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

DIPLOMADO ESTADÍSTICA APLICADA A LA TOMA DE DECISIONES

TERCERA VERSIÓN

PREDICCIÓN DEL TIEMPO DE RESOLUCIÓN DE INCIDENCIAS EN UN PROYECTO DE DESARROLLO DE SOFTWARE ÁGIL USANDO APRENDIZAJE AUTOMÁTICO

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

PREDICCIÓN DEL TIEMPO DE RESOLUCIÓN DE INCIDENCIAS EN UN PROYECTO DE DESARROLLO DE SOFTWARE ÁGIL USANDO APRENDIZAJE AUTOMÁTICO

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. M.Sc. Ericka Patricia Rodríguez Bilbao. (Tribunal)

Ing. M.Sc. Valentin Laime Zapata. (Tribunal)

Ing. M.Sc. Oscar Contreras Carrasco. (Tribunal)

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

La industria del software ha adoptado metodologías ágiles para adaptarse a las demandas cambiantes del mercado, priorizando la colaboración y la entrega continua de valor. Herramientas como JIRA han facilitado la gestión de incidencias mediante tableros visuales y estructurados, permitiendo a los equipos organizar y priorizar tareas durante los sprints. Sin embargo, aunque estas herramientas son esenciales para el monitoreo en tiempo real, no son suficientes para anticipar problemas como retrasos, bloqueos o asignaciones ineficientes que afectan el rendimiento y los tiempos de entrega de los equipos.

Este proyecto propone el uso de técnicas de machine learning para analizar datos históricos de JIRA y desarrollar un modelo predictivo que permita anticipar tiempos de resolución de incidencias y optimizar el flujo de trabajo. Al ajustar estrategias proactivamente, se busca mejorar la eficiencia del equipo, reducir riesgos y asegurar entregas oportunas, contribuyendo al éxito de proyectos ágiles de desarrollo de software.

Para alcanzar el objetivo del proyecto, se implementaron tres algoritmos de aprendizaje automático: Gradient Boosting, XGBoost y Random Forest, evaluando su desempeño en tareas de regresión para predecir los tiempos de resolución de incidencias. Los datos utilizados fueron extraídos de la API de JIRA, correspondientes a un proyecto y equipos específicos, y limitados a incidencias registradas entre mayo y octubre del presente año.

El proceso metodológico incluyó las etapas de análisis y procesamiento inicial de los datos, análisis exploratorio, ingeniería de características para optimizar las variables predictoras, entrenamiento de los modelos con el conjunto de datos procesado, y evaluación para seleccionar el modelo que mejor se ajustó al caso de estudio. Los modelos fueron evaluados utilizando métricas de rendimiento como el Error Absoluto Medio (MAE), Coeficiente de Determinación (R^2) y Mean Squared Error (MSE).

Los resultados del proyecto revelaron que el algoritmo Gradient Boosting demostró ser el más eficiente para predecir los tiempos de resolución de incidencias, logrando un error de validación cruzada de 1.3329. Le siguieron Random Forest con 1.6239 y XGBoost con 1.6569. Aunque Gradient Boosting destacó por su capacidad de generalización, las métricas adicionales, como MAE, MSE y R^2, mostraron que la ventaja de Random Forest fue estadísticamente insignificante en comparación.

El modelo de Gradient Boosting logró una mejora significativa respecto a las estimaciones manuales de los desarrolladores, que tenían un error absoluto medio de 6 días. En contraste, el modelo presentó un MAE de 1.3072 días, lo que representa una reducción del error absoluto medio del 79.87%, superando ampliamente el objetivo inicial del proyecto, que era reducir el error en al menos un 50%. Este resultado subraya el impacto positivo del enfoque predictivo basado en machine learning para optimizar la planificación y la gestión de incidencias en proyectos ágiles.

El proyecto concluye que la implementación de modelos predictivos en plataformas como JIRA tiene un gran potencial para optimizar la toma de decisiones, mejorar la asignación de recursos y aumentar la exactitud de la planificación de sprints. Además, se proponen recomendaciones para futuras mejoras en el modelo, la incorporación de nuevas variables y la integración de este enfoque en tiempo real, con el objetivo de extender su aplicabilidad a otros ámbitos y equipos de trabajo.

Palabras Clave: Aprendizaje Automático, Desarrollo de Software, Gradient Boosting Regressor, Random Forest Regressor, XGBoost Regressor, Tiempo de Resolución.

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 proporcionarme el conjunto de datos y 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.

Al Lic. Rafael Rojas por su colaboración y guía durante la elaboración de este proyecto.

Al Ing. MSc. Oscar Contreras, por su desinteresado apoyo en la revisión de este proyecto, así como por sus valiosas aportaciones y observaciones.

A la Ing. MSc. Roxana Guillen por su apoyo en la culminación del proyecto y las valiosas correcciones que me brindó.

Y finalmente a la Ing. MSc. Patricia Rodríguez y al Ing. MSc. Valentín Laime por su apoyo como miembros del tribunal.

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. Utilizando un enfoque basado en machine learning, el modelo se centrará en estimar los tiempos de resolución a nivel de equipo, detectando patrones de comportamiento que optimicen el flujo de trabajo. Este análisis contribuirá a la toma de decisiones basada en datos, proporcionando una base cuantitativa para anticipar tiempos de resolución y reducir riesgos asociados con 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).

Con el crecimiento de los datos generados en herramientas como JIRA, el análisis predictivo y el machine learning se han convertido en aliados clave para optimizar el rendimiento de los equipos de desarrollo. Según Han, Pei y Kamber (2011), las técnicas de minería de datos permiten identificar patrones en los datos históricos, lo que ayuda a predecir posibles bloqueos y el tiempo de resolución de incidencias. Estas capacidades son fundamentales para fomentar la mejora continua y permiten a los equipos anticiparse a problemas potenciales, ajustando sus estrategias de forma proactiva.

Por otra parte, el auge de las metodologías ágiles en la industria del software ha puesto en primer plano el reto de lograr estimaciones precisas de tiempos y esfuerzos. Esta necesidad ha inspirado diversas investigaciones que exploran el uso de machine learning como una herramienta para optimizar la gestión y planificación en proyectos de desarrollo ágil.

En este contexto, Phan y Jannesari (2022) investigaron el uso de redes neuronales de grafos heterogéneos para estimar el esfuerzo en proyectos de software ágil. Su enfoque se basa en descripciones textuales de incidencias, demostrando que estas arquitecturas son eficaces para identificar patrones complejos y mejorar el rendimiento de las estimaciones. Este trabajo destaca el valor de emplear representaciones avanzadas de datos contextuales en la predicción de esfuerzo, aunque también implica mayores requerimientos computacionales.

Por otro lado, Sánchez, Vázquez-Santacruz y Maceda (2022) propusieron el uso de redes neuronales artificiales para estimar el esfuerzo en el desarrollo de software ágil. Su investigación mostró que estos modelos pueden superar a los métodos tradicionales en términos de rendimiento, proporcionando a los equipos de desarrollo una herramienta poderosa para planificar sus recursos y esfuerzos. Sin embargo, al igual que el enfoque anterior, este método conlleva una mayor complejidad en la implementación y el entrenamiento del modelo.

En conclusión, la integración de metodologías ágiles, herramientas como JIRA y técnicas de análisis de datos ha supuesto un avance significativo en la gestión de proyectos de desarrollo de software. Este proyecto, en particular, adopta algoritmos de regresión para predecir los tiempos de resolución de incidencias. Al basarse en datos históricos y métricas clave del flujo de trabajo, se espera que este modelo predictivo proporcione una solución práctica y eficiente para los equipos de desarrollo ágil, ofreciendo una alternativa más accesible y directa en comparación con enfoques previos más complejos.

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 ágilmente una gran variedad de incidencias, que van desde problemas técnicos hasta nuevas funcionalidades. Sin embargo, debido a la naturaleza dinámica de estos proyectos, los equipos frecuentemente se encuentran con bloqueos, cambios en las prioridades y demoras en la resolución de incidencias, lo que afecta tanto la calidad del producto final como los tiempos de entrega (Meyer, 2014).

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

A pesar de los avances en la gestión ágil, los equipos de desarrollo a menudo subestiman o sobreestiman los tiempos necesarios para resolver incidencias, particularmente en tareas complejas. Esto genera acumulación de trabajo, retrasos en la entrega de funcionalidades críticas y conflictos en la asignación de recursos (Figura 1-1). El problema central radica en la falta de un enfoque predictivo basado en datos que permita a los equipos de desarrollo prever y gestionar de manera proactiva los problemas recurrentes que afectan la eficiencia en la resolución de incidencias.

En este contexto, surge la necesidad de aplicar técnicas de aprendizaje automático para optimizar las estimaciones de tiempos de resolución de incidencias. Este proyecto propone el uso de algoritmos de

regresión para identificar patrones en datos históricos y generar predicciones más precisas sobre el tiempo que tomará la resolución de cada incidencia. Al emplear un enfoque basado en datos, se espera mejorar la planificación y la asignación de recursos, reduciendo así los retrasos y bloqueos en los proyectos de desarrollo ágil.

Figura 1-1: Arbol de problemas
Fuente: Elaboración Propia

1.4. Objetivo general

Desarrollar un modelo predictivo basado en aprendizaje automático para la estimación de 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.
- Evaluar 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. Principales Marcos de Trabajo Ágiles

Dentro de las metodologías ágiles, existen diversos marcos de trabajo que implementan sus valores y principios. Entre los más populares se encuentran Scrum y Kanban, cada uno con características únicas que se adaptan a diferentes tipos de equipos y proyectos.

Kanban

Es un marco de trabajo visual que organiza tareas en un tablero dividido en columnas representando distintas etapas del flujo de trabajo, como "Por hacer", "En progreso" y "Completado". Las tareas avanzan a través del tablero a medida que se completan, promoviendo un flujo de trabajo continuo y eficiente (Figura 2-1).

A diferencia de Scrum, Kanban no utiliza iteraciones de tiempo fijo. Las tareas se gestionan en función de la capacidad del equipo, lo que permite mayor flexibilidad ante cambios en las prioridades (Anderson, 2010).

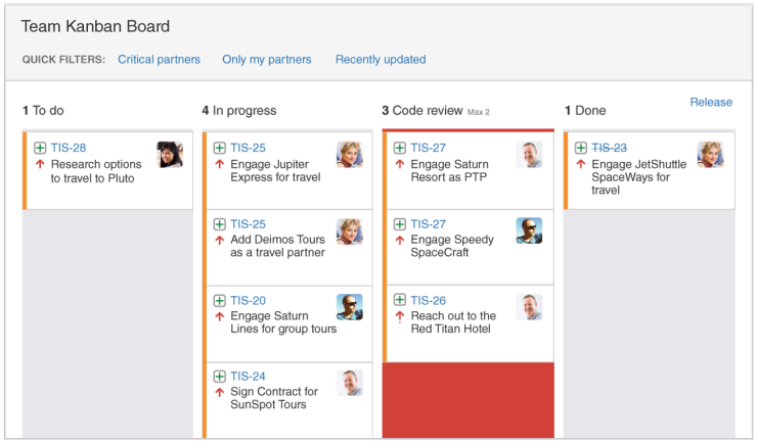


Figura 2-1: Flujo de trabajo de la metodología Kanban
Fuente: Atlassian, 2023

Kanban es especialmente útil para proyectos donde las prioridades cambian frecuentemente o donde el flujo de trabajo continuo es esencial.

Scrum

Es un marco de trabajo basado en iteraciones de tiempo fijo, conocidas como sprints (generalmente de 1 a 4 semanas), durante las cuales los equipos de desarrollo completan un conjunto de tareas previamente planificadas. Este enfoque iterativo busca maximizar la entrega de valor en ciclos cortos y predecibles.

Los roles en Scrum incluyen:

- **Scrum Master:** Facilita el proceso y elimina impedimentos.
- **Product Owner:** Prioriza y gestiona el backlog del producto.
- **Equipo de desarrollo:** Responsable de implementar las tareas planificadas.

Al final de cada sprint, se realiza una revisión para evaluar los resultados y planificar mejoras para los siguientes ciclos (Schwaber & Sutherland, 2013).

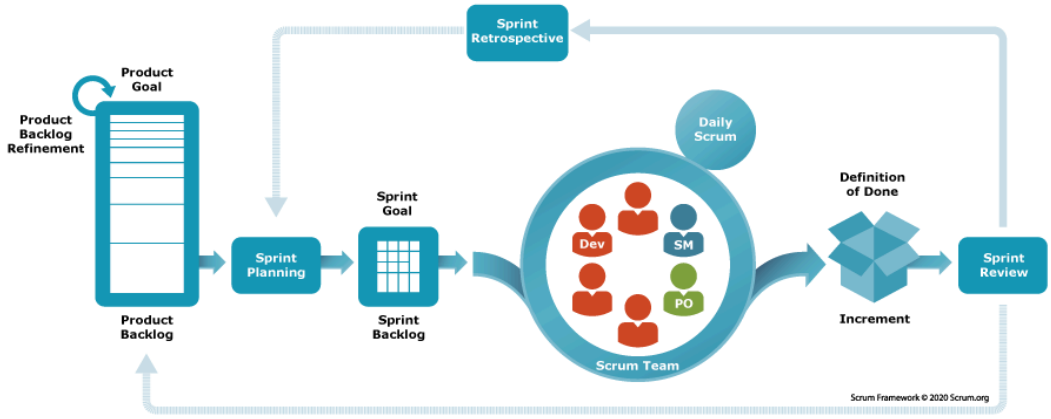


Figura 2-2: Flujo de trabajo de la metodología Scrum
Fuente: Scrum.org, 2023

La figura 2-2 ilustra los principales componentes del flujo de trabajo de Scrum, incluyendo:

Artefactos:

- **Product Backlog:** Lista priorizada de requerimientos del producto.
- **Sprint Backlog:** Tareas seleccionadas para el sprint actual.
- **Incremento:** Funcionalidad completada y lista para entregar.

Eventos:

- **Sprint Planning:** Reunión inicial para planificar las tareas del sprint.
- **Daily Scrum:** Reunión diaria de seguimiento del progreso.
- **Sprint Review:** Presentación de los avances al final del sprint.
- **Sprint Retrospective:** Evaluación del proceso para identificar áreas de mejora.

