



escuela
británica de
artes creativas
y tecnología

PROYECTO FINAL

“Enfoque para Predecir Impurezas en el Concentrado de Mineral”

Carrera:

**CIENTÍFICO DE
DATOS**

Alumna:
Georgina Casillas Rosano

Enero2025

1. Resumen

Este trabajo está dedicado al estudio de modelos de regresión lineal aplicados a un proceso de producción de una industria minera para realizar predicciones del porcentaje de impurezas de sílice al final de proceso para facilitar la toma de decisiones.

Las predicciones se realizaron mediante el entrenamiento de modelos de regresión utilizando técnicas estadísticas y de aprendizaje automático. Los modelos se evaluaron mediante pruebas y ajustes para asegurar un buen ajuste y precisión en las predicciones.

Se espera obtener un modelo de regresión con alta precisión y capacidad de generalización que faciliten el control y optimización del proceso de concentración de hierro.

Los distintos modelos se evaluaron usando el coeficiente de determinación R^2 para medir qué tan bien el modelo explica la variación de los datos. Se determinaron las métricas MSE para medir el error promedio al cuadrado entre los valores predichos y los observados. Un valor bajo de MSE indica un buen ajuste del modelo y la métrica MAE para medir el error promedio absoluto entre las predicciones y los valores reales

Los resultados muestran que el modelo XGB Regressor, presenta valores de R^2 , MSE, y MAE similares para los conjuntos de entrenamiento y prueba, lo que indica que el modelo no está sobreajustado ni subajustado. Los valores del MSE y MAE son pequeños, lo cual indica que las predicciones están cerca de los valores reales.

2. Introducción

Los modelos de regresión en ciencia de datos son técnicas estadísticas y de aprendizaje automático que se utilizan para predecir un valor numérico a partir de una o más variables independientes (también llamadas características o predictores). Estos modelos buscan establecer la relación entre las variables independientes y la variable dependiente (la que se quiere predecir).

Se probaron diferentes modelos de regresión, comenzando con la **regresión lineal** que asumen una relación lineal entre las variables independientes y la variable dependiente.

Seguido por modelos de combinaciones múltiples, que permiten manejar relaciones complejas y no lineales como el modelo **Random Forest Regressor** el cual entrena árboles de decisión con un subconjunto aleatorio de los datos y las predicciones de todos los árboles se promedian para obtener la predicción final. También se probó el modelo **Gradient Boosting Regression Trees (GBRT)** utiliza árboles de decisión, pero los construye de manera secuencial. Cada nuevo árbol se entrena para corregir los errores del modelo anterior, mejorando gradualmente la precisión de las predicciones. Es muy potente y puede capturar relaciones complejas, pero puede ser más propenso al sobreajuste si no se ajusta correctamente.

El propósito del estudio consistió en obtener un modelo de regresión con alta precisión y capacidad de generalización con el cual predecir el porcentaje de impurezas al final del proceso de extracción de Hierro, mediante técnicas de aprendizaje automático.

Para seleccionar el mejor modelo, los distintos modelos se evaluaron usando métricas de evaluación como el **coeficiente de determinación** R^2 para medir qué tan bien el modelo explica la variación de los datos. Se determinaron las métricas MSE para medir el **error promedio al cuadrado** entre los valores predichos y los observados. Un valor bajo de MSE indica un buen ajuste del modelo y la métrica MAE para medir el **error promedio absoluto** entre las predicciones y los valores reales

3. Marco Teórico

Industria de extracción de Hierro

La creciente demanda internacional de acero ha impulsado el uso masivo de plantas de concentración de hierro mediante flotación inversa. Este proceso tiene como objetivo principal reducir al máximo los niveles de sílice (SiO_2) y azufre (S) en los concentrados finales. Además, el agotamiento de reservas de minerales de alta ley y la creciente complejidad mineralógica presentan desafíos adicionales, dificultando la liberación efectiva de partículas. Ante estas condiciones, la flotación inversa aniónica surge como una alternativa viable, ya que emplea reactivos biodegradables que han demostrado eficacia en la última etapa de extracción (Oyarzún, 2013).

La flotación es un proceso de separación que aprovecha las diferencias naturales o inducidas en las propiedades superficiales de los minerales. Estas propiedades determinan si las partículas se humedecen fácilmente con agua (hidrofílicas) o si la repelen (hidrofóbicas). En el caso de las partículas hidrofóbicas, estas se adhieren a burbujas de aire, permitiendo que floten y sean separadas del resto del material (Oyarzún, 2013).

Flotación

El mineral en una operación minera está compuesto por diversas especies, algunas de ellas de valor comercial (generalmente las menos abundantes) y otras de menor valor o sin valor relativo (ganga). El procesamiento de minerales sigue a la explotación minera con el objetivo de: preparar el mineral para la extracción del metal valioso (menas metálicas) o entregar un producto final (minerales industriales y carbón). Tras el procesamiento, el producto adquiere un valor de mercado y puede transarse. Por lo tanto, el procesamiento de minerales genera el primer producto comercializable o con precio y mercado de referencia (Oyarzún, 2013).



Figura 1 Liberación de partícula mineralizada (Oyarzún, 2013)

La **flotación de minerales** corresponde a la separación de especies mineralógicas por diferencia de mojabilidad o hidrofobicidad. Se dice que una partícula es

hidrofóbica cuando no tiene afinidad por el agua, a diferencia de partículas hidrofilicas, que sí tienen. Para lograr la separación es necesario contar con un sistema heterogéneo, que involucre más de una fase. En este caso se consideran las tres fases: sólido (mineral), líquido (agua) y gas (normalmente aire). Al introducir aire en forma de burbujas en una pulpa que contiene partículas de carácter hidrofílico e hidrofóbico, éstas últimas tienden a adherirse a las burbujas para así minimizar su contacto con la fase líquida (Oyarzún, 2013).

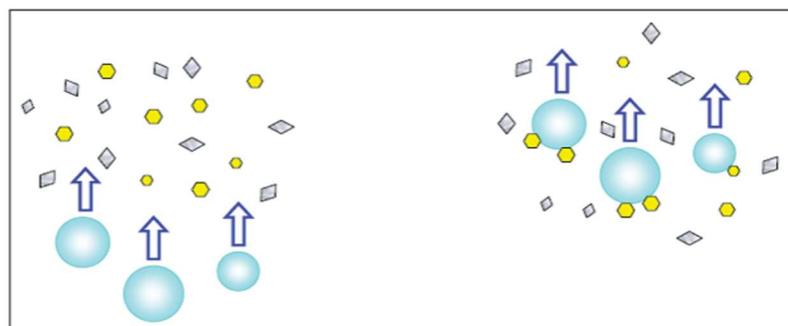


Figura 2 Recolección de partículas hidrofóbicas
(Oyarzún, 2013)

Para que ocurra la flotación deben coexistir dos zonas: colección y espuma. La pulpa entra en contacto con la fase gaseosa (burbujas) en la zona de colección, lugar donde se produce la unión partícula(s) burbuja. Estos agregados ascienden hasta llegar a una zona de espuma, la cual, producto de la adhesión selectiva en la zona de colección, contiene preferentemente partículas hidrofóbicas (Oyarzún, 2013).

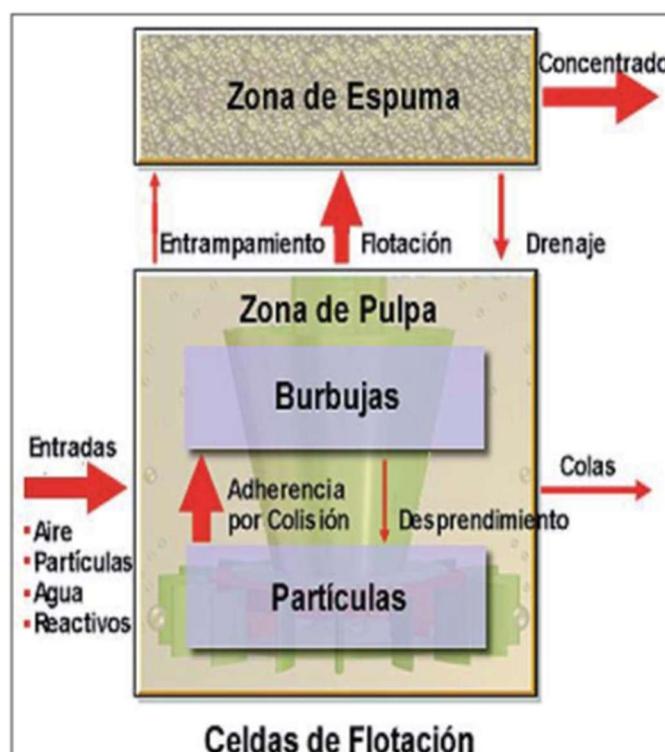


Figura 3 Zonas de colección y espuma en una celda de flotación (Oyarzún, 2013).

Los **índices de evaluación** del proceso de flotación son:

1. **Recuperación vs. Ley:** El objetivo de la concentración de minerales es maximizar tanto la recuperación como la ley del mineral de interés.
2. **Maximización de la ley:** Debido a que no todas las partículas se encuentran totalmente liberadas, la máxima ley obtenible se obtiene a una recuperación muy baja, la máxima ley se da a la mínima recuperación.
3. **Maximización de la recuperación:** Debido a la liberación, para recuperar el máximo del mineral de interés (100%) se tiene presencia de ganga en el concentrado, máxima recuperación la ley es mínima (asumiendo que no llega ganga totalmente liberada al concentrado) (Oyarzún, 2013).

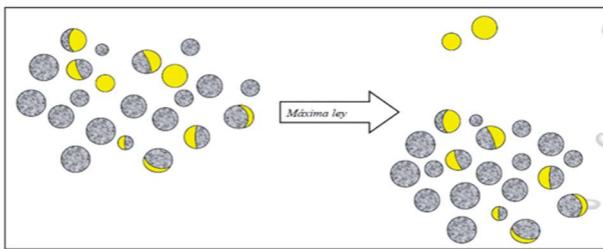


Figura 5 Esquema de maximización de la ley (Oyarzún, 2013)

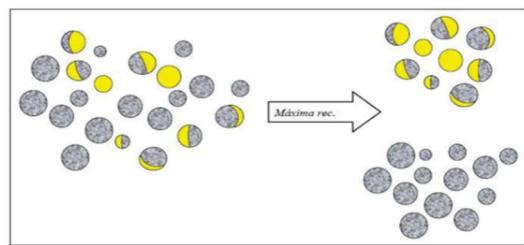


Figura 4 Esquema de maximización de la recuperación (Oyarzún, 2013)

La curva de recuperación y ley de concentrado grafica lo antes visto. Existe una relación inversa entre estas variables (Oyarzún, 2013).

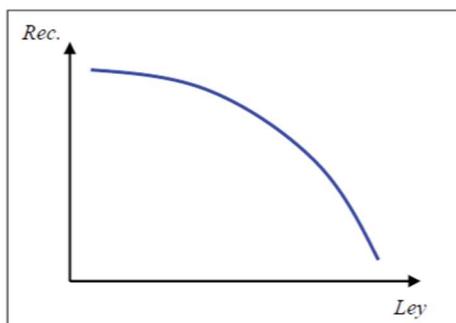


Figura 6 Esquema de recuperación vs ley (Oyarzún, 2013).

Como no se pueden maximizar ambos valores al mismo tiempo, se debe optar por una combinación de ellos que maximice el “óptimo económico” del proceso (Oyarzún, 2013).

Polaridad en minerales.

La mayoría de los minerales son naturalmente hidrofílicos, por lo que, para lograr una separación efectiva mediante flotación, su superficie debe volverse

hidrofóbica. Esto se consigue ajustando la química de la solución, regulada mediante reactivos específicos (Oyarzún, 2013).

Reactivos en flotación

Se clasifican en tres grupos principales:

1. **Colectores:** Su función principal es la de proporcionar propiedades hidrofóbicas a las superficies de los minerales
2. **Espumantes:** Se utilizan para generar una espuma estable y un tamaño de burbuja apropiado
3. **Modificadores:** Sirven para crear condiciones favorables en la superficie de los minerales, principalmente para el funcionamiento selectivo de los colectores (Oyarzún, 2013).

Celdas de flotación neumática.

Una celda de flotación neumática es un equipo que utiliza aire comprimido para separar material valorable de una pulpa. Este proceso se realiza mediante la creación de burbujas finas que se adhieren a las partículas minerales, lo que permite separarlas de la mezcla, mediante un sistema Venturi o tecnología especializada para generar burbujas finas (Oyarzún, 2013).

La flotación neumática es un método que puede utilizarse para separar minerales de cobre, hierro y molibdeno. En comparación con las máquinas de flotación mecánicas, la recuperación de los concentrados puede aumentar de 1% a 4% (Oyarzún, 2013).

