- Analizar y procesar datos históricos de JIRA para identificar las variables más relevantes en la predicción de tiempos de resolución..
- Implementar modelos de aprendizaje automático (Gradient Boosting, Random Forest y XGBoost) con el objetivo de comparar su rendimiento.
- Analizar el desempeño de los modelos de Machine Learning (Gradient Boosting, Random Forest y XGBoost) para seleccionar el más adecuado según las características de los datos.
- Diseñar un plan de implementación que permita integrar el modelo en herramientas como JIRA para su futuro despliegue.

2. Marco Teórico

2.1. Desarrollo Ágil en la Industria de Software

2.1.1. Principios y Valores del Desarrollo Ágil

El desarrollo ágil se basa en el **Manifesto for Agile Software Development** (Beck et al., 2001), creado para abordar la necesidad de un enfoque más flexible y adaptable en la creación de software. Este manifiesto establece cuatro valores clave:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación de contratos.
- Respuesta ante el cambio sobre el seguimiento de un plan.

Estos valores están diseñados para fomentar un enfoque de desarrollo que priorice la colaboración, la adaptabilidad y la entrega continua de valor, lo que permite a los equipos responder a los cambios y requisitos emergentes sin las limitaciones de métodos secuenciales y rígidos como el modelo en cascada.

2.1.2. Metodologías Ágiles Principales

Existen diversas metodologías ágiles que implementan estos valores, siendo **Scrum** y **Kanban** las más populares. Ambas metodologías tienen características únicas que se adaptan a distintos tipos de equipos y proyectos.

- **Scrum**: Es una metodología basada en sprints o iteraciones de tiempo fijo (generalmente de 1 a 4 semanas), durante las cuales los equipos de desarrollo deben completar un conjunto de tareas o incidencias previamente planificadas. Los roles en Scrum incluyen el **Scrum Master** (quien facilita el proceso), el **Product Owner** (responsable de priorizar el backlog) y el **equipo de desarrollo**. Al final de cada sprint, se realiza una revisión para evaluar los resultados y planificar mejoras en futuros sprints (Schwaber & Sutherland, 2013).

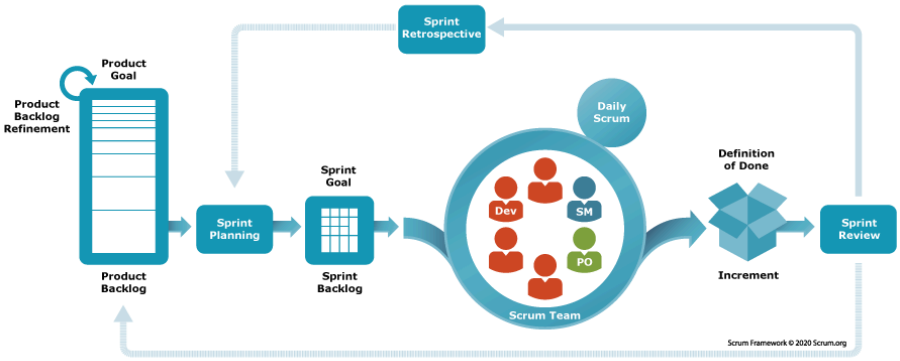

Figura 2-1: Flujo de trabajo de la metodología Scrum
Fuente: Scrum.org, 2023

- **Kanban**: Es una metodología visual que organiza las tareas en un tablero, donde cada tarea pasa por distintas etapas de progreso (por ejemplo, "Por hacer", "En progreso", "Completado"). Kanban permite un flujo de trabajo continuo, en el que las tareas se añaden y mueven en el tablero sin un

límite de tiempo fijo. Esto facilita una gestión flexible de las tareas y permite a los equipos adaptarse rápidamente a los cambios en las prioridades (Anderson, 2010).

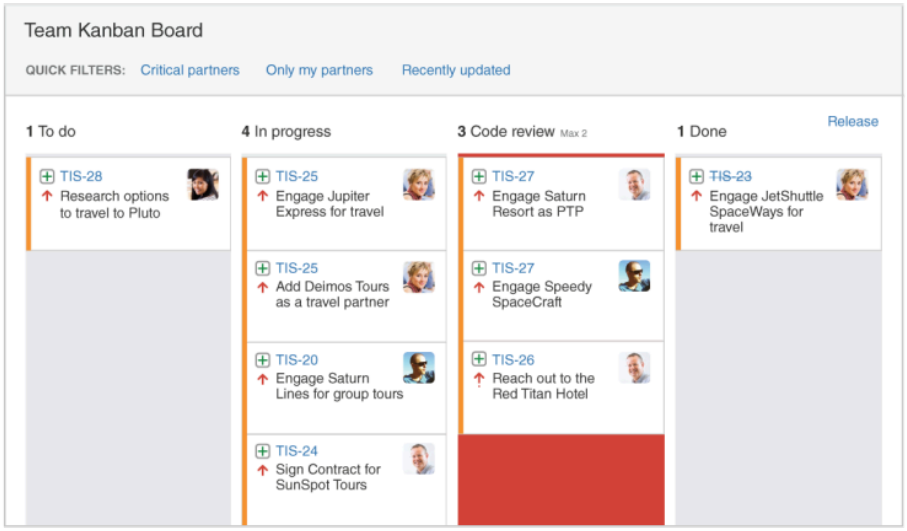

Figura 2-2: Flujo de trabajo de la metodología Kanban
Fuente: Atlassian, 2023

2.1.3. Importancia del Enfoque Iterativo y Adaptativo en Proyectos de Software

El enfoque iterativo y adaptativo del desarrollo ágil permite a los equipos mejorar su producto continuamente. Cada ciclo o sprint permite realizar ajustes en el desarrollo en función de la retroalimentación recibida, reduciendo el riesgo de errores acumulados y asegurando que el software evoluciona en línea con las necesidades del cliente (Rigby, Sutherland & Takeuchi, 2016).

Este enfoque incremental y de rápida entrega resulta en una mejora continua del producto, con entregas parciales que permiten a los equipos de desarrollo adaptarse a los cambios de manera ágil y eficiente.

2.2. Gestión de Incidencias en Proyectos Ágiles

2.2.1. Definición de Incidencias y su Ciclo de Vida

En el desarrollo de software, una **incidencia** o **issue** representa cualquier **tarea**, problema, o requerimiento que deba resolverse para avanzar en el proyecto. Las incidencias pueden clasificarse en distintos tipos, como:

- **Tareas**: Actividades específicas del desarrollo, como implementar una funcionalidad.
- **Bugs o errores**: Problemas técnicos que deben solucionarse para garantizar el funcionamiento correcto del sistema.
- **Mejoras**: Cambios propuestos para mejorar la calidad o la eficiencia del software.

El **ciclo de vida de una incidencia** comienza desde su creación hasta su resolución. En un entorno ágil, este ciclo suele involucrar las siguientes etapas:

- **Backlog**: La incidencia es identificada y priorizada para trabajarse en un sprint futuro.
- **En progreso**: La incidencia es asignada a un miembro del equipo y comienza a trabajarse.
- **Revisión**: La incidencia se somete a revisión, como pruebas de calidad o revisiones de código.
- **Completado**: La incidencia ha sido resuelta y cerrada.

Este ciclo de vida permite mantener la trazabilidad de cada tarea, facilitando la gestión y asignación de recursos en función de las prioridades y necesidades del proyecto (Rubin, 2012).

2.2.2. Tipos de Incidencias Comunes en Desarrollo de Software

Cada tipo de incidencia en un proyecto de software puede tener una naturaleza y nivel de complejidad distintos, lo que afecta directamente la planificación y el esfuerzo necesario para su resolución. En un contexto ágil, los tipos de incidencias más comunes incluyen:

- **Historias de usuario**: Describen funcionalidades desde la perspectiva del usuario final.
- **Tareas técnicas**: Incluyen actividades como la configuración de servidores o ajustes técnicos.
- **Errores o bugs**: Problemas que surgen durante el desarrollo o después de la implementación.
- **Spike**: Actividades de investigación necesarias para evaluar una tecnología o proceso antes de su implementación.

La correcta clasificación de cada incidencia permite al equipo priorizar de manera efectiva, asegurando que se aborden primero los problemas críticos o de alto impacto.

2.2.3. Herramientas de Gestión de Incidencias: JIRA

JIRA, de Atlassian, es una de las herramientas más utilizadas para la gestión de incidencias en equipos ágiles. Su popularidad se debe a su capacidad de adaptación a diferentes metodologías ágiles (Scrum, Kanban) y a su flexibilidad en la gestión visual de tareas. Algunas de las funciones clave de JIRA incluyen:

- **Tableros de trabajo** (Kanban o Scrum): Los equipos pueden organizar y visualizar sus tareas en diferentes estados ("Por hacer", "En progreso", "Completado"), permitiendo una visualización clara del flujo de trabajo.
- **Sprints y Backlogs**: En proyectos de Scrum, JIRA facilita la planificación de sprints, permitiendo a los equipos definir y gestionar los objetivos de cada iteración.
- **Trazabilidad y seguimiento**: JIRA permite registrar cada cambio en el estado y asignación de una incidencia, lo que facilita el análisis del ciclo de vida de las tareas y la identificación de posibles cuellos de botella (Atlassian, 2023).

2.2.4. Impacto de una Gestión Efectiva de Incidencias en el Rendimiento del Equipo

Una gestión eficaz de incidencias en entornos ágiles es crucial para el rendimiento y la productividad del equipo. Al mantener un flujo continuo y organizado de trabajo, los equipos pueden abordar las tareas con mayor eficiencia, minimizar los bloqueos y mantener un ritmo de entrega constante. Esto contribuye a una mayor satisfacción del cliente y facilita la mejora continua del proceso de desarrollo (Rasnacis y Berzisa, 2017).

2.3. Estimaciones y Capacidad de los Equipos de Desarrollo Ágil

2.3.1. Capacidad y Velocidad del Equipo

En el desarrollo ágil, la **capacidad** del equipo se refiere al volumen de trabajo que puede completar en un período determinado, generalmente un sprint, basándose en la disponibilidad de los miembros del equipo y

las horas de trabajo asignadas. Por otro lado, la **velocidad** se define como la cantidad de trabajo completado en sprints previos y se mide generalmente en puntos de historia. La velocidad de un equipo se utiliza como una medida predictiva para determinar cuántos puntos de historia puede manejar el equipo en futuros sprints (Cohn, 2006).

Para calcular la capacidad y la velocidad, los equipos suelen realizar un análisis retrospectivo de los sprints previos, evaluando cuántos puntos completaron en promedio. Esta información se utiliza para planificar la carga de trabajo en el sprint actual, intentando mantener un equilibrio entre la capacidad real del equipo y las expectativas del proyecto (Rubin, 2012).

2.3.2. Problemas Frecuentes en las Estimaciones de Esfuerzo y Tiempo

La estimación precisa de tiempo y esfuerzo es un desafío recurrente en los equipos de desarrollo ágil. Algunos problemas comunes incluyen:

- **Subestimación o sobreestimación de tareas**: Los equipos a menudo asignan menos o más puntos de historia de los necesarios a las incidencias, lo que puede resultar en sprints sobrecargados o con poca carga de trabajo.
- **Variabilidad en la complejidad de las tareas**: Algunas incidencias pueden parecer simples en la planificación, pero luego revelan ser más complejas de lo esperado, afectando la estimación original.
- **Falta de experiencia o conocimiento técnico**: La falta de familiaridad con el código o con la funcionalidad que se está desarrollando puede llevar a estimaciones inexactas.

Estos problemas afectan tanto la precisión de las estimaciones como la capacidad del equipo para completar el sprint planificado, lo que puede resultar en problemas adicionales de planificación y en la necesidad de ajustes continuos (Meyer, 2014).

2.3.4. Impacto de las Estimaciones Inexactas en la Planificación de Sprints y la Satisfacción del Equipo

Las estimaciones inexactas no solo afectan la eficiencia del equipo, sino que también tienen un impacto significativo en la satisfacción del equipo y en la calidad del producto entregado. Cuando los equipos subestiman la cantidad de trabajo necesario, se ven forzados a trabajar bajo presión para completar el sprint, lo que puede generar estrés y afectar la moral. Por otro lado, la sobreestimación puede llevar a un uso ineficiente del tiempo y los recursos, dejando al equipo sin tareas productivas para el resto del sprint (Rigby, Sutherland, y Takeuchi, 2016).

La precisión en las estimaciones es fundamental para asegurar que los equipos puedan trabajar de manera sostenida, minimizando la necesidad de ajustes de última hora en el backlog y evitando bloqueos innecesarios. Una planificación precisa permite que los Product Owners y Project Managers anticipen con mayor claridad los entregables del sprint, mejorando la comunicación con los stakeholders y optimizando la asignación de tareas.

2.3.5. Estrategias para Mejorar la Precisión en las Estimaciones

Existen diversas técnicas y estrategias que los equipos ágiles pueden implementar para mejorar la precisión en sus estimaciones:

- **Planning Poker**: Una técnica en la que los miembros del equipo asignan puntos de historia a cada incidencia de forma individual y luego discuten sus diferencias hasta alcanzar un consenso. Esta técnica fomenta una discusión más detallada y permite reducir la subjetividad en las estimaciones (Cohn, 2006).
- **División de tareas**: Dividir tareas complejas en subtareas más pequeñas y estimar cada una de manera individual permite al equipo asignar puntos de historia más precisos, en lugar de asignar una estimación amplía a tareas complejas.
- **Uso de la velocidad del equipo como guía**: La velocidad histórica del equipo es una referencia valiosa para ajustar las estimaciones de futuros sprints, basándose en el desempeño real y no en estimaciones subjetivas.

Estas estrategias pueden ayudar a los equipos a mejorar la precisión de sus estimaciones, optimizando así la planificación de sprints y asegurando una distribución más equilibrada del trabajo.

2.4. Análisis de Datos en la Gestión de Proyectos de Software

2.4.1. Importancia de la Analítica Descriptiva para Evaluar el Rendimiento en Equipos de Desarrollo

La **analítica descriptiva** es fundamental en la gestión de proyectos de software, ya que permite a los equipos evaluar su rendimiento actual y obtener una visión detallada de sus patrones de trabajo y flujos operativos. En entornos ágiles, donde el cambio constante y la flexibilidad son esenciales, la analítica descriptiva proporciona información clave sobre cómo los equipos están manejando sus tareas, permitiendo realizar ajustes informados y optimizar la toma de decisiones.

A través de la analítica descriptiva, los equipos pueden identificar áreas problemáticas, como cuellos de botella, tareas que consumen más tiempo del previsto o incidencias recurrentes que afectan el rendimiento del equipo. Además, esta analítica es una herramienta crucial para analizar y aprender del historial de los sprints previos, ayudando a los equipos a anticiparse a futuros problemas y mejorar la planificación de los siguientes sprints (Provost & Fawcett, 2013). Esto permite una visión objetiva de la eficiencia del equipo, facilitando el diseño de estrategias basadas en datos para mejorar el flujo de trabajo y la colaboración entre los miembros del equipo.

2.4.2. Principales Métricas de Rendimiento en la Gestión de Incidencias

Las métricas de rendimiento en proyectos de software sirven como indicadores clave para evaluar la productividad y eficiencia de los equipos. Estas métricas cuantifican aspectos críticos del trabajo del equipo, permitiendo monitorear su progreso en tiempo real y hacer ajustes conforme a los objetivos del sprint o del proyecto. Algunas de las métricas más comunes en la gestión de incidencias incluyen:

- **Tiempo de resolución**: Esta métrica mide el tiempo total necesario para completar una incidencia desde su asignación hasta su cierre. Un tiempo de resolución bajo indica un flujo de trabajo eficiente y menos bloqueos, mientras que tiempos altos pueden sugerir problemas en la asignación de recursos o en la planificación.
- **Tiempo de ciclo (cycle time)**: Similar al tiempo de resolución, el tiempo de ciclo mide el tiempo que transcurre desde que se inicia el trabajo en una incidencia hasta que se completa. Esta métrica es

particularmente útil para detectar retrasos en las etapas intermedias de una tarea y evaluar la consistencia en la finalización de las tareas.

- **Frecuencia de bloqueos**: La frecuencia con la que las incidencias enfrentan bloqueos es un indicador de posibles problemas en la planificación o en las dependencias dentro del proyecto. Un alto número de bloqueos puede sugerir la necesidad de mejorar la planificación o de revisar dependencias que afecten el progreso de las tareas.
- **Velocidad del equipo**: Representa el promedio de puntos de historia completados en cada sprint y es una métrica crucial para la planificación de futuros sprints. La velocidad es una medida clave para establecer expectativas realistas sobre el trabajo que el equipo puede manejar en un período determinado y permite ajustar la carga de trabajo conforme al rendimiento histórico del equipo.
- **Throughput**: El número total de incidencias completadas en un sprint. A diferencia de la velocidad, que se mide en puntos de historia, el throughput se mide en unidades de incidencias, proporcionando una visión del volumen total de trabajo completado. Esta métrica es útil para evaluar el rendimiento general del equipo y detectar tendencias en el flujo de trabajo (Cohn, 2006).

Al usar estas métricas, los equipos pueden establecer una línea base para su rendimiento y realizar un seguimiento de su progreso a lo largo del tiempo. Estas métricas también permiten comparar el rendimiento entre diferentes sprints y evaluar el impacto de los cambios en el proceso.

2.5. Fundamentos de Machine Learning y Modelos Predictivos

2.5.1. Machine Learning

Machine Learning (ML) es una rama de la inteligencia artificial que permite a los sistemas aprender de los datos históricos para hacer predicciones o tomar decisiones sin intervención humana explícita. Los modelos ML identifican patrones en los datos, permitiendo realizar análisis predictivos y ayudar a tomar decisiones basadas en evidencia. Esta capacidad de aprender de los datos es fundamental en entornos que requieren adaptabilidad y optimización continua (Han, Pei, & Kamber, 2011).

2.5.2. Aprendizaje Supervisado

En el aprendizaje supervisado, los modelos se entrenan con un conjunto de datos etiquetados, donde cada entrada tiene un valor de salida conocido, permitiendo que el modelo "aprenda" la relación entre los datos de entrada y el resultado esperado.

Dentro de este enfoque, existen dos tipos principales de problemas:

- **Regresión**: En los problemas de regresión, el objetivo es predecir un valor numérico continuo en función de las variables de entrada. Ejemplos comunes incluyen la predicción de precios, temperaturas, o tiempos de espera, donde el modelo calcula un valor cuantitativo.
- **Clasificación**: En los problemas de clasificación, el objetivo es asignar una etiqueta o clase a cada observación. Esto puede ser útil para problemas en los que se deben categorizar elementos, como clasificar correos electrónicos en "spam" o "no spam", o identificar si un cliente está en riesgo de cancelar un servicio (Géron, 2017).

2.5.3. Selección de Características y Preprocesamiento de Datos

Para que los modelos de ML sean efectivos, es fundamental elegir adecuadamente las **características** (features) o variables de entrada. La selección de características (Feature Engineering) implica identificar aquellas variables que tienen mayor relación con la variable de salida, lo cual mejora la precisión y eficiencia del modelo. Esto permite reducir el ruido en los datos y enfocarse en los factores más relevantes.

El **preprocesamiento de datos** es otro paso clave, que incluye la limpieza de datos, normalización y transformación de variables. Este proceso es esencial para evitar problemas como el overfitting y asegurar que el modelo funcione correctamente. El preprocesamiento puede incluir técnicas como el escalado de características, el manejo de valores faltantes y la transformación de variables categóricas en valores numéricos (Géron, 2017).

2.6. Modelos de Regresión para la Gestión de Incidencias

Los **modelos de regresión** se utilizan para problemas en los que el objetivo es predecir un valor numérico continuo. En el contexto de la gestión de incidencias, estos modelos permiten estimar el tiempo necesario para completar una tarea o resolver una incidencia, lo que ayuda en la planificación y asignación de recursos.

Algunos de los modelos de regresión más utilizados son:

2.6.1. Regresión Lineal

Es uno de los modelos de regresión más sencillos, donde se asume una relación lineal entre las variables de entrada y la variable de salida. Este modelo es interpretativo y ofrece una línea base rápida para problemas de predicción (James, Witten, Hastie, y Tibshirani, 2013).

2.6.2. Árboles de Decisión para Regresión

Este modelo divide los datos en diferentes grupos de acuerdo con las características de entrada y predice el valor medio de la variable de salida en cada grupo. Los árboles de decisión son efectivos para capturar relaciones no lineales y son fáciles de interpretar (Bishop, 2006).

2.6.3. Random Forest para Regresión

Este modelo es una extensión de los árboles de decisión, donde se crean múltiples árboles que se combinan para dar una predicción promedio. Es robusto y suele ofrecer un rendimiento superior a los árboles de decisión individuales (Géron, 2017).

2.6.4. Gradient Boosting Machines (GBM)

Los modelos de GBM combinan múltiples árboles de decisión de forma secuencial para corregir los errores de predicciones anteriores. Son especialmente útiles en problemas donde se necesita una predicción precisa y pueden captar patrones complejos en los datos (Géron, 2017).

Cada uno de estos modelos tiene sus propias ventajas y limitaciones. La selección del modelo depende del tipo de datos y de la precisión necesaria en las predicciones.