Este enfoque iterativo permite a los equipos adaptarse rápidamente a los cambios, optimizando la colaboración y la entrega continua de valor.

Desarrollo Ágil y la Adaptación a los Cambios

Ambos marcos de trabajo, Scrum y Kanban, representan enfoques iterativos y adaptativos dentro de las metodologías ágiles. Este enfoque incremental permite a los equipos:

- Incorporar retroalimentación constante.
- Reducir el riesgo de errores acumulados.
- Mejorar continuamente el producto.
- Adaptarse rápidamente a las necesidades del cliente.

El resultado es un ciclo de desarrollo ágil y eficiente, alineado con los objetivos del proyecto y las expectativas de los usuarios (Rigby, Sutherland & Takeuchi, 2016).

2.2. Gestión de Incidencias en Proyectos Ágiles

2.2.1. Incidencias y su Ciclo de Vida

En el desarrollo de software y en JIRA, una **incidencia** o **issue** representan elementos de trabajo individuales, como por ejemplo funcionalidades importantes, requisitos de usuario y errores de software. Las incidencias pueden clasificarse en distintos tipos, como:

- **Epic:** Representa una muestra de trabajo más amplia. Los epics a menudo se representan como un conjunto de varias incidencias.
- **Historia de Usuario:** Representa un requisito expresado desde la perspectiva del usuario.
- **Tareas**: Representa una tarea que debe realizarse. Las tareas se utilizan como "cajones de sastre" y cuando el trabajo no puede representarse con precisión mediante los otros tipos de incidencias.
- **Subtarea:** Representa una descomposición más detallada del trabajo necesario para completar una incidencia estándar. Se puede crear una subtarea para todos los tipos de incidencias
- **Bug o error**: Representa un problema que hay que solucionar. (Atlassian, 2024)

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. 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).

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).

2.3.3. Impacto de las Estimaciones Inexactas en la Planificación de Sprints

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.

2.3.4. 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

La analítica descriptiva es esencial en la gestión de proyectos de software, ya que permite a los equipos evaluar su rendimiento actual y obtener información detallada sobre sus patrones de trabajo. En entornos ágiles, donde la flexibilidad es clave, proporciona una visión clara de cómo se están manejando las tareas, lo que permite realizar ajustes informados y mejorar la toma de decisiones.

Mediante la analítica descriptiva, los equipos pueden identificar problemas recurrentes, como cuellos de botella o tareas que excedan los tiempos previstos. Además, es crucial para analizar el historial de sprints previos y anticipar problemas futuros, optimizando la planificación de sprints siguientes (Provost & Fawcett, 2013). Esto facilita el diseño de estrategias basadas en datos para mejorar la eficiencia y colaboración en el equipo.

Algunos beneficios pueden ser:

- **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).

2.4.1. 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

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).

Esta capacidad de aprendizaje automático se puede implementar de diferentes maneras, siendo una de las más utilizadas el aprendizaje supervisado, que permite a los modelos hacer predicciones precisas a partir de datos etiquetados y ejemplos previos.

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).

2.6. Modelos de Regresión para la Gestión de Incidencias

Los modelos de regresión son fundamentales para la estimación del esfuerzo en la gestión de incidencias, ya que permiten predecir valores numéricos continuos, como el tiempo requerido para completar una tarea o resolver una incidencia. En este contexto, se han utilizado varios modelos avanzados de aprendizaje automático basados en árboles de decisión para mejorar el error de las predicciones. Random Forest, Gradient Boosting y XGBoost son considerados algunos de los algoritmos de mejor rendimiento en una amplia variedad de conjuntos de datos, y han demostrado ser eficaces en la estimación de esfuerzo (Sousa et al., 2023). Estos modelos no solo han mostrado buenos resultados en términos de error, sino que también han destacado por su capacidad para manejar relaciones complejas en los datos.

- **Random Forest para Regresión**

Random Forest es un modelo basado en la creación de múltiples árboles de decisión, cuyas predicciones se promedian para reducir el riesgo de sobreajuste. Este modelo ha demostrado ser eficaz en escenarios con

datos ruidosos y relaciones no lineales, lo cual es común en la estimación de esfuerzo en proyectos ágiles. En el artículo "Investigating the use of Random Forest in Software Effort Estimation", los autores destacan que, al optimizar los parámetros del modelo, Random Forest superó a otros modelos tradicionales, como los árboles de regresión, en términos de error.

La capacidad de Random Forest para manejar datos complejos y mejorar las predicciones lo convierte en una opción robusta para estimar el esfuerzo en proyectos de software, ya que puede captar patrones no lineales entre las características de las tareas y el esfuerzo necesario para completarlas. Esto lo posiciona como una técnica prometedora para la estimación de esfuerzo en el desarrollo de software (Abdelali et al., 2019).

- **Gradient Boosting para Regresión**

Gradient Boosting (GB) es un modelo de aprendizaje automático basado en árboles de decisión que mejora el ajuste de las predicciones mediante un enfoque secuencial. Cada árbol sucesivo corrige los errores de los árboles anteriores, lo que permite al modelo ajustarse de manera más precisa a los datos. Según el artículo "Software Effort Estimation using Machine Learning Techniques", el modelo de Gradient Boosting ha demostrado ser especialmente eficaz para capturar patrones complejos y no lineales en los datos, lo que lo convierte en una opción adecuada para tareas de predicción precisa, como la estimación del esfuerzo y el tiempo en el desarrollo de software.

Este estudio concluye que Gradient Boosting supera a modelos más simples, como la regresión lineal, al considerar las interacciones no lineales entre las variables. La capacidad de corregir errores secuenciales le permite lograr un ajuste más fino a los datos, lo que mejora el error de las estimaciones y lo convierte en un modelo robusto para la estimación de esfuerzo en proyectos de software. (Rahman et al., 2023).

- **XGBoost para Regresión**

XGBoost es una versión optimizada del modelo Gradient Boosting, reconocida por su eficiencia computacional y capacidad para manejar grandes volúmenes de datos, incluyendo aquellos incompletos. Este modelo se ha utilizado ampliamente en tareas de predicción debido a su capacidad para capturar patrones complejos con menor necesidad de recursos computacionales. Según el artículo "Applying Machine Learning to Estimate the Effort and Duration of Individual Tasks in Software Projects", XGBoost demostró ser uno de los mejores modelos en términos de capacidad de predicción y eficiencia.

El estudio concluye que los algoritmos basados en ensamblaje, como XGBoost, superaron consistentemente a los modelos no basados en ensamblaje en todos los conjuntos de datos evaluados. Esto se debe a su capacidad para proporcionar estimaciones más precisas y a su rapidez en el proceso de entrenamiento, lo que lo convierte en una opción atractiva en proyectos ágiles donde los recursos y el tiempo son limitados. Estos resultados confirman que XGBoost, junto con otros algoritmos de ensamblaje, es una opción viable y efectiva para la estimación del esfuerzo y la duración de tareas en proyectos de software, incluso cuando la calidad de las estimaciones individuales puede variar según las características del conjunto de datos (Sousa et al., 2023).

- **Ventajas y Desventajas de los Modelos**

A continuación, se resumen algunas de las ventajas y desventajas de los modelos:

Modelo	Ventajas	Desventajas
Random Forest	**Reducción del overfitting**, al combinar múltiples árboles de decisión, el Random Forest reduce el riesgo de overfitting y mejora la exactitud de las estimaciones. **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 exactitud**, estos modelos suelen ofrecer resultados excepcionales en términos de bajo error en las predicciones, especialmente en problemas complejos y de gran volumen de datos. **Flexibilidad en ajuste de hiper parámetros**, ofrecen una variedad de hiper parámetros 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.	**Mayor complejidad y dificultad de ajuste**, requieren un ajuste de hiper parámetros detallado y cuidadoso para maximizar su rendimiento. **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), basada en Amat Rodrigo (2015)

La elección de Random Forest, Gradient Boosting y XGBoost para la estimación del esfuerzo y la duración de las tareas en este proyecto se fundamenta en su rendimiento probado en estudios previos. Los artículos revisados muestran que estos modelos no solo ofrecen estimaciones más precisas que los modelos tradicionales, como la regresión lineal, sino que también son capaces de manejar relaciones no lineales y datos complejos que son comunes en la gestión de incidencias en proyectos ágiles

La combinación de estos modelos permite obtener predicciones más robustas y confiables, lo que es crucial para mejorar la planificación y la asignación de recursos en proyectos de desarrollo de software.

2.7. Evaluación de Modelos de Regresión

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 del error de predicción 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: Hastie, 2024

- **Error Cuadrático Medio (MSE)** y **Raíz del Error Cuadrático Medio (RMSE)**: Miden el promedio de los errores elevados al cuadrado. Penaliza los errores más grandes, lo que es útil para evaluar el rendimiento del modelo, especialmente cuando se desea reducir el impacto de las predicciones erróneas significativas.

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: Hastie, 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: Hastie, 2024

2.8. Comparación y Selección de Modelos de Machine Learning

La selección del modelo de aprendizaje automático (ML) más adecuado es una fase esencial en cualquier proyecto de análisis predictivo. Implica no solo evaluar el rendimiento del modelo en términos de métricas como el Error Medio Absoluto (MAE) y el Error Cuadrático Medio (MSE), sino también considerar

aspectos clave como la capacidad de generalización, la interpretabilidad y el tiempo de procesamiento. Este proceso permite elegir el modelo que mejor se ajusta a los objetivos y limitaciones del proyecto, garantizando que las predicciones sean efectivas y aplicables en el contexto real del problema.

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 Hiper parámetros**:
    - **Grid Search** explora un espacio predefinido de hiper parámetros, 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 hiper parámetros 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 hiper parámetros (Bergstra & Bengio, 2012).
- **Evaluación en el Conjunto de Prueba**: Una vez seleccionado el modelo y optimizados sus hiper parámetros, 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 se basa en una variedad de criterios que aseguran que el modelo sea práctico y efectivo:

- **Exactitud**: Las métricas de rendimiento permiten evaluar qué tan bien un modelo se ajusta a los datos y predice los resultados. En el caso de problemas de regresión, métricas como el Mean Absolute Error (MAE) y el Root Mean Squared Error (RMSE) se utilizan para medir la exactitud del modelo, es decir, qué tan cercanas son las predicciones a los valores reales en términos numéricos (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 brinda resultados confiables, sino que también sea fácil de entender. Los modelos simples, como la regresión lineal o los árboles de decisión, son interpretables y permiten desentrañar cómo cada característica contribuye a la predicción. En contraste, los modelos más complejos, como los

ensambles de árboles (por ejemplo, Random Forest o Gradient Boosting), pueden ser menos fáciles de interpretar, aunque a menudo generan estimaciones más confiables en términos de errores de predicción (Molnar, 2019).

- **Eficiencia Computacional**: Modelos como XGBoost, diseñados para ser escalables y eficientes, han demostrado ofrecer resultados de última generación en diversos problemas. Sin embargo, el enfoque en patrones de acceso a caché, compresión de datos y particionamiento, esenciales para su escalabilidad, también implica mayores requerimientos computacionales durante su entrenamiento (Chen & Guestrin, 2016). Este balance entre rendimiento y recursos puede representar un desafío en proyectos con restricciones de tiempo o capacidad de procesamiento.

El marco teórico expone la relevancia de integrar metodologías ágiles con herramientas como JIRA para mejorar la gestión de incidencias en proyectos de desarrollo de software. Aunque estas herramientas son eficaces para el monitoreo visual, su alcance es limitado cuando se trata de anticipar problemas complejos como retrasos o asignaciones ineficientes. En este contexto, el uso de machine learning surge como una solución poderosa al permitir el análisis de datos históricos y la generación de modelos predictivos.

A través de los fundamentos de machine learning y la revisión de algoritmos como Gradient Boosting, Random Forest y XGBoost, se destaca su capacidad para manejar relaciones no lineales y datos complejos, características inherentes a los proyectos ágiles. Estas herramientas no solo mejoran la precisión en las estimaciones, sino que también contribuyen a optimizar la planificación y la asignación de recursos, proporcionando una ventaja significativa en un entorno altamente dinámico.

El marco teórico sienta las bases para la implementación de un enfoque predictivo que aborda las limitaciones actuales en la gestión de incidencias, marcando un avance hacia la toma de decisiones basada en datos y el mejoramiento continuo en proyectos de software.