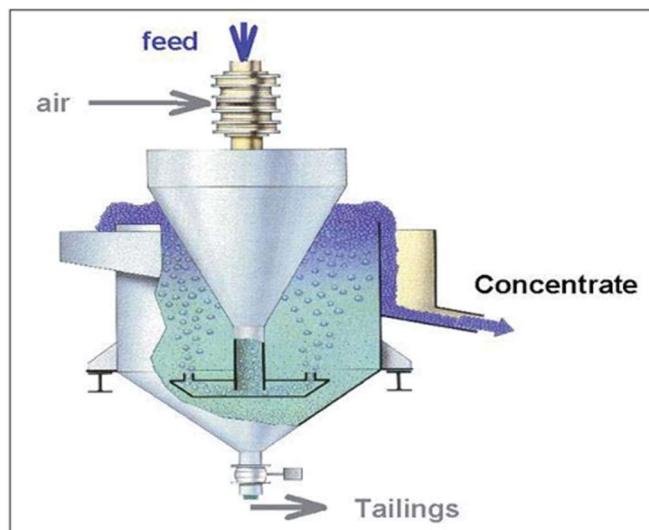


Figura 7 Vista en corte de una celda neumática modelo PNEUFLOT M-FV50NS (Oyarzún, 2013).

Es en esta celda de flotación en la cual se adicionan reactivos como aminas. Actúan como **colectores**, adhiriéndose a superficies minerales cargadas

negativamente y generan **espuma** para facilitar la separación de los minerales flotados (Oyarzún, 2013).

Este proceso se realiza a un pH alto (superior a 8.0), donde las aminas favorecen la colección de sílice en lugar de minerales de hierro debido a las diferencias en sus cargas superficiales, lo cual permite separar sílice de minerales de hierro, determinadas por el potencial zeta de los minerales (Oyarzún, 2013).

Ventajas flotación neumática:

- Mayor selectividad: Al operar con burbujas más finas y reactivos efectivos, mejora la selectividad en la separación de minerales.
- Impacto ambiental reducido: Al aumentar la recuperación, se reduce el material desecharido en los relaves, disminuyendo el impacto ambiental.
- Optimización del proceso: Menos energía requerida en comparación con celdas mecánicas (Oyarzún, 2013).

Modelos de Regresión

Los modelos de regresión en ciencia de datos son métodos estadísticos y de aprendizaje automático diseñados para **predecir un valor numérico** basado en una o más variables independientes (también conocidas como características o predictores). Su objetivo es modelar la relación entre las variables independientes y la variable dependiente (la que se desea predecir) (Orallo, 2004).

Tipos de modelos de regresión

Regresión Lineal

La **regresión** es una técnica estadística que consiste en calcular dicha similitud en forma de función matemática. Hablamos de modelo de regresión cuando la variable de respuesta y las variables explicativas son todas ellas **cuantitativas** (Montero, 2016). Si sólo disponemos de una variable explicativa hablamos de *regresión simple*, mientras que si disponemos de varias variables explicativas se trata de un problema de *regresión múltiple* (Orallo, 2004).

La **regresión lineal** supone que la relación entre dos variables tiene una forma lineal (o linealizable mediante alguna transformación de las variables). La regresión lineal tiene una versión “simple” que empareja dos variables, pero esta suele ser insuficiente para entender fenómenos mínimamente complejos en la que influyen más de dos variables (Orallo, 2004). El modelo se representa de la siguiente forma:

$$y = a + bX + \varepsilon$$

Donde:

- a = intersección en el eje y
- b = pendiente
- ε = error aleatorio

En el modelo de **regresión lineal múltiple** este modelo estadístico examina la relación entre una variable dependiente y dos ó más variables independientes.

$$y = a + b_1X_1 + b_2X_2 + \dots + b_nX_n + \varepsilon$$

Donde:

a = intersección eje y
 b_1, b_2, b_n , = razones de cambio de X_i
 ε = error aleatorio

Cuando las razones de cambio son positivas (b), influyen positivamente en y viceversa (Montero, 2016).

Métricas para evaluar rendimiento de regresiones lineal y múltiple

1. Error cuadrático medio (MSE) .

Medida de error común en modelos de regresión. Se usa para penalizar errores grandes.

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

Donde:

n = total de observaciones
 y_i = valor verdadero
 \hat{y}_i = valor predicho por el modelo. error aleatorio

2. Error absoluto medio (MAE)

Se usa en errores que se distribuyen uniformemente sin importar si son grandes o pequeños.

$$MAE = \frac{\sum|y_i - \hat{y}_i|}{n}$$

Donde:

n = total de observaciones
 y_i = valor verdadero
 \hat{y}_i = valor predicho por el modelo. error aleatorio

3. R² Coeficiente de determinación

Mediada de cuanta variabilidad de la variable dependiente es explicado por el modelo.

$$R^2 = 1 - \frac{SSR}{SST}$$

Donde:

SSR = suma de cuadrados de residuales (errores).

SST = suma de cuadrados de las diferencias ente el valor verdadero y el valor real.

Random Forest Regressor

El modelo de ensamble **Random Forest** combina múltiples técnicas de aprendizaje automático, como árboles de decisión y **Bagging**, para aumentar la precisión, robustez y estabilidad de las predicciones, superando así el desempeño de un único árbol de decisión (Mrabet, 2022).

Una característica distintiva de Random Forest es la selección aleatoria de variables en cada división del árbol, lo que incrementa la diversidad entre los modelos y reduce el riesgo de sobreajuste (Mrabet, 2022).

La técnica de **Bagging** consiste en entrenar varios modelos idénticos utilizando subconjuntos aleatorios de los datos originales. En el caso de problemas de regresión, la predicción final se obtiene promediando los resultados de estos modelos (Mrabet, 2022).

Además, Random Forest calcula la importancia de cada característica en función de su contribución al proceso de decisión, permitiendo identificar y eliminar aquellas características que tienen menor relevancia (Mrabet, 2022).

El modelo de **Bosque Aleatorio (Random Forest)** es un enfoque de conjunto que utiliza múltiples árboles de decisión independientes y no correlacionados, denotados como $F = \{t_1, t_2, \dots, t_t\}$. Estos árboles independientes se generan introduciendo aleatoriedad en su construcción, lo que permite al modelo lograr una generalización precisa. Este enfoque se basa en la técnica de **Bagging** (Bootstrap Aggregating), que combina muestreo aleatorio con reemplazo (**bootstrap**) y agregación de predicciones. Dado un conjunto de entrenamiento (Mrabet, 2022):

$$S = \{X^m, Y^m\}_{(M=1)}^m$$

Donde:

X: representa el espacio de características de entrada, que puede incluir parámetros operativos como flujos de aire, niveles de columnas de flotación, concentraciones de sílice, entre otros.

Y: es un espacio multidimensional continuo que incluye, por ejemplo, el porcentaje final de sílice.

M: es el número de muestras en el conjunto de datos.

Bootstrap crea subconjuntos de entrenamiento S_t seleccionados aleatoriamente de S . Cada subconjunto contiene la misma cantidad de instancias que el conjunto original, pero aproximadamente 1/3 de las muestras se eliminan, mientras que otro 1/3 se duplican debido al muestreo con reemplazo (Mrabet, 2022).

Para cada árbol, se entrena un modelo independiente utilizando un subconjunto S_t . Las predicciones finales se obtienen **promediando los resultados** de todos los árboles para problemas de **regresión** o mediante votación para problemas de clasificación (Mrabet, 2022).

Modelado probabilístico

Suponiendo que las variables de salida Y siguen una distribución gaussiana multivariada con media μ y covarianza Σ , la regresión posterior de un árbol aleatorio t_t se modela como (Mrabet, 2022):

$$P(y|x, P_t) = N_t(y|\mu_t, \sigma_t)$$

Dónde:

P_t = es una partición generada por el árbol t_t

N_t = gaussiana multivariada definida por media μ_t y covarianza Σ_t que mapea el espacio de salida Y a partir de los datos de entrenamiento.

El objetivo del entrenamiento de los árboles es minimizar la incertidumbre asociada con la predicción posterior, seleccionando una función de división f adecuada en cada nodo N_t . Esto permite dividir el subconjunto S_l en dos nuevos subconjuntos, S_l^{right} y S_l^{left} , reduciendo la incertidumbre en las predicciones (Mrabet, 2022).

Un ejemplo de función f incluye la **ganancia de información** y el **índice de Gini**, ampliamente utilizados para evaluar divisiones en problemas de clasificación. En el caso de tareas de regresión, se emplea la **entropía diferencial no ponderada** (SE), una versión continua de la entropía de Shannon, como una función óptima para calcular la ganancia de información (Mrabet, 2022).

La entropía diferencial (SE) se define como (Mrabet, 2022):

$$f(S_l) = \int_{(y \in Y)} \sum_{i=1}^n P(y|S_l) \log(P(y|S_l)) dY$$

Donde:

S_l : subconjunto de datos en el nodo N_t

$P(y|S_l)$: probabilidad condicional de la salida y dada el subconjunto S_l

Y : espacio de salida continuo.

n : número de instancias en S_l

Esta función mide la incertidumbre en la distribución de la salida y para un subconjunto específico S_l . La función SE, se selecciona cuando proporciona resultados satisfactorios en términos de reducción del error de predicción (Mrabet, 2022).

La **ganancia de información** en una división se calcula restando la entropía total ponderada de los subconjuntos generados por una función de división f , ayudando a reducir la incertidumbre y mejorar la precisión del modelo (Mrabet, 2022).

Dado que el modelo posterior se basa en una distribución Gaussiana multivariada, la función f se puede reescribir como (Mrabet, 2022):

$$f(S_l) = \frac{1}{2} \log ((\pi \exp)^{D'} |\Sigma(S_l)|)$$

Dónde:

$\Sigma(S_l)$: es la matriz de covarianza estimada a partir del subconjunto l .

D' : dimensionalidad del espacio de salida Y , es decir, el número de variables dependientes.

π : constante matemática (pi).

e : base del logaritmo natural.

$|\Sigma(S_l)|$: determinante de la matriz de covarianza.

Esta reformulación permite evaluar la incertidumbre en el espacio de salida Y basada en la estructura de la matriz de covarianza. La matriz $\Sigma(S_l)$ captura la dispersión y relaciones entre las variables dentro del subconjunto S_l , y el determinante $|\Sigma(S_l)|$ actúa como un indicador del volumen del espacio de predicción en la salida (Mrabet, 2022).

La introducción de esta representación gaussiana multivariada es clave para medir la ganancia de información en términos de reducción de la incertidumbre a nivel de cada nodo del árbol durante el entrenamiento (Mrabet, 2022).

Una vez que el subconjunto S_l en el nodo N_l se divide en dos subconjuntos, S_l^{right} y S_l^{left} , utilizando la función de división f , la ganancia de información (Δ) se calcula utilizando (Mrabet, 2022):

$$\Delta = f(S_l) - w_l f(S_l^{left}) - w_r f(S_l^{right})$$

Donde:

$f(S_l)$: es la medida de incertidumbre antes de realizar la división.

$f(S_l^{left})$ y $f(S_l^{right})$: son las medidas de incertidumbre después de dividir el nodo en sus dos subconjuntos.

w_l y w_r : son los pesos asignados a los subconjuntos izquierdo y derecho, respectivamente, definidos como:

$$\omega_l = \frac{|S_l|}{|S_l^{left}|}$$

$$\omega_r = \frac{|S_l|}{|S_l^{right}|}$$

Aquí:

$|S_l|$: es el número total de instancias en el nodo S_l .

$|S_l^{left}|$ y $|S_l^{right}|$: son las cantidades de instancias en los subconjuntos izquierdo y derecho tras la división.

Esta ecuación refleja cómo la incertidumbre general disminuye después de dividir el conjunto S_l en dos subconjuntos más homogéneos. El objetivo es maximizar Δ , ya que una mayor ganancia de información implica que la división seleccionada mejora significativamente la capacidad del modelo para distinguir entre las diferentes clases o valores de salida (Mrabet, 2022).

Fase de predicción

Una vez finalizada la fase de entrenamiento, en la **fase de predicción** cada nueva instancia se envía a través de los árboles del bosque. El posterior agregado se calcula como (Mrabet, 2022):

$$P(y|x) = \frac{1}{T} \sum_{t=1}^T P(y|x, P_t)$$

Donde:

T = es el número de árboles en el bosque

P_t = es la partición introducida por t_t

Dada cualquier **nueva instancia**, el modelo puede predecir la duración y ubicación de la falla correspondiente, maximizando una probabilidad posterior (Mrabet, 2022):

$$\hat{Y} = \text{argmax}_{y \in Y} P(y|x)$$

La ecuación corresponde al **cálculo de una predicción** mediante el criterio de **máxima probabilidad posterior**. En el contexto mencionado, el modelo toma una nueva instancia x y selecciona la clase y que maximiza la probabilidad posterior $P(y|x)$. Esto es común en modelos de clasificación probabilística, como el **Naive Bayes** o algunos enfoques basados en redes bayesianas (Mrabet, 2022).

