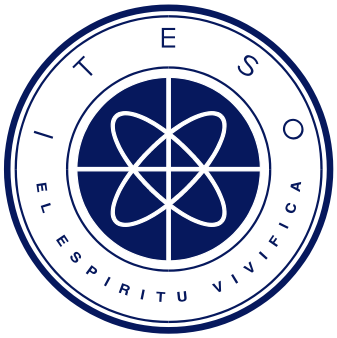
Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Procesos Tecnológicos e Industriales

MAESTRÍA EN INGENIERÍA DE PRODUCTOS Y PROCESOS



SISTEMA DE APRENDIZAJE AUTOMÁTICO PARA LA PREDICCIÓN DEL AVANCE DE FERMENTACIÓN EN CERVECERÍA ARTESANAL

Trabajo recepcional que para obtener el grado de

MAESTRO EN INGENIERÍA DE PRODUCTOS Y PROCESOS

Presenta: Ernesto Sebastián Hermosillo Díaz

Tutor: Mtro. Fermín Castro Aragón

San Pedro Tlaquepaque, Jalisco. 04 de diciembre de 2023.

Tabla de Contenidos

Agradecimientos

Este trabajo está dedicado a mi madre, que en paz descanse, que me enorgullece que haya alcanzado a verme iniciar los primeros meses de este camino.

Agradezco profundamente al ITESO, mi *alma mater*, y comité de posgrados por la oportunidad que me brindaron al haberme otorgado una beca y, además, el haberme permitido extenderme en una materia electiva, Aprendizaje Automático (*Machine Learning*), que tuvo impacto directo en este proyecto sin importar que esto extendiera mi trayecto.

También agradezco a mis profesores, coordinadores y tutores del posgrado que frente a la incertidumbre política que impactaba al que fue mi proyecto inicial me alentaron a redireccionar mi Trabajo de Obtención de Grado hacia mi pasión y cruzar sin miedo a otras áreas del conocimiento, haciendo de este reto algo muy especial.

Por último, a Cerveza Loba por su apertura a colaborar en este proyecto piloto por el beneficio de toda la industria cervecera artesanal, y espero que las semillas plantadas aquí florezcan en más colaboraciones y mejoras para el sector.

Resumen

Tradicionalmente, el monitoreo del avance de fermentación de cerveza artesanal ha estado limitado a mediciones esporádicas fuera de línea de densidad de mosto y de temperatura empleadas solo para retroalimentar sistemas de control simples, sin generar un análisis histórico, en parte debido a restricciones tecnológicas y financieras en la industria a pequeña escala. En consecuencia, las variaciones no detectadas de temperatura impactan la consistencia entre lotes y son una causa de pérdida de producto. Este estudio documenta la intervención en Cerveza Loba en Guadalajara, Jalisco, donde se implementó un sistema basado en la nube para el monitoreo de temperaturas de fermentación en tanque con costo inferior a 150 dólares americanos (*USD*). Este sistema almacena datos en la nube permitiendo la visualización remota en tiempo real, y aprovecha el Aprendizaje Automático fuera de línea para permitir la predicción de la densidad del mosto y la identificación de desviaciones de temperatura, así como la clasificación de etapas y la predicción de la finalización del proceso de fermentación.

Palabras clave: Aprendizaje Automático, fermentación, cerveza artesanal

Abstract

Traditionally, the monitoring of fermentation progress in craft beer production has been constrained to sporadic offline measurements of wort density and temperature. These measurements have primarily fed simple control systems and do not offer real-time tracking, remote visualization, or historical analysis, partly due to technological and financial limitations in small-scale industries. As a result, undetected temperature variations in fermentation tanks are a leading cause of product loss and a key factor in batch inconsistency in craft breweries. This study documents the results of an intervention at Cerveza Loba in Guadalajara, Jalisco, where a low-cost cloud-based solution was implemented for real-time monitoring of fermentation tanks temperature at a cost below 150 USD. This system not only acquires but also stores data in the cloud, allowing for real-time remote visualization, it also leverages offline Machine Learning to enable prediction of wort density and identification of temperature deviations, as well as stage classification and forecasting of fermentation process completion.

Key words: *machine learning, fermentation, craft beer*

1. Fundamentación del trabajo

1.1. Identificación y caracterización del problema a atender

La industria de la cervecería artesanal, debido a su escala menor en comparación con la producción industrial, precisa de implementación de sistemas de visualización y control integrado para sus distintos procesos, como cocimiento, fermentación y maduración. A diferencia de las grandes cervecerías, las micro-cervecerías no cuentan con instrumentos con conectividad ni con personal operativo para supervisar en todo momento su operación, lo que dificulta el monitoreo efectivo de parámetros clave para mantener la consistencia deseada, evitar desviaciones de proceso y optimizar sus procesos.

Como ejemplo, las variaciones no detectadas de temperatura en los tanques de fermentación, atribuibles a factores técnicos, ambientales y humanos, que pueden causar la pérdida de lotes que acumulados pueden representar hasta el 2% de la producción anual (E. Aceves-Vincent Ramírez, comunicación personal, 28 de septiembre de 2023). El gremio de cerveceros artesanales mexicanos necesita una solución tecnológica de bajo costo para visualizar de manera remota parámetros clave del proceso, identificar tempranamente desviaciones, obtener información valiosa sobre el histórico de sus fermentaciones, y generar notificaciones de alerta para enfrentar imprevistos en la producción.

1.2. Contexto de la propuesta de intervención

1.2.1. Contexto de la empresa

Fundada por Alejandro Magallanes, Cerveza Loba comenzando sus operaciones en 2011 con el objetivo de elaborar cerveza de alta calidad en México y competir en el mercado internacional. En 2015, estableciéndose en el corazón de Guadalajara, en un lugar que alberga la cervecería, oficinas y el restaurante Loba Gastropub, donde los visitantes pueden probar diversos estilos de cerveza. El enfoque de la cervecería en la calidad y la excelencia en sus productos les ha permitido exportar a mercados competitivos como el Reino Unido y el estado de California, en los Estados Unidos de América, experimentando un crecimiento sostenido en estos mercados. Esta dedicación ha sido reconocida con varios premios y distinciones.

En 2016, la empresa fue galardonada como la Mejor Cervecería Mediana de México, y en 2019 recibió el título de Mejor Cervecería de Norte América en el *International Beer Challenge*. A lo largo de su trayectoria, han acumulado más de 50 medallas y 5 reconocimientos en distintos países. La operación de la planta, compuesta por siete personas, está liderada por el gerente de operaciones Enrique Aceves-Vincent Ramírez.

1.2.2. Contexto de la industria

La historia moderna de la producción de cerveza en México se remonta a la tercera década del siglo XX, donde para entonces ya existían la Cervecería Cuauhtémoc, Cervecería Modelo y Cervecería Moctezuma (Recio, 2004). A finales de ese siglo, la industria se contrajo, estableciendo el duopolio de Grupo Modelo y Cervecería Cuauhtémoc-Moctezuma, controlada por Fomento Económico Mexicano (FEMSA). En 2010, FEMSA vendería su rama cervecera a Heineken (El Informador, 2010), y en 2013, por su parte se concretaría la venta de Grupo Modelo al conglomerado belga, estadounidense y brasileño Anheuser-Busch InBev (AB InBev); ambas operaciones cerrándose por montos de entre poco más de 7 y 20 mil millones de dólares, respectivamente (CNNExpansión, 2013).

Coincidentemente, mientras la industria cervecera 100% mexicana de gran escala desaparecía, surgía la cervecería artesanal e independiente mexicana con la fundación de Cerveza Minerva en 2003 y Cerveza Loba en 2011, ambas en Jalisco, y de Cervecería de Colima en 2013 en el estado homónimo. Esta primera, la mayor productora de México que a casi 20 años de su fundación emplea a 116 trabajadores y produce 18,000 hectolitros anuales, con crecimientos anuales en producción de doble dígito (Romo, 2022).

A pesar de los desafíos económicos, como pandemia del *SARS-Cov-2* en 2020, el desabasto de insumos clave como botellas de vidrio nacionales y granos malteados de trigo debido a tensiones geopolíticas en 2022, la industria cervecera artesanal mexicana proyectaba un crecimiento de más del 10% para el 2022, según estimaciones de la Asociación Cervecera Mexicana (Acermex). Esta organización, Acermex, además prevé que la producción nacional en 2022 de este sector llegue a los 34 mil hectolitros (34 millones de litros), comparado a los 30 mil hectolitros producidos entre las 1,462

cervecerías independientes reconocidas por la asociación en 2021, el equivalente a 0.8% del volumen de producción total nacional de cerveza y a aproximadamente 100 millones de dólares americanos en ventas (Peláez-Fernández, 2022).

Este gremio ha estado luchando los últimos 12 años para lograr un cambio de políticas públicas, en particular, con una modificación en la ley del Impuesto Especial sobre Producción y Servicios (IEPS), que actualmente tiene una modalidad *ad valorem* referente a una proporción del costo (impuesto basado en el valor de un bien), buscando establecer un esquema *ad quantum*, impuesto en función del contenido alcohólico en las bebidas para hacer más competitiva la producción de cerveza artesanal. Jesús Briseño Gómez, fundador y presidente de Cerveza Minerva, señala que la cerveza artesanal paga hasta 4 veces más el gravamen IEPS por litro que la industrial (Rincón, 2020).

Otro factor decisivo para el éxito de la industria, como lo considera Esteban Silva Ochoa fundador de Cervecería de Colima y directivo relevante en la industria, es la consistencia y no escatimar en la calidad en el producto: enfatizando la importancia de incorporar insumos de alta calidad y tener procesos robustos para minimizar desviaciones respecto al perfil deseado de la bebida (Rodríguez, 2022). Además, una de las principales desventajas para la industria artesanal es el acceso a tecnología con conectividad para la visualización de sus procesos, ya que muchas cervecerías artesanales se ven limitadas a utilizar transmisores de temperatura sin conectividad remota, lo que requiere un monitoreo y control exclusivamente presencial. En este contexto, las cervecerías artesanales mexicanas deberán continuar innovando y buscando soluciones para enfrentar estos desafíos y expandir su presencia en el mercado nacional e internacional.

1.2.3. Análisis causa-efecto

En el siguiente diagrama de Árbol de Problemas se abordan las conexiones entre causas y efectos en torno a un problema central (Figura 1): la industria productora de cerveza artesanal, en gran medida, carece de un sistema de control que permita el monitoreo y visualización remota de los procesos de elaboración, lo que genera una serie de consecuencias negativas, consecuencias secundarias, que afectan la consistencia y márgenes del producto.

Entre las causas principales se encuentran:

- Limitaciones económicas y tecnológicas en la industria a pequeña escala.
- Desconocimiento de soluciones tecnológicas asequibles y efectivas.
- Ignorar importancia de los análisis históricos en la calidad y consistencia del producto.

Estos motivos conducen al problema central de la falta de un sistema de control con monitoreo y visualización remota en la industria cervecera artesanal. A su vez, este desencadenando una serie de sub-efectos negativos:

- Inconsistencia en la calidad y características organolépticas del producto entre lotes.
- Pérdida de producto debido a problemas no detectados en el proceso de fermentación.
- Dificultad para identificar y corregir desviaciones en los procesos de elaboración.
- Mayor vulnerabilidad ante la eficiencia de la competencia macro-industrial.

Abordar estas causas y efectos es fundamental para mejorar la situación inmediata y potencializar a la industria cervecera artesanal, ya que se fomenta una cultura de toma de decisiones basada en datos que permitirá a las cervecerías entender la importancia de los análisis históricos en la calidad del producto, y mejorar su competitividad en el mercado nacional e internacional.

1.2.4. Matriz de marco lógico del problema

Dada la dependencia del monitoreo presencial y las limitantes económicas, la industria cervecera artesanal requiere una solución tecnológica de bajo costo para el monitoreo remoto de sus parámetros de proceso de fermentación. Para abordar este problema, se propone la integración de un sistema que adquiera y transmita datos a la nube (*Cloud database*) para permitir la visualización remota de condiciones en tiempo real, y fuera de línea la aplicación de técnicas de Aprendizaje Automático (*Machine Learning*) para clasificar la etapa y predecir el avance de la fermentación. Asimismo, el sistema permitirá la identificación temprana y alerta sobre desviaciones de proceso, con el fin de reaccionar a imprevistos en la producción y generar información y conocimiento sobre los procesos.

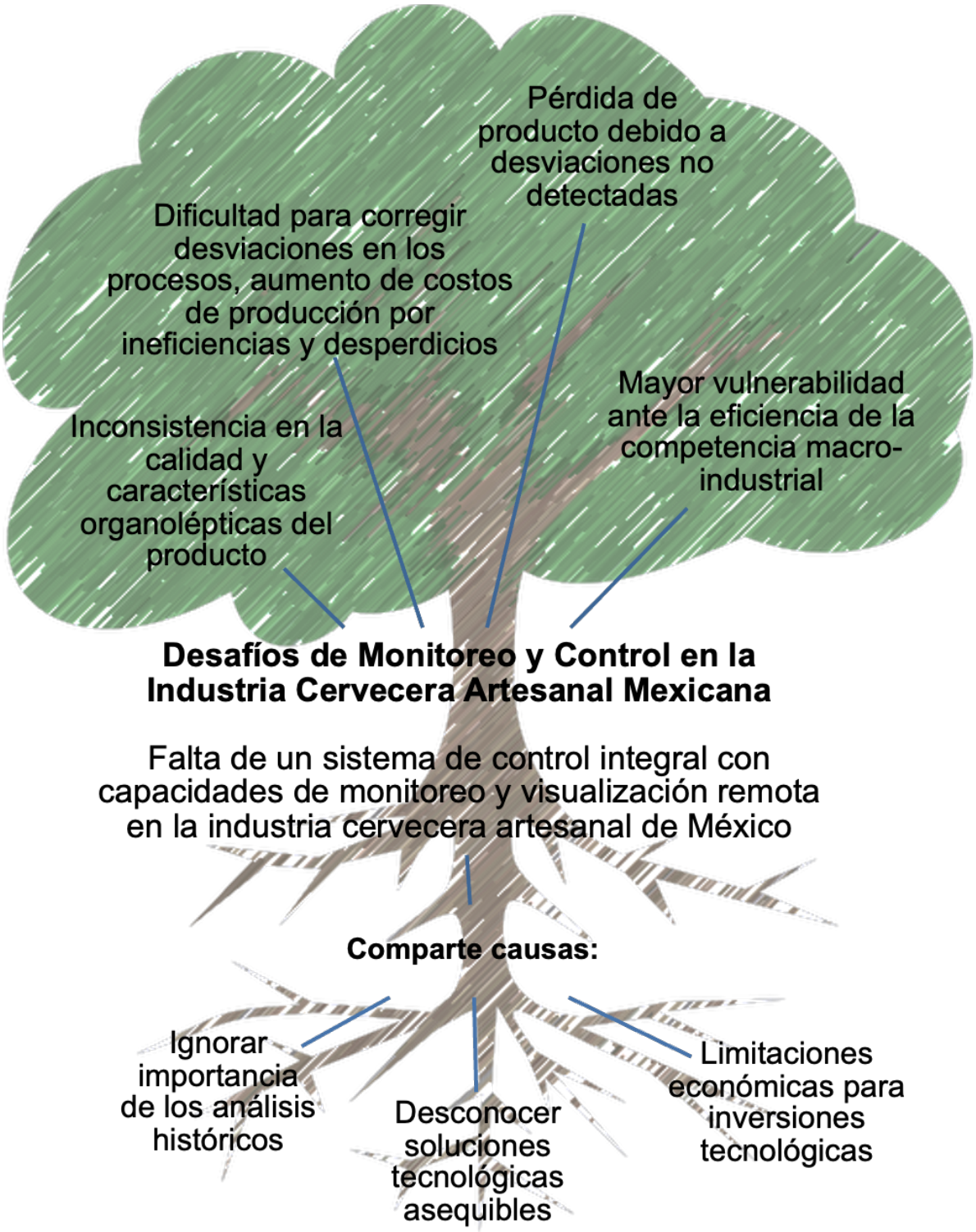


Figura 1. Diagrama de Árbol de Problemas, análisis causa-efecto de la propuesta de intervención.

1.3. Objetivos de la intervención

Desarrollar un sistema de monitoreo remoto para cervecerías artesanales que permita la visualización de procesos en tiempo real. Por medio de la incorporación de herramientas de Aprendizaje Automático, modelos de red neuronal artificial (*ANN*), para predecir el avance y etapa de la fermentación, mejorando así el control operacional de las cervecerías. Los objetivos específicos incluyen:

- Desarrollar un sistema de adquisición y almacenamiento físico de datos y además con conectividad de base de datos en la nube (*Cloud*);
- Integrar un microprocesador, componentes electrónicos y sensores, para instrumentar un prototipo con un costo inferior a 150 dólares americanos por sistema, para garantizar que sea asequible;
- Aplicar técnicas de Aprendizaje Automático (*Machine Learning*) para proporcionar información relevante sobre parámetros del proceso:

    a) Clasificar la etapa de fermentación actual;

    b) Predecir la densidad del mosto;

    c) Proyectar el tiempo de finalización de la conversión del mosto;

    d) Identificar desviaciones en el comportamiento de la fermentación.

1.4. Delimitaciones y área funcional por intervenir

El proyecto busca demostrar la factibilidad técnica y económica de una solución que combine componentes electrónicos, conectividad con la nube e inteligencia artificial para monitorear la etapa de fermentación en cervecerías artesanales. El alcance del proyecto se segmenta en las siguientes fases:

a) Primera fase: Desarrollo de cinco módulos funcionales de *Software* que abarcan:
    i. Adquisición y almacenamiento físico de datos en *Raspberry PI Zero W*;
    ii. Registro de base de datos en la nube de *InfluxDB Cloud*;

iii. Visualización de datos en la nube de *Grafana Cloud*;

iv. Análisis de datos fuera de línea en *Google Colab*;

v. Implementación de algoritmos de Aprendizaje Automático (*Machine Learning*) en *Google Colab*.

El código en *Python* de los módulos funcionales será diseñado para ser altamente interoperable, con el fin de facilitar su integración en diferentes entornos de cervecerías artesanales. Esto incluirá la compatibilidad para integrarse a diferentes plataformas en la nube (*Cloud*) mediante el uso de *APIs* (*Application Programming Interface*).

b) Segunda fase: Implementación de dos sistemas independientes, cada uno con la misma configuración que incluye dos transmisores de temperatura para diferentes tanques de fermentación (cuatro en total). Esta fase se centrará en la integración de los primeros tres módulos funcionales (i a iii): Adquisición de datos, Registro en la nube y Visualización de datos.
c) Tercera fase: Enfocándose en el módulo funcional de Análisis de datos (iv y v). Incluirá la exploración de datos, así como selección, optimización de hiperparámetros y entrenamiento de algoritmos de Aprendizaje Automático.
d) Cuarta fase: Finalmente, la evaluación de la efectividad de las herramientas de Aprendizaje Automático se llevará a cabo, centrándose en el quinto módulo funcional (v).

Para evaluar la efectividad y precisión de las herramientas de Aprendizaje Automático, se utilizarán diversas métricas de evaluación, como Error Cuadrático Medio (*MSE*). Se implementarán conjuntos de datos de validación para probar y validar los modelos. Una vez que el sistema se haya implementado en una configuración controlada, se realizará una fase de validación con datos reales. Esta fase implicará la retroalimentación directa de los cerveceros artesanales y otros operadores del sistema para evaluar su utilidad, eficiencia y facilidad de uso.

1.5. Justificación y pertinencia del trabajo

La intervención propuesta en este proyecto se basa en una combinación de intereses personales y profesionales, buscando una convergencia entre la pasión por la cerveza artesanal, el deseo de apoyar a esta industria en crecimiento, y la fascinación por las aplicaciones prácticas de la inteligencia artificial, al considerarla como el futuro de la tecnología a emplearse en procesos de transformación. Todo esto en el contexto del posgrado en Ingeniería de Productos y Procesos, lo cual ha motivado al autor a explorar e incursionar en áreas de estudio más allá de su quehacer profesional; como la programación para análisis de datos y el Aprendizaje Automático (*Machine Learning*), temas que han sido profundizados a través de materias optativas al plan de estudios del posgrado.

La aplicación de herramientas de análisis de datos y *Machine Learning* en la intervención podría proporcionar múltiples beneficios, como la identificación de etapas o desviaciones en la fermentación (clasificación), la predicción de la densidad en tiempo real (regresión) y la proyección de la finalización del proceso (*forecasting*). Estos avances contribuirían a mejorar la calidad y consistencia del producto, al mismo tiempo que reducirían la intervención humana, el tiempo empleado y la cantidad de lotes perdidos. Asimismo, optimizarían el uso de recursos, equipos y energía (Bowler, Pound, & Watson, *Domain Adaptation and Federated Learning for Ultrasonic Monitoring of Beer Fermentation,* 2021), lo que se traduciría en un aumento de los márgenes para los productores artesanales de cerveza con una inversión de capital relativamente baja. Además, la solución propuesta tiene el potencial de ser escalable y aplicable a otros sectores de la industria de alimentos y bebidas, lo que amplía aún más su relevancia e impacto en el ámbito de la ingeniería de procesos.

2. Marco conceptual o de referencia

2.1. Estado de la cuestión

El proceso de elaboración de cerveza es complejo, involucrando múltiples pasos y variables que deben de ser cuidadosamente controlados para producir consistentemente un producto de calidad. Se puede dividir en dos etapas: la maceración o cocimiento del mosto y la fermentación alcohólica. La primera etapa, la maceración, la cebada previamente malteada, es decir el grano que ha sido germinado y tostado para la activación de enzimas, es después molido a un tamaño en específico, para exponer almidones, y mezclado con agua caliente para efectuar la conversión enzimática de carbohidratos a azúcares fermentables a proporciones de malta, pasos definidos de tiempo y temperatura (*mashing*) ajustados para cada receta (Buonocore, Ciavolino, Di Caro, & Liguori, 2021).

El producto del cocimiento, llamado mosto, es entonces hervido por un periodo al tiempo que puede ser añadido el lúpulo para causar que sus ácidos alfa sean isomerizados para producir amargor y aroma. El mosto clarificado es preferiblemente enfriado a través de un intercambiador de calor, por su rapidez, y transferido a un tanque. Ahí, es inoculado con levadura, ya sea *Saccharomyces cerevisiae* para cervezas tipo *Ale* o *Saccharomyces pastorianus* para tipo *Lager*, comenzando así el proceso de fermentación cervecera.

La fermentación de la cerveza es un proceso exotérmico influenciado por factores clave como la tasa de inoculación, el oxígeno disuelto y la temperatura, al ser estos factores claves para la calidad y perfil del producto final (Bhonsale, Mores, & Van Impe, 2021). En esta etapa que puede durar días o semanas en tanque cónico, la densidad del mosto se ve disminuida a su vez que reduce su pH, consecuencia de que los azúcares sean convertidos en etanol, dióxido de carbono y otros subproductos por el metabolismo de la levadura (Buonocore, Ciavolino, Di Caro, & Liguori, 2021). El rango de temperatura óptimo depende del tipo de levadura que se utilice; una baja puede estancar una fermentación mientras que una alta, aunque acelere el proceso, provoca pérdida de compuestos volátiles y la producción de subproductos indeseables.

En un estudio de De Andrés-Toro *et al.* (1997) se presenta un modelo cinético para la producción de cerveza a partir de datos experimentales, utilizando regresión no lineal, que tiene en cuenta cinco variables de respuesta: biomasa, azúcares, etanol, diacetil y acetato de etilo, en condiciones isotérmicas

como no isotérmicas. Los cerveceros tradicionalmente confían en su experiencia para definir la temperatura de fermentación. El estudio de Buonocore *et al.* (2021) divide el proceso en tres fases:

El periodo de latencia (*lag*) corresponde al inicio de la fermentación, que se da tras la inoculación de levadura a un *Pitch Rate* (tasa de inoculación) típico entre 1.0 y 1.5 millones de células por mL por °P de densidad inicial, esta última está relacionada con la densidad específica inicial o gravedad original (*OG*, *Original Gravity*). Durante esta fase, la gravedad relativa del mosto (conocida como *SG*, *Specific Gravity*) se mantiene estable durante las primeras 12-24 horas, esto es sin conversión etílica. A la par, se observa un leve incremento en la biomasa (conteo) y un consumo del oxígeno disuelto (O_2) en el mosto, estableciendo las condiciones para la subsecuente fermentación anaerobia.

Durante la etapa de fermentación activa, la levadura metaboliza los azúcares fermentables del mosto en etanol y dióxido de carbono (CO_2). La densidad específica final, o gravedad relativa final (*FG, Final Gravity*), se sitúa entre 1.010 y 1.020 hacia el final de esta etapa. Esta fase puede extenderse en duración por varios días dependiendo de factores como la cepa de levadura y las condiciones de fermentación. Las recetas de cerveza se diseñan comúnmente para alcanzar valores finales de 4-6% alcohol por volumen *(ABV%*).

La fase de finalización de la fermentación señala la reducción significativa de la actividad metabólica de la levadura. Durante esta etapa, la muerte celular comienza a ser notable, aunque la biomasa global puede mantenerse constante debido a la acumulación de levadura tanto viva como muerta. A la par, los subproductos intermedios, como ésteres y otros compuestos como el diacetilo, son procesados por la levadura, otorgando a la cerveza sus características organolépticas. Es en este punto cuando se alcanzan los valores máximos de biomasa y concentración de etanol.

Para garantizar una calidad y repetibilidad óptimas, es crucial determinar con precisión el final de la fermentación. Una práctica común es dejar reposar a temperatura de fermentación a la cerveza en el fermentador durante un tiempo determinado para asegurarse de que la fermentación haya concluido, pero este método, aunque válido, no es el más eficiente para las cervecerías comerciales que buscan maximizar la utilización de sus fermentadores y mantener una calidad constante.

La fermentación se monitorea tradicionalmente mediante lecturas de temperatura en tiempo real (*in-line*) y mediciones periódicas de densidad específica del mosto (*SG, off-line*) usando hidrómetros o refractómetros; pero esto requiere de operación manual, tiempo, e involucra una disminución de rendimiento (*yield*) al disponer de muestra. Entre los parámetros iniciales relevantes para medir antes del comienzo de la fermentación, por su relevancia, se encuentran: densidad (*OG*, *original gravity*), pH, grado de aeración (oxígeno disuelto), y tasa de inoculación, vitalidad y viabilidad de levadura.

En las cervecerías industriales, los fermentadores suelen contar con sensores *in-line* que permiten la monitorización continua de parámetros del proceso. Sin embargo, estas soluciones pueden resultar costosas para cervecerías independientes, que requieren alternativas de bajo costo, como en el estudio de Buonocore *et al.* (2021), de Spring Off s.r.l. y Universidad de Salemo, donde se permitió la adquisición de datos con tecnología de Internet de las Cosas (*IoT*) al instalarse sensores en tanques de fermentación que se comunican a través del protocolo *MQTT* usando una red *Wi-Fi* para el monitoreo de parámetros.

El desarrollo de un sensor inteligente para la adquisición de datos en tiempo real de parámetros del mosto, y predicción en tiempo real con inteligencia artificial del % de alcohol por volumen (*ABV%)*, permitiría la identificación de desviaciones en lotes y la determinación precisa de finalización de fermentaciones, la programación de equipo de producción y utilización de insumos, mejorando la consistencia de la cerveza e indicadores de sustentabilidad como resaltan los artículos de Bowler *et al.* (2021) y Watson *et al.* (2021).

Los algoritmos de Aprendizaje Automático o *Machine Learning*, por su término en inglés, son métodos estadísticos capaces de modelar relaciones no lineales, complejas, entre múltiples parámetros o características, aunque requieren grandes cantidades de datos canónicos para el entrenamiento, que en el caso de fermentaciones al ser procesos biológicos se traduce en semanas de operación (Bowler, Pound, & Watson, Domain Adaptation and Federated Learning for Ultrasonic Monitoring of Beer Fermentation, 2021). Estos modelos se consideran como "cajas negras" al no requerir información del mecanismo de reacción biológica subyacente para poder identificar patrones “ocultos” y construir un modelo predictivo basado en los datos utilizados para el entrenamiento (Itto-Nakama, Watanabe, Kondo, & Ohnuki, 2021).

En dos estudios de Bowler *et al.* (2021), de la Universidad de Nottingham, se han combinado registros *off-line* e *in-line* junto a modelos de Aprendizaje Automático (*Machine Learning*), de clasificación y regresión, para corregir o extraer información de las mediciones como la clasificación de la etapa de fermentación o la predicción de *ABV%* a través de mediciones de propagación de señal ultrasónica (*US*). Los avances tecnológicos en sensores y en accesibilidad a herramientas de *Machine Learning* comienzan a proporcionarle a las industrias métodos asequibles, económicamente, para adquirir y analizar datos de proceso permitiéndoles una mejor toma de decisiones basada en evidencias, propiciando la "cuarta revolución industrial" o también llamada Industria 4.0 (Watson *et al.*, 2021).

Como ejemplo, Watson *et al.* (2021) desarrollaron un modelo de *Machine Learning* supervisado para predecir *ABV%* a partir de mediciones *US* (propagación de señal) y de temperatura *in-line*, con una arquitectura de red neuronal *NARX* (*Nonlinear Autoregressive with eXogenous input*) con una única capa oculta y otro de regresión lineal, con ambos métodos obteniendo una predicción dentro del 0.2% de variación con respecto a *ABV%* real. A falta de un modelo cinético para describir la fermentación de vinos tintos, la relación dinámica entre metabolitos, Fernández *et al.* (2020) propusieron estimadores Bayesianos basados en procesos Gaussianos (regresión) para la estimación de concentraciones de alcohol y biomasa a partir de mediciones de sustratos (Fernández *et al.,* 2020) de 18 fermentaciones.

Otro ejemplo donde se proyectó el rendimiento en etanol de una fermentación fue en el estudio realizado por Itto-Nakama *et al.* (2021), Universidad de Tokio, donde se desarrollaron modelos de *Machine Learning* a partir de datos morfológicos celulares de cultivos de levadura *Saccharomyces cerevisiae.* Su modelo de red neuronal obtuvo un valor de R^2 de 0.90 y 0.88, para rendimientos de etanol a 30 y 60 minutos, respectivamente, después de adquisición de datos. Sin embargo, encontraron una limitación en las redes neuronales al no ser capaces de proyectar información que no fuera similar a los datos utilizados en el entrenamiento (Itto-Nakama *et al.*, 2021).

En el mismo artículo de Itto-Nakama *et al.* (2021), se mencionan métodos como regresión lineal, regresión LASSO, *k-Nearest Neighbors* (*kNN*), *Support Vector Machines* (*SVM*), árboles de decisión, *random forest*, procesos Gaussianos, lógica difusa, redes neuronales artificiales (*ANNs, Artificial Neural*

Networks) y análisis de componente principal (*PCA, principal component analysis*) como los algoritmos más comunes para estudios de fermentación.

Siendo consideradas una de las herramientas de entrada a la inteligencia artificial, las redes neuronales artificiales, pudieran explicarse con una analogía entre la estructura neurofisiológica del cerebro humano y un modelo matemático. Al igual que las neuronas biológicas, que transmiten señales a través de dendritas y axones, las unidades en una red artificial reciben y procesan entradas transformadas mediante una combinación lineal ponderada, seguida de una función de activación generalmente no lineal (Figura 2). Los pesos de esta ponderación son ajustables y se optimizan dependiendo de cómo se active la conexión durante el entrenamiento, similar a la plasticidad del conjunto de neuronas en el córtex humano (Sipos, Florea, & Arsin, Using Neural Networks to Obtain Indirect Information about the State Variables in an Alcoholic Fermentation Process, 2020).

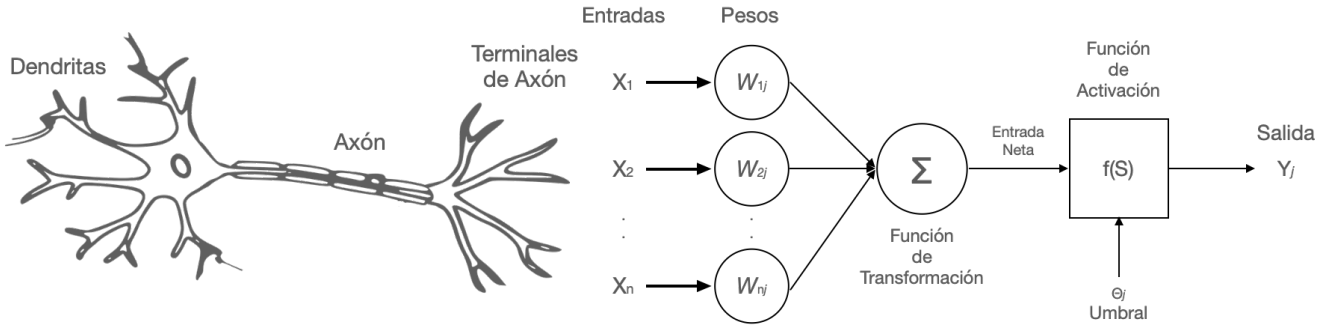


Figura 2. Neurona biológica y neurona artificial: un contraste.

Con sus capacidades extensas de aprendizaje, las *ANNs* son capaces de aproximar funciones continuas y pueden ser utilizadas para modelar procesos no lineales, siendo bien entrenadas y validadas pudieran predecir de manera precisa estados estables y procesos dinámicos; con utilidad en diversas aplicaciones de ingeniería química como en el análisis de datos de sensores, detección de fallas, control y optimización de procesos, por mencionar algunos (Sipos, A Knowledge-Based System as a Sustainable Software Application for the Supervision and Intelligent Control of an Alcoholic Fermentation Process, 2020).

En el estudio Kimutai *et al.* (2021), por la Universidad de Ruanda y su Centro Africano de Excelencia en Internet de las Cosas, se exploró el uso de redes neuronales convolucionales profundas (*DCNNs, Deep Convolutional Neural Networks*) y procesamiento de imágenes para optimizar la fermentación del té negro en un entorno en la nube; utilizando un prototipo integrado una cámara a un microprocesador *Raspberry Pi 3* para el procesamiento de imágenes del progreso de fermentación en tiempo real.

La principal distinción entre redes neuronales *NARX* y *ANNs* de propagación hacia adelante (*feedforward*) o de retropropagación (*backpropagation*) radica en la capacidad de la primera para modelar datos de series temporales con bucles de retroalimentación, su arquitectura permite tomar en cuenta su propia salida previa como entrada para la siguiente predicción, conveniente para el modelado de sistemas dinámicos que presentan comportamientos no lineales a lo largo del tiempo. Por otro lado, las redes *feedforward* con optimización de *backpropagation* constan de una o más capas de neuronas con flujo unidireccional de información, es comúnmente utilizada en tareas de aprendizaje supervisado, como clasificación y regresión.

El equipo de Sipos *et al.* (2021) presentó un primer estudio relacionado a redes neuronales y variables de estado determinó que utilizar pH y CO_2 como entradas adicionales reducen los errores en predicciones de las *ANNs*, por lo tanto, mejorando el desempeño del control de procesos. En un segundo estudio únicamente por Sipos (2021) fue desarrollado un sistema basado en conocimiento, como aplicación de *Software* para el control inteligente de la fermentación alcohólica de vino blanco, para la determinación inicio y final las fases principales: latente, crecimiento exponencial y finalización; considerando variables adicionales como densidad óptica, y las mencionadas en el estudio anterior como pH y concentración de CO_2.

2.2. Conceptos y enfoques teóricos relacionados

Herramientas de análisis de datos y Aprendizaje Automático (*Machine Learning*), se han empleado diversas herramientas y algoritmos, tanto supervisados como no supervisados, para abordar distintos aspectos de los procesos de fermentación en la industria cervecera y de bebidas. Estos

enfoques han sido aplicados en combinación con diversos dispositivos de *Hardware* y arquitecturas de modelos computacionales (*Software*), con el objetivo de identificar etapas o desviaciones en la fermentación (clasificación), estimar la densidad del mosto en tiempo real (regresión) y proyectar la finalización del proceso (*forecasting*). La Tabla 1 resume los algoritmos mencionados en la Sección 2.1 (Estado de la Cuestión).

Tabla 1. Técnicas de Aprendizaje Automático relevantes en el Estado de la Cuestión.

Clasificar etapa de fermentación	Estimación de densidad del mosto en tiempo real	Proyección de curva de fermentación
Clasificación/*Clustering*	**Regresión**	***Forecasting***
*ANNs r*etroalimentadas	Regresión Lineal	Regresión Lineal / Lasso
Árboles de Decisión y Lógica Difusa	*kNN*	*kNN*
kNN	***NARX ANNs***	*SVRs*
k-mean	*ANNs* y *LSTM (Long Short-Term Memory)*	*ANNs* recurrentes
SVCs	Estimador Bayes (Proceso Gaussiano)	***NARX ANNs***

Entre los algoritmos empleados en la literatura se encuentran las redes neuronales artificiales (*ANNs*) retroalimentadas y recurrentes, las redes neuronales *NARX*, entre otros. Estos enfoques han demostrado su capacidad para abordar problemáticas específicas en el contexto de la fermentación de bebidas y otros alimentos, ofreciendo soluciones que permiten mejorar la calidad y eficiencia en la producción.

2.3. Herramientas tecnológicas o de innovación consideradas en el trabajo

La implementación de un sistema de adquisición de datos de bajo costo se fundamenta en los principios de ingeniería frugal, buscando adaptar soluciones eficientes y asequibles a las necesidades de las cervecerías artesanales. En este contexto, se integrará un dispositivo (*Hardware*) asequible, como *Raspberry Pi Zero W*, para monitorear en tiempo real los parámetros del proceso de fermentación.

Raspberry Pi Zero W es una computadora de placa única de bajo costo con conectividad *Wi-Fi*, lo que la convierte en una solución versátil para la adquisición y transmisión de datos. Este dispositivo se utilizará tanto para medir y registrar la temperatura del proceso de fermentación como para transmitirlo de manera inalámbrica a una base de datos en la nube de *InfluxDB Cloud*, y visualizarlo en tiempo real con gráficas en paneles de *Grafana Cloud.* Además, su capacidad de procesamiento le permite funcionar como una plataforma central para el análisis y procesamiento de datos, lo que lo convierte en una herramienta integral para el monitoreo del proceso de fermentación.

Además, se utilizarán herramientas de inteligencia artificial de código abierto (*Open Source*), específicamente en el ámbito del Aprendizaje Automático (*Machine Learning*), para desarrollar aplicaciones enfocadas en la clasificación de etapas de fermentación, la identificación de desviaciones en el proceso y la predicción del porcentaje de alcohol por volumen (*ABV%*). Estas herramientas permitirán mejorar la consistencia del proceso de producción, lo que se traducirá en una mejora en la calidad y márgenes del producto final. Este enfoque permite optimizar la inversión en tecnología y garantizar la viabilidad económica del proyecto. Para ello, se explorarán diversas bibliotecas y *frameworks* de Aprendizaje Automático, como *TensorFlow*, *Scikit-learn* y *Keras*, para la implementación de modelos de predicción y clasificación.

3. Estrategia metodológica o de intervención

3.1. Justificación de la estrategia metodológica o de intervención

El enfoque metodológico para el desarrollo del proyecto de intervención se basa en las metodologías de ingeniería de sistemas, con un énfasis en la definición de requerimientos y objetivos, de acuerdo con las necesidades identificadas a través de entrevistas con mentores dentro de Cerveza Loba y otros referentes en la industria cervecera artesanal e independiente mexicana, como Cerveza Minerva. Un aspecto clave en el desarrollo del proyecto es la innovación frugal, buscando integrar componentes accesibles tanto por su bajo costo como por el uso de conocimientos libres u *open source* cuando sea posible.

Se ha seleccionado el enfoque de niveles de madurez tecnológica (*TRLs,* por sus siglas en inglés *Technology Readiness Level*) como estrategia metodológica. Este enfoque incluye la observación del estado actual del conocimiento, la descripción del problema, la conceptualización de alternativas, la planificación de pruebas de concepto, la construcción de prototipos, la definición de objetivos para validación y la certificación del sistema.

3.1.1. Consideraciones costo/beneficio de la estrategia

Uno de los objetivos explícitos de la intervención es mantener el costo del *Hardware* integrado por debajo de los 150 *USD*, incluyendo sensores, microcontrolador o microprocesador y otros componentes electrónicos, así como elementos de protección del sistema. La meta es desarrollar un sistema técnicamente factible que pueda ser adquirido e instalado por cualquier cervecería artesanal, incluyendo nano y microcervecerías, sin que el costo represente una barrera para su implementación. Los beneficios derivados del monitoreo de parámetros en tiempo real, la identificación de desviaciones en la fermentación y el conocimiento generado para mejorar sus procesos a partir del registro de sus lotes son mejoras valiosas para los cerveceros artesanales. Estas mejoras van más allá de evitar la pérdida del 2% de producción debido a variaciones corregibles en los procesos no detectadas, y tienen el potencial de generar un impacto significativo en la calidad y la eficiencia de la producción cervecera.

3.2. Herramientas e instrumentos

El sistema propuesto, denominado *BreweryHub*, aprovechará la conectividad del Internet de las Cosas (*IoT*) para procesar datos adquiridos del proceso de fermentación, tanto en línea como *off-line*, combinando el conocimiento práctico en la elaboración de cerveza artesanal con técnicas de análisis de datos, cómputo en la nube, servidores web e inteligencia artificial, como se ilustra en la Figura 3.

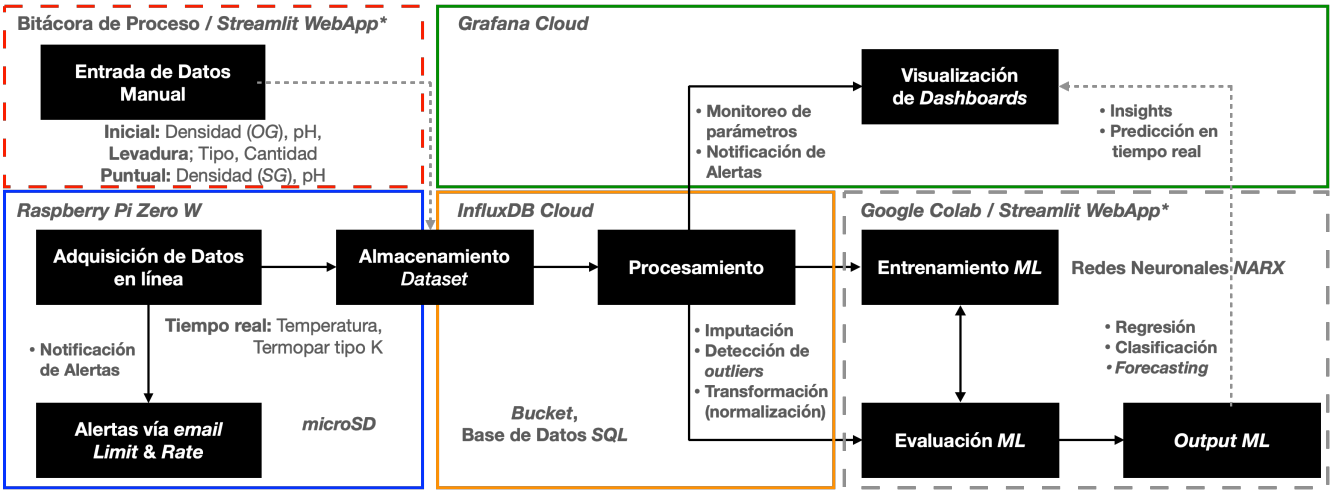


Figura 3. Flujo de información.

3.3. Muestra o sujetos de investigación

El proyecto de intervención se llevará a cabo en Cerveza Loba, ubicada en Guadalajara, Jalisco, bajo la mentoría de Enrique Aceves-Vincent Ramírez, jefe cervecero y encargado de la operación de la cervecería. Se cuenta con el apoyo y acceso para la implementación del sistema *BreweryHub*. Inicialmente, se instalarán sensores de temperatura, cable termopar tipo K, con módulos *MAX6675* en dos tanques de fermentación de 4,500 litros, y se registrarán las señales en un microprocesador *Raspberry Pi Zero W* con almacenamiento de datos en tarjeta *microSD* (con una funcionalidad analógica a un *Data Logger*) y transmisión a base de datos en *InfluxDB Cloud.* Además, se proporcionará acceso a la bitácora de producción, donde se registran manualmente la densidad relativa y otros parámetros importantes del proceso para los lotes (Tabla 2), abarcando un periodo desde septiembre hasta noviembre del 2023.

Tabla 2. Componentes del sistema.

Bloque	Tipo	Uso	Ubicación
Sensor de Temperatura	Cable Termopar tipo K Módulo MAX6675	Transmisión de temperatura de fermentación a microprocesador	*Cellar* de cervecería Tanques de Fermentación F03, F04 y F08 (4,500 L)
Microprocesador	*Raspberry Pi Zero W*	Pre-procesamiento de datos Almacenamiento en tarjeta *microSD* Transmisión a *Cloud* Reportaje y alertas vía *email*	
Base de Datos	*InfluxDB Cloud*	Almacenamiento en la nube	*Cloud*
Panel de Visualización	*Grafana Cloud*	Visualización en tiempo real de *dashboard* de temperaturas	
Análisis de Datos	*GoogleColab*	Tratamiento y análisis de datos Entrenamiento y ejecución de algoritmos de Aprendizaje Automático	Entorno local *Off-line*
Algoritmos Regresión	*NARX ANNs*	Predicción de avance (*ABV%*), clasificación de etapa y pronóstico de finalización de fermentación	

3.4. Etapas del proceso de aplicación/intervención

El proyecto de intervención se divide en cinco etapas principales, las cuales se desarrollarán en paralelo con el avance del programa de cursos de Investigación, Desarrollo e Innovación (IDI). Estas etapas abarcan desde la definición del problema y requerimientos hasta la validación de sistemas y certificación:

1. Definición del problema y requerimientos (IDI I).
2. Búsqueda de solución (IDI II).
3. Desarrollo del primer prototipo (IDI III).
4. Prueba de concepto (IDI IV).
5. Integración de producto y certificación (IDI V).