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 exactitud de 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 exactitud de las predicciones 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 y Mean Absolute Error (MAE) para los modelos de regresión. En base a los resultados de estas métricas, se seleccionó el mejor modelo para el objetivo, asegurando así la mayor exactitud 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, los equipos están 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:

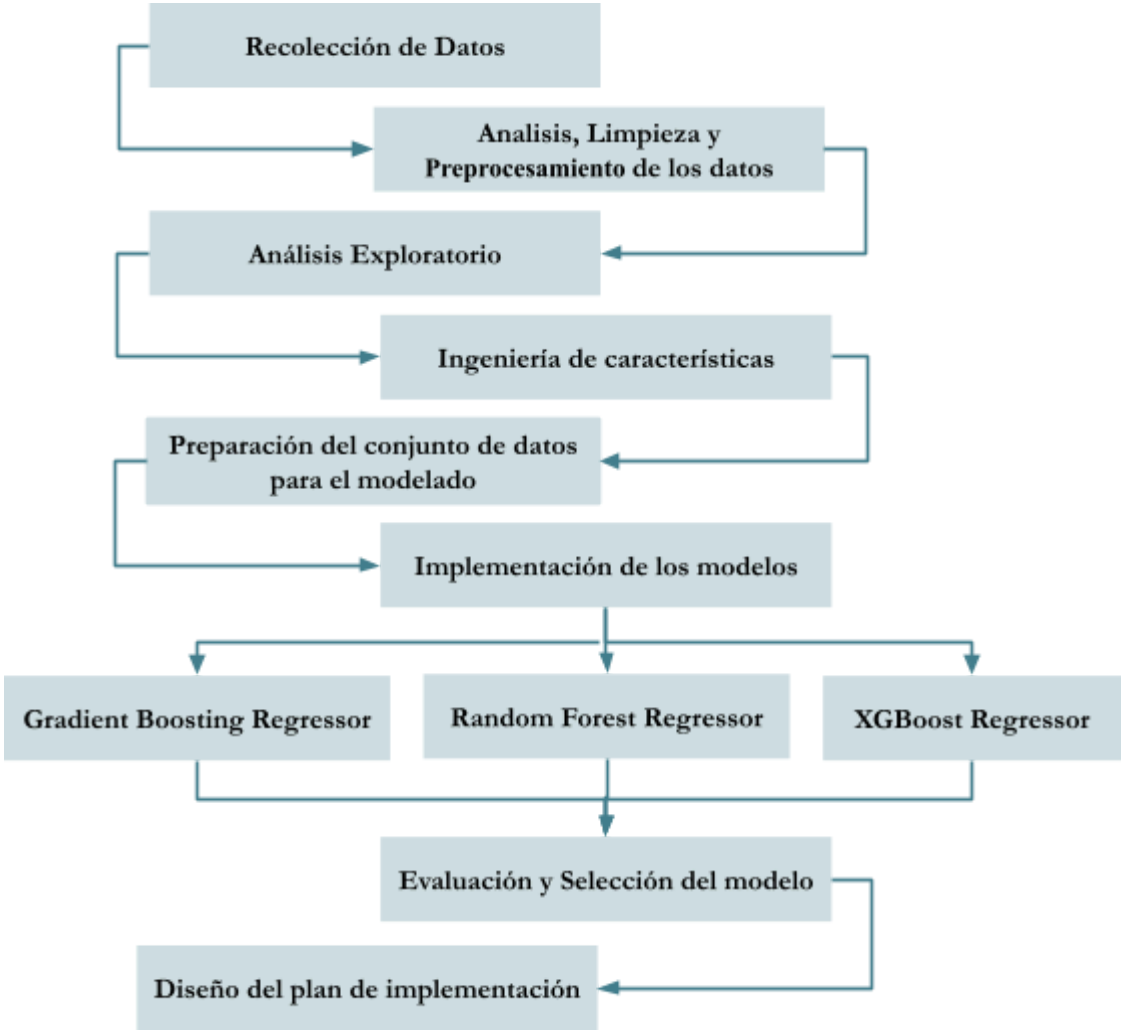


Figura 3-1: Flujograma Metodológico
Fuente: Elaboración Propia, 2024

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** (Anexo 2 y Anexo 3). 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. Analisis, 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 **analizar, 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. (ver código completo de esta etapa en Anexo 11)

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 (Figura 3-2) utilizando la librería de pandas. El conjunto de datos cuenta con 1429 filas y 200 columnas (ver conjunto de datos inicial en Anexo 1).

```
#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 (Figura 3-3), tipos de datos y el porcentaje de **valores faltantes** para identificar posibles problemas en los datos.

```
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.

```
# 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 (Figura 3-7). 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" (Figura 3-8), 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" (Figura 3-9), 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"(Figura 3-10) 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" (Figura 3-11). 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 (ver código en Anexo 5):

- "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**.

Esta conversión es crucial porque permite una manipulación más eficiente y precisa de los datos, además de facilitar los cálculos y transformaciones necesarios. Al asignar tipos específicos como fechas, se habilita el acceso a una variedad de funciones predefinidas en Python que optimizan el análisis temporal y las operaciones relacionadas. Esto no solo mejora la integridad del proceso de limpieza de datos, sino que también asegura que el modelo de machine learning reciba datos en un formato adecuado para maximizar su rendimiento y precisión.

3.2.2.8. Transformación de Transiciones

Las columnas que registran las transiciones de estado, como "To Do - In Progress" y otras similares, fueron sometidas a las siguientes transformaciones (ver código en Anexo 6):

- Reestructuración con melt de pandas: Se transformaron las columnas de transición de estado en filas, manteniendo como referencias las columnas relevantes, tales como "Key", "Priority" e "Issue Type".

- Segmentación de estados: La columna "State Transition" se dividió en dos nuevas columnas: "Initial State" y "Next State", permitiendo un análisis más detallado del flujo de trabajo asociado a cada incidencia.
- Ordenamiento cronológico: Los datos se organizaron por "Key" y "Transition Date", asegurando que la secuencia de transiciones se mantuviera en un orden cronológico correcto.
- Resultado final: Al completar el proceso de limpieza y pivoteo, el conjunto de datos resultante quedó compuesto por 2675 filas y 15 columnas, listo para las etapas posteriores de análisis y modelado.

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 (ver código completo del análisis exploratorio en Anexo 12), 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.

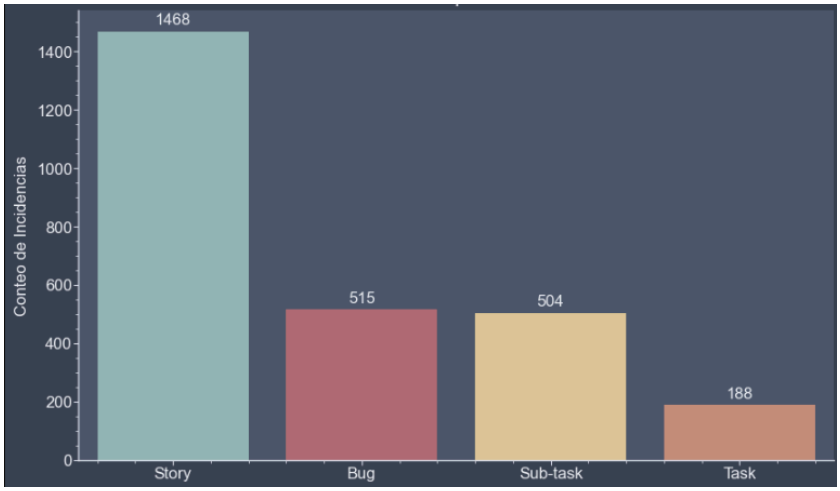


Figura 3-12: Distribución de los tipo de incidencias
Fuente: Elaboración Propia, 2024

La figura 3-12 nos muestra un predominio de incidencias de tipo "Story", lo que indica un enfoque en desarrollar nuevas funcionalidades y entregar valor al cliente. A su vez, el número significativo de "Bugs" y "Sub-tasks" sugiere atención al mantenimiento y descomposición de tareas para facilitar su ejecución. En contraste, las "Tasks" generales son menos frecuentes, lo que sugiere que tienen menor prioridad en este contexto.

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.

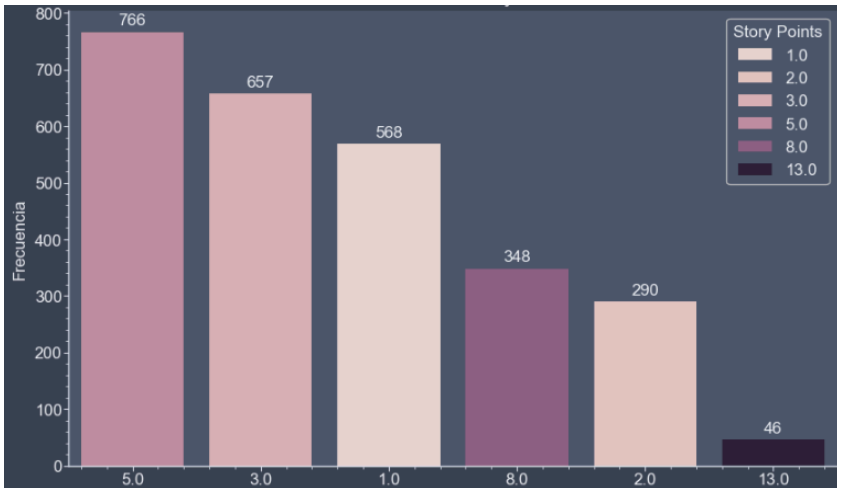


Figura 3-13: Distribución de las estimaciones de las incidencias
Fuente: Elaboración Propia, 2024

La Figura 3-13 muestra que predominan los Story Points medianos, con la mayoría de tareas asignadas a 5 Story Points, seguidas de 3 Story Points y 1 Story Point, 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 y 2 Story Points, 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.

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.

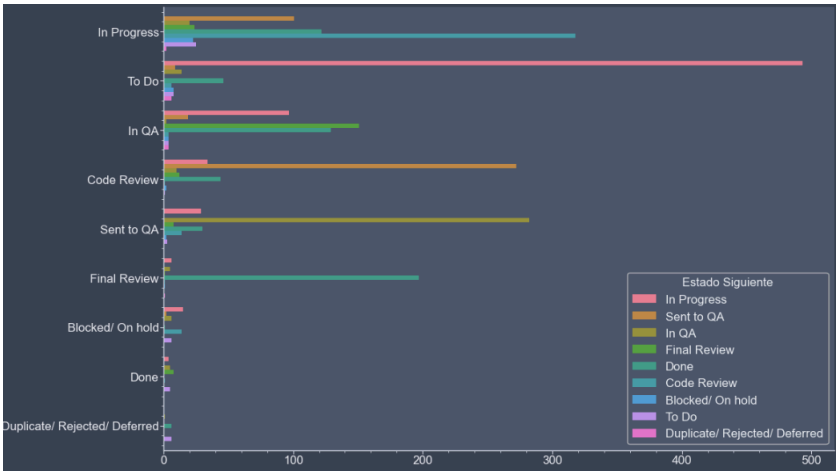


Figura 3-14: Transiciones de las incidencias de un estado a otro
Fuente: Elaboración Propia, 2024

La Figura 3-14 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.

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.

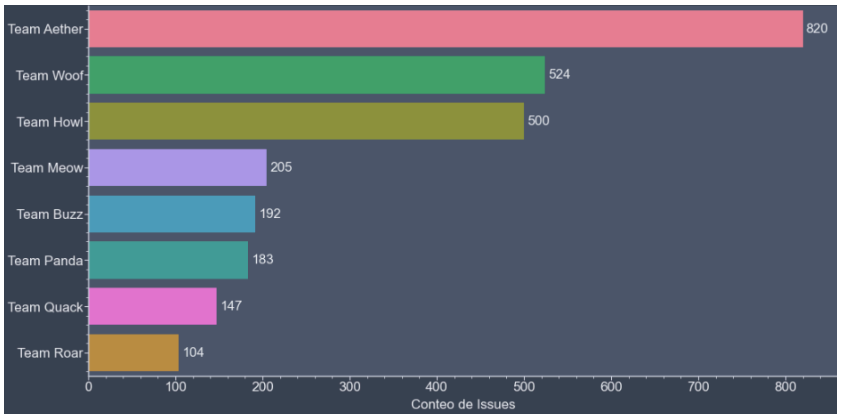


Figura 3-15: Cantidad de incidencias completadas por equipo
Fuente: Elaboración Propia, 2024

La figura 3-15 muestra el número total de incidencias completadas por cada equipo.

- "Team Aether" tiene la mayor cantidad de incidencias completadas, lo que indica **alta productividad** en comparación con los demás equipos.
- "Team Woof" y "Team Howl" también tienen un desempeño significativo.
- 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.

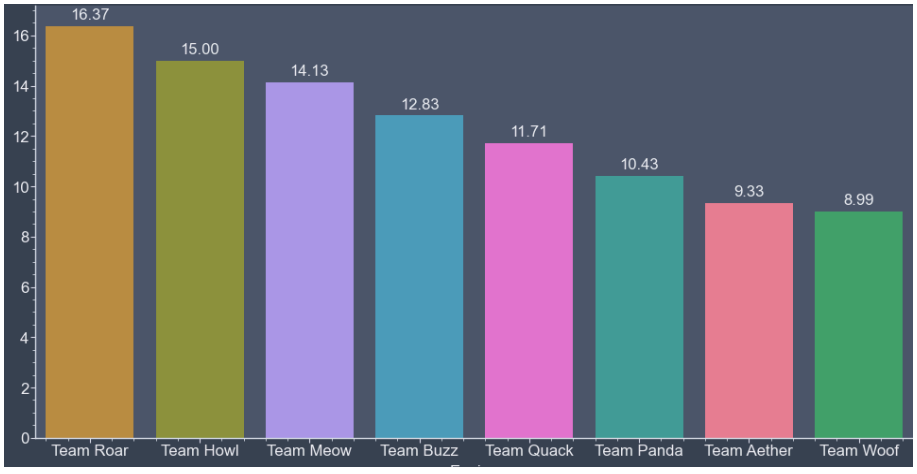


Figura 3-16: Tiempo promedio de resolución de incidencias por equipo
Fuente: Elaboración Propia, 2024

La figura 3-16 analiza el tiempo promedio en días en que cada equipo tarda en resolver una incidencia.

- "Team Roar" tiene el tiempo de resolución más alto, 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, 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.

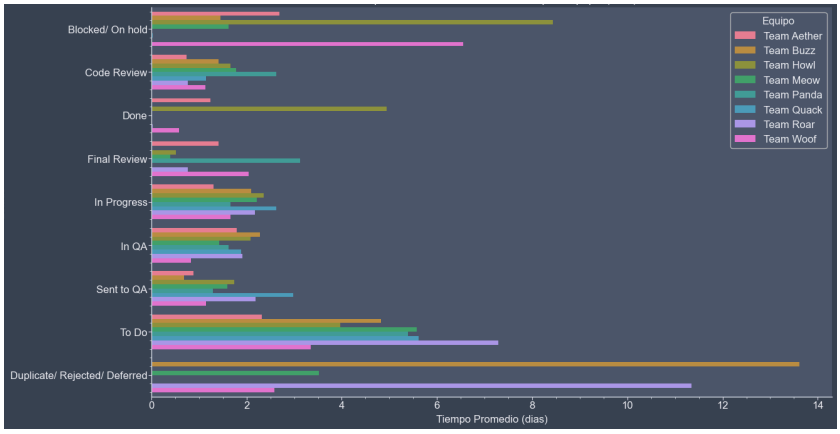


Figura 3-17: Tiempo promedio que pasa una incidencia en cada estado por equipo
Fuente: Elaboración Propia, 2024

La figura 3-17 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.

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.

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 (Figura 3-18) analiza el número de incidencias completadas mensualmente entre mayo y octubre de 2024, revelando fluctuaciones en la productividad a lo largo del período.
Los meses de junio y septiembre 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.

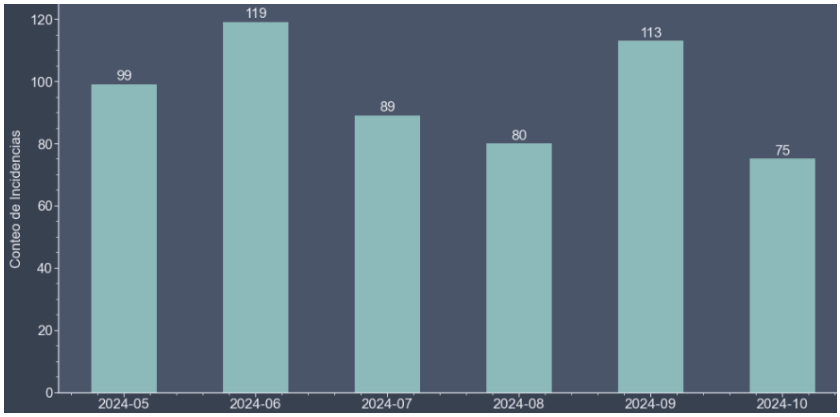


Figura 3-18: Incidencias completadas por mes
Fuente: Elaboración Propia, 2024

En contraste, agosto y octubre 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.

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 el rendimiento y la exactitud 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 3-19: Código para definir si una incidencia ha sido bloqueada
Fuente: Elaboración Propia, 2024

En el código de la Figura 3-19 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 mide la eficiencia de un desarrollador al resolver una incidencia, evaluando la relación entre la estimación inicial (en puntos de historia) y el tiempo real empleado para su resolución. Su principal utilidad radica en identificar patrones de desempeño que revelen si ciertos desarrolladores son más eficientes al abordar tipos específicos de incidencias. Esto no solo facilita la asignación estratégica de tareas, sino que también optimiza la productividad del equipo al aprovechar las fortalezas individuales de cada miembro.

La Figura 3-20 presenta la fórmula utilizada para calcular la eficiencia de un desarrollador, basada en los puntos de historia y el tiempo de resolución. Este cálculo se realiza dividiendo los puntos de historia asignados a una incidencia entre el tiempo necesario para resolverla. Esta métrica permite evaluar la productividad del desarrollador en función de la complejidad de las tareas asignadas, proporcionando una base cuantitativa para mejorar la planificación y distribución de las cargas de trabajo.

$$\text{Eficiencia del Miembro por Ticket} = \begin{cases} \frac{\text{Story Points}}{\text{Tiempo de Resolución}} & \text{si Tiempo de Resolución} > 0 \\ 0 & \text{si Tiempo de Resolución} \leq 0 \end{cases}$$

Figura 3-20: Formula Eficiencia de un desarrollador basado en los story points y el tiempo de resolución
Fuente: Elaboración Propia (2024), basada en Schwaber y Beedle (2002)

3.2.4.3. Retrasos en el sprint

Identificar si un sprint se completó después de la fecha programada proporciona información clave para evaluar la planificación, la capacidad del equipo y la gestión de recursos. Este análisis permite detectar patrones recurrentes en los retrasos, facilitando ajustes en futuras estimaciones, una asignación de tareas más eficiente y, en última instancia, una mejora en la gestión general de los proyectos.

La Figura 3-21 ilustra la fórmula utilizada para calcular la métrica "Sprint Overdue", la cual evalúa si un sprint excedió su plazo programado. El cálculo se basa en comparar la fecha real de finalización del sprint (Sprint Complete Date) con la fecha planificada de conclusión (Sprint End Date). Si la fecha real es posterior a la planificada, el indicador asigna un valor de 1 en la columna Sprint Overdue; en caso contrario, asigna un valor de 0.

$$\text{Sprint Overdue} = \begin{cases} 1 & \text{si Fecha de Compleción del Sprint} > \text{Fecha de Fin del Sprint} \\ 0 & \text{si Fecha de Compleción del Sprint} \leq \text{Fecha de Fin del Sprint} \end{cases}$$

Figura 3-21: Fórmula para identificar si un sprint ha presentado retrasos en su conclusión
Fuente: Elaboración Propia (2024), basada en Sutherland y Schwaber (2013)

Este indicador resulta valioso para identificar desviaciones en los plazos, permitiendo realizar ajustes proactivos en la planificación y asegurar una mayor precisión en la gestión de proyectos futuros.

3.2.4.4. Tiempo restante del sprint

El propósito de esta métrica es ofrecer una visión detallada sobre el tiempo restante en el sprint al momento de completar una incidencia. Este análisis permite 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 la consistencia en el ritmo de trabajo. Al interpretar esta información, los equipos pueden tomar decisiones informadas para mejorar la planificación de los sprints, equilibrar la carga de trabajo y establecer estrategias más efectivas que aseguren el cumplimiento de los plazos, maximizando la productividad.

La Figura 3-22 presenta la fórmula utilizada para calcular los días restantes en el sprint al completar una incidencia. Esta métrica se obtiene restando la fecha real de finalización de la incidencia (Sprint Complete Date) de la fecha planificada de conclusión del sprint (Sprint End Date). El resultado, expresado en días, se almacena en la columna Days to Sprint End. Este indicador permite evaluar la eficiencia en el uso del tiempo disponible durante el sprint y planificar mejoras en los procesos futuros.

$$\text{Días Restantes al Final del Sprint} = \text{Sprint End Date} - \text{Sprint Complete Date}$$

Figura 3-22: Fórmula para calcular los días restantes para la culminación de un sprint
Fuente: Elaboración Propia (2024), basada en Sutherland y Schwaber (2013)

3.2.4.5. Eficiencia del equipo por sprint

Un valor más alto en esta métrica refleja una mayor eficiencia, evidenciando la capacidad del equipo para completar más "Story Points" en menos tiempo. Esta métrica tiene como objetivo evaluar el desempeño colectivo de un equipo durante un sprint, proporcionando una visión integral de su eficiencia promedio al resolver tareas. Su utilidad radica en identificar diferencias en la productividad entre sprints o equipos, medir mejoras a lo largo del tiempo y analizar la efectividad de las asignaciones de tareas y recursos. Con esta información, se pueden ajustar estrategias de planificación y gestión para optimizar el rendimiento en futuros sprints.
La Figura 3-23 presenta la fórmula utilizada para calcular la Eficiencia Promedio del Equipo por Sprint, basada en el promedio de la eficiencia individual de los miembros por cada incidencia. Este cálculo agrupa las incidencias por equipo (T) y sprint (S), promediando los valores de la columna "Member Efficiency per Ticket". El resultado se almacena en la columna "Average Team Efficiency per Sprint".

Fórmula Representada en la Figura 3-23:

- $G_{T,S}$ es el conjunto de incidencias I_i que pertenecen al **equipo** T y al **sprint** S.
- $E(Ii)$ es la eficiencia del miembro del equipo en la incidencia I_i
- $|G_{T,S}|$ es el número total de incidencias en el grupo $G_{T,S}$, es decir, el número de incidencias del equipo Ten el sprint S.

Esta métrica permite identificar tendencias de rendimiento, mejorar estrategias de planificación y gestionar recursos con mayor efectividad en futuros sprints.

$$\text{Avg Efficiency}_{T,S} = \frac{1}{|G_{T,S}|} \sum_{I_i \in G_{T,S}} E(I_i)$$

Figura 3-23: Formula para calcular la eficiencia del equipo por sprint
Fuente: Elaboración Propia(2024), basada en Sutherland y Schwaber (2013)

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 3-24: Código para identificar la complejidad de las incidencias
Fuente: Elaboración Propia, 2024

La Figura 3-24 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 objetivo de esta métrica es evaluar la carga total de trabajo asignada a cada equipo durante un sprint, lo que permite identificar posibles desequilibrios en la distribución de tareas, analizar si la carga está alineada con la capacidad del equipo y comparar el esfuerzo total entre diferentes sprints o equipos. Este análisis facilita la optimización de la planificación y asegura una asignación justa y realista de los puntos de historia, mejorando la eficiencia y el cumplimiento de los objetivos del sprint.

El cálculo de la velocidad del equipo se representa de la siguiente manera:
Para un **equipo** T en un **sprint** S, las incidencias completadas son I_1, I_2,...,I_n. Cada incidencia I_i tiene un número de **Story Points** denotado por $SP(I_i)$.

Donde:

- $C_{T,S}$: Conjunto de incidencias completadas por el equipo T en el sprint S.
- $SP(I_i)$: Story Points asignados a la incidencia I_i.

La Figura 3-25 muestra la fórmula para calcular la suma de los "Story Points" por "Equipo" y "Sprint". Este cálculo se realiza agrupando los datos por las columnas "Team" y "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 representa la velocidad del equipo: la cantidad de puntos de historia que el equipo puede completar durante un sprint.

$$\text{Velocidad del Equipo}_{T,S} = \sum_{I_i \in C_{T,S}} SP(I_i)$$

Figura 3-25: Fórmula para calcular la cantidad de story points completados por sprint y por equipo
Fuente: Elaboración Propia (2024), basada en Sutherland y Schwaber (2013)

3.2.4.8. Velocidad del responsable

Esta métrica ofrece 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 corresponden a las capacidades de cada miembro y analizar patrones de desempeño individual. Esto es esencial para mejorar la planificación, asegurar una distribución equitativa de las tareas y maximizar la productividad del equipo, aprovechando las fortalezas de cada integrante.

La representación matemática aplicado para el cálculo de la velocidad del desarrollador es la siguiente; para un miembro M asignado a un sprint S, las incidencias completadas son I_1, I_2,..., I_n. Cada incidencia I_i tiene un número de **Story Points** denotado por $SP(I_i)$.

Donde:

- $A_{M,S}$: Conjunto de incidencias asignadas al miembro MMM en el sprint SSS.
- $SP(I_i)$: Story Points asignados a la incidencia I_i.

La Figura 3-26 ilustra el cálculo de la suma de los "Story Points" por miembro del equipo y sprint. Para ello, 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".

$$\text{Total Story Points}_{M,S} = \sum_{I_i \in A_{M,S}} SP(I_i)$$

Figura 3-26: Fórmula para calcular los story points completados por cada responsable en cada sprint
Fuente: Elaboración Propia, 2024

3.2.4.9. Exactitud 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.

En el Anexo 7 se presenta la creación de la métrica Exactitud de la Estimación (Estimation Accuracy), 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 real de resolución (Resolution Time) en horas. Se establecen rangos de tiempo específicos para cada valor de puntos de historia, basándose en estándares comunes de la industria para evaluar la exactitud de las estimaciones.

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 3-27 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.

	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

Figura 3-27: Tabla que contabiliza los valores faltantes en el conjunto de datos
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 3-28). Esta estrategia asegura que las imputaciones no distorsionen significativamente el análisis posterior.

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

Figura 3-28: 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 exactitud 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
numeric_columns = [col for col in numeric_columns if col != 'Resolution Time']
data[numeric_columns] = scaler.fit_transform(data[numeric_columns])
data
```