2.6.5. Ventajas y Desventajas de los Modelos

A continuación, se resumen algunas de las ventajas y desventajas de los modelos más utilizados en regresión y clasificación:

Modelo	Ventajas	Desventajas
Regresión Lineal	**Simplicidad e interpretabilidad,** muestra una relación lineal directa entre las variables de entrada y la salida. **Rendimiento en relaciones lineales,** funciona bien cuando la relación entre las variables de entrada y la variable de salida es aproximadamente lineal. **Eficiencia computacional,** es eficiente en términos de cálculo y no requiere gran capacidad de procesamiento.	**Limitaciones en relaciones no lineales,** lo que limita su aplicabilidad en datos complejos. **Sensibilidad a valores atípicos,** los valores atípicos pueden afectar significativamente los coeficientes del modelo, reduciendo su precisión. **Suposiciones restrictivas,** se asume independencia de errores, homocedasticidad y normalidad en la distribución de errores.
Árboles de decisión	**Interpretabilidad y visualización**, son intuitivos y permiten visualizar el proceso de toma de decisiones en una estructura de árbol, facilitando su comprensión y comunicación. **Flexibilidad con datos mixtos,** funcionan bien con variables categóricas y continuas. **Captura de interacciones no lineales**, pueden modelar relaciones complejas y no lineales entre variables.	**Propensos al overfitting**, tienden a sobre ajustarse a los datos de entrenamiento, lo que puede reducir su capacidad de generalización. **Inestabilidad**, pequeñas variaciones en los datos pueden llevar a cambios significativos en la estructura del árbol. **Baja precisión en comparación con modelos ensamblados**, su rendimiento puede ser inferior en comparación con técnicas ensambladas como el Random Forest o Gradient Boosting.
Random Forest	**Reducción del overfitting**, al combinar múltiples árboles de decisión, el Random Forest reduce el riesgo de overfitting y mejora la precisión. **Estabilidad y robustez**, debido a la construcción de múltiples árboles, es menos susceptible a cambios en los datos y ofrece predicciones más estables. **Manejo efectivo de datos faltantes y características irrelevantes**, puede manejar datos faltantes en cierta medida y es menos sensible a características irrelevantes.	**Menor interpretabilidad**, el conjunto de árboles en un Random Forest es más complejo y difícil de entender en su totalidad. **Requerimientos computacionales**, suelen requerir más tiempo y recursos computacionales para entrenar. **No siempre óptimo para problemas altamente no lineales**, puede no capturar relaciones extremadamente complejas como los modelos de boosting o redes neuronales.
Gradient Boosting y XGBoost	**Alta precisión**, suelen obtener excelentes resultados en precisión, especialmente en competencias de machine learning y problemas complejos.	**Mayor complejidad y dificultad de ajuste**, requieren un ajuste de hiperparámetros detallado y cuidadoso para maximizar su rendimiento.

	Flexibilidad en ajuste de hiperparámetros, ofrecen una variedad de hiperparámetros para ajustar el rendimiento del modelo. **Manejo de clases desbalanceadas**, pueden ajustarse para trabajar con clases desbalanceadas, utilizando estrategias para dar mayor peso a clases menos representadas.	**Sensibles a datos ruidosos y outliers**, debido a su método de entrenamiento secuencial, los modelos de boosting pueden sobre ajustarse a datos ruidosos si no se establecen controles adecuados. **Consumo de recursos computacionales**, tienden a ser más lentos en el entrenamiento y requieren más recursos computacionales.

Tabla 2-1: Ventajas y desventajas de los modelos de ML más comunes
Fuente: Elaboración Propia, 2024

Estos modelos son opciones sólidas para tareas de predicción en la gestión de incidencias, permitiendo a los equipos tomar decisiones basadas en datos y optimizar su rendimiento.

Para abordar el problema de estimación de tiempos de resolución, se seleccionaron tres modelos de aprendizaje automático basados en árboles de decisión:

- **Random Forest:** Utiliza múltiples árboles para promediar predicciones, reduciendo el riesgo de sobreajuste. Es ideal para datos ruidosos y problemas con relaciones no lineales.
- **Gradient Boosting:** Entrena iterativamente árboles que corrigen los errores de los árboles anteriores, logrando predicciones más precisas en problemas tabulares.
- **XGBoost:** Una versión optimizada de Gradient Boosting que se destaca por su eficiencia computacional y su capacidad para manejar datos incompletos.

2.7. Evaluación de Modelos

La evaluación de un modelo se realiza no sólo a través de métricas de rendimiento, sino también mediante técnicas de validación que aseguran que el modelo generalice bien a nuevos datos:

- **Validación cruzada (Cross-Validation)**: Consiste en dividir el conjunto de datos en múltiples subconjuntos, utilizando algunos para el entrenamiento y otros para la prueba, en iteraciones sucesivas. La **validación k-fold** es una técnica común, en la que los datos se dividen en "k" partes, y el modelo se entrena y evalúa "k" veces, cada vez utilizando un conjunto de validación diferente.
- **División de datos en entrenamiento, validación y prueba**: Los datos se dividen en tres partes: el conjunto de entrenamiento (para construir el modelo), el conjunto de validación (para ajustar hiperparámetros) y el conjunto de prueba (para evaluar la precisión final). Esto asegura que el modelo se evalúe en datos que no ha visto durante el entrenamiento, proporcionando una evaluación precisa de su rendimiento (Géron, 2017).

2.7.1. Métricas de Evaluación de Modelos de Regresión

Las métricas de evaluación permiten medir qué tan bien el modelo cumple su propósito en distintos contextos:

- **Error Medio Absoluto (MAE)**: Calcula el promedio de las diferencias absolutas entre las predicciones y los valores reales, proporcionando una medida de precisión fácil de interpretar.

Ecuación	Donde
$MAE = \frac{1}{n}\sum_{i=1}^{n} \lvert y_i - \widehat{y_i} \rvert$	• n es el número total de observaciones. • y_i es el valor real para la i-ésima observación. • $\widehat{y_i}$ es el valor predicho por el modelo para la i-ésima observación.

Tabla 2-2: Ecuación Error Medio Absoluto
Fuente: Elaboración Propia, 2024

- **Error Cuadrático Medio (MSE)** y **Raíz del Error Cuadrático Medio (RMSE)**: Miden el promedio de los errores elevados al cuadrado. Son útiles para penalizar errores grandes y reflejan la precisión del modelo en problemas de predicción de valores continuos.

Ecuación	Donde
$MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \widehat{y_i})^2$	• n es el número total de observaciones. • y_i es el valor real para la i-ésima observación. • $\widehat{y_i}$ es el valor predicho por el modelo para la i-ésima observación.

Tabla 2-3: Ecuación Error Cuadrático Medio
Fuente: Elaboración Propia, 2024

- **Coeficiente de Determinación (R^2)**: Indica la proporción de la variación de la variable de salida explicada por el modelo, proporcionando una evaluación de la calidad de la predicción.

Ecuación	Donde
$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \widehat{y_i})^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2}$	• n es el número total de observaciones. • y_i es el valor real para la i-ésima observación. • $\widehat{y_i}$ es el valor predicho por el modelo para la i-ésima observación. • $\overline{y}$ es el valor promedio de los valores reales y_i

Tabla 2-4: Ecuación Coeficiente de Determinación
Fuente: Elaboración Propia, 2024

2.8. Comparación y Selección de Modelos de Machine Learning

La selección del modelo de ML más adecuado es una etapa crucial en cualquier proyecto de análisis predictivo. Implica no solo evaluar el rendimiento del modelo en términos de precisión, sino también considerar factores como la interpretabilidad, la capacidad de generalización y el tiempo de procesamiento. Este proceso permite elegir el modelo que mejor se ajusta a los objetivos y restricciones del proyecto, optimizando su efectividad.

2.8.1. Métodos para la Comparación de Modelos

Existen varias estrategias para evaluar y comparar el rendimiento de distintos modelos, entre las que destacan:

- **Validación Cruzada (Cross-Validation)**: La validación cruzada es un método que divide el conjunto de datos en varios subconjuntos o "folds". En cada iteración, el modelo se entrena con todos los folds excepto uno, que se utiliza para la validación. La técnica más común es la **validación k-fold**, donde el conjunto de datos se divide en k partes, repitiendo el proceso k veces. Esto permite obtener una medida de rendimiento más estable y reduce la dependencia de un único conjunto de validación (James, Witten, Hastie, y Tibshirani, 2013).
- **Grid Search y Random Search para Optimización de Hiperparámetros**:
    - **Grid Search** explora un espacio predefinido de hiperparámetros, evaluando exhaustivamente todas las combinaciones para encontrar la configuración óptima. Este método es efectivo, aunque puede ser costoso computacionalmente.
    - **Random Search** selecciona combinaciones aleatorias de hiperparámetros dentro del espacio definido, lo que reduce el tiempo de cómputo y puede ser suficiente para obtener un buen rendimiento en problemas con un gran número de hiperparámetros (Bergstra & Bengio, 2012).
- **Evaluación en el Conjunto de Prueba**: Una vez seleccionado el modelo y optimizados sus hiperparámetros, se evalúa en el conjunto de prueba, un subconjunto de datos no utilizado en el entrenamiento. Esto proporciona una medida final de la capacidad de generalización del modelo y permite verificar su rendimiento en datos nuevos.

2.8.2. Criterios para la Selección del Modelo Óptimo

La selección del modelo no se basa únicamente en la precisión, sino en una variedad de criterios que aseguran que el modelo sea práctico y efectivo:

- **Precisión y Exactitud**: Las métricas de rendimiento como el MAE, RMSE, F1 score o AUC indican la precisión del modelo, ayudando a identificar qué tan cercano está el modelo a los resultados reales (Géron, 2017).
- **Capacidad de Generalización**: Un buen modelo debe generalizar bien a datos nuevos. El overfitting es un problema común en machine learning, donde el modelo se ajusta demasiado a los datos de entrenamiento, perdiendo efectividad en datos no vistos. La validación cruzada y el uso de un conjunto de pruebas ayudan a evaluar esta capacidad de generalización (James et al., 2013).
- **Interpretabilidad del Modelo**: En ciertos contextos, es crucial que el modelo no solo sea preciso, sino también fácil de entender. Los modelos simples, como la regresión lineal o los árboles de decisión, son interpretables y permiten entender cómo cada característica afecta la predicción. En cambio, los modelos más complejos, como los ensambles de árboles (e.g., Random Forest o Gradient Boosting), tienden a ser menos interpretables, aunque suelen ofrecer mayor precisión (Molnar, 2019).
- **Eficiencia Computacional**: Algunos modelos requieren más tiempo y recursos de procesamiento que otros. Modelos complejos como XGBoost pueden necesitar una mayor capacidad computacional y tiempo de entrenamiento, lo que puede ser un factor limitante en proyectos con restricciones de recursos o que requieren respuestas en tiempo real (Bishop, 2006).

2.8.3. Importancia de la Elección del Modelo Adecuado

Elegir el modelo adecuado es clave para el éxito del proyecto, ya que afecta directamente la utilidad de las predicciones. Un modelo mal ajustado o seleccionado sin considerar estos criterios puede llevar a predicciones poco fiables, decisiones erróneas y, en última instancia, a la pérdida de confianza en el análisis predictivo. Por otro lado, una buena elección de modelo permite a los equipos maximizar el valor de los datos y tomar decisiones informadas, optimizando recursos y mejorando los resultados.

2.9. Impacto del Análisis Predictivo en la Gestión Ágil de Proyectos

El análisis predictivo aplicado a la gestión ágil de proyectos permite optimizar la toma de decisiones al anticipar eventos, como bloqueos o desvíos en los tiempos de entrega, lo cual mejora la eficiencia y la adaptabilidad del equipo. La integración de técnicas predictivas en los proyectos ágiles representa una ventaja significativa, ya que permite planificar de manera proactiva y reaccionar rápidamente ante cambios en el flujo de trabajo.

2.9.1. Beneficios del Análisis Predictivo en la Planificación y Asignación de Tareas

- **Optimización de Recursos**: El análisis predictivo permite prever los recursos necesarios para completar cada tarea, facilitando una asignación óptima del equipo. Esto asegura que cada miembro del equipo esté asignado a tareas adecuadas según su capacidad, evitando sobrecarga o subutilización (Provost & Fawcett, 2013).
- **Mejoras en la Precisión de las Estimaciones**: La capacidad de predecir tiempos de resolución y posibles bloqueos reduce la dependencia de estimaciones subjetivas, mejorando la precisión en la planificación de sprints. Los modelos predictivos pueden ajustar las estimaciones de acuerdo con el desempeño histórico del equipo, proporcionando tiempos de entrega más realistas y adaptados a la capacidad real del equipo (James, Witten, Hastie, y Tibshirani, 2013).
- **Anticipación de Problemas Potenciales**: Los modelos de machine learning pueden identificar patrones que predicen problemas antes de que se presenten, como posibles bloqueos en tareas específicas o cuellos de botella en el flujo de trabajo. Esto permite que los equipos implementen medidas preventivas, asegurando un desarrollo fluido y evitando demoras inesperadas (Géron, 2017).
- **Toma de Decisiones Basada en Datos**: Al contar con un modelo predictivo preciso, los equipos pueden basar sus decisiones en datos y no en suposiciones. Esto mejora la capacidad de respuesta ante cambios y permite una planificación informada, basada en tendencias y patrones observados en datos históricos (Provost & Fawcett, 2013).

2.9.2. Aplicación de Insights Predictivos para Mejorar el Flujo de Trabajo

Los insights obtenidos a partir de un análisis predictivo ayudan a visualizar y comprender el comportamiento de las tareas y el desempeño del equipo. Algunos ejemplos de cómo los insights pueden optimizar el flujo de trabajo son:

- **Ajuste de la Capacidad del Equipo**: Basándose en predicciones del tiempo necesario para completar tareas, los equipos pueden ajustar su carga de trabajo para maximizar la productividad y evitar sprints sobrecargados o insuficientemente ocupados.

- **Priorización de Tareas Críticas**: Al predecir posibles bloqueos, los equipos pueden priorizar tareas que puedan afectar la finalización del sprint, asegurando que se resuelvan antes de que se conviertan en cuellos de botella (Bishop, 2006).
- **Evaluación y Ajuste Continuo**: La retroalimentación generada por los modelos predictivos permite ajustar las estrategias de desarrollo, identificando áreas de mejora en el flujo de trabajo y optimizando cada sprint en función de los resultados previos.

2.9.3. Retos y Consideraciones al Implementar Modelos Predictivos

La implementación de modelos predictivos en proyectos de desarrollo ágil no está exenta de desafíos. Entre las principales consideraciones se encuentran:

- **Calidad y volumen de datos**: Para que un modelo predictivo sea eficaz, es esencial contar con datos históricos de calidad y en cantidad suficiente. Datos incompletos o inconsistentes pueden limitar la precisión del modelo y afectar las predicciones (Provost & Fawcett, 2013).
- **Interpretabilidad del Modelo**: En entornos ágiles, es fundamental que los equipos puedan entender y confiar en las predicciones generadas por los modelos. Modelos muy complejos, como los de ensamble o redes neuronales, pueden ofrecer alta precisión, pero su falta de interpretabilidad puede dificultar su adopción (Molnar, 2019).
- **Adaptabilidad a Cambios**: Los proyectos ágiles están en constante cambio, lo que requiere que los modelos se adapten rápidamente a nuevos datos y patrones. La actualización y reentrenamiento continuo del modelo es crucial para que mantenga su relevancia en un entorno dinámico.
- **Costos Computacionales y de Mantenimiento**: Los modelos predictivos pueden requerir una cantidad significativa de recursos computacionales y monitoreo constante para asegurar su precisión y efectividad. Esto implica un costo adicional en términos de infraestructura y mantenimiento, lo cual debe considerarse en la planificación del proyecto (Géron, 2017).

2.9.4. Ventaja Competitiva en la Gestión de Proyectos

El uso de análisis predictivo en la gestión ágil de proyectos permite a los equipos de desarrollo alcanzar una mayor eficiencia y capacidad de respuesta, adaptándose proactivamente a las necesidades del proyecto. Al anticipar problemas y optimizar la asignación de recursos, los equipos pueden entregar productos de mayor calidad y cumplir con los tiempos de entrega de manera consistente. En un entorno de alta competitividad, la capacidad de integrar análisis predictivo en los procesos de desarrollo representa una ventaja estratégica significativa que facilita la mejora continua y la satisfacción del cliente.

3. Marco Metodológico

El enfoque metodológico se centra en el siguiente objetivo fundamental:

- **Predicción del Tiempo de Resolución de Incidencias**: Utilizando modelos de regresión, se busca predecir cuánto tiempo tomará resolver una incidencia, optimizando así la planificación de los sprints y mejorando la precisión en las estimaciones de los equipos.

El marco metodológico incluye la aplicación de técnicas avanzadas de machine learning, como el uso de **modelos de regresión** (Gradient Boosting Regressor, Random Forest Regressor, XGBoost Regressor) para predecir el tiempo de resolución. Estos modelos fueron seleccionados por su robustez y su capacidad para manejar datos complejos, optimizando tanto la precisión como la eficiencia del flujo de trabajo ágil.

Además, se aplicaron técnicas de feature engineering para transformar y enriquecer los datos, asegurando que los modelos tuvieran la información necesaria para realizar predicciones precisas. El proceso incluyó la limpieza de datos, la creación de nuevas características a partir de los datos existentes, y la optimización de los hiper parámetros de los modelos mediante validación cruzada.

En este apartado, también se realizó una comparación entre distintos modelos para cada uno de los objetivos. La evaluación se llevó a cabo utilizando métricas como **R^2, Mean Absolute Error (MAE)** para los modelos de regresión. Basándonos en los resultados de estas métricas, se seleccionó el mejor modelo para el objetivo, asegurando así la mayor precisión posible en las predicciones.

3.1. Área de Estudio

El presente proyecto se desarrolla en el contexto de una empresa multinacional dedicada a la gestión de proyectos de desarrollo de software, con operaciones tanto a nivel local como internacional. La empresa tiene su sede principal en la ciudad de **Cochabamba, Bolivia**, con oficinas adicionales en **Santa Cruz** y **La Paz**. Además, gracias a un modelo de trabajo remoto, la empresa colabora con profesionales ubicados en otros departamentos de Bolivia y en diversos países de América Latina.

La empresa se especializa en proporcionar personal altamente capacitado para empresas de software en **Europa** y **Norteamérica**, operando bajo un modelo de tercerización de equipos especializados. Esto le permite trabajar estrechamente con clientes internacionales que requieren soporte en el desarrollo y entrega de soluciones tecnológicas de manera ágil y eficiente.

Uno de los proyectos estratégicos en los que participa la empresa involucra varios equipos distribuidos en distintas ubicaciones. Para este análisis, nos enfocamos en los datos de los equipos compuestos por personal de la empresa, con algunas contribuciones de Project Managers y Product Owners que colaboran desde **Estonia** y **Estados Unidos**.

Los equipos de desarrollo están conformados por entre 3 a 7 personas, trabajando bajo una **metodología híbrida de Scrum y Kanban** para optimizar el flujo de trabajo de acuerdo con las necesidades específicas del proyecto.

Los datos utilizados para este análisis provienen principalmente del sistema **JIRA**, donde se gestionan las incidencias, tareas y sprints del proyecto. El enfoque del análisis es optimizar la **gestión de incidencias,** mejorando la **planificación de sprints** y la asignación de tareas mediante técnicas avanzadas de análisis de datos y machine learning.

El análisis se enfoca en:

- Incrementar la **eficiencia operativa** de los equipos mediante la optimización de los tiempos de resolución de incidencias.
- Identificar **cuellos de botella** en el flujo de trabajo para ajustar las estimaciones de esfuerzo y mejorar la asignación de recursos.
- Proporcionar **insights basados en datos** para la toma de decisiones, lo que contribuye a un mejor rendimiento del equipo y una mayor satisfacción del cliente.

3.2. Metodología Adoptada

El enfoque metodológico adoptado para este proyecto se centró en un análisis sistemático y estructurado que incluyó varias etapas clave, desde la recolección y limpieza de datos hasta el desarrollo y evaluación de modelos de Machine Learning. A continuación, se detallan los pasos seguidos:

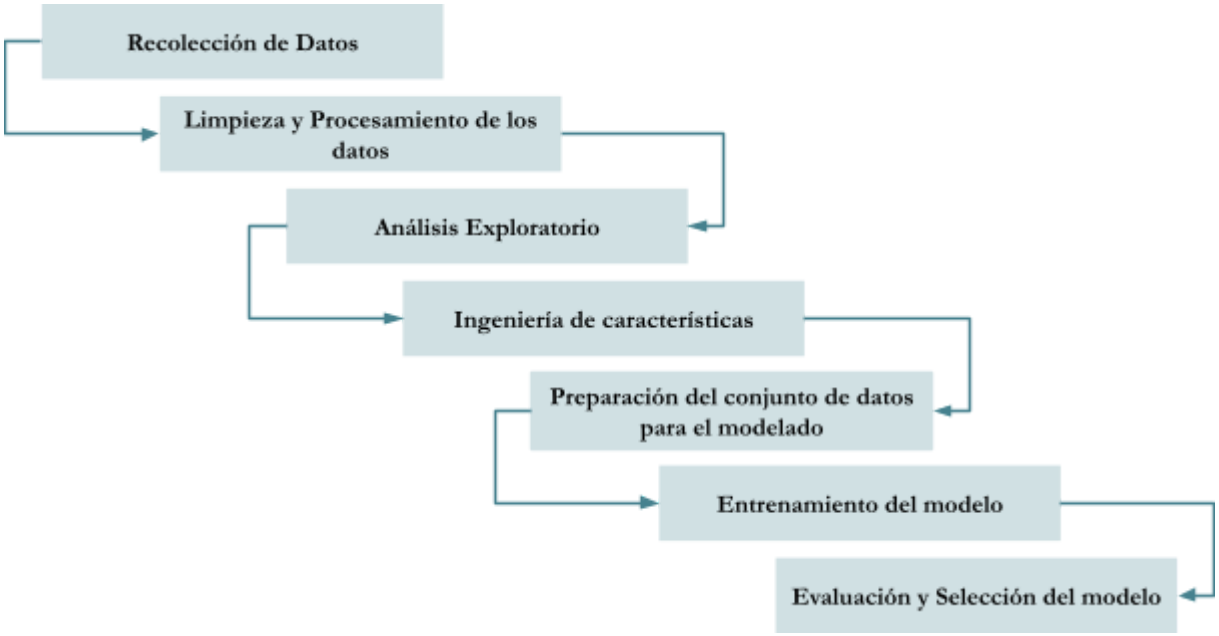

Figura 3-1: Flujograma Metodológico
Fuente: Elaboración Propia, 2024

3.2.1. Recolección de Datos

3.2.1.1. Origen de los Datos

Los datos utilizados en este proyecto provienen de **JIRA**, una plataforma ampliamente utilizada para la gestión de incidencias, seguimiento de tareas y planificación de sprints en entornos de desarrollo ágil. JIRA permite registrar información detallada sobre el ciclo de vida de cada incidencia, incluyendo su estado, prioridad, tipo, fechas importantes y asignaciones de equipo.

La recolección de datos se centró en los **equipos de desarrollo ubicados en Bolivia**, que colaboran en proyectos clave para un cliente internacional de la empresa.

3.2.1.2. Proceso de Recolección

Para obtener los datos, se utilizó un enfoque automatizado mediante **scripts en Javascript** que interactúan con la **API de JIRA**. El uso de esta API permite la extracción periódica y estructurada de datos directamente desde la plataforma, garantizando que la información recolectada sea precisa y actualizada.