En términos prácticos significa que:

1. **$P(y|x)$:** Es la probabilidad posterior de la clase y , dado un conjunto de características x . Esta probabilidad resume cuánto "confía" el modelo en que y es la clase correcta para la instancia x .
2. **argmax**: Indica que seleccionaremos el valor de y (dentro del conjunto de clases posibles Y) que maximice la probabilidad posterior $P(y|x)$.
3. **Predicción del modelo**: Dada una nueva observación x , el modelo determinará la clase \hat{Y} que tiene la probabilidad más alta ($y|x$) (Mrabet, 2022).

Aplicación a fallas

En este caso:

- x : Podría ser un conjunto de variables relacionadas con el sistema, como concentración, niveles de columnas, flujo de aire en las columnas, entre otros.
- y : Representa la clase de falla (la concentración final de impurezas de sílice).
- $P(y|x)$: La probabilidad de que ocurra una cierta falla y , dadas las condiciones actuales x .

El modelo elegiría la falla \hat{Y} que tiene la mayor probabilidad, permitiendo predecir su duración y ubicación.

Gradient Boosting Regression Trees (Mohan, 2011)

1. Fundamentos de GBRT

- **Base en Árboles Secuenciales:** GBRT no entrena múltiples árboles completos de manera independiente, como RF. En cambio, agrega secuencialmente **árboles pequeños ($d \approx 4$)** para corregir los errores de predicción de la iteración anterior (Mohan, 2011).
- **Optimización por Gradiente:** GBRT realiza una optimización basada en descenso de gradiente en el espacio de instancias. La predicción actual $T(x_i)T(x_{-i})T(x_i)$ se actualiza en cada iteración mediante (Mohan, 2011):

$$T(x_i) \leftarrow T(x_i) - \alpha \frac{\partial L}{\partial T(x_i)}$$

donde:

- $\alpha > 0$: Tasa de aprendizaje.
- L : Función de pérdida convexa, continua y diferenciable.
- $\partial L / \partial T(x_i)$: Gradiente de la pérdida con respecto a la predicción.

- **Pérdida Cuadrada:** En el caso de pérdida cuadrada, el gradiente equivale al residuo (Mohan, 2011):

•

$$r_i = y_i - T(x_i)$$

Donde:

- y_i : es el valor objetivo
- $T(x_i)$: es la predicción actual

- **Parámetros Clave:**

1. **Tasa de aprendizaje (α):** Tasa más pequeña mejora la precisión, pero requiere más iteraciones (Mohan, 2011).
2. **Profundidad de los árboles (d):** Usualmente pequeña ($d=4$).
3. **Número de iteraciones (MB):** Si es demasiado grande, puede causar sobreajuste (Mohan, 2011).

2. Comparación entre GBRT y RF

- **Estructura de los Modelos:**

- RF promedia **árboles completos y de alta varianza ($d=\infty$)** para evitar el sobreajuste (Mohan, 2011).
- GBRT agrega árboles pequeños con alto sesgo, centrándose en los errores restantes (Mohan, 2011).

- **Rendimiento Experimental:**

- En el conjunto de datos **Yahoo Learning to Rank Challenge**:
 - RF superó a GBRT en métricas como **ERR, NDCG y RMSE**.
 - Incrementar iteraciones ($MB=5000$, $MB = 5000$, $MB=5000$) en GBRT no mejoró el desempeño, sino que exacerbó el sobreajuste.
- En el conjunto de datos **Microsoft Learning to Rank**, los resultados fueron mixtos, sin un ganador claro.

3. Ventajas y Desventajas de GBRT

Ventajas:

- Excelente desempeño en tareas con funciones de pérdida diferenciables y problemas de clasificación.
- Flexibilidad para modelar relaciones no lineales mediante árboles pequeños y tasa de aprendizaje ajustada.

Desventajas:

- **Sobreajuste:** Puede ocurrir si MB es demasiado grande o si no se utiliza regularización adecuada.
- **Más Iteraciones:** Las tasas de aprendizaje pequeñas requieren más iteraciones, aumentando el tiempo de entrenamiento.
- **Sensible a Hiperparámetros:** El rendimiento depende críticamente de la configuración de α , d , y MB .

4. Conclusión

GBRT y RF son herramientas poderosas, pero tienen enfoques y aplicaciones diferentes. Mientras que RF sobresale en tareas donde la independencia entre árboles reduce el riesgo de sobreajuste, GBRT es ideal para problemas donde se necesita una optimización precisa.

4. El problema (contexto)

Una empresa minera enfrenta el desafío de mejorar el control de su proceso de producción, específicamente en la separación y concentración de minerales. El problema clave radica en la presencia de **impurezas**, como la **sílice**, en el concentrado de mineral. Estas impurezas se miden cada hora, pero actualmente no cuentan con una herramienta que les permita anticipar su concentración antes de realizar el análisis fisicoquímico correspondiente.

Si se logra predecir con precisión la cantidad de sílice presente en el concentrado, los ingenieros podrían contar con información temprana para tomar decisiones correctivas de manera proactiva. Este empoderamiento les permitiría ajustar variables del proceso en tiempo real, lo cual generaría múltiples beneficios:

1. **Reducción de impurezas:** Mejora en la calidad del concentrado de mineral.
2. **Mayor eficiencia operativa:** Evitar pérdidas por reprocesamiento.
3. **Menor impacto ambiental:** Disminución de la cantidad de mineral enviado a los relaves, optimizando el uso de recursos y reduciendo desechos.

En esencia, la capacidad de predecir las impurezas antes de que ocurran facilitaría la optimización del proceso de flotación y el uso adecuado de reactivos, ayudando tanto en el desempeño económico como en la sostenibilidad ambiental de la operación minera.

5. Propósito del estudio

El objetivo principal es **predecir mediante algoritmos de regresión la cantidad de sílice (impureza)** en el concentrado de mineral cada hora.

Este propósito se desglosa en los siguientes objetivos específicos:

1. **Desarrollar un código en Python:** Crear un programa que utilice técnicas de análisis de datos para predecir el porcentaje de impurezas de sílice al final del proceso de obtención de hierro, considerando todas las variables críticas involucradas en el proceso.
2. **Aplicar algoritmos de regresión:** Implementar modelos de aprendizaje automático enfocados en regresión para realizar predicciones numéricas precisas del contenido de impurezas de sílice en el concentrado de mineral.
3. **Seleccionar el mejor algoritmo:** Evaluar y comparar diferentes algoritmos de regresión utilizando métricas de rendimiento clave, como el error cuadrático medio (MSE), el coeficiente de determinación (R^2), y la raíz del error cuadrático medio (RMSE), para determinar cuál proporciona las predicciones más precisas y confiables.
4. **Reportar los resultados:** Generar un informe detallado que incluya los resultados obtenidos, las métricas de desempeño, y las conclusiones del análisis, para facilitar la implementación en tiempo real del modelo seleccionado en el sistema de control del proceso minero.

6. Descripción del proyecto

Las actividades realizadas en este proyecto involucran:

1. **Extracción de los datos:** de la página Kaggle se extraen los datos "Quality Prediction in a Mining Process" (Magalhães, 2017), esta base contiene los factores operativos como concentraciones de entrada y salida, flujo de aire, nivel de las columnas de flotación, pH, etc.
2. **Análisis exploratorio:** Identificar correlaciones entre las variables operativas y las impurezas. Detectar valores atípicos y patrones estacionales o cíclicos. Evaluar la calidad de los datos.
3. **Selección del Modelo** Los modelos a probar adecuados incluyen:
 - 3.1 **Modelos de regresión basados en árboles:**
 - Random Forest (RF).
 - Gradient Boosting (GBRT).
 - XGBoost o LightGBM (para mejor rendimiento y flexibilidad).
4. **Evaluación y Métricas**
 - Métrica principal: **Error Absoluto Medio (MAE)** para interpretar los errores en las mismas unidades de la concentración de sílice.

- Métrica secundaria: R^2 para evaluar qué tan bien el modelo captura la variabilidad.
- Validación con un conjunto de prueba separado para medir la generalización del modelo.

5. Beneficios Operativos y Ambientales

- **Eficiencia Operativa:** Menos impurezas significa menos reprocesamiento y menor desperdicio.
- **Sostenibilidad:** Reducir las impurezas minimiza los desechos, mejorando el impacto ambiental.
- **Empoderamiento:** Los ingenieros pueden anticiparse a los problemas y tomar decisiones basadas en datos.

7. Hipótesis

Si se utiliza un modelo de aprendizaje automático basado en algoritmos de regresión y se consideran todas las variables críticas del proceso de extracción de hierro, entonces será posible predecir con precisión el porcentaje de impurezas de sílice en el concentrado de mineral, permitiendo ajustes oportunos en el proceso para mejorar su eficiencia y sostenibilidad.

8. Flujo de trabajo

1. Definición del Problema y Objetivo

Identificar las variables críticas del proceso que afectan la presencia de impurezas de sílice.

Establecer el objetivo de predecir el porcentaje de impurezas para optimizar el control del proceso.

2. Limpieza y Preparación de Datos

Recolectar datos históricos del proceso de extracción, incluyendo variables operativas y mediciones de impurezas.

Limpiar y transformar los datos para garantizar su calidad (manejo de valores faltantes, imputación de datos, normalización, etc.).

Dividir los datos en conjuntos de entrenamiento y prueba.

3. Análisis Exploratorio

Realizar análisis exploratorio para identificar patrones, relaciones entre variables y posibles correlaciones.

Generar visualizaciones para comprender mejor la distribución y comportamiento de las variables.

4. Construcción de Modelos y algoritmos

Elegir algoritmos de regresión adecuados (e.g., regresión lineal, árbol de decisión, Random Forest, Gradient Boosting, etc.).

Entrenar los modelos utilizando el conjunto de datos de entrenamiento.

5. Evaluación de Modelos

Evaluar el desempeño de los modelos con métricas como RMSE, MSE y R² en el conjunto de prueba.

Comparar los resultados de los modelos para seleccionar el más preciso y eficiente.

6. Predicciones

Aplicar algoritmos de regresión para obtener predicciones numéricas del contenido de impurezas de sílice del proceso de extracción de hierro.

7. Reporte de Resultados y Documentación

Documentar el flujo de trabajo, el desempeño del modelo y las recomendaciones de implementación.

Presentar los resultados a los ingenieros y responsables del proceso para su aplicación operativa.



Figura 8 Flujo de trabajo

9. Secuencias de código en Python

9.1 Importar librerías

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

9.2 Cargar dataframe

```
import os
os.chdir("/Users/Gina/OneDrive/Documentos/EBAC")

df = pd.read_csv("MiningProcess_Flotation_Plant_Database.csv")
df.head()
```

	date	% Iron Feed	% Silica Feed	Starch Flow	Amina Flow	Ore Pulp Flow	Ore Pulp pH	Ore Pulp Density	Flotation Column 01 Air Flow	Flotation Column 02 Air Flow	...	Flotation Column 07 Air Flow	Flotation Column 01 Level	Flotation Column 02 Level	Flotation Column 03 Level	Flotation Column 04 Level	Flotation Column 05 Level
0	2017-03-10 01:00:00	55,2	16,98	3019,53	557,434	395,713	10,0664	1,74	249,214	253,235	...	250,884	457,396	432,962	424,954	443,558	502,2
1	2017-03-10 01:00:00	55,2	16,98	3024,41	563,965	397,383	10,0672	1,74	249,719	250,532	...	248,994	451,891	429,56	432,939	448,086	496,3
2	2017-03-10 01:00:00	55,2	16,98	3043,46	568,054	399,668	10,068	1,74	249,741	247,874	...	248,071	451,24	468,927	434,61	449,688	484,4
3	2017-03-10 01:00:00	55,2	16,98	3047,36	568,665	397,939	10,0689	1,74	249,917	254,487	...	251,147	452,441	458,165	442,865	446,21	471,4
4	2017-03-10 01:00:00	55,2	16,98	3033,69	558,167	400,254	10,0697	1,74	250,203	252,136	...	248,928	452,441	452,9	450,523	453,67	462,5

5 rows x 24 columns

```
df.shape
(737453, 24)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 737453 entries, 0 to 737452
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   date             737453 non-null   object 
 1   % Iron Feed      737453 non-null   object 
 2   % Silica Feed    737453 non-null   object 
 3   Starch Flow      737453 non-null   object 
 4   Amina Flow       737453 non-null   object 
 5   Ore Pulp Flow    737453 non-null   object 
 6   Ore Pulp pH      737453 non-null   object 
 7   Ore Pulp Density 737453 non-null   object 
 8   Flotation Column 01 Air Flow 737453 non-null   object 
 9   Flotation Column 02 Air Flow 737453 non-null   object 
 10  Flotation Column 03 Air Flow 737453 non-null   object 
 11  Flotation Column 04 Air Flow 737453 non-null   object 
 12  Flotation Column 05 Air Flow 737453 non-null   object 
 13  Flotation Column 06 Air Flow 737453 non-null   object 
 14  Flotation Column 07 Air Flow 737453 non-null   object 
 15  Flotation Column 01 Level   737453 non-null   object 
 16  Flotation Column 02 Level   737453 non-null   object 
 17  Flotation Column 03 Level   737453 non-null   object 
 18  Flotation Column 04 Level   737453 non-null   object 
 19  Flotation Column 05 Level   737453 non-null   object 
 20  Flotation Column 06 Level   737453 non-null   object 
 21  Flotation Column 07 Level   737453 non-null   object 
 22  % Iron Concentrate 737453 non-null   object 
 23  % Silica Concentrate 737453 non-null   object 
dtypes: object(24)
memory usage: 135.0+ MB
```