Figura 3-29: Código para normalizar variables cuantitativas
Fuente: Elaboración Propia, 2024

La Figura 3-29 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 3-30). 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 3-30: 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 (Anexo 4 y Anexo 13), 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** (ver código completo del entrenamiento en Anexo 14). 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 exactitud 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 exactitud.

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 exactitud 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 3-31 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.

```
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 3-31: 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 (ver código en Anexo 8) 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.

Los hiper parámetros finales son:

- **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.

Para el entrenamiento se utilizó el modelo GradientBoostingRegressor de la librería sklearn.ensemble (Figura 3-32), configurado con los hiper parámetros optimizados previamente mediante la función RandomizedSearchCV. Además, se generó un gráfico de importancia de características que muestra las variables con mayor influencia en el rendimiento del modelo. Este análisis es especialmente útil, ya que permite identificar y visualizar las características más relevantes en el proceso de predicción, lo que facilita una mejor comprensión del comportamiento del modelo y aporta información valiosa para refinar futuros análisis y decisiones basadas en datos.

```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 3-32: Código de entrenamiento del 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 3-33). 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.

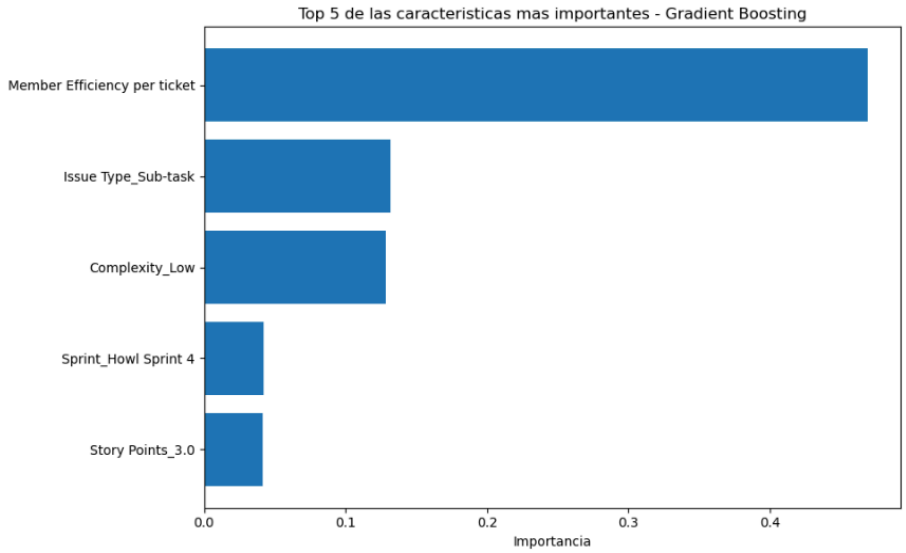


Figura 3-33: Importancia de las características para el modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

3.2.6.4. Random Forest Regressor

De manera similar al modelo anterior, se utilizó Randomized Search para identificar los hiper parámetros más óptimos, asegurando un ajuste eficiente y mejorando el rendimiento del modelo (ver código en Anexo 9).

Los hiper parámetros finales son:

- **n_estimators = 500:** Un valor de 500 significa que el modelo creará 500 árboles para mejorar la estabilidad del modelo y reducir el sesgo de las predicciones

- **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 = 10:** Limita la profundidad de los árboles para evitar 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.