3.4.1. Cronograma de trabajo

La Tabla 3 muestra el cronograma del proyecto delineado en niveles de madurez tecnológica (*TRLs*, por sus siglas en inglés *Technology Readiness Level*). Este enfoque permite un seguimiento riguroso y medible del avance, alineado con la estrategia de innovación frugal y definición precisa de requerimientos y objetivos. Los *TRLs* ofrecen hitos claros para cada etapa, desde la conceptualización hasta la certificación, asegurando que el proyecto cumpla eficazmente con las necesidades específicas de la industria cervecera artesanal mexicana.

Tabla 3. Ruta del proyecto.

<table>
<tr><td colspan="2">TRL 1</td><td>TRL 2</td><td colspan="3">TRL 3</td><td>TRL 4</td><td colspan="2">TRL 5</td><td>TRL 6</td></tr>
<tr><td colspan="2">Primavera 2022
(IDI I y II)</td><td colspan="2">Otoño 2022
(IDI III)</td><td colspan="2">Primavera 2023
(IDI IV)</td><td colspan="2">Otoño 2023
(IDI V)</td><td>…</td></tr>
<tr><td colspan="2">Principios y Preconceptos</td><td>Concepto y Aplicación</td><td colspan="3">Análisis de Concepto</td><td colspan="3">Validación Sistema y Componentes</td><td>Certificación</td></tr>
<tr><td colspan="4">Desarrollo de Hardware, herramientas de Software y Machine Learning</td><td colspan="3">Entrenamiento y Evaluación Machine Learning con Simulación</td><td colspan="3">Evaluación Machine Learning con Datos Reales</td></tr>
<tr><td colspan="4">Desarrollo Conceptual</td><td colspan="3">Funcionalidad Data Logger (Almacenamiento Físico)</td><td colspan="2">Cloud Database Visualización Remota y Alertas</td><td>WebApps ML & Bitácora</td></tr>
<tr><td colspan="3">Definición del Problema y Requerimientos</td><td>Primer Prototipo</td><td colspan="3">Prueba de Concepto</td><td colspan="2">Integración e Implementación de Dispositivo en Cervecería</td><td>Transferencia de Dispositivo</td></tr>
</table>

3.4.2. Imprevistos

Debe de tomarse en consideración la naturaleza del proceso a medir, la fermentación alcohólica de cerveza artesanal tiene una duración aproximada de 10 días (E. Aceves-Vincent Ramírez, comunicación personal, 28 de septiembre de 2023), cuyos datos adquiridos durante el proceso (temperatura y densidad relativa) serán utilizados una vez finalizada la fermentación para entrenar los algoritmos de inteligencia artificial de regresión propuesto; existe el riesgo falla electrónica del dispositivo que realiza la adquisición de datos (*Raspberry Pi Zero W*), de programación o de comunicación que pueda ocasionar pérdida de registro parcial en uno o varios tanques de fermentación simultáneamente, comprometiendo mediciones de hasta entre cada respaldo (8 horas).

3.5. Metas de información

El propósito de la intervención es generar información y permitir la visualización de parámetros para el monitoreo de procesos, y como producto del entrenamiento de algoritmos de Aprendizaje Automático (*Machine Learning*) se obtendrían herramientas de análisis de datos que directamente podrían identificar la etapa o desviación de una fermentación (clasificación), predecir la densidad en tiempo real (regresión) o proyectar la finalización del proceso de producción alcohólica (*forecasting*). Estas herramientas proporcionarán información valiosa para mejorar la calidad, eficiencia y control de la producción cervecera en Cerveza Loba.

4. Exposición de hallazgos

4.1. Prueba de Concepto, sistematización y aplicación de escalas de medición

Se llevó a cabo la implementación del modelo cinético desarrollado por De Andrés-Toro et al. (1997) en *Matlab* como primer paso de Prueba de Concepto (ver Anexo 1). Este modelo categoriza la biomasa en tres tipos de células: latentes (X_{lag}), activas (X_{act}) y muertas (X_{dea}). Tras la inoculación de la levadura (X_{inc}), el proceso se divide en dos fases (Ecuación 1 y 2): una fase de latencia y una de fermentación, marcadas por el tiempo que finaliza una e inicia la otra denominado t_{lag}. Durante la fase de latencia, no se observa fermentación; sólo las células muertas se depositan con velocidad μ_{SD} (Ecuación 3 y 4), las células en reposo se activan a razón de μ_L (Ecuación 5) y tienen un crecimiento con una tasa específica de crecimiento μ_X (Ecuación 6 y 7).

$$X_{act}(0) + X_{lag}(0) = 0.5 \cdot X_{inc}(0), \quad t < t_{lag} \qquad (1)$$

$$X_{sus}(t) = X_{act}(t) + X_{lag}(t) + X_{dea}(t) \qquad (2)$$

$$\frac{dX_{sus}(t)}{dt} = -\mu_{SD} \cdot X_{DT}(t) = -\mu_{SD}(X_{sus}(t) - 0.5 \cdot X_{inc}), \quad t < t_{lag} \qquad (3)$$

$$\mu_{SD} = \frac{\mu_{SD_0} \cdot 0.5 \cdot C_{s_0}}{0.5 \cdot C_{S_0} + C_e(t)} \qquad (4)$$

$$\frac{dX_{act}(t)}{dt} = \mu_{lag} \cdot X_{lag}(t) = \mu_L(0.5 \cdot X_{inc} - X_{act}(t)), \quad t < t_{lag} \qquad (5)$$

$$\frac{dX_{act}(t)}{dt} = \mu_x \cdot X_{act}(t) - \mu_{DT} \cdot X_{act}(t) + \mu_L \cdot X_{lag}(t), \quad t < t_{lag} \qquad (6)$$

$$\mu_x = \frac{\mu_{x_0} \cdot C_s(t)}{k_x + C_e(t)} \qquad (7)$$

La tasa de consumo de azúcares C_s se describe en Ecuación 8 y 9, donde μ_s es la velocidad de consumo de sustrato específica. La tasa de producción de etanol μ_e es modelada en función de la biomasa activa, incluyendo un factor de inhibición f proporcional a la cantidad máxima posible de etanol como se muestra en las ecuaciones 10 a 13.

$$\frac{dC_s(t)}{dt} = -\mu_s \cdot X_{act}(t) \qquad (8)$$

$$\mu_s = \frac{\mu_{s_0} \cdot C_s(t)}{k_s + C_s(t)} \qquad (9)$$

$$\frac{dC_e(t)}{dt} = \mu_e \cdot X_{act}(t) \qquad (10)$$

$$\frac{dC_e(t)}{dt} = f \cdot \mu_e \cdot X_{act}(t) \qquad (11)$$

$$f = 1 - \frac{C_e(t)}{0.5 \cdot C_{S_0}} \qquad (12)$$

$$\mu_e = \frac{\mu_{e_0} \cdot C_s(t)}{k_e + C_s(t)} \qquad (13)$$

Además, se modelan las tasas de producción de subproductos como diacetilo C_{dy} y acetato de etilo C_{ea}, de este último se considera un coeficiente estequiométrico de 1 entre producto y sustrato (Ecuación 14). La dinámica de diacetilo se complica debido a su conversión a otros productos, que se asume proporcional a la concentración de etanol, tal como se expresa en la Ecuación 15.

$$\frac{dC_{ea}(t)}{dt} = Y_{eas} \cdot \frac{dC_s(t)}{dt} \qquad (14)$$

$$\frac{dC_{dy}(t)}{dt} = \mu_{dy} \cdot C_s(t) \cdot X_{act}(t) - \mu_{ab} \cdot C_{dy}(t) \cdot C_e(t) \qquad (15)$$

El objetivo de implementar este modelo desarrollado por De Andrés-Toro *et al.* (1997) era simular fermentaciones bajo diferentes perfiles de temperatura para generar conjuntos de datos que se analizarán posteriormente y utilizarán en una etapa de pre-entrenamiento en una etapa de Prueba de Concepto.

Para la simulación del sistema de ecuaciones diferenciales ordinarias (*ODE*) se utilizaron los parámetros de Tabla 4, adicionalmente incluyendo fluctuaciones diurnas y nocturnas en la temperatura con una función de ruido senoidal (amplitud 0.2 K, periodo 24 horas), así como un ruido uniforme

aleatorio ($\pm$ 0.2 K), tal como se ilustra en color negro en la Figura 4. Estos perfiles reflejan condiciones comunes para las levaduras *Ale* y *Lager*, y se consideraron parámetros comunes en el ámbito de la cervecería artesanal, como la concentración de mosto (en g/L) y el conteo de levadura (la cantidad de levadura por litro de mosto, 100 g por 100 L), ver Tabla 5 y Figura 4 a 7.

Tabla 4. Valores de parámetros calculados experimentales en función de la temperatura (*T* en K) por De Andrés-Toro et al. (1997).

Parámetro	Valor
μ_{x_0}	$e^{108.31 - \frac{31934.09}{T}}$
Y_{eas}	$e^{89.92 - \frac{26589}{T}}$
μ_{s_0}	$e^{-41.92 - \frac{11654.64}{T}}$
μ_{lag}	$e^{30.72 - \frac{9501.54}{T}}$
μ_{dy}	$-6.1344 \times 10^{-8} \cdot T^2 + 8.4266 \times 10^{-6} \cdot T - 1.7672 \times 10^{-2}$
μ_{ab}	$-9.1384 \times 10^{-7} \cdot T^2 + 6.7071 \times 10^{-5} \cdot T - 0.1251 \times 10^{-3}$
μ_{DT}	$e^{130.16 - \frac{38313}{T}}$
μ_{SD_0}	$e^{33.82 - \frac{10033.28}{T}}$
μ_{e_0}	$e^{3.27 - \frac{1267.24}{T}}$
μ_e	$e^{-119.63 - \frac{34203.95}{T}}$

Tabla 5. Perfiles de fermentación simulados en *Matlab* para Prueba de Concepto, *'dataset_training_PoC'*.

Perfil de Fermentación	Valores de Temperatura (°C)	Tiempo (h)
A	14.5, 14.5, 22.0, 22.0, 0.0	0, 150, 175, 195, 220
B	13.5, 13.5, 22.0, 22.0, 0.0	0, 150, 175, 195, 220
C	15.0, 15.0, 22.0, 22.0, 0.0	0, 150, 175, 195, 220
D	14.0, 14.0, 22.0, 22.0, 0.0	0, 150, 175, 195, 220

Concentración de Sustrato (g/L)	Conteo de Levadura (g/100 L)
88 : 116, pasos cada 2 g/L	100

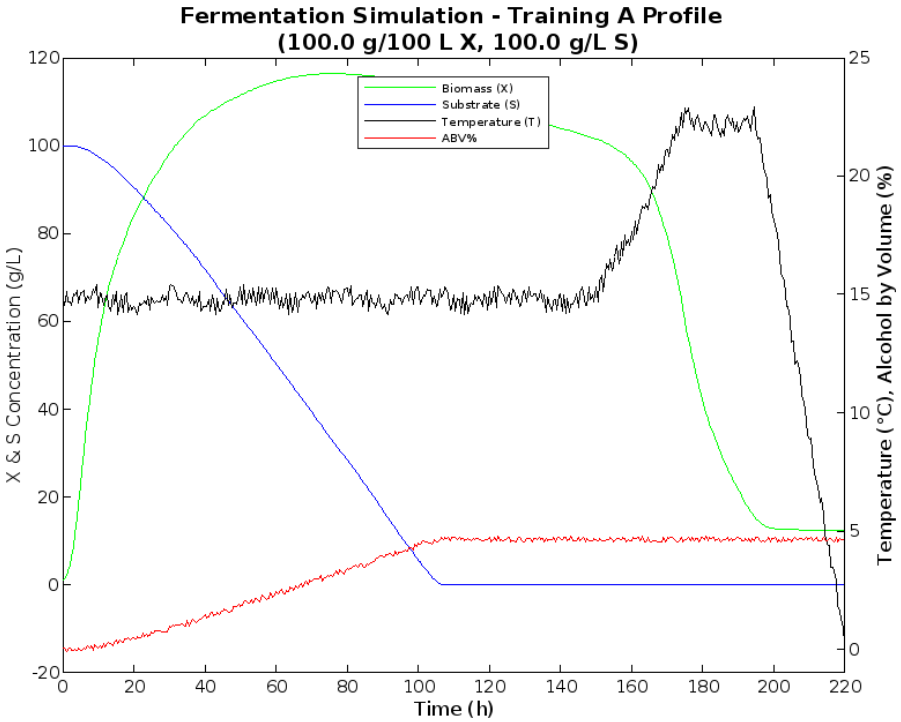


Figura 4. Simulación de fermentación con perfil de temperatura A con modelo modificado de De Andrés-Toro et al. (1997), Prueba de Concepto. Parámetros S: 100 g/L, X: 100 g/100 L.

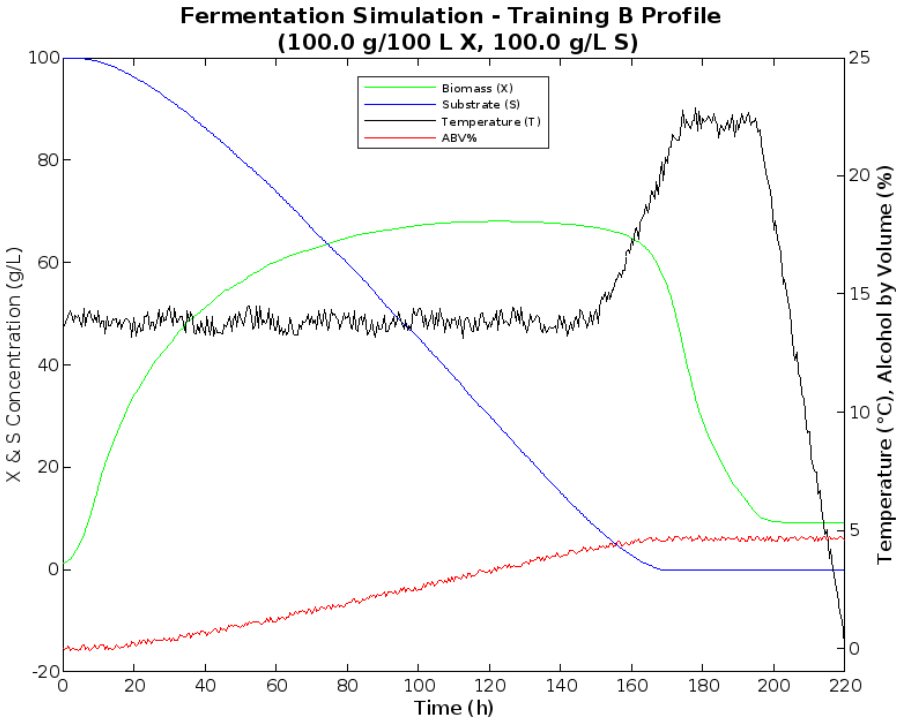


Figura 5. Simulación de fermentación con perfil de temperatura B con modelo modificado de De Andrés-Toro et al. (1997), Prueba de Concepto. Parámetros S: 100 g/L, X: 100 g/100 L.

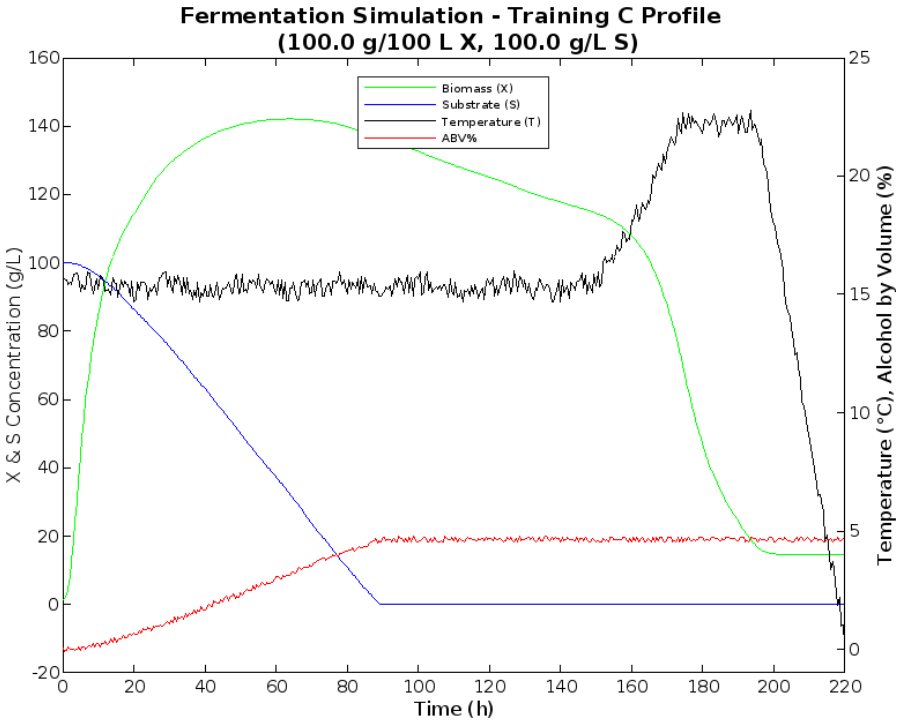


Figura 6. Simulación de fermentación con perfil de temperatura C con modelo modificado de De Andrés-Toro et al. (1997), Prueba de Concepto. Parámetros S: 100 g/L, X: 100 g/100 L.

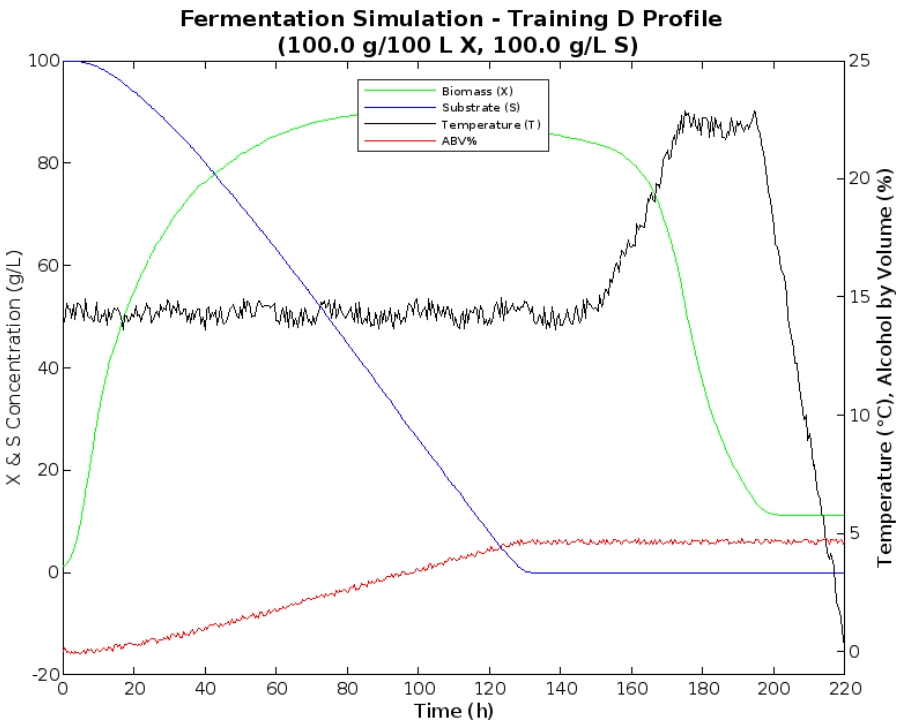


Figura 7. Simulación de fermentación con perfil de temperatura D con modelo modificado de De Andrés-Toro et al. (1997), Prueba de Concepto. Parámetros S: 100 g/L, X: 100 g/100 L.

El conjunto de datos resultante de esta simulación (*dataset*) para Prueba de Concepto (*Proof of Concept*), contiene diversos parámetros clave del proceso de fermentación pero que deben de ser preparados, con funciones de transformación o codificaciones, para poder ser utilizados en el entrenamiento de modelos de Aprendizaje Automático.

El modelo de De Andrés-Toro *et al.* (1997) ha servido como un punto de partida sólido para representar las cinéticas de fermentación. No obstante, su aplicación directa en entornos de cervecerías artesanales ha revelado ciertas limitaciones representando perfiles reales de temperatura para fermentaciones *Lager* y *Ale.* Ya que la fermentación de la cerveza es un proceso biológico complejo sujeto a variaciones debido a factores prácticos como las cepas de levadura utilizadas, las condiciones del entorno y las técnicas específicas de la cervecería. Por lo que después de una exploración exhaustiva de datos, se propone modificar el modelo original ajustando el perfil de temperatura después de finalizada la simulación, corrigiendo las temperaturas en -1.25°C para perfiles de fermentación *Lager* y +1.25°C para *Ales*. Este ajuste le permite al modelo ser más generalizable y aplicable en un contexto de producción de cerveza artesanal, lo cual aumenta su utilidad práctica para los pequeños productores.

En Figura 8 se presentan todos los perfiles de fermentación mencionados en Tabla 5, con su respectiva variable de respuesta de % alcohol por volumen (*ABV%*).

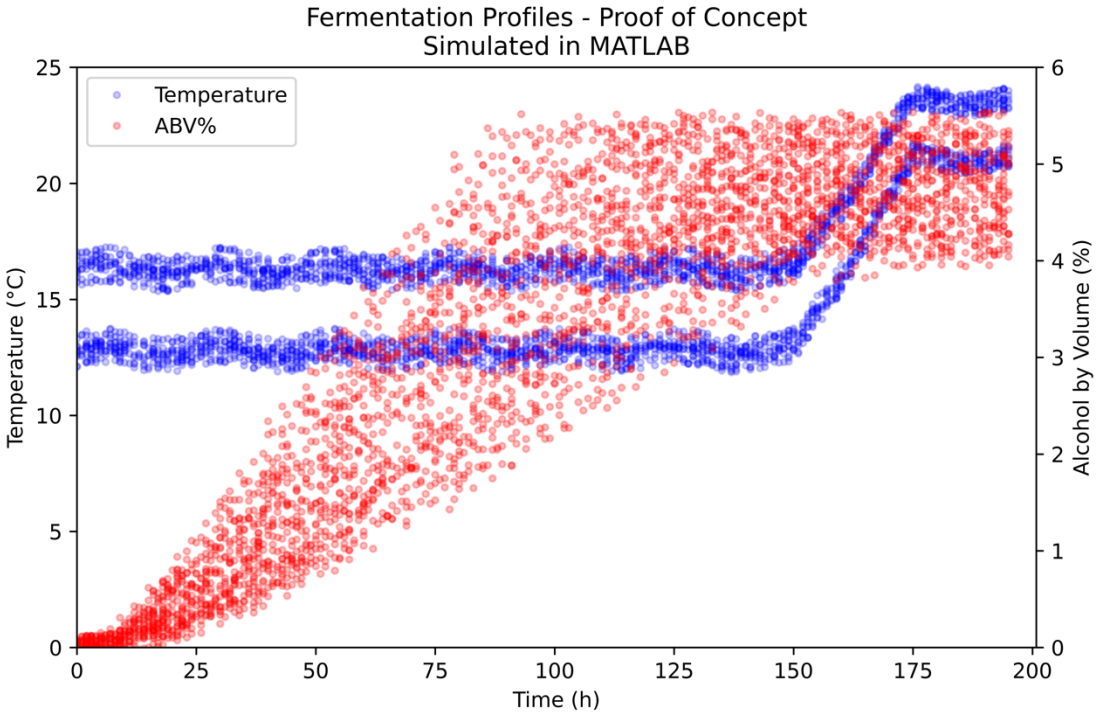


Figura 8. Perfiles de fermentación A - D (Tabla 4) del conjunto de datos (dataset), drop-rate 80%, resultado de simulación en Matlab de modelo de Andrés-Toro et al. (1997).

Entre los atributos a considerar, está el tiempo transcurrido en horas y una normalización de este con Ecuación 16 (t, *Timepoint*), que ajusta el tiempo desde el inicio de la fermentación hasta 195 horas, aproximadamente 32 horas después de alcanzar el pico de alcohol por volumen (*ABV%* máximo).

$$Timepoint(n) = \frac{t\_horas(n) - t\_horas.min()}{t\_horas.max() - t\_horas.min()} \qquad (16)$$

La función *sigmoid*, formulada en la Ecuación 17, describe una curva en forma de 'S' que muestra un cambio rápido en el centro (t = 0.5) y cambios más lentos a medida que el tiempo normalizado (*t)* cambian del 0 al 1, esta propiedad la hace útil para representar transiciones suaves. Sin embargo, para tener una base de referencia clara y un rango estandarizado, se introduce la función *scaled sigmoid* en la Ecuación 18 que escala la función original de manera que sus valores evolucionan del 0 al 1.

$$sigmoid(t) = \frac{1}{1 + e^{-10(t-0.5)}} \qquad (17)$$

$$scaled\ sigmoid(t) = \frac{sigmoid(t) - sigmoid(0)}{sigmoid(1) - sigmoid(0)} \qquad (18)$$

La función de transformación *Gain*, formulada en la Ecuación 19, emplea tanto el tiempo normalizado *t* como la temperatura *T* en grados centígrados. En esta función, el componente de temperatura se eleva a la potencia 2.5, lo que sugiere una relación no lineal y posiblemente una sensibilidad creciente a medida que la temperatura del proceso aumenta. La multiplicación con el complemento de la sigmoide escalada (es decir, 1 − *scaled sigmoid(t)*), tiene el efecto de ponderar la contribución de la temperatura en relación al tiempo. A medida que el tiempo normalizado *t* avanza, si la sigmoide escalada produce valores más altos, su complemento disminuirá, y esto podría usarse para modelar fenómenos donde la influencia de la temperatura se reduce con el tiempo o bajo ciertas condiciones temporales.

$$Gain(t) = \frac{T(t)^{2.5} \times (1 - scaled\ sigmoid(t))}{55^{2.5}} \qquad (19)$$

Esta función permite obtener una visión agregada de cómo se desarrolla la dinámica fermentativa en relación con la temperatura. Para facilitar su análisis y manejo, se ha optado por escalar el resultado dividiéndolo por 55 elevado a la potencia 2.5; esta decisión de escala no es arbitraria, sino que surge de un análisis minucioso de los datos. En conjunto, esta formulación permite que la función *Gain* capture dinámicas complejas que involucran tanto la temperatura como el tiempo normalizado, siendo adecuada para representar comportamientos específicos del proceso que se está modelando. La función *Gainsum*, definida en la Ecuación 20, representa una métrica acumulativa que incorpora el valor de *Gain* a lo largo del tiempo.

$$Gainsum(t) = \sum_{0}^{t} Gain(t) \qquad (20)$$

El propósito detrás de la introducción de estas transformaciones es ofrecer una representación que sea adimensional y, al mismo tiempo, magnifique la respuesta fermentativa a la temperatura. Esto resulta particularmente útil para correlacionar y comparar variables que, en su forma original, podrían ser difíciles de interpretar o comparar. En este contexto, se destaca la correlación entre *Gainsum* y el porcentaje de alcohol por volumen (*ABV%*), lo cual se presenta en la Figura 9 para todos los perfiles de fermentación del A al D. Esta gráfica muestra la acumulación de efectos, representada por *Gainsum*, está relacionada con el contenido alcohólico del producto resultante. Esta transformación *Gainsum* será utilizada como atributo en el conjunto de datos *'dataset_training_PoC'*.

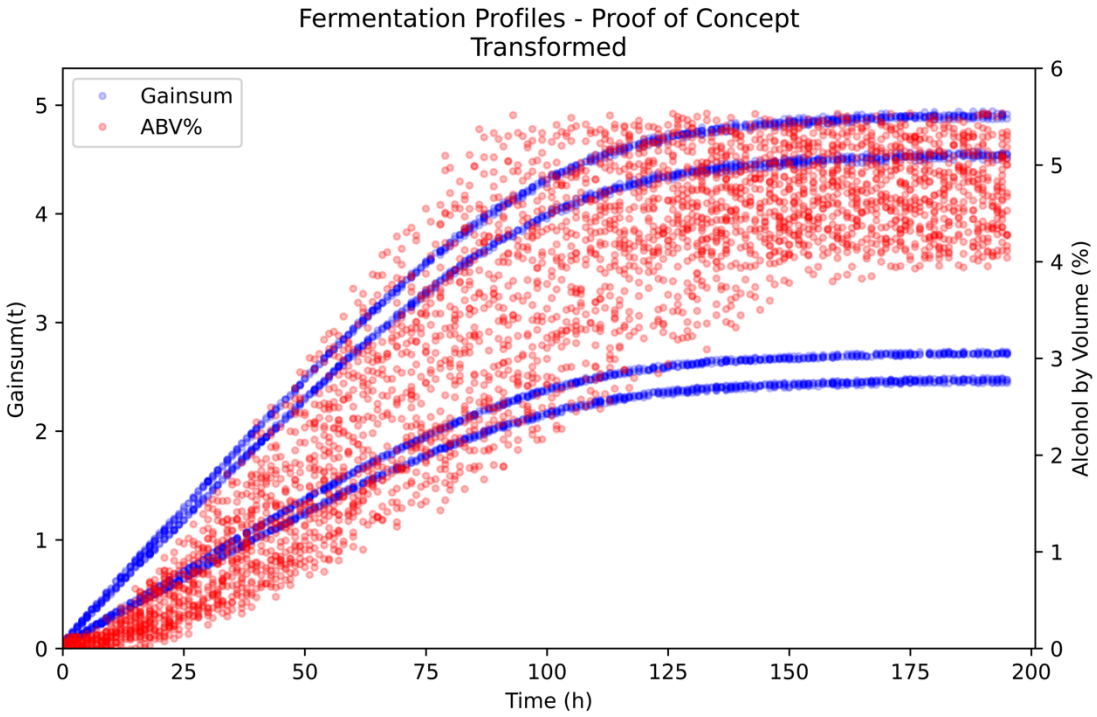


Figura 9. Exploración de relación entre función de transformación acumulada (nombrada Gainsum) y alcohol por volumen (ABV%), drop-rate 80%, para diferentes perfiles de fermentación A - D, 'dataset_training_PoC'.

Este tipo de representaciones gráficas suman al entendimiento de la dinámica de la fermentación y cómo factores como la temperatura influyen en la velocidad de conversión, a esta tasa de cambio de azúcares en alcohol y CO_2 se le conoce como actividad de fermentación, entendida como la tasa de disminución en grados Plato (°P) por hora. Los grados Plato representan la concentración de extracto soluble en el mosto, comúnmente utilizado en la industria cervecera para medir la densidad del líquido en función de los azúcares presentes.

Se evaluó la transformación de los atributos mediante matrices de correlación, y se destaca un coeficiente cercano a 0.854 entre *ABV%* y *Gainsum*. Esta correlación positiva evidente, ilustrada en la Figura 10, se puede vincular a factores físicos y biológicos en el proceso de fermentación, como el impacto de la temperatura sobre la energía cinética de las moléculas, la energía de activación y la actividad enzimática de las levaduras.

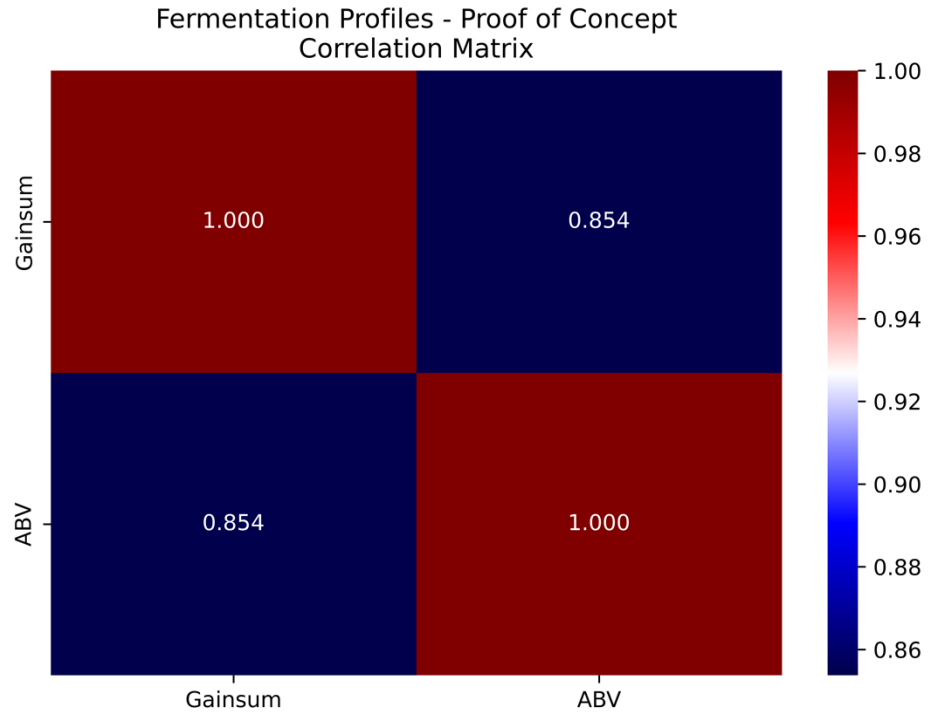


Figura 10. Matriz de correlación entre atributos ABV% y Gainsum de diferentes perfiles de fermentación, 'dataset_training_PoC'.

Se utilizó un gráfico de pares, *pairplot*, que permite visualizar tanto la distribución de variables individuales como las relaciones entre dos variables, ayudando a identificar tendencias y correlaciones en un conjunto de datos. En Figura 11, se presenta una tendencia lineal positiva entre *ABV%* y *Gainsum* es claramente visible en la dispersión diagonal, reforzando aún más la correlación observada en la Figura 10.

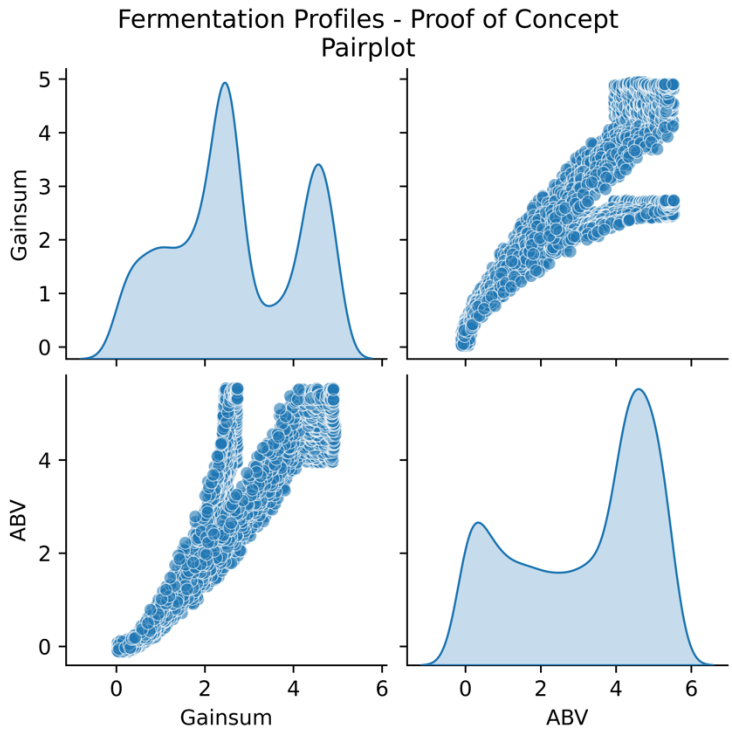


Figura 11. Gráfico por pares, pairplot, entre atributos ABV% y Gainsum en diferentes perfiles de fermentación, 'dataset_training_PoC'.

Adicionalmente, se ha empleado la técnica no paramétrica de densidad de Kernel (*KDE*, por las siglas en inglés de *Kernel Density Estimation*) para estimar la función de densidad de probabilidad (*PDF*, por sus siglas en inglés) y graficar la distribución de cada atributo, visualizándolo como un histograma suavizado. En este análisis, se presenta el comportamiento de la moda asentándose en el punto final de alcohol por volumen, a partir de los datos generados en simulaciones computacionales, como se muestra en Figura 11.

El conjunto de datos simulados en *Matlab*, *dataset*, contiene la concentración del sustrato en función del tiempo (azúcares fermentables) en g/L, pero en el contexto de cervecerías artesanales esta dimensión tiene poco uso, por lo que se transforma a una unidad más conocida antes mencionada, grados Plato (°P). A su vez, este parámetro de concentración es transformado en gravedad relativa (*SG*) por la Ecuación 21. Como medida del potencial de conversión de azúcares fermentables a etanol, se determina con Ecuación 22 una relación entre la gravedad relativa final (*Final Gravity, FG*), obtenida como $SG(t_{final})$, y la gravedad relativa inicial (*Original Gravity*, *OG*), calculadas como $SG(0)$. Al ser un parámetro de gravedad relativa, el valor de estas variables es adimensional.

$$SG(t) = 1.0 + \frac{^{\circ}P(t)}{258.6 - \frac{^{\circ}P(t)}{258.2 \times 227.1}} \qquad (21)$$

$$G(t) = \frac{\big(SG(1) - SG(0)\big) \times 1000}{1.015} \qquad (22)$$

Otro atributo por considerar es el tipo de levadura, en donde se asume tipo *Ale* para perfiles de fermentación A y C, y tipo *Lager* para perfiles B y D. Esta distinción se realizó eligiendo a los perfiles nones como tipo *Ale* por tener una magnitud mayor en su evaluación de *Gainsum* y a los perfiles pares arbitrariamente como tipo *Lager* por a su vez tener una menor magnitud menor. Esto se refleja en las Figuras 6 y 7, donde se muestra un extracto del conjunto de datos de entrenamiento (*'dataset_training_PoC'*) para cada perfil de fermentación, donde la etiquetada como cerveza tipo *Ale* por tener una temperatura promedio más alta muestra un aumento más acelerado del *ABV%* en comparación con la fermentación etiquetada como tipo *Lager*. Dada la relación entre la temperatura y la velocidad de reacción, es natural esperar una conversión más rápida de los azúcares a alcohol a temperaturas más elevadas.

El pH inicial (pH0) es otro parámetro a considerar como atributo para el conjunto de datos, ya que es crucial en la fermentación; un rango óptimo es esencial para la actividad y el crecimiento de la levadura, afectando directamente su metabolismo, lo que incide en el rendimiento y en los perfiles de sabor y aroma de la cerveza. Además, el pH influye en la estabilidad microbiológica, donde un valor inapropiado puede aumentar el riesgo de contaminación, comprometiendo la calidad y la seguridad del producto final.

En la práctica cervecera, a partir de la gravedad relativa inicial (*OG*) y valor final (*FG*), se calcula el *%* alcohol por volumen (*ABV%*) utilizando la Ecuación 23. El coeficiente 131.25, empleado en esta fórmula, se basa en aproximaciones empíricas de la industria, si bien es una estimación ampliamente aceptada, su precisión puede variar según las condiciones de fermentación y el tipo específico de cerveza. Este atributo 'ABV' es la variable objetivo del conjunto de datos, *'dataset_training_PoC'*.

$$\%ABV = (OG - FG) \times 131.25 \qquad (23)$$

Hasta el momento se han mencionado los atributos '*t_horas*', '*Timepoint*', '*Yeast_type*', '*G*', '*pH0*, '*Gainsum*' y la variable objetivo '*ABV*'. Pero se identifica que por el tipo de variable que contiene a estos parámetros, algunos valores numéricos podrían implicar una relación ordinal que no existe, es decir, la herramienta Aprendizaje Automático (*Machine Learning*) podría asumir incorrectamente que *Lager* > *Ale* porque el tipo de levadura '*Yeast_type'* 1 > 0, de la misma forma con atributos '*G*' y '*pH0*'. Por esta razón, a estos tres atributos se les aplica una codificación *One-Hot Encoder*, que es un método de procesamiento de datos que es especialmente útil en algoritmos que no soportan variables categóricas, sin orden inherente, y esperan entrada numérica, como la mayoría de los modelos de regresión de *Machine Learning*.

El *One-Hot Encoder* se configuró para tomar una categoría, definiendo sus compartimientos (*bins*) y etiquetas (*labels*), ver Figura 12, para convertir el atributo a columnas binarias (con valores 0 ó 1) correspondiente a la cantidad de etiquetas, en el conjunto de datos. Por ejemplo, para el atributo '*G*' se crearían once nuevas columnas correspondiente a las etiquetas definidas, si valor del parámetro está dentro de los límites de un *bin*, la codificación resultaría en un vector binario para cada observación que sólo tiene un 1 en la posición de la categoría a la que pertenece y 0s en todas las demás posiciones.

```
gravity_bins = [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]
gravity_labels = ['31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41']
pH_bins = [4.0, 5.0, 5.5]
pH_labels = ['5.0', '5.5']
```

Figura 12. Compartimientos (bins) y etiquetas (labels), definidas para aplicar codificación One-Hot Encoder en atributos 'G' y 'pH0' (ver Anexo 4).

Una vez codificados los atributos categóricos '*Yeast_type'*, '*G*' y '*pH0*', y conformado el conjunto de datos *'dataset_training_PoC'* con las nuevas columnas binarias (categorías codificadas) se procede por último a reducir su tamaño en la dimensión de filas, para acelerar el procesamiento durante el entrenamiento de modelos, con una tasa de abandono del 70% (*drop-rate*), para finalizar con un tamaño de 3,528 registros. Ver Tabla 6 y Anexo 4.

Tabla 6. Atributos del conjunto de datos, *'dataset_training_PoC', X: (3528, 18) y Y: (3528, 1), con drop-rate 70%.*

Atributo	Descripción	Tipo de Variable
t_horas	Tiempo transcurrido desde el inicio de la fermentación, medido en horas	Entrada, numérica
Timepoint	Tiempo transcurrido normalizado con Ecuación 16	Entrada, numérica
Gainsum	Suma acumulada de la función de transformación *Gain* (Ecuación 23)	Entrada, numérica
Yeast_type (2 *bins*)	Tipo de levadura utilizada, p. ej. 0 – *Ale*, 1 – *Lager,* a este atributo se le aplica codificación *One-Hot Encoder* para categorizar entre etiquetas '0' y '1'	Entrada, binaria
G (11 *bins*)	Relación entre Gravedad relativa inicial (*OG*) y Gravedad relativa final (*FG*), Ecuación 18. Se le aplica codificación *One-Hot Encoder* para categorizar a etiquetas de '31' a '41'	Entrada, binaria
pH0 (2 *bins*)	*pH* inicial, a este atributo se le aplica codificación *One-Hot Encoder* para categorizar entre etiquetas '5.0' y '5.5'	Entrada, binaria
ABV	Porcentaje de alcohol por volumen (*ABV%),* calculado a partir de Ecuación 19	Variable objetivo, numérica

Tras la etapa de análisis y exploración de datos, se continúa con el desarrollo y optimización de un modelo de red neuronal (*ANNs*) usando el enfoque de *Nonlinear AutoRegressive with eXogenous inputs* (*NARX*) con el fin de predecir una variable objetivo (*ABV%*) basada en valores rezagados y atributos de entrada (ver Tabla 6 y Anexo 5). Para lograr este objetivo, se utilizó la biblioteca *scikit-learn* para dividir los datos en conjuntos de entrenamiento y prueba, y *TensorFlow* con *Keras* para construir y entrenar el modelo definido de red neuronal.

Se importaron las bibliotecas necesarias y se definió el número de valores rezagados (*n_lags*) que se utilizarían como entradas adicionales al modelo. Luego, se creó una versión rezagada de la variable objetivo (*y_lagged)* y se combinó con las variables de entrada originales (*X_narx*). A continuación, se dividió el conjunto de datos en de entrenamiento y de prueba, usando la función *train_test_split*.

Se predefinió una lista de posibles combinaciones de hiperparámetros para optimizar el modelo, los cuales se mencionan a continuación:

1. *firstlayer_act*: Refiere a las funciones de activación utilizadas en la capa de entrada del modelo, estas determinan la salida de un nodo dado un conjunto de entradas. Ejemplos de estas funciones son:
    a. *'relu' (ReLU, Rectified Linear Unit):* Función lineal rectificada que devuelve el valor de entrada si es positivo, y cero en caso contrario.
    b. *'elu' (ELU, Exponential Linear Unit):* Función de activación exponencial lineal que puede producir valores negativos.
    c. *'sigmoid':* Función que devuelve valores entre 0 y 1.
    d. *'tanh':* Función tangente hiperbólica que devuelve valores entre -1 y 1.

2. *hiddenlayer_act*: Refiere a las funciones de activación para las capa oculta intermedia del modelo. Ejemplos:
    a. *'linear':* Una función que no aplica ninguna transformación y devuelve el valor de entrada.
    b. *'hard_sigmoid':* Una aproximación más eficiente computacionalmente de la función sigmoid.
    c. *'tanh':* Como se mencionó anteriormente, es una función tangente hiperbólica.

3. *outputlayer_act:* Funciones de activación específicas para la capa de salida. Para este modelo solo se utiliza *'linear'* que no realiza ninguna transformación.

4. *regularizer*: Son técnicas que penalizan ciertos patrones de parámetros para evitar el sobreajuste en los pesos del modelo. Ejemplos de regularizadores incluyen:
    a. *"None"*: Sin empleo de regularización.
    b. L1: Penaliza los pesos en función del valor absoluto de los mismos.
    c. L2: Penaliza los pesos en función del cuadrado de los mismos.

5. *dropouts*: Es una técnica de regularización donde aleatoriamente se abandonan (*dropout)* o desactivan ciertas neuronas durante el entrenamiento del modelo para evitar sobreajuste (*overfitting)*.

6. *losses*: Son funciones de pérdida que miden qué tan bien el modelo está realizando predicciones. En este caso, *'mse'* (*MSE*, *mean squared error*) o Error Cuadrático Medio mide el promedio de los cuadrados de los errores entre predicciones y valores reales.

7. *optimizers*: Son algoritmos de optimización utilizados para ajustar los parámetros del modelo a fin de minimizar la función de pérdida. Ejemplos de optimizadores son:
    a. *'adam' (Adaptive Moment Estimation):* Optimizador basado en la estimación adaptativa de momentos.
    b. *'RMSprop' (Root Mean Square Propagation):* Divide la tasa de aprendizaje para un peso por un promedio móvil de las magnitudes de los gradientes recientes para ese peso.
    c. *'SGD' (Stochastic Gradient Descent):* Descenso de gradiente estocástico, donde los pesos se ajustan en la dirección opuesta al gradiente, con una tasa fija de aprendizaje.
    d. *'Adagrad' (Adaptive Gradient Algorithm):* ajusta la tasa de aprendizaje según la frecuencia con la que un parámetro recibe actualizaciones, los pesos que se actualizan con menos frecuencia recibirán mayores ajustes.

8. *epochs*: Una época, *epoch*, es una pasada completa a través del conjunto de datos de entrenamiento.