9.3 Limpiar dataframe

```
# Reemplazar ' ' por '_'
df.columns = df.columns.str.replace(' ','_')

# Reemplazar simbolo '%' con palabra 'porciento' para evitar errores de sintaxis.
df.columns = df.columns.str.replace('%','percent')

# Cambiar nombres de las columnas por minúsculas
df.columns= df.columns.str.lower()

# Reemplazar , por . en todos los números de todas las columnas
for column in df.columns:
    df[column] = df[column].apply(lambda x: x.replace(',', '.'))

# Eliminar La columna date ya que no se utilizará para este análisis
df = df.drop('date', axis=1)
df.head()

# Cambiar tipo de dato del dataset
df = df.astype(np.float)
df.head()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 737453 entries, 0 to 737452
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   percent_iron_feed    737453 non-null   float64
 1   percent_silica_feed  737453 non-null   float64
 2   starch_flow          737453 non-null   float64
 3   amina_flow           737453 non-null   float64
 4   ore_pulp_flow        737453 non-null   float64
 5   ore_pulp_ph          737453 non-null   float64
 6   ore_pulp_density     737453 non-null   float64
 7   flotation_column_01_air_flow 737453 non-null   float64
 8   flotation_column_02_air_flow 737453 non-null   float64
 9   flotation_column_03_air_flow 737453 non-null   float64
 10  flotation_column_04_air_flow 737453 non-null   float64
 11  flotation_column_05_air_flow 737453 non-null   float64
 12  flotation_column_06_air_flow 737453 non-null   float64
 13  flotation_column_07_air_flow 737453 non-null   float64
 14  flotation_column_01_level   737453 non-null   float64
 15  flotation_column_02_level   737453 non-null   float64
 16  flotation_column_03_level   737453 non-null   float64
 17  flotation_column_04_level   737453 non-null   float64
 18  flotation_column_05_level   737453 non-null   float64
 19  flotation_column_06_level   737453 non-null   float64
 20  flotation_column_07_level   737453 non-null   float64
 21  percent_iron_concentrate 737453 non-null   float64
 22  percent_silica_concentrate 737453 non-null   float64
dtypes: float64(23)
memory usage: 129.4 MB
```

9.4 Análisis exploratorio

Estadísticas descriptivas del dataset.

```
print("\nEstadísticas Descriptivas:\n", df.describe())
```

```
Estadísticas Descriptivas:
   percent_iron_feed  percent_silica_feed  starch_flow  amina_flow \
count      737453.000000      737453.000000  737453.000000  737453.000000
mean       56.204739      14.651716   2869.140569   488.144697
std        5.157744      6.887439  1215.283734   91.230534
min        42.740000      1.310000     0.002026   241.669000
25%       52.670000      8.940000   2076.320000   431.796000
50%       56.080000     13.850000   3018.430000   504.393000
75%       59.720000     19.600000   3727.730000   553.257000
max       65.780000     33.400000   6100.230000   739.518000

   ore_pulp_flow  ore_pulp_ph  ore_pulp_density \
count      737453.000000      737453.000000  737453.000000
mean       397.578372      9.767639     1.680180
std        9.699785      0.387007     0.069249
min        376.249000      8.753340     1.519820
25%       394.264000      9.527360     1.647310
50%       399.249000      9.798100     1.697600
75%       402.968000     10.038000     1.728330
max       418.641000     10.808100     1.851250

   flotation_column_01_air_flow  flotation_column_02_air_flow \
count      737453.000000                  737453.000000
mean       288.151856                  277.159965
std        29.621288                  30.149357
min        175.510000                  175.156000
25%       250.281000                  250.457000
50%       299.344000                  296.223000
75%       300.149000                  300.690000
max       373.871000                  375.992000

   flotation_column_03_air_flow  ...  flotation_column_07_air_flow \
count      737453.000000  ...                  737453.000000
mean       281.082397  ...                  290.754856
std        28.558268  ...                  28.670105
min        176.469000  ...                  185.962000
25%       250.855000  ...                  256.302000
50%       298.696000  ...                  299.011000
75%       300.382000  ...                  301.964000
max       364.346000  ...                  371.593000
```

```

flotation_column_01_level flotation_column_02_level \
jax/extensions/Safe.js 737453.000000 737453.000000 \
mean 520.244823 522.649555

std 131.014924 128.165050
min 149.218000 210.752000
25% 416.978000 441.883000
50% 491.878000 495.956000
75% 594.114000 595.464000
max 862.274000 828.919000

flotation_column_03_level flotation_column_04_level \
count 737453.000000 737453.000000 \
mean 531.352662 420.320973
std 150.842164 91.794432
min 126.255000 162.281000
25% 411.325000 356.679000
50% 494.318000 411.974000
75% 601.249000 485.549000
max 886.822000 680.359000

flotation_column_05_level flotation_column_06_level \
count 737453.000000 737453.000000 \
mean 425.251706 429.941018
std 84.535822 89.862225
min 166.991000 155.841000
25% 357.653000 358.497000
50% 408.773000 424.664575
75% 484.329000 492.684000
max 675.644000 698.861000

flotation_column_07_level percent_iron_concentrate \
count 737453.000000 737453.000000 \
mean 421.021231 65.050068
std 84.891491 1.118645
min 175.349000 62.050000
25% 356.772000 64.370000
50% 411.065000 65.210000
75% 476.465000 65.860000
max 659.902000 68.010000

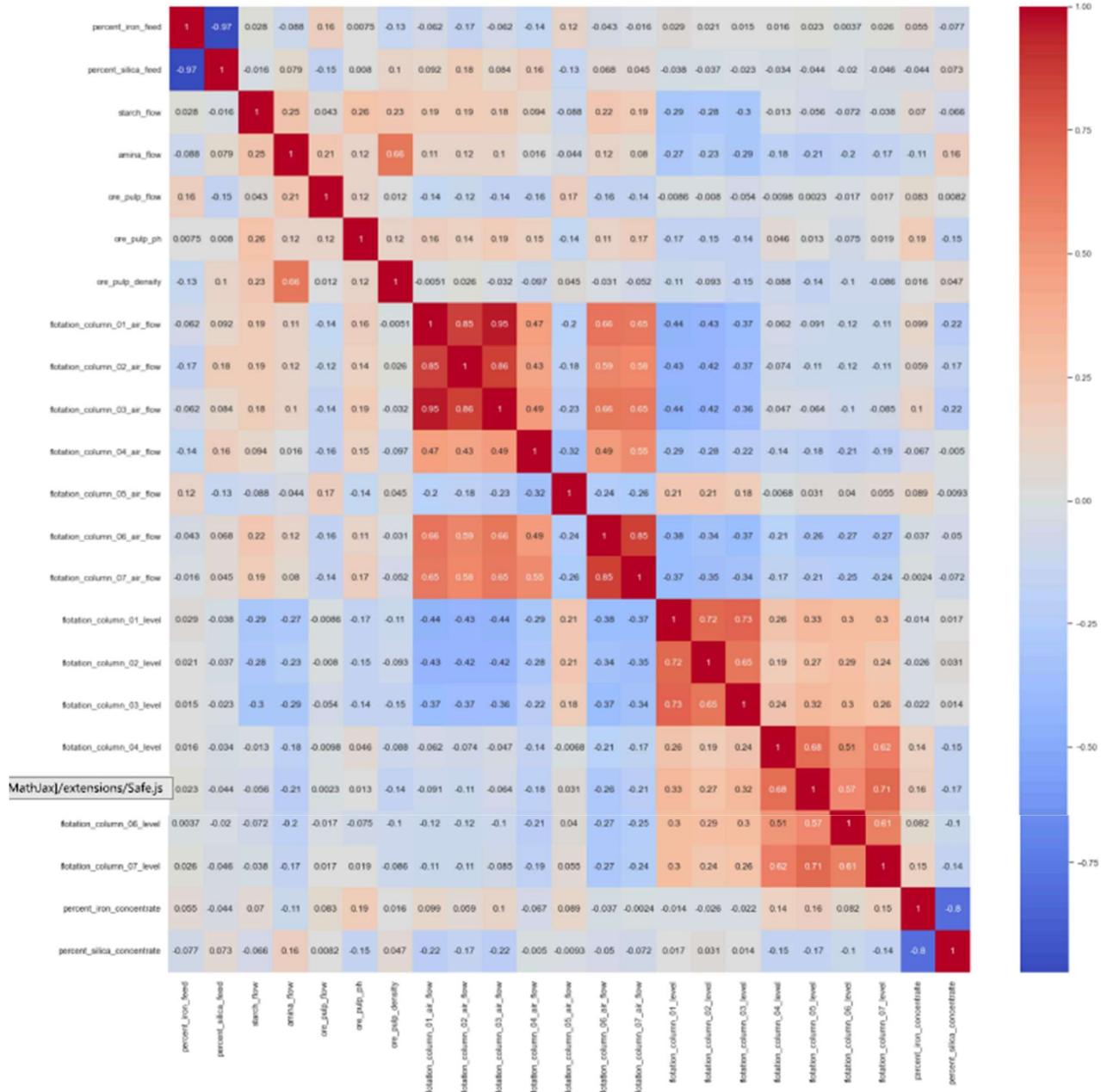
percent_silica_concentrate
count 737453.000000
mean 2.326763
std 1.125554
min 0.600000
jax/extensions/Safe.js 1.440000
50% 2.000000

75% 3.010000
max 5.530000

```

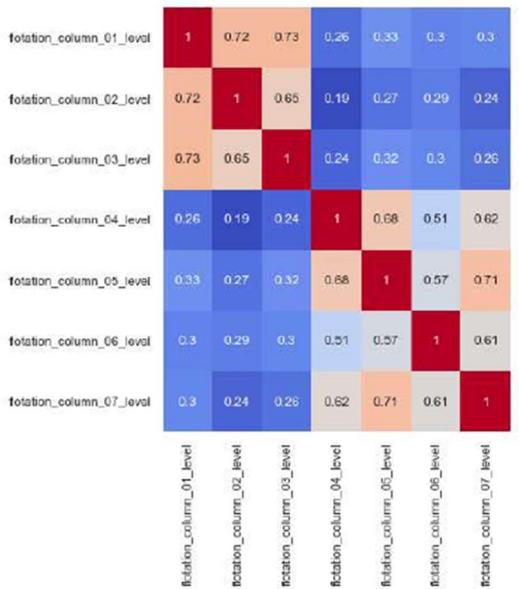
[8 rows x 23 columns]

Matriz de Correlación

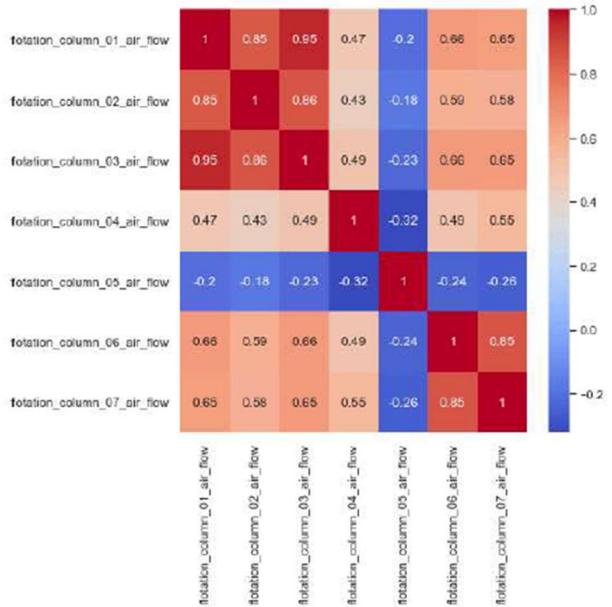


Acercamiento a la matriz de correlación en zonas de mayor correlación.

```
cor = df.iloc[:,14:21].corr();
plt.figure(figsize=(7,7))
sns.set(font_scale=1)
sns.heatmap(cor, annot=True, cmap=plt.cm.coolwarm);
```



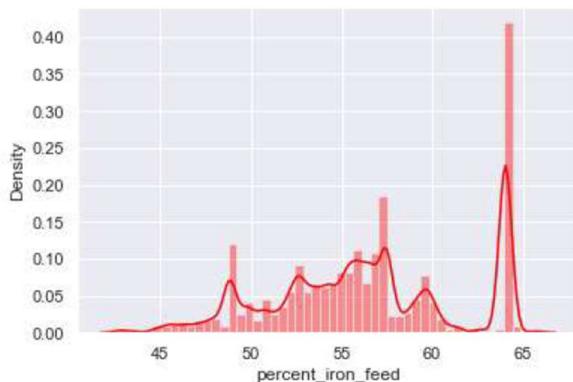
```
# Aceramiento a los puntos donde se observan colores más intensos
cor = df.iloc[:,7:14].corr();
plt.figure(figsize=(7,7))
sns.set(font_scale=1)
sns.heatmap(cor, annot=True, cmap=plt.cm.coolwarm);
```



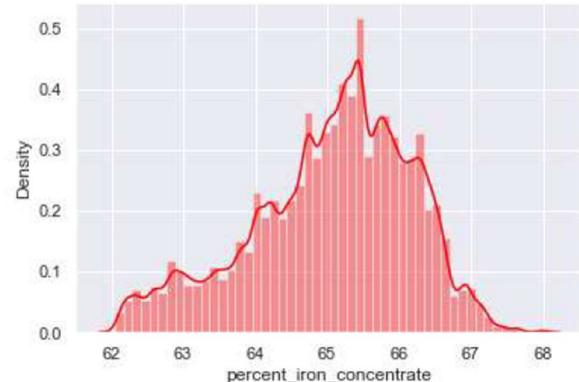
Se observa que existe alta correlación entre los flujos de aire y los niveles de las columnas de flotación.