Con estos hiper parámetros, se configuró el modelo RandomForestRegressor de la librería sklearn.ensemble (Figura 3-34) y se procedió a su entrenamiento mediante la función .fit, utilizando el conjunto de datos 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=10,
    n_jobs = -1
)
rf_model.fit(X_train, y_train)
```

Figura 3-34: Código de entrenamiento del modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

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

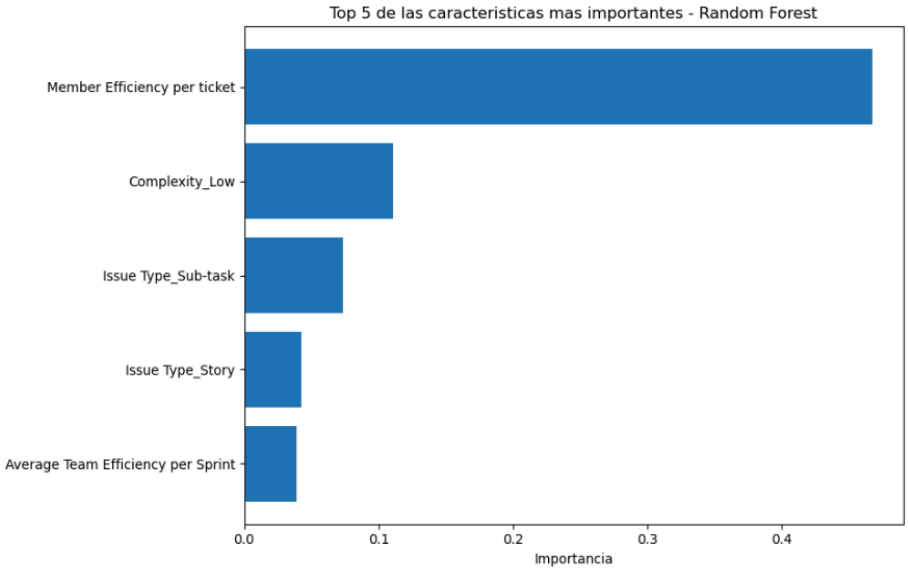


Figura 3-35: Importancia de las características para el modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

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 3-35). 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

El Anexo 10 muestra la aplicación de Randomized Search para encontrar los hiper parámetros óptimos del modelo XGBoost Regressor, permitiendo optimizar el entrenamiento y maximizar su rendimiento.

Los hiper parámetros finales son:

- **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 (L2 regularización):** Penaliza valores altos de los coeficientes para evitar sobreajuste. Un valor de 1.2 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 = 100:** Usa 100 árboles, permitiendo que el modelo aprenda mejor patrones complejos. Más árboles implican más tiempo de entrenamiento.
- **max_depth = 3:** Restringe la profundidad de los árboles a 3 niveles, equilibrando el aprendizaje entre captar detalles y evitar sobreajuste.
- **learning_rate = 0.2:** Un valor de 0.2 representa una tasa de aprendizaje moderada, lo que significa que el modelo realizará ajustes significativos pero no excesivos en cada paso.
- **gamma = 0.1:** 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.

Con estos hiper parámetros se configura el modelo (Figura 3-36) y se procede al entrenamiento.

```python
from xgboost import XGBRegressor

xgb_model = XGBRegressor(
    subsample=1.0,
    reg_lambda=1,
    reg_alpha=0,
    n_estimators=100,
    max_depth=3,
    learning_rate=0.2,
    gamma=0.1,
    colsample_bytree=1.0,
    random_state=42
)

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

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

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

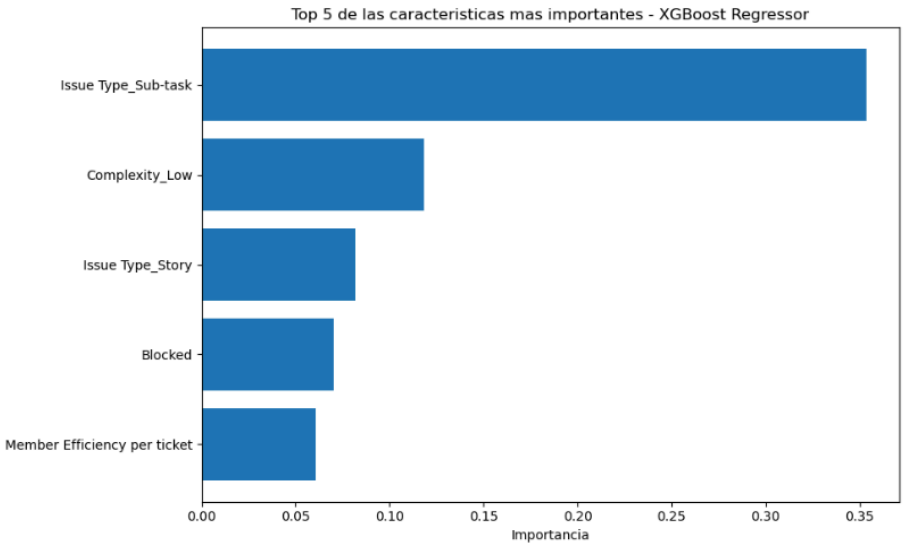


Figura 3-37: Importancia de las características para el modelo XGBoost Regressor
Fuente: Elaboración Propia, 2024

En este modelo, la variable "Issue Type_Sub-task" tiene la mayor influencia en el entrenamiento, seguida de "Complexity_Low" y "Issue Type_Story" (Figura 3-37). Es el modelo que tiene una mayor dependencia a la variable Issue Type en comparación con los modelos previos.

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 llevar a cabo una evaluación rigurosa que permita comparar resultados y seleccionar el modelo que ofrezca el mejor equilibrio entre exactitud, capacidad de 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 el error promedio absoluto en las predicciones de valores continuos.
- **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 las funciones utilizadas para evaluar el rendimiento de los modelos.

3.2.7.1. Evaluación del Rendimiento de Gradient Boosting Regressor

La evaluación de este modelo emplea las funciones mean_absolute_error, mean_squared_error y r2_score, proporcionadas por la biblioteca scikit-learn en Python (Figura 3-38). Estas métricas permiten calcular el

error medio absoluto (MAE), el error cuadrático medio (MSE) y el coeficiente de determinación (R^2), respectivamente. Estas herramientas son esenciales para evaluar el desempeño del modelo y su capacidad de generalización sobre datos no vistos.

```
y_pred_gbr = gbr_model.predict(X_test)

mae_gbr = mean_absolute_error(y_test, y_pred)
mse_gbr  = mean_squared_error(y_test, y_pred)
r2_gbr  = r2_score(y_test, y_pred)

print("Gradient Boosting Regressor")
print(f"Mean Absolute Error (MAE): {mae_gbr }")
print(f"Mean Squared Error (MSE): {mse_gbr }")
print(f"R² Score: {r2_gbr }")
```

```
Gradient Boosting Regressor
Mean Absolute Error (MAE): 1.307176045442665
Mean Squared Error (MSE): 12.75481194945286
R² Score: 0.8290343923338849
```

Figura 3-38: Código de evaluación del modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

Para esta evaluación y las posteriores, se utiliza el modelo entrenado junto con dos subconjuntos de datos: y_test y y_pred, que representan el conjunto de valores reales de prueba y el conjunto de valores predichos por el modelo, respectivamente. Para obtener y_pred, se emplea la función .predict del modelo sobre el conjunto X_test, que contiene las características del conjunto de prueba. De esta manera, el modelo genera las predicciones basadas en los datos de entrada proporcionados en X_test, y estas predicciones se comparan con y_test para calcular las métricas de evaluación, como el MAE, el MSE y el R^2. Este proceso permite validar el desempeño del modelo en datos no vistos, asegurando su capacidad de generalización.

```
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: 1.332932037302276
```

Figura 3-39: Código de validación cruzada para el modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

En la validación cruzada (Figura 3-39), se utilizó la función cross_val_score de la biblioteca scikit-learn, configurada con el parámetro scoring=neg_mean_absolute_error para evaluar el error absoluto medio negativo como métrica de desempeño. Este enfoque permite medir la capacidad predictiva del modelo en diferentes particiones del conjunto de datos, lo que ayuda a garantizar una evaluación robusta y evitar sobreajuste. Además, se estableció cv=5 para realizar una validación cruzada con cinco pliegues, dividiendo los datos en cinco subconjuntos diferentes. Esto asegura que cada parte del conjunto de datos se use tanto para entrenamiento como para prueba, promoviendo una evaluación más equilibrada y representativa del modelo.

3.2.7.2. Evaluación del Rendimiento de Random Forest Regressor

De manera similar al modelo anterior, se emplean las mismas funciones para calcular las métricas de evaluación: MAE, MSE y R^2, utilizando los mismos subconjuntos de datos (y_test y y_pred). La única diferencia radica en que las predicciones se generan utilizando el modelo de Random Forest previamente entrenado, aplicando la función .predict sobre el conjunto X_test (Figura 3-40). Este enfoque asegura que las métricas de evaluación sean consistentes y comparables entre ambos modelos.

```
y_pred_rf = rf_model.predict(X_test)

mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred)

print("Random Forest Regressor")
print(f"Mean Absolute Error (MAE): {mae_rf}")
print(f"Mean Squared Error (MSE): {mse_rf}")
print(f"R² Score: {r2_rf}")
```

```
Random Forest Regressor
Mean Absolute Error (MAE): 0.8803930887661476
Mean Squared Error (MSE): 10.3881296584871
R² Score: 0.8545479406483211
```

Figura 3-40: Código de evaluación del modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

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: 1.6239391977641688
```

Figura 3-41: Código de validación cruzada para el modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

En la validación cruzada (Figura 3-41), el modelo Random Forest Regressor 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

Se utiliza el modelo entrenado xgb_model y su función .predict para generar y_pred (predicciones). Estas predicciones se comparan con los valores reales del conjunto de prueba para calcular las métricas de evaluación, como el MAE, MSE y R^2 (Figura 3-42), lo que permite medir el desempeño del modelo en términos de exactitud y capacidad de generalización.

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.

En la validación cruzada (Figura 3-43), el modelo XGB Regressor 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.

```
y_pred_xgb = xgb_model.predict(X_test)

mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print("XGB Regressor")
print(f"Mean Absolute Error (MAE): {mae_xgb}")
print(f"Mean Squared Error (MSE): {mse_xgb}")
print(f"R² Score: {r2_xgb}")
```

```
XGB Regressor
Mean Absolute Error (MAE): 1.3639388874052727
Mean Squared Error (MSE): 25.375582170277326
R² Score: 0.6598654811363958
```

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

```
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: 1.656899891013495
```

Figura 3-43: Código de validación cruzada para el modelo XGBoost Regressor
Fuente: Elaboración Propia, 2024

3.2.7.4. Selección del Modelo

Después del entrenamiento de los modelos y el análisis de las métricas de evaluación se puede deducir lo siguiente:

- Gradient Boosting Regressor por su equilibrio entre exactitud y capacidad de generalización, es el modelo más adecuado.
- Random Forest Regressor si se prioriza la exactitud en los datos de entrenamiento sobre la generalización.
- 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 3-4: 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 exactitud 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. (ver código ejemplo del plan de implementación en Anexo 15)

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 (Figura 3-44).

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

Figura 3-44: 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.
- **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 (Figura 3-45).

```
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 3-45: 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 (Figura 3-46).