El proceso de recolección se llevó a cabo de la siguiente manera:

- **Conexión a la API de JIRA**:
    - Autenticación mediante **tokens de acceso** para asegurar la protección y confidencialidad de los datos.
    - Uso de **endpoints específicos** para extraer datos relacionados con incidencias, sprints y transiciones de estado.
- **Extracción y Almacenamiento**:
    - Las respuestas de los endpoints, inicialmente en formato **JSON**, fueron convertidas y exportadas en formato **CSV** para facilitar su análisis.
- **Validación y Limpieza de Datos**:
    - Se llevó a cabo una **limpieza preliminar** para eliminar valores nulos y duplicados.
    - Conversión de campos de fecha al formato datetime para facilitar su manipulación y análisis.

3.2.1.3. Herramientas Utilizadas en el Proceso de Recolección

- **Javascript**: lenguaje de programación ligero y orientado a objetos, utilizado para la extracción de los datos.
- **Librerías**:
    - **axios**: biblioteca de JavaScript que permite hacer solicitudes HTTP de forma sencilla desde el navegador y Node.js, facilitando la integración con APIs. Se utiliza para interactuar con la API de JIRA.
    - **csv-writer**: biblioteca de Node.js que permite escribir datos en archivos CSV de forma fácil y rápida.
- **Entorno**:
    - **NodeJS**: entorno de ejecución de JavaScript en el lado del servidor
    - **VSCode**: editor de código fuente gratuito y multiplataforma desarrollado por Microsoft.

3.2.1.4. Descripción de los Datos Recopilados

El conjunto de datos incluye información tanto cualitativa como cuantitativa, recopilada durante un período de **seis meses** (Mayo a Octubre del presente año). Las principales categorías de datos extraídas son las siguientes:

- **Información General de Incidencias**:

Columna	Descripción
Key	Identificador único para cada incidencia.

Parent Key	Identificador del Epic al que la incidencia pertenece.
Summary	Título de la incidencia.
Issue Type	Tipo de incidencia (bug, tarea, historia de usuario, etc.).
Priority	Prioridad asignada a cada incidencia (Alta, Media, Baja, etc.).
Assignee	Persona responsable de la resolución de la incidencia.
Story Points	Puntos de esfuerzo asignados por el equipo responsable de la resolución de la incidencia.
Team	Equipo al que pertenece la persona responsable de la resolución de la incidencia
Sprint	Identificador o nombre del sprint en la que la incidencia fue trabajada.

Tabla 3-1: Información general de las incidencias
Fuente: Elaboración Propia, 2024

- **Información Temporal**:

Columna	Descripción
Created	Fecha de creación de la incidencia.
Updated	Fecha de la última actualización de la incidencia.
Sprint Start Date	Fecha en la que el sprint inició.
Sprint End Date	Fecha en la que el sprint debería completarse.
Sprint Complete Date	Fecha en la que el sprint se completó.

Tabla 3-2: Información temporal de las incidencias
Fuente: Elaboración Propia, 2024

- **Gestión de Estados**

La tabla a continuación muestra el flujo ideal de una incidencia desde "**To Do**" hasta "**Done**". Sin embargo, en el conjunto de datos real también se incluyen estados adicionales como Blocked, o transiciones desde "**In QA**" de vuelta a "**In Progress**", lo que indica que la incidencia no superó la etapa de control de calidad y necesita regresar al desarrollo. Aunque esta tabla no refleja todos los posibles estados, captura el propósito principal de las columnas incluidas.

Columna	Descripción
To Do - In progress	Fecha en la que la incidencia pasa de "To do" a "In Progress".
In progress - Code Review	Fecha en la que la incidencia pasa de "In Progress" a "Code Review".
Code Review - Sent to QA	Fecha en la que la incidencia pasa de "Code Review" a "Sent to QA".
Sent to QA - In QA	Fecha en la que la incidencia pasa de "Sent to QA" a "In QA ".

In QA - Final Review	Fecha en la que la incidencia pasa de "In QA " a "Final Review".
Final Review - Done	Fecha en la que la incidencia pasa de "Final Review" a "Done".
…	Todas las demás columnas representan el paso de las incidencias por todos los estados del board de JIRA.

Tabla 3-3: Información sobre la gestión de estados de las incidencias
Fuente: Elaboración Propia, 2024

3.2.1.5. Desafíos y Consideraciones

Durante el proceso de recolección, se presentaron algunos desafíos relacionados con:

- **Límites en la API de JIRA**: La API tiene restricciones en la cantidad de datos que se pueden extraer por cada solicitud, lo que requirió implementar un **paginado** en las peticiones.
- **Datos incompletos**: Algunas incidencias contenían datos faltantes, lo que hizo necesario realizar un preprocesamiento adicional para asegurar la integridad de los datos utilizados en el análisis.
- **Confidencialidad de la información:** Dado que los datos están relacionados con terceros y clientes, no es posible mostrar información de equipos fuera de Bolivia, en cumplimiento con las restricciones establecidas por acuerdos de confidencialidad.

3.2.2. Limpieza y Preprocesamiento de Datos

El proceso de preprocesamiento y limpieza de datos es fundamental para asegurar que el análisis posterior y la fase de modelado se lleven a cabo de manera efectiva.

El objetivo principal del preprocesamiento es **limpiar y transformar** el conjunto de datos extraído de JIRA para que estuviera en un formato adecuado para el análisis exploratorio y la creación de modelos predictivos.

A continuación, se detallan los pasos realizados:

3.2.2.1. Carga de Datos y Análisis Inicial

- Se cargaron los datos desde un archivo CSV utilizando la librería **pandas**. El conjunto de datos cuenta con 1429 filas y 200 columnas.

```
#libraries
import pandas as pd

file_path = './DATA/issues_history.csv'

# carga y vistazo del dataset
df = pd.read_csv(file_path, encoding='utf-8')
df.head()
```

Figura 3-2: Código de carga del conjunto de datos
Fuente: Elaboración Propia, 2024

- Se generó un resumen inicial de las columnas, tipos de datos y el porcentaje de **valores faltantes** para identificar posibles problemas en los datos.

```python
data_info = pd.DataFrame({
    'Data Type': df.dtypes,
    'Missing Values (%)': df.isnull().mean() * 100
}).sort_values(by='Missing Values (%)', ascending=False)

data_info
```

Figura 3-3: Código del resumen de los datos faltantes del conjunto de datos
Fuente: Elaboración Propia, 2024

	Data Type	Missing Values (%)
Reopen-Done	object	99.930021
Sent to QA-Blocked	object	99.930021
Final Review-Pending - Internal	object	99.930021
Refinement-Blocked/ On hold	object	99.930021
Icebox-Refinement	object	99.930021
...	...	...
Team	object	0.000000
Status Category Change Date	object	0.000000
Summary	object	0.000000
Priority	object	0.000000
Key	object	0.000000

Figura 3-4: Vista resultante del resumen del conjunto de datos
Fuente: Elaboración Propia, 2024

En la Figura 3-4 se observa que, en su mayoría, los datos están clasificados como tipo ***object***. Además, se identifica la presencia de múltiples columnas que contienen valores faltantes, lo que podría requerir un tratamiento adicional durante el preprocesamiento de los datos.

3.2.2.2. Filtrado de Equipos Relevantes

Dado que el enfoque del análisis está en los equipos de Bolivia, se filtraron las filas para incluir solo los equipos relevantes (Figura 3-5): **"Team Woof", "Team Roar", "Team Buzz", "Team Meow", "Team Quack", "Team Aether", "Team Howl", "Team Panda".** El conjunto de datos se redujo de 1429 filas a 1056 hasta este punto.

```python
# Manteniendo solo los equipos de Bolivia
teams_to_keep = ['team woof', 'team roar', 'team buzz', 'team meow',
                 'team quack', 'team aether', 'team howl', 'team panda']

# Filtrar filas para que solo queden los equipos especificados
df_filtered_teams = df[df['Team'].str.lower().isin(teams_to_keep)]
df_filtered_teams.shape
```

Figura 3-5: Código que filtra y conserva únicamente los datos relacionados con los equipos de Bolivia
Fuente: Elaboración Propia, 2024

3.2.2.3. Eliminación de Columnas con Valores Faltantes

Se eliminaron las columnas que tenían el 100% de valores faltantes (Figura 3-6), ya que no aportan información útil al análisis. El número de columnas se redujo de 200 a 191.

```
# Eliminar columnas que tienen el 100% de valores faltantes
df_cleaned = df_filtered_teams.dropna(axis=1, how='all')
df_cleaned.shape

(1056, 191)
```

Figura 3-6: Código que elimina columnas con 100% de valores faltantes
Fuente: Elaboración Propia, 2024

3.2.2.4. Filtrado de Columnas por Transiciones Válidas

Se definió una lista de **estados válidos** (por ejemplo, "To Do", "In Progress", "Done", etc.) y se filtraron las columnas basadas en estas transiciones. El número de columnas se redujo de 191 a 89.

```
# Definicion de las etapas por las que pasa una incidencia
no_stages = [col for col in df_cleaned.columns if '-' not in col]
stages = ['To Do', 'In Progress', 'Code Review', 'Final Review', 'In QA', 'Done', 'Sent to QA', 'Blocked/ On hold', 'Duplicate/ Rejected/ Deferred', ]

def is_valid_column(column):
    parts = column.split('-')
    return len(parts) == 2 and all(stage.strip() in stages for stage in parts)

# Filtrar las columnas según la nueva lógica
valid_columns = [col for col in df_cleaned.columns if is_valid_column(col)]
final_columns = no_stages + valid_columns

df_with_valid_columns = df_cleaned[final_columns]
df_with_valid_columns.head()
```

Figura 3-7: Código que filtra los estados válidos de los inválidos
Fuente: Elaboración Propia, 2024

3.2.2.5. Filtrado de Incidencias Completadas

- Se filtraron las filas para incluir únicamente las incidencias cuyo "Status" sea "Done" y cuyo "Sprint State" sea "Closed", debido a que el objetivo del análisis es explorar únicamente incidencias completadas dentro de sprints finalizados durante el periodo de tiempo establecido (6 meses).

```
# Filtrar filas donde "Status" sea "Done" y "Sprint State" sea "Closed"
filtered_data = data[(data['Status'] == 'Done') & (data['Sprint State'] == 'closed')]
```

Figura 3-8: Código que filtra las incidencias y sprints completados
Fuente: Elaboración Propia, 2024

- Se eliminaron columnas redundantes como "Sprint State", "Status", "Epic", "Updated" y "Parent Status", ya que las tres primeras tienen un valor único en todos los registros, mientras que las últimas no es relevante para el análisis.

```
# Eliminar columnas que tienen un unico valor
filtered_data = filtered_data.drop(columns=['Sprint State', 'Status', 'Epic', 'Parent Status', 'Updated'], errors='ignore')
```

Figura 3-9: Código que elimina columnas irrelevantes
Fuente: Elaboración Propia, 2024

3.2.2.6. Asignación de Valores de Story Points

- Para las tareas de tipo "Sub-task", se asignó un valor de 1 a la columna "Story Points" debido a que no suelen ser estimadas, pero si toman cierto tiempo en completarlas.

```
# asignar el valor 1 a los subtask debido a que estos no se estiman
filtered_data.loc[filtered_data['Issue Type'] == 'Sub-task', 'Story Points'] = 1
```

Figura 3-10: Código que asigna un valor a los puntos de historia de las sub tareas
Fuente: Elaboración Propia, 2024

- Se filtraron las incidencias para incluir solo tipos de incidencias importantes como "Bug", "Story", "Task", y "Sub-task". El conjunto de datos se redujo a 793 filas y 89 columnas.

```
# filtrar unicamente los tipos de tickets importantes
filtered_data = filtered_data[filtered_data['Issue Type'].isin(['Bug', 'Story', 'Task', 'Sub-task'])]
```

Figura 3-11: Código que filtra las incidencias más importantes
Fuente: Elaboración Propia, 2024

3.2.2.7. Conversión y Asignación de Tipos de Datos

Se aseguraron los tipos de datos correctos para las columnas críticas:

- "Story Points" se convirtió a tipo **numérico**.
- "Priority", "Issue Type", "Sprint", y "Team" se convirtieron en variables **categóricas**.
- Las columnas relacionadas con fechas ("Sprint Start Date", "Sprint End Date", "Created" y columnas de transición) se transforman al formato **datetime**.

```
# asignacion de tipos a las columnas
filtered_data['Story Points'] = pd.to_numeric(filtered_data['Story Points'], errors='coerce')
filtered_data['Priority'] = filtered_data['Priority'].astype('category')
filtered_data['Issue Type'] = filtered_data['Issue Type'].astype('category')
filtered_data['Sprint'] = filtered_data['Sprint'].astype('category')
filtered_data['Team'] = filtered_data['Team'].astype('category')

# Convertir las columnas de fechas a tipo datetime
date_columns = ['Sprint Start Date', 'Sprint End Date', 'Sprint Complete Date', 'Created']
for col in date_columns:
    filtered_data[col] = pd.to_datetime(filtered_data[col], errors='coerce', utc=True)

# Convertir las columnas de transición a tipo datetime
transition_columns = [col for col in filtered_data.columns if '-' in col]
for col in transition_columns:
    filtered_data[col] = pd.to_datetime(filtered_data[col], errors='coerce', utc=True
```

Figura 3-12: Código que asigna los correspondientes tipos a las columnas del conjunto de datos
Fuente: Elaboración Propia, 2024

3.2.2.8. Transformación de Transiciones

- Se transformaron las columnas de **transición de estado** a filas utilizando la función melt de pandas, manteniendo las columnas relevantes como "Key", "Priority", y "Issue Type".
- Se dividió la columna "State Transition" en "Initial State" y "Next State", lo que permitió un análisis más detallado del flujo de trabajo de cada incidencia.
- Los datos se ordenaron por "Key" y "Transition Date" para preservar la secuencia cronológica de las transiciones.
- Tras completar el proceso de limpieza y el pivote el conjunto de datos ahora contiene 2675 filas y 15 columnas.

```python
# Crear una tabla pivotada que incluya todas las columnas originales menos las columnas de transiciones
# Identificar todas las columnas que no son transiciones
non_transition_columns = [col for col in df.columns if '-' not in col or 'Date' in col]

# Transformar las columnas de transición en filas con 'melt', manteniendo las demás columnas
df_melted_full = pd.melt(df, id_vars=non_transition_columns, value_vars=transition_columns,
                         var_name='State Transition', value_name='Transition Date')

# Eliminar filas donde la fecha de transición sea nula
df_melted_full = df_melted_full.dropna(subset=['Transition Date'])

# Dividir la columna 'State Transition' en 'Initial State' y 'Next State'
df_melted_full[['Initial State', 'Next State']] = df_melted_full['State Transition'].str.split('-', expand=True)

# Ordenar la tabla por 'Key' y 'Transition Date'
df_melted_full = df_melted_full.sort_values(by=['Key', 'Transition Date'])

# Mostrar las primeras filas para confirmación
df_melted_full
```

Figura 3-13: Código que transforma las columnas de transición a filas
Fuente: Elaboración Propia, 2024

3.2.2.9. Resultados del Proceso de Limpieza

- El proceso de limpieza y filtrado resultó en un conjunto de datos que está listo para el análisis exploratorio.
- La limpieza redujo significativamente el ruido en los datos, permitiendo enfocarse en los equipos y tareas más relevantes.

3.2.3. Análisis Exploratorio

El Análisis Exploratorio de Datos (EDA) se llevó a cabo con el objetivo de comprender mejor el comportamiento y la distribución de las variables en el conjunto de datos, así como para identificar posibles patrones, relaciones, y cuellos de botella en la gestión de incidencias. A continuación, se detallan los principales hallazgos obtenidos a partir de este análisis.

3.2.3.1. Análisis de Tipos de Incidencias

El análisis de la columna "Issue Type" se realizó para entender qué tipos de incidencias predominan en el conjunto de datos.

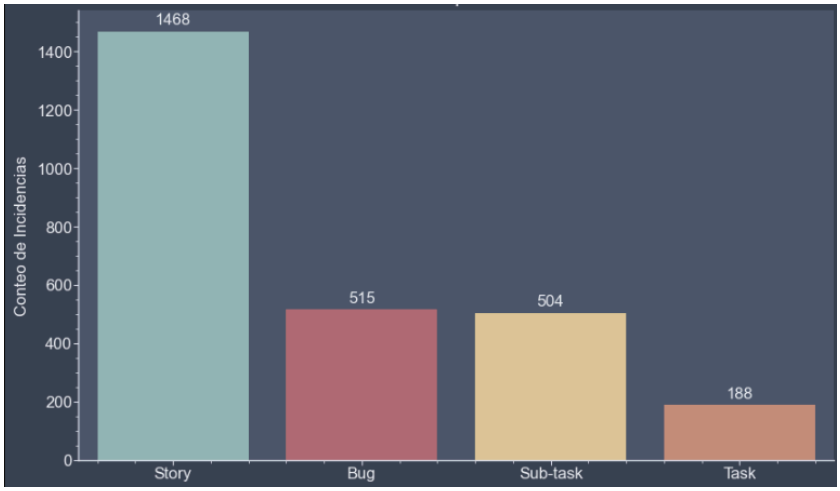

Figura 3-14: Distribución de los tipo de incidencias
Fuente: Elaboración Propia, 2024

La figura 3-14 nos muestra un predominio de incidencias de tipo "Story" (1,468), lo que indica un enfoque en desarrollar nuevas funcionalidades y entregar valor al cliente. A su vez, el número significativo de "Bugs" (515) y "Sub-tasks" (504) sugiere atención al mantenimiento y descomposición de tareas para facilitar su ejecución. En contraste, las "Tasks" generales son menos frecuentes (188), lo que sugiere que tienen menor prioridad en este contexto.

Recomendaciones

Priorizar la resolución de bugs para mejorar la calidad del producto, dado el volumen significativo de incidencias de este tipo. Este análisis ayuda a entender dónde se enfoca el esfuerzo del equipo y puede guiar la planificación para mejorar la eficiencia en los próximos sprints.

3.2.3.2. Análisis de Story Points

La columna "Story Points" fue analizada para evaluar cómo se distribuyen las estimaciones de esfuerzo asignadas a las tareas.

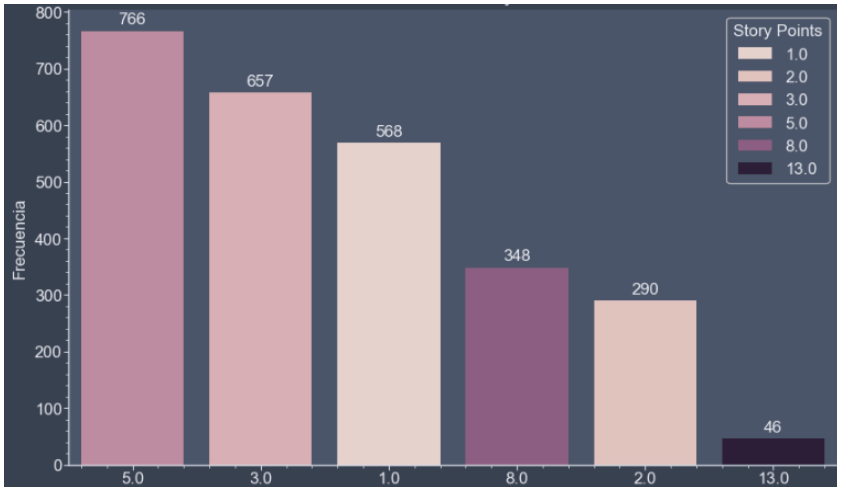

Figura 3-15: Distribución de las estimaciones de las incidencias
Fuente: Elaboración Propia, 2024

La Figura 3-15 muestra que predominan los Story Points medianos, con la mayoría de tareas asignadas a 5 Story Points (766 incidencias), seguidas de 3 Story Points (657) y 1 Story Point (568), lo que indica que la mayoría de las tareas tienen una complejidad media-baja. También se observan tareas de mayor tamaño con 8 Story Points (348) y 2 Story Points (290), mientras que sólo unas pocas alcanzan 13 Story Points (46). Esto sugiere un enfoque en dividir el trabajo en tareas más pequeñas y manejables para reducir riesgos, típico en metodologías ágiles. Las tareas más grandes podrían beneficiarse de un análisis adicional para su posible descomposición y facilitar su ejecución.

Recomendaciones

Considerar la revisión y optimización de tareas grandes para ver si pueden dividirse en subtareas más pequeñas, lo que podría mejorar la predictibilidad y la eficiencia en los sprints. Evaluar si las estimaciones de 5 y 3 Story Points son precisas, ya que representan la mayoría de las tareas; una mejora en la precisión podría optimizar la planificación de sprints.

3.2.3.3. Análisis de Transiciones entre Estados

Se analizó la frecuencia de transiciones entre diferentes estados ("To Do", "In Progress", "Done", etc.) para entender cómo fluyen las tareas a través del ciclo de vida de desarrollo.

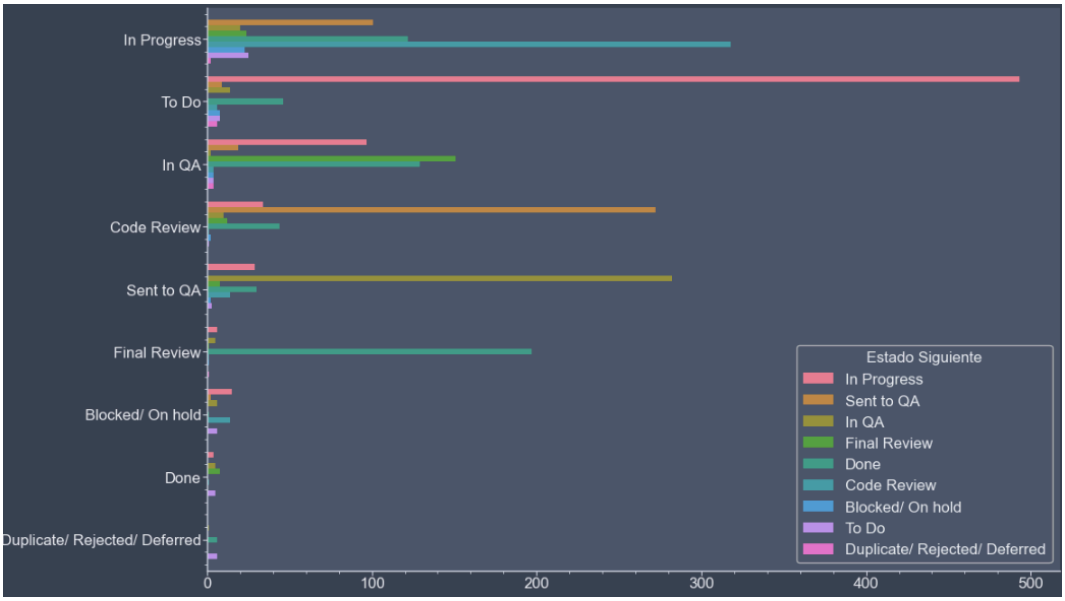

Figura 3-16: Transiciones de las incidencias de un estado a otro
Fuente: Elaboración Propia, 2024

La Figura 3-16 muestra que, en general, las transiciones entre estados siguen un flujo ideal desde "To Do" hasta "Done", lo que sugiere una gestión eficiente de la mayoría de las incidencias. No obstante, se identifican ciertos patrones que podrían reflejar posibles retrasos o ineficiencias. Por ejemplo, algunas incidencias regresan de "Sent to QA" a "In Progress", lo que indica que no pasaron la revisión de calidad en el primer intento, requiriendo ajustes adicionales. También se observan transiciones de "In Progress" a "Blocked", señalando posibles cuellos de botella, y casos donde incidencias marcadas como "Done" vuelven a "In Progress", lo que sugiere que hubo que reabrir tareas completadas debido a problemas detectados posteriormente. Además, ciertas incidencias avanzan directamente de "To Do" a "Done", omitiendo etapas intermedias. Esto podría reflejar estimaciones inexactas, desarrollo apresurado sin revisión adecuada, duplicidad en historias de usuario o la irreproducibilidad en el caso de bugs.