9.5 Visualización de datos Visualización de porcientos de Hierro a la entrada y salida.

```
# Histograma de Seaborn
import seaborn as sns
fig1=sns.distplot(df["percent_iron_feed"], color="red")
fig1
<AxesSubplot:xlabel='percent_iron_feed', ylabel='Density'>
```



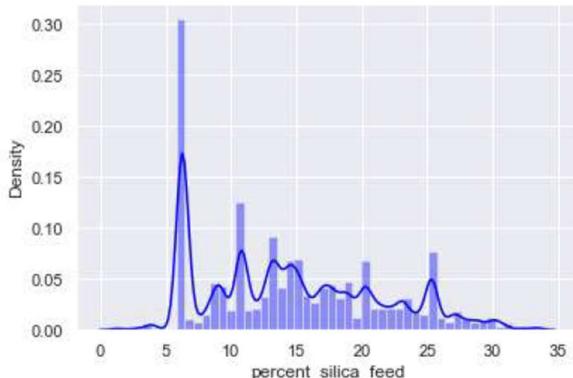
```
fig2=sns.distplot(df["percent_iron_concentrate"], color="red")
fig2
<AxesSubplot:xlabel='percent_iron_concentrate', ylabel='Density'>
```



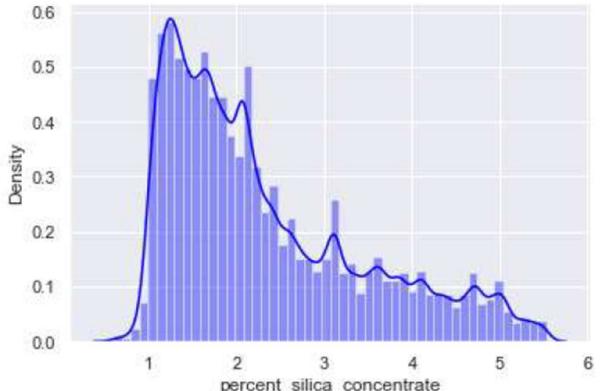
Se observa que las concentraciones de hierro a la entrada son muy dispersos que van desde el 45 al 65%, al final del proceso se incrementan los contenidos de hierro a un rango de **62 - 67%**.

Visualización de porciones de impurezas de Sílice a la entrada y salida

```
fig3=sns.distplot(df["percent_silica_feed"], color="blue")
fig3
Jax].extensions/Safe.js xlabel='percent_silica_feed', ylabel='Density'>
```



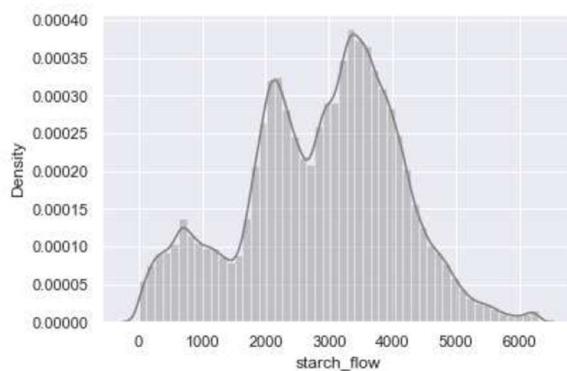
```
fig3A=sns.distplot(df["percent_silica_concentrate"],color="blue")
fig3A
<AxesSubplot:xlabel='percent_silica_concentrate', ylabel='Density'>
```



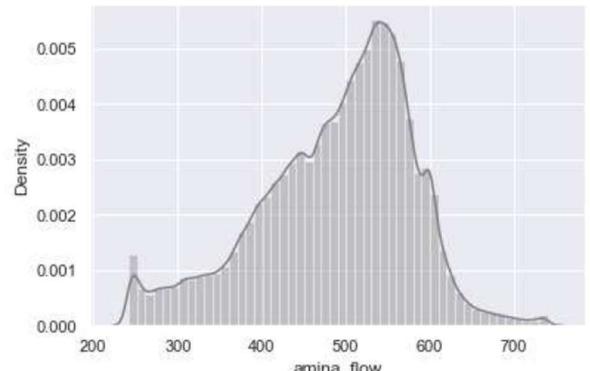
Se observa que las concentraciones de **sílice** a la entrada son muy dispersas que van desde el **5 al 30%**, al final del proceso se reducen las impurezas hasta el **1 a 5%**.

Visualización de porciones de impurezas de Flujos de Amina y Starch (almidón) en la entrada y salida.

```
fig4=sns.distplot(df["starch_flow"],color="gray")
fig4
Jax].extensions/Safe.js xlabel='starch_flow', ylabel='Density'>
```



```
fig5=sns.distplot(df["amina_flow"],color="gray")
fig5
<AxesSubplot:xlabel='amina_flow', ylabel='Density'>
```



Flujo de Almidón ("starch_flow"):

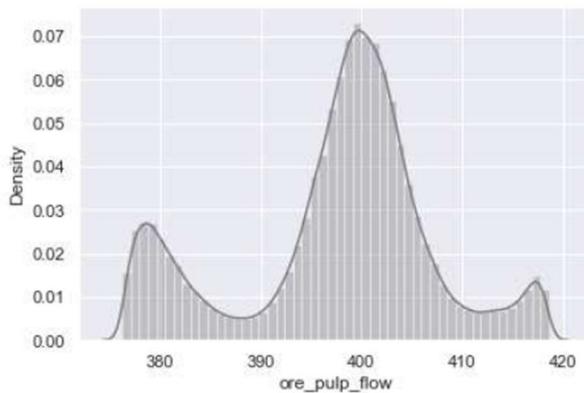
- La distribución muestra una mayor variabilidad y varios picos, lo cual indica que los valores del flujo de almidón tienden a fluctuar ampliamente en el proceso.
- Los picos sugieren que puede haber diferentes condiciones operativas o ajustes en el proceso que resulten en concentraciones específicas de almidón.
- La densidad máxima se encuentra entre los valores cercanos a 3000-4000, lo que podría ser el rango operativo más común del flujo.

Flujo de Aminas ("amina_flow"):

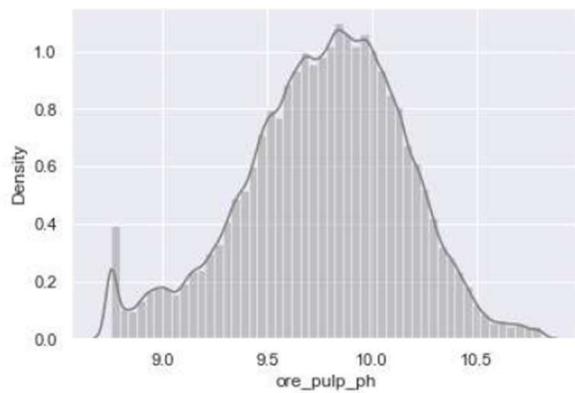
- La distribución es más estrecha y presenta un solo pico significativo, indicando menor variabilidad y un control más estable en el flujo de aminas.
- La mayoría de los valores se concentran entre 400 y 600, con un pico cerca de 500, lo que probablemente representa el valor óptimo o más común utilizado en el proceso.

Visualización de porciones de impurezas de Flujos de Amina y Starch (ganga) en la entrada y salida.

```
fig6=sns.distplot(df["ore_pulp_flow"], color="gray")
fig6
Jax]/extensions/Safe.js  xlabel='ore pulp flow', ylabel='Density'>
```



```
fig7=sns.distplot(df["ore_pulp_ph"], color="gray")
fig7
<AxesSubplot:xlabel='ore pulp ph', ylabel='Density'>
```



Flujo de Ganga ("ore_flow"):

- La distribución muestra una de tres grandes picos, lo cual indica que los valores del flujo de ganga pueden considerarse como condiciones bajas medias y altas, que fluctúan durante el proceso.
- Los picos sugieren que puede haber diferentes condiciones operativas o ajustes en el proceso ser que resulten en concentraciones específicas de alimentación de la ganga.
- La densidad máxima se encuentra entre los valores cercanos a 400, lo que podría ser el rango operativo más común del flujo.

pH de ganga ("ore_pulp_ph"):

- La distribución es más estrecha y presenta un solo pico significativo, indicando menor variabilidad y un control más estable en el pH.
- La mayoría de los valores se concentran entre 9.0 – 10.0, con un pico cerca de 9.8, lo que probablemente representa el valor óptimo o más común utilizado en el proceso.

Visualización de Flujos de las columnas de flotación en las celdas de extracción de hierro.

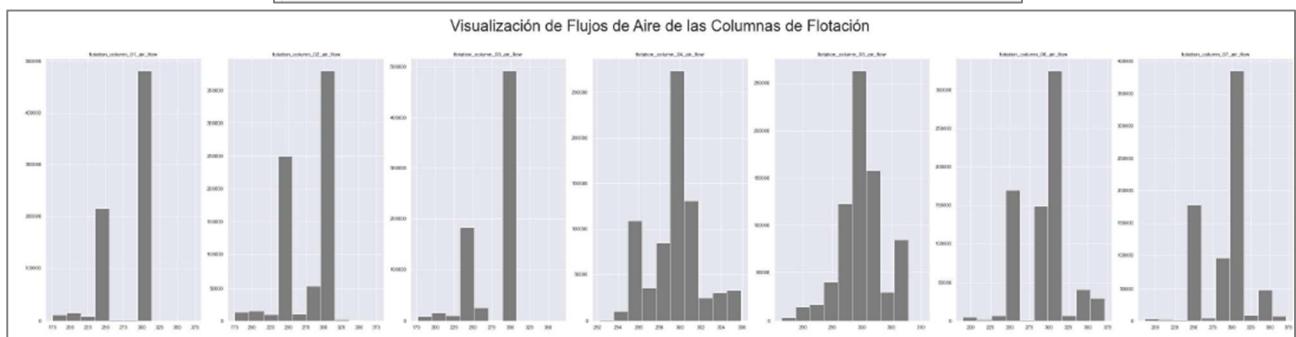
```
# Visualización de Flujos de Aire de Las columnas de flotación.
viz = df[['floatation_column_01_air_flow', 'floatation_column_02_air_flow',
          'floatation_column_03_air_flow', 'floatation_column_04_air_flow',
          'floatation_column_05_air_flow', 'floatation_column_06_air_flow',
          'floatation_column_07_air_flow']]
# Crear La figura y los ejes
figFlujo, axes = plt.subplots(1,7, figsize=(40, 10)) # Ajusta el tamaño aquí

# Generar histogramas para cada columna
viz.hist(ax=axes, color="gray")

# Añadir título a toda La gráfica
figFlujo.suptitle('Visualización de Flujos de Aire de las Columnas de Flotación', fontsize=35)

# Ajustar el espacio entre gráficos
plt.tight_layout(rect=[0, 0, 1, 0.96]) # Ajuste para que no solape el título

Jax/extensions/Safejs paciado entre gráficos
plt.show()
```



Distribución del Flujo de Aire en las Columnas:

- En todas las columnas, los flujos de aire parecen estar concentrados en uno o dos rangos principales, lo que sugiere que se operan en condiciones relativamente constantes para la mayoría del tiempo.
- Hay ciertas columnas donde se observa una mayor dispersión o valores extremos, indicando posibles variaciones operativas o ajustes en esas columnas específicas.

Comparación Entre Columnas:

- Algunas columnas muestran picos únicos bien definidos, lo que indica una operación más estable (por ejemplo, las columnas donde un solo rango de flujo domina significativamente).
- Otras columnas presentan distribuciones más amplias, lo que podría implicar que están sujetas a mayor variabilidad operativa.
- Impacto Potencial en el Proceso:
- Las columnas con flujos de aire más estables probablemente contribuyen a un rendimiento más consistente en el proceso de flotación.
- Las columnas con distribuciones más amplias o picos múltiples podrían estar operando fuera de los rangos óptimos de flujo, lo que puede afectar la eficiencia de la flotación y la calidad del hierro extraído.