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

Figura 3-46: 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 (Figura 3-47).

```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 3-47: 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 exactitud 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 desempeño (Figura 3-48). 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 3-48: Código de ejemplo para el reentrenamiento del modelo
Fuente: Elaboración Propia, 2024

En este capítulo se abordó el procesamiento integral del conjunto de datos, comenzando con la limpieza y análisis exhaustivo de la información para garantizar la calidad y coherencia de los datos. Este paso permitió obtener una comprensión profunda del conjunto de datos, lo cual es fundamental para la creación de nuevas características (feature engineering) que optimicen el rendimiento de los modelos de regresión. A través de esta técnica, se generaron variables adicionales que facilitan un mejor ajuste de los modelos y mejoran su capacidad para predecir con mayor exactitud.

Posteriormente, se llevó a cabo la búsqueda de hiper parámetros, un proceso esencial para la optimización de los modelos. Utilizando herramientas como Randomized Search, se lograron ajustar los parámetros clave para maximizar el desempeño y la generalización de los modelos, lo que resultó en modelos más robustos y confiables.

Finalmente, se desarrolló un plan de implementación del modelo, que incluye la propuesta de integración de los modelos predictivos en plataformas como JIRA. Esta integración permitiría obtener las predicciones de manera más eficiente, presentándose como una columna personalizada dentro de la interfaz de la plataforma, lo que facilita la toma de decisiones en tiempo real.

Este capítulo presenta los modelos entrenados, los cuales serán validados en etapas posteriores. Además, se realizó una selección cuidadosa del modelo más adecuado, basándonos en su rendimiento general y en su capacidad para abordar el problema específico de este proyecto, lo que proporciona una base sólida para la implementación en un entorno de producción.

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	15.346312	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	14.541528	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	6.449189	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	9.453486	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	14.172117	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 error y capacidad de ajuste.

4.2.1. Resultados del modelo Gradient Boosting Regressor

Los resultados de las métricas son las siguientes:

- **Mean Absolute Error (MAE): 1.3072**

  Este valor promedio indica que el modelo tiene un error absoluto medio de 1.31 días en las predicciones, lo que refleja un buen desempeño en la estimación del tiempo de resolución.

- **Mean Squared Error (MSE): 12.7548**

  El MSE, que penaliza los errores más grandes, es consistente con la naturaleza del problema y demuestra que el modelo maneja bien la mayoría de las desviaciones sin que estas impacten significativamente en la calidad general de las predicciones.

- **R^2 Score: 0.8290**

  Con un R^2 de 82.90%, el modelo explica una proporción significativa de la variabilidad en los datos de tiempo de resolución. Este nivel de ajuste es excelente en un contexto de datos reales, donde existen factores externos que no pueden modelarse directamente.

- **Cross-Validation: 1.3329**

  Durante la validación cruzada, el modelo mostró un error absoluto promedio de 1.33 días, lo que confirma su capacidad generalizadora y su fiabilidad para realizar predicciones en nuevos datos.

4.2.2. Resultados del modelo Random Forest Regressor

Los resultados de las métricas son las siguientes:

- **Mean Absolute Error (MAE): 0.8804**

  El error promedio absoluto es de 0.88 días, lo que indica que el modelo tiene un buen desempeño, logrando predicciones que se aproximan de manera consistente a los valores reales. Este MAE bajo es clave para proporcionar estimaciones confiables en aplicaciones prácticas.

- **Mean Squared Error (MSE): 10.3881**

  El MSE refleja que el modelo maneja bien los errores más grandes, analizándolos de manera apropiada, y demuestra un ajuste general adecuado a los datos sin grandes desviaciones que impacten significativamente en las predicciones.

- **R^2 Score: 0.8545**

  Con un R^2 de 85.45%, el modelo captura una alta proporción de la variabilidad en los tiempos de resolución. Este resultado indica que el modelo tiene un ajuste fuerte y es capaz de capturar patrones relevantes en los datos, proporcionando predicciones alineadas con las tendencias observadas.

- **Cross-Validation: 1.6239**

  Durante la validación cruzada, el error promedio absoluto fue de 1.62 días, lo que confirma que el modelo tiene una buena capacidad generalizadora, aunque el error promedio es ligeramente mayor en nuevos datos, sigue siendo confiable para realizar predicciones útiles.

4.2.3. Resultados del modelo XGBoost Regressor

Los resultados de las métricas son las siguientes:

- **Mean Absolute Error (MAE): 1.3639**

  El valor obtenido indica que, en promedio, las predicciones del modelo presentan una desviación de 1.36 días respecto a los valores reales. Aunque este error promedio absoluto puede considerarse considerable, podría ser adecuado para aplicaciones donde una desviación de un día y medio sea tolerable y suficiente para los fines prácticos del modelo.

- **Mean Squared Error (MSE): 25.3756**

  Este valor sugiere que el modelo experimenta algunos errores más grandes en las predicciones. Al penalizar de manera significativa las desviaciones más amplias, el MSE refleja que, si bien el modelo tiene un rendimiento razonable en general, existen casos extremos que afectan su desempeño.

- **R^2 Score: 0.6599**

  El modelo explica el 65.99% de la variabilidad en los datos. Este resultado muestra que el modelo es capaz de capturar una parte considerable de los patrones en los datos, aunque queda margen para una mayor explicación de la variabilidad.

- **Cross-Validation: 1.6569**

  Este valor de error absoluto promedio durante la validación cruzada indica que el modelo mantiene una consistencia moderada en sus predicciones cuando se enfrenta a datos nuevos. Aunque el error promedio es significativo, el modelo muestra cierta capacidad generalizadora.

4.2.4. Comparación de las Métricas de evaluación

A continuación, se comparan los resultados de la evaluación de los modelos con el objetivo de analizar y justificar la decisión tomada en la selección del modelo. Este análisis permitió concluir que el Gradient Boosting Regressor es el modelo más adecuado para este proyecto, gracias a su desempeño superior en términos de las métricas de evaluación empleadas.

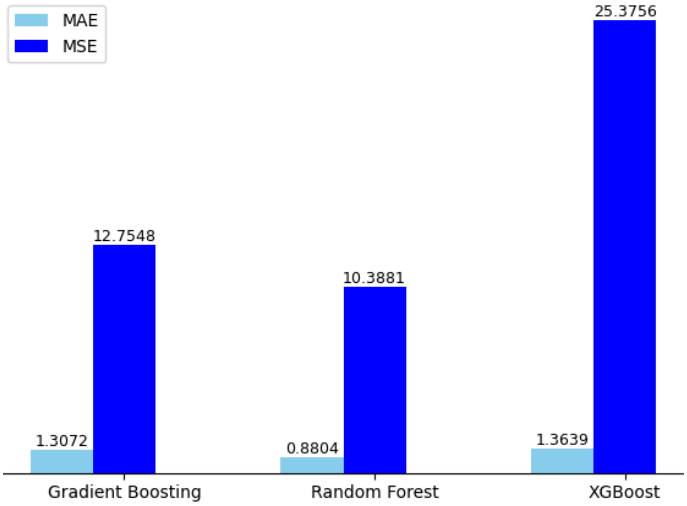


Figura 4-2: Gráfico comparativo del MAE y el MSE de los tres modelos evaluados
Fuente: Elaboración Propia, 2024

En la Figura 4-2 se evidencia que, aunque las diferencias entre las métricas de desempeño de Random Forest y Gradient Boosting son mínimas, con un MAE de 1.3072 frente a 0.8804 y un MSE de 12.7548 frente a 10.3881, Random Forest muestra valores ligeramente superiores, consolidándose como el modelo con mejor desempeño. Ambos modelos se destacan como excelentes opciones para la tarea propuesta, mientras que XGBoost queda notablemente rezagado, presentando un MAE de 1.3639 y un MSE de 25.3756, lo que indica un error considerablemente mayor.

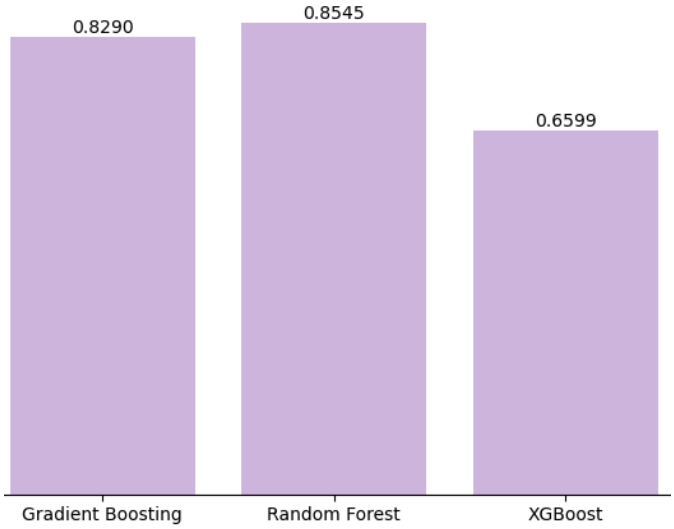


Figura 4-3: Gráfico comparativo del coeficiente de determinación de los tres modelos evaluados
Fuente: Elaboración Propia, 2024

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 85% de la variabilidad de los datos, ligeramente por encima del 83% 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 66%, un porcentaje que no resulta óptimo para el problema planteado.

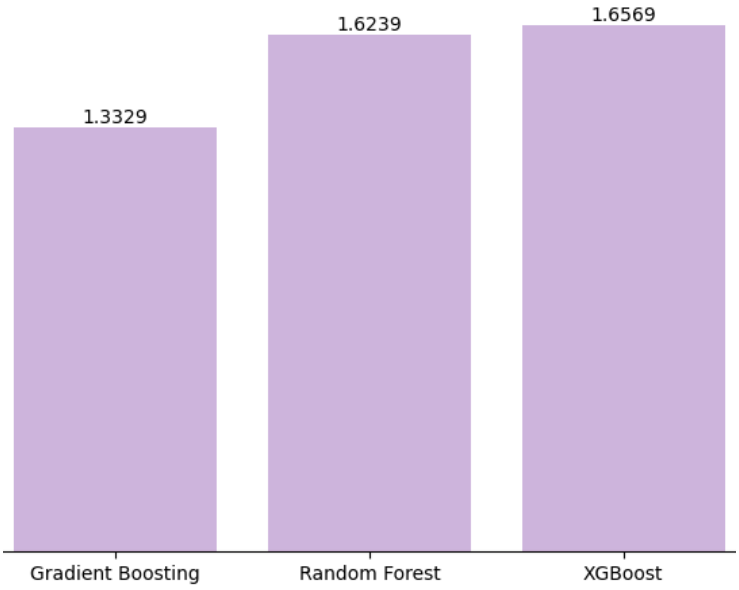


Figura 4-4: Gráfico comparativo de la validación cruzada de los tres modelos evaluados
Fuente: Elaboración Propia, 2024

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 1.3329, menor al 1.6239 de Random Forest, lo que reafirma su consistencia en diferentes subconjuntos de datos. Por otro lado, XGBoost muestra el mayor error (1.6569), 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. A continuación, se presenta una tabla comparativa (Tabla 4-1) que resume las métricas de evaluación para cada modelo analizado, permitiendo una comparación directa de su desempeño y facilitando la selección del modelo más adecuado para este proyecto

Modelo	MAE	MSE	R^2 Score	Validación Cruzada
Gradient Boosting Regressor	1.3072	12.7548	0.8290	1.3329
Random Forest Regressor	0.8804	10.3881	0.8545	1.6239
XGBoost Regressor	1.3639	25.3756	0.6599	1.6569

Tabla 4-1: Tabla comparativa del rendimiento de los modelos regresores
Fuente: Elaboración Propia, 2024

- **Fortalezas y Debilidades de Gradient Boosting Regressor**

A continuación, se presenta una tabla que resalta las fortalezas y debilidades del modelo Gradient Boosting Regressor en función de las métricas obtenidas, evidenciando que sus puntos fuertes superan a las debilidades:

Fortalezas	Debilidades
Su MAE (1.3072) y MSE (12.7548) 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 baja.
Tiene un R^2 Score alto (0.8290), lo que significa que explica más del 82% de la variabilidad en los tiempos de resolución, asegurando un ajuste robusto.	
En la validación cruzada (1.3329), tiene el mejor desempeño generalizado, lo que indica que es menos propenso a sobre ajustarse y funciona de manera confiable con datos nuevos.	

Tabla 4-2: Tabla de fortalezas y debilidades del modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

- **Fortalezas y Debilidades de Random Forest Regressor**

En cuanto a Random Forest se presencia un equilibrio en las fortalezas y debilidades:

Fortalezas	Debilidades
Es el modelo más preciso en términos de MAE (0.8804) y MSE (10.3881), lo que significa que sus predicciones están más cerca de los valores reales.	En la validación cruzada, su MAE (1.6239) 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 (0.8545), indicando que explica un 85% 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 4-3: Tabla de fortalezas y debilidades del modelo Random Forest Regressor
Fuente: Elaboración Propia, 2024

- **Fortalezas y Debilidades de XGBoost Regressor**

A continuación, se presenta una tabla que resalta las fortalezas y debilidades del modelo Gradient Boosting Regressor en función de las métricas obtenidas, evidenciando que sus debilidades superan a las fortalezas:

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 (1.3639) y MSE (25.3756), lo que indica un mayor error promedio y un manejo deficiente de errores grandes.
	Su R^2 Score (0.6599) 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 (1.6569), confirmando que no generaliza bien y es menos confiable en datos nuevos.

Tabla 4-4: Tabla de fortalezas y debilidades del modelo XGBoost Regressor

Fuente: Elaboración Propia, 2024

Tras una evaluación detallada, se concluyó que el Gradient Boosting Regressor mostró un rendimiento destacado, sobresaliendo por su capacidad para identificar patrones complejos en los datos y reducir los errores de predicción. Este modelo fue seleccionado como el más adecuado para el problema en cuestión, ya que ofrece un equilibrio entre exactitud, eficiencia computacional y, lo más importante, capacidad de generalización.

La capacidad de generalización es el factor clave para su elección, dada la naturaleza del problema. Dado que los datos se actualizan constantemente, con nuevos conjuntos de datos cada semana o sprint, es fundamental que el modelo sea capaz de adaptarse a estos cambios y hacer predicciones confiables en tiempo real. La generalización asegura que el modelo no se limite a los datos previos, sino que mantenga su rendimiento incluso con información nueva y no vista anteriormente.

Si no fuera por esta necesidad de adaptación continua a nuevos datos, el Random Forest Regressor podría haber sido la mejor opción, debido a sus excelentes métricas de rendimiento. Sin embargo, la habilidad del Gradient Boosting Regressor para generalizar adecuadamente en un entorno dinámico lo convierte en la opción más adecuada para ser integrado en plataformas como JIRA, donde los datos se actualizan regularmente.

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.

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.

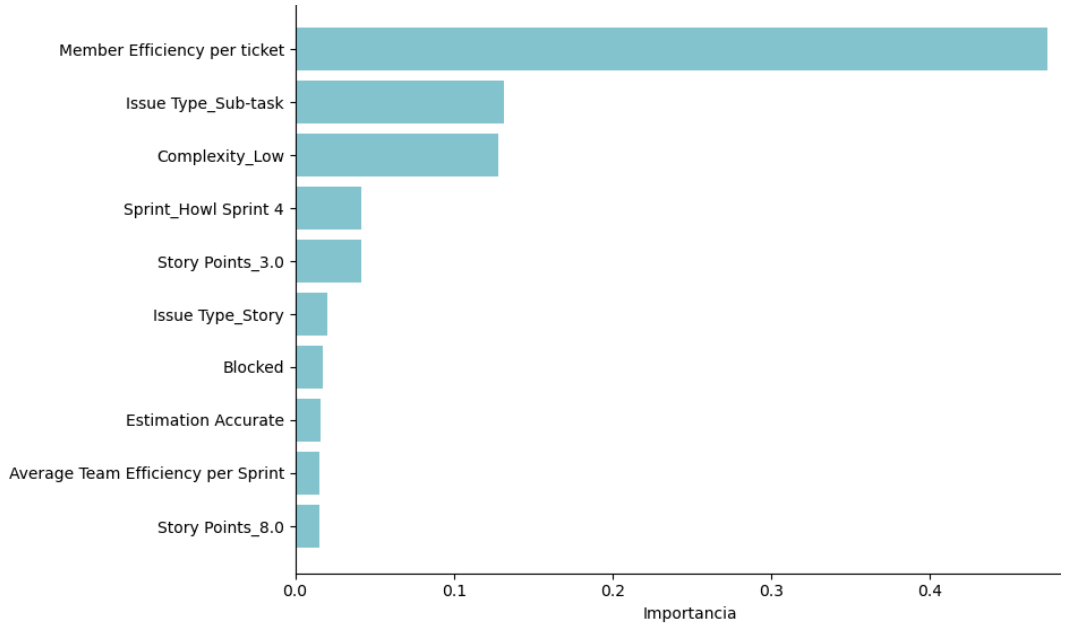


Figura 4-5: Top 10 de las variables más importantes del modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

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

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 4-5), 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): 6.495037195881937
```

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 4-5: 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 1.3072 días, lo que representa una reducción significativa del error promedio. En términos porcentuales, el modelo logró una reducción del **79.87%** del MAE en comparación con las estimaciones manuales, superando ampliamente el objetivo planteado.

```
mae_model = 1.3072  # MAE del modelo
improvement = ((mae_base - mae_model) / mae_base) * 100
print(f"Reducción del MAE: {improvement:.2f}%")
```

```
Reducción del MAE: 79.87%
```

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

La reducción del 79.87% en el MAE (Figura 4-7) demuestra que el modelo predictivo no solo cumple con el objetivo del proyecto, sino que también establece un estándar significativamente más bajo en el error absoluto medio para la estimación de los tiempos de resolución en entornos ágiles. Este nivel de exactitud en las predicciones 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 1.3072 días, el modelo demuestra su capacidad para reducir significativamente el error en 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 4-9, se observa cómo las predicciones del modelo (para un subconjunto de 20 incidencias) se alinean estrechamente con los valores reales, lo que demuestra la capacidad del modelo para realizar predicciones precisas en esta tarea.