9. *batchs*: El tamaño del lote (*batch*) se refiere al número de ejemplos de entrenamiento utilizados en una iteración para entrenar el modelo.

10. *early_stopping*: Es una técnica de parada temprana del entrenamiento del modelo si se dejan de observar mejoras en el rendimiento del conjunto de validación después de un número especificado de épocas para evitar el sobreajuste del modelo.

11. *n_lags*: Refiere al número de valores previos (rezagados) que se utilizan como entrada adicional al modelo para hacer predicciones en series temporales. Por ejemplo, si *n_lags* es 2, entonces los valores en los dos periodos de tiempo anteriores se usarán.

Previo al entrenamiento de herramienta de Aprendizaje Automático, modelo de red neuronal artificial *NARX,* se llevó a cabo un proceso selectivo durante ensayos de exploración donde se priorizó la carga computacional, donde se descartaron configuraciones de hiperparámetros de la lista predefinida que demostraron un rendimiento subóptimo. Entonces con una lista más reducida, se llevó a cabo una búsqueda exhaustiva (*grid*) de una primera configuración óptima de hiperparámetros (Figura 13).

La iteración sistemática sobre este conjunto reducido permitió entrenar el modelo de red neuronal *NARX* con diversas combinaciones de hiperparámetros, y se evaluó su rendimiento utilizando la métrica de Error Cuadrático Medio (*MSE*, por sus siglas en inglés). Los resultados de cada configuración se almacenaron en una lista (*best_params_1st*), y se ordenaron para identificar la de menor *MSE*. Un resumen del desempeño de todas las combinaciones evaluadas en esta primera optimización se muestra en el *Heatmap* de la Figura 14.

```
firstlayer_act = ['hard_sigmoid','sigmoid', 'tanh', 'relu', 'elu'] # 'sigmoid', 'tanh', 'relu', 'selu', 'elu', 'gelu', 'linear', 'hard_sigmoid'
hiddenlayer_act = ['hard_sigmoid', 'sigmoid', 'tanh'] # 'hard_sigmoid', 'sigmoid', 'linear',
outputlayer_act = ['linear']
regularizer = [None, regularizers.l1(0.0005), regularizers.l2(0.0005)] # none, regularizers.l2(0.0005)
dropouts = [0] # 0.1
losses = ['mse'] # 'mae'
optimizers = ['adam'] # 'RMSprop', 'SGD', 'Adagrad'
epochs = [100, 150]; batchs = [100, 150] # 150, 225, 300
early_stopping = EarlyStopping(monitor = 'mse', patience = 3)
n_lags = [0, 1] # 1
fl_values = [colm_X * 5] # Columns x
hl_values = [round(fl_values[0] / 2)] # fl/2
headers = ['n','Lagged','Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', '1st Layer Activation', 'Hidden Layer Activation', 'Output
Layer Activation', 'First Layer Neurons', 'Hidden Layer Neurons', 'MSE*MAE', 'MSE']
total_iterations = len(n_lags) * len(batchs) * len(epochs) * len(optimizers) * len(losses) * len(dropouts) * len(regularizer) * len(firstlayer_act) *
len(hiddenlayer_act) * len(outputlayer_act) * len(fl_values) * len(hl_values)
n_1st = 0
best_params_1st = []

for lag in n_lags:
    y_lagged = y.shift(periods=lag, fill_value=0)
    X_narx = pd.concat([X, y_lagged], axis=1).iloc[lag:]
    x_train, x_test, y_train, y_test = train_test_split(X_narx.iloc[:,:-1], X_narx.iloc[:,-1], test_size=0.10, random_state=0)
    for batch in batchs:
        for epoch in epochs:
            for opt in optimizers:
                for l in losses:
                    for dropout in dropouts:
                        for reg in regularizer:
                            for flact in firstlayer_act:
                                for hlact in hiddenlayer_act:
                                    for olact in outputlayer_act:
                                        for fl in fl_values:
                                            for hl in hl_values:
                                                model = Sequential()
                                                model.add(Dense(fl, input_dim=X_narx.shape[1]-1, activation=flact, kernel_regularizer=reg))
                                                model.add(Dropout(dropout))
                                                model.add(Dense(hl, activation=hlact, kernel_regularizer=reg))
                                                model.add(Dense(1, activation=olact))
                                                model.compile(loss=l, optimizer=opt, metrics=['mse', 'mae'])
                                                model.fit(x_train, y_train, epochs=epoch, batch_size=batch, verbose=0, callbacks=[early_stopping])
                                                _, mse, mae = model.evaluate(X, y, verbose=0)
                                                n_1st += 1
                                                progress_percent = (n_1st / total_iterations) * 100
                                                best_params_1st.append((lag, batch, epoch, opt, l, dropout, reg, flact, hlact, olact, fl, hl, mse*mae, mse))
                                                results = [(n_1st, lag, batch, epoch, opt, l, dropout, str(type(reg)).split(".")[-1][:-2], flact, hlact, olact, fl, hl, mse, mae) for n_1st,
(lag, batch, epoch, opt, l, dropout, reg, flact, hlact, olact, fl, hl, mse, mae) in enumerate(best_params_1st, 1)]
                                                print(f'\rProgress: {n_1st}/{total_iterations} ({progress_percent:.2f}%)', end='', flush=True)
```

Figura 13. Sección de código Python de primer etapa de optimización de hiperparámetros, Prueba de Concepto (Anexo 5).

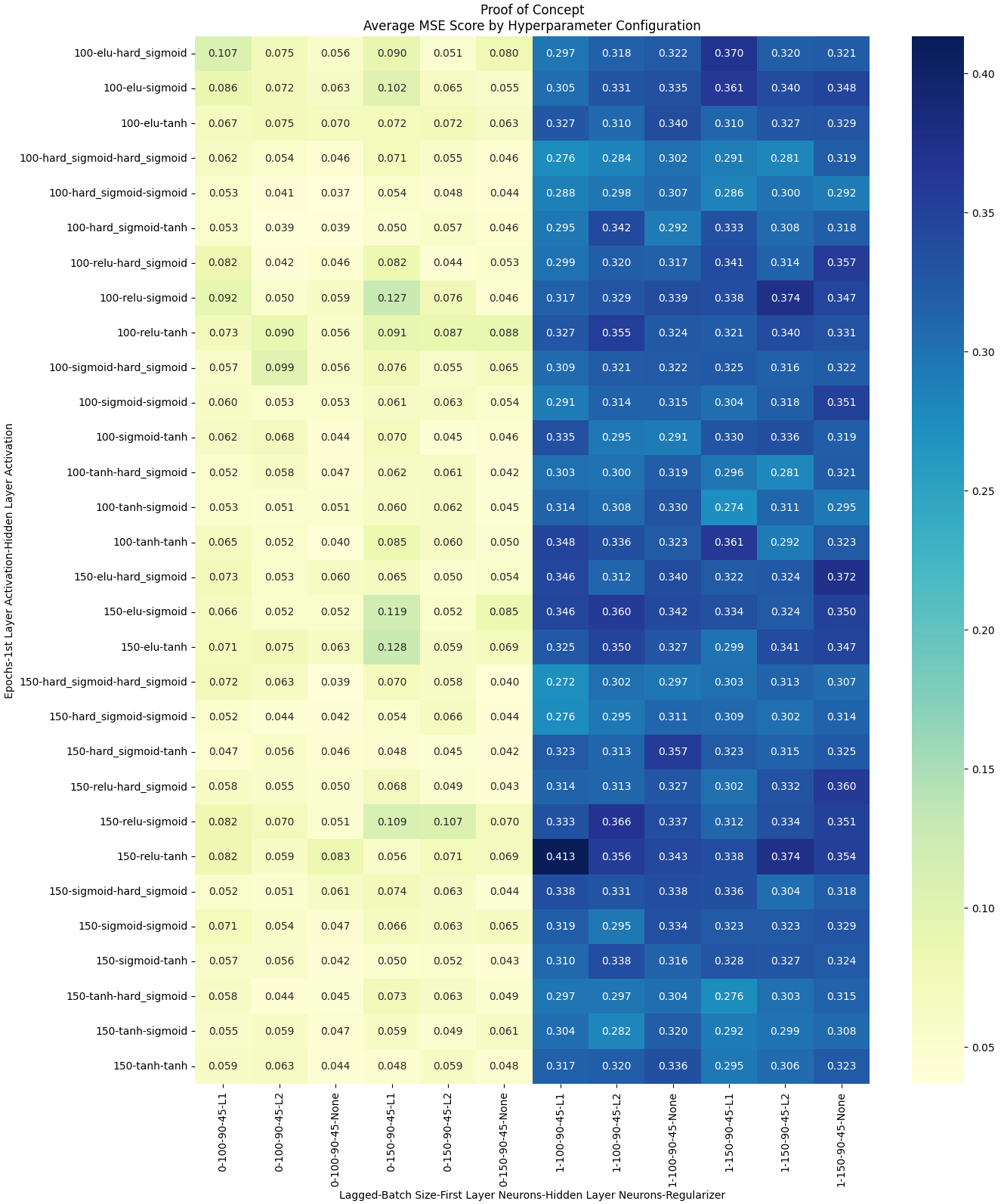


Figura 14. Resumen de MSE obtenida por configuración de hiperparámetros de primer búsqueda grid, Prueba de Concepto.

Una vez obtenida la combinación inicial de hiperparámetros óptimos, se emplean para una subsecuente fase de búsqueda *grid*, donde se varió el número de neuronas en la primera capa y capa oculta (*First Layer Neurons* y *Hidden Layer Neurons,* respectivamente). Este proceso se ilustra en el código de la Figura 15 y en el Anexo 6, y los resultados se detallan en la Tabla 7. Los resultados se almacenaron en una lista (*best_params_2nd*), un resumen del desempeño de esta segunda optimización se muestra en el *Heatmap* de la Figura 16.

```
fl_values = [colm_X * 3, colm_X * 4, colm_X * 5] # Columns x n
hl_values = [(2 * colm_X)-2, round(fl_values[0] / 2), round(fl_values[1] / 2), round(fl_values[2] / 2)] # ROUND(fl/2)

total_iterations = len(fl_values) * len(hl_values)
early_stopping = EarlyStopping(monitor = 'mse', patience = 5)
headers_2nd = ['First Layer Neurons', 'Hidden Layer Neurons', 'Lagged', 'Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', 'First
Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'MSE']

best_mse = float('inf')
best_model = None

n_2nd = 0
best_params_2nd = []

for fl in fl_values:
  for hl in hl_values:
    y_lagged = y.shift(periods=best_param[0], fill_value=0)          # Variable target
    X_narx = pd.concat([X, y_lagged], axis=1).iloc[best_param[0]:] # Combina input original + lagged
    x_train, x_test, y_train, y_test = train_test_split(X_narx.iloc[:,:-1], X_narx.iloc[:,-1], test_size=0.30, random_state=0)
    model = Sequential()
    model.add(Dense(fl, input_dim=X_narx.shape[1]-1, activation=best_param[7], kernel_regularizer=best_param[6]))
    model.add(Dropout(best_param[5]))
    model.add(Dense(hl, activation=best_param[8], kernel_regularizer=best_param[6]))
    model.add(Dense(1, activation=best_param[9]))
    model.compile(loss=best_param[4], optimizer=best_param[3], metrics=['mse'])
# Compilación modelo
    model.fit(X, y, epochs=best_param[2], batch_size=best_param[1], verbose=0, callbacks=[early_stopping])              # Ajuste modelo a dataset
    _, mse = model.evaluate(X, y, verbose=0)
    if mse < best_mse:  # Si el mse actual es mejor que el mejor mse registrado
      best_mse = mse  # Actualiza el mejor mse
      best_model = model                    # Evaluación del modelo
    n_2nd += 1
    progress_percent = (n_2nd / total_iterations) * 100
    best_param_tuple_2nd = (best_param[0], best_param[1], best_param[2], best_param[3], best_param[4], best_param[5], best_param[6],
best_param[7], best_param[8], best_param[9])
    best_params_2nd.append((fl, hl, *best_param_tuple_2nd, mse)) # Almacena parámetros utilizados
    results_2nd = [(n_2nd, fl, hl, *best_param_tuple_2nd,  mse) for n_2nd, (fl, hl, *best_param_tuple_2nd, mse) in enumerate(best_params_2nd, 1)]
    print(f'\rProgress: {n_2nd}/{total_iterations} ({progress_percent:.2f}%)', end='', flush=True)
```

Figura 15. Sección de código Python de segunda etapa de optimización de hiperparámetros, Prueba de Concepto (Anexo 6).

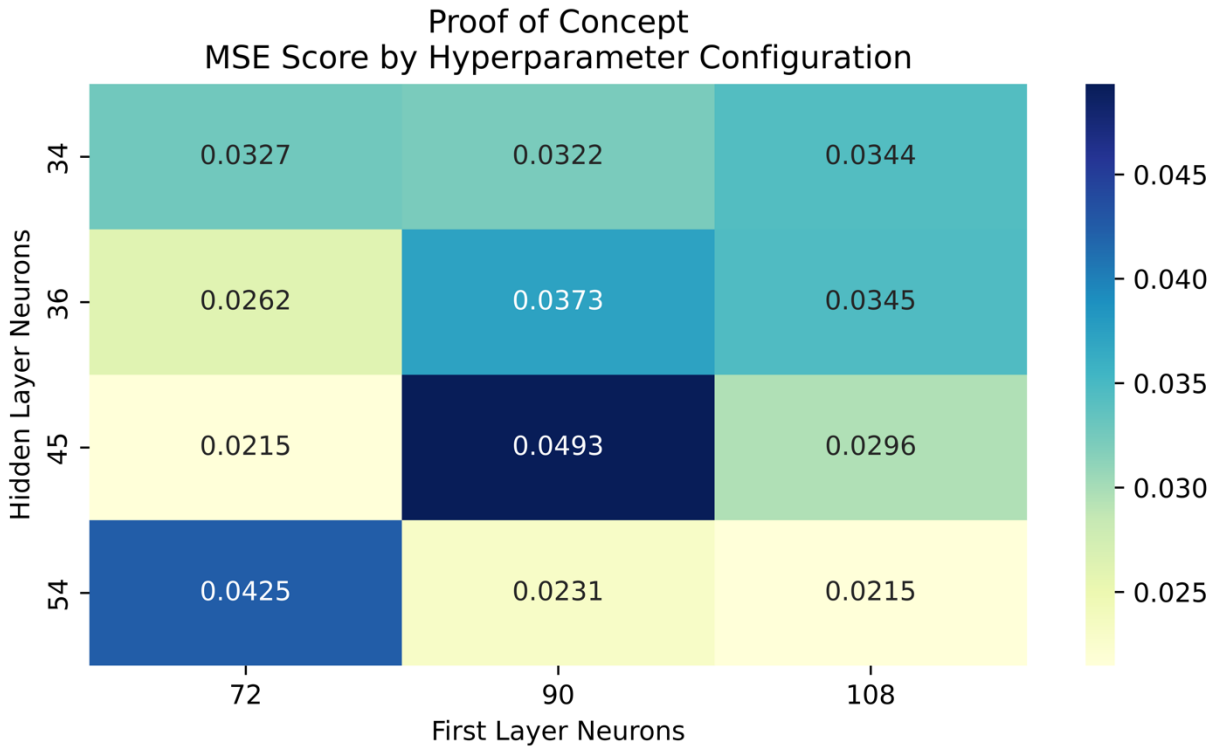


Figura 16. Resumen de MSE obtenida por configuración de hiperparámetros de segunda búsqueda grid, Prueba de Concepto.

De los hiperparámetros óptimos y arquitectura del modelo *NARX* (ver Tabla 7), se pueden inferir varios aspectos clave. Los valores rezagados ajustados a cero sugieren que el modelo pondera variables de entrada sobre valores pasados para predecir siguientes valores, debido posiblemente al *drop-rate* en el *dataset* y al ruido aleatorio del atributo '*ABV%*' pero arbitrariamente generado para la simulación de *Matlab* (como se ve en Figuras 4 a 7), aunque coherente con un proceso dinámico como la fermentación, donde factores concurrentes pueden ser más significativos que las observaciones anteriores solas.

La ausencia de una técnica de abandono sugiere un enfoque centrado en evitar el sobreajuste. El uso de un tamaño de lote de 100 y 100 épocas en el entrenamiento indica un equilibrio cuidadoso entre la velocidad de cómputo y la precisión del modelo, permitiendo que la red aprenda de manera efectiva sin caer en el excesivo ajuste a los datos de entrenamiento. El resultado con regularizador *'None'* y una técnica de abandono de 0 como hiperparámetros, muestran una preferencia por la simplicidad del modelo.

La elección de 72 neuronas en la capa de entrada y 45 en la capa oculta refleja una estructura que puede capturar la complejidad de los datos sin sobreajustarse (alta dimensionalidad), lo que es fundamental en conjuntos de datos de alta dimensión como los utilizados en la fermentación de cerveza aplicando la codificación *One-Hot Encoder*. Las funciones de activación resultantes *'hard_sigmoid'* para la capa de entrada y *'sigmoid'* en la capa oculta sugieren un modelado que puede manejar transiciones no lineales y complejas entre estados, mientras que la función 'linear' se mantuvo arbitrariamente en la capa de salida ya que es típica para problemas de regresión, facilitando la interpretación directa de las predicciones. Finalmente, obteniendo una métrica *MSE* de 0.02146 como una cuantificación objetiva de la capacidad predictiva del modelo, como se muestra en Tabla 7 y en Figura 17 evaluando con datos de entrenamiento.

Tabla 7. Hiperparámetros óptimos para el modelo *NARX* producto de segunda etapa de optimización (Figura 16), *'dataset_training_PoC'.*

Hiperparámetro	Valor	Hiperparámetro	Valor
Valores Rezagados	0	Regularizador	*None*
Tamaño de Lote	100	Neuronas de Capa Entrada	72
Épocas	100	Función de Activación de Capa de Entrada	*hard_sigmoid*
Optimizador	*adam*	Neuronas de Capa Oculta	45
Función de Pérdida	*mse*	Función de Activación de Capa de Oculta	*sigmoid*
Técnica de Abandono	0	Función de Activación de Capa de Salida	*linear*
Métrica (*MSE*)		0.02146	

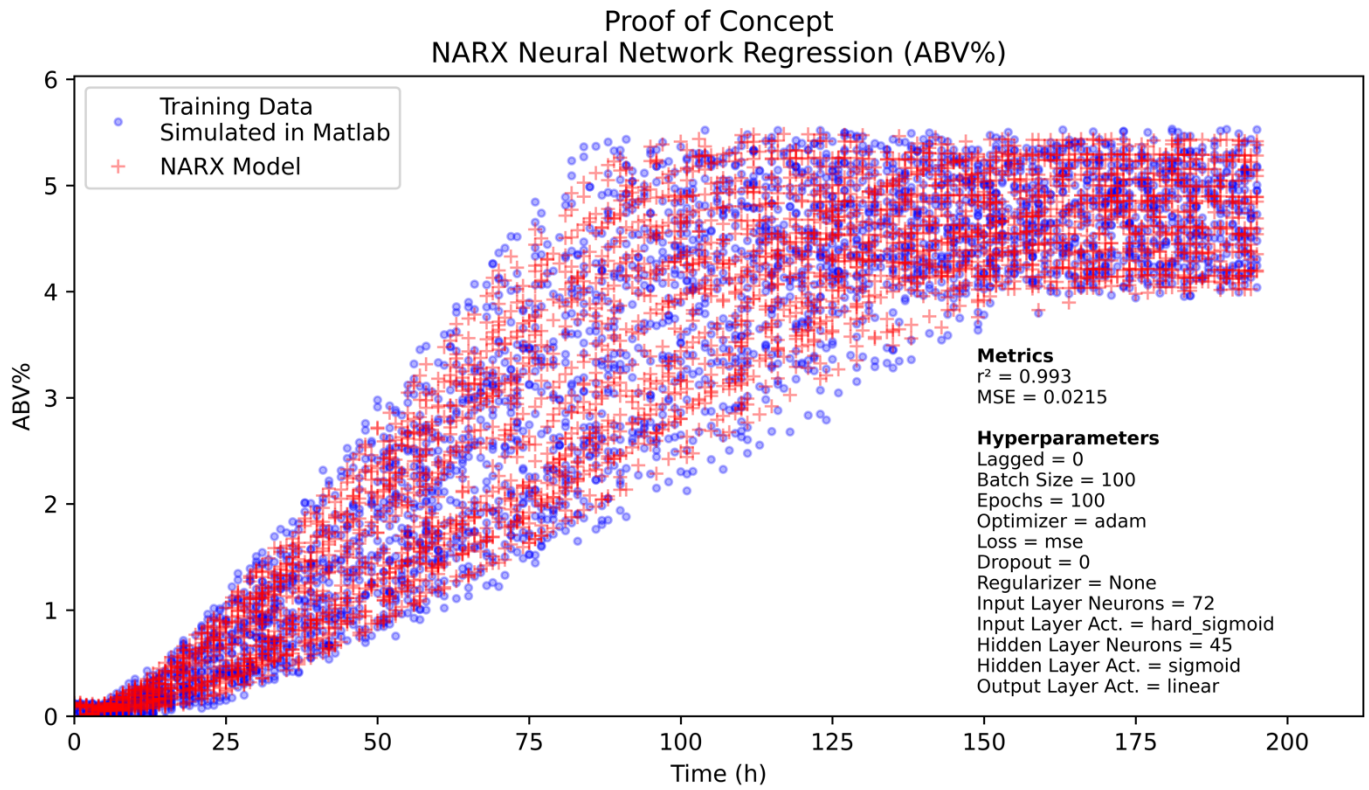


Figura 17. Evaluación de conjunto de datos de entrenamiento, 'dataset_training_PoC', con modelo de red neuronal NARX optimizado, Prueba de Concepto (Anexo 7).

La fermentación cervecera típica, llevada a cabo tanto por *Saccharomyces cerevisiae* para cervezas tipo *Ale* como *Saccharomyces pastorianus* para tipo *Lager*, muestra un crecimiento exponencial tras un período de latencia. Esta progresión es influenciada tanto por la concentración del sustrato disponible como por la temperatura, siendo este último un factor crítico que influye directamente en la energía cinética de las moléculas y por ende en su actividad. La naturaleza exotérmica de la fermentación implica que un incremento en la temperatura puede potenciar la velocidad de reacción, pero superar ciertos límites puede afectar negativamente las características organolépticas dependiendo el tipo de cerveza. Al modelar la fermentación desde un enfoque temporal, es esencial considerar estos aspectos cinéticos y organolépticos. Este modelo *NARX* optimizado, con o sin valor rezagado, encapsula la respuesta fermentativa a cambios de temperatura influyen en la velocidad de reacción puntual. Las funciones de activación, *'hard_sigmoid'* y *'sigmoid'*, seleccionadas a través de la optimización *grid*, buscan representar matemáticamente a estas no linealidades inherentes.

La búsqueda *grid* empleada se destaca por su enfoque sistemático y meticuloso, descrita como una técnica de "fuerza bruta" entre las técnicas de optimización ya que evalúa todas las combinaciones posibles dentro de un conjunto predefinido de hiperparámetros. Esta exhaustividad, aunque garantiza que se consideren todas las posibles configuraciones, conlleva un alto costo computacional especialmente en modelos complejos, haciéndolo menos práctico en contextos con recursos o tiempo limitados. Por otro lado, delimitar previamente el espacio de búsqueda fue clave, no todas las combinaciones de hiperparámetros son inherentemente compatibles; como fue mencionado anteriormente resultan en configuraciones subóptimas durante el proceso de entrenamiento.

Utilizando el mismo modelo cinético modificado de De Andrés-Toro et al. (1997), se generó en *Matlab* un segundo conjunto de datos al simular también cuatro perfiles de temperatura (ver Anexo 2), variando ligeramente los contenidos en *dataset* original, la Tabla 8 muestra los parámetros utilizados en la simulación (ver Figura 18).

Tabla 8. Perfiles de fermentación utilizados para simulación en *Matlab*, *dataset_evaluation_PoC*.

Perfil de Fermentación	Valores de Temperatura (°C)	Tiempo (h)
A	14.34, 14.86, 21.6, 21.25, 0.00	0.00, 147.13, 172.94, 198.78, 223.47
B	14.25, 13.17, 22.22, 22.03, 0.00	0.00, 151.46, 175.18, 197.07, 224.26
C	15.19, 15.54, 21.52, 22.94, 0.00	0.00, 151.46, 175.18, 197.07, 224.26
D	13.53, 13.97, 21.7, 21.51, 0.00	0.00, 150.28, 178.76, 194.64, 216.3
Concentración de Sustrato (g/L)		**Conteo de Levadura (g/100 L)**
96 : 108, cada 4 g/L		100
Total de Puntos		220, aproximadamente cada 60 minutos

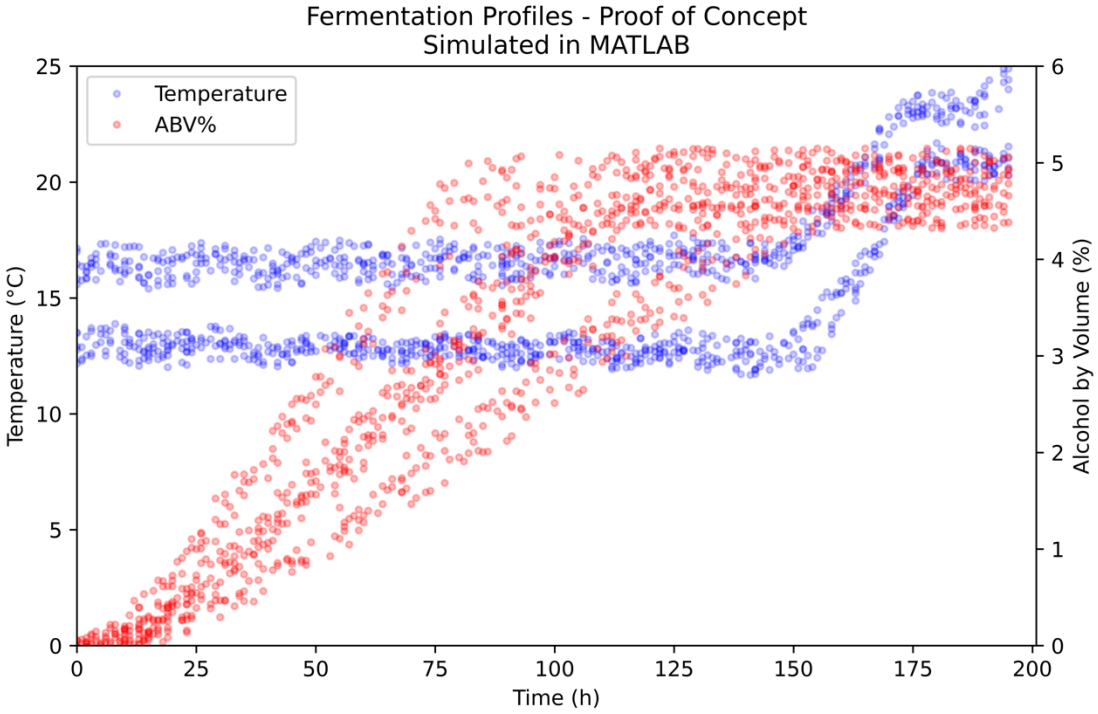


Figura 18. Perfiles de fermentación A - D (Tabla 8) del conjunto de datos ('dataset_evaluation_PoC'), drop-rate 80%, resultado de simulación en Matlab de modelo de Andrés-Toro et al. (1997).

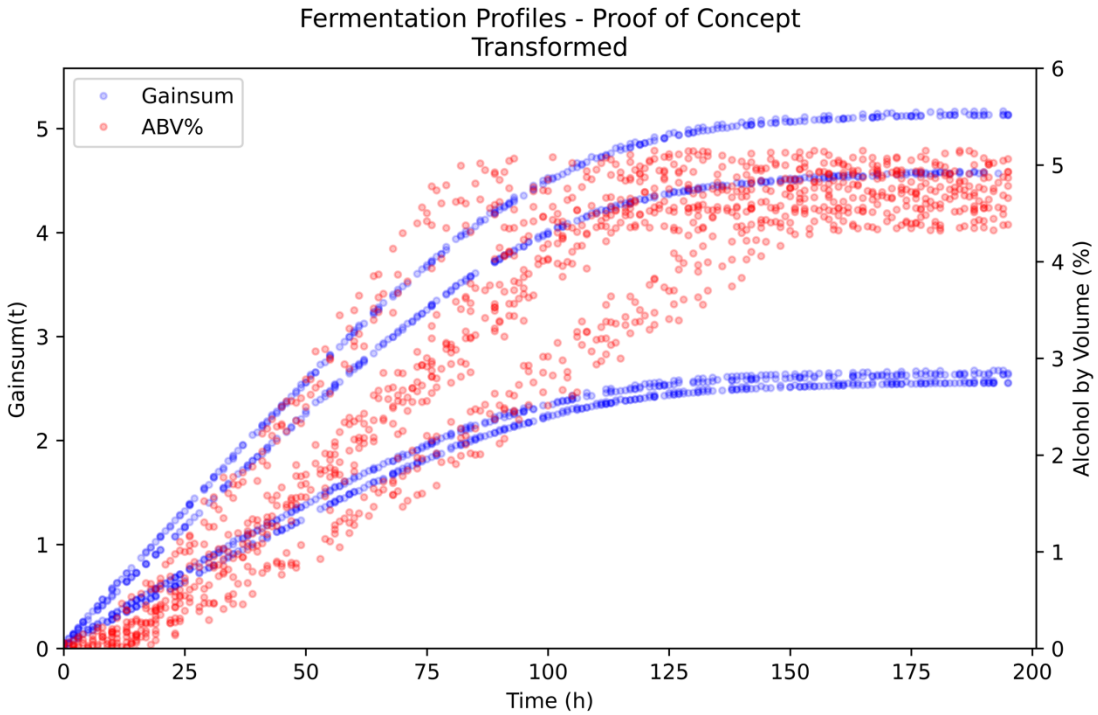


Figura 19. Exploración de relación entre función de transformación acumulada (nombrada Gainsum) y alcohol por volumen (ABV%), drop-rate 80%, para diferentes perfiles de fermentación A - D, 'dataset_evaluation_PoC'.

Los datos se procesaron y transformaron de la misma forma utilizada en *'dataset_training_PoC'* para generar un conjunto de datos (*'dataset_evaluation_PoC'*) diferente al de entrenamiento para evaluación del modelo de red neuronal *NARX* optimizado mostrado en Tabla 7 y Figura 17. Así también en Anexo 9, con *'dataset_evaluation_PoC'* se exploró la relación entre la función de transformación acumulada (nombrada *Gainsum*) y el alcohol por volumen (*ABV%*) en la Figura 19.

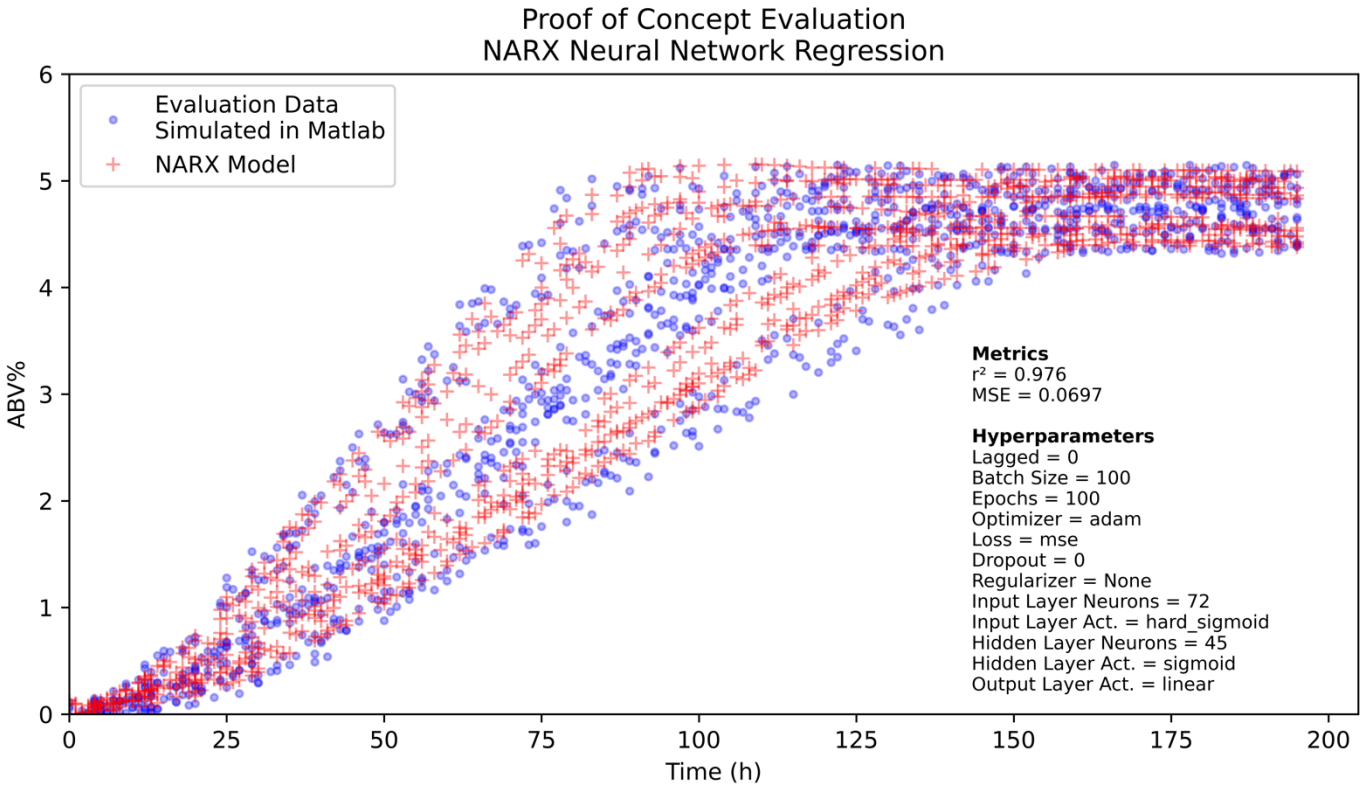


Figura 20. Evaluación de conjunto de datos de entrenamiento, 'dataset_evaluation_PoC', con modelo de red neuronal NARX optimizado, Prueba de Concepto.

Los resultados obtenidos de la evaluación del modelo *NARX* optimizado demostraron un alto grado de ajuste entre las predicciones y los valores de *'dataset_evaluation_PoC'* (Anexo 10), se presentan coeficientes de determinación r^2 de 0.976 y métrica *MSE* de 0.0697, esto destaca la capacidad del modelo para capturar la dinámica compleja de la fermentación y predecir con precisión el contenido de alcohol por volumen (*ABV%*) utilizando datos sencillos de adquirir al inicio del lote y únicamente procesando la temperatura tomada en tiempo real (ver Figura 20).

La evaluación sugiere que el modelo maneja eficazmente las variaciones inherentes al proceso de fermentación, respaldando la Prueba de Concepto y la viabilidad del sistema *BreweryHub* para ser aplicado en entornos de producción reales; una solución basada en *Machine Learning* para monitorear la etapa de fermentación en cervecerías artesanales.

4.2 Organización de la información obtenida

4.2.1. Integración de sistema *BreweryHub*

Paralelamente a la Prueba de Concepto enfocada en el *Software*, descrita en la primer parte de los hallazgos del capítulo 4 entre las etapas de generación de datos simulando fermentaciones, preprocesamiento de estos y validación de algoritmos de Aprendizaje Automático *NARX*; se integró el *Hardware del sistema BreweryHub* con *Raspberry Pi Zero W* como componente principal (ver Tabla 2, en sección 3.3).

Este microprocesador (Figura 21), equipado como un pequeño ordenador *LINUX* capaz de ejecutar rutinas y códigos de *Python*, no sólo cuenta con almacenamiento en tarjeta *microSD* y conectividad inalámbrica (*Wi-fi*), sino también con pines *GPIO* (*General Purpose Input/Output*), para comunicación directa con equipos externos. Estas entradas y salidas digitales permitieron la adquisición de datos de temperatura en tiempo real mediante termopares tipo K y módulos *MAX6675* (Figura 22), convertidores termopar-a-digital, cada *Raspberry Pi Zero W* con capacidad de integrar directamente dos de estos módulos (ver en Tabla 9 conexiones de terminales).

Se desarrolló un programa ejecutable como servicio en el *Raspberry Pi Zero W*, para disminuir el consumo de recursos (procesador, memoria y energía), para que iniciara automáticamente al energizar el dispositivo y buscara la conexión a la red inalámbrica local, preconfigurando las credenciales del sitio de instalación. Este programa, ver Anexo 18, *BreweryHub v1.0.0* tiene la funcionalidad de adquirir registros de temperatura, obtenidos periódicamente cada 120 segundos, de dos conjuntos termopar tipo K y módulo *MAX6675* (idealmente dos tanques de fermentación distintos), para almacenar estos localmente en una base de datos formato *CSV* (*Comma-Separated Values*).

Tabla 9. Conexiones de terminales de entrada de módulo *MAX6675* a *Raspberry Pi Zero W* utilizadas por sistema *BreweryHub*.

Pin / Termopar	No. 1	No. 2
GND	Pin 6, *Ground*	Pin 34, *Ground*
VCC	Pin 1, 3.3V	Pin 17, 3.3V
SCK	Pin 23, *GPIO11*	Pin 37/8, *GPIO20*
CS	Pin 24, *GPIO8*	Pin 12, *GPIO18*
MISO	Pin 21, *GPIO9*	Pin 35, *GPIO19*

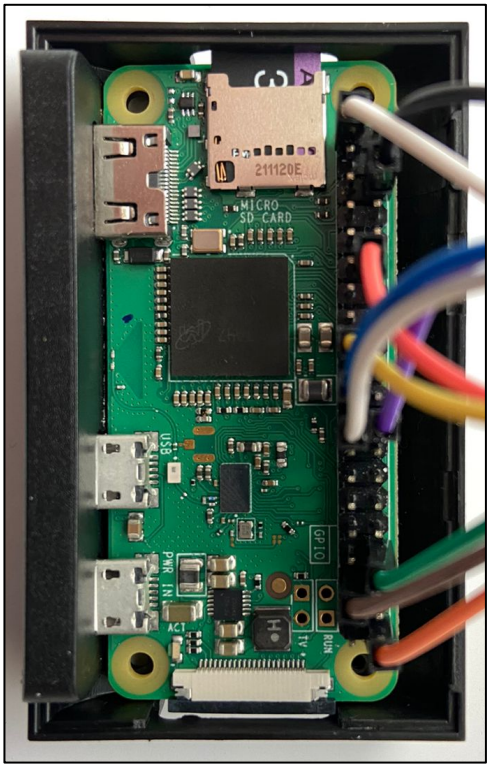


Figura 21. Raspberry Pi Zero W integrado en sistema BreweryHub.

Figura 22. Módulo MAX6675 integrado en sistema BreweryHub.

La memoria física del *Raspberry Pi Zero W* en la realidad funge como respaldo o para garantizar la integridad de los datos durante la intermitencia de red inalámbrica o *Cloud*, ya que la funcionalidad principal es la transmisión en tiempo real de los registros a una base de datos de la nube de *InfluxDB Cloud*. El programa envía vía *email* reportes periódicamente de manera automática con los perfiles de

temperatura, adjuntado una gráfica generada, así como una copia hasta el momento del archivo *CSV*. Para esta funcionalidad se pre-configuran las credenciales de un cliente *Gmail*, y una lista de receptores con las direcciones de correo electrónico.

El sistema está diseñado para enviar notificaciones cuando se detectan condiciones anómalas por umbrales de temperatura personalizados para cada sensor (límites inferior y superior pre-configurados), asegurando la detección automática y oportuna de cualquier exceso en los límites establecidos. En caso de anomalías durante un cierto número consecutivo de registros, se generan alertas detalladas con la identificación del tanque (sensor instalado) y el umbral comprometido, incluyendo una gráfica de tendencia, que se envían automáticamente vía *email* a una lista predefinida B de destinatarios.

Además, se ha implementado la detección de tasas de cambio anormales (*rate*), como un aumento o descenso superior a 3°C/min, que podrían indicar fallos en una electroválvula del circuito de glicol o del sistema de refrigeración (*chiller* eléctrico), permitiendo a los usuarios una intervención temprana. Cualquier alerta se registra en una bitácora local (*log* en *CSV*) que es independiente a una posible intermitencia de la base de datos en la nube, aunque el envío por correo electrónico depende de la conexión a la red disponible.

Estas funcionalidades de registro y alertas resultan particularmente beneficiosas para las cervecerías artesanales, donde, a pesar de que los sistemas de control del *Cellar*, área de fermentación, sean digitales, comúnmente carecen de conectividad para su monitoreo. Las temperaturas de tanques de fermentación, aunque adquiridos en todo momento e incluso en varios puntos para un mismo recipiente, tienden a ser subutilizados o directamente no utilizados más allá del *loop* de control por la ausencia de herramientas para su análisis, convirtiéndose en lo que se denomina datos oscuros (*Dark Data*). Estos datos, aunque valiosos, no contribuyen al conocimiento acumulativo de la cervecería ni permiten realizar análisis históricos para la toma de decisiones basadas en datos. El sistema *BreweryHub* aborda esta brecha al transformar el *Dark Data* en información aprovechable, permitiendo a los cerveceros artesanales no solo analizar tendencias históricas sino también reaccionar a desviaciones en tiempo real y, una vez con información suficiente, posibilita la optimización de los procesos de fermentación basándose en evidencia empírica.

Como se menciona anteriormente, simultáneo al almacenamiento local, se transmiten los registros de temperatura a través de un *token* (credencial) a *InfluxDB Cloud* para alimentar una base de datos *SQL* (*Structured Query Language database)*. La arquitectura de esta plataforma está específicamente diseñada para almacenar, procesar y visualizar series de temporales en tiempo real. En caso de que el dispositivo *BreweryHub* sufra de intermitencia de señal o de servicios de Internet, los datos se guardan en un búfer temporal y hasta que se restablezca la conexión de nueva cuenta.

Un *bucket* estructurado en *InfluxDB Cloud*, es el contenedor donde se almacenan series de datos temporales, permite el acceso y la gestión de la base de datos para que a través de una *API* (*Application Programming Interface*) puedan consultarse y visualizarse desde otras plataformas en tiempo real, como *Grafana Cloud* (ver Figura 28), de código abierto de visualización y monitoreo de métricas a escala mediante *dashboards*. Estos paneles son personalizables, permitiendo adaptarse a las necesidades específicas de cada cervecero, más allá del monitoreo de parámetros, esta plataforma ofrece también alertas por límites de temperatura o de desconexión del dispositivo *BreweryHub* (tiempo sin actualización). Dichas alertas se pueden configurar para un segundo nivel de notificaciones a usuarios a través de correo electrónico, en paralelo a las integradas en el ejecutable en *Raspberry Pi Zero W*, garantizando una respuesta robusta ante cualquier eventualidad.

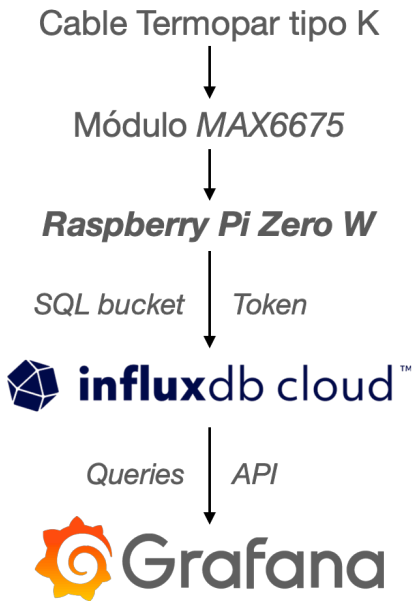


Figura 23. Esquema de integración de componentes y plataformas tecnológicas del sistema BreweryHub.

En la Figura 23 y 24 (Figura 3 de sección 3.2), se muestra cómo interactúan todos estos dispositivos, componentes, plataformas y tecnologías en el flujo de datos. En resumen, el sistema *BreweryHub* inicia con el registro en tiempo real de temperaturas de tanques de fermentación, a través de termopares tipo K y módulos *MAX6675*. Siendo el programa ejecutable en un microprocesador *Raspberry Pi Zero W* el corazón del sistema de adquisición de datos, al almacenarlos localmente y, además, transmitidos a *InfluxDB Cloud*, un servicio de base de datos especializado en la gestión de series temporales, para después ser visualizados a través de *Grafana Cloud*. Tanto el ejecutable *BreweryHub* como las estas plataformas en la nube permiten monitorear en tiempo el proceso de fermentación, pero en conjunto también ofrecen niveles redundantes de alertas configurables, facilitando una respuesta rápida a condiciones anómalas del proceso.

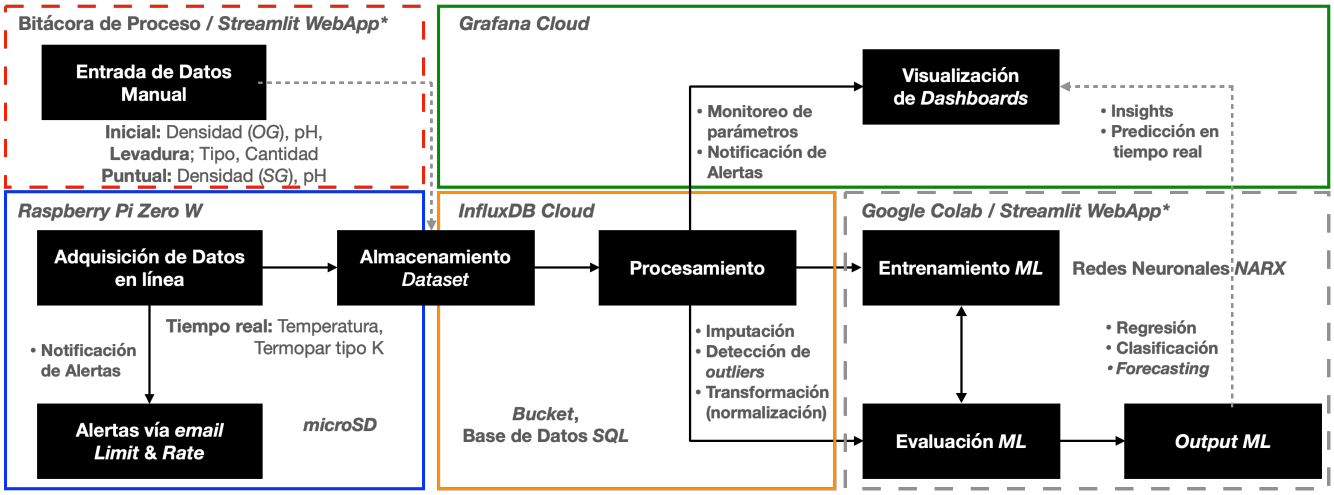


Figura 24 (sección 3.2). Flujo de información.