Recomendaciones

Sería útil revisar los casos en los que las incidencias pasan de estados finales a estados intermedios (como de "Done" a "In Progress") para entender mejor qué provoca estos retrocesos.

Analizar más a fondo las transiciones directas de "To Do" a "Done" podría ayudar a identificar gaps en la planificación y estimación de tareas, permitiendo optimizar la precisión en la asignación de esfuerzos.

Estos hallazgos indican áreas donde el equipo puede mejorar la precisión en las estimaciones y optimizar el flujo de trabajo para minimizar los retrasos y bloqueos, asegurando que las tareas se gestionen de manera más eficiente.

3.2.3.4. Análisis de la productividad y eficiencia de los equipos

Este análisis permite evaluar de manera integral tanto la capacidad como la velocidad de trabajo de los equipos. Esto ayuda a optimizar el flujo de trabajo, asegurando una asignación eficiente de recursos y mejorando la **eficacia** en la entrega de proyectos.

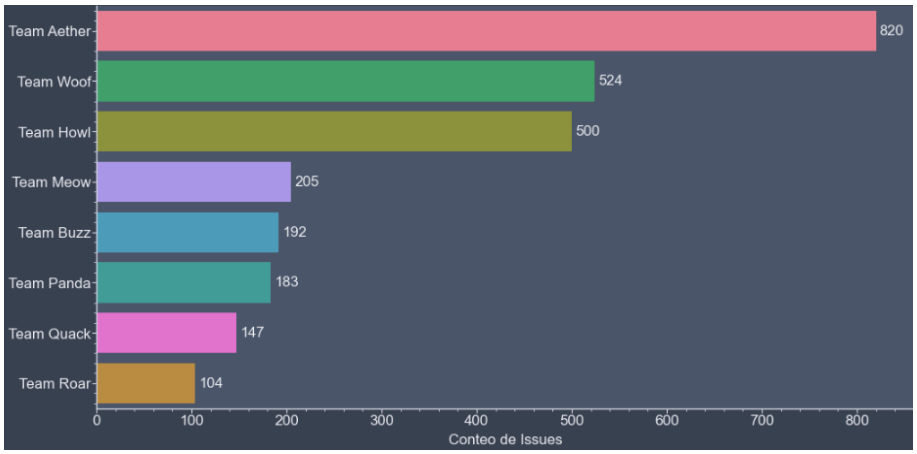

Figura 3-17: Cantidad de incidencias completadas por equipo
Fuente: Elaboración Propia, 2024

La figura 3-17 muestra el número total de incidencias completadas por cada equipo.

- "Team Aether" tiene la mayor cantidad de incidencias completadas (820), lo que indica **alta productividad** en comparación con los demás equipos.
- "Team Woof" y "Team Howl" también tienen un desempeño significativo con 524 y 500 incidencias completadas, respectivamente.
- Equipos como "Team Roar" y "Team Quack" completaron menos incidencias, lo que podría sugerir que están trabajando en tareas más complejas o con menor carga de trabajo.

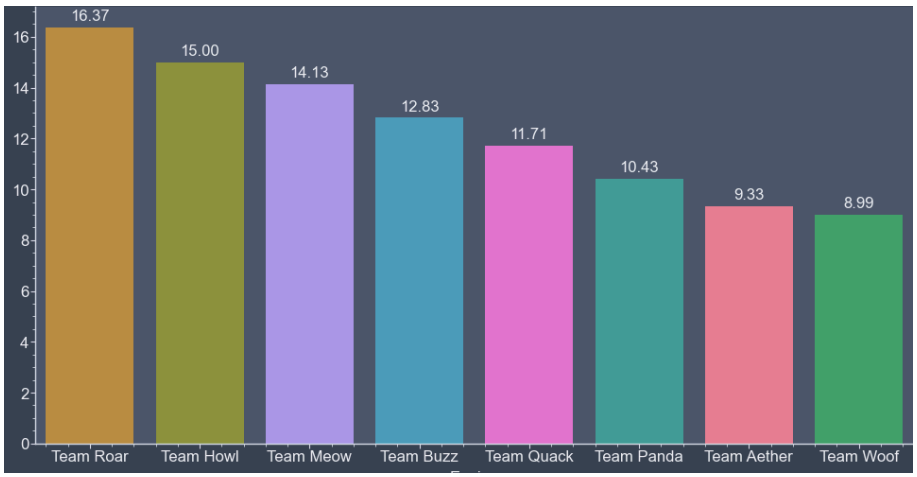

Figura 3-18: Tiempo promedio de resolución de incidencias por equipo
Fuente: Elaboración Propia, 2024

La figura 3-18 analiza el tiempo promedio en días en que cada equipo tarda en resolver una incidencia.

- "Team Roar" tiene el tiempo de resolución más alto (16.37 días), lo que sugiere que, aunque completan menos tareas, estas podrían ser más complejas o tener mayores bloqueos.
- "Team Aether" y "Team Woof", que tienen un alto volumen de incidencias completadas, mantienen tiempos de resolución relativamente bajos (9.33 y 8.99 días, respectivamente), lo que refleja **eficiencia** en su flujo de trabajo.
- Equipos como "Team Howl" y "Team Meow" tienen tiempos de resolución elevados, lo que podría indicar **cuellos de botella** o desafíos en la gestión de sus tareas.

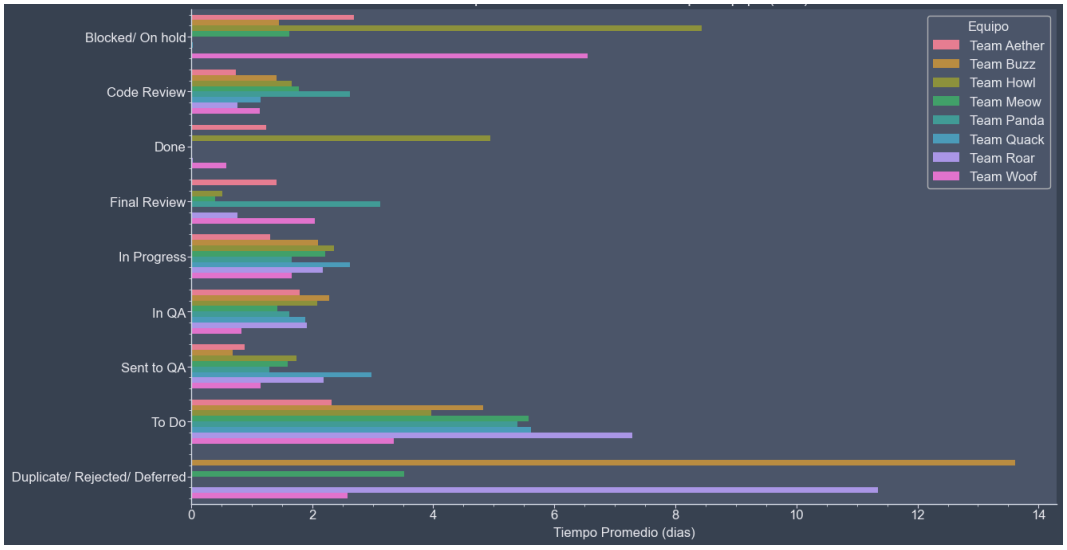

Figura 3-19: Tiempo promedio que pasa una incidencia en cada estado por equipo
Fuente: Elaboración Propia, 2024

La figura 3-19 muestra el tiempo promedio que los equipos pasan en cada estado del flujo de trabajo.

- Equipos como "Team Roar" y "Team Buzz" pasan un tiempo significativo en estados como "To Do" y "Duplicate/Rejected/Deferred", lo que podría sugerir demoras antes de que las tareas entren en desarrollo activo.
- El estado "In QA" es un punto de retraso para varios equipos, lo que podría indicar la necesidad de mejorar los procesos de prueba y revisión.
- "Blocked/On hold" es otro estado que consume tiempo para ciertos equipos, lo que sugiere problemas recurrentes que bloquean el progreso.

Identificación de Equipos Altamente Productivos

Equipos como "Team Aether" y "Team Woof" no solo completan una gran cantidad de tareas, sino que también mantienen tiempos de resolución bajos, lo que refleja un flujo de trabajo eficiente.

Detección de Áreas de Mejora

Equipos con tiempos de resolución elevados, como "Team Roar" y "Team Howl", podrían beneficiarse de un análisis más detallado para identificar cuellos de botella y optimizar sus procesos internos. El tiempo prolongado en estados como "In QA" y "Blocked" sugiere que hay oportunidades para mejorar la colaboración y la gestión de dependencias.

3.2.3.5. Análisis de la Tendencia Mensual de Incidencias Completadas

El siguiente gráfico analiza el número de incidencias completadas mensualmente entre mayo y octubre de 2024, revelando fluctuaciones en la productividad a lo largo del período.

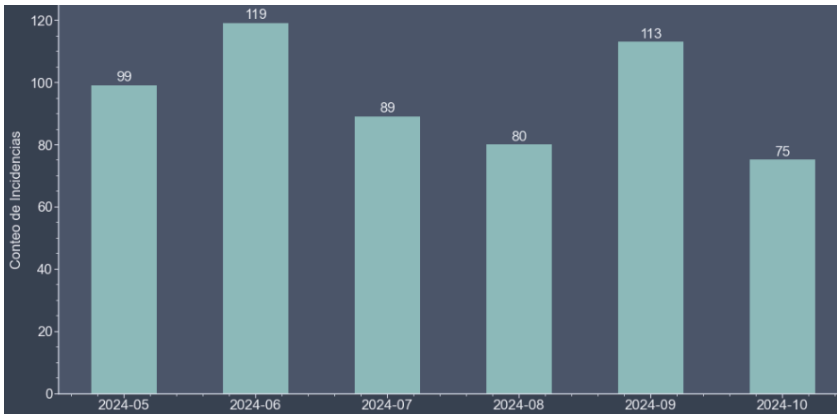

Figura 3-20: Incidencias completadas por mes
Fuente: Elaboración Propia, 2024

Picos de productividad en junio y septiembre

Los meses de junio (119 incidencias) y septiembre (113 incidencias) destacan como los más productivos, lo que sugiere picos de actividad posiblemente vinculados a sprints críticos o el cierre de fases importantes del proyecto. Estos aumentos podrían coincidir con fechas límite o entregas cruciales, lo que impulsa a los equipos a completar un mayor número de tareas.

Descenso en los meses de julio, agosto y octubre

En contraste, agosto (80 incidencias) y octubre (75 incidencias) muestran una disminución significativa en la cantidad de incidencias resueltas. Estas caídas pueden estar asociadas con vacaciones, cambios en la planificación o ajustes en el equipo y el backlog.

Recomendaciones

- Investigar las estrategias que contribuyeron a los picos de productividad en junio y septiembre para aplicarlas en otros períodos.
- Evaluar las razones detrás de las disminuciones en agosto y octubre, con el fin de identificar oportunidades de mejora en la gestión de recursos y la planificación de sprints.

3.2.3.6. Conclusiones

El análisis exploratorio permitió identificar varias áreas críticas que requieren atención, como los cuellos de botella en estados específicos ("Blocked" y "In QA") y las diferencias en la eficiencia entre equipos. Estos hallazgos proporcionan un punto de partida sólido para el modelado predictivo y la optimización de procesos.

3.2.4. Ingeniería de Características

La Ingeniería de características es un paso crucial en el proceso de creación de modelos de Machine Learning, ya que permite mejorar la calidad de los datos mediante la creación de nuevas características derivadas de las existentes, con el objetivo de maximizar la precisión y el rendimiento del modelo.

En este proyecto, dado que los datos provienen de incidencias de **JIRA** relacionadas con equipos de desarrollo ágil, se han generado características adicionales a partir de las columnas existentes para optimizar el análisis predictivo. A continuación, se detallan las principales características derivadas que se han creado y el propósito de cada una.

3.2.4.1. Incidencias bloqueadas

El propósito de este paso es identificar si una incidencia ha estado bloqueada en algún momento de su ciclo de vida, agregando una característica que lo refleje. Esto permite analizar el impacto de los bloqueos en el desempeño del equipo y detectar cuellos de botella que puedan afectar la eficiencia en la resolución de tareas, proporcionando información clave para la toma de decisiones y la mejora de los procesos.

```
data['Blocked'] = data['Initial State'].apply(lambda x: 1 if 'Blocked/ On hold' in x else 0)
data['Blocked'] = data.groupby('Key')['Blocked'].transform('max')
data.head()
```

Figura 3-21: Código para definir si una incidencia ha sido bloqueada
Fuente: Elaboración Propia, 2024

En el código de la Figura 3-21 se crea una nueva característica llamada"Blocked" para identificar si un ticket estuvo bloqueado en algún momento. Primero, se asigna un valor de 1 a los registros donde la columna "Initial State" contiene "Blocked/ On hold" y 0 en caso contrario. Luego, se agrupa por la columna "Key" y se toma el valor máximo de "Blocked" dentro de cada grupo, asegurando que cualquier ticket asociado a una clave quede marcado como bloqueado si al menos un registro lo estuvo.

3.2.4.2. Eficiencia del responsable

Este indicador refleja la eficiencia de un desarrollador al completar una incidencia, evaluando la precisión entre la estimación inicial y el tiempo real de resolución. Su utilidad radica en identificar patrones de desempeño, permitiendo determinar si ciertos desarrolladores son más eficientes al abordar tipos específicos de incidencias, lo que puede guiar asignaciones futuras y optimizar la productividad del equipo.

```
# Calculo eficiencia del responsable de la incidencia
data['Member Efficiency per ticket'] = np.where(data['Resolution Time'] > 0, data['Story Points'] / data['Resolution Time'], 0)
data.head()
```

Figura 3-22: Código para calcular la eficiencia de un desarrollador
Fuente: Elaboración Propia, 2024

La Figura 3-22 ilustra el cálculo de la eficiencia del o los desarrolladores, donde se determina dividiendo los puntos de historia asignados a una incidencia entre el tiempo empleado para resolverla. Este cálculo permite evaluar la productividad del desarrollador en función de la complejidad de las tareas asignadas y el tiempo invertido en completarlas.

3.2.4.3. Retrasos en el sprint

Al identificar si un sprint se completó después de la fecha programada, proporciona información clave para analizar la planificación, la capacidad del equipo y la gestión de recursos. Además, permite detectar patrones en retrasos, lo que puede ayudar a ajustar estimaciones futuras, optimizar la asignación de tareas y mejorar la eficiencia general en la gestión de proyectos.

```
# Calcular si el sprint se pasó de la fecha programada
data['Sprint Overdue'] = (data['Sprint Complete Date'] > data['Sprint End Date']).astype(int)
```

Figura 3-23: Código para identificar si un sprint ha presentado retrasos en su conclusión
Fuente: Elaboración Propia, 2024

La Figura 3-23 muestra el cálculo de la métrica "Sprint Overdue" que evalúa si un sprint se completó después de la fecha programada. Esto se logra comparando la fecha de finalización real del sprint (Sprint Complete Date) con la fecha planificada de finalización (Sprint End Date). Si la fecha real es posterior a la planificada, se asigna un valor de 1 en la columna Sprint Overdue; de lo contrario, se asigna un valor de 0. Este indicador permite identificar desviaciones en los plazos

3.2.4.4. Tiempo restante del sprint

El propósito de esta métrica es proporcionar una visión detallada sobre el tiempo restante en el sprint cuando se completa una incidencia. Esto ayuda a identificar patrones en el flujo de trabajo del equipo, como la acumulación de tareas hacia el final del sprint, la finalización temprana de incidencias o si el equipo mantiene un ritmo constante de trabajo. Al analizar esta información, se pueden tomar decisiones informadas para mejorar la planificación de los sprints, equilibrar la carga de trabajo y establecer estrategias que permitan un mejor cumplimiento de los plazos establecidos, maximizando la productividad del equipo.

```
# Días restantes en el sprint al momento de completar una incidencia
data['Days to Sprint End'] = (data['Sprint End Date'] - data['Sprint Complete Date']).dt.days
```

Figura 3-24: Código calcular los días restantes para la culminación de un sprint
Fuente: Elaboración Propia, 2024

La Figura 3-24 muestra el cálculo de los días restantes en el sprint al momento de completar una incidencia. Esta métrica se obtiene restando la fecha de finalización real de la incidencia (Sprint Complete Date) de la fecha programada de término del sprint (Sprint End Date). El resultado se almacena en la columna "Days to Sprint End", expresado en días.

3.2.4.5. Eficiencia del equipo por sprint

Un valor más alto en esta métrica indica una mayor eficiencia, ya que refleja la capacidad del equipo para resolver más "Story Points" en menos tiempo. El objetivo de esta métrica es evaluar el desempeño colectivo de un equipo durante un sprint, proporcionando una visión integral de su eficiencia promedio en completar tareas. Esto es útil para identificar diferencias en la productividad entre sprints o equipos, medir mejoras en el tiempo, y analizar si las asignaciones de tareas y recursos están siendo efectivas. Esta información permite ajustar estrategias de planificación y gestión de equipos para mejorar el rendimiento general en futuros sprints.

```
# Calculando la eficiencia media del equipo por sprint
data['Average Team Efficiency per Sprint'] = data.groupby(['Team', 'Sprint'])['Member Efficiency per ticket'].transform('mean')
data.head()
```

Figura 3-25: Código para calcular la eficiencia del equipo por sprint
Fuente: Elaboración Propia, 2024

La Figura 3-25 muestra el cálculo de la Eficiencia Promedio del Equipo por Sprint, donde se obtiene el promedio de la eficiencia de los miembros del equipo por ticket dentro de cada sprint. Esto se calcula

agrupando los datos por equipo (Team) y sprint (Sprint), y luego promediando los valores de la columna "Member Efficiency per ticket". El resultado se almacena en la columna "Average Team Efficiency per Sprint".

3.2.4.6. Complejidad de las incidencias

Se añadió una columna denominada "Complexity", el objetivo de esta métrica es proporcionar una segmentación clara de las incidencias según su nivel de complejidad, lo que facilita el análisis y la planificación. Al categorizar los Story Points, se pueden identificar patrones en el desempeño del equipo dependiendo de la complejidad de las tareas, ajustar las estimaciones de tiempo y esfuerzo, y optimizar la asignación de recursos. Esta clasificación también ayuda a comparar el impacto de tareas simples, medianas y complejas en el sprint y el flujo de trabajo general.

```
# Categorización de 'Story Points'

# Definir los bins para las categorías de complejidad
bins = [0, 3, 5, 21]  # Los límites incluyen el valor más bajo y el más alto de cada rango
labels = ['Low', 'Medium', 'High']

# Categorizar 'Story Points' usando cut de pandas
data['Complexity'] = pd.cut(data['Story Points'], bins=bins, labels=labels, right=True, include_lowest=True)

# Verificar las categorías asignadas
data.head()
```

Figura 3-26: Código para identificar la complejidad de las incidencias
Fuente: Elaboración Propia, 2024

La Figura 3-26 muestra la categorización de los Story Points en diferentes niveles de complejidad. Para ello, se utilizan rangos predefinidos (bins) y etiquetas descriptivas (labels), asignando una categoría a cada registro en función del valor de Story Points. En este caso:

- Valores de 0 a 3 se clasifican como "Low" (baja complejidad).
- Valores mayores a 3 y hasta 5 se clasifican como "Medium" (complejidad media).
- Valores mayores a 5 y hasta 21 se clasifican como "High" (alta complejidad).

3.2.4.7. Velocidad del equipo

El propósito de esta métrica es evaluar la carga total de trabajo asignada a cada equipo durante un sprint, permitiendo identificar desequilibrios en la distribución de tareas, analizar si la carga está alineada con la capacidad del equipo y comparar el esfuerzo total entre sprints o equipos. Esto facilita la optimización de la planificación y asegura una asignación equitativa y realista de los puntos de historia, mejorando la eficiencia y el cumplimiento de los objetivos del sprint.

```
# Suma de story points por equipo y sprint
data['Total Story Points per Sprint and team'] = data.groupby(['Team', 'Sprint'])['Story Points'].transform('sum')
data.head()
```

Figura 3-27: Código para calcular la cantidad de story points completados por sprint y por equipo
Fuente: Elaboración Propia, 2024

La Figura 3-27 muestra la suma de "Story Points" por "Equipo" y "Sprint", donde se calcula el total de puntos de historia asignados a cada equipo dentro de un sprint específico. Esto se realiza agrupando los

datos por las columnas Team (equipo) y Sprint (sprint) y sumando los valores de la columna "Story Points" dentro de cada grupo. El resultado se almacena en la nueva columna "Total Story Points per Sprint and team" que es la velocidad del equipo, la cantidad de puntos de historia que puede completar en un sprint.

3.2.4.8. Velocidad del responsable

Esta métrica proporciona una visión clara de la carga de trabajo individual dentro de un sprint, permitiendo identificar desequilibrios en la asignación de tareas, evaluar si los puntos de historia asignados son acordes a las capacidades de cada persona, y analizar patrones de desempeño individual. Esto es fundamental para mejorar la planificación, garantizar una distribución equitativa de las tareas y maximizar la productividad del equipo al aprovechar las fortalezas individuales de los miembros.

```
# Suma de story points por miembro del equipo y sprint
data['Total Story Points per Member per Sprint'] = data.groupby(['Assignee', 'Sprint'])['Story Points'].transform('sum')
data.head()
```

Figura 3-28: Código para determinar los story points completados por cada responsable en cada sprint
Fuente: Elaboración Propia, 2024

La Figura 3-28 muestra el cálculo de la suma de "Story Points" por miembro de equipo y el sprint. Para esto, se agrupan los datos por las columnas "Assignee" (miembro del equipo responsable de la tarea) y "Sprint", y se calcula la suma de los puntos de historia (Story Points) asignados a cada miembro en cada sprint. El resultado de esta agrupación se almacena en la nueva columna "Total Story Points per Member per Sprint".