Identificación de Problemas Potenciales:

- Columnas con valores extremos: Si algunas columnas presentan frecuentemente flujos de aire en valores extremos, podrían estar enfrentando problemas como bloqueos, fallas en el suministro de aire o desajustes en la configuración del equipo.

- Variabilidad entre columnas: Si las diferencias entre columnas son significativas, puede ser necesario un análisis para verificar si las condiciones operativas están optimizadas para cada una.

Recomendaciones:

- Analizar la Variabilidad: Investigar las causas detrás de la variabilidad en las columnas con distribuciones más amplias.
- Optimizar Configuraciones: Evaluar si las columnas con picos múltiples requieren ajustes para alinearlas con las columnas más estables.
- Monitorear Consistencia: Implementar un sistema de monitoreo en tiempo real para garantizar que los flujos de aire permanezcan dentro de los rangos óptimos establecidos.

Visualización de Niveles de las columnas de flotación en las celdas de extracción de hierro.



Distribución de los Niveles de Aire por Columna:

- Cada columna tiene una distribución distinta de los niveles de aire. Algunas columnas presentan una mayor concentración en un rango estrecho, mientras que otras muestran mayor dispersión o múltiples picos.
- La mayoría de las columnas parecen operar dentro de ciertos valores dominantes, pero con variaciones significativas en la forma y dispersión de las distribuciones.

Estabilidad Operativa:

- Columnas Estables: Columnas con distribuciones concentradas y un solo pico predominante indican niveles de aire más controlados y consistentes, lo cual es ideal para optimizar el rendimiento del proceso.

- Columnas Variables: Las columnas con distribuciones más amplias o con múltiples picos podrían estar experimentando fluctuaciones operativas o inconsistencias en el suministro de aire.

Comparación Entre Columnas:

- Hay diferencias significativas en los niveles dominantes de aire entre las columnas. Esto podría reflejar configuraciones distintas o necesidades específicas de cada columna, pero también puede indicar oportunidades de optimización para estandarizar los niveles.
- Las columnas que tienen niveles de aire significativamente diferentes a las demás podrían estar afectando la uniformidad del proceso.

Impacto en el Proceso de Flotación:

- Columnas Consistentes: Las columnas con niveles de aire más estables probablemente contribuyen a un proceso de flotación más eficiente y consistente.
- Columnas Variables: Las columnas con mayor variabilidad en los niveles de aire pueden provocar ineficiencias, como una separación inconsistente del hierro o pérdida de mineral en el proceso.

Recomendaciones:

- Monitoreo y Control: Implementar sistemas de control automatizado para estabilizar los niveles de aire en las columnas que presentan mayor variabilidad.
- Optimización: Analizar si las columnas con distribuciones más amplias o niveles significativamente diferentes están afectando la calidad del proceso y realizar ajustes en las configuraciones.
- Homogeneidad: Evaluar si es posible alinear los niveles de aire entre columnas para mejorar la consistencia general del proceso.

9.6 Transformación de datos

	percent_iron_feed	percent_silica_feed	starch_flow	amina_flow	ore_pulp_flow	ore_pulp_ph	ore_pulp_density	flotation_column_01_air_flow	floatation
655712	55.89	16.46	3304.19000	465.808	382.737	10.02110	1.705770	299.727	
345388	64.03	6.26	636.22600	594.141	396.144	10.20280	1.719010	298.783	
501681	56.76	13.34	3493.90000	509.939	395.091	9.96154	1.652330	250.306	
416019	53.55	19.94	767.14552	296.914	400.573	9.77634	1.520908	299.385	
626512	56.20	13.03	3670.03000	583.439	418.116	9.72958	1.696090	299.988	

5 rows × 23 columns

Dividir la base de datos en entrenamiento y prueba.

```
X_train = train_df.drop(columns=['percent_silica_concentrate'])
y_train = train_df['percent_silica_concentrate']

X_test = test_df.drop(columns=['percent_silica_concentrate'])
y_test = test_df['percent_silica_concentrate']

X_train.columns

Index(['percent_iron_feed', 'percent_silica_feed', 'starch_flow', 'amina_flow',
       'ore_pulp_flow', 'ore_pulp_ph', 'ore_pulp_density',
       'flotation_column_01_air_flow', 'flotation_column_02_air_flow',
       'flotation_column_03_air_flow', 'flotation_column_04_air_flow',
       'flotation_column_05_air_flow', 'flotation_column_06_air_flow',
       'flotation_column_07_air_flow', 'flotation_column_01_level',
       'flotation_column_02_level', 'flotation_column_03_level',
       'flotation_column_04_level', 'flotation_column_05_level',
       'flotation_column_06_level', 'flotation_column_07_level',
       'percent_iron_concentrate'],
       dtype='object')
```

9.7 Modelado

9.7.1 Regresión Lineal

```
from sklearn.pipeline import FeatureUnion, Pipeline, make_pipeline
from sklearn.preprocessing import (MinMaxScaler, StandardScaler)
from sklearn.compose import ColumnTransformer, make_column_transformer

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# separar la base datos de entrenamiento y prueba
train_df, test_df = train_test_split(df, test_size=0.3, random_state=111)
train_df.head()
```

```
# Divide el conjunto en entrenamiento y prueba
X_train = train_df.drop(columns=['percent_silica_concentrate'])
y_train = train_df['percent_silica_concentrate']

X_test = test_df.drop(columns=['percent_silica_concentrate'])
y_test = test_df['percent_silica_concentrate']
```

```
# Escalar las variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Modelo de regresión lineal múltiple
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train) #hacer el ajuste por regresión lineal

LinearRegression()
```

```
# impresión de coeficientes de regresión múltiple
print("Intercepto: ", linreg.intercept_)
print("Coeficientes: ", linreg.coef_)

Intercepto:  46.13504472985562
Coeficientes: [ 2.48350587e-02  2.83760776e-02 -1.68059609e-05  6.78098609e-04
   4.14071944e-03  4.74576637e-02  1.78547427e-01 -5.08542280e-03
  -1.40544907e-03 -1.02464101e-03  2.50264255e-03  1.44825321e-02
  -1.21077403e-04  1.20893283e-03 -2.82494600e-04 -1.60951537e-04
  -7.93323247e-05 -1.43495226e-04 -1.81441666e-04 -2.01964453e-04
  2.00113166e-04 -7.87205384e-01]
```

Son los coeficientes de cada una de las variables.

```
# Realiza predicciones
y_pred = linreg.predict(X_test)
y_pred
array([1.10685761, 1.73456752, 2.28229122, ..., 2.25933075, 1.54463481,
       3.48798333])

# Calcular indicadores de error de La bondad de ajuste
from sklearn.metrics import r2_score
from sklearn import metrics

# Impresión de indicadores de bondad de ajuste
print("Valor de R cuadrada: ", r2_score(y_test, y_pred))
print("Error Absoluto Medio: ", metrics.mean_absolute_error(y_test, y_pred))
print("Error Cuadrático Medio: ", metrics.mean_squared_error(y_test, y_pred))
print("Raíz del Error cuadrático Medio: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Valor de R cuadrada:  0.6791508227479826
Error Absoluto Medio:  0.49214080370273733
Error Cuadrático Medio:  0.4056501453067106
Raíz del Error cuadrático Medio:  0.6369067006294648
```

Para la **regresión lineal** el R-cuadrada es de 0.68, indica que el modelo solo puede **explicar el 68% de la variación de los datos**, lo cual no es un buen ajuste.

9.7.2 Regresión Ridge

Valor de Alpha = 0

```
from sklearn.linear_model import Ridge

#Regresión Ridge con Alpha =0 (equivalente a una regresión Lineal)

ridgereg = Ridge(alpha =0, normalize =True) #modelo
ridgereg.fit(X_train, y_train)               #ajuste
y_pred= ridgereg.predict(X_test)             #predicción

# Impresión de indicadores de bondad de ajuste
print("Valor de R cuadrada: ", r2_score(y_test, y_pred))
print("Error Absoluto Medio: ", metrics.mean_absolute_error(y_test, y_pred))
print("Error Cuadrático Medio: ", metrics.mean_squared_error(y_test, y_pred))
print("Raíz del Error cuadrático Medio: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Valor de R cuadrada:  0.6791508227479857
Error Absoluto Medio:  0.49214080370271696
Error Cuadrático Medio:  0.4056501453067066
Raíz del Error cuadrático Medio:  0.6369067006294616
```

Para la regresión lineal Ridge, los indicadores de bondad de la regresión son los mismos, comprobándose que es una regresión lineal al asignarle un valor Alpha = 0.

Valor de Alpha = 0.1

```
#Regresión Ridge con Alpha =0.1

ridgereg = Ridge(alpha =0.1, normalize =True) #modelo
ridgereg.fit(X_train, y_train) #ajuste
y_pred= ridgereg.predict(X_test) #predicción

# Impresión de indicadores de bondad de ajuste
print("Valor de R cuadrada: ", r2_score(y_test, y_pred))
print("Error Absoluto Medio: ", metrics.mean_absolute_error(y_test, y_pred))
print("Error Cuadrático Medio: ", metrics.mean_squared_error(y_test, y_pred))
print("Raíz del Error cuadrático Medio: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Valor de R cuadrada:  0.6726243873879312
Error Absoluto Medio:  0.49678580518219756
Error Cuadrático Medio:  0.41390152832353583
Raíz del Error cuadrático Medio:  0.6433517920419091

Con alpha 0.1 se tiene un R cuadrada de 0.6727, NO mejoró la predicción.
```

Con el valor Alpha = 0.1, se tiene una R-cuadrada de **0.6727** y que NO mejoró la predicción.

Ahora se prueban diferentes valores de Alpha.

```
# Probar diversos valores de alpha con un rango de prueba para alpha
alpha_range = 10.0**np.arange(-2,3)
alpha_range

array([1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02])

# Importar librería para encontrar La Combination Value
from sklearn.linear_model import RidgeCV
ridgeregcv = RidgeCV(alphas=alpha_range, normalize=True, scoring='neg_mean_squared_error')
ridgeregcv.fit(X_train, y_train)
```

0.01

El programa elige que el mejor Alpha es 0.01.

```
# Predicción Ridge mediante el mejor valor de Alpha
y_pred= ridgeregcv.predict(X_test)

# Impresión de indicadores de bondad de ajuste
print("Valor de R cuadrada: ", r2_score(y_test, y_pred))
print("Error Absoluto Medio: ", metrics.mean_absolute_error(y_test, y_pred))
print("Error Cuadrático Medio: ", metrics.mean_squared_error(y_test, y_pred))
print("Raíz del Error cuadrático Medio: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Valor de R cuadrada:  0.6789335588493579
Error Absoluto Medio:  0.49231846057220274
Error Cuadrático Medio:  0.40592483241297594
Raíz del Error cuadrático Medio:  0.6371223056941077
```

Se obtienen valores menores R cuadrada de Ridge con alpha (0.01) de (0.6789), comparada con regresión lineal (0.6791).

Para la **regresión lineal de Ridge** con un Alpha de 0.01 se tiene una R-Cuadrada de **(0.6789)** comparada con la regresión lineal (0.6791), por lo tanto, no mejora la predicción.

9.7.3 Regresión de Lasso

Esta regresión elimina de forma automática las variables no significativas, colocándoles el valor cero.

```
# Prueba con alpha = 0.001
from sklearn.linear_model import Lasso
lassoreg = Lasso(alpha= 0.001, normalize= True) #modelo
lassoreg.fit(X_train, y_train) #ajuste

# Examinacion de coeficientes de la regresión lasso
print("Intercepto: ", lassoreg.intercept_)
print("Coeficientes: ", lassoreg.coef_)

Intercepto: 12.973184488471594
Coeficientes: [-0.         0.         -0.         0.         0.         -0.
 0.         -0.         -0.         -0.         0.         0.
 -0.         -0.         0.         0.         -0.         -0.
 -0.         -0.         -0.         -0.16366157]
```

*#Selección del valor óptimo de alpha para regresión Lasso
La maquina elige el valor optimo de alpha por un número de intentos.*

```
from sklearn.linear_model import LassoCV
lassoregcv = LassoCV(n_alphas=100, normalize=True, random_state=1)
lassoregcv.fit(X_train, y_train)
print("Alpha óptimo: ", lassoregcv.alpha_)
```

Alpha óptimo: 1.2549439475317922e-06

```
# Examinacion de coeficientes de la regresión LassoCV
print("Intercepto: ", lassoregcv.intercept_)
print("Coeficientes: ", lassoregcv.coef_)
```

```
Intercepto: 46.78553604641771
Coeficientes: [ 1.81111577e-02  2.32767743e-02 -1.50885780e-05  6.75285291e-04
 4.13747257e-03  4.60493631e-02  1.52675028e-01 -4.94982539e-03
 -1.40032470e-03 -1.05878752e-03  2.11080993e-03  1.40493656e-02
 -0.00000000e+00  1.04317442e-03 -2.69165521e-04 -1.59601771e-04
 -7.39131280e-05 -1.35907779e-04 -1.67350267e-04 -1.89785225e-04
 1.46495775e-04 -7.85889200e-01]
```

```
# Predicción mediante regresión Lasso CV con alpha optimo
y_pred = lassoregcv.predict(X_test) #predicción
```

```
# Impresión de indicadores de bondad de ajuste
print("Valor de R cuadrada: ", r2_score(y_test, y_pred))
print("Error Absoluto Medio: ", metrics.mean_absolute_error(y_test, y_pred))
print("Error Cuadrático Medio: ", metrics.mean_squared_error(y_test, y_pred))
print("Raíz del Error cuadrático Medio: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Valor de R cuadrada: 0.6790425928764803
Error Absoluto Medio: 0.4923365979838596
Error Cuadrático Medio: 0.4057869805122032
Raíz del Error cuadrático Medio: 0.6370141132755248
```

Para la **regresión lineal de Lasso** con un alpha óptimo de 1.25e-06, se tiene una R-Cuadrada de **(0.6790)** comparada con la regresión lineal (0.6791), por lo tanto, no mejora la predicción.