La Figura 4-8 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 exactitud 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.

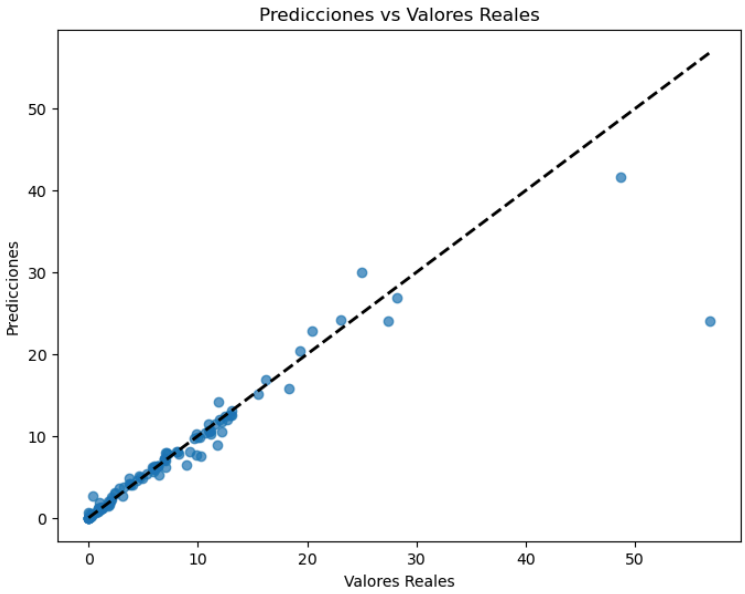


Figura 4-8: Relación entre Predicciones del Modelo y Valores Reales de Tiempo de Resolución
Fuente: Elaboración Propia, 2024

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 1.3072, lo que lo convierte en una herramienta confiable para optimizar la gestión de incidencias.

Valores Reales	Valores Predichos	Diferencia Absoluta	Porcentaje de Acierto
13.045678	13.043473	0.002206	99.983093
6.060243	6.067752	0.007510	99.876085
6.995511	7.004487	0.008975	99.871697
1.147588	1.149424	0.001836	99.839988
6.280846	6.270098	0.010748	99.828877
12.479970	12.453294	0.026676	99.786252
1.038688	1.041476	0.002787	99.731662
11.990054	11.953676	0.036378	99.696602
8.065104	8.024562	0.040542	99.497317
6.305160	6.269790	0.035370	99.439031
6.321908	6.286167	0.035741	99.434646
4.038767	4.065463	0.026696	99.339013
1.785343	1.773331	0.012012	99.327166
9.650792	9.722816	0.072024	99.253699
9.971505	9.891116	0.080390	99.193805
5.371150	5.313348	0.057803	98.923827
11.596643	11.470295	0.126348	98.910479
6.103489	6.178200	0.074711	98.775930
8.187905	8.082067	0.105838	98.707381
5.909637	5.986095	0.076458	98.706221

Figura 4-9: Predicciones del Tiempo de Resolución de Incidencias por el modelo Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

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

Este proyecto evaluó tres modelos clave de Machine Learning para la estimación de esfuerzo en proyectos ágiles: Gradient Boosting Regressor (GBR), Random Forest Regressor (RFR) y XGBoost Regressor. Tras comparar el rendimiento de los modelos utilizando métricas como MAE, MSE y R^2, se determinó que Gradient Boosting es el modelo más adecuado debido a su considerable capacidad de generalización. Sin embargo, Random Forest mostró un mejor rendimiento en cuanto a MAE, MSE y R^2, lo que sugiere que este modelo es particularmente eficaz en la estimación de esfuerzo en escenarios con datos ruidosos o no lineales.

El artículo de Abdelali et al. (2019), "Investigating the use of Random Forest in Software Effort Estimation", profundiza en el uso de Random Forest en la estimación de esfuerzo, señalando que la precisión de las estimaciones depende de factores como el número de árboles y los atributos seleccionados para entrenar el modelo. La investigación también demuestra que Random Forest supera a modelos tradicionales, como los árboles de regresión, en todas las métricas evaluadas, lo que resalta su robustez en tareas de estimación de esfuerzo. Estos hallazgos son consistentes con los resultados obtenidos en este proyecto, donde Random Forest destacó en cuanto a MAE y otras métricas de evaluación, confirmando su potencial en la estimación del esfuerzo en proyectos ágiles.

Por otro lado, el estudio de Rahman et al. (2023), "Software Effort Estimation using Machine Learning Techniques", también exploró diversas técnicas de Machine Learning, incluidos Random Forest y Gradient Boosting, y comparó su rendimiento utilizando MAE, MSE y R^2. Este estudio también encontró que los modelos basados en árboles, como Random Forest, son altamente efectivos en la estimación del esfuerzo, mostrando un buen equilibrio entre precisión y capacidad de generalización. Aunque en este proyecto Gradient Boosting mostró una ligera ventaja en términos de generalización, los resultados de Random Forest fueron igualmente sólidos y efectivos.

En cuanto a XGBoost, el artículo de Sousa et al. (2023), "Applying Machine Learning to Estimate the Effort and Duration of Individual Tasks in Software Projects", destacó la eficacia de los modelos de ensamblaje como XGBoost para capturar patrones complejos y mejorar la precisión en la estimación del esfuerzo. Sin

embargo, aunque XGBoost mostró buenos resultados en su estudio, los resultados de este proyecto indicaron que Random Forest y Gradient Boosting fueron más adecuados para el problema específico de estimación de tiempo de resolución de incidencias en entornos ágiles. Este fenómeno podría deberse a la necesidad de un ajuste de hiper parámetros más exhaustivo en XGBoost, lo que puede haber afectado su rendimiento comparativo en nuestro estudio.

4.7.1. 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.

4.7.2. 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 exactitud.
- **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.

En resumen, los resultados obtenidos en este proyecto coinciden con las conclusiones de los artículos revisados, que destacan la efectividad de los modelos basados en árboles, como Random Forest y Gradient Boosting, para la estimación de esfuerzo en proyectos de software. Sin embargo, Gradient Boosting fue seleccionado como el modelo más adecuado debido a su capacidad de generalización y flexibilidad. Estos hallazgos refuerzan la importancia de elegir el modelo adecuado según el tipo de datos y el problema específico a resolver, lo que en este caso resalta la ventaja de Gradient Boosting para estimaciones de tiempos en proyectos ágiles.

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 79.87%. El modelo obtuvo un MAE de 1.3072 días, un MSE de 12.7548 días, un coeficiente de determinación (R^2) de 82.90% y un MAE de 1.3329 en la validación cruzada, lo que demuestra su capacidad para capturar las complejas relaciones entre las variables involucradas en la resolución de incidencias. Estos resultados reflejan una considerable exactitud, eficacia y capacidad de generalización en las predicciones, consolidando al Gradient Boosting como la mejor opción para este proyecto.

En comparación con otros trabajos similares, las conclusiones de estos estudios respaldan los resultados obtenidos en este proyecto. Los modelos basados en árboles de decisión se presentan como opciones altamente efectivas para la estimación de esfuerzo y duración (tiempo de resolución) en el contexto del desarrollo de software. Esto refuerza la idea de que estos enfoques son particularmente adecuados para hacer predicciones precisas en este ámbito.

El proyecto también resalta la importancia de la optimización de los hiper parámetros para mejorar la exactitud 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:

Incorporar datos de múltiples equipos y proyectos para mejorar la robustez y la capacidad de generalización del modelo.

Analizar otras características potencialmente relevantes, como el historial de bloqueos o el tamaño del equipo, lo que podría mejorar aún más la exactitud de las predicciones.

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.

Aunque Gradient Boosting ha demostrado un buen rendimiento, se recomienda seguir optimizándolo y recalibrándolo periódicamente con nuevos datos para mantener su exactitud y adaptabilidad a cambios en el equipo y el proyecto.

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.

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.

Implementar un pipeline que permita actualizar el modelo con datos nuevos, asegurando que su rendimiento se mantenga en entornos dinámicos.

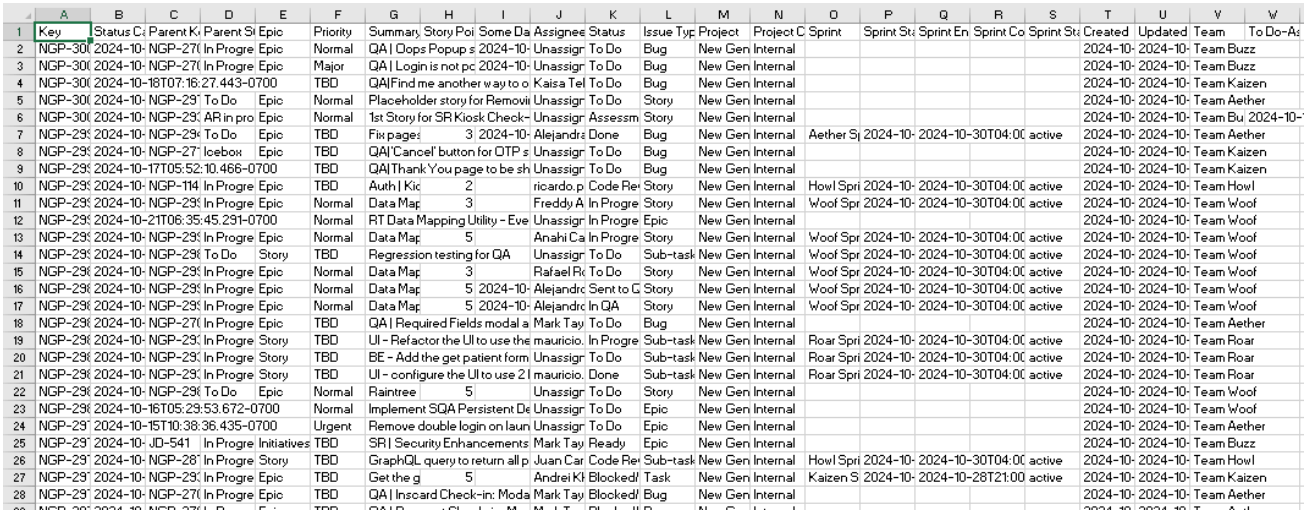
Aunque este estudio se centró en el equipo de desarrollo de software, los enfoques y modelos pueden aplicarse a otros sectores, como marketing, recursos humanos o manufactura, para mejorar la planificación y optimizar la toma de decisiones.

Estas recomendaciones tienen el potencial de optimizar diversas etapas del proceso, desde el entrenamiento del modelo hasta su implementación práctica. Mejorar la exactitud 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.

7. Anexos

Anexo 1: Conjunto original de datos

- **Ubicación en CD:** /DATA/issues_history.csv
- **URL:** https://drive.google.com/file/d/1dUF4BnF2Tr4TXCan92FtjG3_TB5r5Hcn/view?usp=drive_link