3.2.4.9. Precisión de las Estimaciones

El objetivo de esta métrica es analizar la calidad de las estimaciones realizadas al planificar las incidencias. Identificar estimaciones precisas o imprecisas permite comprender si los puntos de historia reflejan adecuadamente el esfuerzo necesario para completar una tarea. Esta información es crucial para mejorar la capacidad del equipo en la planificación, ajustar los criterios de estimación y reducir desajustes que puedan afectar la entrega y el desempeño del sprint.

```
def estimate_accuracy(row):
    points = row['Story Points']
    resolution_time_days = row['Resolution Time']  # Asegurarse de que está en días
    resolution_time_hours = resolution_time_days * 24  # Convertir días en horas para mayor precisión

    if (points == 1 and resolution_time_hours <= 12) or \
       (points == 2 and 12 < resolution_time_hours <= 36) or \
       (points == 3 and 36 < resolution_time_hours <= 72) or \
       (points == 5 and 72 < resolution_time_hours <= 96) or \
       (points == 8 and 96 < resolution_time_hours <= 168) or \
       (points == 13 and resolution_time_hours > 168):
        return 1
    else:
        return 0

# Aplica la función para crear la nueva columna
data['Estimation Accurate'] = data.apply(estimate_accuracy, axis=1)
data
```

Figura 3-29: Código para calcular la precisión de una estimación
Fuente: Elaboración Propia, 2024

La Figura 3-29 muestra la creación de la métrica Precisión de la Estimación (Estimation Accurate), que evalúa si las estimaciones iniciales de los puntos de historia son precisas en relación con el tiempo real de

resolución de la incidencia. Para ello, se define una función personalizada, **estimate_accuracy**, que asigna un valor de **1** (estimación precisa) o **0** (estimación imprecisa) comparando los puntos de historia asignados (Story Points) con el tiempo de resolución real (Resolution Time) en horas. Se establecen rangos de tiempo específicos para cada valor de puntos de historia, considerando estándares comunes en la industria para evaluar precisión.

Para concluir esta etapa de feature engineering, se eliminaron filas duplicadas y columnas redundantes que ya no aportan información, ya que fueron utilizadas para calcular métricas que ahora están representadas en nuevas variables más relevantes. El conjunto de datos resultante está listo para el proceso de one-hot encoding y normalización, previo al entrenamiento de los modelos. Hasta este punto, el dataset cuenta con 564 filas y 19 columnas.

3.2.5. Preparación del conjunto de datos para el modelado

La preparación del conjunto de datos es un paso fundamental en el flujo de trabajo de Machine Learning. Después del análisis exploratorio y el Feature Engineering, es necesario transformar y limpiar los datos para que los modelos puedan **entrenarse de manera efectiva**. En este apartado, se detalla los pasos seguidos para preparar el conjunto de datos.

3.2.5.1. Manejo de valores faltantes

La Figura 3-30 presenta un conteo de valores nulos en cada columna del conjunto de datos. Se observa que la columna **Resolution Time** contiene algunos valores nulos. Esto indica que, para ciertas incidencias, la información sobre el tiempo de resolución no está disponible.

```
# Ver la cantidad de valores faltantes por columna
missing_values = data.isnull().sum()
print(missing_values)
```

```
Key                                     0
Parent Key                              0
Priority                                0
Story Points                            0
Assignee                                0
Issue Type                              0
Sprint                                  0
Team                                    0
Resolution Time                         2
Blocked                                 0
Member Efficiency per ticket            0
Sprint Overdue                          0
Days to Sprint End                      0
Complexity                              0
Total Story Points per Sprint and team  0
Total Story Points per Member per Sprint 0
Average Team Efficiency per Sprint      0
Block Rate by Issue Type                0
Estimation Accurate                     0
dtype: int64
```

Figura 3-30: Código para contar valores nulos
Fuente: Elaboración Propia, 2024

Dado que la cantidad de valores faltantes es baja, se optó por completar estos datos utilizando la **mediana** para la columna **Resolution Time** (Figura 3-31). Esta estrategia asegura que las imputaciones no distorsionen significativamente el análisis posterior.

```
data['Resolution Time'].fillna(data['Resolution Time'].median(), inplace=True)
```

Figura 3-31: Código para completar valores faltantes
Fuente: Elaboración Propia, 2024

3.2.5.2. Normalización y Estandarización

Dado que las **escalas de los datos** pueden variar considerablemente (por ejemplo, "Total Story Points per Sprint and Team" frente a "Resolution Time"), es fundamental **normalizar** o **estandarizar** los datos para mejorar la eficiencia y precisión del modelo.

```
from sklearn.preprocessing import MinMaxScaler

# Inicializar el escalador
scaler = MinMaxScaler()

# Escalar todas las columnas numéricas automáticamente
numeric_columns = data.select_dtypes(include=['float64', 'int64']).columns
data[numeric_columns] = scaler.fit_transform(data[numeric_columns])
data
```

Figura 3-32: Código para normalizar variables cuantitativas
Fuente: Elaboración Propia, 2024

La Figura 3-32 muestra la implementación del Min-Max Scaler para normalizar las columnas cuantitativas o numéricas del conjunto de datos. Esta técnica ajusta los valores dentro de un rango de 0 a 1, lo que ayuda a garantizar que todas las características tengan el mismo peso al entrenar los modelos de machine learning.

3.2.5.3. Codificación de Variables Categóricas

Después de normalizar las variables cuantitativas, es necesario transformar también las variables cualitativas mediante un proceso llamado **one-hot encoding** (Figura 3-33). Este método convierte las categorías en nuevas columnas binarias con valores de 0 y 1, indicando si un registro pertenece o no a una categoría específica. Esto permite que los modelos de machine learning puedan interpretar y utilizar las características cualitativas de manera eficiente.

```
# one hot encoding
# Seleccionar columnas categóricas
columns_to_exclude = ['Key', 'Blocked', 'Estimation Accurate']
category_columns = [col for col in data.select_dtypes(include=['category']).columns if col not in columns_to_exclude]

# Aplicar pd.get_dummies solo a las columnas categóricas seleccionadas
data = pd.get_dummies(data, columns=category_columns, dtype=int, drop_first=True)
data
```

Figura 3-33: Código para aplicar one-hot encoding a las variables cualitativas
Fuente: Elaboración Propia, 2024

En la figura se observa que se excluyen ciertas columnas. En primer lugar, la columna "Key", que corresponde al identificador único de cada incidencia, será eliminada posteriormente debido a que no aporta valor para el entrenamiento del modelo. Además, las columnas "Blocked" y "Estimation Accurate" también

se excluyen, ya que sus valores están en formato binario (únicamente ceros y unos), lo que indica que ya no requieren transformaciones adicionales para su procesamiento.

Con este proceso se finaliza la preparación del conjunto de datos, el cual queda completamente listo para ser utilizado en el entrenamiento de los modelos de machine learning.

3.2.6. Entrenamiento de los modelos

Para alcanzar los objetivos planteados en este proyecto, es fundamental seleccionar y entrenar los modelos de machine learning más adecuados para **predecir el tiempo de resolución de incidencias**. Para ello, se llevarán a cabo entrenamientos utilizando tres modelos principales: **Gradient Boosting, Random Forest y XGBoost**.

La predicción del **tiempo de resolución** es un componente clave del rendimiento de los equipos. Al predecir con precisión cuánto tiempo tomará resolver una incidencia, los equipos pueden ajustar sus sprints y mejorar sus estimaciones, alineando sus esfuerzos con los objetivos del proyecto.

El enfoque consistirá en emplear **regresores** para la predicción del tiempo de resolución de incidencias. Esta combinación de modelos permitirá capturar tanto patrones complejos no lineales como relaciones ocultas en los datos, optimizando la precisión.

Para el entrenamiento de todos los modelos, los hiper parámetros se ajustaron de manera flexible con el objetivo de optimizar los resultados. Para identificar los valores más adecuados, se utilizó la herramienta Randomized Search de Scikit-learn para llevar a cabo una búsqueda eficiente de los hiper parámetros óptimos que ofrecieran el mejor rendimiento y desempeño en los modelos. Este enfoque permitió explorar de manera estratégica un espacio de parámetros, equilibrando precisión y eficiencia computacional.

3.2.6.1. División del conjunto de datos

Antes de entrenar los modelos, es fundamental dividir el conjunto de datos en dos partes: un conjunto de entrenamiento y un conjunto de prueba. Para este proyecto, se ha optado por una división de **80% para entrenamiento** y **20% para prueba**.

Al asignar el 80% de los datos al entrenamiento, se permite que el modelo aprenda de una gran cantidad de datos, lo que le proporciona una base sólida para identificar patrones. El 20% restante se utiliza para evaluar su rendimiento en datos que no ha visto previamente, simulando cómo se comportaría en un entorno real.

Esta proporción de 80/20 es una práctica común que permite reducir el riesgo de sobreajuste (overfitting), ya que se deja una porción suficiente de datos reservada para la evaluación, asegurando que el modelo generalice bien a datos nuevos.

3.2.6.2. Definición de la variable objetivo

La Figura 3-34 ilustra cómo se eliminan las columnas "Resolution Time" y el identificador único de cada incidencia. La columna "Resolution Time" se elimina porque representa la variable objetivo en este entrenamiento, mientras que el identificador no aporta valor predictivo. Posteriormente, se utiliza **train_test_split** para dividir los datos en 80% para entrenamiento y 20% para prueba. El parámetro **random_state** se emplea para asegurar que la división sea reproducible, garantizando resultados consistentes en cada ejecución.

```python
from sklearn.model_selection import train_test_split

X = data.drop(columns=['Resolution Time', 'Key'])
y = data['Resolution Time']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figura 3-34: Código que divide el conjunto de datos para el entrenamiento
Fuente: Elaboración Propia, 2024

3.2.6.3. Gradient Boosting Regressor

Para optimizar el tiempo de búsqueda de los hiper parámetros más adecuados para el conjunto de datos y el modelo de machine learning, en este caso Gradient Boosting Regressor, se optó por definir rangos preestablecidos para los hiper parámetros. Estos rangos corresponden a valores comúnmente utilizados, que suelen ofrecer un buen rendimiento en modelos similares.

Se empleó la herramienta RandomizedSearchCV, una alternativa más adecuada en este caso debido a que ofrece una búsqueda más rápida en comparación con GridSearchCV. Esto resulta especialmente útil cuando se enfrentan limitaciones en la capacidad del equipo para soportar una optimización exhaustiva como la que realiza GridSearchCV. Por ello, se optó por RandomizedSearchCV para realizar una búsqueda eficiente dentro de los rangos definidos, evaluando múltiples combinaciones de hiper parámetros.

Esta búsqueda se llevó a cabo utilizando la métrica del error medio absoluto (MAE) como criterio de evaluación, lo que permitió identificar la combinación de valores que proporcionó el mejor desempeño del modelo. Posteriormente, los hiper parámetros óptimos resultantes fueron aplicados para entrenar el modelo de manera efectiva, maximizando su rendimiento.

```python
# Definir el espacio de búsqueda
param_distributions = {
    'n_estimators': [100, 200, 500, 1000],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.5, 0.7, 1.0]
}

# Configurar Randomized Search
random_search = RandomizedSearchCV(
    estimator=GradientBoostingRegressor(random_state=42),
    param_distributions=param_distributions,
    n_iter=50,  # Número de combinaciones a probar
    scoring='neg_mean_absolute_error',
    cv=5,  # Validación cruzada de 5 pliegues
    random_state=42,
    n_jobs=-1  # Paralelismo
)

# Entrenar Randomized Search
random_search.fit(X_train, y_train)

# Mostrar los mejores hiperparámetros y el mejor MAE
best_params = random_search.best_params_
print(f"Best Parameters: {best_params}")
```

```
Best Parameters: {'subsample': 1.0, 'n_estimators': 500, 'min_samples_split': 2, 'min_samples_leaf': 1,
'max_depth': 5, 'learning_rate': 0.05}
```

Figura 3-35: Búsqueda de hiper parámetros óptimos para Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

Como resultado de la búsqueda de hiper parámetros se tiene la siguiente configuración:

- **subsample = 1.0:** Usa el 100% de los datos para cada árbol, sin introducir aleatoriedad.

- **n_estimators = 500:** Genera 500 árboles para mejorar el rendimiento, pero incrementa el tiempo de entrenamiento.
- **min_samples_split = 2 y min_samples_leaf = 1:** Permiten divisiones con pocos datos, capturando detalles finos, aunque con riesgo de sobreajuste.
- **max_depth = 5:** Limita la profundidad de los árboles para evitar sobreajuste.
- **learning_rate = 0.05:** Reduce el impacto de cada árbol, asegurando aprendizaje estable y menos sobreajuste.
- **random_state = 42:** El uso de una semilla fija asegura que los resultados sean reproducibles en cada ejecución.

```python
from sklearn.ensemble import GradientBoostingRegressor

gbr_model = GradientBoostingRegressor(
    subsample=1.0,
    n_estimators=500,
    min_samples_split=2,
    min_samples_leaf=1,
    max_depth=5,
    learning_rate=0.05,
    random_state=42
)

# Entrenar el modelo con el conjunto de entrenamiento
gbr_model.fit(X_train, y_train)
```

Figura 3-36: Código de entrenamiento del modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

Además del entrenamiento del modelo (Figura 3-36), se generó un gráfico de importancia de características para mostrar qué variables tuvieron mayor influencia en el rendimiento del modelo. Esto resulta especialmente útil, ya que permite visualizar cuáles son las características que más contribuyeron al proceso de predicción, facilitando una mejor comprensión del comportamiento del modelo y ayudando a refinar el análisis.

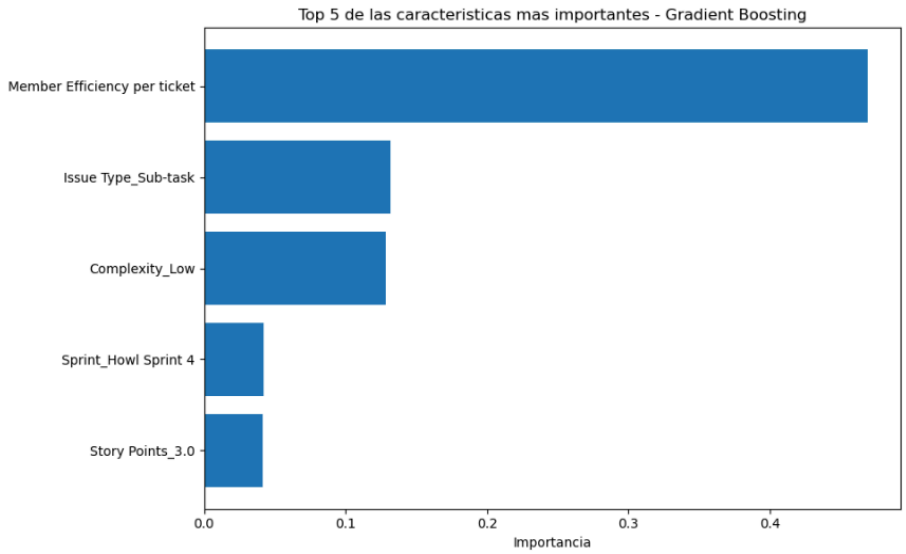

Figura 3-37: Importancia de las características para el modelo Gradient Boosting Regressor

Fuente: Elaboración Propia, 2024

En este modelo, se observa que la característica "Member Efficiency per ticket" tuvo la mayor influencia en el entrenamiento, seguida de "Issue Type_Sub-task" y "Complexity_Low" (Figura 3-37). En general, el modelo muestra que su desempeño depende principalmente de estos tres factores: la eficiencia de los desarrolladores, el tipo de incidencia y su nivel de complejidad.

3.2.6.4. Random Forest Regressor

De manera similar al modelo anterior, se utilizará Randomized Search para identificar los hiper parámetros más óptimos, asegurando un ajuste eficiente y mejorando el rendimiento del modelo (Figura 3-38).

```python
from sklearn.model_selection import RandomizedSearchCV

# Definir el espacio de búsqueda de hiperparámetros
param_distributions = {
    'n_estimators': [100, 200, 500, 1000],
    'max_depth': [None, 10, 20, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [1.0, 'sqrt', 'log2']
}

# Configurar RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_distributions=param_distributions,
    n_iter=50,  # Número de combinaciones aleatorias a probar
    scoring='neg_mean_absolute_error',
    cv=5,  # Validación cruzada de 5 pliegues
    random_state=42,
    n_jobs=-1  # Paralelismo para acelerar el entrenamiento
)

# Entrenar la búsqueda aleatoria
random_search.fit(X_train, y_train)

# Mostrar los mejores hiperparámetros encontrados
best_params = random_search.best_params_
print(f"Mejores Hiper parametros: {best_params}")
```

```
Best Parameters: {'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 1.0, 'max_depth': 10}
```

Figura 3-38: Búsqueda de hiper parámetros óptimos para Random Forest Regressor
Fuente: Elaboración Propia, 2024

Como resultado de la búsqueda de hiper parámetros tenemos la siguiente configuración:

- **n_estimators = 500:** Incrementa la estabilidad y precisión del modelo, pero aumenta el tiempo de entrenamiento/predicción.
- **min_samples_split = 5:** Simplifica los árboles al limitar las divisiones, reduciendo el riesgo de sobreajuste.
- **min_samples_leaf = 2:** Evita hojas pequeñas y simplifica el modelo, reduciendo el riesgo de sobreajuste.
- **max_features = 1.0:** Usa todas las características, lo que puede ser útil para un dataset con pocas características.
- **max_depth = None:** Permite árboles profundos para capturar relaciones complejas, pero con mayor riesgo de sobreajuste.
- **random_state = 42:** Número de semilla para asegurar la reproducibilidad de los resultados.
- **n_jobs = -1:** Utiliza todos los núcleos disponibles del procesador, acelerando significativamente el proceso de entrenamiento.

```python
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(
    random_state=42,
    n_estimators=500,
    min_samples_split=5,
    min_samples_leaf=2,
    max_features=1.0,
    max_depth=None,
    n_jobs = -1
)
rf_model.fit(X_train, y_train)
```

Figura 3-39: Código de entrenamiento del modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

La importancia de características para este modelo se muestra en la figura siguiente:

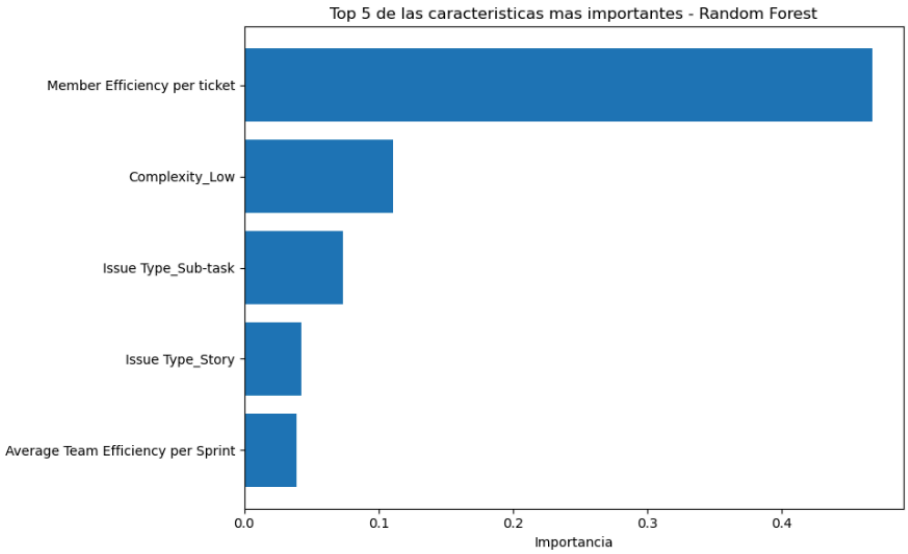

Figura 3-40: Importancia de las características para el modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

En este modelo, la variable "Member Efficiency per ticket" es la más influyente en el entrenamiento, seguida de "Complexity_Low" y "Issues Type_Sub-task" (Figura 3-40). Al igual que el modelo anterior, su desempeño depende principalmente de la eficiencia individual del desarrollador, así como de la complejidad y el tipo de incidencia.

3.2.6.5. XGBoost Regressor

La Figura 3-41 muestra la aplicación de Randomized Search para encontrar los hiper parámetros óptimos del modelo XGBoost Regressor, permitiendo optimizar el entrenamiento y maximizar su rendimiento.

```python
# Espacio de búsqueda para Randomized Search
param_distributions = {
    'n_estimators': [100, 200, 500, 1000],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 5, 10, None],
    'subsample': [0.5, 0.7, 1.0],
    'colsample_bytree': [0.5, 0.7, 1.0],
    'gamma': [0, 0.1, 0.2, 1],
    'reg_alpha': [0, 0.1, 1],
    'reg_lambda': [1, 1.5, 2]
}

# Configurar Randomized Search CV
random_search_xgb = RandomizedSearchCV(
    estimator=XGBRegressor(random_state=42),
    param_distributions=param_distributions,
    n_iter=50,  # Número de combinaciones a probar
    scoring='neg_mean_absolute_error',
    cv=5,  # Validación cruzada de 5 pliegues
    random_state=42,
    n_jobs=-1  # Paralelismo
)

# Entrenar Randomized Search
random_search_xgb.fit(X_train, y_train)

# Obtener los mejores hiperparámetros y el mejor MAE
best_params_xgb = random_search_xgb.best_params_

print(f"Best Parameters: {best_params_xgb}")
```

```
Best Parameters: {'subsample': 1.0, 'reg_lambda': 1.5, 'reg_alpha': 0, 'n_estimators': 500, 'max_depth': 5,
'learning_rate': 0.05, 'gamma': 0, 'colsample_bytree': 1.0}
```

Figura 3-41: Búsqueda de hiper parámetros óptimos para XGBoost Regressor
Fuente: Elaboración Propia, 2024

El modelo fue configurado con los siguientes hiper parámetros (figura 3-41) cuidadosamente seleccionados para optimizar su rendimiento:

- **subsample = 1.0:** Utiliza el 100% de los datos en cada iteración. Introducir aleatoriedad (valores menores) puede prevenir sobreajuste, pero aquí no se aplica.
- **reg_lambda = 1.5 (L2 regularización):** Penaliza valores altos de los coeficientes para evitar sobreajuste. Un valor de 1.5 agrega una moderada regularización.
- **reg_alpha = 0 (L1 regularización):** No se aplica penalización para la selección de características (sparse weights). Si se incrementa, puede eliminar características irrelevantes.
- **n_estimators = 500:** Usa 500 árboles, permitiendo que el modelo aprenda mejor patrones complejos. Más árboles implican más tiempo de entrenamiento.
- **max_depth = 5:** Restringe la profundidad de los árboles a 5 niveles, equilibrando el aprendizaje entre captar detalles y evitar sobreajuste.
- **learning_rate = 0.05:** Controla la velocidad de aprendizaje. Un valor bajo como 0.05 asegura estabilidad en el ajuste, pero requiere más iteraciones.
- **gamma = 0:** No se aplica penalización en el criterio de división. Si aumenta, se requiere mayor ganancia en reducción de error para dividir un nodo.
- **colsample_bytree = 1.0:** Usa el 100% de las características en cada árbol. Reducir este valor puede introducir aleatoriedad y mejorar la generalización.