9.7.4 Reporte automatizado de regresión lineal

Tratamiento de datos para la regresión lineal

```
from sklearn.pipeline import FeatureUnion, Pipeline, make_pipeline
from sklearn.preprocessing import (MinMaxScaler, StandardScaler)
from sklearn.compose import ColumnTransformer, make_column_transformer

from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.3, random_state=111)
train_df.head()
```

```
X_train = train_df.drop(columns=['percent_silica_concentrate'])
y_train = train_df['percent_silica_concentrate']

X_test = test_df.drop(columns=['percent_silica_concentrate'])
y_test = test_df['percent_silica_concentrate']

numeric_features = ['percent_iron_feed', 'percent_silica_feed', 'starch_flow', 'amina_flow',
    'ore_pulp_flow', 'ore_pulp_ph', 'ore_pulp_density',
    'flotation_column_01_air_flow', 'flotation_column_02_air_flow',
    'flotation_column_03_air_flow', 'flotation_column_04_air_flow',
    'flotation_column_05_air_flow', 'flotation_column_06_air_flow',
    'flotation_column_07_air_flow', 'flotation_column_01_level',
    'flotation_column_02_level', 'flotation_column_03_level',
    'flotation_column_04_level', 'flotation_column_05_level',
    'flotation_column_06_level', 'flotation_column_07_level',
    'percent_iron_concentrate']
```

```
# Aplicar feature transformation a datos de Entrenamiento (Train)
from sklearn.preprocessing import StandardScaler

numeric_transformer = make_pipeline(StandardScaler())
preprocessor = make_column_transformer((numeric_transformer, numeric_features))

preprocessor.fit(X_train)
preprocessor.named_transformers_
Jax/extensions/Safe.js pipeline(steps=[('standardscaler', StandardScaler())])}
```

```
X = X_train
Y = y_train

# Numero de renglones en matriz X base de entrenamiento
X.shape[0]
```

516217

```
import statsmodels.api as sm

X = sm.add_constant(X)
X.shape[0]
```

```
# Numero de columnas en Matriz X base de entrenamiento
X.shape[1]
```

23

```
# Reporte Automatizado de La regresión en Python
import statsmodels.api as sm

regressor = sm.OLS(Y, X).fit()      #minimos cuadrados ordinarios
print(regressor.summary())
```

OLS Regression Results

Dep. Variable:	percent_silica_concentrate	R-squared:	0.679			
Model:	OLS	Adj. R-squared:	0.679			
Method:	Least Squares	F-statistic:	4.974e+04			
Date:	Mon, 30 Dec 2024	Prob (F-statistic):	0.00			
Time:	14:35:41	Log-Likelihood:	-5.0007e+05			
No. Observations:	516217	AIC:	1.000e+06			
Df Residuals:	516194	BIC:	1.000e+06			
Df Model:	22					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	46.1350	0.186	248.594	0.000	45.771	46.499
percent iron feed	0.0248	0.001	32.845	0.000	0.023	0.026
percent silica feed	0.0284	0.001	49.845	0.000	0.027	0.029
starch_flow	-1.681e-05	8.2e-07	-20.487	0.000	-1.84e-05	-1.52e-05
amina flow	0.0007	1.46e-05	46.600	0.000	0.001	0.001
ore pulp flow	0.0041	0.000	40.414	0.000	0.004	0.004
ore_pulp_ph	0.0475	0.003	18.670	0.000	0.042	0.052
ore pulp density	0.1785	0.018	9.759	0.000	0.143	0.214
flotation_column_01_air_flow	-0.0051	0.000	-48.431	0.000	-0.005	-0.005
flotation column 02 air flow	-0.0014	6.13e-05	-22.916	0.000	-0.002	-0.001
flotation_column_03_air_flow	-0.0010	0.000	-8.889	0.000	-0.001	-0.001
flotation_column_04_air_flow	0.0025	0.000	5.599	0.000	0.002	0.003
flotation_column_05_air_flow	0.0145	0.000	53.762	0.000	0.014	0.015
flotation_column_06_air_flow	-0.0001	6.12e-05	-1.979	0.048	-0.000	-1.14e-06
flotation_column_07_air_flow	0.0012	6.38e-05	18.946	0.000	0.001	0.001
flotation_column_01_level	-0.0003	1.15e-05	-24.466	0.000	-0.000	-0.000
flotation column 02 level	-0.0002	1.06e-05	-15.138	0.000	-0.000	-0.000
flotation_column_03_level	-7.933e-05	9.18e-06	-8.638	0.000	-9.73e-05	-6.13e-05
flotation_column_04_level	-0.0001	1.39e-05	-10.328	0.000	-0.000	-0.000
flotation_column_05_level	-0.0002	1.73e-05	-10.463	0.000	-0.000	-0.000
flotation_column_06_level	-0.0002	1.33e-05	-15.174	0.000	-0.000	-0.000
flotation_column_07_level	0.0002	1.64e-05	12.217	0.000	0.000	0.000
percent_iron_concentrate	-0.7872	0.001	-921.092	0.000	-0.789	-0.786
Omnibus:	9388.106	Durbin-Watson:	2.000			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	13949.676			
Skew:	0.205	Prob(JB):	0.00			
Kurtosis:	3.693	Cond. No.	7.22e+05			

JaxJ/extensjons/Safe.js

Se realiza la misma transformación realizada a los datos de prueba:

```
# Aplicar feature transformation a datos de Prueba (Test)
from sklearn.preprocessing import StandardScaler

numeric_transformer = make_pipeline(StandardScaler())
preprocessor = make_column_transformer((numeric_transformer, numeric_features))
preprocessor.fit(X_test)
preprocessor.named_transformers_
X_test
```

```
# Cálculo de Predicciones en base de Prueba y Entrenamiento
X_test = sm.add_constant(X_test)
X_train = sm.add_constant(X_train)
y_train_pred = regressor.predict(X_train)
y_test_pred = regressor.predict(X_test)
```

Finalmente se calculan las métricas de rendimiento para la regresión lineal automatizada:

```
# Cálculo de métricas
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)

# Resultados
print("Métricas para el conjunto de entrenamiento:")
print(f"R²: {r2_train:.4f}")
print(f"MSE: {mse_train:.4f}")
print(f"MAE: {mae_train:.4f}")

print("\nMétricas para el conjunto de prueba:")
print(f"R²: {r2_test:.4f}")
print(f"MSE: {mse_test:.4f}")
print(f"MAE: {mae_test:.4f}")
```

Métricas para el conjunto de entrenamiento:

R²: 0.6795
 MSE: 0.4064
 MAE: 0.4926

Métricas para el conjunto de prueba:

R²: 0.6792
 MSE: 0.4057
 MAE: 0.4921

Para la **regresión lineal automatizada**, se tiene una R-Cuadrada de **(0.6792)** para la base de entrenamiento comparada con la regresión lineal (0.6791), por lo tanto, no mejora la predicción.

9.7.5 Random Forest Regressor

Es un modelo basado en un conjunto de árboles de decisión que combina múltiples árboles para mejorar la precisión y reducir el riesgo de sobreajuste.

Dividir la base de entrenamiento.

```
#Dividir base de entrenamiento y de prueba
train_df, test_df = train_test_split(df, test_size=0.3, random_state=111)
train_df.head()
```

```
# Crear X_train, y_train, X_test and y_test
X_train = train_df.drop(columns=['percent_silica_concentrate'])
y_train = train_df['percent_silica_concentrate']

X_test = test_df.drop(columns=['percent_silica_concentrate'])
y_test = test_df['percent_silica_concentrate']
```

Importación de librerías y creación del modelo.

```
from sklearn.ensemble import RandomForestRegressor
rf_reg = RandomForestRegressor(max_depth=2, random_state=0)
```

Entrenar el modelo.

```
rf_reg.fit(X_train, y_train)

RandomForestRegressor(max_depth=2, random_state=0)
```

Predicciones usando la base de prueba (X_test)

```
# Predicciones
y_train_pred = rf_reg.predict(X_train)
y_test_pred = rf_reg.predict(X_test)
```

Cálculo de las métricas

```
# Cálculo de métricas
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)

# Resultados
print("Métricas para el conjunto de entrenamiento:")
print(f"R²: {r2_train:.4f}")
print(f"MSE: {mse_train:.4f}")
print(f"MAE: {mae_train:.4f}")

print("\nMétricas para el conjunto de prueba:")
print(f"R²: {r2_test:.4f}")
print(f"MSE: {mse_test:.4f}")
print(f"MAE: {mae_test:.4f}")
```

Métricas para el conjunto de entrenamiento:

R²: 0.6456
 MSE: 0.4493
 MAE: 0.5065

Métricas para el conjunto de prueba:

R²: 0.6444
 MSE: 0.4495
 MAE: 0.5065

Para el modelo de **Random Forest Regressor**, el valor de R² para los conjuntos de entrenamiento y prueba están alrededor de **0.64**, lo que sugiere que el modelo explica aproximadamente el 64% de la varianza en los datos. Como los valores de entrenamiento y prueba son muy cercanos, el modelo parece generalizar bien y no presenta sobreajuste.

- El MSE mide el error promedio al cuadrado entre los valores predichos y los observados. Un **valor bajo de MSE** indica un **buen ajuste del modelo**. En este caso, los valores de entrenamiento y prueba son similares, lo que nuevamente sugiere que el modelo tiene un rendimiento consistente entre los conjuntos de datos.
- El MAE mide el error promedio absoluto entre las predicciones y los valores reales. Un valor **MAE de 0.506** significa que, en promedio, las predicciones del modelo tienen un error absoluto de 0.506 unidades. La cercanía entre los valores de entrenamiento y prueba refuerza la idea de que **el modelo es confiable para datos nuevos**.

9.7.6 Gradient Boosting Regressor

```

from sklearn.ensemble import GradientBoostingRegressor

#Dividir base de entrenamiento y de prueba
train_df, test_df = train_test_split(df, test_size=0.3, random_state=111)
train_df.head()

# Crear X_train, y_train, X_test and y_test
X_train = train_df.drop(columns=['percent_silica_concentrate'])
y_train = train_df['percent_silica_concentrate']

X_test = test_df.drop(columns=['percent_silica_concentrate'])
y_test = test_df['percent_silica_concentrate']

gbt_reg = GradientBoostingRegressor(n_estimators=100, random_state=1)

gbt_reg.fit(X_train, y_train)

GradientBoostingRegressor(random_state=1)

# Predicciones
y_train_pred = gbt_reg.predict(X_train)
y_test_pred = gbt_reg.predict(X_test)

# Cálculo de métricas
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

mae_train = mean_absolute_error(y_train, y_train_pred)
mae_test = mean_absolute_error(y_test, y_test_pred)

# Resultados
print("Métricas para el conjunto de entrenamiento:")
print(f"R2: {r2_train:.4f}")
print(f"MSE: {mse_train:.4f}")
print(f"MAE: {mae_train:.4f}")

print("\nMétricas para el conjunto de prueba:")
print(f"R2: {r2_test:.4f}")
print(f"MSE: {mse_test:.4f}")
print(f"MAE: {mae_test:.4f}")

```

Métricas para el conjunto de entrenamiento:

R²: 0.7927

MSE: 0.2628

MAE: 0.3787

Métricas para el conjunto de prueba:

R²: 0.7902

MSE: 0.2652

MAE: 0.3801

Para el modelo de **Gradien Boosting Regressor**, el valor de R² para los conjuntos de entrenamiento y prueba están alrededor de **0.79**, un el R² más alto indica que podría ser un modelo más adecuado para **capturar la relación entre las variables**.

- MSE: 0.2652 (indicando menor error medio cuadrático).
- MAE: 0.3801 (indicando un menor error absoluto medio).
- Este modelo tiene el mejor desempeño en todas las métricas, siendo más preciso que los anteriores.
- Ofrece una mejor precisión, aunque puede ser menos interpretable debido a su naturaleza no lineal.