Un componente fundamental del sistema *BreweryHub* es la integración de datos procedentes de entradas manuales, que incluyen parámetros del proceso de bitácora de Cerveza Loba. Estos abarcan desde perfiles de fermentación, características del mosto y la levadura, hasta condiciones iniciales y mediciones específicas tomadas durante el proceso de fermentación. La Tabla 10 resume los parámetros cruciales que afectan la fermentación en la cervecería, destacando cuáles fueron considerados, medidos o no registrados dentro del ámbito de *BreweryHub*.

Tabla 10. Parámetros clave de fermentación identificados.

Parámetro	Tipo	Función	Técnica
Parámetros medidos o registrados			
Gravedad Relativa	*Off-line* Inicial, puntual	Contenido inicial de azúcares fermentables en mosto y avance de fermentación (conversión)	Hidrómetro, cuantitativo
Gravedad Relativa Final (*FG*)	Clase	Atenuación final esperada para fermentación	Cuantitativa
Perfil de Temperatura	Serie	Perfil de *set points* de temperatura esperados durante fermentación	Cuantitativa
Temperatura	Continua, puntual	Medición de temperatura en tanque de fermentación	Termopar tipo K, cuantitativa
pH	*Off-line* Inicial, puntual	Medida del mosto, afecta la actividad de la levadura	Medidor de pH, cuantitativa
Cepa	Clase	Tipo de levadura, define las características y el perfil de sabor	Genética, categórica
Conteo/*Pitch Rate*	Inicial	Cantidad de levadura inoculada, unidades celulares por mL de mosto	Cuantitativa
Parámetros no medidos			
Generación	Inicial	Indica número de reutilizaciones de levadura, puede afectar su rendimiento	Cualitativa
Vitalidad	Inicial	Describe la salud y la capacidad metabólica de la levadura	Microscopio, cualitativa
Viabilidad	Inicial	Mide la proporción de células vivas en la población de levaduras	Microscopio, cuantitativa
Presión del tanque	Puntual	Medición de presión en tanque de fermentación, indica actividad	Manómetro, cuantitativa
Oxígeno disuelto	*Off-line* Inicial, puntual	Mide la concentración de oxígeno en mosto, afecta el crecimiento de levadura	Sensor óptico, cuantitativa
Parámetros no registrados o considerados por *BreweryHub*			
Floculación	Clase	Capacidad de la levadura para aglutinarse y sedimentarse al finalizar fermentación	Genética, categórica
Tolerancia	Clase	Capacidad de la levadura para sobrevivir y continuar fermentando en altos *ABV%*	Genética, categórica
Perfil de *Mashing*	Clase	Perfil de temperaturas, descansos y tiempo de cocimiento del mosto	Categórica
IBUs	Inicial	Nivel de amargura, calculada por adiciones de lúpulo	Cuantitativa
Perfil del Agua	Inicial	Concentración de iones añadidos o presentes, y nutrientes en agua	Cuantitativa o categórica
Color	Inicial	Indicador de malteado de malta y tipo de cerveza	Cuantitativa o categórica

4.2.2. Implementación de sistema *BreweryHub*

El sistema *BreweryHub* se implementó con éxito en el *Cellar* de Cerveza Loba, instalándose específicamente entre los tanques F03 y F04 (ver Figuras 25 y 26), el 28 de septiembre de 2023, alcanzando operatividad plena el 3 de octubre del mismo año. Semanas después se instaló un segundo sistema, entre los tanques F08 y F09, el 26 de octubre, para aumentar la capacidad del proyecto para monitorear lotes de fermentación en diferentes ubicaciones. Los sensores termopar tipo K de cada *BreweryHub* se integraron dentro de los termopozos existentes, adaptándolos alrededor de los transmisores de temperatura del sistema de control local, como un sistema de medición en paralelo para no intervenir instrumentos ni impactar la operación de la cervecería.

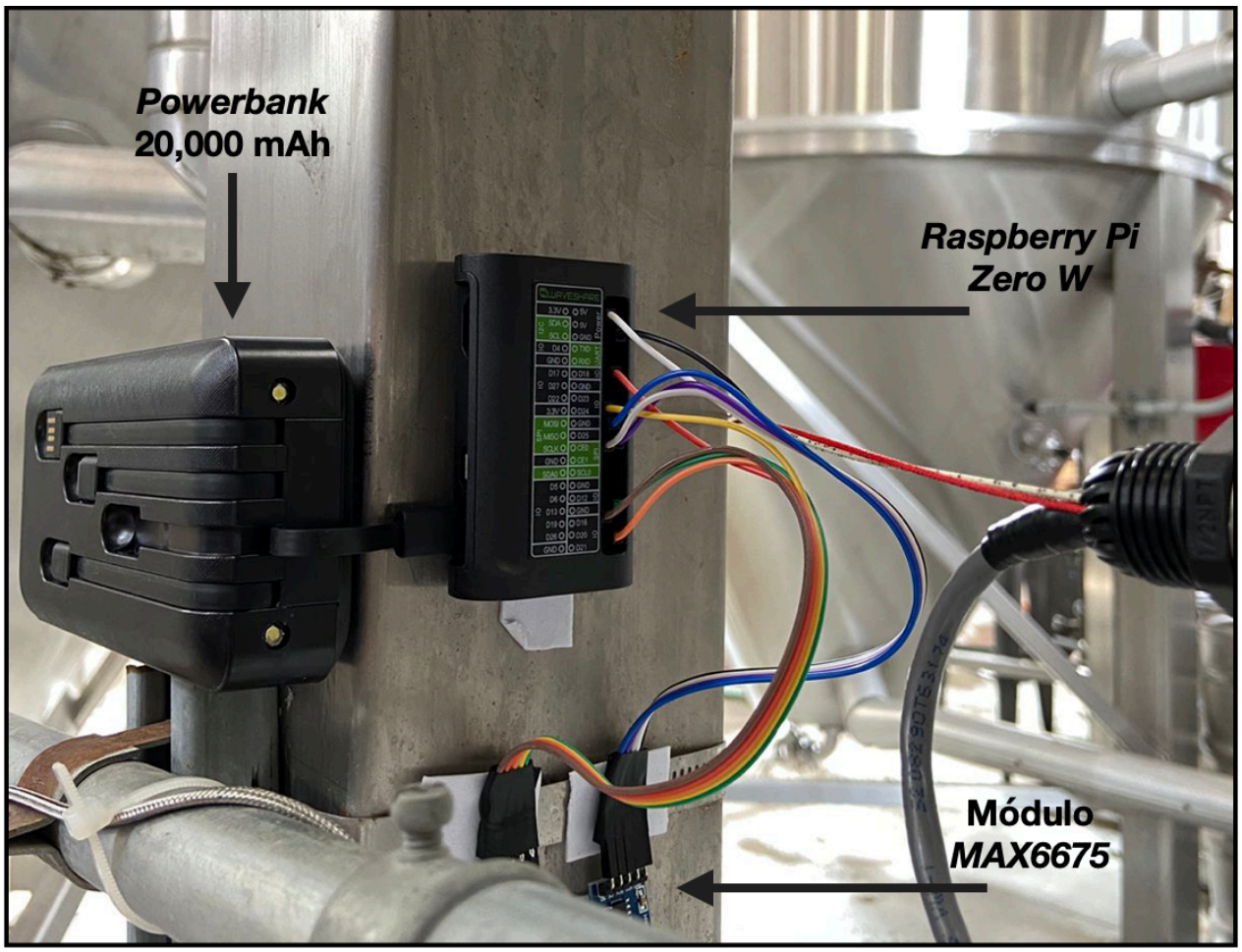


Figura 25. Primer sistema BreweryHub instalado en 28/09/2023 en tanques F03 y F04 de Cerveza Loba.

Figura 26. Primer sistema BreweryHub instalado en 28/09/2023 en tanques F03 y F04 de Cerveza Loba.

La configuración de la planta cervecera requirió el uso de baterías externas de 20,000 mAh para alimentar los dispositivos (*Powerbank*), dada la falta de acceso a fuentes de electricidad de bajo voltaje cercanas a los tanques. Estas dos baterías con autonomía aproximada de cuatro días cada una, alternándolas periódicamente para garantizar la continuidad operativa. Otra consideración, se instaló un repetidor de señal para extender el alcance de la red inalámbrica, y asegurar la transmisión de datos constante del *BreweryHub*, superando así los desafíos de intermitencia por la infraestructura existente.

En el periodo entre el 28 de septiembre y el 15 de noviembre de 2023, se llevó a cabo la recolección en automático y la transmisión a la nube de datos de temperatura de dos lotes de producción mediante el sistema *BreweryHub*. Los procesos y herramientas empleadas para la captura, almacenamiento, transmisión y visualización de los datos se detallan en la Sección 4.2.1. La Figura 27 muestra el panel que estuvo disponible en tiempo real de *Grafana Cloud* para la visualización de la temperatura en tanque F03 para el lote #395, con datos recabados cada 120 segundos y visualizados con una media móvil de 6 puntos. En las Figuras 28 y 29 se presentan los registros local de *Raspberry Pi Zero W* para ambos lotes #395 y #400, en F03 y F04 respectivamente, con media móvil de 4 horas.

Figura 27. Dashboard de Grafana Cloud del tanque de fermentación F03 (09/10/2023 - 13/10/2023). Datos registrados cada 120 segundos con media móvil (n=6).

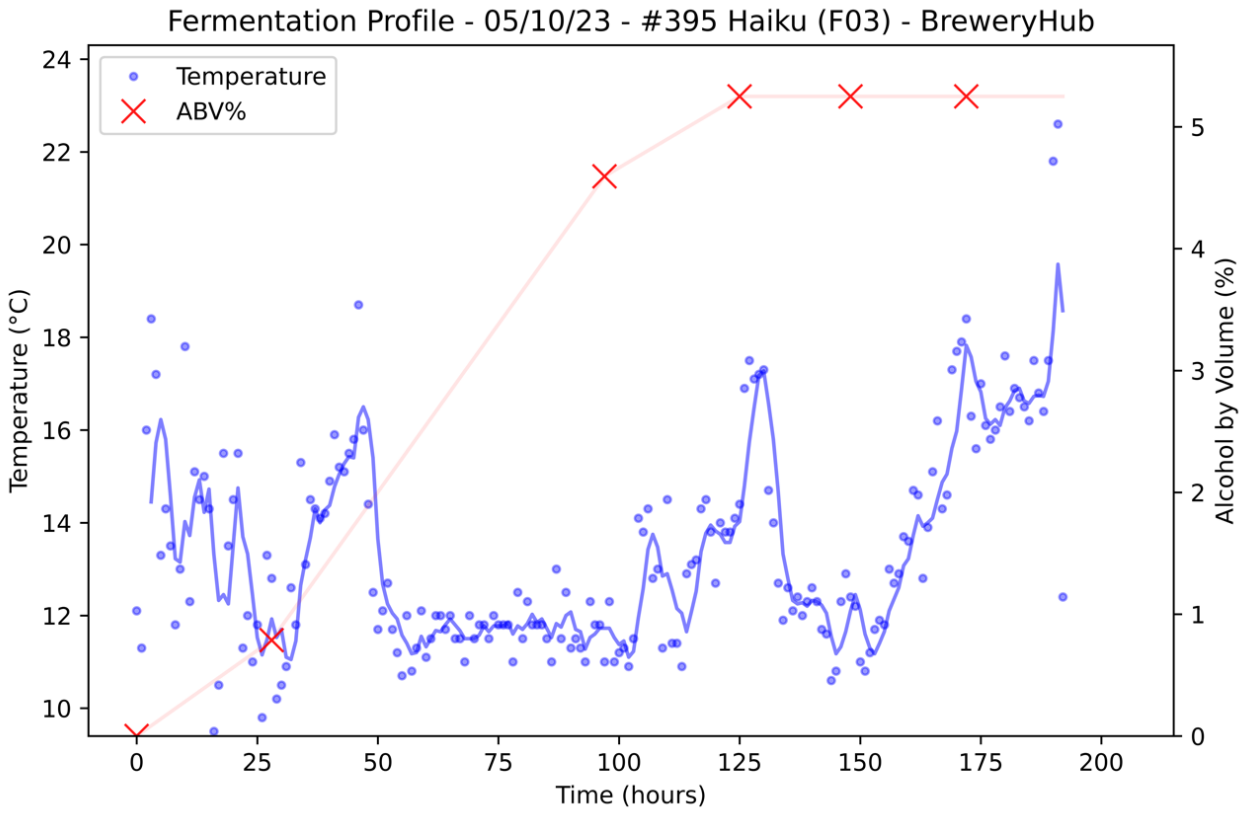


Figura 28. Registros de temperatura de lote #395 Haiku del 05/10/2023 al 14/10/2023, tanque F03. Datos obtenidos cada 120 segundos, promediados por hora y con media móvil (n = 4 horas).

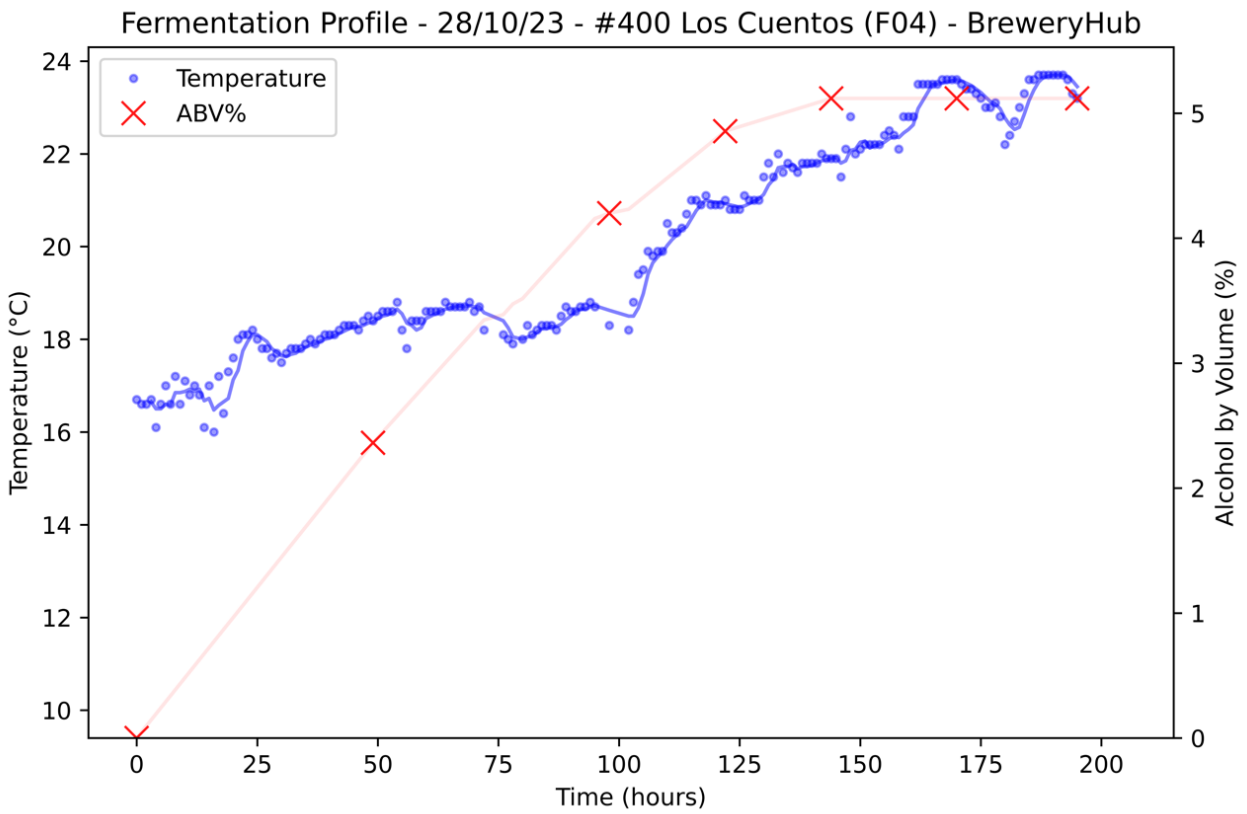
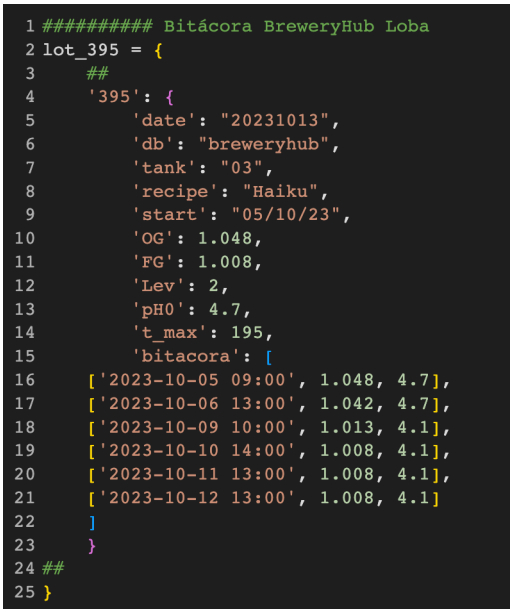


Figura 29. Registros de temperatura de lote #400 Los Cuentos del 28/10/2023 al 06/11/2023, tanque F04. Datos obtenidos cada 120 segundos, promediados por hora y con media móvil (n = 4 horas).

Complementariamente, se efectuó la entrada manual (*off-line*) de parámetros de la bitácora de proceso de Cerveza Loba, detallados en la Tabla 10, como la gravedad relativa inicial (*OG*) y la final esperada (*FG*), y el pH inicial, además de mediciones puntuales de gravedad relativa (*SG*) y pH realizadas durante la fermentación. Estos datos se consolidaron en un diccionario (Figura 30 muestra un ejemplo para el lote #395), repositorio llamado así por su estructura de datos, y se sometieron a un tratamiento que abarcó exploración de datos, imputación de valores faltantes, detección de valores atípicos (*outliers*), normalización temporal y las transformaciones especificadas en las Ecuaciones 16 a 23. El resultado fue el conjunto de datos denominado *'dataset_brew'*, que incorpora los atributos descritos en la Tabla 6, tal como se explica en la Sección 4.1.



```
######### Bitácora BreweryHub Loba
lot_395 = {
    ##
    '395': {
        'date': "20231013",
        'db': "breweryhub",
        'tank': "03",
        'recipe': "Haiku",
        'start': "05/10/23",
        'OG': 1.048,
        'FG': 1.008,
        'Lev': 2,
        'pH0': 4.7,
        't_max': 195,
        'bitacora': [
    ['2023-10-05 09:00', 1.048, 4.7],
    ['2023-10-06 13:00', 1.042, 4.7],
    ['2023-10-09 10:00', 1.013, 4.1],
    ['2023-10-10 14:00', 1.008, 4.1],
    ['2023-10-11 13:00', 1.008, 4.1],
    ['2023-10-12 13:00', 1.008, 4.1]
    ]
    }
##
}
```

Figura 30. Extracto de entrada de datos manuales, parámetros de proceso del lote #395, PI_BreweryHub_DataAnalysis_v1.0.ipynb (Anexo 16).

Tabla 11. Parámetros de fermentación esperados para lotes de producción #395 y #400, del 28/09/2023 al 15/11/2023.

Receta / Lote	Gravedad Relativa		Levadura		Perfil de Fermentación	
	OG	***FG* esperada**	**Tipo**	**Inoculación (mL)**	***Setpoint* (°C)**	**Tiempo (h)**
Haiku #395 05/10/2023	1.048	1.008	*Lager W34-70*	4,500 en 4,500 L	12.5 - 12.5, 18.0 – 18.0	0 - 150, 151 – 200
Los Cuentos #400 28/10/2023	1.052	1.012	*Ale US-05*	6,000 en 2,500 L	19.0 - 19.0, 19.0 – 22.0	0 - 100, 101 – 200

En un paso anterior, con conocimiento de los parámetros iniciales del mosto, las expectativas de atenuación, las características de la levadura a inocularse y los perfiles de temperatura para la fermentación de los lotes #395 y #400, tal como se detalla en la Tabla 11. Empleando una metodología paralela a la utilizada en la Prueba de Concepto, se llevaron a cabo simulaciones en Matlab de seis perfiles de temperatura, presentados en la Tabla 12 y cuyo desarrollo se puede consultar en el Anexo 2, se ajustaron a distintas concentraciones iniciales de sustrato (ver Figuras 31 y 32).

Los datos resultantes se sometieron a un proceso de corrección y tratamiento detallado en la Sección 4.1, culminando en la creación del conjunto de datos *'dataset_training_brew'*, disponible en el Anexo 11. A este nuevo conjunto se le integraron *'dataset_training_PoC'* y *'dataset_evaluation_PoC'* previamente utilizados en secciones anteriores, estos dos últimos con un factor de reducción adicional (*drop-rate*) del 50% para ponderar más a los perfiles similares a los esperados de fermentaciones de lotes #395 y #400 en el entrenamiento y optimización.

Tabla 12. Perfiles de fermentación utilizados para simulación en *Matlab*, antes de corrección de temperatura en *'dataset_training_brew'*.

Perfil de Fermentación	Valores de Temperatura (°C)	Tiempo (h)
A	14.0, 19.0, 20.5, 20.5	0, 150, 175, 195
B	14.0, 14.0, 19.5, 14.0, 14.0, 19.5, 19.5	0, 25, 35, 50, 150, 175, 195
C	14.5, 19.5, 20.5, 20.5	0, 150, 175, 195
D	14.5, 14.5, 19.5, 14.5, 14.5, 19.5, 19.5	0, 25, 35, 50, 150, 175, 195
E	15.0, 20.0, 21.0, 21.0	0, 150, 175, 195
F	13.5, 13.5, 19.0, 13.5, 13.5, 19.0, 19.0	0, 25, 35, 50, 150, 175, 195

Concentración de Sustrato (g/L)	Conteo de Levadura (g/100 L)
88 : 112, cada 2 g/L	100
Total de Puntos	220, aproximadamente cada 60 minutos

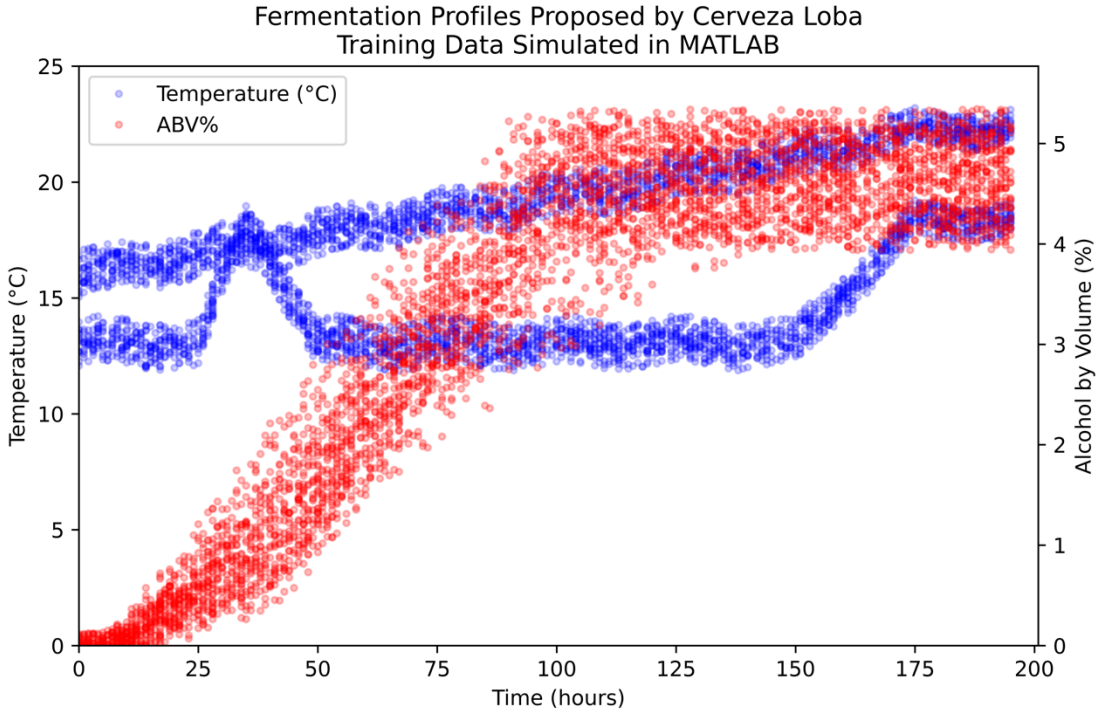


Figura 31. Perfiles de fermentación A - F (Tabla 12) del conjunto de datos 'dataset_training_brew', drop-rate 70%, resultado de simulación en Matlab de modelo de Andrés-Toro et al. (1997).

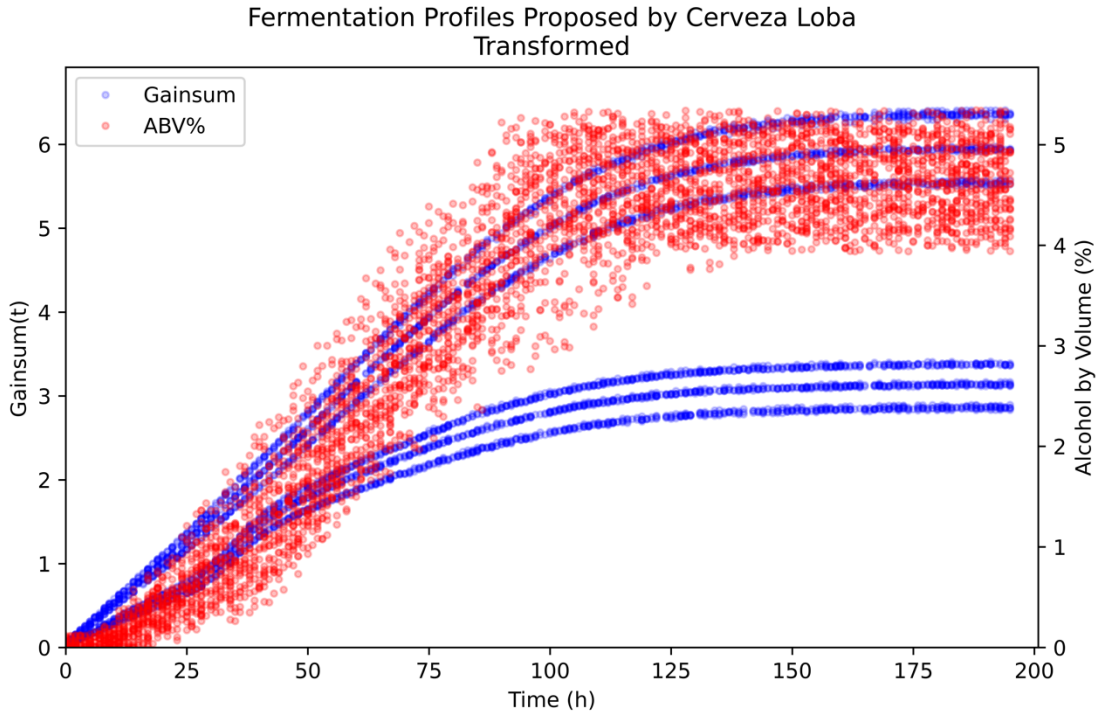


Figura 32. Exploración de relación entre función de transformación acumulada (Gainsum) y alcohol por volumen (ABV%), drop-rate 70%, para diferentes perfiles de fermentación A - F, 'dataset_training_brew'.

El conjunto de datos resultante de la combinación, con atributos mencionados en Tabla 6, fue utilizado como entrada para una nueva búsqueda exhaustiva (*grid*), evaluando rendimiento de combinación de hiperparámetros por Error Cuadrático Medio (*MSE*, por sus siglas en inglés), repitiendo el proceso detallado en sección 4.2.1. (Anexo 12), resultados son resumidos en el *Heatmap* de la Figura 33.

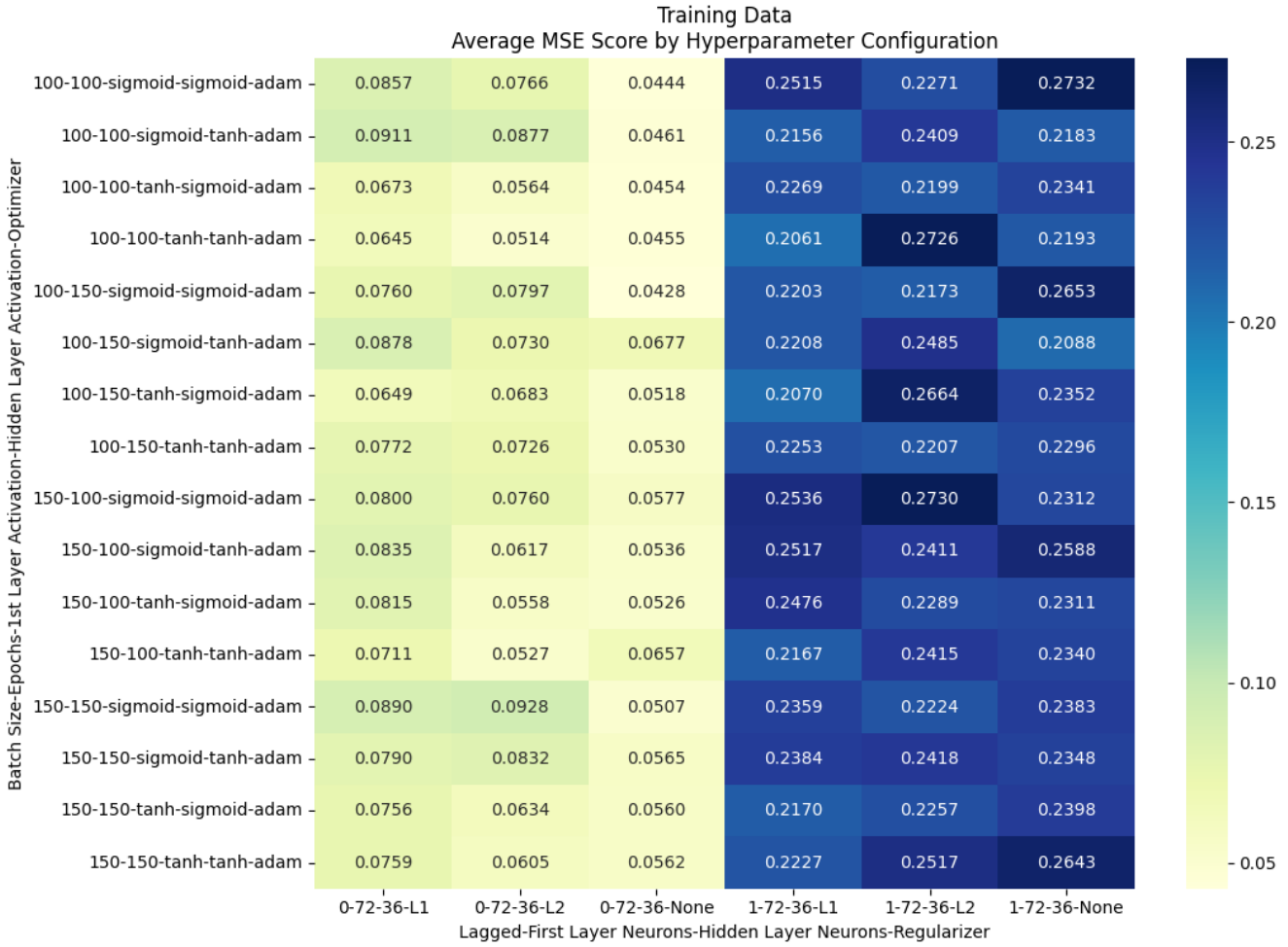


Figura 33. Resumen de MSE obtenida por configuración de hiperparámetros de primer búsqueda grid, para entrenamiento de 'dataset_training_brew'.

De igual manera a la Sección 4.2.1, se lleva a cabo una segunda fase de búsqueda *grid* para encontrar el número de neuronas en la primera capa y capa oculta (*First Layer Neurons* y *Hidden Layer Neurons,* respectivamente) compatible con la combinación de hiperparámetros encontrada anteriormente. Este proceso se ilustra en el código del Anexo 13, y los resultados se detallan en la Tabla

13. Los resultados de desempeño de esta segunda optimización se resumen en el *Heatmap* de la Figura 34.

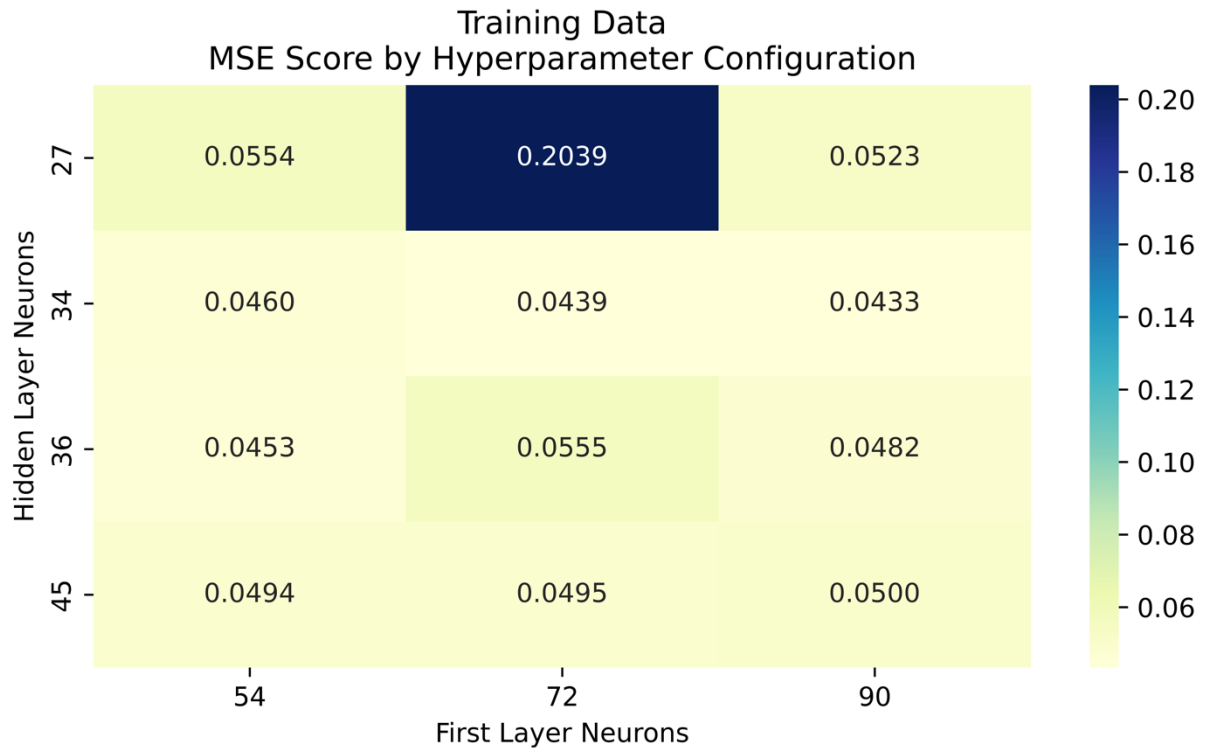


Figura 34. Resumen de MSE obtenida por configuración de hiperparámetros de segunda búsqueda grid, para entrenamiento de 'dataset_training_brew'.

La Tabla 13 proporciona un resumen de la configuración óptima para el modelo *NARX* tras ambos procesos de búsqueda *grid*. De igual forma que durante la Prueba de Concepto, los hiperparámetros seleccionados indican que el modelo prioriza la entrada actual, marcada por la ausencia de valores rezagados (*Lagged* = 0), aunque esto entra en conflicto con la idea que en modelos de procesos dinámicos como la fermentación, las condiciones presentes podrían ser más indicativas que la historia pasada, este acercamiento deberá de continuarse explorando en futuras aplicaciones de *BreweryHub*. El modelo apunta a un diseño que equilibra la complejidad y la capacidad de generalización, con un tamaño de lote moderado (*Batch Size* de 100) y un número de épocas (*Epochs* de 150) que sugieren una búsqueda meticulosa de precisión y convergencia sin caer en el sobreajuste, facilitado por la omisión de un regularizador y técnica de abandono.

Tabla 13. Hiperparámetros óptimos para el modelo *NARX* producto de segunda etapa de optimización (Figura 34), *'dataset_training_brew'*.

Hiperparámetro	Valor	Hiperparámetro	Valor
Valores Rezagados	0	Regularizador	*None*
Tamaño de Lote	100	Neuronas de Capa Entrada	90
Épocas	150	Función de Activación de Capa de Entrada	*sigmoid*
Optimizador	*adam*	Neuronas de Capa Oculta	34
Función de Pérdida	*mse*	Función de Activación de Capa de Oculta	*sigmoid*
Técnica de Abandono	0	Función de Activación de Capa de Salida	*linear*
Métrica (*MSE*)		0.0285	

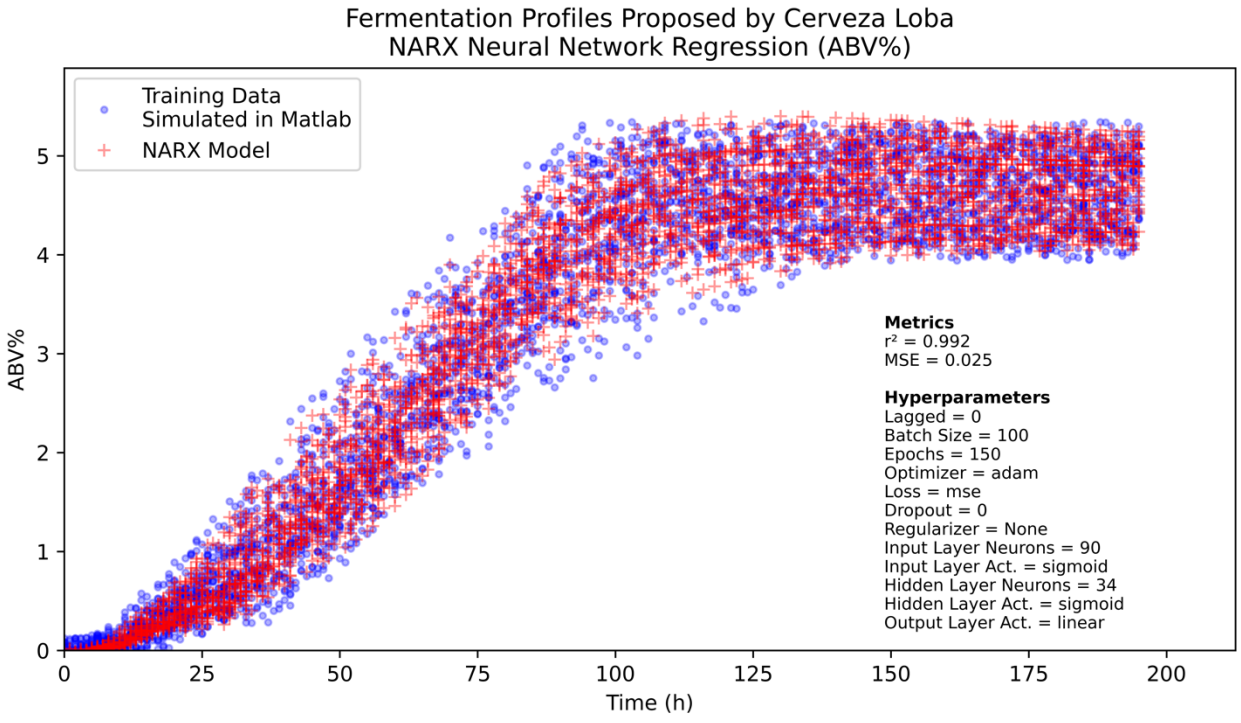


Figura 35. Evaluación de conjunto de datos de entrenamiento, 'dataset_training_brew', con modelo de red neuronal NARX optimizado (Anexo 14).

La arquitectura de la red neuronal, consistente en 90 neuronas en la primera capa y 34 en la capa oculta, está diseñada para procesar datos de alta dimensionalidad, típicos en los sistemas de fermentación complejos que emplean la codificación *One-Hot Encoder*. Esta configuración, junto con las funciones de activación *'sigmoid'* y *'sigmoid'*, permite al modelo captar patrones y transiciones no lineales dentro de los datos de fermentación, similar a los resultados de la optimización de la Prueba de Concepto. La métrica de rendimiento resultante *MSE* de 0.0285 subrayan la eficiencia del modelo en la predicción de datos de entrenamiento, lo cual se visualiza en la Figura 35.

Estos resultados de optimización revelan un modelo robusto y simplificado, capaz de capturar la esencia de los procesos fermentativos sin ser excesivamente complejo, un equilibrio clave para la implementación práctica de cómputo en pequeños microprocesadores como *Raspberry Pi Zero W* o de cómputo en la nube, además de aplicarse en entornos de producción de cerveza artesanal para predecir el avance de la fermentación y otros parámetros clave en tiempo real.

4.2.3. Evaluación de sistema *BreweryHub*

El modelo de red neuronal *NARX* de regresión para aproximar el % de alcohol por volumen (*ABV%*), generado en la Sección 4.2.2 producto del entrenamiento y búsqueda exhaustiva con *'dataset_training_brew'* es sometido a una evaluación práctica empleando el *dataset* de datos reales adquiridos por *BreweryHub* para los lotes #395 y #400, y de las entradas manuales de bitácora de proceso de Cerveza Loba, con base a Anexo 16, *'dataset_brew'*.

Los resultados de esta evaluación se visualizan en las Figuras 36 a 39, demostrando un alto grado de ajuste entre predicciones de *ABV%* y los valores empíricos, respaldando la Prueba de Concepto y la viabilidad del sistema *BreweryHub* para ser aplicado en entornos de producción reales; una solución basada en *Machine Learning* para monitorear el avance de fermentación en cervecerías artesanales, como se documenta en Anexo 17.

Se presentan coeficientes de determinación r^2 de 0.994 para el lote #395 y de 0.975 para el lote #400, métricas *MSE* de 0.0312 y 0.0822 respectivamente, esto destaca la capacidad de las técnicas de *Machine Learning* y en específico del modelo de red neuronal para capturar la dinámica compleja de la fermentación, predecir con precisión el contenido de alcohol por volumen (*ABV%*), detectar desviaciones y proveer *insights* operativos, utilizando condiciones iniciales del lote y procesando la temperatura en tiempo real en paralelo del sistema de control local.

Una anomalía común en la producción cervecera es que la levadura no metabolice los azúcares del mosto a una razón esperada en anteriores lotes, entre los parámetros que influyen en la actividad o desviaciones (ver también Tabla 10):

- Características del mosto: composición de malta, perfil y rendimiento de maceración (concentración de azúcares fermentables), oxígeno disuelto, química del agua (nutrientes y cofactores) y pH.
- Características de la levadura: cepa, generación, vitalidad y viabilidad.
- Cantidad inoculada de levadura.
- Temperatura de inoculación.
- Temperatura durante fermentación.

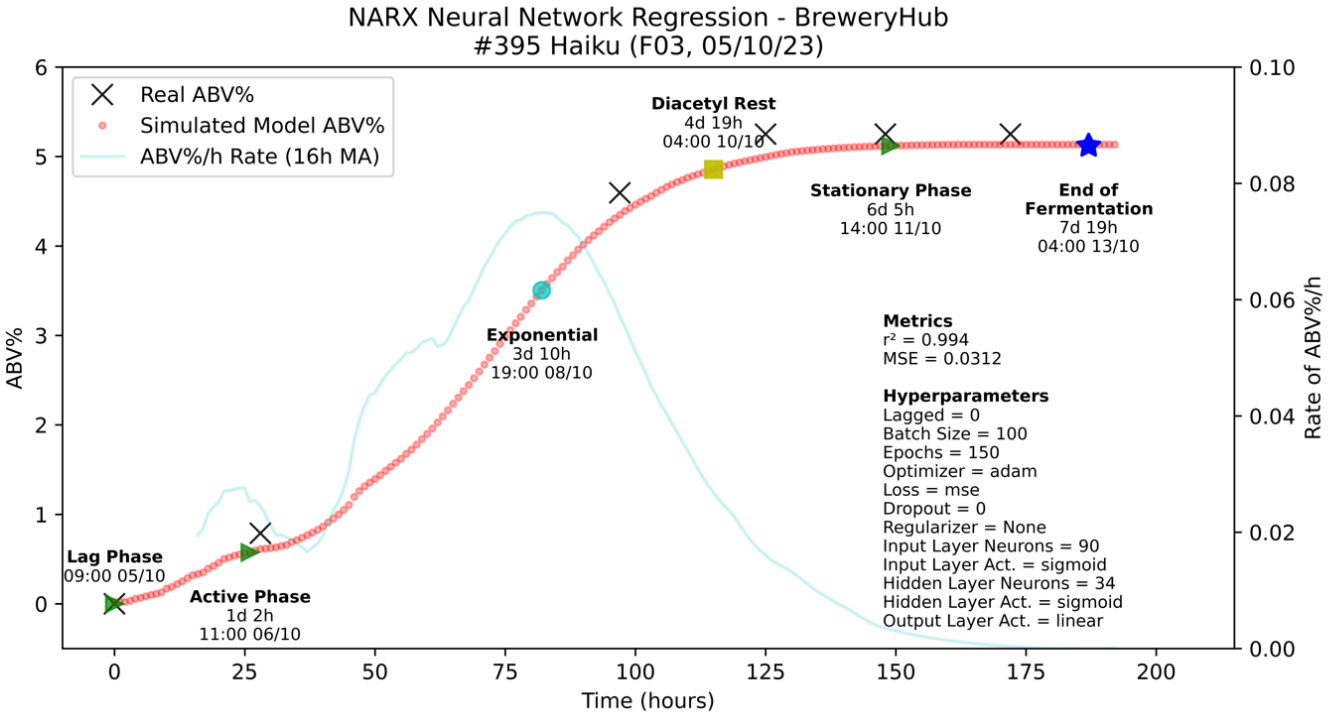


Figura 36. Evaluación de conjunto de datos 'dataset_brew' para lote #395 Haiku, con modelo de red neuronal NARX optimizado y avance de fermentación ABV%/h Rate (ver detalle en Anexo 17).