```python
from xgboost import XGBRegressor

xgb_model = XGBRegressor(
    subsample=1.0,
    reg_lambda=1.5,
    reg_alpha=0,
    n_estimators=500,
    max_depth=5,
    learning_rate=0.05,
    gamma=0,
    colsample_bytree=1.0,
    random_state=42
)

# Entrenar el modelo con el conjunto de entrenamiento
xgb_model.fit(X_train, y_train)
```

Figura 3-42: Código de entrenamiento del modelo XGBoost Regressor
Fuente: Elaboración Propia, 2024

La importancia de características para este modelo se muestra en la figura siguiente:

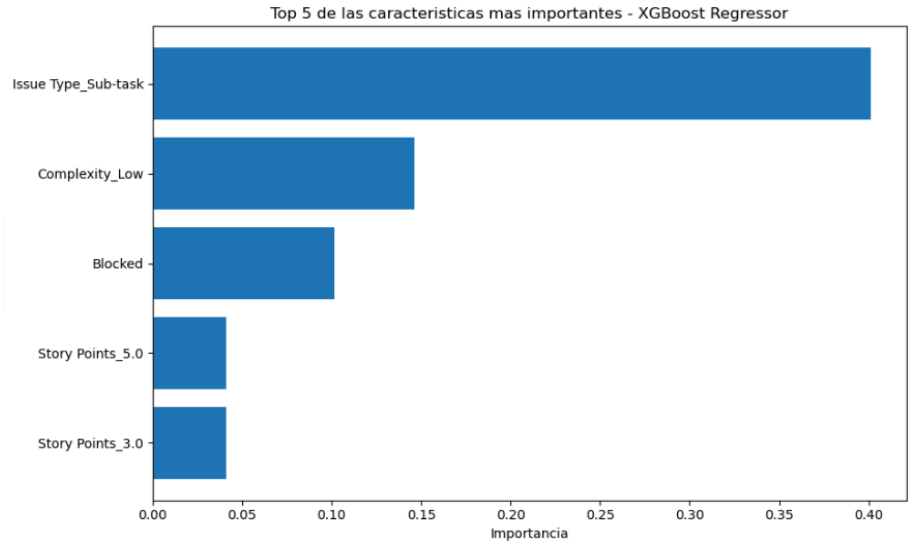

Figura 3-43: Importancia de las características para el modelo XGBoost Regressor
Fuente: Elaboración Propia, 2024

En este modelo, la variable "Issue Type_Sub-task" tiene la mayor influencia en el entrenamiento, seguida de "Complexity_Low" y "Blocked" (Figura 3-43). A diferencia de los modelos previos, este muestra una mayor dependencia de la característica "Blocked" (si la incidencia estuvo bloqueada), aunque también mantiene cierta influencia del tipo y la complejidad de la incidencia.

3.2.7. Evaluación y Selección del Modelo

En esta sección, se lleva a cabo un análisis exhaustivo de los modelos entrenados para evaluar su **rendimiento** y determinar cuál es el más adecuado para alcanzar los objetivos del proyecto. Después de realizar el preprocesamiento de datos y entrenar diversos modelos de machine learning, es crucial una evaluación rigurosa que permita comparar resultados y seleccionar el modelo que ofrezca el mejor equilibrio entre precisión, generalización y eficiencia.

El enfoque de evaluación se basa en el uso de **métricas clave** que varían según el tipo de problema.

Para la predicción del tiempo de resolución de incidencias, se emplearán métricas como:

- **R^2 Score**: para medir la capacidad del modelo de explicar la variabilidad de los datos.
- **Mean Absolute Error (MAE)**: para evaluar la **precisión** en la predicción de valores absolutos.
- **Mean Squared Error (MSE)**: para penalizar los errores más grandes y evaluar el ajuste del modelo.

El análisis se centrará en tres modelos principales para cada uno de los dos objetivos: **Gradient Boosting, Random Forest y XGBoost**

Finalmente, se seleccionará el **modelo más efectivo**, optimizando así la capacidad de los equipos de desarrollo para anticipar posibles bloqueos y gestionar sus tareas de forma más eficiente en un entorno ágil.

A continuación, se presentan los resultados de las métricas utilizadas para evaluar el rendimiento de los modelos, proporcionando una visión clara de su capacidad predictiva y su ajuste a los datos.

3.2.7.1. Evaluación del Rendimiento de Gradient Boosting Regressor

```
y_pred = gbr_model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Gradient Boosting Regressor")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")
```

```
Gradient Boosting Regressor
Mean Absolute Error (MAE): 0.00971009077452371
Mean Squared Error (MSE): 0.0012999961502389518
R² Score: 0.8534653405191435
```

Figura 3-44: Código de evaluación del modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

- **Mean Absolute Error (MAE): 0.0097**
  El MAE, con un valor promedio de error de **0.0097** días (aproximadamente 14 minutos), muestra que el modelo realiza **predicciones precisas**.
- **Mean Squared Error (MSE): 0.0013**
  El bajo valor del MSE confirma que el modelo **minimiza los errores extremos**, lo que refleja su capacidad para **generalizar bien** en todo el conjunto de datos. Esto asegura que las predicciones no se vean distorsionadas por grandes desviaciones, permitiendo una planificación más confiable.
- **R^2 Score: 0.8535**
  Con un R^2 de **85.35%**, el modelo explica una proporción significativa de la variabilidad en los tiempos de resolución. Este nivel de ajuste es excelente en contextos de datos reales, donde factores externos impredecibles también pueden influir.

Estos resultados indican que el modelo es **consistente, preciso y confiable**, lo que lo convierte en una herramienta eficaz para predecir tiempos de resolución. Su capacidad para minimizar errores y explicar una

alta proporción de la variabilidad mejora la planificación de sprints y permite a los equipos optimizar su flujo de trabajo.

```
cv_scores = cross_val_score(gbr_model, X_train, y_train, scoring='neg_mean_absolute_error', cv=5)
print("Mean CV MAE:", -cv_scores.mean())

Mean CV MAE: 0.014653467065191179
```

Figura 3-45: Código de validación cruzada para el modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

En la validación cruzada (Figura 3-45), el modelo obtuvo 0.0147 días (aproximadamente 21 minutos). Esto significa que, en promedio, el error absoluto en las predicciones del tiempo de resolución de incidencias durante la validación es bajo y consistente, lo que indica que el modelo tiene una buena capacidad generalizadora y es fiable para predecir tiempos de resolución en nuevos datos.

3.2.7.2. Evaluación del Rendimiento de Random Forest Regressor

```
y_pred = rf_model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Random Forest Regressor")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")

Random Forest Regressor
Mean Absolute Error (MAE): 0.009545405321226974
Mean Squared Error (MSE): 0.0012336088533237578
R² Score: 0.8609484703311312
```

Figura 3-46: Código de evaluación del modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

- **Mean Absolute Error (MAE): 0.0095**
  El error promedio absoluto es de 0.0095 días (aproximadamente 14 minutos). Este resultado indica que el modelo tiene una buena precisión, logrando predicciones que se aproximan de manera consistente a los valores reales. Un MAE bajo es esencial para garantizar que las predicciones sean útiles en la planificación y ejecución operativa.
- **Mean Squared Error (MSE): 0.0012**
  El MSE bajo sugiere que el modelo evita errores significativos y mantiene **consistencia** en sus predicciones. Aunque penaliza más los errores grandes, este resultado refleja que el modelo realiza un buen ajuste general a los datos y no se ve afectado por grandes desviaciones.
- **R^2 Score: 0.8609**
  Con un R^2 de **86.09%**, el modelo explica una proporción considerable de la variabilidad en los tiempos de resolución. Este resultado indica que el modelo tiene un **ajuste sólido** y es capaz de

capturar patrones importantes en los datos, proporcionando predicciones que se alinean bien con las tendencias observadas.

El modelo de Random Forest Regressor demuestra ser **preciso y consistente**, capaz de realizar predicciones útiles para anticipar tiempos de resolución. Sus métricas reflejan que es una herramienta confiable para apoyar la toma de decisiones en la planificación operativa y la optimización del flujo de trabajo.

```
cv_scores = cross_val_score(rf_model, X_train, y_train, scoring='neg_mean_absolute_error', cv=5)
print("Mean CV MAE:", -cv_scores.mean())

Mean CV MAE: 0.017713011637737
```

Figura 3-47: Código de validación cruzada para el modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

En la validación cruzada, el modelo Random Forest Regressor obtuvo 0.0177 días (aproximadamente 25 minutos). Esto sugiere que, aunque el modelo es preciso durante el entrenamiento, su capacidad para generalizar a nuevos datos tiene un leve incremento en el error promedio, pero sigue siendo suficientemente confiable para predecir tiempos de resolución de incidencias con buena exactitud en aplicaciones prácticas.

3.2.7.3. Evaluación del Rendimiento de XGBoost Regressor

```
y_pred = xgb_model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("XGB Regressor")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")

XGB Regressor
Mean Absolute Error (MAE): 0.014060011146285734
Mean Squared Error (MSE): 0.0029666006200942184
R² Score: 0.6656068469115848
```

Figura 3-48: Código de evaluación del modelo XGBoost Regressor
Fuente: Elaboración Propia, 2024

- **Mean Absolute Error (MAE): 0.0141**
  El error promedio absoluto es de 0.0141 días (aproximadamente 20 minutos). Este valor muestra que el modelo tiene una precisión razonable, aunque implica que, en promedio, las predicciones pueden desviarse casi un minuto de los valores reales.
- **Mean Squared Error (MSE): 0.0029**
  El MSE refleja que el modelo tiene errores ligeramente mayores que penalizan las desviaciones más grandes. Este valor indica que el modelo tiene un buen rendimiento general, aunque podrían existir algunos casos con errores más significativos.
- **R^2 Score: 0.6656**
  Muestra que el modelo explica solo el 66.56% de la variabilidad en los tiempos de resolución. Esto

sugiere que el XGB Regressor tiene un rendimiento significativamente menor que el Random Forest Regressor y el Gradient Boosting Regressor

Aunque tiene un rendimiento sólido, su capacidad explicativa es ligeramente inferior en comparación con el Random Forest Regressor, lo que podría hacerlo menos eficiente en ciertos escenarios.

```
cv_scores = cross_val_score(xgb_model, X_train, y_train, scoring='neg_mean_absolute_error', cv=5)
print("Mean CV MAE:", -cv_scores.mean())

Mean CV MAE: 0.018399319934724233
```

Figura 3-49: Código de validación cruzada para el modelo XGBoost Regressor
Fuente: Elaboración Propia, 2024

En la validación cruzada (Figura 3-49), el modelo XGB Regressor obtuvo un Mean CV MAE de 0.0184 días (aproximadamente 26 minutos). Esto indica que el error absoluto promedio al generalizar a nuevos datos es relativamente alto en comparación con otros modelos como Random Forest o Gradient Boosting. Aunque el modelo muestra cierta capacidad predictiva, su rendimiento en validación cruzada refuerza que es menos preciso y menos consistente, sugiriendo que podría no ser la mejor opción para predecir los tiempos de resolución de incidencias en este caso

3.2.7.4. Tabla Comparativa

Modelo	MAE	MSE	R^2 Score	Validación Cruzada
Gradient Boosting Regressor	0.00971	0.00130	0.85347	0.01465
Random Forest Regressor	0.00954	0.00123	0.86094	0.01771
XGBoost Regressor	0.01406	0.00297	0.66561	0.01840

Tabla 3-4: Tabla comparativa del rendimiento de los modelos regresores
Fuente: Elaboración Propia, 2024

- **Gradient Boosting Regressor**

Fortalezas	Debilidades
Su MAE (0.00971) y MSE (0.00130) están entre los más bajos, indicando que es un modelo preciso y que maneja bien los errores grandes.	Aunque es ligeramente menos preciso que Random Forest en el conjunto de entrenamiento, la diferencia es marginal (0.00971 vs. 0.00954 en MAE).
Tiene un R^2 Score alto (0.85347), lo que significa que explica más del 85% de la variabilidad en los tiempos de resolución, asegurando un ajuste robusto.	
En la validación cruzada (0.01465), tiene el mejor desempeño generalizado, lo que indica que es menos propenso a sobre ajustarse y funciona de manera confiable con datos nuevos.	

Tabla 3-5: Tabla de fortalezas y debilidades del modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

Conclusión:
Este modelo es la mejor opción, especialmente si el objetivo es un modelo confiable que generalice bien y mantenga la precisión al predecir tiempos de resolución en diferentes escenarios.

- **Random Forest Regressor**

Fortalezas	Debilidades
Es el modelo más preciso en términos de MAE (0.00954) y MSE (0.00123), lo que significa que sus predicciones están más cerca de los valores reales.	En la validación cruzada, su MAE (0.01771) es más alto que el de Gradient Boosting, lo que sugiere que es más susceptible al sobreajuste y puede no generalizar tan bien en datos nuevos.
Tiene el R^2 Score más alto (0.86094), indicando que explica un 86% de la variabilidad en los datos, lo que lo convierte en el modelo mejor ajustado en el conjunto de entrenamiento.	Su buen desempeño en el conjunto de entrenamiento podría deberse a que explota más las características del conjunto de datos, sacrificando algo de generalización.

Tabla 3-6: Tabla de fortalezas y debilidades del modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

Conclusión:
Este modelo puede ser útil si la precisión en datos conocidos es la prioridad, pero su menor capacidad de generalización lo hace menos confiable para datos no vistos. Se puede considerar como una alternativa secundaria al Gradient Boosting.

- **XGBoost Regressor**

Fortalezas	Debilidades
XGBoost es generalmente conocido por su capacidad para manejar grandes volúmenes de datos y su robustez ante sobreajuste en otros escenarios. Sin embargo, en este caso, no ha destacado frente a los otros modelos.	Tiene el peor MAE (0.01406) y MSE (0.00297), lo que indica un mayor error promedio y un manejo deficiente de errores grandes.
	Su R^2 Score (0.66561) es significativamente inferior, explicando solo el 66% de la variabilidad en los tiempos, lo que demuestra que no se ajusta bien a los datos.
	En la validación cruzada, tiene el peor desempeño (0.01840), confirmando que no generaliza bien y es menos confiable en datos nuevos.

Tabla 3-7: Tabla de fortalezas y debilidades del modelo XGBoost Regressor
Fuente: Elaboración Propia, 2024

Conclusión:
Este modelo no es adecuado para este problema específico. Se podría explorar un ajuste más fino de los hiper parámetros si se desea utilizar, pero su rendimiento actual sugiere que no es la mejor opción.

3.2.7.5. Selección del Modelo

Como resultado del proceso de entrenamiento y evaluación de los modelos, se concluye lo siguiente:

- **Primera Opción:** Gradient Boosting Regressor por su equilibrio entre precisión y capacidad de generalización.
- **Segunda Opción:** Random Forest Regressor si se prioriza la precisión en los datos de entrenamiento sobre la generalización.
- **No recomendado:** XGBoost Regressor, debido a su menor rendimiento en todas las métricas evaluadas.

3.2.8. Herramientas utilizadas

A lo largo del desarrollo del presente proyecto, se emplearon diversas herramientas tecnológicas para llevar a cabo la recolección, procesamiento y análisis de datos, así como la implementación y evaluación de los modelos de aprendizaje automático. Estas herramientas fueron seleccionadas en función de su adecuación a los objetivos planteados, su facilidad de uso y su capacidad para manejar los datos provenientes de JIRA. A continuación, se describen las principales herramientas utilizadas:

- **Python**
    - **Propósito**: Desarrollo del pipeline de análisis y modelado.
    - **Librerías Utilizadas:**

Herramienta	Propósito	Rol en el Proyecto
Scikit-learn	Modelado y evaluación	Implementar y validar modelos predictivos.
Pandas/NumPy	Manipulación de datos	Preprocesar y transformar los datos de entrada.
Matplotlib/Seaborn	Visualización de resultados	Crear gráficos de análisis exploratorio y modelos.

Tabla 3-8: Tabla resumen de las librerías de python
Fuente: Elaboración Propia, 2024

    - **Justificación:** Python es una herramienta ampliamente utilizada en ciencia de datos debido a su flexibilidad y el ecosistema de librerías disponible. (McKinney, 2010)
- **Gradient Boosting, Random Forest y XGBoost**
    - **Propósito**: Implementación de modelos de aprendizaje automático para la predicción de tiempos de resolución.
    - **Justificación**:
        - **Gradient Boosting:** Excelente precisión en problemas de regresión tabular.
        - **Random Forest:** Robustez frente a ruido y capacidad de interpretar características.
        - **XGBoost:** Alta eficiencia computacional y manejo de valores faltantes. (Hastie, 2009)

- **Jupyter Notebooks**
  - **Propósito**: Desarrollo interactivo de scripts y análisis exploratorio de datos.
  - **Justificación**: Permite una experimentación rápida con los datos y modelos, así como la creación de reportes reproducibles. (Kluyver, 2016)
- **Validación de Modelos**
  - **Cross-validation (Scikit-learn):** Para evaluar la generalización de los modelos.
  - **Randomized Search:** Optimización de hiper parámetros.
  - **Justificación**: Estas herramientas aseguraron que los modelos seleccionados fueran robustos y optimizados para los datos del proyecto.

El uso de estas herramientas fue fundamental para garantizar un flujo de trabajo eficiente y la implementación exitosa de los modelos predictivos. Desde la extracción de datos hasta la evaluación de los modelos, cada herramienta desempeñó un papel clave en el cumplimiento de los objetivos del proyecto.

3.3. Plan de Implementación

El objetivo principal de la implementación es integrar el modelo predictivo de Gradient Boosting con herramientas de gestión de proyectos como JIRA, para estimar los tiempos de resolución de incidencias de manera automática. Esto permitirá a los equipos anticipar demoras, optimizar la planificación de sprints y tomar decisiones basadas en datos.

Esta sección describe los pasos necesarios para implementar el modelo, cubriendo desde la preparación técnica hasta la automatización del flujo y las estrategias de validación continua.

Aunque la implementación oficial no se llevó a cabo debido a restricciones de acceso, este plan detalla claramente que el modelo está técnicamente listo para ser desplegado en un entorno real, asegurando una transición fluida cuando las condiciones lo permitan.

3.3.1. Preparación del Modelo

- **Guardar el modelo entrenado:** Es fundamental serializar el modelo una vez entrenado, utilizando un formato que permita su fácil carga en entornos de producción. Para este propósito, se recomienda el uso de la librería Joblib, ya que está optimizada para almacenar objetos de gran tamaño como modelos de machine learning.

```
import joblib
joblib.dump(gbr_model, 'gradient_boosting_model.pkl')
```

Figura 3-50: Código de ejemplo para guardar el modelo seleccionado
Fuente: Elaboración Propia, 2024

- **Definir los datos de entrada esperados:** Es necesario especificar las columnas y los tipos de datos que el modelo requiere como entrada. Esto asegura que los datos proporcionados cumplan con el formato adecuado para el correcto funcionamiento del modelo. Por ejemplo, las columnas deben coincidir con las utilizadas durante el entrenamiento, y los tipos de datos (numéricos, categóricos, etc.) deben ser los mismos.

```
required_columns = ['Priority', 'Story Points', 'Assignee', 'Issue Type', 'Team']
```

Figura 3-51: Código de ejemplo para definir los datos de entrada
Fuente: Elaboración Propia, 2024

- **Crear un script para preprocesar los datos:** El modelo requiere que los datos de entrada estén pre procesados antes de realizar predicciones. Se recomienda crear un script que automatice tareas como la transformación de variables categóricas, la imputación de valores faltantes y cualquier otro paso de limpieza necesario. Alternativamente, estas tareas pueden realizarse manualmente siguiendo las instrucciones detalladas previamente en la sección de limpieza y preprocesamiento de datos.

3.3.2. Integración con JIRA

- **Conexión a JIRA mediante su API:** Para extraer datos de JIRA, se puede utilizar la librería jira (de Python), que facilita la autenticación y el acceso a la API de JIRA. Esta herramienta permite interactuar con los proyectos, incidencias y campos disponibles en la plataforma de manera eficiente.

```
from jira import JIRA

jira = JIRA(server='https://your-jira-instance.com', basic_auth=('user', 'password'))
issues = jira.search_issues('project=PROJECT_KEY AND status="In Progress"', maxResults=50)
```

Figura 3-52: Código de ejemplo para conectarse a JIRA con python
Fuente: Elaboración Propia, 2024

- **Flujo de Datos:** El proceso debe incluir la extracción de datos relevantes de las incidencias activas desde JIRA, su transformación para alinearse con los requisitos del modelo predictivo y, finalmente, el envío de los datos procesados al modelo. Este flujo garantiza la generación de estimaciones precisas para los tiempos de resolución.
- **Simulación de Actualización en JIRA:** Se puede utilizar un campo personalizado en JIRA, como Predicted Resolution Time, para almacenar las predicciones generadas por el modelo. Este campo debe configurarse previamente en JIRA para que las actualizaciones puedan realizarse de forma automática o semiautomática desde el script.