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Key	Status C	Parent K	Parent St	Epic	Priority	Summary	Story Poi	Some Da	Assignee	Status	Issue Typ	Project	Project C	Sprint	Sprint Sta	Sprint En	Sprint Co	Sprint Sta	Created	Updated	Team	To Do-As
2	NGP-30(	2024-10-	NGP-27(	In Progre	Epic	Normal	QA | Oops Popup s		2024-10-	Unassigr	To Do	Bug	New Gen	Internal						2024-10-	2024-10-	Team Buzz	
3	NGP-30(	2024-10-	NGP-27(	In Progre	Epic	Major	QA | Login is not pc		2024-10-	Unassigr	To Do	Bug	New Gen	Internal						2024-10-	2024-10-	Team Buzz	
4	NGP-30(	2024-10-18T07:16:27.443-0700				TBD	QA|Find me another way to o			Kaisa Tel	To Do	Bug	New Gen	Internal						2024-10-	2024-10-	Team Kaizen	
5	NGP-30(	2024-10-	NGP-29	To Do	Epic	Normal	Placeholder story for Removi			Unassigr	To Do	Story	New Gen	Internal						2024-10-	2024-10-	Team Aether	
6	NGP-30(	2024-10-	NGP-29	AR in pro	Epic	Normal	1st Story for SR Kiosk Check-			Unassigr	Assessm	Story	New Gen	Internal						2024-10-	2024-10-	Team Bu	2024-10-
7	NGP-29	2024-10-	NGP-29	To Do	Epic	TBD	Fix pages	3	2024-10-	Alejandra	Done	Bug	New Gen	Internal	Aether S	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Aether	
8	NGP-29	2024-10-	NGP-27	Icebox	Epic	TBD	QA|'Cancel' button for OTP s			Unassigr	To Do	Bug	New Gen	Internal						2024-10-	2024-10-	Team Kaizen	
9	NGP-29	2024-10-17T05:52:10.466-0700				TBD	QA|Thank You page to be sh			Unassigr	To Do	Bug	New Gen	Internal						2024-10-	2024-10-	Team Kaizen	
10	NGP-29	2024-10-	NGP-114	In Progre	Epic	TBD	Auth | Kic	2		ricardo.p	Code Rev	Story	New Gen	Internal	Howl Spri	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Howl	
11	NGP-29	2024-10-	NGP-29	In Progre	Epic	Normal	Data Map	3		Freddy A	In Progre	Story	New Gen	Internal	Woof Spr	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Woof	
12	NGP-29	2024-10-21T06:35:45.291-0700				Normal	RT Data Mapping Utility - Eve			Unassigr	In Progre	Epic	New Gen	Internal						2024-10-	2024-10-	Team Woof	
13	NGP-29	2024-10-	NGP-29	In Progre	Epic	Normal	Data Map	5		Anahi Ca	In Progre	Story	New Gen	Internal	Woof Spr	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Woof	
14	NGP-29	2024-10-	NGP-29	To Do	Story	TBD	Regression testing for QA			Unassigr	To Do	Sub-task	New Gen	Internal	Woof Spr	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Woof	
15	NGP-29	2024-10-	NGP-29	In Progre	Epic	Normal	Data Map	3		Rafael Re	To Do	Story	New Gen	Internal	Woof Spr	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Woof	
16	NGP-29	2024-10-	NGP-29	In Progre	Epic	Normal	Data Map	5	2024-10-	Alejandra	Sent to C	Story	New Gen	Internal	Woof Spr	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Woof	
17	NGP-29	2024-10-	NGP-29	In Progre	Epic	Normal	Data Map	5	2024-10-	Alejandra	In QA	Story	New Gen	Internal	Woof Spr	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Woof	
18	NGP-29	2024-10-	NGP-27(	In Progre	Epic	TBD	QA | Required Fields modal a			Mark Tay	To Do	Bug	New Gen	Internal						2024-10-	2024-10-	Team Aether	
19	NGP-29	2024-10-	NGP-29	In Progre	Story	TBD	UI - Refactor the UI to use the			mauricio.	In Progre	Sub-task	New Gen	Internal	Roar Spri	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Roar	
20	NGP-29	2024-10-	NGP-29	In Progre	Story	TBD	BE - Add the get patient form			Unassigr	To Do	Sub-task	New Gen	Internal	Roar Spri	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Roar	
21	NGP-29	2024-10-	NGP-29	In Progre	Story	TBD	UI - configure the UI to use 2 I			mauricio.	Done	Sub-task	New Gen	Internal	Roar Spri	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Roar	
22	NGP-29	2024-10-	NGP-29	To Do	Epic	Normal	Raintree	5		Unassigr	To Do	Story	New Gen	Internal						2024-10-	2024-10-	Team Woof	
23	NGP-29	2024-10-16T05:29:53.672-0700				Normal	Implement SQA Persistent De			Unassigr	To Do	Epic	New Gen	Internal						2024-10-	2024-10-	Team Woof	
24	NGP-29	2024-10-15T10:38:36.435-0700				Urgent	Remove double login on laun			Unassigr	To Do	Epic	New Gen	Internal						2024-10-	2024-10-	Team Aether	
25	NGP-29	2024-10-	JD-541	In Progre	Initiatives	TBD	SR | Security Enhancements			Mark Tay	Ready	Epic	New Gen	Internal						2024-10-	2024-10-	Team Buzz	
26	NGP-29	2024-10-	NGP-28	In Progre	Story	TBD	GraphQL query to return all p			Juan Car	Code Rev	Sub-task	New Gen	Internal	Howl Spri	2024-10-	2024-10-30T04:00	active		2024-10-	2024-10-	Team Howl	
27	NGP-29	2024-10-	NGP-29	In Progre	Epic	TBD	Get the g	5		Andrei Kl	Blocked/	Task	New Gen	Internal	Kaizen S	2024-10-	2024-10-28T21:00	active		2024-10-	2024-10-	Team Kaizen	
28	NGP-29	2024-10-	NGP-27(	In Progre	Epic	TBD	QA | Inscard Check-in: Moda			Mark Tay	Blocked/	Bug	New Gen	Internal						2024-10-	2024-10-	Team Aether	

Figura 7-1: Conjunto original de datos

Fuente: Elaboración propia, basada en los datos proporcionados por el cliente a través de la API de JIRA (Atlassian, 2024).

Anexo 2: Código JS para la extracción de los datos de JIRA

```
const fetchAllIssues = async () => {
  let startAt = 0;
  const maxResults = 50;
  let allIssues = [];
  let total = 0;

  do {
    const response = await axios.get(baseUrl, {···
    });

    allIssues = allIssues.concat(response.data.issues);
    startAt += maxResults;
    total = response.data.total;
  } while (startAt < total);

  return allIssues;
};

const checkCredentials = async () => {
  try {
    const allIssues = await fetchAllIssues();
    fs.writeFileSync(
      "jira_data.json",
      JSON.stringify(allIssues, null, 2)
    );
    console.log("JSON file was written successfully");
  } catch (error) {
    if (error.response && error.response.status === 401) {
      console.log("Credenciales inválidas.");
    } else {
      console.error("Error al verificar credenciales:", error);
    }
  }
};
```

Figura 7-2: Código JS para la extracción de los datos de JIRA

Fuente: Elaboración Propia, 2024

Anexo 3: Código para la conversión de JSON a CSV

```
function jsonToCSV(jsonData, outputFilePath) {
  const jsonArray = JSON.parse(jsonData);

  const headers = [···
  ];

  const dynamicHeaders = new Set();
  jsonArray.forEach((issue) => {
    if (issue.changelog && issue.changelog.histories) {
      issue.changelog.histories.forEach((history) => {
        history.items.forEach((item) => {
          if (item.field === "status") {
            const headerId = `${item.fromString}-${item.toString}`;
            dynamicHeaders.add(headerId);
          }
        });
      });
    }
  });

  dynamicHeaders.forEach((header) => {
    headers.push({ id: header, title: header });
  });

  const csvWriter = createCsvWriter({
    path: outputFilePath,
    header: headers,
  });

  const issues = jsonArray.map((issue) => {···
  });

  csvWriter
    .writeRecords(issues)
    .then(() => {
      console.log("CSV file was written successfully");
    })
    .catch((err) => {
      console.error("Error writing CSV file:", err);
    });
}
```

Figura 7-3: Código para la conversión de JSON a CSV
Fuente: Elaboración Propia, 2024

Anexo 4: Conjunto de datos normalizados

- **Ubicación en CD:** /DATA/issues_history_ready.csv
- **URL:** https://drive.google.com/file/d/1StaESujMzrZKRR-s4p4pD_H81tkGym00/view?usp=drive_link

Key	Resolution	Blocked	Member E	Sprint Ove	Days to Sp	Total Stor	Total Stor	Average Ti	Block Rate	Estimatior	Parent Ke	Parent Ke	Parent Ke	Parent Ke	Parent Ke	Parent Ke	Parent Ke
NGP-1035	15.3463	1	3.27E-05	1	0.81818	0.3871	0.33333	0.00638	0.5	0	0	1	0	0	0	0	0
NGP-1036	14.5415	0	5.52E-05	1	0.81818	0.47312	0.45455	0.49286	0	0	0	0	0	0	0	0	0
NGP-1037	6.44919	0	0.00012	1	0.81818	0.47312	0.45455	0.49286	0	1	0	0	0	0	0	0	0
NGP-1042	9.45349	1	5.30E-05	0	0.90909	0.30108	0.18182	0.00696	0.33333	0	0	0	0	0	0	0	0
NGP-1043	14.1721	1	3.54E-05	0	0.90909	0.96774	1	0.64865	0.44444	0	0	0	0	0	0	0	0
NGP-1048	3.94844	0	7.62E-05	0	0.90909	0.30108	0.18182	0.00696	0.33333	0	0	0	0	0	0	0	0
NGP-1092	15.4741	0	5.18E-05	0	0.90909	0.96774	1	0.64865	0.44444	0	0	0	0	0	0	0	0
NGP-1093	33.2582	0	3.92E-05	1	0.81818	0.3871	0.33333	0.00638	0.5	1	0	1	0	0	0	0	0
NGP-1107	19.7392	0	4.06E-05	1	0.81818	0.50538	0.22727	0.00078	0	0	0	0	0	0	0	0	0
NGP-1112	25.0131	1	3.21E-05	1	0.72727	1	0.60606	0.79882	0.75	0	0	0	0	0	0	0	0
NGP-1114	3.78234	0	0.00013	0	0.90909	0.17204	0.16667	0.0044	0	1	0	0	0	0	0	0	0
NGP-1407	48.0187	0	6.26E-06	1	0.72727	0.35484	0.07576	0.00034	0	0	0	0	0	0	0	0	0
NGP-1416	15.0323	0	2.00E-05	0	0.90909	0.30108	0.27273	0.00134	0	0	0	0	0	0	0	0	0
NGP-1417	5.62338	0	3.57E-05	0	0.90909	0.30108	0.13636	0.00091	0	0	0	0	0	0	0	0	0
NGP-1420	4.98581	0	0.00016	0	0.90909	0.30108	0.15152	0.00091	0	1	0	0	0	0	0	0	0
NGP-1452	12.48	0	2.41E-05	1	0.81818	0.29032	0.19697	0.0066	0.66667	0	0	1	0	0	0	0	0
NGP-1474	14.8301	0	5.41E-05	0	0.90909	0.44086	0.22727	0.00233	0	0	0	0	0	0	0	0	0
NGP-1488	22.0803	0	1.36E-05	1	0.81818	0.37634	0.27273	0.00033	0	0	0	0	0	0	0	0	0
NGP-1525	13.9293	0	2.16E-05	1	0.81818	0.05376	0.0303	0.00106	0	0	0	0	0	0	0	0	0

Figura 7-4: Conjunto de datos normalizados
Fuente: Elaboración Propia, 2024

Anexo 5: Código que asigna los correspondientes tipos a las columnas del conjunto de datos

```
# 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 7-5: Código que asigna los correspondientes tipos a las columnas del conjunto de datos
Fuente: Elaboración Propia, 2024

Anexo 6: Código que transforma las columnas de transición a filas

```
# 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 7-6: Código que transforma las columnas de transición a filas
Fuente: Elaboración Propia, 2024

Anexo 7: Código para calcular la exactitud de una estimación

```
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 7-7: Código para calcular la exactitud de una estimación
Fuente: Elaboración Propia, 2024

Anexo 8: Búsqueda de hiper parámetros óptimos para Gradient Boosting Regressor

```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 7-8: Búsqueda de hiper parámetros óptimos para Gradient Boosting Regressor
Fuente: Elaboración Propia, 2024

Anexo 9: Búsqueda de hiper parámetros óptimos para Random Forest Regressor

```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}")
```

```
Mejores Hiper parametros: {'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 2,
'max_features': 1.0, 'max_depth': 10}
```

Figura 7-9: Búsqueda de hiper parámetros óptimos para Random Forest Regressor
Fuente: Elaboración Propia, 2024

Anexo 10: Búsqueda de hiper parámetros óptimos para XGBoost Regressor

```
# 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, 'reg_alpha': 0, 'n_estimators': 100,
'max_depth': 3, 'learning_rate': 0.2, 'gamma': 0.1, 'colsample_bytree': 1.0}
```

Figura 7-10: Búsqueda de hiper parámetros óptimos para XGBoost Regressor
Fuente: Elaboración Propia, 2024

Anexo 11: Código completo de limpieza

- **Ubicación en CD:** /NOTEBOOK FILES/1_cleaning_process.ipynb
- **Google Colab link:** https://drive.google.com/file/d/1TUAUSKZ7oOY-4AskNyW4_uaczGPyA9o0/view?usp=drive_link

Anexo 12: Código completo del análisis exploratorio y el feature engineering

- **Ubicación en CD:** /NOTEBOOK FILES/2_eda_feature_engineering.ipynb
- **Google Colab link:** https://drive.google.com/file/d/1cf7YdvhY6BUfYIfrj9C9wjVGGz3hLUGv/view?usp=drive_link

Anexo 13: Código completo del pre procesamiento del modelo

- **Ubicación en CD:** /NOTEBOOK FILES/3_pre_model_process.ipynb
- **Google Colab link:** https://drive.google.com/file/d/1JUfsHZ9tqI_64kPefUyVcTfZ38MKWRFO/view?usp=drive_link

Anexo 14: Código completo para entrenamiento y la evaluación

- **Ubicación en CD:** /NOTEBOOK FILES/4_training_Evaluation.ipynb
- **Google Colab link:** https://drive.google.com/file/d/1yeZVDLcnngL6IjhfvZUI3YXRg1vhc4dd/view?usp=drive_link

Anexo 15: Código del diseño de implementación del modelo en JIRA

- **Ubicación en CD:** /NOTEBOOK FILES/5_implementation_plan.ipynb
- **Google Colab link:** https://drive.google.com/file/d/1KbMwcMV5rZkTB0ZRIftIyUZmgJikQ0wl/view?usp=drive_link

Anexo 16: Repositorio Github del proyecto

- https://github.com/anahicallejas/PROYECTO_FINAL.git

Código QR para ingresar al repositorio del proyecto

Referencias Bibliográficas

Abdelali, Z., Mustapha, H., & Abdelwahed, N. (2019). Investigating the use of random forest in software effort estimation. Procedia Computer Science, 148, 343-352. https://doi.org/10.1016/j.procs.2019.01.042

Amat Rodrigo, J. (2015). Árboles de decisión, Random Forest, Gradient Boosting y C5.0. Cienciadedatos.net. Recuperado el 10 de diciembre de 2024, de https://www.cienciadedatos.net/documentos/33_arboles_decision_random_forest_gradient_boosting_C50.html

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

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

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

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

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

Atlassian. (2024). Introducción a las incidencias de Jira Software | Atlassian. https://www.atlassian.com/es/software/jira/guides/issues/overview#what-are-issue-types

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/

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

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

Chen, T., & Guestrin, C. (2016, 8 agosto). XGBoost. 785-794. https://doi.org/10.1145/2939672.2939785

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

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.

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

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

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

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

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

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

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

Phan, H., & Jannesari, A. (2022, septiembre). Heterogeneous graph neural networks for software effort estimation. En Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 103-113).

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.

Rahman, M., Roy, P. P., Ali, M., Gonc, T., & Sarwar, H. (2023). Software effort estimation using machine learning technique. International Journal of Advanced Computer Science and Applications, 14(4).

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

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

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

Sánchez, E. R., Vázquez-Santacruz, E., & Maceda, H. C. (2022). Estimación de esfuerzo en desarrollo de software ágil utilizando redes neuronales artificiales. Res. Comput. Sci., 151(7), 77-91.

Schwaber, K., & Beedle, M. (2002). Agile Software Development with Scrum. Pearson.

Sousa, A. O., Veloso, D. T., Gonçalves, H. M., Faria, J. P., Mendes-Moreira, J., Graça, R., Gomes, D., Castro, R. N., & Henriques, P. C. (2023). Applying Machine Learning to Estimate the Effort and Duration of Individual Tasks in Software Projects. IEEE Access, 11, 89933-89946. https://doi.org/10.1109/access.2023.3307310

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

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