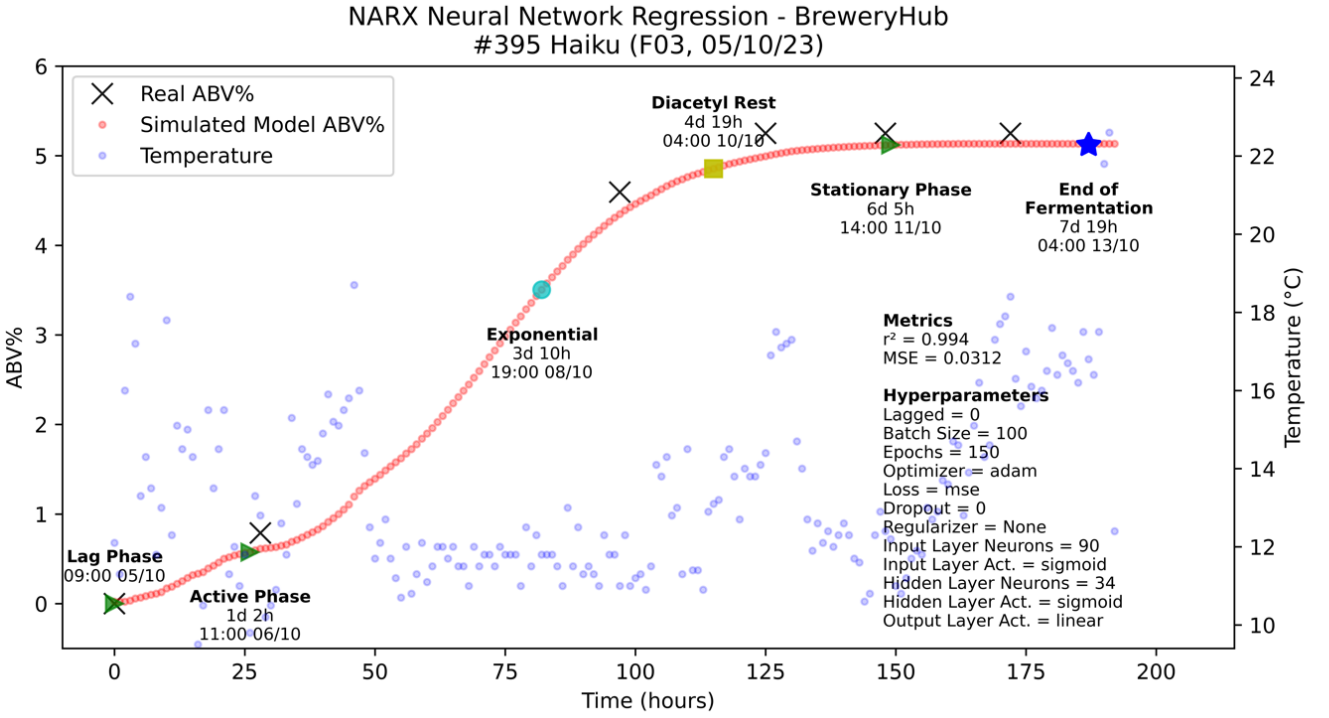


Figura 37. Evaluación de conjunto de datos 'dataset_brew' para lote #395 Haiku, con modelo de red neuronal NARX optimizado y perfil de temperaturas (ver detalle en Anexo 17).

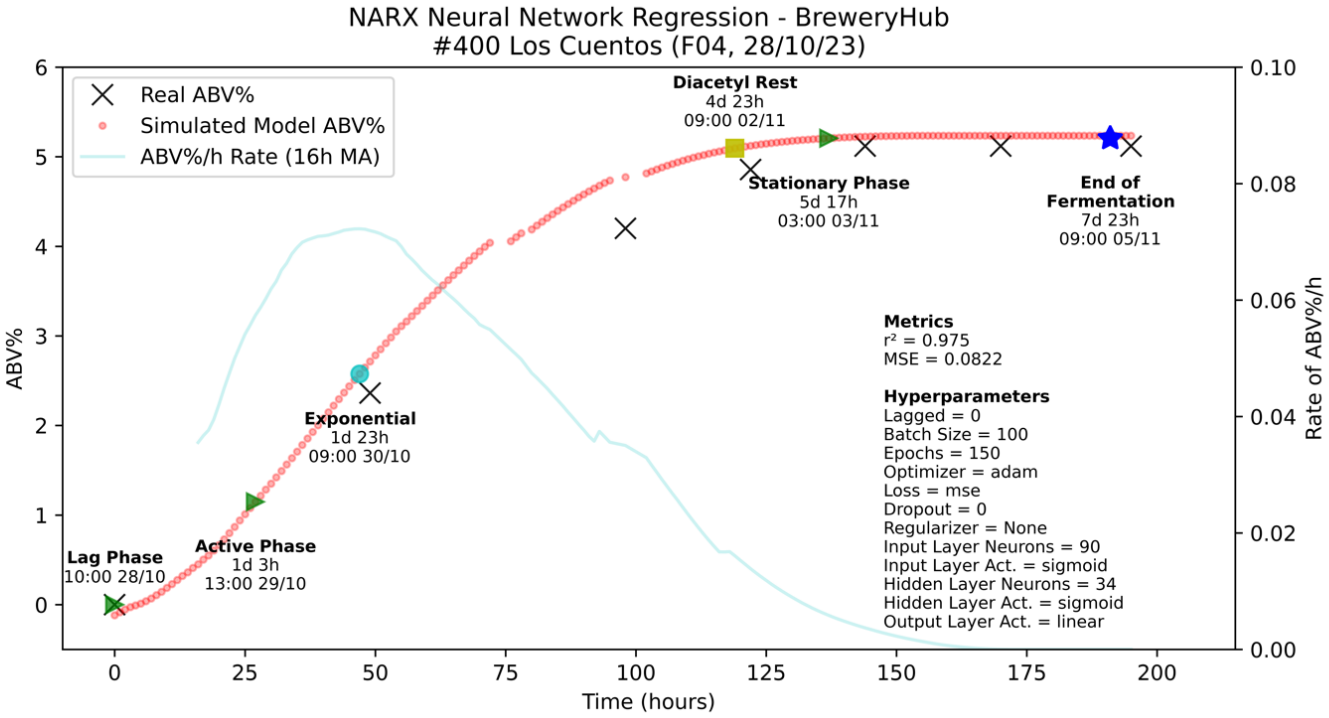


Figura 38. Evaluación de conjunto de datos 'dataset_brew' para lote #400 Los Cuentos, con modelo de red neuronal NARX optimizado y avance de fermentación ABV%/h Rate (ver detalle en Anexo 17).

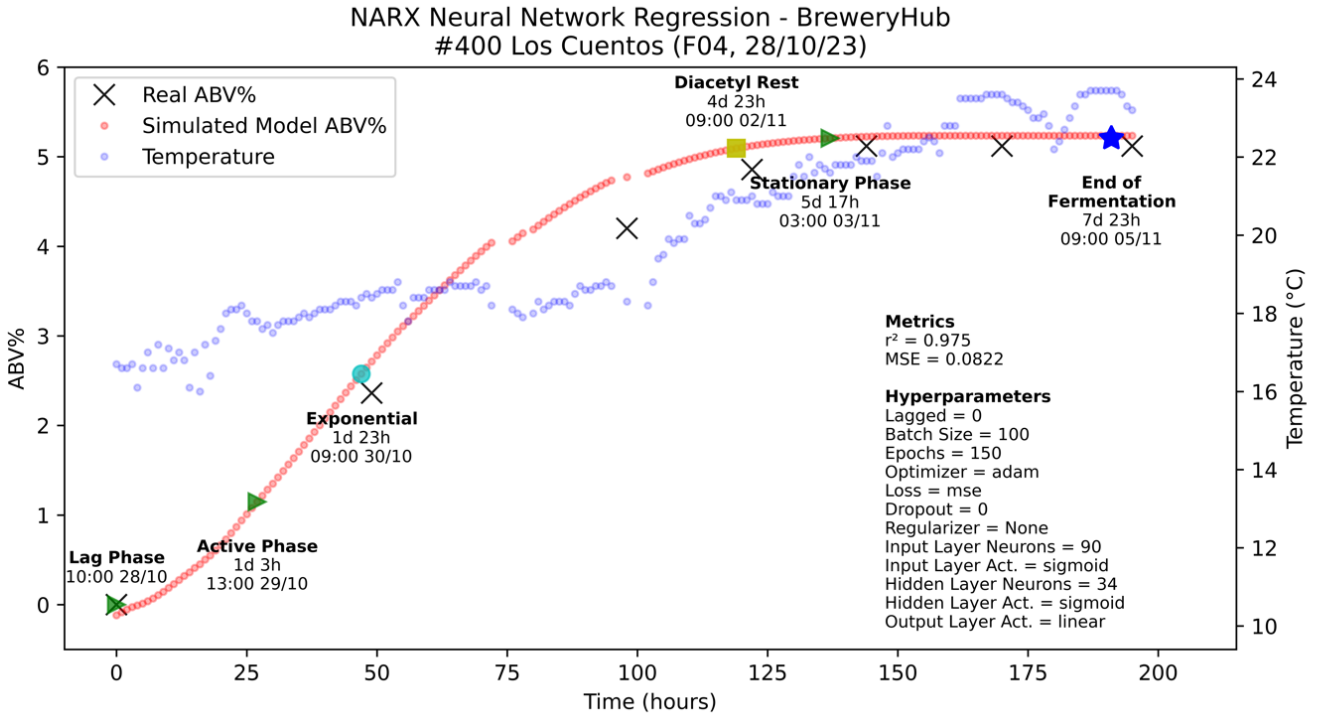


Figura 39. Evaluación de conjunto de datos 'dataset_brew' para lote #400 Los Cuentos, con modelo de red neuronal NARX optimizado y perfil de temperaturas (ver detalle en Anexo 17).

Entre estos *insights* operativos, está el análisis de la actividad de fermentación (*ABV%/h Rate*) que es una medida que representa el estado actual de la fermentación midiendo la tasa de disminución de la gravedad, en °P/h o *ABV%*/h. Esta métrica, análoga a la primer derivada de una función, indica la velocidad con la que se están convirtiendo los azúcares en alcohol y dióxido de carbono (CO_2), por ende, otro parámetro indirecto para medir la dinámica fermentativa es la medición del flujo de CO_2 o el aumento de presión del tanque, o cambio de masa en un tanque abierto.

La integración a *BreweryHub* del análisis de la evolución de este factor, *ABV%/h Rate*, en tiempo real y posiblemente en un futuro entre diferentes lotes de la misma receta o de similares perfiles de fermentación, se observa en Figuras 36 y 38. Convirtiéndose en una herramienta esencial para la toma de decisiones informada al proporcionar una visualización de tendencias. Entre estas perspectivas derivadas del análisis de datos:

1. Identificación el inicio de la conversión alcohólica (*Active Phase*) tras la fase de latencia (*Lag Phase*), esto permite detectar problemas de salud de levadura, insuficiente cantidad inoculada o desviaciones de temperatura. Además, esto se puede monitorizar remotamente y recibir alertas en tiempo real.

   En Anexo 17 se desarrolla el algoritmo para pronosticar el inicio de la fase activa, por lo tanto clasificar *Active Phase*, tras identificar un mínimo local en *ABV%/h Rate* aproximadamente después de las primeras 24 horas.

2. Detección del punto más alto de la conversión alcohólica (*Exponential Period*). En Anexo 17 se desarrolla en el algoritmo para pronosticar el periodo más vigoroso de la fermentación, al ser el máximo global de la tendencia *ABV%/h Rate.*

3. Pronóstico de condiciones para el comienzo del descanso de diacetilo (*Diacetyl Rest*), al aproximar el tiempo en que la conversión a *ABV%* se encuentre a 2°P de la gravedad relativa final esperada (*FG*).

   El descanso de diacetilo (*Diacetyl Rest*) es crucial en la elaboración de ciertos estilos de cerveza, especialmente tipo *Lager*, donde se eleva temporalmente la temperatura hacia el final de la fermentación primaria, de 24 a 72 horas, para permitir que las levaduras reabsorban y

metabolizen el diacetilo, un subproducto que puede impartir un sabor no deseado similar a la mantequilla, antes de pasar a la fase de maduración o *lagering.* En Anexo 17 se codifica una condición de búsqueda para marcar la ventana de inicio para este proceso.

4. Indicación de la finalización de la conversión alcohólica (*Stationary Phase*), fase donde el *ABV%* ha alcanzado el estimado de gravedad relativa final (*FG*). En Anexo 17 se desarrolla en el algoritmo para indicar el periodo menos vigoroso de la fermentación, al ser el mínimo global de la tendencia *ABV%/h Rate.*

5. Identificación del final de la fermentación, desplazando 72 horas la ventana para el inicio de descanso de diacetilo (*Diacetyl Rest*), indicativo que optimiza tiempos para procesos adicionales como *dry hopping*, cosecha de levadura y el *cold crashing*, mejorando la eficiencia en la producción y la rotación de fermentadores.

   La técnica de *Dry Hopping* consiste en adicionar lúpulos, en *pellets* o en copos, a la cerveza después de la fermentación principal, a diferencia de los que se agregan durante el *mashing* que contribuyen al amargor y estabilidad de la cerveza, estos adicionales se añaden para aumentar el aroma y el sabor a lúpulo sin añadir amargor adicional, al no isomerizar los ácidos alfa presentes. Especialmente popular en estilos *IPA*, de fermentación *Ale*.

   Mientras que el *Cold Crashing* es un método para clarificar la cerveza antes de embotellarla o envasarla. Después de que la fermentación ha terminado, la temperatura del fermentador se baja significativamente (a 0°C) durante varios días. Esto hace que la levadura y otras partículas en suspensión se precipiten y se asienten en el fondo del fermentador, resultando en una cerveza más limpia y clara. Empleado especialmente en fermentaciones *Lager*.

4.3. Impacto de la estrategia en la solución del problema

La implementación del sistema *BreweryHub* tiene la capacidad de transformar significativamente el proceso de fermentación en cervecerías artesanales. Mediante el monitoreo remoto en tiempo real desde plataformas como *InfluxDB Cloud* y los paneles de *Grafana Cloud*, posibilita a los cerveceros incluso desde un dispositivo móvil el identificar proactivamente ajustes necesarios a los parámetros críticos del proceso de fermentación, como *set points* de temperatura y el tiempo, lo que podría mejorar notablemente la calidad y consistencia del producto.

Las alertas implementadas por *BreweryHub*, tanto por el microprocesador como las plataformas *Cloud,* facilitarán la detección y mitigación temprana de desviaciones, lo que podría resultar en la reducción de errores, sobre-correcciones y, en última instancia, en la prevención de la pérdida de lotes.

En el futuro, con más tiempo de operación y acumulación de datos de múltiples lotes, se anticipa que se incorporarán ajustes que mejoren la efectividad del sistema, beneficiándose de la retroalimentación de los usuarios cerveceros. La proyección de integrar la plataforma *Streamlit* para generar una *WebApp* interactiva que se comunique con *InfluxDB Cloud y Grafana Cloud*, permitirá un cómputo en tiempo real de las herramientas de *Machine Learning* en la nube, con modelos posiblemente más enriquecidos a los presentados en las Figuras 36 a 39. Además, se planea el desarrollo de una *WebApp* para facilitar la entrada de datos de bitácora por parte del usuario, mejorando así la experiencia y la funcionalidad general del sistema incluso desde un dispositivo móvil.

La inversión requerida en *Hardware* se mantiene por debajo los 150 *USD*, para dos sensores de temperatura, y el uso de plataformas de código abierto (*Open Source*), prometen un impacto económico favorable al optimizar tiempos de procesos clave y reduciendo los costos operativos. Aunque estos beneficios aún no se han materializado en la práctica, el potencial indicado por la Prueba de Concepto y el despliegue inicial piloto en cervecería (Secciones 4.1 y 4.2, respectivamente) indican que *BreweryHub* tiene el potencial para convertirse en una herramienta valiosa para impulsar la eficiencia y competitividad en el sector de la cervecería artesanal.

4.3.1. Alineación con la estrategia general de la organización

El enfoque tradicional en la producción cervecera, que a menudo conduce a dejar la cerveza en el fermentador más tiempo del necesario para prevenir subfermentación y subproductos no deseados como el diacetilo, puede resultar en una utilización subóptima de los fermentadores —un periodo no confundible con *lagering* o maduración—. Sin monitoreo en tiempo real, *insights* y un análisis histórico, es difícil determinar con precisión cuándo ha terminado este proceso.

La calidad y consistencia de la cerveza dependen de diversos factores controlables, ya mencionados en secciones anteriores, como características de levadura y mosto, perfiles de maceración y de temperatura de fermentación. Sin embargo, existen variables no controlables, como fallos en el sistema de refrigeración o instrumentación del sistema de control local, que pueden impactar desde un tanque a toda la cervecería. *BreweryHub*, a través de su monitoreo continuo y alertas en tiempo real, permite a los cerveceros realizar ajustes proactivos, reduciendo el riesgo de intervenciones incorrectas o innecesarias y mejorando la consistencia del producto.

Además, *BreweryHub* abre un panorama basado en datos que contrasta con las prácticas convencionales que dependen de mediciones puntuales y observaciones en sitio. El sistema ofrece facilitar cambios de digitalización en la organización:

- Monitoreo continuo: La capacidad de visualizar datos en tiempo real permite a los cerveceros identificar tendencias, programar sus operaciones y realizar ajustes proactivos en lugar de reactivos, también, reduciendo el riesgo de sobre-correcciones.
- Alertas en tiempo real: En lugar de depender únicamente de supervisión en sitio, el sistema facilita la detección temprana de problemas potenciales como anomalías de temperatura y actividad fermentativa, al recibir notificaciones, posibilitando intervenciones oportunas para mantener la calidad del producto.
- Análisis de datos históricos: Comparando conjuntos de datos de múltiples lotes, se puede analizar el impacto de diferentes variables en la calidad de la cerveza y realizar ajustes informados en el proceso de fermentación.

- Consistencia en calidad del producto: Un monitoreo más cercano conduce a una fermentación más controlada, resultando en una cerveza de mayor calidad y consistencia.

Las aplicaciones de *BreweryHub* se enfocan en la optimización de procesos cerveceros y toma de decisiones basadas en el análisis de datos. Al contar con conjuntos de datos de múltiples lotes de la misma receta, los cerveceros podrán generar una nueva corriente de conocimiento de cada fermentación y su impacto en la calidad de la cerveza. Como los siguientes *insights* específicos:

1. Monitoreo de etapa de latencia (*Lag Phase*): Tras ingresar el mosto al fermentador, la levadura pasa de un estado de dormancia a actividad metabólica. Un tiempo de latencia prolongado o desviaciones en avances de fermentación puede ser indicativo de la necesidad de ajustar la tasa de inoculación o la viabilidad/vitalidad de la levadura. Monitorizar la actividad de fermentación elimina conjeturas, permitiendo decisiones basadas en datos.

2. Monitoreo de temperatura de fermentación: Controlar la temperatura desde el inicio de la fermentación es fundamental para la calidad de la cerveza. Un flujo de datos en tiempo real posibilita ajustes rápidos ante variaciones de temperatura que puedan estresar a la levadura.

3. Optimización del tiempo para descanso de diacetilo (*Diacetyl Rest*): Como se menciona en secciones anteriores, este es un compuesto que se forma naturalmente en las primeras fases de fermentación, considerado como un subproducto generalmente indeseable por lo que se busca controlar su nivel. Anteriormente (hace décadas) esto se lograba madurando la cerveza durante meses, pero actualmente en la elaboración de cerveza comercial se ha optimizado, logrando reducir sus niveles en pocos días al ajustar la temperatura al final de la fermentación. Mantener un control estricto de los niveles de diacetilo es esencial para la producción de cervezas de fermentación *Lager*.

4. Optimización del tiempo para *Dry Hopping*: Determinar el momento preciso es esencial para maximizar aromas y evitar riesgos como el "*hop creep*" por re-fermentación residual o contaminaciones, por lo que es vital esperar a que el pH descienda y haya presencia de alcohol para proteger contra crecimiento bacteriano.

5. Optimización del tiempo para cosecha de levadura: La recolección y reutilización de levadura puede representar un ahorro significativo. Comparando perfiles de fermentación entre lotes se puede evaluar el rendimiento de la levadura, determinando el final exacto de la fermentación y el momento ideal para la recolección. Retrasos en este proceso pueden reducir la viabilidad de la levadura.

6. Optimización del tiempo para *Cold Crashing*: Esta operación realizada en el momento adecuado ayuda a precipitar levaduras y otras partículas al fondo del fermentador, clarificando la cerveza. Sin embargo, hacerlo prematuramente puede afectar la calidad del producto.

5. Discusión final

El despliegue eficaz de *BreweryHub* ofrece avances significativos en el monitoreo del proceso de producción de cerveza artesanal con posibilidad para optimizar ciertos procesos dentro de este. El sistema, fundamentado en una técnica de *Machine Learning* en específico un modelo de red neuronal *NARX* bien afinado, ha evidenciado su capacidad para predecir avance de conversión e identificar anomalías en tiempo real, que pudiera reflejarse en mayor consistencia del producto final.

BreweryHub logra una adquisición y gestión de datos robusta y eficiente, combinando almacenamiento local y en la nube que facilita la monitorización remota y continua del proceso de fermentación, además que satisface plenamente el propósito de proporcionar una solución costeable para cervecerías artesanales, manteniendo el coste total por *Hardware* dentro del estimado de 150 *USD* por sistema, sin comprometer funcionalidad, ver información sobre costos detallada en Anexo 19.

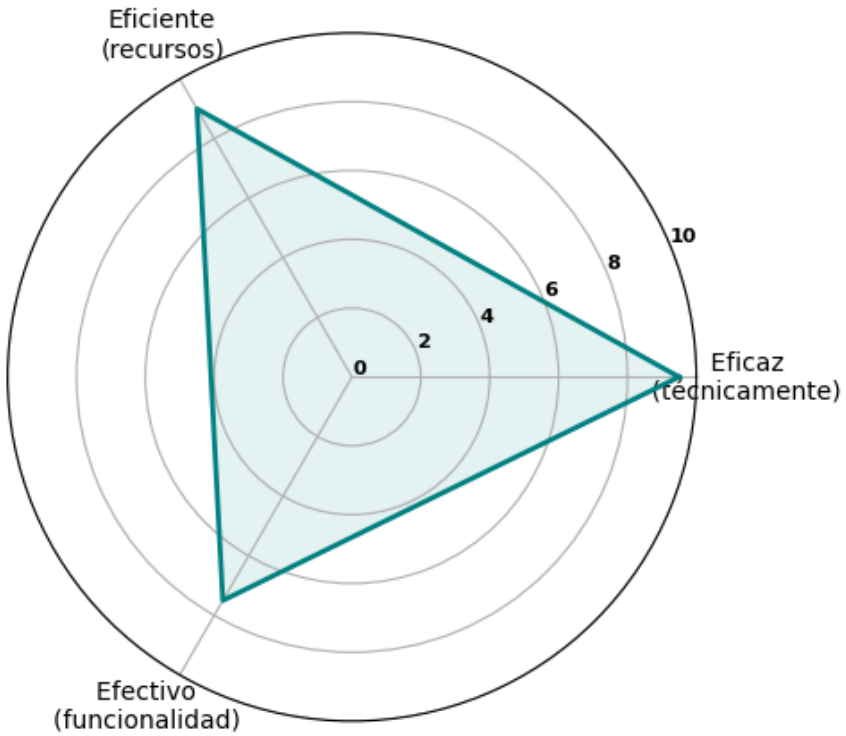


Figura 40. Autoevaluación Triple E ('3 Es') del impacto de implementación del sistema BreweryHub en Cerveza Loba.

Finalmente, la integración sinérgica de datos manuales y automáticos, procesados a través de métodos analíticos avanzados y herramientas de *Machine Learning*, brinda a los cerveceros artesanales una base sólida para tomar decisiones informadas. *BreweryHub* con el potencial de ser más efectivo conforme al tiempo de implementación, con más entrenamiento con datos reales, conocimiento empírico del proceso y la integración de los usuarios a sus operaciones, para reducir la intervención humana en el muestreo, optimizar de recursos como el tiempo empleado por equipo, eficiencia energética y reducir el riesgo de pérdida de lotes. Por ende, con potencial también para contribuir a la rentabilidad y competitividad de las cervecerías artesanales.

La Figura 40 presenta una autoevaluación del impacto de *BreweryHub* utilizando el marco de las '3 Es': Eficacia, Eficiencia y Efectividad, destacando las fortalezas y oportunidades del sistema. A través de este análisis, se puede apreciar que al sistema se le otorga un alto nivel de eficacia técnica, al cumplir con creces el objetivo del monitoreo remoto. En cuanto a la eficiencia, se destaca por uso de recursos, manteniendo accesible el costo de integrar el dispositivo. Aunque en términos de efectividad funcional del sistema demuestra competencia, es evidente la necesidad de una integración más profunda en la nube entre los módulos de monitoreo y las herramientas de Aprendizaje Automático, para proporcionar predicciones e *insights* en tiempo real. El involucramiento del usuario es otro aspecto crítico que requiere atención, indicando la necesidad de realizar ajustes para optimizar la funcionalidad y alcanzar el máximo potencial del sistema.

5.1. Consecuencias de la aplicación de la estrategia de innovación

La implementación de *BreweryHub* en Cerveza Loba ha iniciado un avance significativo en la digitalización y la monitorización remota de la fermentación (ver Figura 41, en Sección 4.2.2 encontrada como Figura 27 como ejemplo de un panel de visualización de *Grafana Cloud*), con potencial para ser escalable y extenderse a otras etapas del proceso cervecero como la maceración (*mashing*), y extrapolable a otros sectores de alimentos y bebidas con similares limitantes tecno-económicas a la industria de la cerveza artesanal.

Este cambio está respaldado por la integración de plataformas de visualización y dispositivos de adquisición de datos que alimentan modelos predictivos avanzados de Aprendizaje Automático (*Machine Learning*), como las redes neuronales *NARX*, para predecir la conversión del mosto y detectar anomalías, apoyando así la mejora en la consistencia del producto y la competitividad en el mercado.

La estrategia metodológica, centrada en el monitoreo remoto, responde a una necesidad no solo de Cerveza Loba sino identificada tras visitas en gran parte del sector de cervecerías artesanales, que a menudo enfrentan limitaciones técnicas de instrumentación. La incorporación de *Hardware* accesible y un *Software* que requiere bajos recursos de procesamiento, combinados con el uso de plataformas en la nube de uso libre para el almacenamiento y la visualización, así como el empleo de herramientas avanzadas de *Machine Learning* para análisis predictivo del avance de fermentación y generación de *insights* operativos, reflejan un enfoque innovador, incremental y adaptable. Estos elementos metodológicos han demostrado su eficacia al proporcionar a los cerveceros artesanales herramientas para una gestión más informada y proactiva de la fermentación.

Figura 41 (Sección 4.2.2). Dashboard de Grafana Cloud del tanque de fermentación F03 (09/10/2023 - 13/10/2023). Datos registrados cada 120 segundos con media móvil (n=6).

5.1.1. Aspectos de mejora para intervenciones subsecuentes

Pese a los avances durante el despliegue piloto con monitoreo remoto exitoso, se reconoce que la intervención tiene limitaciones en su funcionalidad, particularmente para que el usuario capture datos de su bitácora de procesos, y en el procesamiento *off-line* de las herramientas de Aprendizaje Automático (*Machine Learning*), y la generalización de los resultados debido a la diversidad de perfiles de fermentación en cervecerías artesanales.

Además, la operación de *BreweryHub* en fase piloto fue un desafío constante, tras superar las condiciones de instalación como se detalla en Sección 4.2.2, pero también en ajustes iniciales en *Software* para mejorar la funcionalidad, sin embargo, lo que requiere de mayor atención en futuras implementaciones es la transferencia completa del dispositivo a la cervecería, y solo intervenir en temas sobre la infraestructura de comunicación entre microprocesador *Raspberry Pi Zero W* y las plataformas en la nube, *InfluxDB Cloud* y *Grafana Cloud*. La implementación de *BreweryHub* ha demostrado ser un paso adelante en el monitoreo y análisis del proceso de fermentación. No obstante, existen varios módulos donde se pueden realizar mejoras para futuras versiones del sistema.

5.1.1.1. Interfaz de usuario y experiencia:

- Captura de datos de bitácora de proceso en la nube: La incorporación de interfaces de usuario más intuitivas, en una *WebApp* de la plataforma en la nube *Streamlit*, para el ingreso de datos manual y mejorar la usabilidad del sistema.
- Transferencia del dispositivo: Trasferir la responsabilidad de mantenimiento en operación al usuario, principalmente para la depuración de base de datos locales y el cambio de baterías, de requerirse como en Cerveza Loba.

5.1.1.2. Procesamiento de datos y análisis en tiempo real:

- Cómputo en la nube: Creación de un módulo, *WebApp* de la plataforma en la nube *Streamlit,* para el cómputo en tiempo real de los datos capturados, incluyendo transformación de estos para obtener atributos requeridos por el modelo, y la evaluación de este último para generar predicciones de avance de fermentación e *insights* en tiempo real.

- Integración entre *WebApps* y plataformas en la nube: Lograr funcionalidad entre todos los diferentes módulos, para alcanzar la interoperabilidad entre captura automática de registros por *BreweryHub*, entradas manuales de bitácora en *WebApp,* escritura en base de datos *InfluxDB Cloud,* cómputo en la nube por *WebApp* y visualización en tiempo real tanto de parámetros como de predicciones e *insights* en *Grafana Cloud*.

5.1.1.3. Base de datos y gestión de la información:

- Creación de base de datos maestra por cervecería: Es necesaria la generación de una base de datos centralizada, adicional a los *buckets* actuales en *InfluxDB Cloud* creados para cada sensor del dispositivo. Pero almacenando datos ya pre-transformados, con gran parte de los atributos mencionados en Tabla 6. Por ejemplo, con codificaciones *One-Hot Encoder* para las condiciones iniciales (G, pH, tipo de levadura) así como la media horaria de la temperatura.

  Esto para facilitar la actualización de conjuntos de datos y consulta automática durante el re-entrenamiento de modelos de *Machine Learning* para una misma cervecería.

- Escalabilidad y Generalización: El dispositivo debe extenderse al mayor número de tanques de fermentación y cervecerías, para evaluar la escalabilidad y la generalización del sistema en diferentes entornos de producción y de perfiles de fermentación.

5.1.1.4. Hardware:

- Explorar alternativas de *Hardware*: Evaluación de componentes más robustos, grado industrial, alternativas más sensibles al módulo *MAX 6675*, y mejores protecciones *IPX* contra el agua sin causar sobrecalentamiento.

- Explorar la integración de un módulo replicador de señales análogas: Posibilidad de instalar módulos replicadores en gabinetes de control que concentran transmisores de temperatura análogos usados para el sistema de control, para transformar las señales a salidas digitales que pueda leer el microprocesador. Potencialmente solo necesitar un *Raspberry Pi Zero W*, o cualquier dispositivo utilizado, para todos los tanques de fermentación de una cervecería.

5.1.1.5. Integración de parámetros y atributos de conjunto de datos adicionales:

- Explorar integración de parámetros adicionales con instrumentación en línea: Medidores ultrasónicos en línea de gravedad relativa, medidores en línea de pH, manómetros digitales o análogos.

- Inclusión parámetros microbiológicos al estudio, condiciones morfológicas cualitativas y cuantitativas de la levadura (ver Tabla 10). Dependiendo de las capacidades operativas del usuario, al requerir mínimamente de microscopio, reactivos y de conocimiento en métodos analíticos de microbiología.

5.1.1.6. Exploración y desarrollo de algoritmos de Clasificación y Forecasting:

- Desarrollo de herramientas de *Clasificación* y *Forecasting*: El desarrollo de algoritmos supervisados o no supervisados, adicionales al desarrollado para Regresión, propios para la clasificación de fases de fermentación, desviaciones estadísticas de parámetros y de pronóstico de finalización de fermentación. Para evaluar en paralelo con la herramienta avanzada ya implementada, pero esperando considerar una gama más amplia de variables podría proporcionar *insights* más profundos.

Cada una de estas áreas de mejora se debe abordar con consideración de las limitaciones actuales y las necesidades específicas del sector cervecero artesanal. La colaboración con los usuarios finales será crucial para evaluar las mejores y asegurar que sean pertinentes y valiosas para su aplicación práctica.

5.2. Relevancia y trascendencia disciplinaria del caso

El proyecto *BreweryHub* destaca por su abordaje interdisciplinario, donde se entrelazan la ingeniería de procesos y de productos, la ciencia de datos y el Aprendizaje Automático para abordar desafíos en la industria cervecera artesanal. La implementación de modelos de *Machine Learning*, como la red neuronal con su variante de arquitectura *NARX*, demuestra cómo esta técnica de análisis avanzada puede ser aplicada en contextos tradicionalmente manuales o considerados artesanales, facilitando una transición hacia prácticas más digitalizadas y automatizadas, en línea con la tendencia hacia Industria 4.0.

Este trabajo se subraya por su contribución a la aplicación de la ciencia de datos y su utilidad práctica para incrementar la consistencia y eficiencia en la producción de cerveza artesanal. Este caso de estudio aporta al conocimiento sobre la digitalización en sectores que tradicionalmente ha dependido de métodos menos tecnológicos. La investigación proporciona evidencia de cómo el *Machine Learning* puede generar información, predicciones e *insights* de anteriormente datos oscuros, no utilizados. El trabajo se alinea con las investigaciones de Alexander Bowler y Nik Watson, de la Universidad de Nottingham y Universidad Leeds respectivamente, que han colaborado en la aplicación de métodos de *Machine Learning* en la fermentación, y otros procesos de la industria alimenticia.

Los hallazgos de la primer versión de *BreweryHub*, aunque en una fase temprana, señalan el camino para futuras investigaciones y colaboraciones que podrían aplicar estos métodos en diferentes etapas de la producción cervecera y contextos industriales variados. Los resultados en Cerveza Loba, aunque preliminares, establecen un precedente para la adopción de soluciones digitales en el sector, resonando con la tendencia global hacia la digitalización y la aplicación de herramientas avanzadas de análisis de datos.

Este trabajo invita a la comunidad científica y a los profesionales de la industria a seguir explorando la integración de tecnologías emergentes con métodos convencionales, potenciando la innovación y la competitividad en industrias pequeñas de alimentos y bebidas, no solo de cervecería artesanal.

Referencias bibliográficas

1. Bhonsale, S., Mores, W., & Van Impe, J. (2021). Dynamic Optimisation of Beer Fermentation under Parametric Uncertainty. *Fermentation, MDPI*.

2. Bowler, A. L., Escrig, J., Pound, M. P., & Watson, N. J. (2021). Predicting Alcohol Concentration during Beer Fermentation Using Ultrasonic Measurements and Machine Learning XE "*Machine Learning*" . *Fermentation, MDPI*.

3. Bowler, A. L., Pound, M. P., & Watson, N. J. (2021). Domain Adaptation and Federated Learning for Ultrasonic Monitoring of Beer Fermentation. *Fermentation, MDPI*, 7(1), 34.

4. Buonocore, D., Ciavolino, G., Di Caro, D., & Liguori, C. (2021). An IoT-based beer fermentation monitoring system. *2021 IEEE International Workshop on Metrology for Agriculture and Forestry*.

5. CNNExpansión. (2013, Junio 04). Modelo ya es oficialmente de AB InBev. *De CNNExpansión:* https://expansion.mx/negocios/2013/06/04/modelo-ya-es-oficialmente-de-ab-inbev

6. De Andres-Toro, B., Giron-Sierra, J., & Lopez-Orozco, J. (1997). A Kinetic Model for Beer Production: Simulation Under Industrial Operational Conditions. *Mathematics And Computers in Simulation*.

7. El Informador. (2010, Enero 12). Heineken compra las cervezas de Cuauhtémoc Moctezuma. *De El Informador:* https://www.informador.mx/Economia/Heineken-compra-las-cervezas-de-Cuauhtemoc-Moctezuma-20100112-0267.html

8. Fernández, C., Pantano, N., Rossomando, F., & Amicarelli, A. (2020). Fermentation monitoring by Bayesian states estimators. Application to red wines elaboration. *Control Engineering Practice, Elsevier*.

9. Itto-Nakama, K., Watanabe, S., Kondo, N., & Ohnuki, S. (2021). AI-based forecasting of ethanol fermentation using yeast morphological data. *Bioscience, Biotechnology, and Biochemistry*, 2022, Vol. 86, No. 1, 125-134.

10. Karim Khaleghi, M., Pour Savizi, I. S., & Lewis, N. E. (2021). Synergisms of machine learning and constraint-based modeling of metabolism for analysis and optimization of fermentation parameters. *Biotechnology Journal*.

11. Kimutai, G., Ngenzi, A., Ngonga, S., Ramkat, R. C., & Förster, A. (2020). An internet of things (IoT)-based optimum tea fermentation detection model using convolutional neural networks (CNNs) and majority voting techniques. *Journal of Sensors and Sensor Systems*.

12. Peláez-Fernández, A. (2022, Octubre 21). Mexican craft beer to gain ground despite soaring costs, says trade group. *De Reuters:* https://www.reuters.com/business/retail-consumer/mexican-craft-beer-gain-ground-despite-soaring-costs-says-trade-group-2022-10-22/

13. Recio, G. (2004). El nacimiento de la industria cervecera en México, 1880-1910. *Segundo Congreso Nacional de Historia Económica.* Ciudad de México: Facultad de Economía de la UNAM.

14. Rincón, S. (2020, Octubre 17). Actualizar IEPS hará más competitiva cerveza artesanal: fundador Cerveza Minerva. *De Forbes México:* https://www.forbes.com.mx/negocios-actualizar-ieps-hara-mas-competitiva-cerveza-artesanal-briseno-gomez/

15. Rodríguez, G. (2022, Julio 22). Colimita: embajador insignia de la cerveza artesanal en México. *De Líderes Empresariales:* https://www.liderempresarial.com/colimita-embajador-insignia-de-la-cerveza-artesanal-en-mexico/

16. Romo, P. (2022, Agosto 29). Cerveceros artesanales van por un IEPS más competitivo. *De El Economista:* https://www.eleconomista.com.mx/estados/Cerveceros-artesanales-van-por-un-IEPS-mas-competitivo-20220828-0043.html

17. Sipos, A. (2020). A Knowledge-Based System as a Sustainable Software Application for the Supervision and Intelligent Control of an Alcoholic Fermentation Process. *Sustainability, MDPI*. (n.d.).

18. Sipos, A., Florea, A., & Arsin, M. (2020). Using Neural Networks to Obtain Indirect Information about the State Variables in an Alcoholic Fermentation Process. *Processes, MDPI*.

19. Watson, N. J., Bowler, A. L., Rady, A., Fisher, O. J., & Simeone, A. (2021). Intelligent Sensors for Sustainable Food and Drink Manufacturing. *Frontiers in Sustainable Food Systems*.

Índice de Materias

Índice de Figuras

Índice de Tablas

Índice de Siglas

Índice de Anexos

Anexo 1. *'simulation_training_v2.mlx'*, simulación del modelo cinético desarrollado por De Andrés-Toro et al. (1997) en *Matlab* con parámetros de Tabla 5, para generar *dataset* para Prueba de Concepto.

```
clear
global t0 S0 l T_v t_p
Et0 =0;        % g/L
Ac0 = 0;      % g/L (slider en ppm)
Dy0 = 0;      % g/L (slider en ppm)

pEt = 789;                          % Densidad de Etanol
tint = 0:0.02083:9.1667;
l = tint(end);
t0 = 0.0;

dataset = '/MATLAB Drive/dataset_training_v2.csv';
r = 2 ;

S0_values = {88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116};    % g/L -- OG 1043 ≈ 93.4 g/L
X0_values = {1.0};   % g/L -- 80 to 120 g per 100 L ≈ 1 g/L

% Define los perfiles de temperatura y tiempo para rampas/descansos de fermentación
T_values = {[14.5, 14.5, 22, 22, 0], ...   % Ale
        [13.5, 13.5, 22, 22, 0], ...   % Lager
        [15, 15, 22, 22, 0], ...       % Ale
        [14, 14, 22, 22, 0]};          % Lager

t_points = {[0, 150, 175, 195, 220], ...   % Ale
        [0, 150, 175, 195, 220], ...   % Lager
        [0, 150, 175, 195, 220], ...   % Ale
        [0, 150, 175, 195, 220]};      % Lager

names =     {'Training Profile A', ... % Ale
        'Training Profile B', ...  % Lager
        'Training Profile C', ...  % Ale
        'Training Profile D'};     % lager

titles =    {'training_a', ...
        'training_b', ...
        'training_c', ...
        'training_d'};

t_horas = zeros(size(tint));
for i = 1:length(tint)
   t_horas(i) = tint(i);
end
t_horas=t_horas'*24;
t_horas_data = t_horas(1:r:end); % Guarda 1 de cada r datos, de 1401x1 a ((1401-1)/r + 1)x1

Tabla_t = table(t_horas_data);
Tabla_t.Properties.VariableNames{1} = 't_horas';
writetable(Tabla_t, dataset);

% Loop para integrar en todos los perfiles de temperatura 5x5x14 = 350
for n = 1:length(S0_values)
   S0 = S0_values{n};
   table_n = table();

   for k = 1:length(X0_values)
```

```
    X0 = X0_values{k};

    C0 =[X0;S0;Et0;Ac0;Dy0;t0]; % Integración
    table_k = table();

    for j = 1:length(T_values)
        T_v = T_values{j};
        t_p = t_points{j};
        T_ = zeros(length(tint),1);
        ABV_ = zeros(length(tint),1);

        [x,C]= ode45(@beerferm,tint,C0);
        X = C(:,1); S = C(:,2); Et = C(:,3); Ac = C(:,4)/1e-3; Dy = C(:,5)/1e-3; t = C(:,6);
        for i = 1:length(tint)
            tx = tint(i) * 24;
            T_(i) = temperature(tx, T_v, t_p);
            X_(i) = X0_values{k}*100;
            S_(i) = S0_values{n};
            ABV_(i) = (((S_(i)/10)/258.6 * (227.1/(258.6-(S_(i)/10))) + 1.0) - ((S(i)/10)/258.6 * (227.1/(258.6-(S(i)/10))) + 1.0)) * 131.25 + (0.25 * rand(1) -
0.125);
        end

        figure;
        yyaxis left
        plot(t_horas, X,'g-', t_horas, S,'b-', 'MarkerSize', 1)
        ylabel('X & S Concentration (g/L)', 'FontSize', 8)
        set(gca, 'YColor', 'k');
        yyaxis right
        plot(t_horas, T_-273.15,'black-', t_horas, ABV_,'r-', 'MarkerSize', 1)
        axis([0 220 -1 25])
        set(gca, 'XColor', 'k', 'YColor', 'k');
        set(gca, 'FontSize', 7)
        ylabel('Temperature (°C), Alcohol by Volume (%)', 'FontSize', 8)
        xlabel('Time (h)', 'FontSize', 8)
        title(sprintf('Fermentation Simulation - %s \n (%0.1f g/100 L X, %0.1f g/L S)', names{j}, X0_values{k} * 100, S0_values{n}), 'FontSize', 9)
        legend('Biomass (X)', 'Substrate (S)', 'Temperature (T)', 'ABV%', 'Location', 'north', 'FontSize', 5)
        fig = gcf;
        saveas(fig, ['/MATLAB Drive/Entrenamiento/PoC_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' titles{j} '_plot.png']);

        X_data = X_';
        S_data = S_';

        T_data = T_(1:r:end);              % (1:r:end) Guarda 1 de cada 2 datos, de 1401x1 a 701x1
        ABV_data = ABV_(1:r:end);
        X_data = X_data(1:r:end);
        S_data = S_data(1:r:end);

        Tabla = table(T_data, ABV_data, X_data, S_data);

        Tabla.Properties.VariableNames{1} = ['T_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{2} = ['ABV_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{3} = ['X_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{4} = ['S_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];

        original_table = table_k;
        table_k = horzcat(original_table, Tabla);
    end
    original_table = table_n;
```

```
    table_n = horzcat(original_table, table_k);
  end
  original_table = readtable(dataset);
  merged_table = horzcat(original_table, table_n);
  writetable(merged_table, dataset);
end

function dC = beerferm(x,C)
  X = C(1); S = C(2); Et = C(3); Ac = C(4); Dy = C(5); t = C(6);
  global S0 T_v t_p
  tx = t*24;
  T = temperature(tx, T_v, t_p);
  % constantes función de la temperatura
  mu_x0 = exp(108.31 - 31934.09/T);
  mu_lag = exp(30.72 - 9501.54/T);
  mu_DT = exp(130.16 - 38313/T);
  Yeas = exp(89.92 - 26589/T);
  mu_s0 = exp(-41.92 + 11654.64/T);
  mu_dy = -6.1344e-8 * T^2 + 8.4266e-6 * T - 1.7672e-2;
  mu_ab = -9.1384e-7 * T^2 + 6.7071e-5 * T - 0.1251e-3;
  mu_e0 = exp(3.27 - 1267.24/T);
  ke = exp(-119.63 + 34203.95/T);
  kx = ke; ks = ke;
  % constantes cinéticas
  mu_x = mu_x0 * S/(kx + Et);
  mu_s = mu_s0 * S/(ks + S);
  mu_e = mu_e0 * S/(ke + S);
  f = 1 - Et/(0.5 * S0);
  % ecuaciones diferenciales
  dX = mu_x * X - mu_DT * X;
  dS = -mu_s * X;
  dEt = f * mu_e * X;
  dAc = -Yeas * dS;
  dDy = mu_dy * S * X - mu_ab * Dy * Et;
  dt = 1;
  if Dy <= 0
    dDy = 0;
  end
  % salidas de la función
  dC =[dX;dS;dEt;dAc;dDy;dt];
end

function T = temperature(tx, T_value, t_point)
  global l
  % Interpolate between the given time points
  if tx >= t_point(end)
    T = T_value(end) + 273.15 + sin_wave(tx) + uniform_noise(tx, 0, l*24);
  else
    T = interp1(t_point, T_value, tx, 'linear') + 273.15 + sin_wave(tx) + uniform_noise(tx, 0, l*24);
  end
end

function y = sin_wave(tx)
  A = 0.2;          % amplitude
  h = 24;          % period
  w = 2*pi/h;       % angular frequency
  y = A*sin(w*tx);    % sinusoidal wave equation
end
```

```
function y = uniform_noise(tx, a, b)
  if tx < a || tx > b
    y = 0;           % return 0 if tx is outside of range [a, b]
  else
    y = rand(1) - 0.2;  % generate random value between -0.3 and 0.3
  end
end
```