```
for issue, prediction in zip(issues, predictions):
    issue.update(fields={'customfield_time_estimation': prediction})
```

Figura 3-53: Código de ejemplo para actualizar un campo personalizado en el tablero de JIRA
Fuente: Elaboración Propia, 2024

3.3.3. Automatización del Flujo de Trabajo

- **Pipeline Automatizado:** Diseñar un script o servicio que se ejecute de manera periódica, como cada noche, para llevar a cabo todo el proceso automáticamente. Este pipeline debe incluir la extracción de datos desde JIRA, la generación de predicciones utilizando el modelo entrenado, y la actualización de los campos personalizados en JIRA con las predicciones obtenidas.
- **Implementación en la Nube o Servidor Local:** Desarrollar una API REST utilizando Flask que permita recibir datos de entrada y devolver predicciones generadas por el modelo. Esta API facilitará la integración con otros sistemas o flujos de trabajo, garantizando la accesibilidad del modelo desde diversas plataformas.

```python
from flask import Flask, request, jsonify
import joblib

app = Flask(__name__)
model = joblib.load('gradient_boosting_model.pkl')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    predictions = model.predict(pd.DataFrame(data))
    return jsonify(predictions.tolist())

app.run()
```

Figura 3-54: Código de ejemplo para construir un recurso para la actualización de las predicciones
Fuente: Elaboración Propia, 2024

- **Despliegue en la Nube:** Implementar la solución en servicios de computación en la nube, como AWS Lambda, Google Cloud Functions o Heroku. Estas plataformas ofrecen flexibilidad y escalabilidad, permitiendo el despliegue del pipeline y la API en un entorno seguro y de alta disponibilidad.

3.3.4. Visualización y Reportes

- **Dashboards**: Diseñar un tablero interactivo utilizando herramientas como **Power BI**, **Tableau** o **Grafana** para proporcionar una visión clara y detallada del rendimiento del equipo y la precisión del modelo. El tablero debe incluir visualizaciones clave como:
    - Comparaciones entre los tiempos de resolución reales y los tiempos estimados por el modelo.
    - Predicciones de posibles demoras en los sprints, resaltando incidencias que podrían impactar la entrega.
- **Gráficos Dinámicos**: Incorporar gráficos dinámicos que permitan analizar el impacto de las predicciones en tiempo real.
- **Ejemplo de Simulación Visual**: Incluir simulaciones en el proyecto para ilustrar cómo se integran las predicciones con los datos de JIRA.

3.3.5. Validación Continua y Reentrenamiento:

- **Monitoreo del Modelo:** Implementar un sistema de monitoreo continuo para evaluar el desempeño del modelo en producción. Esto incluye comparar las predicciones generadas por el modelo con los tiempos reales de resolución registrados en JIRA.
- **Reentrenamiento del Modelo:** Establecer un flujo de trabajo periódico para reentrenar el modelo cuando se detecte una disminución significativa en su precisión. Esto puede incluir:
    - Incorporar datos nuevos provenientes de JIRA para reflejar cambios en los patrones de trabajo o actualizaciones en los procesos del equipo.
    - Automatizar el reentrenamiento utilizando scripts programados que integren nuevas incidencias como parte del conjunto de datos de entrenamiento.

```
# Cargar nuevos datos
new_data = extract_data_from_jira()
# Reentrenar el modelo
gbr_model.fit(new_data['X'], new_data['y'])
joblib.dump(gbr_model, 'gradient_boosting_model_updated.pkl')
```

Figura 3-55: Código de ejemplo para el reentrenamiento del modelo
Fuente: Elaboración Propia, 2024

4. Resultados y Discusión

Este capítulo presenta los resultados del análisis y la evaluación de modelos, organizados según los objetivos planteados al inicio del proyecto.

4.1. Análisis y Procesamiento de Datos

El conjunto de datos extraído desde la plataforma JIRA pasó por una serie de procesos rigurosos que incluyen transformaciones y relevamiento de información clave para garantizar su calidad y utilidad en el entrenamiento del modelo predictivo.

- **Extracción de Datos:** Antes de la extracción, se evaluó la relevancia de las variables para priorizar aquellas con mayor impacto en la predicción, como Descripción de la Incidencia, Persona Final Asignada, Puntos de Historia, Información del Sprint, Epic Padre y Fechas de Transición entre Estados, entre otras. Detalles adicionales se presentan en las tablas 3-1, 3-2 y 3-3 del capítulo anterior.

  Durante este proceso, un equipo designado de responsables evaluó cuidadosamente las variables seleccionadas para garantizar que la información incluida cumpliera con los lineamientos del proyecto. Se aseguró que ninguna variable contuviera datos sensibles o confidenciales que excedieran los límites permitidos, respetando así las políticas internas de seguridad de datos y los estándares éticos establecidos.

- **Transformaciones y Procesamiento:** El conjunto de datos extraído fue sometido a diversas etapas de transformación para hacerlo adecuado tanto para el análisis como para el entrenamiento del modelo, incluyendo la Limpieza de Datos, el Enriquecimiento del Dataset y la Codificación y Estandarización. Los detalles específicos de estos procesos se encuentran en las secciones 3.2.2 (Limpieza y Preprocesamiento de Datos), 3.2.4 (Ingeniería de Características) y 3.2.5 (Preparación del conjunto de datos para el modelado) del capítulo anterior.
- **Análisis de Variables:** El conjunto de datos final incluyó únicamente las variables que cumplían con los criterios establecidos durante el relevamiento inicial y las verificaciones de confidencialidad. Este enfoque garantizó que el modelo se entrenara con información relevante y segura.

Como resultado de estos procesos, se obtuvo un conjunto de datos completamente preparado para el análisis exploratorio y el entrenamiento del modelo. Inicialmente, el dataset contaba con 1429 filas y 200 columnas, las cuales se depuraron y transformaron para garantizar la relevancia y calidad de los datos. Después de las etapas de limpieza y selección, el dataset se redujo a 564 filas y 19 columnas, sin incluir aún los procesos de codificación. Finalmente, tras la codificación de variables categóricas y otras transformaciones, el conjunto quedó listo para el entrenamiento, con un total de 564 filas y 263 columnas (Figura 4-1), optimizado para los algoritmos de aprendizaje automático.

Key	Resolution Time	Blocked	Member Efficiency per ticket	Sprint Overdue	Days to Sprint End	Total Story Points per Sprint and team	Total Story Points per Member per Sprint	Average Team Efficiency per Sprint	Block Rate by Issue Type	...	Sprint_Woof Sprint 5	Team_Team Buzz	Team_Team Howl	Team_Team Meow	Team_Team Panda	Team_Team Quack	Team_Team Roar	Team_Team Woof	Complexity_Low	Complexity_Medium
NGP-1035	0.167349	1	0.000033	1.0	0.818182	0.387097	0.333333	0.006379	0.500000	...	0	0	0	0	0	0	0	0	0	1
NGP-1036	0.158573	0	0.000055	1.0	0.818182	0.473118	0.454545	0.492863	0.000000	...	0	0	0	0	0	0	0	0	0	0
NGP-1037	0.070327	0	0.000124	1.0	0.818182	0.473118	0.454545	0.492863	0.000000	...	0	0	0	0	0	0	0	0	0	0
NGP-1042	0.103088	1	0.000053	0.0	0.909091	0.301075	0.181818	0.006957	0.333333	...	0	0	0	0	0	0	0	0	0	1
NGP-1043	0.154544	1	0.000035	0.0	0.909091	0.967742	1.000000	0.648653	0.444444	...	0	0	0	0	0	0	0	0	0	1

Figura 4-1: Conjunto de datos final procesado
Fuente: Elaboración Propia, 2024

4.2. Implementación y Evaluación de Modelos

En el capítulo tres se llevó a cabo la implementación de tres modelos de regresión: **Random Forest Regressor, Gradient Boosting Regressor y XGBoost Regressor**. Cada modelo fue entrenado y evaluado siguiendo un enfoque consistente para garantizar la comparabilidad de sus resultados.

El proceso de evaluación incluyó métricas clave como el **Mean Absolute Error (MAE), Mean Squared Error (MSE) y el Coeficiente de Determinación (R^2)**, permitiendo analizar el desempeño de cada modelo en términos de precisión y robustez. Tras una comparación exhaustiva, se concluyó que el **Gradient Boosting Regressor presentó el mejor rendimiento**, destacándose por su capacidad para capturar patrones complejos en los datos y minimizar los errores de predicción.

Este modelo fue seleccionado como el más adecuado para el problema planteado debido a su equilibrio entre precisión y eficiencia computacional, lo que lo hace ideal para ser integrado en el flujo de trabajo del equipo.

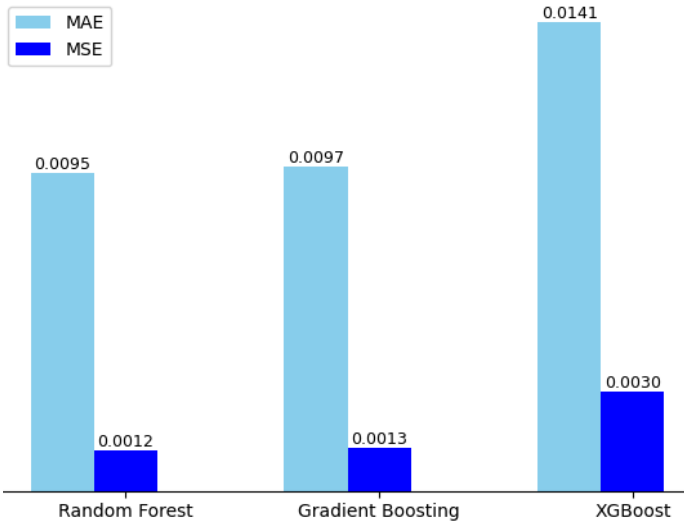

Figura 4-2: Gráfico comparativo del MAE y el MSE de los tres modelos evaluados
Fuente: Elaboración Propia, 2024

En la Figura 4-2, se observa que, aunque las diferencias entre las métricas de desempeño de Random Forest y Gradient Boosting son insignificantes, Random Forest presenta valores ligeramente mejores, posicionándose como el modelo con las mejores métricas. Ambos modelos demuestran ser excelentes candidatos para la tarea planteada, mientras que XGBoost queda significativamente rezagado debido a sus métricas de error considerablemente más elevadas.

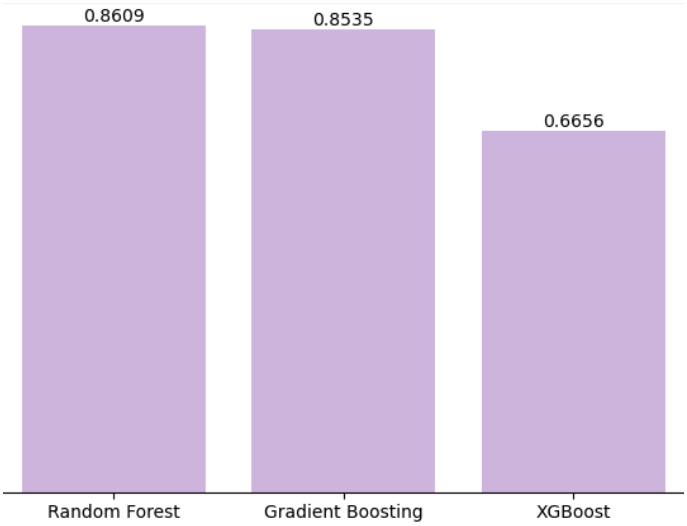

Figura 4-3: Gráfico comparativo del coeficiente de determinación de los tres modelos evaluados
Fuente: Elaboración Propia, 2024

En la Figura 4-3, donde se compara el coeficiente de determinación (R^2) entre los tres modelos, se aprecia que Random Forest sigue siendo superior al explicar el 86% de la variabilidad de los datos, ligeramente por encima del 85% alcanzado por Gradient Boosting. Aunque la diferencia entre estos dos modelos es mínima, ambos se mantienen como los mejores candidatos para la predicción de tiempos de resolución de incidencias, destacándose por su capacidad predictiva. En contraste, XGBoost queda significativamente rezagado, con un 67%, un porcentaje que no resulta óptimo para el problema planteado.

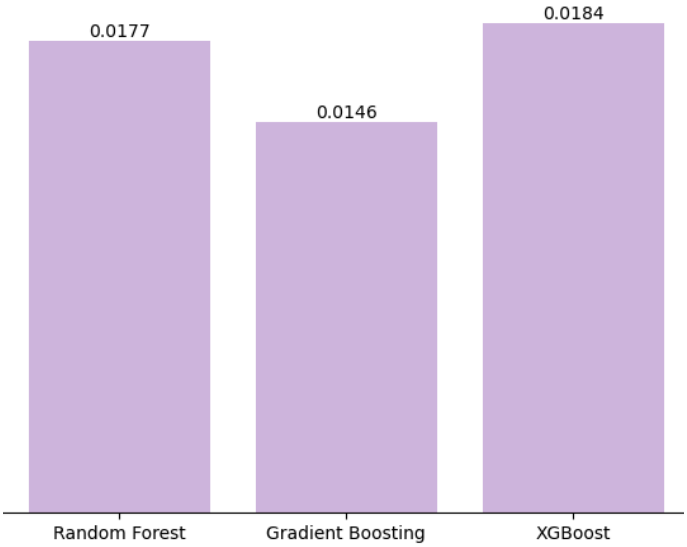

Figura 4-4: Gráfico comparativo de la validación cruzada de los tres modelos evaluados
Fuente: Elaboración Propia, 2024

Finalmente, la Figura 4-4 que compara los tres modelos en términos de validación cruzada, donde Gradient Boosting supera a Random Forest, esta vez con una diferencia significativa. Gradient Boosting presenta un error de 0.0146, menor al 0.0177 de Random Forest, lo que reafirma su consistencia en diferentes subconjuntos de datos. Por otro lado, XGBoost muestra el mayor error (0.0184), quedando nuevamente como la opción menos viable.

Esta métrica es particularmente importante porque evalúa la capacidad del modelo para generalizar y desempeñarse bien con datos nuevos. En este caso, el resultado confirma que Gradient Boosting no solo es preciso, sino también robusto y confiable, lo que lo posiciona como la mejor opción para resolver el problema planteado.

4.3. Importancia de las características

Aunque en el capítulo tres y durante los procesos descritos se generaron nuevas variables y se priorizaron algunas en función de su relevancia para el entrenamiento y optimización del modelo, es fundamental identificar las variables que tuvieron mayor influencia en la predicción de los tiempos de resolución. A continuación, se presentan las variables más influyentes y determinantes para el modelo Gradient Boosting Regressor, destacando aquellas que deben priorizarse para mejorar la calidad de las predicciones.

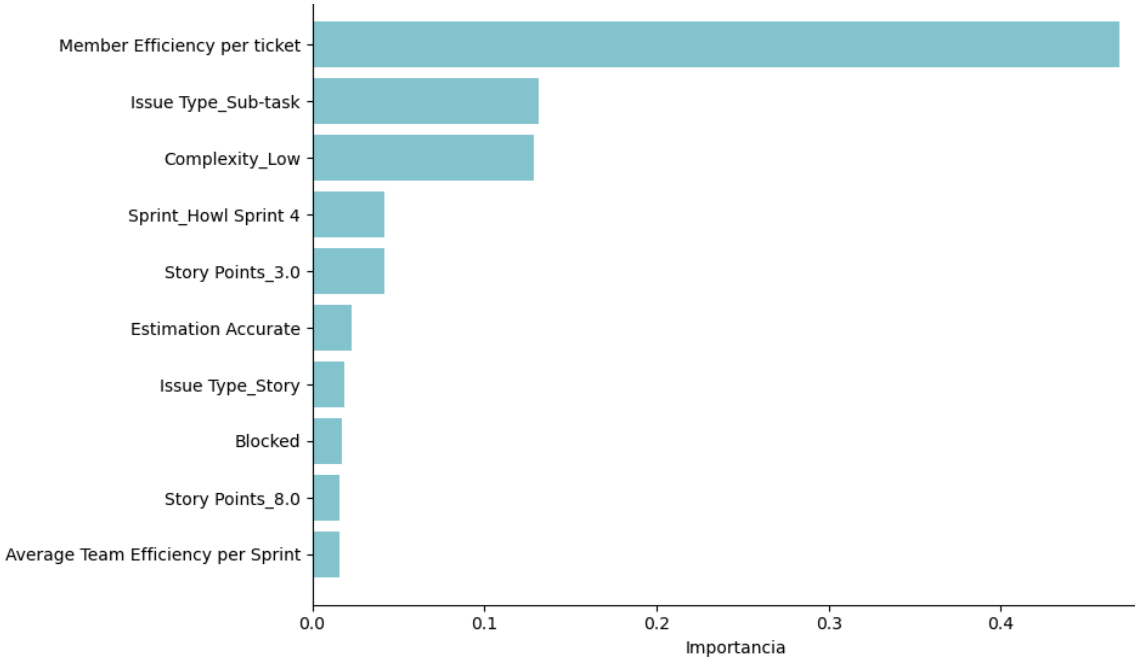

Figura 4-5: Top 10 de las variables más importantes del modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

La Figura 4-5 presenta el ranking de las 10 variables más influyentes para el modelo **Gradient Boosting Regressor**, destacando aquellas que tuvieron mayor impacto en la predicción de los tiempos de resolución. Según este análisis:

1. **Eficiencia del desarrollador o miembro del equipo**: Es la variable más importante, lo cual tiene sentido, ya que la finalización de una incidencia depende, en gran medida, de la persona responsable de trabajar en ella. Este resultado refuerza la relevancia del desempeño individual en el proceso.
2. **Tipo de incidencia**: Ocupa el segundo lugar, debido a que categoriza las incidencias en función de su complejidad. Por ejemplo, las subtareas suelen ser más simples que una historia de usuario y tienden a tener asignaciones de **Story Points** más bajas, lo que las hace más rápidas de completar.
3. **Complejidad de la incidencia**: En tercer lugar, esta variable también tiene una lógica clara, ya que las estimaciones realizadas por el equipo generalmente consideran qué tan compleja o prioritaria es una tarea. El modelo refleja este comportamiento al darle una alta relevancia.

4. **Sprint**: Aunque ocupa el cuarto lugar, su influencia podría deberse a patrones detectados por el modelo. En un contexto real, la duración del sprint podría ser un factor más relevante que el sprint en sí mismo.
5. **Story Points**: Como quinta variable más influyente, su importancia radica en que representa una estimación manual realizada por el equipo. El modelo la considera significativa, lo que sugiere que, en cierto porcentaje, las predicciones del equipo han sido acertadas.

Las demás variables del top 10 tienen una menor influencia en comparación, pero podrían estar relacionadas con patrones específicos detectados por el modelo.

Conclusión

El ranking de variables identificado por el modelo **Gradient Boosting Regressor** es coherente y justificado. Las variables más influyentes, como la eficiencia del desarrollador, el tipo de incidencia y la complejidad, reflejan factores clave en la predicción de tiempos de resolución. Esto refuerza la validez del modelo seleccionado como la mejor opción para abordar el problema planteado.

4.4. Reducción del MAE

El objetivo principal del proyecto era desarrollar un modelo predictivo que lograra reducir el Error Absoluto Medio (MAE) en al menos un 50% respecto a las estimaciones manuales realizadas por el equipo utilizando Story Points. A continuación, se presentan los resultados de la comparación entre el MAE base (estimaciones manuales) y el MAE del modelo predictivo seleccionado (Gradient Boosting).

- **MAE Base: Estimaciones Manuales**

  El MAE base, calculado a partir de las estimaciones manuales de tiempos de resolución basadas en los puntos de historia estimados por los equipos (Tabla 4-1), fue de 6 días (Figura 4-6). Este valor refleja el error promedio que los equipos enfrentan al predecir los tiempos de resolución utilizando únicamente las estimaciones manuales. Dicho error es considerablemente alto, lo que puede ocasionar problemas como:

  - Retrasos frecuentes en la finalización de los sprints.
  - Ineficiencias en la asignación de recursos.
  - Incapacidad de cumplir con los plazos establecidos.

```
estimated_time = filtered_data['Story Points'].apply(estimate_resolution_time_from_story_points)
actual_time = filtered_data['Resolution Time']

# Calcular MAE base
mae_base = mean_absolute_error(actual_time, estimated_time)
print(f"MAE base (estimaciones manuales): {mae_base}")
```

```
MAE base (estimaciones manuales): 6.319475603314057
```

Figura 4-6 : Código para calcular el MAE base de las estimaciones manuales
Fuente: Elaboración Propia, 2024

Puntos de historia	Esfuerzo (Tiempo en días)
1	<= 0.5

2	<= 1.5
3	<= 2.5
5	<= 4
8	<= 5
13	<= 7

Tabla 4-1: Tabla de puntos de historia y estimación de esfuerzo
Fuente: Elaboración Propia, 2024

- **Rendimiento del Modelo Predictivo**

  El modelo seleccionado, Gradient Boosting, logró un MAE de **0.00971** días, lo que representa una reducción significativa del error promedio. En términos porcentuales, el modelo logró una reducción del **99.85%** del MAE en comparación con las estimaciones manuales, superando ampliamente el objetivo planteado.