Anexo 2. *'simulation_evaluation_v2.mlx'*, simulación del modelo cinético desarrollado por De Andrés-Toro et al. (1997) en *Matlab* con parámetros de Tabla 8, para generar *'dataset_ev'* para Prueba de Concepto.

```
clear
global t0 S0 l T_v t_p
Et0 =0;        % g/L
Ac0 = 0;      % g/L (slider en ppm)
Dy0 = 0;      % g/L (slider en ppm)

pEt = 789;                        % Densidad de Etanol
tint = 0:0.02083:9.1667;
l = tint(end);
t0 = 0.0;

dataset = '/MATLAB Drive/dataset_evaluation_v2.csv';
r = 2 ;

S0_values = {96, 100, 104, 108};    % g/L -- OG 1043 ≈ 93.4 g/L
X0_values = {1.0};   % g/L -- 80 to 120 g per 100 L ≈ 1 g/L

% Define los perfiles de temperatura y tiempo para rampas/descansos de fermentación
T_values = {[14.34, 14.86, 21.6, 21.25, 0], ...   % 1 Lager
        [14.25, 13.17, 22.22, 22.03, 0], ...     % 2 Andrés-Toro et al. (1997)
        [15.19, 15.54, 21.52, 22.94, 0], ...   % 3 Ale
        [13.53, 13.97, 21.7, 21.51, 0]};      % 7 Requiem

t_points = {[0.0, 147.13, 172.94, 198.78, 223.47], ...   % Lager
        [0.0, 154.5, 179.19, 196.26, 220.95], ...   % Andrés-Toro et al. (1997)
        [0.0, 151.46, 175.18, 197.07, 224.26], ...   % Ale
        [0.0, 150.28, 178.76, 194.64, 216.3]};     % Requiem

names =    {'Training A Profile', ... % Lager
        'Training B Profile', ...  % Andrés-Toro et al. (1997)
        'Training C Profile', ...  % Ale
        'Training D Profile'};    % Requiem

titles =   {'training_a', ...
        'training_b', ...
        'training_c', ...
        'training_d'};

t_horas = zeros(size(tint));
for i = 1:length(tint)
   t_horas(i) = tint(i);
end
t_horas=t_horas'*24;
```

```
t_horas_data = t_horas(1:r:end); % Guarda 1 de cada r datos, de 1401x1 a ((1401-1)/r + 1)x1

Tabla_t = table(t_horas_data);
Tabla_t.Properties.VariableNames{1} = 't_horas';
writetable(Tabla_t, dataset);

% Loop para integrar en todos los perfiles de temperatura 5x5x14 = 350
for n = 1:length(S0_values)
   S0 = S0_values{n};
   table_n = table();

   for k = 1:length(X0_values)
      X0 = X0_values{k};

      C0 =[X0;S0;Et0;Ac0;Dy0;t0]; % Integración
      table_k = table();

      for j = 1:length(T_values)
         T_v = T_values{j};
         t_p = t_points{j};
         T_ = zeros(length(tint),1);
         ABV_ = zeros(length(tint),1);

         [x,C]= ode45(@beerferm,tint,C0);
         X = C(:,1); S = C(:,2); Et = C(:,3); Ac = C(:,4)/1e-3; Dy = C(:,5)/1e-3; t = C(:,6);
         for i = 1:length(tint)
            tx = tint(i) * 24;
            T_(i) = temperature(tx, T_v, t_p);
            X_(i) = X0_values{k}*100;
            S_(i) = S0_values{n};
            ABV_(i) = (((S_(i)/10)/258.6 * (227.1/(258.6-(S_(i)/10))) + 1.0) - ((S(i)/10)/258.6 * (227.1/(258.6-(S(i)/10))) + 1.0)) * 131.25 + (0.25 * rand(1) -
0.125);
         end

         figure;
         yyaxis left
         plot(t_horas, X,'g-', t_horas, S,'b-', 'MarkerSize', 1)
         ylabel('X & S Concentration (g/L)', 'FontSize', 8)
         set(gca, 'YColor', 'k');
         yyaxis right
         plot(t_horas, T_-273.15,'black-', t_horas, ABV_,'r-', 'MarkerSize', 1)
         axis([0 220 -1 25])
         set(gca, 'XColor', 'k', 'YColor', 'k');
         set(gca, 'FontSize', 7)
         ylabel('Temperature (°C), Alcohol by Volume (%)', 'FontSize', 8)
         xlabel('Time (h)', 'FontSize', 8)
         title(sprintf('Fermentation Simulation - %s \n (%0.1f g/100 L X, %0.1f g/L S)', names{j}, X0_values{k} * 100, S0_values{n}), 'FontSize', 9)
         legend('Biomass (X)', 'Substrate (S)', 'Temperature (T)', 'ABV%', 'Location', 'north', 'FontSize', 5)
         fig = gcf;
         saveas(fig, ['/MATLAB Drive/Evaluación/PoC_Ev_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' titles{j} '_plot.png']);

         X_data = X_';
         S_data = S_';

         T_data = T_(1:r:end);              % (1:r:end) Guarda 1 de cada 2 datos, de 1401x1 a 701x1
         ABV_data = ABV_(1:r:end);
         X_data = X_data(1:r:end);
         S_data = S_data(1:r:end);
```

```
        Tabla = table(T_data, ABV_data, X_data, S_data);

        Tabla.Properties.VariableNames{1} = ['T_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{2} = ['ABV_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{3} = ['X_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{4} = ['S_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];

        original_table = table_k;
        table_k = horzcat(original_table, Tabla);
      end
      original_table = table_n;
      table_n = horzcat(original_table, table_k);
    end
    original_table = readtable(dataset);
    merged_table = horzcat(original_table, table_n);
    writetable(merged_table, dataset);
end

function dC = beerferm(x,C)
    X = C(1); S = C(2); Et = C(3); Ac = C(4); Dy = C(5); t = C(6);
    global S0 T_v t_p
    tx = t*24;
    T = temperature(tx, T_v, t_p);
    % constantes función de la temperatura
    mu_x0 = exp(108.31 - 31934.09/T);
    mu_lag = exp(30.72 - 9501.54/T);
    mu_DT = exp(130.16 - 38313/T);
    Yeas = exp(89.92 - 26589/T);
    mu_s0 = exp(-41.92 + 11654.64/T);
    mu_dy = -6.1344e-8 * T^2 + 8.4266e-6 * T - 1.7672e-2;
    mu_ab = -9.1384e-7 * T^2 + 6.7071e-5 * T - 0.1251e-3;
    mu_e0 = exp(3.27 - 1267.24/T);
    ke = exp(-119.63 + 34203.95/T);
    kx = ke; ks = ke;
    % constantes cinéticas
    mu_x = mu_x0 * S/(kx + Et);
    mu_s = mu_s0 * S/(ks + S);
    mu_e = mu_e0 * S/(ke + S);
    f = 1 - Et/(0.5 * S0);
    % ecuaciones diferenciales
    dX = mu_x * X - mu_DT * X;
    dS = -mu_s * X;
    dEt = f * mu_e * X;
    dAc = -Yeas * dS;
    dDy = mu_dy * S * X - mu_ab * Dy * Et;
    dt = 1;
    if Dy <= 0
        dDy = 0;
    end
    % salidas de la función
    dC =[dX;dS;dEt;dAc;dDy;dt];
end

function T = temperature(tx, T_value, t_point)
    global I
    % Interpolate between the given time points
    if tx >= t_point(end)
```

```
    T = T_value(end) + 273.15 + sin_wave(tx) + uniform_noise(tx, 0, l*24);
  else
    T = interp1(t_point, T_value, tx, 'linear') + 273.15 + sin_wave(tx) + uniform_noise(tx, 0, l*24);
  end
end

function y = sin_wave(tx)
  A = 0.2;          % amplitude
  h = 24;           % period
  w = 2*pi/h;       % angular frequency
  y = A*sin(w*tx);    % sinusoidal wave equation
end

function y = uniform_noise(tx, a, b)
  if tx < a || tx > b
    y = 0;          % return 0 if tx is outside of range [a, b]
  else
    y = rand(1) - 0.2;  % generate random value between -0.3 and 0.3
  end
end
```

Anexo 3. *'simulation_training_v3.mlx'*, simulación del modelo cinético desarrollado por De Andrés-Toro et al. (1997) en *Matlab* con parámetros de Tabla 12 y Figura 31, para generar *'dataset_brew'*.

```
clear
global t0 S0 l T_v t_p
Et0 =0;       % g/L
Ac0 = 0;      % g/L (slider en ppm)
Dy0 = 0;      % g/L (slider en ppm)

pEt = 789;                      % Densidad de Etanol
tint = 0:0.02083:8.125;
l = tint(end);
t0 = 0.0;

dataset = '/MATLAB Drive/dataset_training_v33.1.csv';
r = 2;

S0_values = {88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112};    % g/L -- OG 1043 ≈ 93.4 g/L
X0_values = {1.0};     % g/L -- 80 to 120 g per 100 L ≈ 1 g/L

% Define los perfiles de temperatura y tiempo para rampas/descansos de fermentación
T_values = {[14, 19, 20.5, 20.5], ...        % Evaluation A
        [14, 14, 19.5, 14, 14, 19.5, 19.5], ...
        [14.5, 19.5, 20.5, 20.5], ...
        [14.5, 14.5, 19.5, 14.5, 14.5, 19.5, 19.5], ...
        [15, 20, 21, 21],...
        [13.5, 13.5, 19., 13.5, 13.5, 19, 19]};                % Evaluation F

t_points = {[0, 150, 175, 195], ...          % Evaluation A
        [0, 25, 35, 50, 150, 175, 195], ...
        [0, 150, 175, 195], ...
        [0, 25, 35, 50, 150, 175, 195], ...
        [0, 150, 175, 195],...
        [0, 25, 35, 50, 150, 175, 195]};               % Evaluation F
```

```
names =    {'Evaluation A Profile', ...
        'Evaluation B Profile', ...
        'Evaluation C Profile', ...
        'Evaluation D Profile', ...
        'Evaluation E Profile',...
        'Evaluation F Profile'};

titles =   {'evaluation_a', ...                 % Evaluation A
        'evaluation_b', ...                  % Evaluation B
        'evaluation_c', ...
        'evaluation_d', ...
        'evaluation_e',...
        'evaluation_f'};                     % Evaluation h

t_horas = zeros(size(tint));
for i = 1:length(tint)
   t_horas(i) = tint(i);
end
t_horas=t_horas'*24;
t_horas_data = t_horas(1:r:end); % Guarda 1 de cada r datos, de 1401x1 a ((1401-1)/r + 1)x1

Tabla_t = table(t_horas_data);
Tabla_t.Properties.VariableNames{1} = 't_horas';
writetable(Tabla_t, dataset);

% Loop para integrar en todos los perfiles de temperatura 5x5x14 = 350
for n = 1:length(S0_values)
   S0 = S0_values{n};
   table_n = table();

   for k = 1:length(X0_values)
      X0 = X0_values{k};

      C0 =[X0;S0;Et0;Ac0;Dy0;t0]; % Integración
      table_k = table();


      for j = 1:length(T_values)
         T_v = T_values{j};
         t_p = t_points{j};
         T_ = zeros(length(tint),1);
         ABV_ = zeros(length(tint),1);

         [x,C]= ode45(@beerferm,tint,C0);
         X = C(:,1); S = C(:,2); Et = C(:,3); Ac = C(:,4)/1e-3; Dy = C(:,5)/1e-3; t = C(:,6);
         for i = 1:length(tint)
            tx = tint(i) * 24;
            T_(i) = temperature(tx, T_v, t_p);
            X_(i) = X0_values{k}*100;
            S_(i) = S0_values{n};
            ABV_(i) = (((S_(i)/10)/258.6 * (227.1/(258.6-(S_(i)/10))) + 1.0) - ((S(i)/10)/258.6 * (227.1/(258.6-(S(i)/10))) + 1.0)) * 131.25 + (0.25 * rand(1) -
0.125);
         end

         figure;
         yyaxis left
         plot(t_horas, X,'g-', t_horas, S,'b-', 'MarkerSize', 1)
         ylabel('X & S Concentration (g/L)', 'FontSize', 8)
```

```
        set(gca, 'YColor', 'k');
        yyaxis right
        plot(t_horas, T_-273.15,'black-', t_horas, ABV_,'r-', 'MarkerSize', 1)
        axis([0 200 -1 25])
        set(gca, 'XColor', 'k', 'YColor', 'k');
        set(gca, 'FontSize', 7)
        ylabel('Temperature (°C), Alcohol by Volume (%)', 'FontSize', 8)
        xlabel('Time (hours)', 'FontSize', 8)
        title(sprintf('Fermentation Simulation - %s \n (%0.1f g/100 L X, %0.1f g/L S)', names{j}, X0_values{k} * 100, S0_values{n}), 'FontSize', 9)
        legend('Biomass (X)', 'Substrate (S)', 'Temperature (T)', 'ABV%', 'Location', 'best', 'FontSize', 5)
        fig = gcf;
        saveas(fig, ['/MATLAB Drive/Entrenamiento/' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' titles{j} '_plot.png']);

        X_data = X_';
        S_data = S_';

        T_data = T_(1:r:end);              % (1:r:end) Guarda 1 de cada 2 datos, de 1401x1 a 701x1
        ABV_data = ABV_(1:r:end);
        X_data = X_data(1:r:end);
        S_data = S_data(1:r:end);

        Tabla = table(T_data, ABV_data, X_data, S_data);

        Tabla.Properties.VariableNames{1} = ['T_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{2} = ['ABV_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{3} = ['X_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];
        Tabla.Properties.VariableNames{4} = ['S_' num2str(S0_values{n}) '_' num2str(X0_values{k} * 100) '_' num2str(j)];

        original_table = table_k;
        table_k = horzcat(original_table, Tabla);
      end
      original_table = table_n;
      table_n = horzcat(original_table, table_k);
    end
    original_table = readtable(dataset);
    merged_table = horzcat(original_table, table_n);
    writetable(merged_table, dataset);
end

function dC = beerferm(x,C)
    X = C(1); S = C(2); Et = C(3); Ac = C(4); Dy = C(5); t = C(6);
    global S0 T_v t_p
    tx = t*24;
    T = temperature(tx, T_v, t_p);
    % constantes función de la temperatura
    mu_x0 = exp(108.31 - 31934.09/T);
    mu_lag = exp(30.72 - 9501.54/T);
    mu_DT = exp(130.16 - 38313/T);
    Yeas = exp(89.92 - 26589/T);
    mu_s0 = exp(-41.92 + 11654.64/T);
    mu_dy = -6.1344e-8 * T^2 + 8.4266e-6 * T - 1.7672e-2;
    mu_ab = -9.1384e-7 * T^2 + 6.7071e-5 * T - 0.1251e-3;
    mu_e0 = exp(3.27 - 1267.24/T);
    ke = exp(-119.63 + 34203.95/T);
    kx = ke; ks = ke;
    % constantes cinéticas
    mu_x = mu_x0 * S/(kx + Et);
    mu_s = mu_s0 * S/(ks + S);
```

```
    mu_e = mu_e0 * S/(ke + S);
    f = 1 - Et/(0.5 * S0);
    % ecuaciones diferenciales
    dX = mu_x * X - mu_DT * X;
    dS = -mu_s * X;
    dEt = f * mu_e * X;
    dAc = -Yeas * dS;
    dDy = mu_dy * S * X - mu_ab * Dy * Et;
    dt = 1;
    if Dy <= 0
        dDy = 0;
    end
    % salidas de la función
    dC =[dX;dS;dEt;dAc;dDy;dt];
end

function T = temperature(tx, T_value, t_point)
    global l
    % Interpolate between the given time points
    if tx >= t_point(end)
        T = T_value(end) + 273.15 + sin_wave(tx) + uniform_noise(tx, 0, l*24);
    else
        T = interp1(t_point, T_value, tx, 'linear') + 273.15 + sin_wave(tx) + uniform_noise(tx, 0, l*24);
    end
end

function y = sin_wave(tx)
    A = 0.2;            % amplitude
    h = 24;           % period
    w = 2*pi/h;       % angular frequency
    y = A*sin(w*tx);    % sinusoidal wave equation
end

function y = uniform_noise(tx, a, b)
    if tx < a || tx > b
        y = 0;            % return 0 if tx is outside of range [a, b]
    else
        y = rand(1) - 0.2;  % generate random value between -0.3 and 0.3
    end
end
```

Anexo 4. *'PoC_Training_DataAnalysis_v1.0.ipynb'*, sección de pre-procesamiento de datos para generación de *'dataset_training_PoC'*.

```
import pandas as pd; from pandas import read_csv
import numpy as np; from numpy import loadtxt
import matplotlib.pyplot as plt; import math
from matplotlib.legend import Legend
from sklearn.preprocessing import OneHotEncoder
import seaborn as sns
from google.colab import drive

from datetime import datetime
date_string = datetime.now().strftime("%Y%m%d")

# Funciones
def sigmoid(x):
  return 1 / (1 + np.exp(-10*(x - 0.5)))

def scaled_sigmoid(x):
  return (sigmoid(x) - sigmoid(0)) / (sigmoid(1) - sigmoid(0))

def k_to_c(x):
  return x['T'] - 273.15

def normalize_time(x):
  y = (x['t_horas'] - x['t_horas'].min()) / (x['t_horas'].max() - x['t_horas'].min())
  return y

def gain_function(x):
  return (x['T']) **2.5 * (1 - scaled_sigmoid(x['Timepoint'])) / 55 **2.5

def reorder_columns(x):
  new_order = ['T', 't_horas', 'Timepoint', 'Yeast_type_0', 'Yeast_type_1', 'OG_31', 'OG_32', 'OG_33', 'OG_34', 'OG_35', 'OG_36', 'OG_37','OG_38',
'OG_39', 'OG_40', 'OG_41', 'pH0_5.0', 'pH0_5.5', 'Gainsum', 'ABV']
  return x.reindex(columns=new_order)

def eval_graph(Z, text, z):
  plt.rcParams["figure.figsize"] = [7.50, 5.0]
  x, y1, y2 = Z['t_horas'], Z['T'], Z['ABV']
  fig, ax1 = plt.subplots()
  ax1.plot(x, y1, 'b.', label='Temperature', alpha=0.2)
  ax1.set_xlabel('Time (h)')
  ax1.set_ylim([0, 25])
  ax1.set_ylabel('Temperature (°C)')
  ax1.set_title(text + z + "\nSimulated in MATLAB")
  ax2 = ax1.twinx()
  ax2.plot(x, y2, 'r.', label='ABV%', alpha=0.25)
  ax2.set_xlim([0, Z['t_horas'].max() * 1.03])
  ax2.set_ylim([0, round(Z['ABV'].max())])
  ax2.set_ylabel('Alcohol by Volume (%)')

  lines1, labels1 = ax1.get_legend_handles_labels()
  lines2, labels2 = ax2.get_legend_handles_labels()
  lines = lines1 + lines2
  labels = labels1 + labels2
  ax1.legend(lines, labels, loc='upper left')

  plt.tight_layout()
```

```
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_fermentation_profile.png', dpi = 1000)
    plt.show()
    x, y1, y2 = Z['t_horas'], Z['Gainsum'], Z['ABV']
    fig, ax1 = plt.subplots()
    ax1.plot(x, y1, 'b.', label='Gainsum', alpha=0.2)
    ax1.set_xlabel('Time (h)')
    ax1.set_ylim([0, Z['Gainsum'].max() * 1.08])
    ax1.set_ylabel('Gainsum(t)')
    ax1.set_title(text + z + "\nTransformed")
    ax2 = ax1.twinx()
    ax2.plot(x, y2, 'r.', label='ABV%', alpha=0.25)
    ax2.set_xlim([0, Z['t_horas'].max() * 1.03])
    ax2.set_ylim([0, round(Z['ABV'].max())])
    ax2.set_ylabel('Alcohol by Volume (%)')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')

    plt.tight_layout()
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_fermentation_transformed.png', dpi = 1000)
    plt.show()

    mat_corr_1 = sns.heatmap(Z[['Gainsum', 'ABV']].corr(), annot = True, cmap = 'seismic', fmt='.3f')
    plt.title(text + z + "\nCorrelation Matrix")
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_corr_ABV.png', dpi = 1000)
    plt.show()

    pair_plot = sns.pairplot(Z[['Gainsum', 'ABV']], diag_kind='kde', plot_kws={'alpha': 0.6})
    plt.subplots_adjust(top=0.9)
    plt.suptitle(text + z + "\nPairplot", fontsize=12)
    plt.savefig(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_pairplot_ABV.png', dpi = 1000)
    plt.show()

def preprocess_data(dt, y_values, x_values, VAR_values, rango):
    n_values = list(rango)
    combinations = [(x, y, n) for x in x_values for y in y_values for n in n_values]
    data_dict = {}
    for x, y, n in combinations:
        cols = [f'{VAR}_{y}_{x}_{n}' for VAR in VAR_values]
        datacol = dt[cols]
        datacol = datacol.rename(columns=dict(zip(cols, VAR_values)))
        data = dt['t_horas']
        data = pd.concat([data, datacol], axis=1)
        data_dict[f'dt_{y}_{x}_{n}'] = data
    data_trans = pd.DataFrame()
    for x, y, n in combinations:
        data_key = f"dt_{y}_{x}_{n}"
        if data_key not in data_dict:
            continue
        dt = data_dict[data_key]
        dt = dt[dt['t_horas'] <= 195]
        dt['Timepoint'] = normalize_time(dt)
        dt['OG'] = (dt['S']/10)/258.6 * (227.1/(258.6-(dt['S']/10))) * (1000/1.015) # Diferencia, Conversión S de g/L a gravedad relativa con corrección 1/1.015
vs simulación
        dt.drop(columns=['S'], inplace=True)
```

```
        dt['T'] = k_to_c(dt) # +1.0 corrección
        dt['Yeast_type'] = np.where(n%2 == 0, 2, 1)  # Tipo de levadura pendiente (1 Ale / 2 Lager / # Híbrida)
        dt['pH0'] = np.where(dt['Yeast_type'] == 2, 4.7, 5.2)
        dt['T'] = np.where(dt['Yeast_type'] == 2, dt['T']-1.25, dt['T']+1.25)
        dt['Gain'] = gain_function(dt)
        dt['Gainsum'] = dt['Gain'].cumsum()
        dt.drop(columns=['Gain'], inplace=True)
        dt = pd.DataFrame(dt)
        dt = dt.reset_index(drop=True)
        data_trans = pd.concat([data_trans, dt], axis=0, ignore_index=True)

    data_trans.drop(columns=['X'], inplace=True)
    data_trans = encode_and_bind(data_trans, 'Yeast_type')
    data_trans = encode_and_bin_ranges(data_trans, 'OG', gravity_bins, gravity_labels)
    data_trans = encode_and_bin_ranges(data_trans, 'pH0', pH_bins, pH_labels)
    data_trans = reorder_columns(data_trans)

    rows_to_drop = int(len(data_trans) * 0.7)
    indices_to_drop = data_trans.sample(n=rows_to_drop).index
    data_trans = data_trans.drop(indices_to_drop)

    data_trans.to_csv("/content/drive/MyDrive//Colab Notebooks/PoC/dataset_training_v2_transformed.csv")
    return data_trans

def encode_and_bind(original_dataframe, column_to_encode):
    encoder = OneHotEncoder(sparse_output=False)
    encoded_columns = encoder.fit_transform(original_dataframe[[column_to_encode]])
    encoded_df = pd.DataFrame(encoded_columns,
                              index=original_dataframe.index,
                              columns=[f'{column_to_encode}_{i}' for i in range(encoded_columns.shape[1])])
    without_column = original_dataframe.drop(column_to_encode, axis=1)
    encoded_dataframe = pd.concat([without_column, encoded_df], axis=1)
    return encoded_dataframe

def encode_and_bin_ranges(df, column_to_encode, bins, labels):
    df['bin'] = pd.cut(df[column_to_encode], bins=bins, labels=labels, right=False)
    one_hot_encoded_df = pd.get_dummies(df['bin'], prefix=column_to_encode)
    df = df.drop([column_to_encode, 'bin'], axis=1)
    df = pd.concat([df, one_hot_encoded_df], axis=1)
    return df

########## Datasets Generados en Matlab
drive.mount('/content/drive', force_remount=True)
dt = pd.read_csv("/content/drive/MyDrive//Colab Notebooks/PoC/dataset_training_v2.csv", delimiter = ',')                    # 320 - 24 horas de
estabilización de ABV%
gravity_bins = [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]
gravity_labels = ['31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41']
pH_bins = [4.0, 5.0, 5.5]
pH_labels = ['5.0', '5.5']

VAR_values = ['T', 'ABV', 'X', 'S']       # Nombres de columnas
y_values = [88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116]        # g/L
x_values = [100]        # g/L -- 80 to 120 g per 100 L ≈ 1 g/L
rango = range(1, 5)                  # número de perfiles simulados dentro de dataset original = 4
data_trans  = preprocess_data(dt, y_values, x_values, VAR_values, rango)
eval_graph(data_trans, "Fermentation Profiles - ", "Proof of Concept")

X = data_trans.iloc[:, 1:-1]
```

```
y = data_trans.iloc[:, [-1]]  # Variable objetivo

print("El tamaño del arreglo de entrada (X) es:", X.shape)
print("El tamaño del arreglo de salida (y) es:", y.shape)
colm_X = X.shape[1]
```

Anexo 5. *'PoC_Training_DataAnalysis_v1.0.ipynb'*, sección de primera optimización por búsqueda exhaustiva (*grid*) para una arquitectura de modelo de red neuronal *NARX*, con *'dataset_training_PoC'*.

```
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from keras.layers import Dense, Dropout
from keras import regularizers
from keras.optimizers import Adam
from tabulate import tabulate
import os
import pickle
from datetime import datetime
date_string = datetime.now().strftime("%Y%m%d")

firstlayer_act = ['hard_sigmoid','sigmoid', 'tanh', 'relu', 'elu'] # 'sigmoid', 'tanh', 'relu', 'selu', 'elu', 'gelu', 'linear', 'hard_sigmoid'
hiddenlayer_act = ['hard_sigmoid', 'sigmoid', 'tanh'] # 'hard_sigmoid', 'sigmoid', 'linear',
outputlayer_act = ['linear']
regularizer = [None, regularizers.l1(0.0005), regularizers.l2(0.0005)] # none, regularizers.l2(0.0005)
dropouts = [0] # 0.1
losses = ['mse'] # 'mae'
optimizers = ['adam'] # 'RMSprop', 'SGD', 'Adagrad'
epochs = [100, 150]; batchs = [100, 150] # 150, 225, 300
early_stopping = EarlyStopping(monitor = 'mse', patience = 3)
n_lags = [0, 1] # 1
fl_values = [colm_X * 5] # Columns x
hl_values = [round(fl_values[0] / 2)] # fl/2
headers = ['n','Lagged','Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', '1st Layer Activation', 'Hidden Layer Activation', 'Output
Layer Activation', 'First Layer Neurons', 'Hidden Layer Neurons', 'MSE*MAE', 'MSE']

total_iterations = len(n_lags) * len(batchs) * len(epochs) * len(optimizers) * len(losses) * len(dropouts) * len(regularizer) * len(firstlayer_act) *
len(hiddenlayer_act) * len(outputlayer_act) * len(fl_values) * len(hl_values)
n_1st = 0
best_params_1st = []

for lag in n_lags:
    y_lagged = y.shift(periods=lag, fill_value=0)
    X_narx = pd.concat([X, y_lagged], axis=1).iloc[lag:]
    x_train, x_test, y_train, y_test = train_test_split(X_narx.iloc[:,:-1], X_narx.iloc[:,-1], test_size=0.10, random_state=0)
    for batch in batchs:
        for epoch in epochs:
            for opt in optimizers:
                for l in losses:
                    for dropout in dropouts:
                        for reg in regularizer:
                            for flact in firstlayer_act:
                                for hlact in hiddenlayer_act:
                                    for olact in outputlayer_act:
                                        for fl in fl_values:
```

```
                                for hl in hl_values:
                                    model = Sequential()
                                    model.add(Dense(fl, input_dim=X_narx.shape[1]-1, activation=flact, kernel_regularizer=reg))
                                    model.add(Dropout(dropout))
                                    model.add(Dense(hl, activation=hlact, kernel_regularizer=reg))
                                    model.add(Dense(1, activation=olact))
                                    model.compile(loss=l, optimizer=opt, metrics=['mse', 'mae'])
                                    model.fit(x_train, y_train, epochs=epoch, batch_size=batch, verbose=0, callbacks=[early_stopping])
                                    _, mse, mae = model.evaluate(X, y, verbose=0)
                                    n_1st += 1
                                    progress_percent = (n_1st / total_iterations) * 100
                                    best_params_1st.append((lag, batch, epoch, opt, l, dropout, reg, flact, hlact, olact, fl, hl, mse*mae, mse))
                                    results = [(n_1st, lag, batch, epoch, opt, l, dropout, str(type(reg)).split(".")[-1][:-2], flact, hlact, olact, fl, hl, mse, mae) for n_1st,
(lag, batch, epoch, opt, l, dropout, reg, flact, hlact, olact, fl, hl, mse, mae) in enumerate(best_params_1st, 1)]
                                    print(f'\rProgress: {n_1st}/{total_iterations} ({progress_percent:.2f}%)', end='', flush=True)

print(" Finished!")
print(tabulate(results, headers=headers))
best_params_pivot = pd.DataFrame(best_params_1st, columns=['Lagged','Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', '1st
Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'First Layer Neurons', 'Hidden Layer Neurons', 'MSE*MAE', 'MSE'])
best_params_pivot['Regularizer'] = best_params_pivot['Regularizer'].apply(lambda x: x.__class__.__name__)
best_params_pivot['Regularizer'] = np.where(best_params_pivot['Regularizer'] == "NoneType", "None", best_params_pivot['Regularizer'])
best_params_sorted_1st = sorted(best_params_1st, key=lambda x: x[-1], reverse=False)   # Ordenar lista de parámetros basado a mínimo MSE
best_param_1st = best_params_sorted_1st[0]
best_param_sorted_1st_df = pd.DataFrame(best_params_sorted_1st, columns=['Lagged','Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout',
'Regularizer', '1st Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'First Layer Neurons', 'Hidden Layer Neurons', 'MSE*MAE',
'MSE'])
best_param_sorted_1st_df['Regularizer'] = best_param_sorted_1st_df['Regularizer'].apply(lambda x: x.__class__.__name__)
best_param_sorted_1st_df.to_csv("/content/drive/MyDrive//Colab Notebooks/PoC/PoC_best_param_sorted_1st.csv")
print("Best combination of parameters:")
best_combination = best_param_sorted_1st_df.iloc[0, :-2]
print(best_combination.to_string())
print("MSE:", best_param_1st[-1])

pivot = best_params_pivot.pivot_table(
  index=['Batch Size', 'Epochs', '1st Layer Activation', 'Hidden Layer Activation', 'Optimizer'],
  columns= ('Lagged', 'First Layer Neurons', 'Hidden Layer Neurons','Regularizer'),
  values='MSE',
  aggfunc='mean'
)
fig = plt.figure(figsize=(1.8 * len(n_lags) * len(fl_values) * len(hl_values) * len(regularizer), 0.5 * len(batchs) * len(epochs) * len(optimizers *
len(firstlayer_act) * len(hiddenlayer_act))))
sns.heatmap(pivot, annot=True, fmt=".4f", cmap="YlGnBu")
plt.title('Proof of Concept\nAverage MSE Score by Hyperparameter ConFiguration')
plt.show()
fig.savefig(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_pivot_MSEheatmap_1st.png', dpi=1000)

with open(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_best_param_1st.pkl', 'wb') as f:
  pickle.dump(best_param_1st, f)
```

Anexo 6. *'PoC_Training_DataAnalysis_v1.0.ipynb'*, sección de segunda optimización por búsqueda exhaustiva (*grid*) para una arquitectura de modelo de red neuronal *NARX*, con *'dataset_training_PoC'*.

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Dense, Dropout
from keras import regularizers
from keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tabulate import tabulate
import os
import pickle
from datetime import datetime

date_string = datetime.now().strftime("%Y%m%d")

with open(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_best_param_1st.pkl', 'rb') as f:
  best_param = pickle.load(f)

fl_values = [colm_X * 4, colm_X * 5, colm_X * 6] # Columns x n
hl_values = [(2 * colm_X)-2, round(fl_values[0] / 2), round(fl_values[1] / 2), round(fl_values[2] / 2)] # ROUND(fl/2)

total_iterations = len(fl_values) * len(hl_values)
early_stopping = EarlyStopping(monitor = 'mse', patience = 5)
headers_2nd = ['First Layer Neurons', 'Hidden Layer Neurons', 'Lagged', 'Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', 'First
Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'MSE']
best_mse = float('inf')
best_model = None
n_2nd = 0
best_params_2nd = []

for fl in fl_values:
  for hl in hl_values:
    y_lagged = y.shift(periods=best_param[0], fill_value=0)                  # Variable target
    X_narx = pd.concat([X, y_lagged], axis=1).iloc[best_param[0]:]              # Combina input original + lagged
    x_train, x_test, y_train, y_test = train_test_split(X_narx.iloc[:,:-1], X_narx.iloc[:,-1], test_size=0.30, random_state=0)
    model = Sequential()
    model.add(Dense(fl, input_dim=X_narx.shape[1]-1, activation=best_param[7], kernel_regularizer=best_param[6])) # best_param[6]
    model.add(Dropout(best_param[5]))
    model.add(Dense(hl, activation=best_param[8], kernel_regularizer=best_param[6]))
    model.add(Dense(1, activation=best_param[9]))
    model.compile(loss=best_param[4], optimizer=best_param[3], metrics=['mse'])         # Compilación modelo
    model.fit(X, y, epochs=best_param[2], batch_size=best_param[1], verbose=0, callbacks=[early_stopping])               # Ajuste modelo a dataset
    _, mse = model.evaluate(X, y, verbose=0)
    if mse < best_mse:  # Si el mse actual es mejor que el mejor mse registrado
      best_mse = mse  # Actualiza el mejor mse
      best_model = model                                  # Evaluación del modelo
    n_2nd += 1
    progress_percent = (n_2nd / total_iterations) * 100
    best_param_tuple_2nd = (best_param[0], best_param[1], best_param[2], best_param[3], best_param[4], best_param[5], best_param[6],
best_param[7], best_param[8], best_param[9])
    best_params_2nd.append((fl, hl, *best_param_tuple_2nd, mse)) # Almacena parámetros utilizados
    results_2nd = [(n_2nd, fl, hl, *best_param_tuple_2nd,  mse) for n_2nd, (fl, hl, *best_param_tuple_2nd, mse) in enumerate(best_params_2nd, 1)]
    print(f'\rProgress: {n_2nd}/{total_iterations} ({progress_percent:.2f}%)', end='', flush=True)

best_model.save(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_NARX_2nd.pb')
#print(tabulate(results_2nd, headers=headers_2nd));
```

```
best_params_sorted_2nd = sorted(best_params_2nd, key=lambda x: x[-1], reverse=False)  # Ordenar lista de parámetros basado a mínimo MSE
best_param_2nd = best_params_sorted_2nd[0]
best_params_sorted_2nd_df = pd.DataFrame(best_params_sorted_2nd, columns=['First Layer Neurons', 'Hidden Layer Neurons', 'Lagged', 'Batch Size',
'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', 'First Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'MSE'])

print("Finished")
print("Best combination of parameters:", best_param_2nd[:-1])
print("MSE:", best_param_2nd[-1])

pivot = best_params_sorted_2nd_df.pivot_table(
  index=['Hidden Layer Neurons'],
  columns= ('First Layer Neurons'),
  values='MSE',
  aggfunc='mean'
)
pivot['MSE Average'] = pivot.mean(axis=1)
#pivot_sorted = pivot.sort_values(by='MSE Comb', ascending=False)
pivot.drop('MSE Average', axis=1, inplace=True)

fig = plt.figure(figsize=(8, 4))
sns.heatmap(pivot, annot=True, fmt=".4f", cmap="YlGnBu")
plt.title('Proof of Concept\nMSE Score by Hyperparameter ConFiguration')
plt.show()
fig.savefig(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_pivot_MSEheatmap_2nd.png', dpi=1000)

with open(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_best_param_2nd.pkl', 'wb') as f:
  pickle.dump(best_param_2nd, f)
```

Anexo 7. *'PoC_Training_DataAnalysis_v1.0.ipynb'*, sección de evaluación de modelo óptimo de red neuronal *NARX*, con *'dataset_training_PoC'*.

```
from keras import backend as K
from sklearn.metrics import mean_squared_error
from keras.models import load_model
import os
import pickle
from datetime import datetime
import numpy as np
date_string = datetime.now().strftime("%Y%m%d")

with open(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_best_param_2nd.pkl', 'rb') as f:
  best_param_ev = pickle.load(f)
model = load_model(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_NARX_2nd.pb')

def evaluation_model(x):
  inlet = x.iloc[:, 1:-1]
  outlet = x['ABV'].values.reshape(-1, 1)  # Assuming 'ABV' is the last column
  evaluation = model.predict(inlet).flatten()  # Flatten to make it a 1D array
  prediction = inlet.copy()
  prediction['Real ABV%'] = outlet.flatten()  # Flatten in case 'outlet' is 2D
  prediction['Model ABV%'] = evaluation
  prediction_graph(prediction)
  return prediction

def r_square(y_true, y_pred):
  ss_res = np.sum(np.square(y_true - y_pred))
```

```
    ss_tot = np.sum(np.square(y_true - np.mean(y_true)))
    r2 = 1 - (ss_res / ss_tot)
    return r2

def prediction_graph(Z):
    plt.rcParams["figure.figsize"] = [10, 5.0]
    #temp = Z.dropna(subset=['Real ABV%'])
    temp = Z
    r2 = r_square(temp['Real ABV%'], temp['Model ABV%'])
    r2 = r_square(temp['Real ABV%'], temp['Model ABV%'])
    mse = mean_squared_error(temp['Real ABV%'], temp['Model ABV%'])
    x, y1, y2   = Z['t_horas'], Z['Real ABV%'], Z['Model ABV%']
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    ax1.plot(x, y1,'b.',label = 'Training Data\nSimulated in Matlab', alpha = 0.3), plt.legend(); ax1.plot(x, y2,'r+',label = 'NARX Model', alpha = 0.4),
plt.legend()
    ax1.set_xlabel('Time (h)'); ax1.set_xlim([0, Z['t_horas'].max() * 1.09])
    ax1.set_ylim([0, max(Z['Real ABV%'].max(), Z['Model ABV%'].max()) * 1.09])
    ax1.set_ylabel('ABV%'),
    ax1.set_title("Proof of Concept\nNARX Neural Network Regression (ABV%)")

    if isinstance(best_param_ev[8], str) and len(best_param_ev[8]) > 4:
        regularizer = str(type(best_param_ev[8])).split(".")[-1][:-2]
    else:
        regularizer = 'None'

    ax1_text = (
        r'$\mathbf{Metrics}$'+'\n'+
        'r² = '+str(round(r2,3))+'\n'+
        'MSE = '+str(round(mse,4))+'\n'+
        '\n'+
        r'$\mathbf{Hyperparameters}$'+'\n'+
        'Lagged = '+str(best_param_ev[2])+'\n'+
        'Batch Size = '+str(best_param_ev[3])+'\n'+
        'Epochs = '+str(best_param_ev[4])+'\n'+
        'Optimizer = '+str(best_param_ev[5])+'\n'+
        'Loss = '+str(best_param_ev[6])+'\n'+
        'Dropout = '+str(best_param_ev[7])+'\n'+
        'Regularizer = '+regularizer+'\n'+
        'Input Layer Neurons = '+str(best_param_ev[0])+'\n'+
        'Input Layer Act. = '+str(best_param_ev[9])+'\n'+
        'Hidden Layer Neurons = '+str(best_param_ev[1])+'\n'+
        'Hidden Layer Act. = '+str(best_param_ev[10])+'\n'+
        'Output Layer Act. = '+str(best_param_ev[11])+'\n'
    )
    ax1.text(0.7, 0.575, ax1_text, fontsize=8, color='black', ha='left', va='top', transform=ax1.transAxes)
    plt.show(); fig.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_NARX_evaluation_model.png', dpi = 1000)

data_eva = data_trans
prediction = evaluation_model(data_eva)
```

Anexo 8. *'PoC_Training_DataAnalysis_v1.0.ipynb'*, sección de evaluación de ajuste con gráficos de dispersión de modelo óptimo de red neuronal *NARX*, con *'dataset_training_PoC'*.

```
from sklearn.model_selection import train_test_split
import os
import pickle
date_string = datetime.now().strftime("%Y%m%d")

with open(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_best_param_1st.pkl', 'rb') as f:
  best_param = pickle.load(f)

headers_ev = ['1st Layer Neurons', 'Hidden Layer Neurons', 'Lagged', 'Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', '1st Layer
Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'MAE', 'MSE']
results_ev_model = [(best_param_ev[0], best_param_ev[1], best_param_ev[2], best_param_ev[3], best_param_ev[4], best_param_ev[5],
best_param_ev[6], best_param_ev[7], best_param_ev[8], best_param_ev[9], best_param_ev[10], mae, mse)]
print(tabulate(results_ev_model, headers=headers_ev))

# Generación de predicciones con modelo
trainPredict = model.predict(x_train)
testPredict = model.predict(x_test)
plt.rcParams["figure.figsize"] = [7.50, 5.0]
plt.scatter(y_train, trainPredict, color='b', alpha=0.1)
plt.xlabel('Training Values (ABV%)')
plt.ylabel('Predicted Values (ABV%)')
plt.title('Proof of Concept\nTraining Data')
plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_scatter_training.png', dpi = 1000)
plt.show()

plt.scatter(y_test, testPredict, color='b', alpha=0.1)
plt.xlabel('Testing Values (ABV%)')
plt.ylabel('Predicted Values (ABV%)')
plt.title('Proof of Concept\nTesting Data')
plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_scatter_testing.png', dpi = 1000)
plt.show()
```

Anexo 9. *'PoC_Evaluation_DataAnalysis_v1.0.ipynb',* sección de pre-procesamiento de datos para generación de *'dataset_evaluation_PoC'*.

```
import pandas as pd; from pandas import read_csv
import numpy as np; from numpy import loadtxt
import matplotlib.pyplot as plt; import math
from matplotlib.legend import Legend
from sklearn.preprocessing import OneHotEncoder
import seaborn as sns
from google.colab import drive

from datetime import datetime
date_string = datetime.now().strftime("%Y%m%d")

# Funciones
def sigmoid(x):
  return 1 / (1 + np.exp(-10*(x - 0.5)))

def scaled_sigmoid(x):
  return (sigmoid(x) - sigmoid(0)) / (sigmoid(1) - sigmoid(0))

def k_to_c(x):
```

```
    return x['T'] - 273.15

def normalize_time(x):
    y = (x['t_horas'] - x['t_horas'].min()) / (x['t_horas'].max() - x['t_horas'].min())
    return y

def gain_function(x):
    return (x['T']) **2.5 * (1 - scaled_sigmoid(x['Timepoint'])) / 55 **2.5

def reorder_columns(x):
    new_order = ['T', 't_horas', 'Timepoint', 'Yeast_type_0', 'Yeast_type_1', 'OG_31', 'OG_32', 'OG_33', 'OG_34', 'OG_35', 'OG_36', 'OG_37','OG_38',
'OG_39', 'OG_40', 'OG_41', 'pH0_5.0', 'pH0_5.5', 'Gainsum', 'ABV']
    return x.reindex(columns=new_order)


def eval_graph(Z, text, z):
    plt.rcParams["figure.figsize"] = [7.50, 5.0]
    x, y1, y2 = Z['t_horas'], Z['T'], Z['ABV']
    fig, ax1 = plt.subplots()
    ax1.plot(x, y1, 'b.', label='Temperature', alpha=0.2)
    ax1.set_xlabel('Time (h)')
    ax1.set_ylim([0, 25])
    ax1.set_ylabel('Temperature (°C)')
    ax1.set_title(text + z + "\nSimulated in MATLAB")
    ax2 = ax1.twinx()
    ax2.plot(x, y2, 'r.', label='ABV%', alpha=0.25)
    ax2.set_xlim([0, Z['t_horas'].max() * 1.03])
    ax2.set_ylim([0, round(Z['ABV'].max()+0.5)])
    ax2.set_ylabel('Alcohol by Volume (%)')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')

    plt.tight_layout()
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoCEv/{date_string}_PoCEv_fermentation_profile.png', dpi = 1000)
    plt.show()

    x, y1, y2 = Z['t_horas'], Z['Gainsum'], Z['ABV']
    fig, ax1 = plt.subplots()
    ax1.plot(x, y1, 'b.', label='Gainsum', alpha=0.2)
    ax1.set_xlabel('Time (h)')
    ax1.set_ylim([0, Z['Gainsum'].max() * 1.08])
    ax1.set_ylabel('Gainsum(t)')
    ax1.set_title(text + z + "\nTransformed")
    ax2 = ax1.twinx()
    ax2.plot(x, y2, 'r.', label='ABV%', alpha=0.25)
    ax2.set_xlim([0, Z['t_horas'].max() * 1.03])
    ax2.set_ylim([0, round(Z['ABV'].max()+0.5)])
    ax2.set_ylabel('Alcohol by Volume (%)')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')
```

```
    plt.tight_layout()
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoCEv/{date_string}_PoCEv_fermentation_transformed.png', dpi = 1000)
    plt.show()

    mat_corr_1 = sns.heatmap(Z[['Gainsum', 'ABV']].corr(), annot = True, cmap = 'seismic', fmt='.3f')
    plt.title(text + z + "\nCorrelation Matrix")
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoCEv/{date_string}_PoCEv_corr_ABV.png', dpi = 1000)
    plt.show()

    pair_plot = sns.pairplot(Z[['Gainsum', 'ABV']], diag_kind='kde', plot_kws={'alpha': 0.6})
    plt.subplots_adjust(top=0.9)
    plt.suptitle(text + z + "\nPairplot", fontsize=12)
    plt.savefig(f'/content/drive/MyDrive/Colab Notebooks/PoCEv/{date_string}_PoCEv_pairplot_ABV.png', dpi = 1000)
    plt.show()

def preprocess_data(dt, y_values, x_values, VAR_values, rango):
    n_values = list(rango)
    combinations = [(x, y, n) for x in x_values for y in y_values for n in n_values]
    data_dict = {}
    for x, y, n in combinations:
        cols = [f'{VAR}_{y}_{x}_{n}' for VAR in VAR_values]
        datacol = dt[cols]
        datacol = datacol.rename(columns=dict(zip(cols, VAR_values)))
        data = dt['t_horas']
        data = pd.concat([data, datacol], axis=1)
        data_dict[f'dt_{y}_{x}_{n}'] = data
    data_trans = pd.DataFrame()
    for x, y, n in combinations:
        data_key = f"dt_{y}_{x}_{n}"
        if data_key not in data_dict:
            continue
        dt = data_dict[data_key]
        dt = dt[dt['t_horas'] <= 195]
        dt['Timepoint'] = normalize_time(dt)
        dt['OG'] = (dt['S']/10)/258.6 * (227.1/(258.6-(dt['S']/10))) * (1000/1.015) # Diferencia, Conversión S de g/L a gravedad relativa con corrección 1/1.015
vs simulación
        dt.drop(columns=['S'], inplace=True)
        dt['T'] = k_to_c(dt) # +1.0 corrección
        dt['Yeast_type'] = np.where(n%2 == 0, 2, 1)  # Tipo de levadura pendiente (1 Ale / 2 Lager / # Híbrida)
        dt['pH0'] = np.where(dt['Yeast_type'] == 2, 4.7, 5.2)
        dt['T'] = np.where(dt['Yeast_type'] == 2, dt['T']-1.25, dt['T']+1.25)
        dt['Gain'] = gain_function(dt)
        dt['Gainsum'] = dt['Gain'].cumsum()
        dt.drop(columns=['Gain'], inplace=True)
        dt = pd.DataFrame(dt)
        dt = dt.reset_index(drop=True)
        data_trans = pd.concat([data_trans, dt], axis=0, ignore_index=True)

    data_trans.drop(columns=['X'], inplace=True)
    data_trans = encode_and_bind(data_trans, 'Yeast_type')
    data_trans = encode_and_bin_ranges(data_trans, 'OG', gravity_bins, gravity_labels)
    data_trans = encode_and_bin_ranges(data_trans, 'pH0', pH_bins, pH_labels)
    data_trans = reorder_columns(data_trans)

    rows_to_drop = int(len(data_trans) * 0.6)
    indices_to_drop = data_trans.sample(n=rows_to_drop).index
    data_trans = data_trans.drop(indices_to_drop)
```

```
    data_trans.to_csv("/content/drive/MyDrive//Colab Notebooks/PoCEv/dataset_evaluation_v2_transformed.csv")
    return data_trans

def encode_and_bind(original_dataframe, column_to_encode):
    encoder = OneHotEncoder(sparse_output=False)
    encoded_columns = encoder.fit_transform(original_dataframe[[column_to_encode]])
    encoded_df = pd.DataFrame(encoded_columns,
                              index=original_dataframe.index,
                              columns=[f'{column_to_encode}_{i}' for i in range(encoded_columns.shape[1])])
    without_column = original_dataframe.drop(column_to_encode, axis=1)
    encoded_dataframe = pd.concat([without_column, encoded_df], axis=1)
    return encoded_dataframe

def encode_and_bin_ranges(df, column_to_encode, bins, labels):
    df['bin'] = pd.cut(df[column_to_encode], bins=bins, labels=labels, right=False)
    one_hot_encoded_df = pd.get_dummies(df['bin'], prefix=column_to_encode)
    df = df.drop([column_to_encode, 'bin'], axis=1)
    df = pd.concat([df, one_hot_encoded_df], axis=1)
    return df

########## Datasets Generados en Matlab
drive.mount('/content/drive', force_remount=True)
dt = pd.read_csv("/content/drive/MyDrive//Colab Notebooks/PoCEv/dataset_evaluation_v2.csv", delimiter = ',')                    # 320 - 24 horas de
estabilización de ABV%
gravity_bins = [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]
gravity_labels = ['31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41']
pH_bins = [4.0, 5.0, 5.5]
pH_labels = ['5.0', '5.5']

VAR_values = ['T', 'ABV', 'X', 'S']       # Nombres de columnas
y_values = [96, 100, 104, 108]          # g/L
x_values = [100]        # g/L -- 80 to 120 g per 100 L ≈ 1 g/L
rango = range(1, 5)                  # número de perfiles simulados dentro de dataset original = 4

data_trans  = preprocess_data(dt, y_values, x_values, VAR_values, rango)
eval_graph(data_trans, "Fermentation Profiles - ", "Proof of Concept")

X = data_trans.iloc[:, 1:-1]
y = data_trans.iloc[:, [-1]]  # Variable objetivo

print("El tamaño del arreglo de entrada (X) es:", X.shape)
print("El tamaño del arreglo de salida (y) es:", y.shape)
colm_X = X.shape[1]
```

Anexo 10. *'PoC_Evaluation_DataAnalysis_v1.0.ipynb',* sección de evaluación de modelo óptimo de red neuronal *NARX*, con *'dataset_evaluation_PoC'*.

```
from keras import backend as K
from sklearn.metrics import mean_squared_error
from keras.models import load_model
import os
import pickle
from datetime import datetime
import numpy as np
date_string = datetime.now().strftime("%Y%m%d")

with open(f'/content/drive/MyDrive/Colab Notebooks/PoC/{date_string}_PoC_best_param_2nd.pkl', 'rb') as f:
  best_param_ev = pickle.load(f)
model = load_model(f'/content/drive/MyDrive//Colab Notebooks/PoC/{date_string}_PoC_NARX_2nd.pb')

def evaluation_model(x):
  inlet = x.iloc[:, 1:-1]
  outlet = x['ABV'].values.reshape(-1, 1)  # Assuming 'ABV' is the last column
  evaluation = model.predict(inlet).flatten()  # Flatten to make it a 1D array
  prediction = inlet.copy()
  prediction['Real ABV%'] = outlet.flatten()  # Flatten in case 'outlet' is 2D
  prediction['Model ABV%'] = evaluation
  prediction_graph(prediction)
  return prediction

def r_square(y_true, y_pred):
  ss_res = np.sum(np.square(y_true - y_pred))
  ss_tot = np.sum(np.square(y_true - np.mean(y_true)))
  r2 = 1 - (ss_res / ss_tot)
  return r2

def prediction_graph(Z):
    plt.rcParams["figure.figsize"] = [10, 5.0]
    #temp = Z.dropna(subset=['Real ABV%'])
    temp = Z
    r2 = r_square(temp['Real ABV%'], temp['Model ABV%'])
    r2 = r_square(temp['Real ABV%'], temp['Model ABV%'])
    mse = mean_squared_error(temp['Real ABV%'], temp['Model ABV%'])
    x, y1, y2   = Z['t_horas'], Z['Real ABV%'], Z['Model ABV%']
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    ax1.plot(x, y1,'b.',label = 'Evaluation Data\nSimulated in Matlab', alpha = 0.3), plt.legend(); ax1.plot(x, y2,'r+',label = 'NARX Model', alpha = 0.4),
plt.legend()
    ax1.set_xlabel('Time (h)'); ax1.set_xlim([0, Z['t_horas'].max() * 1.05])
    ax1.set_ylim([0, max(round(Z['Real ABV%'].max()), round(Z['Model ABV%'].max())) + 1])
    ax1.set_ylabel('ABV%'),
    #ax1.set_title(f"NARX Neural Network Regression - {date_string} - PoCEv Evaluation")
    ax1.set_title("Proof of Concept Evaluation\nNARX Neural Network Regression")

    if isinstance(best_param_ev[8], str) and len(best_param_ev[8]) > 4:
      regularizer = str(type(best_param_ev[8])).split(".")[-1][:-2]
    else:
      regularizer = 'None'

    ax1_text = (
      r'$\mathbf{Metrics}$'+'\n'+
      'r² = '+str(round(r2,3))+'\n'+
```

```
        'MSE = '+str(round(mse,4))+'\n'+
        '\n'+
        r'$\mathbf{Hyperparameters}$'+'\n'+
        'Lagged = '+str(best_param_ev[2])+'\n'+
        'Batch Size = '+str(best_param_ev[3])+'\n'+
        'Epochs = '+str(best_param_ev[4])+'\n'+
        'Optimizer = '+str(best_param_ev[5])+'\n'+
        'Loss = '+str(best_param_ev[6])+'\n'+
        'Dropout = '+str(best_param_ev[7])+'\n'+
        'Regularizer = '+regularizer+'\n'+
        'Input Layer Neurons = '+str(best_param_ev[0])+'\n'+
        'Input Layer Act. = '+str(best_param_ev[9])+'\n'+
        'Hidden Layer Neurons = '+str(best_param_ev[1])+'\n'+
        'Hidden Layer Act. = '+str(best_param_ev[10])+'\n'+
        'Output Layer Act. = '+str(best_param_ev[11])+'\n'
    )
    ax1.text(0.7, 0.575, ax1_text, fontsize=8, color='black', ha='left', va='top', transform=ax1.transAxes)
    plt.show(); fig.savefig(f'/content/drive/MyDrive//Colab Notebooks/PoCEv/{date_string}_PoCEv_NARX_evaluation_model.png', dpi = 1000)

data_eva = data_trans
prediction = evaluation_model(data_eva)
```

Anexo 11. *'PoC_BreweryHub_DataAnalysis_v1.0.ipynb',* sección de pre-procesamiento de datos para generación de *'dataset_training_brew'*.

```
import pandas as pd; from pandas import read_csv
import numpy as np; from numpy import loadtxt
import matplotlib.pyplot as plt; import math
from matplotlib.legend import Legend
from sklearn.preprocessing import OneHotEncoder
import seaborn as sns
from google.colab import drive
from datetime import datetime

date_string = datetime.now().strftime("%Y%m%d")

# Funciones
def sigmoid(x):
  return 1 / (1 + np.exp(-10*(x - 0.5)))

def scaled_sigmoid(x):
  return (sigmoid(x) - sigmoid(0)) / (sigmoid(1) - sigmoid(0))

def k_to_c(x):
  return x['T'] - 273.15

def normalize_time(x):
  y = (x['t_horas'] - x['t_horas'].min()) / (x['t_horas'].max() - x['t_horas'].min())
  return y

def gain_function(x):
  return (x['T']) **2.5 * (1 - scaled_sigmoid(x['Timepoint'])) / 55 **2.5

def reorder_columns(x):
  new_order = ['T', 't_horas', 'Timepoint', 'Yeast_type_0', 'Yeast_type_1', 'OG_31', 'OG_32', 'OG_33', 'OG_34', 'OG_35', 'OG_36', 'OG_37','OG_38',
'OG_39', 'OG_40', 'OG_41', 'pH0_5.0', 'pH0_5.5', 'Gainsum', 'ABV']
```

```
    return x.reindex(columns=new_order)

def eval_graph(Z, text, z):
    plt.rcParams["figure.figsize"] = [7.50, 5.0]
    x, y1, y2 = Z['t_horas'], Z['T'], Z['ABV']
    fig, ax1 = plt.subplots()
    ax1.plot(x, y1, 'b.', label='Temperature (°C)', alpha=0.2)
    ax1.set_xlabel('Time (hours)')
    ax1.set_ylim([0, 25])
    ax1.set_ylabel('Temperature (°C)')
    ax1.set_title(text + z + "\nTraining Data Simulated in MATLAB")
    ax2 = ax1.twinx()
    ax2.plot(x, y2, 'r.', label='ABV%', alpha=0.25)
    ax2.set_xlim([0, Z['t_horas'].max() * 1.03])
    ax2.set_ylim([0, Z['ABV'].max() * 1.08])
    ax2.set_ylabel('Alcohol by Volume (%)')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')

    plt.tight_layout()
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_Training_fermentation_profile.png', dpi = 1000)
    plt.show()

    x, y1, y2 = Z['t_horas'], Z['Gainsum'], Z['ABV']
    fig, ax1 = plt.subplots()
    ax1.plot(x, y1, 'b.', label='Gainsum', alpha=0.2)
    ax1.set_xlabel('Time (h)')
    ax1.set_ylim([0, Z['Gainsum'].max() * 1.08])
    ax1.set_ylabel('Gainsum(t)')
    ax1.set_title(text + z + "\nTransformed")
    ax2 = ax1.twinx()
    ax2.plot(x, y2, 'r.', label='ABV%', alpha=0.25)
    ax2.set_xlim([0, Z['t_horas'].max() * 1.03])
    ax2.set_ylim([0, Z['ABV'].max() * 1.08])
    ax2.set_ylabel('Alcohol by Volume (%)')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')

    plt.tight_layout()
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_Training_fermentation_transformed.png', dpi = 1000)
    plt.show()

    mat_corr_1 = sns.heatmap(Z[['Gainsum', 'ABV']].corr(), annot = True, cmap = 'seismic', fmt='.3f')
    plt.title(text + z + "\nCorrelation Matrix")
    plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_Training_corr_ABV.png', dpi = 1000)
    plt.show()

    pair_plot = sns.pairplot(Z[['Gainsum', 'ABV']], diag_kind='kde', plot_kws={'alpha': 0.6})
    plt.subplots_adjust(top=0.9)
    plt.suptitle(text + z + "\nPairplot", fontsize=12)
```

```
    plt.savefig(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_Training_pairplot_ABV.png', dpi = 1000)
    plt.show()

def preprocess_data(dt, y_values, x_values, VAR_values, rango):
    n_values = list(rango)
    combinations = [(x, y, n) for x in x_values for y in y_values for n in n_values]
    data_dict = {}
    for x, y, n in combinations:
        cols = [f'{VAR}_{y}_{x}_{n}' for VAR in VAR_values]
        datacol = dt[cols]
        datacol = datacol.rename(columns=dict(zip(cols, VAR_values)))
        data = dt['t_horas']
        data = pd.concat([data, datacol], axis=1)
        data_dict[f'dt_{y}_{x}_{n}'] = data
    data_trans = pd.DataFrame()
    for x, y, n in combinations:
        data_key = f"dt_{y}_{x}_{n}"
        if data_key not in data_dict:
            continue
        dt = data_dict[data_key]
        dt = dt[dt['t_horas'] <= 195]
        dt['Timepoint'] = normalize_time(dt)
        dt['OG'] = (dt['S']/10)/258.6 * (227.1/(258.6-(dt['S']/10))) * (1000/1.015) # Diferencia, Conversión S de g/L a gravedad relativa con corrección 1/1.015
vs simulación
        dt.drop(columns=['S'], inplace=True)
        dt['T'] = k_to_c(dt)
        dt['Yeast_type'] = np.where(n%2 == 0, 2, 1)  # Tipo de levadura pendiente (1 Ale / 2 Lager / # Híbrida)
        dt['pH0'] = np.where(dt['Yeast_type'] == 2, 4.7, 5.2)
        dt['T'] = np.where(dt['Yeast_type'] == 2, dt['T']-1.25, dt['T']+1.25)
        dt['Gain'] = gain_function(dt)
        dt['Gainsum'] = dt['Gain'].cumsum()
        dt.drop(columns=['Gain'], inplace=True)
        dt = pd.DataFrame(dt)
        dt = dt.reset_index(drop=True)
        data_trans = pd.concat([data_trans, dt], axis=0, ignore_index=True)

    data_trans.drop(columns=['X'], inplace=True)
    data_trans = encode_and_bind(data_trans, 'Yeast_type')
    data_trans = encode_and_bin_ranges(data_trans, 'OG', gravity_bins, gravity_labels)
    data_trans = encode_and_bin_ranges(data_trans, 'pH0', pH_bins, pH_labels)
    data_trans = reorder_columns(data_trans)

    rows_to_drop = int(len(data_trans) * 0.7)
    indices_to_drop = data_trans.sample(n=rows_to_drop).index
    data_trans = data_trans.drop(indices_to_drop)

    data_trans.to_csv("/content/drive/MyDrive//Colab Notebooks/BreweryHub/dataset_training_v3_transformed.csv")
    return data_trans

def encode_and_bind(original_dataframe, column_to_encode):
    encoder = OneHotEncoder(sparse_output=False)
    encoded_columns = encoder.fit_transform(original_dataframe[[column_to_encode]])
    encoded_df = pd.DataFrame(encoded_columns,
                              index=original_dataframe.index,
                              columns=[f'{column_to_encode}_{i}' for i in range(encoded_columns.shape[1])])
    without_column = original_dataframe.drop(column_to_encode, axis=1)
    encoded_dataframe = pd.concat([without_column, encoded_df], axis=1)
    return encoded_dataframe
```

```
def encode_and_bin_ranges(df, column_to_encode, bins, labels):
  df['bin'] = pd.cut(df[column_to_encode], bins=bins, labels=labels, right=False)
  one_hot_encoded_df = pd.get_dummies(df['bin'], prefix=column_to_encode)
  df = df.drop([column_to_encode, 'bin'], axis=1)
  df = pd.concat([df, one_hot_encoded_df], axis=1)
  return df

########## Datasets Generados en Matlab
drive.mount('/content/drive', force_remount=True)
dt = pd.read_csv("/content/drive/MyDrive//Colab Notebooks/BreweryHub/dataset_training_v3.csv", delimiter = ',')                    # 320 - 24 horas de
estabilización de ABV%
gravity_bins = [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]
gravity_labels = ['31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41']
pH_bins = [4.0, 5.0, 5.5]
pH_labels = ['5.0', '5.5']

VAR_values = ['T', 'ABV', 'X', 'S']       # Nombres de columnas
y_values = [88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112]         # g/L
x_values = [100]        # g/L -- 80 to 120 g per 100 L ≈ 1 g/L
rango = range(1, 7)                  # número de simulaciones dentro de dataset original = 10

data_trans  = preprocess_data(dt, y_values, x_values, VAR_values, rango)
eval_graph(data_trans, "Fermentation Profiles Proposed by ", "Cerveza Loba")

data_trans_train = pd.read_csv("/content/drive/MyDrive//Colab Notebooks/PoC/dataset_training_v2_transformed.csv", delimiter = ',')
data_trans_train = data_trans_train.drop(columns=['T'])
rows_to_drop = int(len(data_trans_train) * 0.8)
indices_to_drop = data_trans_train.sample(n=rows_to_drop).index
data_trans_train = data_trans_train.drop(indices_to_drop)

data_trans_eva = pd.read_csv("/content/drive/MyDrive//Colab Notebooks/PoCEv/dataset_evaluation_v2_transformed.csv", delimiter = ',')
data_trans_eva = data_trans_eva.drop(columns=['T'])
rows_to_drop = int(len(data_trans_eva) * 0.8)
indices_to_drop = data_trans_eva.sample(n=rows_to_drop).index
data_trans_eva = data_trans_eva.drop(indices_to_drop)

data_trans_brew = pd.read_csv("/content/drive/MyDrive//Colab Notebooks/BreweryHub/dataset_training_v3_transformed.csv", delimiter = ',')
data_trans_brew = data_trans_brew.drop(columns=['T'])

data_trans = pd.concat([data_trans_train, data_trans_eva, data_trans_brew], axis=0, ignore_index=True)

X = data_trans.iloc[:, 1:-1]
y = data_trans.iloc[:, [-1]]  # Variable objetivo
colm_X = X.shape[1]

print("El tamaño del arreglo de entrada (X) es:", X.shape)
print("El tamaño del arreglo de salida (y) es:", y.shape)
```

Anexo 12. *'PoC_BreweryHub_DataAnalysis_v1.0.ipynb'*, sección de primera optimización por búsqueda exhaustiva (*grid*) para una arquitectura de modelo de red neuronal *NARX*, con *'dataset_training_PoC'*, *'dataset_evaluation_PoC'* y *'dataset_training_brew'*.

```
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from keras.layers import Dense, Dropout
from keras import regularizers
from keras.optimizers import Adam
from tabulate import tabulate
import os
import pickle
from datetime import datetime
date_string = datetime.now().strftime("%Y%m%d")

firstlayer_act = ['hard_sigmoid', 'sigmoid', 'tanh'] # 'sigmoid', 'tanh', 'relu', 'selu', 'elu', 'gelu', 'linear', 'hard_sigmoid'
hiddenlayer_act = ['sigmoid', 'tanh'] # 'hard_sigmoid', 'sigmoid', 'linear',
outputlayer_act = ['linear']
regularizer = [None, regularizers.l1(0.0005), regularizers.l2(0.0005)] # none, regularizers.l2(0.0005)
dropouts = [0] # 0.1
losses = ['mse'] # 'mae'
optimizers = ['adam'] # 'RMSprop', 'SGD', 'Adagrad'
epochs = [100, 150]; batchs = [100, 150] # 150, 225, 300
early_stopping = EarlyStopping(monitor = 'mse', patience = 3)
n_lags = [0, 1] # 1
fl_values = [colm_X * 4] # Columns x
hl_values = [round(fl_values[0] / 2)] # fl/2
headers = ['n','Lagged','Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', '1st Layer Activation', 'Hidden Layer Activation', 'Output
Layer Activation', 'First Layer Neurons', 'Hidden Layer Neurons', 'MSE*MAE', 'MSE']

total_iterations = len(n_lags) * len(batchs) * len(epochs) * len(optimizers) * len(losses) * len(dropouts) * len(regularizer) * len(firstlayer_act) *
len(hiddenlayer_act) * len(outputlayer_act) * len(fl_values) * len(hl_values)
n_1st = 0
best_params_1st = []

for lag in n_lags:
  y_lagged = y.shift(periods=lag, fill_value=0)
  X_narx = pd.concat([X, y_lagged], axis=1).iloc[lag:]
  x_train, x_test, y_train, y_test = train_test_split(X_narx.iloc[:,:-1], X_narx.iloc[:,-1], test_size=0.10, random_state=0)
  for batch in batchs:
    for epoch in epochs:
      for opt in optimizers:
        for l in losses:
          for dropout in dropouts:
            for reg in regularizer:
              for flact in firstlayer_act:
                for hlact in hiddenlayer_act:
                  for olact in outputlayer_act:
                    for fl in fl_values:
                      for hl in hl_values:
                        model = Sequential()
                        model.add(Dense(fl, input_dim=X_narx.shape[1]-1, activation=flact, kernel_regularizer=reg))
                        model.add(Dropout(dropout))
                        model.add(Dense(hl, activation=hlact, kernel_regularizer=reg))
                        model.add(Dense(1, activation=olact))
                        model.compile(loss=l, optimizer=opt, metrics=['mse', 'mae'])
```

```
                                model.fit(x_train, y_train, epochs=epoch, batch_size=batch, verbose=0, callbacks=[early_stopping])
                                _, mse, mae = model.evaluate(X, y, verbose=0)
                                n_1st += 1
                                progress_percent = (n_1st / total_iterations) * 100
                                best_params_1st.append((lag, batch, epoch, opt, l, dropout, reg, flact, hlact, olact, fl, hl, mse*mae, mse))
                                results = [(n_1st, lag, batch, epoch, opt, l, dropout, str(type(reg)).split(".")[-1][:-2], flact, hlact, olact, fl, hl, mse, mae) for n_1st,
(lag, batch, epoch, opt, l, dropout, reg, flact, hlact, olact, fl, hl, mse, mae) in enumerate(best_params_1st, 1)]
                                print(f'\rProgress: {n_1st}/{total_iterations} ({progress_percent:.2f}%)', end='', flush=True)

print(" Finished!")
print(tabulate(results, headers=headers))
best_params_pivot = pd.DataFrame(best_params_1st, columns=['Lagged','Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', '1st
Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'First Layer Neurons', 'Hidden Layer Neurons', 'MSE*MAE', 'MSE'])
best_params_pivot['Regularizer'] = best_params_pivot['Regularizer'].apply(lambda x: x.__class__.__name__)
best_params_pivot['Regularizer'] = np.where(best_params_pivot['Regularizer'] == "NoneType", "None", best_params_pivot['Regularizer'])
best_params_sorted_1st = sorted(best_params_1st, key=lambda x: x[-1], reverse=False)  # Ordenar lista de parámetros basado a mínimo MSE
best_param_1st = best_params_sorted_1st[0]
best_param_sorted_1st_df = pd.DataFrame(best_params_sorted_1st, columns=['Lagged','Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout',
'Regularizer', '1st Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'First Layer Neurons', 'Hidden Layer Neurons', 'MSE*MAE',
'MSE'])
best_param_sorted_1st_df['Regularizer'] = best_param_sorted_1st_df['Regularizer'].apply(lambda x: x.__class__.__name__)
best_param_sorted_1st_df.to_csv("/content/drive/MyDrive//Colab Notebooks/BreweryHub/best_param_sorted_1st.csv")
print("Best combination of parameters:")
best_combination = best_param_sorted_1st_df.iloc[0, :-2]
print(best_combination.to_string())
print("MSE:", best_param_1st[-1])

pivot = best_params_pivot.pivot_table(
  index=['Batch Size', 'Epochs', '1st Layer Activation', 'Hidden Layer Activation', 'Optimizer'],
  columns= ('Lagged', 'First Layer Neurons', 'Hidden Layer Neurons','Regularizer'),
  values='MSE',
  aggfunc='mean'
)
fig = plt.figure(figsize=(1.8 * len(n_lags) * len(fl_values) * len(hl_values) * len(regularizer), 0.55 * len(batchs) * len(epochs) * len(optimizers *
len(firstlayer_act) * len(hiddenlayer_act))))
sns.heatmap(pivot, annot=True, fmt=".4f", cmap="YlGnBu")
plt.title('Training Data\nAverage MSE Score by Hyperparameter ConFiguration')
fig.savefig(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_Training_pivot_MSEheatmap_1st.png', dpi=1000)
plt.show()

with open(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_best_param_1st.pkl', 'wb') as f:
  pickle.dump(best_param_1st, f)
```

Anexo 13. *'PoC_BreweryHub_DataAnalysis_v1.0.ipynb'*, sección de segunda optimización por búsqueda exhaustiva (*grid*) para una arquitectura de modelo de red neuronal *NARX*, con *'dataset_training_PoC', 'dataset_evaluation_PoC'* y *'dataset_training_brew'*.

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Dense, Dropout
from keras import regularizers
from keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tabulate import tabulate
import os
import pickle
from datetime import datetime
date_string = datetime.now().strftime("%Y%m%d")
#date_string_ant = 20231111

with open(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_best_param_1st.pkl', 'rb') as f:
  best_param = pickle.load(f)

fl_values = [colm_X * 3, colm_X * 4, colm_X * 5] # Columns x n
hl_values = [(2 * colm_X)-2, round(fl_values[0] / 2), round(fl_values[1] / 2), round(fl_values[2] / 2)] # ROUND(fl/2)

total_iterations = len(fl_values) * len(hl_values)
early_stopping = EarlyStopping(monitor = 'mse', patience = 5)
headers_2nd = ['First Layer Neurons', 'Hidden Layer Neurons', 'Lagged', 'Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', 'First
Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'MSE']
best_mse = float('inf')
best_model = None
n_2nd = 0
best_params_2nd = []

for fl in fl_values:
  for hl in hl_values:
    y_lagged = y.shift(periods=best_param[0], fill_value=0)                  # Variable target
    X_narx = pd.concat([X, y_lagged], axis=1).iloc[best_param[0]:]              # Combina input original + lagged
    x_train, x_test, y_train, y_test = train_test_split(X_narx.iloc[:,:-1], X_narx.iloc[:,-1], test_size=0.30, random_state=0)
    model = Sequential()
    model.add(Dense(fl, input_dim=X_narx.shape[1]-1, activation=best_param[7], kernel_regularizer=best_param[6])) # best_param[6]
    model.add(Dropout(best_param[5]))
    model.add(Dense(hl, activation=best_param[8], kernel_regularizer=best_param[6]))
    model.add(Dense(1, activation=best_param[9]))
    model.compile(loss=best_param[4], optimizer=best_param[3], metrics=['mse'])          # Compilación modelo
    model.fit(X, y, epochs=best_param[2], batch_size=best_param[1], verbose=0, callbacks=[early_stopping])                # Ajuste modelo a dataset
    _, mse = model.evaluate(X, y, verbose=0)
    if mse < best_mse:  # Si el mse actual es mejor que el mejor mse registrado
      best_mse = mse  # Actualiza el mejor mse
      best_model = model                                    # Evaluación del modelo
    n_2nd += 1
    progress_percent = (n_2nd / total_iterations) * 100
    best_param_tuple_2nd = (best_param[0], best_param[1], best_param[2], best_param[3], best_param[4], best_param[5], best_param[6],
best_param[7], best_param[8], best_param[9])
    best_params_2nd.append((fl, hl, *best_param_tuple_2nd, mse)) # Almacena parámetros utilizados
    results_2nd = [(n_2nd, fl, hl, *best_param_tuple_2nd,  mse) for n_2nd, (fl, hl, *best_param_tuple_2nd, mse) in enumerate(best_params_2nd, 1)]
    print(f'\rProgress: {n_2nd}/{total_iterations} ({progress_percent:.2f}%)', end='', flush=True)

best_model.save(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_NARX_2nd.pb')
```

```
#print(tabulate(results_2nd, headers=headers_2nd));
best_params_sorted_2nd = sorted(best_params_2nd, key=lambda x: x[-1], reverse=False)  # Ordenar lista de parámetros basado a mínimo MSE
best_param_2nd = best_params_sorted_2nd[0]
best_params_sorted_2nd_df = pd.DataFrame(best_params_sorted_2nd, columns=['First Layer Neurons', 'Hidden Layer Neurons', 'Lagged', 'Batch Size',
'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', 'First Layer Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'MSE'])
print("Finished")
print("Best combination of parameters:", best_param_2nd[:-1])
print("MSE:", best_param_2nd[-1])

pivot = best_params_sorted_2nd_df.pivot_table(
  index=['Hidden Layer Neurons'],
  columns= ('First Layer Neurons'),
  values='MSE',
  aggfunc='mean'
)
pivot.drop('MSE Average', axis=1, inplace=True)

fig = plt.figure(figsize=(8, 4))
sns.heatmap(pivot, annot=True, fmt=".4f", cmap="YlGnBu")
plt.title('Training Data\nMSE Score by Hyperparameter ConFiguration')
fig.savefig(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_Training_pivot_MSEheatmap_2nd.png', dpi=1000)
plt.show()

with open(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_best_param_2nd.pkl', 'wb') as f:
  pickle.dump(best_param_2nd, f)
```

Anexo 14. *'PoC_BreweryHub_DataAnalysis_v1.0.ipynb'*, sección de evaluación de modelo óptimo de red neuronal *NARX*, con *'dataset_training_PoC', 'dataset_evaluation_PoC'* y *'dataset_training_brew'.*

```
from keras import backend as K
from sklearn.metrics import mean_squared_error
from keras.models import load_model
import os
import pickle
from datetime import datetime
import numpy as np
date_string = datetime.now().strftime("%Y%m%d")

with open(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_best_param_2nd.pkl', 'rb') as f:
  best_param_ev = pickle.load(f)
model = load_model(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_NARX_2nd.pb')

def evaluation_model(x):
  inlet = x.iloc[:, 1:-1]
  outlet = x['ABV'].values.reshape(-1, 1)  # Assuming 'ABV' is the last column
  evaluation = model.predict(inlet).flatten()  # Flatten to make it a 1D array
  prediction = inlet.copy()
  prediction['Real ABV%'] = outlet.flatten()  # Flatten in case 'outlet' is 2D
  prediction['Model ABV%'] = evaluation
  prediction_graph(prediction)
  return prediction

def r_square(y_true, y_pred):
  ss_res = np.sum(np.square(y_true - y_pred))
  ss_tot = np.sum(np.square(y_true - np.mean(y_true)))
  r2 = 1 - (ss_res / ss_tot)
  return r2
```

```
def prediction_graph(Z):
    plt.rcParams["figure.figsize"] = [10, 5.0]
    #temp = Z.dropna(subset=['Real ABV%'])
    temp = Z
    r2 = r_square(temp['Real ABV%'], temp['Model ABV%'])
    r2 = r_square(temp['Real ABV%'], temp['Model ABV%'])
    mse = mean_squared_error(temp['Real ABV%'], temp['Model ABV%'])
    x, y1, y2   = Z['t_horas'], Z['Real ABV%'], Z['Model ABV%']
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    ax1.plot(x, y1,'b.',label = 'Training Data\nSimulated in Matlab', alpha = 0.3), plt.legend(); ax1.plot(x, y2,'r+',label = 'NARX Model', alpha = 0.4),
plt.legend()
    ax1.set_xlabel('Time (h)'); ax1.set_xlim([0, Z['t_horas'].max() * 1.09])
    ax1.set_ylim([0, max(Z['Real ABV%'].max(), Z['Model ABV%'].max()) * 1.09])
    ax1.set_ylabel('ABV%'),
    #ax1.set_title(f"NARX Neural Network Regression - {date_string} - BreweryHub Evaluation")
    ax1.set_title("Fermentation Profiles Proposed by Cerveza Loba\nNARX Neural Network Regression (ABV%)")
    if isinstance(best_param_ev[8], str) and len(best_param_ev[8]) > 4:
        regularizer = str(type(best_param_ev[8])).split(".")[-1][:-2]
    else:
        regularizer = 'None'

    ax1_text = (
        r'$\mathbf{Metrics}$'+'\n'+
        'r² = '+str(round(r2,3))+'\n'+
        'MSE = '+str(round(mse,4))+'\n'+
        '\n'+
        r'$\mathbf{Hyperparameters}$'+'\n'+
        'Lagged = '+str(best_param_ev[2])+'\n'+
        'Batch Size = '+str(best_param_ev[3])+'\n'+
        'Epochs = '+str(best_param_ev[4])+'\n'+
        'Optimizer = '+str(best_param_ev[5])+'\n'+
        'Loss = '+str(best_param_ev[6])+'\n'+
        'Dropout = '+str(best_param_ev[7])+'\n'+
        'Regularizer = '+regularizer+'\n'+
        'Input Layer Neurons = '+str(best_param_ev[0])+'\n'+
        'Input Layer Act. = '+str(best_param_ev[9])+'\n'+
        'Hidden Layer Neurons = '+str(best_param_ev[1])+'\n'+
        'Hidden Layer Act. = '+str(best_param_ev[10])+'\n'+
        'Output Layer Act. = '+str(best_param_ev[11])+'\n'
    )
    ax1.text(0.7, 0.575, ax1_text, fontsize=8, color='black', ha='left', va='top', transform=ax1.transAxes)
    plt.show(); fig.savefig(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_Training_NARX_evaluation_model.png', dpi = 1000)

data_trans_brew = pd.read_csv("/content/drive/MyDrive//Colab Notebooks/BreweryHub/dataset_training_v3_transformed.csv", delimiter = ',')

rows_to_drop = int(len(data_trans_brew) * 0.5)
indices_to_drop = data_trans_brew.sample(n=rows_to_drop).index
data_trans_brew = data_trans_brew.drop(indices_to_drop)
data_trans_brew = data_trans_brew.drop(columns=['T'])

data_eva = data_trans_brew
prediction = evaluation_model(data_eva)
```

Anexo 15. *'PoC_BreweryHub_DataAnalysis_v1.0.ipynb'*, sección de evaluación de ajuste con gráficos de dispersión de modelo óptimo de red neuronal *NARX*, con *'dataset_training_PoC'*.

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from keras.layers import Dense, Dropout
from keras import regularizers
from keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tabulate import tabulate
import os
import pickle

with open(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_best_param_1st.pkl', 'rb') as f:
    best_param = pickle.load(f)

y_lagged = y.shift(periods=best_param_ev[2], fill_value=0)                    # Variable target
X_narx = pd.concat([X, y_lagged], axis=1).iloc[best_param_ev[2]:]             # Combina input original + lagged
x_train, x_test, y_train, y_test = train_test_split(X_narx.iloc[:,:-1], X_narx.iloc[:,-1], test_size=0.20, random_state=0)

model = Sequential()
model.add(Dense(best_param_ev[0], input_dim=X_narx.shape[1]-1, activation=best_param_ev[9], kernel_regularizer=best_param_ev[8]))
model.add(Dropout(best_param_ev[7]))
model.add(Dense(best_param_ev[1], activation=best_param_ev[10], kernel_regularizer=best_param_ev[8]))
model.add(Dense(1, activation=best_param_ev[11]))
model.compile(loss=best_param_ev[6], optimizer=best_param_ev[5], metrics=['mae', 'mse'])          # Compilación modelo
model.fit(X, y, epochs=best_param_ev[4]+50, batch_size=best_param_ev[3]+50, verbose=0)            # Ajuste modelo a dataset
_, mae, mse = model.evaluate(X, y, verbose=0)                                                    # Evaluación del modelo

headers_ev = ['1st Layer Neurons', 'Hidden Layer Neurons', 'Lagged', 'Batch Size', 'Epochs', 'Optimizer', 'Loss Function', 'Dropout', 'Regularizer', '1st Layer
Activation', 'Hidden Layer Activation', 'Output Layer Activation', 'MAE', 'MSE']
results_ev_model = [(best_param_ev[0], best_param_ev[1], best_param_ev[2], best_param_ev[3], best_param_ev[4], best_param_ev[5],
best_param_ev[6], best_param_ev[7], best_param_ev[8], best_param_ev[9], best_param_ev[10], mae, mse)]
print(tabulate(results_ev_model, headers=headers_ev))

# Generación de predicciones con modelo
trainPredict = model.predict(x_train)
testPredict = model.predict(x_test)
plt.rcParams["figure.figsize"] = [7.50, 5.0]
plt.scatter(y_train, trainPredict, color='b', alpha=0.1)
plt.xlabel('Training Values (ABV%)')
plt.ylabel('Predicted Values (ABV%)')
plt.title('Proof of Concept\nTraining Data')
plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_Training_scatter_training.png', dpi = 1000)
plt.show()

plt.scatter(y_test, testPredict, color='b', alpha=0.1)
plt.xlabel('Testing Values (ABV%)')
plt.ylabel('Predicted Values (ABV%)')
plt.title('Proof of Concept\nTesting Data')
plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_Training_scatter_testing.png', dpi = 1000)
plt.show()
```

Anexo 16. *'PI_BreweryHub_DataAnalysis_v1.0.ipynb',* sección de pre-procesamiento de datos para generación de *'dataset_brew'* (para lotes #395 y #400).

```
import pandas as pd; from pandas import read_csv
import numpy as np; from numpy import loadtxt
import matplotlib.pyplot as plt; import math
from matplotlib.legend import Legend
from sklearn.preprocessing import OneHotEncoder
import seaborn as sns
from google.colab import drive

# Constants
DATE_FORMAT = '%Y-%m-%d %H:%M'
CSV_DATE_FORMAT = '%d/%m/%y %H:%M'
ABV_FACTOR = 131.25

# Funciones
def sigmoid(x):
  return 1 / (1 + np.exp(-10*(x - 0.5)))

def scaled_sigmoid(x):
  return (sigmoid(x) - sigmoid(0)) / (sigmoid(1) - sigmoid(0))

def normalize_time(x):
  y = (x['t_horas'] - x['t_horas'].min()) / (x['t_horas'].max() - x['t_horas'].min())
  return y

def gain_function(x):
  return (x['T']) **2.5 * (1 - scaled_sigmoid(x['Timepoint'])) / 55 **2.5

def eval_graph(Z, z, lot):
    plt.rcParams["figure.figsize"] = [7.5, 5]
    x, y1, y2, y3 = Z['t_horas'], Z['T'], Z['ABV'], Z['T'].rolling(window=4).mean()

    fig = plt.figure(); ax1 = fig.add_subplot(111)
    ax1.plot(x, y1,'b.', label = 'Temperature', alpha=0.4), ax1.plot(x, y3,'b-', alpha=0.5), ax1.set_xlabel('Time (hours)')
    ax1.set_ylim([9.4, 24.3]), ax1.set_ylabel('Temperature (°C)'), ax1.set_title("Fermentation Profile - "+ z)
    y2_interpolated = pd.Series(y2).reindex(x.index).interpolate()
    ax2 = ax1.twinx(); ax2.plot(x, y2_interpolated, 'r-', alpha=0.1),
    ax2.plot(x, y2, 'rx', label = 'ABV%', markersize=10, alpha=0.9),
    ax2.set_xlim(-10, 215),
    ax2.set_ylim([0, Z['ABV'].max() * 1.08])
    ax2.set_ylabel('Alcohol by Volume (%)'), ax2.set_xlabel('Time (hours)')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')
    plt.tight_layout(), plt.show(); plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/Loba/{lot}_fermentation_profile.png', dpi = 1000)

    x, y1, y2 = Z['t_horas'], Z['Gainsum'], Z['ABV']
    fig = plt.figure(); ax1 = fig.add_subplot(111)
    ax1.plot(x, y1,'b.', label = 'Gainsum', alpha=0.6), ax1.set_xlabel('Time (hours)')
    ax1.set_ylim([0, Z['Gainsum'].max() * 1.08]), ax1.set_ylabel('Gainsum(t)'), ax1.set_title("Transformed Data - "+ z)
    ax2 = ax1.twinx(); ax2.plot(x, y2, 'rx', label = 'ABV%', markersize=10, alpha=0.9),
    ax2.set_xlim(-10, 215),
    ax2.set_ylim([0, Z['ABV'].max() * 1.08])
```

```
    ax2.set_ylabel('Alcohol by Volume (%)'), ax2.set_xlabel('Time (hours)')

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')
    plt.tight_layout(), plt.show(), plt.savefig(f'/content/drive/MyDrive//Colab Notebooks/Loba/{lot}_fermentation_transformed.png', dpi = 1000)
    plt.figure(figsize=[7.5, 5])
    correlation_matrix = Z[['Gainsum', 'ABV']].corr()
    mat_corr_1 = sns.heatmap(correlation_matrix, annot=True, cmap='seismic', fmt='.3f')
    plt.title(f'Correlation Matrix - ' + z)  # Assuming 'label' is a column in Z that contains the desired title.
    plt.show()
    plt.savefig(f'/content/drive/MyDrive/Colab Notebooks/Loba/corr_ABV_{lot}.png', dpi=1000)

    pair_plot = sns.pairplot(Z[['Gainsum', 'ABV']], diag_kind='kde', plot_kws={'alpha': 0.8})
    plt.subplots_adjust(top=0.9)
    plt.suptitle('Pairplot - ' + z, fontsize=12)
    plt.show()
    plt.savefig(f'/content/drive/MyDrive/Colab Notebooks/Loba/pairplot_ABV_{lot}.png', dpi=1000)

def reorder_columns(x):
  new_order = ['Lot', 't_horas', 'Timepoint', 'T', 'Gainsum', 'OG', 'Yeast_type', 'FG', 'pH0', 'SG', 'pH', 'ABV']
  return x.reindex(columns=new_order)

def final_columns(x):
  final_order = ['T', 't_horas', 'Timepoint', 'Yeast_type_0', 'Yeast_type_1', 'OG_31', 'OG_32', 'OG_33', 'OG_34', 'OG_35', 'OG_36', 'OG_37','OG_38',
'OG_39', 'OG_40', 'OG_41', 'pH0_5.0', 'pH0_5.5', 'Gainsum', 'ABV']
  return x.reindex(columns=final_order)

def create_batch_log(bitacora, OG):
  bi = pd.DataFrame(bitacora, columns=['Timestamp', 'SG', 'pH'])
  bi['Timestamp'] = pd.to_datetime(bi['Timestamp'], format=DATE_FORMAT)
  bi['ABV'] = (OG - bi['SG']) * ABV_FACTOR
  return bi

def process_dataset(date, db, tank, lot, bitacora, OG, FG, pH0, Lev):
  drive.mount('/content/drive', force_remount=True)
  file_path = f"/content/drive/MyDrive/Colab Notebooks/Loba/{date}_{db}_{tank}_{lot}.csv"
  ed_file_path = f"/content/drive/MyDrive/Colab Notebooks/Loba/{date}_{db}_{tank}_{lot}_ed.csv"
  bi = create_batch_log(bitacora, OG)
  dt = pd.read_csv(file_path, delimiter=',')
  dt.rename(columns={dt.columns[0]: 'Timestamp', dt.columns[1]: 'T'}, inplace=True)
  dt['Timestamp'] = pd.to_datetime(dt['Timestamp'], format=CSV_DATE_FORMAT)
  dt['t_horas'] = dt.index
  dt['OG'] = OG
  dt['FG'] = FG
  dt['OG'] = (dt['OG'] - dt['FG']) * 1000
  dt['pH0'] = pH0
  dt['Yeast_type'] = Lev
  dt = dt.dropna(subset=['Timestamp'])
  dt = pd.merge(dt, bi, on='Timestamp', how='left')
  dt = dt[dt['t_horas'] <= 195]
  dt['Timepoint'] = normalize_time(dt)
  dt = dt[dt['T'] != 0]
  dt['Gainsum'] = gain_function(dt).cumsum()
  dt.drop(columns=['Timestamp'], inplace=True)
  dt = reorder_columns(dt)
```

```
    dt.to_csv(ed_file_path)
    return dt

def encode_and_bind(original_dataframe, column_to_encode):
    encoder = OneHotEncoder(sparse=False)
    encoded_columns = encoder.fit_transform(original_dataframe[[column_to_encode]])
    encoded_df = pd.DataFrame(encoded_columns,
                              index=original_dataframe.index,
                              columns=[f'{column_to_encode}_{i}' for i in range(encoded_columns.shape[1])])
    without_column = original_dataframe.drop(column_to_encode, axis=1)
    encoded_dataframe = pd.concat([without_column, encoded_df], axis=1)
    return encoded_dataframe

def encode_and_bin_ranges(df, column_to_encode, bins, labels):
    df['bin'] = pd.cut(df[column_to_encode], bins=bins, labels=labels, right=False)
    one_hot_encoded_df = pd.get_dummies(df['bin'], prefix=column_to_encode)
    df = df.drop([column_to_encode, 'bin'], axis=1)
    df = pd.concat([df, one_hot_encoded_df], axis=1)
    return df

def process_lots(lots_info):
    results = {}
    dfs = []  # List to store individual DataFrames for each lot
    for lot, info in lots_info.items():
        combined_file_path = f"/content/drive/MyDrive/Colab Notebooks/Loba/{info['db']}_combined.csv"
        reduced_file_path = f"/content/drive/MyDrive/Colab Notebooks/Loba/{info['db']}_reduced.csv"
        processed_df = process_dataset(info['date'], info['db'], info['tank'], lot, info['bitacora'], info['OG'], info['FG'], info['pH0'], info['Lev'])
        eval_graph(processed_df, f"{info['start']} - #{lot} {info['recipe']} (F{info['tank']}) - BreweryHub", lot)
        processed_df['Lot'] = lot
        dfs.append(processed_df)
        results[lot] = processed_df

        degreeP_FG = (info['FG'] - 1) * 258.6 / (1.8808 - 0.1808 * info['FG'])
        degreeP_DR = degreeP_FG + 2
        SG_DR = degreeP_DR / 258.6 * (227.1 / (258.6 - degreeP_DR)) + 1.0
        ABV_DR = (info['OG'] - SG_DR) * 131.25
        timestamp = info['bitacora'][0][0]

    combined_df = pd.concat(dfs, ignore_index=True)
    combined_df.drop(columns=['Lot', 'SG', 'pH'], inplace=True)
    combined_df = encode_and_bin_ranges(combined_df, 'Yeast_type', yeast_bins, yeast_labels)
    combined_df = encode_and_bin_ranges(combined_df, 'OG', gravity_bins, gravity_labels)
    combined_df = encode_and_bin_ranges(combined_df, 'pH0', pH_bins, pH_labels)
    combined_df = final_columns(combined_df)
    reduced_df = combined_df.dropna(subset=['ABV'])
    reduced_df = reduced_df.reset_index(drop=True)

    not_null_abv_df = combined_df[combined_df['ABV'].isna()]
    rows_to_drop = int(len(not_null_abv_df) * 0.0)
    indices_to_drop = not_null_abv_df.sample(n=rows_to_drop).index
    combined_df = combined_df.drop(indices_to_drop)
    combined_df = combined_df.reset_index(drop=True)

    combined_df.to_csv(combined_file_path)
    reduced_df.to_csv(reduced_file_path)
    return timestamp, ABV_DR, combined_df, reduced_df, results

########## Rangos de Gravedad Relativa
```

```
yeast_bins = [0, 1.1, 2.1]
yeast_labels = ['0', '1']
gravity_bins = [30.1, 31.1, 32.1, 33.1, 34.1, 35.1, 36.1, 37.1, 38.1, 39.1, 40.1, 41]
gravity_labels = ['31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41']
pH_bins = [4.0, 5.0, 5.5]
pH_labels = ['5.0', '5.5']

########## Bitácora BreweryHub Loba
lot_395 = {
  ##
  '395': {
    'date': "20231013",
    'db': "breweryhub",
    'tank': "03",
    'recipe': "Haiku",
    'start': "05/10/23",
    'OG': 1.048,
    'FG': 1.008,
    'Lev': 2,
    'pH0': 4.7,
    't_max': 195,
    'bitacora': [
  ['2023-10-05 09:00', 1.048, 4.7],
  ['2023-10-06 13:00', 1.042, 4.7],
  ['2023-10-09 10:00', 1.013, 4.1],
  ['2023-10-10 14:00', 1.008, 4.1],
  ['2023-10-11 13:00', 1.008, 4.1],
  ['2023-10-12 13:00', 1.008, 4.1]
  ]
  }

##
}

lot_400 = {
  ##
  '400': {
    'date': "20231105",
    'db': "breweryhub",
    'tank': "04",
    'recipe': "Los Cuentos",
    'start': "28/10/23",
    'OG': 1.052, #1.044 OG + 50 kg de azúcar en 2500 L
    'FG': 1.012,
    'Lev': 1,
    'pH0': 5.2,
    't_max': 195,
    'bitacora': [
  ['2023-10-28 10:00', 1.052, 5.2], #1.044 OG + 50 kg de azúcar en 2500 L
  ['2023-10-30 11:00', 1.034, 4.5],
  ['2023-10-31 11:00', 1.030, 4.3],
  ['2023-11-01 12:00', 1.020, 4.2],
  ['2023-11-02 12:00', 1.015, 4.2],
  ['2023-11-03 10:00', 1.013, 4.3],
  ['2023-11-04 12:00', 1.013, 4.3],
  ['2023-11-05 13:00', 1.013, 4.3]
  ]
  }
```

```
##
}

timestamp_395, ABV_DR_395, data_395, data_trans, results = process_lots(lot_395)
timestamp_400, ABV_DR_400, data_400, data_trans, results = process_lots(lot_400)

X = data_trans.iloc[:, 1:-1]
y = data_trans.iloc[:, [-1]]  # Variable objetivo

print("El tamaño del arreglo de entrada (X) es:", X.shape)
print("El tamaño del arreglo de salida (y) es:", y.shape)
```

Anexo 17. *'PI_BreweryHub_DataAnalysis_v1.0.ipynb',* sección de evaluación de modelo óptimo de red neuronal *NARX* obtenido en *'PoC_BreweryHub_DataAnalysis_v1.0.ipynb'*, con *'dataset_brew'* (para lotes #395 y #400).