```
mae_model = 0.00971  # MAE del modelo
improvement = ((mae_base - mae_model) / mae_base) * 100
print(f"Reducción del MAE: {improvement:.2f}%")
```

```
Reducción del MAE: 99.85%
```

Figura 4-7: Código para calcular la reducción del MAE
Fuente: Elaboración Propia, 2024

La reducción del 99.85% del MAE demuestra que el modelo predictivo no solo cumple con el objetivo del proyecto, sino que también establece un estándar considerablemente más preciso para la estimación de tiempos de resolución en entornos ágiles. Este nivel de precisión tiene los siguientes impactos potenciales:

- **Planificación de Sprints:** Los Product Owners y equipos de desarrollo pueden anticipar mejor los tiempos requeridos para completar tareas, ajustando los objetivos del sprint de manera más realista.
- **Optimización de Recursos:** Con predicciones precisas, es posible asignar recursos de manera más eficiente, priorizando tareas críticas.
- **Reducción de Bloqueos y Retrasos:** Un error promedio menor implica una menor probabilidad de que los plazos proyectados se vean afectados.

Los resultados obtenidos confirman que el modelo predictivo desarrollado supera ampliamente las estimaciones manuales basadas en Story Points. Con un MAE de 0.00971, el modelo demuestra su capacidad para mejorar significativamente la precisión de las predicciones, ofreciendo una herramienta confiable para optimizar la gestión de incidencias en proyectos ágiles.

4.5. Predicción del Tiempo de Resolución de Incidencias

El modelo Gradient Boosting Regressor generó predicciones para los tiempos de resolución de las incidencias utilizando las variables clave identificadas en el análisis. En la Figura 4-8, se evidencia cómo las predicciones del modelo (para un subconjunto de 20 incidencias) se alinean estrechamente con los valores reales, demostrando la precisión del modelo en la tarea de predicción.

Valores Reales	Valores Predichos	Diferencia Absoluta	Porcentaje de Acierto
0.142261	0.142237	0.000024	99.983093
0.066086	0.066168	0.000082	99.876085
0.076285	0.076383	0.000098	99.871697
0.012514	0.012534	0.000020	99.839988
0.068491	0.068374	0.000117	99.828877
0.136092	0.135801	0.000291	99.786252
0.011327	0.011357	0.000030	99.731662
0.130749	0.130353	0.000397	99.696602
0.087948	0.087506	0.000442	99.497317
0.068757	0.068371	0.000386	99.439031
0.068939	0.068549	0.000390	99.434646
0.044042	0.044333	0.000291	99.339013
0.019469	0.019338	0.000131	99.327166
0.105240	0.106025	0.000785	99.253699
0.108737	0.107861	0.000877	99.193805
0.058571	0.057941	0.000630	98.923827
0.126459	0.125081	0.001378	98.910479
0.066557	0.067372	0.000815	98.775930
0.089288	0.088133	0.001154	98.707381
0.064443	0.065277	0.000834	98.706221

Figura 4-8: Predicciones del Tiempo de Resolución de Incidencias hechas por el modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

La figura 4-9 muestra la relación entre las **predicciones del modelo de Gradient Boosting** y los **valores reales de resolución de tiempo**. Cada punto en el gráfico representa una incidencia, donde el eje X indica el tiempo real tomado para resolverla, y el eje Y indica el tiempo estimado por el modelo. La línea diagonal negra representa una predicción perfecta, es decir, cuando los valores predichos coinciden exactamente con los valores reales.

Observaciones clave:

1. **Alineación cercana a la línea diagonal**: La mayoría de los puntos están muy cerca de la línea diagonal, lo que indica que el modelo generó predicciones altamente precisas para la mayoría de las incidencias.
2. **Distribución general**: Los puntos están distribuidos de manera uniforme a lo largo de la línea diagonal, lo que sugiere que el modelo mantiene una buena precisión en todo el rango de valores.
3. **Puntos fuera de la línea**: Aunque existen algunos puntos alejados de la línea (outliers), representan una proporción muy pequeña del total, lo que indica que el modelo tiene un bajo error en general.

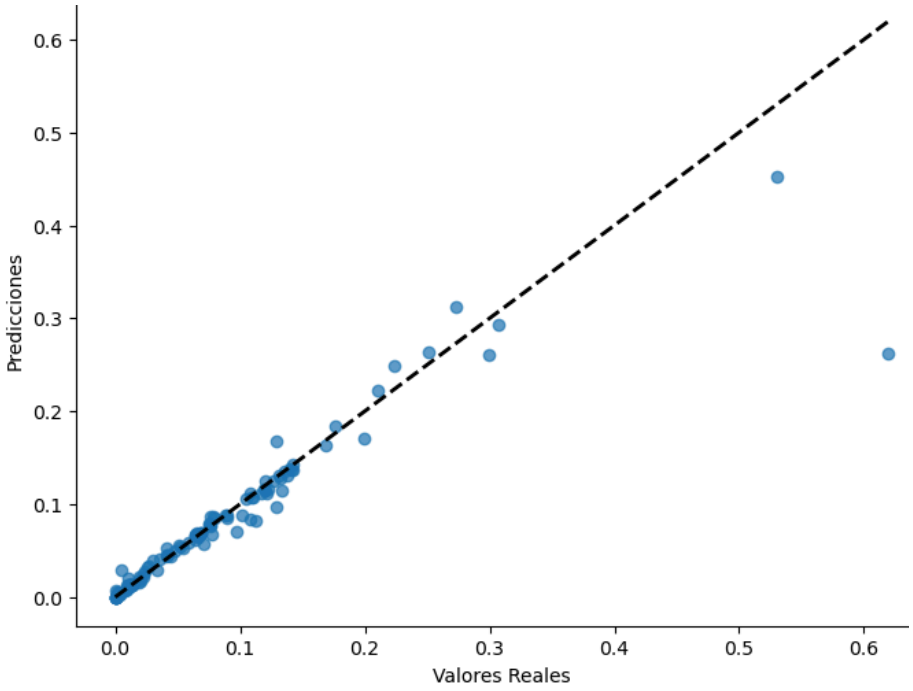

Figura 4-9: Relación entre Predicciones del Modelo y Valores Reales de Tiempo de Resolución
Fuente: Elaboración Propia, 2024

Este gráfico refuerza la efectividad del modelo al predecir los tiempos de resolución, demostrando una alta correlación entre las predicciones y los valores reales. Este comportamiento consistente es respaldado por métricas como el MAE de 0.00971, lo que lo convierte en una herramienta confiable para optimizar la gestión de incidencias.

4.6. Diseño del plan de implementación

En el capítulo 3, sección 3.3 (Plan de Implementación), se detallan los pasos necesarios para una futura integración del modelo en el tablero de JIRA, incluyendo su despliegue y las herramientas más adecuadas para lograrlo. Sin embargo, como se ha mencionado en diversas partes del proyecto, este trabajo se centra únicamente en la implementación del modelo, lo que abarca el análisis de su rendimiento, el ajuste de hiper parámetros y otras optimizaciones.

Aunque se tenía planificada su implementación práctica, esto no será posible debido a restricciones por parte de la empresa que proporcionó los datos, la cual limita el uso del modelo fuera de su control y supervisión. Adicionalmente, la integración en el tablero, propiedad de un cliente tercero, requiere negociaciones directas y procedimientos adicionales que no están contemplados en este proyecto. Estas limitaciones, junto con los costos asociados al despliegue, impiden probar el modelo en entornos reales por el momento.

No obstante, el diseño del plan de implementación proporciona una guía clara y sencilla que ofrece una visión general de cómo podría llevarse a cabo en un futuro, siempre que se superen las restricciones actuales.

4.7. Discusión

Los resultados obtenidos en este proyecto no solo cumplen con los objetivos planteados, sino que también se alinean con estudios previos en los que se han utilizado modelos de aprendizaje automático para la

predicción en contextos similares. Aunque este trabajo se centra en equipos y proyecto específicos, la metodología aplicada y los resultados alcanzados pueden compararse con investigaciones que han evaluado la eficacia de modelos como Gradient Boosting, Random Forest y XGBoost.

4.7.1. Resultados del Modelo

El modelo de Gradient Boosting desarrollado en este proyecto alcanzó un MAE de 0.00971 días, logrando una reducción del 99.85% respecto al MAE base obtenido a partir de las estimaciones manuales (Story Points), que era de 6 días. Este nivel de precisión destaca la capacidad del modelo para superar significativamente las limitaciones de las estimaciones manuales, posicionándose como una herramienta valiosa para optimizar la gestión de incidencias en proyectos ágiles.

4.7.2. Comparación con Proyectos Similares

Aunque no es posible comparar este proyecto directamente con otro de contexto y objetivos similares, ni siquiera utilizando los mismos modelos, debido a la especificidad de los datos, equipos y el proyecto en sí, sí es factible realizar comparaciones a nivel de resultados con otros proyectos que hayan empleado los mismos modelos aplicados en este estudio.

Fatima et al. (2023): Comparación de XGBoost y Random Forest

En el artículo **"XGBoost and Random Forest Algorithms: An In-Depth Analysis"**, Fatima et al. evaluaron la eficacia de XGBoost y Random Forest en problemas de regresión. Los resultados destacaron:

- **XGBoost** superó a Random Forest en términos de menor MSE y mayor R^2, mostrando una capacidad superior para capturar correlaciones complejas en los datos.
- **Random Forest**, aunque robusto, presentó un menor rendimiento general debido a su incapacidad para modelar relaciones intrincadas con la misma eficacia.

TABULAR COMPARISON OF REGRESSION MODELS

Models	MEAN SQUARED ERROR	R-squared
XGBoost	0.29	0.77
Random Forest	0.60	0.54

Figura 4-10: Tabla de comparación de los modelos XGBoost y Random Forest
Fuente: Fatima et al., 2023

Comparación con el Proyecto Actual:

- Al igual que en el estudio de Fatima et al., este proyecto encontró que los modelos de boosting, como Gradient Boosting, ofrecen ventajas significativas sobre Random Forest en tareas que requieren alta precisión y la capacidad de manejar datos complejos.
- Sin embargo, a diferencia de Fatima et al., este trabajo se centró en Gradient Boosting como modelo final debido a su menor MAE y mejor consistencia en validación cruzada frente a XGBoost y Random Forest. Esto subraya la importancia de seleccionar el modelo adecuado según las características específicas del problema.

Nadkarni et al. (2023): Predicción del Ruido en Perfiles Alares

El estudio **"Comparative Study of Random Forest and Gradient Boosting Algorithms to Predict Airfoil Self-Noise"** de Nadkarni et al. comparó Random Forest y Gradient Boosting en un problema complejo de predicción del ruido en perfiles alares. Los hallazgos clave incluyen:

- **Gradient Boosting** destacó por su menor tiempo de entrenamiento y su capacidad para reducir la incertidumbre en las predicciones mediante el promedio de múltiples aprendices débiles.
- **Random Forest**, en cambio, alcanzó el mayor R^2, demostrando una capacidad superior para explicar la varianza en los datos, aunque a costa de un tiempo de entrenamiento más prolongado.

ML Model	MSE	R^2	MAPE	SD	Training Time (s)
Gradient Boost	6.5337	0.8620	0.0159	5.8407	0.1250
XGBoost	3.0026	0.9365	0.0095	6.4960	0.1258
LightGB	3.4792	0.9265	0.0109	6.4331	0.1366
CatBoost	2.6331	0.9443	0.0092	6.4633	1.5546
RF	3.3765	0.9287	0.0107	6.1658	0.5082
Extra Trees	2.4631	0.9479	0.0090	6.3133	0.3750

Figura 4-11: Tabla comparativa; XGBoost, Gradient Boost, LightCB, CatBoost y Random Forest
Fuente: Nadkarni et al., 2023

Comparación con el Proyecto Actual:

- Los hallazgos de Nadkarni et al. refuerzan los resultados de este proyecto, donde Gradient Boosting mostró ventajas en precisión y consistencia frente a Random Forest.
- Sin embargo, la observación de que Random Forest puede superar a Gradient Boosting en R^2 sugiere que este modelo podría ser una opción viable en escenarios con mayor variabilidad en los datos. En este proyecto, el enfoque específico en tiempos de resolución dentro de un equipo ágil favoreció a Gradient Boosting debido a la estructura controlada del conjunto de datos.

4.7.3. Impacto y Conexión Práctica

La comparación con los estudios de Fatima et al. (2023) y Nadkarni et al. (2023) refuerza la elección de Gradient Boosting como modelo final en este proyecto. Tanto en este trabajo como en las investigaciones mencionadas, los modelos de boosting demostraron ser superiores en:

- Captura de Patrones Complejos: Su capacidad para modelar relaciones no lineales y manejar datos tabulares es clave en tareas de predicción.
- Flexibilidad y Precisión: Los modelos de boosting son consistentes y eficaces cuando se optimizan adecuadamente, lo que los hace ideales para aplicaciones prácticas como la gestión ágil de incidencias.

Por otro lado, ambos estudios coinciden en que Random Forest, aunque menos preciso en algunos casos, sigue siendo una alternativa sólida en escenarios donde la generalización y la estabilidad son factores clave. En este proyecto se evaluó esa posibilidad y, aunque Random Forest destacó como el mejor modelo frente a Gradient Boosting con diferencias mínimas e insignificantes, la capacidad de generalización resultó decisiva. Si bien Random Forest era el candidato más prometedor según este análisis y otros estudios comparativos, en este contexto y con los datos específicos utilizados, Gradient Boosting demostró un desempeño superior en términos de generalización, justificando así su selección como modelo final.

4.7.4. Impacto Potencial en el Entorno Real

La implementación de un modelo predictivo para estimar tiempos de resolución tiene el potencial de transformar la gestión de incidencias en proyectos ágiles. Al proporcionar predicciones más precisas, los equipos pueden:

- **Mejorar la planificación de sprints:** Las estimaciones más precisas permiten establecer objetivos realistas, reduciendo la presión sobre el equipo y mejorando la entrega de valor.
- **Optimizar la asignación de recursos:** Identificar tareas que requieren más tiempo permite distribuir mejor la carga de trabajo y evitar cuellos de botella.
- **Anticipar bloqueos:** Las predicciones pueden alertar sobre tareas críticas que podrían retrasar el sprint, permitiendo a los Product Owners tomar medidas correctivas de forma proactiva.

Además, el uso de un enfoque basado en datos reduce la dependencia de estimaciones subjetivas, lo que puede minimizar sesgos y errores en la planificación.

4.7.5. Limitaciones Identificadas

Aunque los resultados del modelo son prometedores, se identificaron ciertas limitaciones que podrían influir en su desempeño:

- **Dependencia de datos históricos consistentes:** El modelo requiere datos completos y de alta calidad provenientes de JIRA. Incidencias con datos incompletos o inconsistentes pueden reducir su precisión.
- **Generalización limitada:** El modelo fue entrenado con datos específicos de los equipos bolivianos y contexto. Su aplicabilidad a otros equipos o proyectos podría requerir ajustes y reentrenamiento.
- **Outliers no explicados:** Algunos valores alejados de la línea diagonal en la Figura 4-9 sugieren que hay casos en los que el modelo no logra capturar completamente la variabilidad de los datos.
- **Falta de implementación real:** Aunque se diseñó un plan de implementación, no fue posible integrarlo en un entorno real, lo que limita la evaluación de su impacto práctico.

Este proyecto, junto con los estudios de Fatima et al. (2023) y Nadkarni et al. (2023), demuestra cómo los modelos de boosting, como Gradient Boosting, pueden superar a Random Forest en tareas de regresión. Los resultados obtenidos destacan la capacidad de Gradient Boosting para manejar relaciones complejas en los datos, consolidándose como una herramienta eficaz y confiable en el contexto de la gestión ágil. Esto refuerza la importancia de ajustar y seleccionar modelos según las necesidades específicas de cada problema.

5. Conclusiones

Este proyecto se ha centrado en la implementación y evaluación de modelos de aprendizaje automático para predecir los tiempos de resolución de incidencias/tickets en un entorno de desarrollo de software, utilizando datos históricos extraídos de la plataforma JIRA. Los modelos empleados fueron Gradient Boosting, Random Forest y XGBoost, y su desempeño fue evaluado a través de métricas estándar como el MAE, MSE, R^2 y validación cruzada, con el objetivo de identificar el modelo más adecuado para el tipo de datos y para cumplir con los objetivos planteados.

A lo largo del análisis, se observó que el modelo de Gradient Boosting se destacó como el mejor, logrando una reducción significativa en el MAE en comparación con las estimaciones manuales, alcanzando una mejora de hasta el 99.85%. Esto demuestra la efectividad de este algoritmo en la captura de las complejas relaciones entre las variables involucradas en la resolución de incidencias, lo que resulta en una mayor precisión de las predicciones.

En comparación con otros trabajos similares, como el análisis realizado por Fatima et al. (2023) sobre XGBoost y Random Forest, o el estudio de Shantaram et al. (2021) sobre la predicción del ruido en perfiles alares, los resultados obtenidos en este proyecto son consistentes con la literatura existente. A pesar de las diferencias en los dominios y los conjuntos de datos utilizados, la superioridad de Gradient Boosting en términos de precisión y reducción de error en comparación con Random Forest es una tendencia comúnmente observada en estudios previos. Este hallazgo resalta la robustez del enfoque de Gradient Boosting para la predicción en contextos de regresión y el valor de su capacidad para modelar relaciones complejas.

El proyecto también resalta la importancia de la optimización de los hiper parámetros para mejorar la precisión de los modelos, ya que el ajuste fino de parámetros fue fundamental para maximizar el desempeño de Gradient Boosting. Este aspecto es crucial para cualquier implementación práctica de modelos predictivos, especialmente en entornos dinámicos como el desarrollo de software, donde los datos están sujetos a cambios constantes.

Como parte del plan de implementación, se diseñaron simulaciones para integrar el modelo seleccionado en el flujo de trabajo de JIRA, permitiendo que los equipos de desarrollo puedan aprovechar las predicciones para mejorar la planificación de sus sprints. Esto abre la puerta a futuras investigaciones sobre la integración de modelos predictivos en herramientas de gestión de proyectos, un área que podría beneficiar enormemente a las empresas al reducir la incertidumbre en la estimación de tiempos y esfuerzos.

En conclusión, este proyecto no solo ha cumplido con los objetivos iniciales de análisis y predicción de tiempos de resolución de incidencias, sino que también ha demostrado que el aprendizaje automático, y en particular el modelo de Gradient Boosting, es una herramienta poderosa y eficiente para abordar problemas de predicción en entornos reales. Este trabajo sienta las bases para la implementación de sistemas predictivos avanzados en la gestión de proyectos de software, abriendo nuevas posibilidades para la optimización de procesos y la mejora continua.

6. Recomendaciones

Para futuros desarrollos e investigaciones, se proponen las siguientes recomendaciones:

- **Ampliar el dataset:** Incorporar datos de múltiples equipos y proyectos para mejorar la robustez y la capacidad de generalización del modelo.
- **Integración de variables adicionales:** Analizar otras características potencialmente relevantes, como el historial de bloqueos o el tamaño del equipo, lo que podría mejorar aún más la precisión de las predicciones.
- **Exploración de modelos adicionales:** Si bien Gradient Boosting fue el modelo que mostró mejor rendimiento en este estudio, se recomienda explorar otros enfoques avanzados, como Redes Neuronales o Ensamblajes de Modelos, que podrían captar patrones aún más complejos en los datos.
- **Optimización continua del modelo:** Aunque Gradient Boosting ha demostrado un buen rendimiento, se recomienda seguir optimizándolo y recalibrándolo periódicamente con nuevos datos para mantener su precisión y adaptabilidad a cambios en el equipo y el proyecto.
- **Evaluación de estimaciones manuales:** Aunque el modelo automatizado es más preciso, es clave seguir mejorando las estimaciones manuales mediante capacitación y retroalimentación, para reforzar la confianza en las predicciones automatizadas.
- **Implementación en tiempo real:** Se sugiere integrar el modelo predictivo en plataformas como JIRA para ofrecer estimaciones dinámicas durante el sprint, lo que mejoraría la asignación de tareas y permitiría ajustes en tiempo real para optimizar la eficiencia operativa.
- **Automatización del reentrenamiento:** Implementar un pipeline que permita actualizar el modelo con datos nuevos, asegurando que su rendimiento se mantenga en entornos dinámicos.
- **Ampliación a otros contextos y dominios:** Aunque este estudio se centró en el equipo de desarrollo de software, los enfoques y modelos pueden aplicarse a otros sectores, como marketing, recursos humanos o manufactura, para mejorar la planificación y optimizar la toma de decisiones.

Estas recomendaciones tienen el potencial de optimizar diversas etapas del proceso, desde el entrenamiento del modelo hasta su implementación práctica. Mejorar la precisión del modelo y la calidad de los datos permitirá obtener predicciones más fiables, lo que resultará en una mayor eficiencia en la toma de decisiones. La optimización de la ingeniería de características también facilitará la extracción de información más relevante, mejorando aún más el rendimiento del modelo. Además, con estos ajustes, futuros investigadores podrán ampliar este trabajo, aplicándolo a diferentes contextos o sectores, y generalizar el modelo para que pueda ser útil en una amplia gama de escenarios, promoviendo su adaptabilidad y efectividad en otros ámbitos.

Referencias Bibliográficas

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile Software Development. Agile Alliance. Recuperado de https://agilemanifesto.org/

Miller, T. (2020). Agile Project Management with JIRA: A Practical Approach. O'Reilly Media.

Sutherland, J., & Schwaber, K. (2013). The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org. Recuperado de https://scrumguides.org/scrum-guide.html

Drury-Grogan, M. L. (2014). Performance measurement in agile development teams: A case study. International Journal of Agile Systems and Management, 7(2), 122-135. doi:10.1504/IJASM.2014.062098

Meyer, B. (2014). Agile!: The Good, the Hype and the Ugly (pp. 15-30, 130-150). Springer Science & Business Media. Disponible en: https://link.springer.com/book/10.1007/978-3-319-05155-0

Provost, F., & Fawcett, T. (2013). Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. O'Reilly Media. pp. 275-280.

What is Scrum? (2023). Scrum.org. https://www.scrum.org/resources/what-scrum-module

Atlassian. (2023). Kanban. https://www.atlassian.com/agile/kanban

Anderson, D. J. (2010). Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press.

Atlassian. (2023). What is JIRA Software? Atlassian Documentation. https://www.atlassian.com/software/jira

Rasnacis, A., & Berzisa, S. (2017). Method for Adaptation and Implementation of Agile Project Management Methodology. Procedia Computer Science, 104, 43-50. doi:10.1016/j.procs.2017.01.055

Rubin, K. S. (2012). Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley.

Cohn, M. (2006). Agile Estimating and Planning. Prentice Hall.

Atlassian. (2024). Working in an agile project | Jira Software Data Center 10.1 | Atlassian Documentation. Recuperado de https://confluence.atlassian.com/jirasoftwareserver/working-in-an-agile-project-938845521.html

Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

Han, J., Pei, J., & Kamber, M. (2011). Data Mining: Concepts and Techniques (3ra ed.). Morgan Kaufmann.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: With Applications in R. Springer.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13, 281-305.

Molnar, C. (2019). Interpretable Machine Learning. Leanpub.

Rigby, D. K., Sutherland, J., & Takeuchi, H. (2016). Embracing agile. Harvard Business Review, 94(5), 40-50. Recuperado de https://hbr.org/2016/05/embracing-agile

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.

Willmott, C. J., & Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Climate Research, 30(1), 79–82.

Atlassian. (2024). JIRA Software API Documentation. Recuperado de https://developer.atlassian.com/server/jira/platform/rest-apis/.

McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference.

Kluyver, T., et al. (2016). *Jupyter Notebooks - a publishing format for reproducible computational workflows*. In ELPUB.

Fatima, S., Hussain, A., Bin Amir, S., Ahmed, S. H., & Aslam, S. M. H. (2023). XGBoost and Random Forest Algorithms: An In-Depth Analysis. *Pakistan Journal of Scientific Research*, 3(1), 26–31. https://doi.org/10.57041/pjosr.v3i1.946.

Nadkarni, S. B., Vijay, G. S., & Kamath, R. C. (2023, 12 diciembre). Comparative study of random forest and gradient boosting algorithms to predict Airfoil Self-Noise. https://doi.org/10.3390/engproc2023059024