```
from keras import backend as K
from sklearn.metrics import mean_squared_error
from keras.models import load_model
import os
import pickle
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import numpy as np

date_string = datetime.now().strftime("%Y%m%d")

with open(f'/content/drive/MyDrive/Colab Notebooks/BreweryHub/{date_string}_best_param_2nd.pkl', 'rb') as f:
  best_param_ev = pickle.load(f)
model = load_model(f'/content/drive/MyDrive//Colab Notebooks/BreweryHub/{date_string}_NARX_2nd.pb')

def add_hours_to_timestamp(timestamp, hours):
  new_timestamp = timestamp + timedelta(hours=hours)
  return new_timestamp.strftime('%H:%M %d/%m')

def evaluation_model(x, ABV_DR, timestamp, lots_info, lot, ma):
  inlet = x.iloc[:, 1:-1]
  outlet = x.iloc[:, [-1]]
  evaluation = model.predict(inlet)
  inlet = np.array(inlet)
  outlet = np.array(outlet).flatten()
  evaluation = np.array(evaluation).flatten()
  inlet_df = pd.DataFrame(inlet, columns=['t_horas', 'Timepoint', 'Yeast_type_0', 'Yeast_type_1',
                                          'OG_31', 'OG_32', 'OG_33', 'OG_34', 'OG_35', 'OG_36',
                                          'OG_37','OG_38', 'OG_39', 'OG_40', 'OG_41', 'pH0_5.0',
                                          'pH0_5.5', 'Gainsum'])
  outlet_df = pd.DataFrame(outlet, columns=['Real ABV%'])
  evaluation_df = pd.DataFrame(evaluation, columns=['Model ABV%'])
  eval_list = evaluation_df['Model ABV%'].tolist()
  for i in range(1, len(eval_list)):
    if eval_list[i] < eval_list[i - 1]:
      if eval_list[i - 1] - eval_list[i] >= 3:
        continue
      else:
        eval_list[i] = eval_list[i - 1]
```

```
    evaluation_df['Model ABV%'] = eval_list
    prediction = pd.concat([inlet_df, outlet_df, evaluation_df], axis=1)

    prediction['ABV% Rate'] = prediction['Model ABV%'].diff()/prediction['t_horas'].diff()
    prediction['MA ABV% Rate'] = prediction['ABV% Rate'].rolling(window=ma).mean()

    prediction['act_phase'] = np.nan
    prediction['exp_phase'] = np.nan
    prediction['sta_phase'] = np.nan

    first_index = prediction[prediction['Model ABV%'] >= ABV_DR].index.min()
    prediction['dia_rest'] = np.nan
    if pd.notna(first_index):
        prediction.at[first_index, 'dia_rest'] = prediction.at[first_index, 'Model ABV%']

    after_24h = prediction[prediction['t_horas'] > 24].index
    global_max = prediction.loc[after_24h, 'MA ABV% Rate'].idxmax()
    global_min = prediction.loc[after_24h, 'MA ABV% Rate'].idxmin()
    minimos = (np.diff(np.sign(np.diff(prediction.loc[after_24h, 'MA ABV% Rate']))) > 0).nonzero()[0] + 1
    minimos += after_24h[0]

    if minimos.size > 0:
        if minimos[0] < global_max:
            prediction.loc[minimos[0], 'act_phase'] = prediction.loc[minimos[0], 'Model ABV%']
        else:
            prediction.loc[global_max - 20, 'act_phase'] = prediction.loc[global_max - 20, 'Model ABV%']
    if global_max is not None:
        prediction.loc[global_max, 'exp_phase'] = prediction.loc[global_max, 'Model ABV%']
    if global_min is not None:
        if (global_min - 48) > first_index:
            prediction.loc[global_min - 48, 'sta_phase'] = prediction.loc[global_min - 48, 'Model ABV%']
        else:
            prediction.loc[first_index + 48, 'sta_phase'] = prediction.loc[first_index + 48, 'Model ABV%']

    prediction.reset_index(drop=True)
    timestamp = datetime.strptime(timestamp, '%Y-%m-%d %H:%M')
    prediction['T'] = x['T']
    prediction_graph(prediction, timestamp, lots_info, lot, ma)

def r_square_(y_true, y_pred):
    y_true = K.constant(y_true)
    y_pred = K.constant(y_pred)
    ss_res =  K.sum(K.square(y_true - y_pred))
    ss_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return ( 1 - ss_res/(ss_tot + K.epsilon()) )

def r_square(y_true, y_pred):
    ss_res = np.sum(np.square(y_true - y_pred))
    ss_tot = np.sum(np.square(y_true - np.mean(y_true)))
    r_squared = 1 - (ss_res / ss_tot)
    return r_squared

def prediction_graph(Z, timestamp, lots_info, lot, ma):
    plt.rcParams["figure.figsize"] = [10, 5]
    temp = Z.dropna(subset=['Real ABV%'])
    r2 = r_square(temp['Real ABV%'], temp['Model ABV%'])
    mse = mean_squared_error(temp['Real ABV%'], temp['Model ABV%'])                    #4
```

```
    x, y1, y2, y3, y4, y5, y6, y7, y8 = Z['t_horas'], Z['Real ABV%'], Z['Model ABV%'], Z['MA ABV% Rate'], Z['act_phase'], Z['exp_phase'], Z['sta_phase'],
Z['dia_rest'], Z['T']
    ##
    fig, ax1 = plt.subplots()
    ax1.plot(x, y1, 'kx', label='Real ABV%', markersize=10, alpha=0.9)
    ax1.plot(x, y2, 'r.', label='Simulated Model ABV%', alpha=0.3)

    ax1.set_xlabel('Time (hours)')
    ax1.set_ylabel('ABV%', color='black')
    ax1.tick_params('y', colors='black')

    ax1.plot(0, 0, 'g>', markersize=8, alpha=0.7)
    lag_timestamp = add_hours_to_timestamp(timestamp, 0)
    ax1.text(0, 0, r'$\mathbf{Lag}$'+' '+r'$\mathbf{Phase}$'+f'\n{lag_timestamp}\n', verticalalignment='bottom', horizontalalignment='center',
fontsize=8)
    for index in y4.dropna().index:
        ax1.plot(x[index], y4[index], 'g>', markersize=8, alpha=0.7)
        y4_timestamp = add_hours_to_timestamp(timestamp, x[index])
        ax1.text(x[index], y4[index], '\n\n'+r'$\mathbf{Active}$'+' '+r'$\mathbf{Phase}$'f'\n{int((x[index])/24)}d {int((x[index])%24)}h\n{y4_timestamp}',
verticalalignment='top', horizontalalignment='center', fontsize=8)
    for index in y5.dropna().index:
        ax1.plot(x[index], y5[index], 'co', markersize=8, alpha=0.7)
        y5_timestamp = add_hours_to_timestamp(timestamp, x[index])
        ax1.text(x[index], y5[index], '\n\n'+r'$\mathbf{Exponential}$'+f'\n{int((x[index])/24)}d {int((x[index])%24)}h\n'f'{y5_timestamp}',
verticalalignment='top', horizontalalignment='center', fontsize=8)
    for index in y6.dropna().index:
        ax1.plot(x[index], y6[index], 'g>', markersize=8, alpha=0.7)
        y6_timestamp = add_hours_to_timestamp(timestamp, x[index])
        ax1.text(x[index], y6[index], '\n\n'+r'$\mathbf{Stationary}$'+' '+r'$\mathbf{Phase}$'+f'\n{int((x[index])/24)}d
{int((x[index])%24)}h\n{y6_timestamp}', verticalalignment='top', horizontalalignment='center', fontsize=8)
        end_index = y6[index]
    for index in y7.dropna().index:
        ax1.plot(x[index], y7[index], 'ys', markersize=8, alpha=0.9)
        y7_timestamp = add_hours_to_timestamp(timestamp, x[index])
        ax1.text(x[index], y7[index], r'$\mathbf{Diacetyl}$'+' '+r'$\mathbf{Rest}$'+f'\n{int((x[index])/24)}d {int((x[index])%24)}h\n{y7_timestamp}'+'\n',
verticalalignment='bottom', horizontalalignment='center', fontsize=8)
        ax1.plot(x[index]+72, end_index, marker=(5, 1), color='blue', markersize=12)
        end_timestamp = add_hours_to_timestamp(timestamp, x[index]+72)
        ax1.text(x[index]+72, end_index, '\n\n'+r'$\mathbf{End}$'+' '+r'$\mathbf{of}$'+'\n'+r'$\mathbf{Fermentation}$'+f'\n{int((x[index]+72)/24)}d
{int((x[index]+72)%24)}h\n{end_timestamp}', verticalalignment='top', horizontalalignment='center', fontsize=8)

    ax2 = ax1.twinx()
    ax2.set_ylabel('Rate of ABV%/h', color='black')
    ax2.plot(x, y3, 'c-', label=f'ABV%/h Rate ({ma}h MA)', alpha=0.20)
    ax2.tick_params('y', colors='black')
    ax2.set_ylim(0, 0.10)

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')

    title_date = timestamp.strftime('%Y/%m/%d')
    lot_key = lot
    info = lots_info[lot_key]
    ax1.set_title(f"NARX Neural Network Regression - BreweryHub\n#{lot} {info['recipe']} (F{info['tank']}, {info['start']})")
```

```
    if Z['t_horas'].max() > 215:
      ax1.set_xlim(-10, Z['t_horas'].max())
    else:
      ax1.set_xlim(-10, 215)
    if Z['Model ABV%'].max() > 6 or Z['Real ABV%'].max() > 6:
      ax1.set_ylim(-0.5, Z['Model ABV%'].max() + 0.2)
    else:
      ax1.set_ylim(-0.5, 6)

    if isinstance(best_param_ev[8], str) and len(best_param_ev[8]) > 4:
        regularizer = str(type(best_param_ev[8])).split(".")[-1][:-2]
    else:
        regularizer = 'None'

    ax1_text = (
        r'$\mathbf{Metrics}$'+'\n'+
        'r² = '+str(round(r2,3))+'\n'+
        'MSE = '+str(round(mse,4))+'\n'+
        '\n'+
        r'$\mathbf{Hyperparameters}$'+'\n'+
        'Lagged = '+str(best_param_ev[2])+'\n'+
        'Batch Size = '+str(best_param_ev[3])+'\n'+
        'Epochs = '+str(best_param_ev[4])+'\n'+
        'Optimizer = '+str(best_param_ev[5])+'\n'+
        'Loss = '+str(best_param_ev[6])+'\n'+
        'Dropout = '+str(best_param_ev[7])+'\n'+
        'Regularizer = '+regularizer+'\n'+
        'Input Layer Neurons = '+str(best_param_ev[0])+'\n'+
        'Input Layer Act. = '+str(best_param_ev[9])+'\n'+
        'Hidden Layer Neurons = '+str(best_param_ev[1])+'\n'+
        'Hidden Layer Act. = '+str(best_param_ev[10])+'\n'+
        'Output Layer Act. = '+str(best_param_ev[11])+'\n'
    )
    ax1.text(0.7, 0.575, ax1_text, fontsize=8, color='black', ha='left', va='top', transform=ax1.transAxes)
    plt.show(); fig.savefig(f"/content/drive/MyDrive/Colab Notebooks/Loba/{date_string}_{lot}_{info['recipe']}_simulated_model.png", dpi=1000)
    ##
    ##
    fig, ax1 = plt.subplots()
    ax1.plot(x, y1, 'kx', label='Real ABV%', markersize=10, alpha=0.9)
    ax1.plot(x, y2, 'r.', label='Simulated Model ABV%', alpha=0.3)

    ax1.set_xlabel('Time (hours)')
    ax1.set_ylabel('ABV%', color='black')
    ax1.tick_params('y', colors='black')

    ax1.plot(0, 0, 'g>', markersize=8, alpha=0.7)
    lag_timestamp = add_hours_to_timestamp(timestamp, 0)
    ax1.text(0, 0, r'$\mathbf{Lag}$'+' '+r'$\mathbf{Phase}$'+f'\n{lag_timestamp}\n', verticalalignment='bottom', horizontalalignment='center',
fontsize=8)
    for index in y4.dropna().index:
        ax1.plot(x[index], y4[index], 'g>', markersize=8, alpha=0.7)
        y4_timestamp = add_hours_to_timestamp(timestamp, x[index])
        ax1.text(x[index], y4[index], '\n\n'+r'$\mathbf{Active}$'+' '+r'$\mathbf{Phase}$'f'\n{int((x[index])/24)}d {int((x[index])%24)}h\n{y4_timestamp}',
verticalalignment='top', horizontalalignment='center', fontsize=8)
    for index in y5.dropna().index:
        ax1.plot(x[index], y5[index], 'co', markersize=8, alpha=0.7)
        y5_timestamp = add_hours_to_timestamp(timestamp, x[index])
```

```
        ax1.text(x[index], y5[index], '\n\n'+r'$\mathbf{Exponential}$'+f'\n{int((x[index])/24)}d {int((x[index])%24)}h\n'f'{y5_timestamp}',
verticalalignment='top', horizontalalignment='center', fontsize=8)
    for index in y6.dropna().index:
        ax1.plot(x[index], y6[index], 'g>', markersize=8, alpha=0.7)
        y6_timestamp = add_hours_to_timestamp(timestamp, x[index])
        ax1.text(x[index], y6[index], '\n\n'+r'$\mathbf{Stationary}$'+' '+r'$\mathbf{Phase}$'+f'\n{int((x[index])/24)}d
{int((x[index])%24)}h\n{y6_timestamp}', verticalalignment='top', horizontalalignment='center', fontsize=8)
        end_index = y6[index]
    for index in y7.dropna().index:
        ax1.plot(x[index], y7[index], 'ys', markersize=8, alpha=0.9)
        y7_timestamp = add_hours_to_timestamp(timestamp, x[index])
        ax1.text(x[index], y7[index], r'$\mathbf{Diacetyl}$'+' '+r'$\mathbf{Rest}$'+f'\n{int((x[index])/24)}d {int((x[index])%24)}h\n{y7_timestamp}'+'\n',
verticalalignment='bottom', horizontalalignment='center', fontsize=8)
        ax1.plot(x[index]+72, end_index, marker=(5, 1), color='blue', markersize=12)
        end_timestamp = add_hours_to_timestamp(timestamp, x[index]+72)
        ax1.text(x[index]+72, end_index, '\n\n'+r'$\mathbf{End}$'+' '+r'$\mathbf{of}$'+'\n'+r'$\mathbf{Fermentation}$'+f'\n{int((x[index]+72)/24)}d
{int((x[index]+72)%24)}h\n{end_timestamp}', verticalalignment='top', horizontalalignment='center', fontsize=8)

    ax2 = ax1.twinx()
    ax2.set_ylabel('Temperature (°C)', color='black')
    ax2.plot(x, y8, 'b.', label='Temperature', alpha=0.20)
    ax2.tick_params('y', colors='black')
    ax2.set_ylim(9.4, 24.3)

    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines = lines1 + lines2
    labels = labels1 + labels2
    ax1.legend(lines, labels, loc='upper left')

    title_date = timestamp.strftime('%Y/%m/%d')
    lot_key = lot
    info = lots_info[lot_key]
    ax1.set_title(f"NARX Neural Network Regression - BreweryHub\n#{lot} {info['recipe']} (F{info['tank']}, {info['start']})")

    if Z['t_horas'].max() > 215:
      ax1.set_xlim(-10, Z['t_horas'].max())
    else:
      ax1.set_xlim(-10, 215)
    if Z['Model ABV%'].max() > 6 or Z['Real ABV%'].max() > 6:
      ax1.set_ylim(-0.5, z['Model ABV%'].max() + 0.2)
    else:
      ax1.set_ylim(-0.5, 6)

    if isinstance(best_param_ev[8], str) and len(best_param_ev[8]) > 4:
        regularizer = str(type(best_param_ev[8])).split(".")[-1][:-2]
    else:
        regularizer = 'None'

    ax1_text = (
        r'$\mathbf{Metrics}$'+'\n'+
        'r² = '+str(round(r2,3))+'\n'+
        'MSE= '+str(round(mse,4))+'\n'+
        '\n'+
        r'$\mathbf{Hyperparameters}$'+'\n'+
        'Lagged = '+str(best_param_ev[2])+'\n'+
        'Batch Size = '+str(best_param_ev[3])+'\n'+
        'Epochs = '+str(best_param_ev[4])+'\n'+
```

```
        'Optimizer = '+str(best_param_ev[5])+'\n'+
        'Loss = '+str(best_param_ev[6])+'\n'+
        'Dropout = '+str(best_param_ev[7])+'\n'+
        'Regularizer = '+regularizer+'\n'+
        'Input Layer Neurons = '+str(best_param_ev[0])+'\n'+
        'Input Layer Act. = '+str(best_param_ev[9])+'\n'+
        'Hidden Layer Neurons = '+str(best_param_ev[1])+'\n'+
        'Hidden Layer Act. = '+str(best_param_ev[10])+'\n'+
        'Output Layer Act. = '+str(best_param_ev[11])+'\n'
    )
    ax1.text(0.7, 0.575, ax1_text, fontsize=8, color='black', ha='left', va='top', transform=ax1.transAxes)
    plt.show(); fig.savefig(f"/content/drive/MyDrive/Colab Notebooks/Loba/{date_string}_{lot}_{info['recipe']}_simulated_model_temp.png", dpi=1000)
    ##
evaluation_model(data_395, ABV_DR_395, timestamp_395, lot_395, "395", 16)
evaluation_model(data_400, ABV_DR_400, timestamp_400, lot_400, "400", 16)
```

Anexo 18. *'PI01_BreweryHub _v1.0.0.ipynb'*, código ejecutable por *Raspberry Pi Zero W* para la operación del sistema *BreweryHub*.

```
import RPi.GPIO as GPIO
import spidev
import subprocess
import time
from datetime import datetime
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.application import MIMEApplication
import os
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import logging
import configparser
from influxdb import InfluxDBClient
from datetime import datetime

# Initialize config.ini
config = configparser.ConfigParser()
config.read('/home/pi/breweryhub/config.ini')

SLEEP_TIME = config.getint('General', 'SLEEP_TIME')
filepath = config.get('General', 'filepath')
COUNTER_FILE = config.get('General', 'COUNTER_FILE')
records = config.getfloat('General', 'Records')
records_b = config.getfloat('General', 'Records_B')
roll_back=config.getint('General', 'Roll_Back')
TEMP_UPPER_LIMIT_1 = config.getfloat('Alerts', 'TEMP_UPPER_LIMIT_1')
TEMP_LOWER_LIMIT_1 = config.getfloat('Alerts', 'TEMP_LOWER_LIMIT_1')
TEMP_UPPER_LIMIT_2 = config.getfloat('Alerts', 'TEMP_UPPER_LIMIT_2')
TEMP_LOWER_LIMIT_2 = config.getfloat('Alerts', 'TEMP_LOWER_LIMIT_2')
TEMP_COMPENSATION_1 = config.getfloat('Temperature', 'Compensation_1')
TEMP_COMPENSATION_2 = config.getfloat('Temperature', 'Compensation_2')

# Local InfluxDB settings
```

```
DB_NAME = config.get('Influx', 'DB_NAME') # 'breweryhub'
DB_HOST = config.get('Influx', 'DB_HOST')
DB_PORT = config.get('Influx', 'DB_PORT')
DB_USER = config.get('Influx', 'DB_USER')
DB_PASSWORD = config.get('Influx', 'DB_PASSWORD')

# InfluxDB Cloud settings
TOKEN = config.get('Influx', 'TOKEN')
ORG = config.get('Influx', 'ORG')
CLOUD_URL = config.get('Influx', 'CLOUD_URL')
BUCKET = config.get('Influx', 'BUCKET')

# WiFi credentials & SMTP email server credentials
ssid = config.get('WiFi', 'SSID')
passwordwifi = config.get('WiFi', 'PASSWORD')
ssid_b = config.get('WiFi', 'SSID_B')
passwordwifi_b = config.get('WiFi', 'PASSWORD_B')
smtp_server = config.get('SMTP', 'SERVER')
smtp_port = config.getint('SMTP', 'PORT')  # Use getint for integers
smtp_username = config.get('SMTP', 'USERNAME')
smtp_password = config.get('SMTP', 'PASSWORD')
sender = config.get('SMTP', 'SENDER')
recipients = config.get('SMTP', 'RECIPIENTS').split(', ')
recipients_b = config.get('SMTP', 'RECIPIENTS_B').split(', ')

# Log & Counters
logging.basicConfig(filename='breweryhub.log', level=logging.DEBUG,
                    format='%(asctime)s [%(levelname)s] - %(message)s')
sensor_data = 0  # Count of sensor data logs
out_of_range_count = [0, 0]  # Maintaining separate counters
last_temperature_1 = None  # To store the previous temperature reading
last_temperature_2 = None  # To store the previous temperature reading

class MAX6675:
    def __init__(self, spi, cs_pin):
        self.spi = spi
        self.cs = cs_pin

    def read_temp(self):
        GPIO.output(self.cs, GPIO.LOW)
        data = self.spi.readbytes(2)
        GPIO.output(self.cs, GPIO.HIGH)
        temp_data = (data[0] << 8) | data[1]
        if temp_data & 0x4:
            return None
        temp_data >>= 3
        return temp_data * 0.25

# Set GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()  # Clean up any previously set conFigurations

# SPI setup
spi0 = spidev.SpiDev()
spi0.open(0, 0)
spi0.max_speed_hz = 1000000

spi1 = spidev.SpiDev()
```

```
spi1.open(1, 0)
spi1.max_speed_hz = 1000000

# GPIO setup
GPIO.setup(8, GPIO.OUT)  # CS0 for SPI0
GPIO.setup(18, GPIO.OUT)  # CS0 for SPI1

thermocouple_1 = MAX6675(spi0, 8)
thermocouple_2 = MAX6675(spi1, 18)

# Check if WiFi is connected
def is_wifi_connected():
    try:
        status = subprocess.run(["iwgetid", "-r"], text=True, capture_output=True).stdout.strip()
        if status:
            return True
        return False
    except Exception as e:
        print('An error occurred checking WiFi status:', e)
        logging.error(f'Error checking WiFi status: {e}')
        return False

# Connect to WiFi
def attempt_wifi_connection(ssid, password):
    if is_wifi_connected():
        print('Already connected to WiFi. Skipping connection attempt.')
        logging.info('Already connected to WiFi. Skipping connection attempt.')
        return True
    try:
        subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "disconnect"])
        subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "remove_network", "all"])
        add_network_response = subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "add_network"], text=True, capture_output=True)
        subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "set_network", add_network_response.stdout.strip(), "ssid", f'"{ssid}"'])
        subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "set_network", add_network_response.stdout.strip(), "psk", f'"{password}"'])
        subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "enable_network", add_network_response.stdout.strip()])
        subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "save_config"])
        subprocess.run(["sudo", "wpa_cli", "-i", "wlan0", "reconnect"])
        print('Network connection attempt successful!')
        logging.info('Network connection attempt successful!')
        return True
    except Exception as e:
        print('An error occurred:', e)
        logging.error(f'Error connecting to WiFi: {e}')
        return False

def connectWifi(ssid, password, ssid_b, password_b):
    # Try the original credentials first
    if attempt_wifi_connection(ssid, password):
        return True
    else:
        print("Trying alternate credentials...")
        return attempt_wifi_connection(ssid_b, password_b)  # Try the alternate credentials

# Get date and time
def obtain_Date_Time():
    current_time = time.localtime()
    date_string = time.strftime("%Y-%m-%d", current_time)
    time_string = time.strftime("%H:%M:%S", current_time)
```

```
    print(f'DateTime: {date_string} {time_string}')
    return date_string, time_string

# Obtain sensor readings
def obtainReadings():
    try:
        temperature_1 = thermocouple_1.read_temp()
        if temperature_1 is not None:
            temperature_1 += TEMP_COMPENSATION_1
        else:
            logging.error('Error reading from thermocouple_1')

        temperature_2 = thermocouple_2.read_temp()
        if temperature_2 is not None:
            temperature_2 += TEMP_COMPENSATION_2
        else:
            logging.error('Error reading from thermocouple_2')

        logging.debug(f'Retrieved temperature reading: {temperature_1}, {temperature_2}')
        return temperature_1, temperature_2
    except Exception as e:
        logging.error(f'Error obtaining temperature reading: {e}')
        return None, None

def save_counter_to_file():
    with open(COUNTER_FILE, 'w') as file:
        file.write(str(sensor_data))

def get_counter_from_file():
    global sensor_data
    if os.path.exists(COUNTER_FILE):
        with open(COUNTER_FILE, 'r') as file:
            sensor_data = int(file.read())
    else:
        sensor_data = 0

def is_connected():
    response = os.system("ping -c 1 8.8.8.8 > /dev/null 2>&1")
    return response == 0

def restart_wifi():
    subprocess.run(["sudo", "service", "networking", "restart"])
    time.sleep(10)  # Give some time for the WiFi to restart and attempt to reconnect.

buffered_data = []

def data_logging(date_string, time_string, temperature_1, temperature_2):
    global sensor_data
    current_data = {
        "date": date_string,
        "time": time_string,
        "temp1": temperature_1,
        "temp2": temperature_2,
        "counter": sensor_data
    }
    process_data(current_data)

def process_data(data):
```

```
    global sensor_data
    # Write to CSV first
    if not os.path.exists(filepath):
        with open(filepath, "w") as file:
            file.write("Reading,Date,Time,Temperature_1,Temperature_2\n")

    with open(filepath, "a") as file:
        file.write("{},{},{},{},{}\n".format(data["counter"], data["date"], data["time"], data["temp1"], data["temp2"]))

    # Check WiFi connection only after writing to CSV
    if is_connected():
        while buffered_data:
            write_to_influx(buffered_data.pop(0))
        write_to_influx(data)
    else:
        buffered_data.append(data)
        print(f"Data buffered. Buffer size: {len(buffered_data)}")

    sensor_data += 1
    print(f'Counter: {sensor_data}, Tank 1: {data["temp1"]}°C, Tank 2: {data["temp2"]}°C')
    save_counter_to_file()

    if sensor_data % records == 0:  # Sending email every n logs
        #plot_data()
        send_email(data["date"], data["time"])
    if sensor_data % records_b == 0:  # Condition to send status email
        #plot_data()
        send_status(data["date"], data["time"], data["temp1"], data["temp2"])

def cloud_influx_cli_write(url, token, org, bucket, body):
    headers = {"Authorization": f"Token {token}"}
    cmd = ["curl", "-X", "POST", f"{url}/api/v2/write?org={org}&bucket={bucket}&precision=ns", "-H", f"Authorization: Token {token}", "--data-raw", body]
    subprocess.run(cmd, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, check=True)

def write_to_influx(data):
    #local_client = InfluxDBClient(DB_HOST, DB_PORT, DB_USER, DB_PASSWORD, DB_NAME)
    datetime_str = data["date"] + " " + data["time"]
    datetime_obj = datetime.strptime(datetime_str, '%Y-%m-%d %H:%M:%S')
    timestamp_ns = int(time.mktime(datetime_obj.timetuple()) * 1e9)

    # Write to Local InfluxDB
    json_body = [
        {
            "measurement": "temperature_data",
            "tags": {
                "source": "sensor_data"
            },
            "time": timestamp_ns,
            "fields": {
                "temp1": data["temp1"],
                "temp2": data["temp2"]
            }
        }
    ]
    # Write to InfluxDB Cloud
    cloud_body = f"temperature_data,source=sensor_data temp1={data['temp1']},temp2={data['temp2']} {timestamp_ns}"
    cloud_influx_cli_write(CLOUD_URL, TOKEN, ORG, BUCKET, cloud_body)
```

```
    print('Influx updated.')

def plot_data(row_threshold=3000):
    global TEMP_UPPER_LIMIT_1, TEMP_LOWER_LIMIT_1, TEMP_UPPER_LIMIT_2, TEMP_LOWER_LIMIT_2
    df = pd.read_csv(filepath)
    df['DateTime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'])
    row_count = len(df)
    if row_count < row_threshold:
        x_min = df['DateTime'].iloc[0]
    # If x_min is not provided as an argument, calculate it from the DataFrame
    if row_count < row_threshold:
        x_min = df['DateTime'].iloc[0]
    else:
        x_min = df['DateTime'].iloc[-row_threshold]
    x_max = df['DateTime'].iloc[-1]
    fig = plt.figure(figsize=(12, 7), facecolor='lightgrey')  # Set figure's background color
    ax = fig.add_subplot(1, 1, 1)
    ax.set_facecolor('lightgrey')  # Set the plot's background color
    ax.set_axisbelow(True)  # To ensure the grid lines are below the plot lines
    ax.yaxis.grid(color='white', linestyle='dashed')  # Set y-axis grid lines to white for contrast
    ax.xaxis.grid(color='white', linestyle='dashed')  # Set x-axis grid lines to white for contrast
    plt.plot(df['DateTime'], df['Temperature_1'], color='blue', marker='.', linestyle=‘, alpha=0.25, markersize=0.8)
    plt.plot(df['DateTime'], df['Temperature_2'], color='green', marker='.', linestyle=‘, alpha=0.25, markersize=0.8)
    rolling_window_size = round(60 / (SLEEP_TIME / 40))
    ewma_avg_temp1 = df['Temperature_1'].ewm(span=rolling_window_size).mean()
    ewma_avg_temp2 = df['Temperature_2'].ewm(span=rolling_window_size).mean()
    plt.plot(df['DateTime'], ewma_avg_temp1, label='Tanque F04', color='blue', linewidth=1.5, alpha=0.7)
    plt.plot(df['DateTime'], ewma_avg_temp2, label='Tanque F03', color='green', linewidth=1.5, alpha=0.7)
    plt.axhline(y=TEMP_UPPER_LIMIT_1, color='red', linestyle='--', alpha=0.65)
    plt.axhline(y=TEMP_LOWER_LIMIT_1, color='red', linestyle='--', alpha=0.65)
    plt.axhline(y=TEMP_UPPER_LIMIT_2, color='orange', linestyle='--', alpha=0.65)
    plt.axhline(y=TEMP_LOWER_LIMIT_2, color='orange', linestyle='--', alpha=0.65)
    y_min = min(max(df['Temperature_1'].quantile(0.05), df['Temperature_2'].quantile(0.05)), TEMP_LOWER_LIMIT_1, TEMP_LOWER_LIMIT_2) - 0.25
    y_max = max(df['Temperature_1'].quantile(0.95), df['Temperature_2'].quantile(0.95), TEMP_UPPER_LIMIT_1, TEMP_UPPER_LIMIT_2) + 0.25
    y_min = int(y_min // 2) * 2  # Round down to nearest even number
    y_max = int(-(-y_max // 2) * 2)  # Round up to nearest even number
    plt.ylim(y_min, y_max)  # Set y-axis limits
    plt.yticks(np.arange(y_min, y_max + 1, 2))  # Set y-ticks at 2-degree intervals
    plt.xlim(x_min, x_max)
    time_diff = (df['DateTime'].iloc[-1] - df['DateTime'].iloc[0]) / 12
    interval_x = int(time_diff.total_seconds() / 3600)
    interval_x = max(1, interval_x)  # Ensure that the interval is at least 1 hour
    plt.gca().xaxis.set_major_locator(mdates.HourLocator(interval=interval_x))
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d/%m %H:%M'))
    plt.gcf().autofmt_xdate()
    plt.ylabel('Temperatura (°C)', fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.legend(loc='center left', bbox_to_anchor=(0.7, 0.8), fontsize=12)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.gca().spines['top'].set_visible(False)
    plt.gca().spines['right'].set_visible(False)
    plt.savefig("temp_plot.png", facecolor=fig.get_facecolor())  # Save with the figure's background color

# Send email Readings
def send_email(date_string, time_string):
    plot_data()
    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.starttls()
```

```
    server.login(smtp_username, smtp_password)
    msg = MIMEMultipart()
    msg['From'] = sender
    msg['To'] = ', '.join(recipients)
    msg['Subject'] = f'BreweryHub No.1: {date_string} {time_string}'
    body = ('<p style="font-size: 11px;">Se adjuntan los registros de temperatura.</p>'
        '<p style="font-size: 12px;"><b>BreweryHub v1.0.0</b></p>'
        '<p style="font-size: 11px;">sebastian.hermosillo@iteso.mx - requiemforabeer@gmail.com<br></p>'
        '<p style="font-size: 11px;"><b>ITESO, Universidad Jesuita de Guadalajara</b> en colaboración con <b>Cerveza Loba</b></p>')
    msg.attach(MIMEText(body, 'html'))
    #current_timestamp = datetime.datetime.now().strftime('%Y%m%d_%H%M%S')  # Format the current datetime
    current_timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    # Attach the CSV
    with open(filepath, 'r') as f:
      attachment = MIMEApplication(f.read())
      attachment.add_header('Content-Disposition', 'attachment', filename=f'{current_timestamp}_temperature_readings.csv')
      msg.attach(attachment)
    # Attach the plot
    try:
      with open("temp_plot.png", 'rb') as f:
        image = MIMEImage(f.read())
        image.add_header('Content-Disposition', 'attachment', filename=f"{current_timestamp}_temp_plot.png")
        msg.attach(image)
    except FileNotFoundError:
      print("temp_plot.png not found.")
    server.send_message(msg)
    print('Readings Email sent.')
    # Delete the plot image if it exists
    try:
      os.remove("temp_plot.png")
    except FileNotFoundError:
      print("temp_plot.png not found, so couldn't delete.")

def send_status(date_string, time_string, temperature_1, temperature_2):
  plot_data()
  with smtplib.SMTP(smtp_server, smtp_port) as server:
    server.starttls()
    server.login(smtp_username, smtp_password)
    msg = MIMEMultipart()
    msg['From'] = sender
    msg['To'] = ', '.join(recipients_b)  # Note: Using Recipients_b list
    msg['Subject'] = f'BreweryHub No.1 Status: OK {date_string} {time_string}'
    body = (f'<p style="font-size: 11px;">#{sensor_data} - Tanque F04: {temperature_1} °C, Tanque F03: {temperature_2} °C</p><br>'
        '<p style="font-size: 12px;"><b>BreweryHub v1.0.0</b></p>'
        '<p style="font-size: 11px;">sebastian.hermosillo@iteso.mx - requiemforabeer@gmail.com<br></p>'
        '<p style="font-size: 11px;"><b>ITESO, Universidad Jesuita de Guadalajara</b> en colaboración con <b>Cerveza Loba</b></p>')
    msg.attach(MIMEText(body, 'html'))
    current_timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    try:      # Attach the plot
      with open("temp_plot.png", 'rb') as f:
        image = MIMEImage(f.read())
        image.add_header('Content-Disposition', 'attachment', filename=f"{current_timestamp}_temp_plot.png")
        msg.attach(image)
    except FileNotFoundError:
      print("temp_plot.png not found.")
    server.send_message(msg)
    print('Status Email sent.')
```

```
def send_alert(current_temperature, date_string, time_string, sensor_data, tank_number=1):
    plot_data()
    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.starttls()
        server.login(smtp_username, smtp_password)
        msg = MIMEMultipart()
        msg['From'] = sender
        msg['To'] = ', '.join(recipients)
        # Determine the alert type for the subject and body of the email
        if current_temperature > float(config[f'Alerts'][f'TEMP_UPPER_LIMIT_{tank_number}']):
            alert_type = "Alta"
        else:
            alert_type = "Baja"
        msg['Subject'] = f'BreweryHub No.1 Alerta Límite Tanque {tank_number}: {alert_type} Temperatura {date_string} {time_string}'
        body = (f'<p style="font-size: 11px;">¡Atención ({sensor_data})! El registro de temperatura de {current_temperature}°C, en Tanque {tank_number},
está fuera del rango establecido.</p>'
                '<p style="font-size: 12px;"><b>BreweryHub v1.0.0</b></p>'
                '<p style="font-size: 11px;">sebastian.hermosillo@iteso.mx - requiemforabeer@gmail.com<br></p>'
                '<p style="font-size: 11px;"><b>ITESO, Universidad Jesuita de Guadalajara</b> en colaboración con <b>Cerveza Loba</b></p>')
        msg.attach(MIMEText(body, 'html'))
        current_timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        try:        # Attach the plot
            with open("temp_plot.png", 'rb') as f:
                image = MIMEImage(f.read())
                image.add_header('Content-Disposition', 'attachment', filename=f"{current_timestamp}_temp_plot.png")
                msg.attach(image)
        except FileNotFoundError:
            print("temp_plot.png not found.")
        server.send_message(msg)
        print(f'Alerta {alert_type} Tanque {tank_number}. Email sent.')

def send_rate_alert(current_temperature, rate, tank_number=1):
    """Send an alert email when rate of temperature change is too high."""
    plot_data()
    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.starttls()
        server.login(smtp_username, smtp_password)
        msg = MIMEMultipart()
        msg['From'] = sender
        msg['To'] = ', '.join(recipients)

        msg['Subject'] = f'BreweryHub No.1 Alerta Cambio Tanque {tank_number}: Cambio Rápido Detectado'
        body = (f'¡Atención! El registro de temperatura de {current_temperature}°C '
                f'en Tanque {tank_number} cambió a razón de {rate:.2f}°C por minuto, por encima del límite establecido.')
        msg.attach(MIMEText(body, 'plain'))
        current_timestamp = datetime.now().strftime('%Y%m%d_%H%M%S') # Format the current datetime
        try:        # Attach the plot
            with open("temp_plot.png", 'rb') as f:
                image = MIMEImage(f.read())
                image.add_header('Content-Disposition', 'attachment', filename=f"{current_timestamp}_temp_plot.png")
                msg.attach(image)
        except FileNotFoundError:
            print("temp_plot.png not found.")
        server.send_message(msg)
        print(f'Tanque {tank_number} Rapid temperature change alert email sent.')

def check_temperature_limits_and_rate(date_string, time_string, temp, last_temp, tank_number):
    global out_of_range_count
```

```
    # Fetch the right temperature limits depending on the tank_number
    temp_upper_limit = float(config[f'Alerts'][f'TEMP_UPPER_LIMIT_{tank_number}'])
    temp_lower_limit = float(config[f'Alerts'][f'TEMP_LOWER_LIMIT_{tank_number}'])
    # Check temperature limits
    if temp > temp_upper_limit or temp < temp_lower_limit:
        out_of_range_count[tank_number-1] += 1
    else:
        out_of_range_count[tank_number-1] = 0
    if out_of_range_count[tank_number-1] >= int(config['Alerts']['CONSECUTIVE_LIMIT']):
        send_alert(temp, date_string, time_string, sensor_data, 1)
        out_of_range_count[tank_number-1] = 0  # Reset the counter after sending an alert

    # Check rate of change if last_temperature exists
    if last_temp is not None:
        rate_of_change = abs(temp - last_temp) / (SLEEP_TIME / 60)
        if rate_of_change > float(config['Alerts']['RATE_LIMIT']):
            send_rate_alert(temp, rate_of_change, tank_number)
    return temp  # returning the current temperature to be saved as last temperature

# Main function
def main():
    global last_temperature_1, last_temperature_2
    get_counter_from_file()
    connectWifi(ssid, passwordwifi, ssid_b, passwordwifi_b)  # Connect to WiFi
    connection_attempts = 0  # To keep a count of connection attempts

    # Main loop
    while True:
        date_string, time_string = obtain_Date_Time()
        temperature_1, temperature_2 = obtainReadings()
        data_logging(date_string, time_string, temperature_1, temperature_2)
        last_temperature_1 = check_temperature_limits_and_rate(date_string, time_string, temperature_1, last_temperature_1, 1)
        last_temperature_2 = check_temperature_limits_and_rate(date_string, time_string, temperature_2, last_temperature_2, 2)

        # Attempt to reconnect every 10 iterations, for instance
        connection_attempts += 1
        if connection_attempts >= 10:
            if not is_connected():
                connectWifi(ssid, passwordwifi, ssid_b, passwordwifi_b)
            connection_attempts = 0

        now = datetime.now()
        minutes_to_next_hour = 60 - now.minute
        seconds_to_next_hour = minutes_to_next_hour * 60 - now.second
        # Check if time to next reading is less than SLEEP_TIME
        if seconds_to_next_hour < SLEEP_TIME:
            print('Data saved. Sleeping for {} seconds!'.format(seconds_to_next_hour))
            time.sleep(seconds_to_next_hour)  # Sleep till the next hour
        else:
            print('Data saved. Sleeping for {} seconds!'.format(SLEEP_TIME))
            time.sleep(SLEEP_TIME)  # Regular sleep time

# Run the program
if __name__ == "__main__":
    try:
        main()
    finally:
        GPIO.cleanup()
```

Anexo 19. Resumen de costos de componentes de *Hardware*.

Tabla 14. Inversión total en componentes de *Hardware* del sistema *BreweryHub*.

Componente	Cantidad	Precio por unidad	Monto
Termopar tipo K	15 m	80 MXN/m	1,200 MXN
Módulo *MAX6675*	4 pz	89 MXN/pz	356 MXN
Raspberry Pi Zero W	2 pz	558 MXN/pz	1,116 MXN
Caja de protección	2 pz	267 MXN/pz	534 MXN
Tarjeta *microSD*	2 pz	115 MXN/pz	230 MXN
Powerbank	4 pz	339 MXN/pz	1,356 MXN

Costo total del proyecto: 4.792 MXN. Costo total por sistema *BreweryHub*: 2,396 MXN (aproximadamente 140 *USD* al 24/11/23).