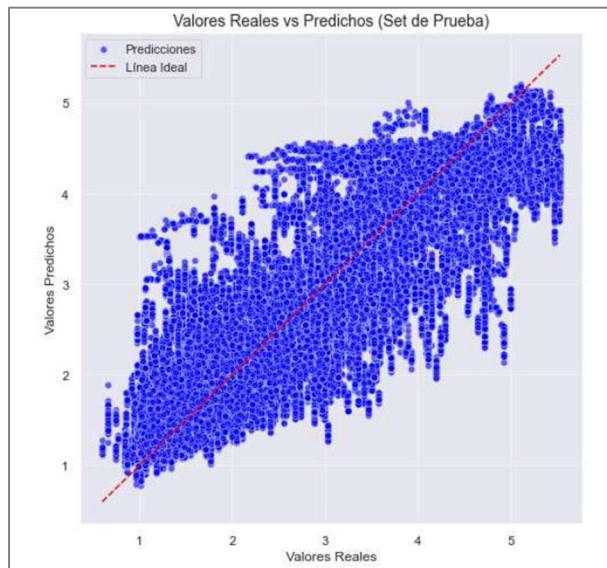
Crear Gráfico de Residuales vs Predicciones para Gradien Boosting Regressor

```
# Crear una figura
plt.figure(figsize=(8, 8))

# Gráfica de dispersión
sns.scatterplot(x=y_test, y=y_test_pred, alpha=0.6, color='blue', label='Predicciones')

# Línea de referencia (perfecto ajuste)
max_val = max(np.max(y_test), np.max(y_test_pred))
min_val = min(np.min(y_test), np.min(y_test_pred))
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--', label='Línea Ideal')

# Configuración de La gráfica
plt.title("Valores Reales vs Predichos (Set de Prueba)", fontsize=14)
plt.xlabel("Valores Reales", fontsize=12)
plt.ylabel("Valores Predichos", fontsize=12)
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```



Distribución de los residuales: Los residuales parecen distribuidos alrededor de la línea roja ($y=0$), lo cual es una señal positiva.

Sin embargo, hay un ligero patrón en forma de abanico: la dispersión de los residuales parece aumentar para valores más bajos y más altos de las predicciones. Esto podría indicar heterocedasticidad, es decir, que la variabilidad de los errores no es constante.

Patrones visibles: Algunos valores extremos (outliers) están presentes, especialmente para predicciones más bajas y más altas.

Estos puntos podrían estar afectando la calidad del ajuste del modelo.

Conclusión preliminar: El modelo parece capturar bien la relación entre las variables, pero la presencia de heterocedasticidad sugiere que el modelo no ajusta de manera uniforme en todo el rango de las predicciones.

9.7.7 Modelo XGB Regressor

```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error

#Dividir base de entrenamiento y de prueba
train_df, test_df = train_test_split(df, test_size=0.3, random_state=111)
train_df.head()
```

```
# Crear X_train, y_train, X_test and y_test
X_train = train_df.drop(columns=['percent_silica_concentrate'])
y_train = train_df['percent_silica_concentrate']

X_test = test_df.drop(columns=['percent_silica_concentrate'])
y_test = test_df['percent_silica_concentrate']
```

```
# Crear el modelo XGBoost
xgb_reg = XGBRegressor(
    n_estimators=500,           # Número de árboles
    learning_rate=0.1,          # Tasa de aprendizaje
    max_depth=6,                # Profundidad máxima de los árboles
    subsample=0.8,              # Fracción de datos para cada árbol
    colsample_bytree=0.8,        # Fracción de características para cada árbol
    random_state=42             # Reproducibilidad
)
```

```
# Entrenar el modelo
xgb_reg.fit(X_train, y_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
            colsample_bylevel=None, colsample_bynode=None,
            colsample_bytree=0.8, early_stopping_rounds=None,
            enable_categorical=False, eval_metric=None, feature_types=None,
            gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
            interaction_constraints=None, learning_rate=0.1, max_bin=None,
            max_cat_threshold=None, max_cat_to_onehot=None,
            max_delta_step=None, max_depth=6, max_leaves=None,
            min_child_weight=None, missing=nan, monotone_constraints=None,
            n_estimators=500, n_jobs=None, num_parallel_tree=None,
            predictor=None, random_state=42, ...)
```

```
# Hacer predicciones
y_train_pred_xgb = xgb_reg.predict(X_train)
y_test_pred_xgb = xgb_reg.predict(X_test)
```

```
# Cálculo de métricas
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

r2_train = r2_score(y_train, y_train_pred_xgb)
r2_test = r2_score(y_test, y_test_pred_xgb)

mse_train = mean_squared_error(y_train, y_train_pred_xgb)
mse_test = mean_squared_error(y_test, y_test_pred_xgb)

mae_train = mean_absolute_error(y_train, y_train_pred_xgb)
mae_test = mean_absolute_error(y_test, y_test_pred_xgb)
```

```
# Resultados
print("Métricas para el conjunto de entrenamiento:")
print(f"R²: {r2_train:.4f}")
print(f"MSE: {mse_train:.4f}")
print(f"MAE: {mae_train:.4f}")

print("\nMétricas para el conjunto de prueba:")
print(f"R²: {r2_test:.4f}")
print(f"MSE: {mse_test:.4f}")
print(f"MAE: {mae_test:.4f}")
```

Métricas para el conjunto de entrenamiento:

R²: 0.9632
MSE: 0.0466
MAE: 0.1508

Métricas para el conjunto de prueba:

R²: 0.9611
MSE: 0.0492
MAE: 0.1551

1. R² (Coeficiente de Determinación)

El R² mide qué tan bien el modelo explica la variación de los datos. Su valor oscila entre:

- R² = 1: Predicciones perfectas.
- R² = 0: El modelo no explica nada mejor que el promedio de los datos.
- R² < 0: El modelo es peor que asumir el promedio como predicción.
- Entrenamiento: R² = 0.9632. El modelo explica el 96.32% de la variabilidad de los datos de entrenamiento.
- Prueba: R² = 0.9611. El modelo explica el 96.11% de la variabilidad de los datos de prueba.

El modelo generaliza bien, ya que el R² en el conjunto de prueba es cercano al de entrenamiento. No parece haber sobreajuste.

2. MSE (Mean Squared Error)

El MSE mide el error cuadrático promedio entre los valores reales (yreal) y predichos (ypred).

- Entrenamiento: MSE=0.0466
- Prueba: MSE=0.0492

Ambos valores son cercanos, indicando que el modelo está generalizando bien. El **MSE es bajo**, lo que significa que **las predicciones están cerca de los valores reales**.

3. MAE (Mean Absolute Error)

El MAE mide el error absoluto promedio, es decir, cuánto se desvían las predicciones de los valores reales en promedio.

- Entrenamiento: MAE=0.1508
- Prueba: MAE=0.1551

Los valores MAE de entrenamiento y prueba son cercanos, lo que sugiere que el modelo generaliza bien.

Para estimar el rango en el que podría estar el valor real, podemos usar el valor de la R^2 y los errores absolutos promedio (MAE) o errores cuadrados medios (MSE). Aunque R^2 por sí sola no proporciona un rango exacto de predicciones, podemos hacer una aproximación utilizando la métrica **MAE** que indica el error promedio en las predicciones.

Dado que el **MAE** en el conjunto de entrenamiento fue **0.1815**, lo que significa que el modelo tiene un error promedio de aproximadamente 0.1815 unidades en las predicciones, podemos utilizar esto para construir un intervalo de confianza aproximado alrededor de la predicción.

Resumen general XGB Regressor:

- Los valores de R^2 , MSE, y MAE son similares para los conjuntos de entrenamiento y prueba, lo que indica que el modelo no está sobreajustado ni subajustado.
- Los valores del MSE y MAE son pequeños, lo cual indica que las predicciones están cerca de los valores reales.

Cálculo de los residuales modelo XGB Regressor

```
# Calcular los residuales
residuals = y_test - y_test_pred_xgb

# Crear un DataFrame para facilitar el análisis (opcional)
import pandas as pd
residuals_df = pd.DataFrame({
    "Real": y_test,
    "Predicción": y_test_pred_xgb,
    "Residual": residuals
})
```

```

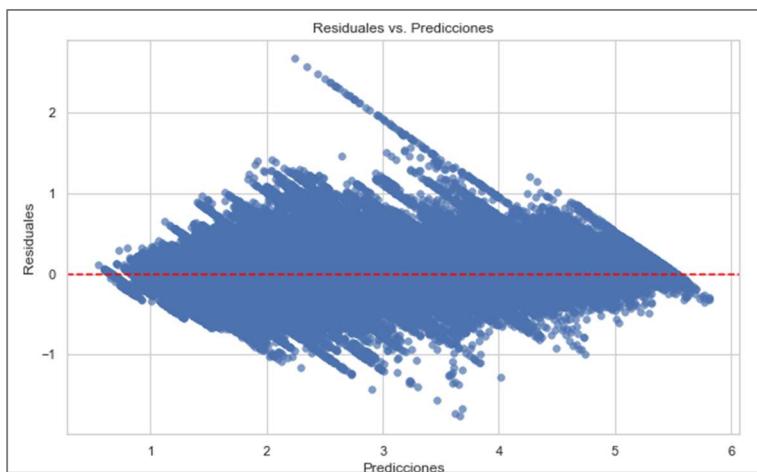
# Configuración de estilo para las gráficas
sns.set(style="whitegrid")

# 1. Gráfico de los residuales vs predicciones
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test_pred_xgb, y=residuals, alpha=0.7, edgecolor=None)
plt.axline(0, color='red', linestyle='--')
plt.title("Residuales vs. Predicciones")
plt.xlabel("Predicciones")
plt.ylabel("Residuales")
plt.show()

# 2. Histograma de los residuales
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True, bins=30, color="blue", alpha=0.7)
plt.axline(0, color='red', linestyle='--')
plt.title("Distribución de los Residuales")
plt.xlabel("Residual")
plt.ylabel("Frecuencia")
plt.show()

# 3. Gráfico Q-Q para verificar normalidad de los residuales
import scipy.stats as stats
plt.figure(figsize=(10, 6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Gráfico Q-Q de los Residuales")
plt.show()

```



Distribución de los residuales:

- Los residuales (diferencias entre las predicciones y los valores reales) parecen estar distribuidos alrededor de la línea roja ($y = 0$), lo cual indica que el modelo no tiene un sesgo significativo en general.
- Sin embargo, hay una concentración densa de puntos y algunas regiones con una forma particular, lo cual podría indicar patrones sistemáticos.

Patrones de heterocedasticidad:

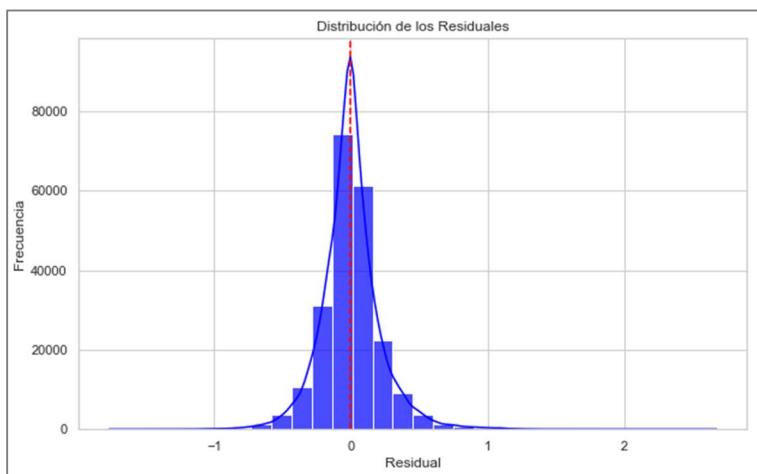
- Parece haber un incremento en la dispersión de los residuales a medida que aumenta el valor de las predicciones, especialmente en los extremos.
- Esto podría indicar heterocedasticidad, es decir, que la variabilidad de los errores no es constante y depende del valor de las predicciones.

Possible relación no capturada:

Aunque los residuales están centrados en 0, la forma de los puntos podría sugerir que el modelo no está capturando perfectamente ciertas relaciones entre las variables.

Puntos extremos o outliers:

Se observan algunos puntos con valores residuales muy altos o bajos, que podrían ser outliers o errores significativos del modelo.

**Forma de la distribución:**

La distribución de los residuales parece aproximarse a una curva normal (campana de Gauss), lo cual es una buena señal de que el modelo está funcionando adecuadamente. Sin embargo, hay ligeros indicios de asimetría en los extremos, con residuales más positivos o negativos que podrían deberse a errores sistemáticos o a datos atípicos.

Media cercana a 0:

La media de los residuales está muy cerca de 0 (línea roja), lo que sugiere que el modelo no tiene un sesgo significativo hacia predicciones por encima o por debajo de los valores reales.

Varianza controlada:

La mayor parte de los residuales está concentrada alrededor de 0, lo que indica que los errores son pequeños en su mayoría, pero hay algunos valores residuales más alejados (colas largas).

Simetría de las colas:

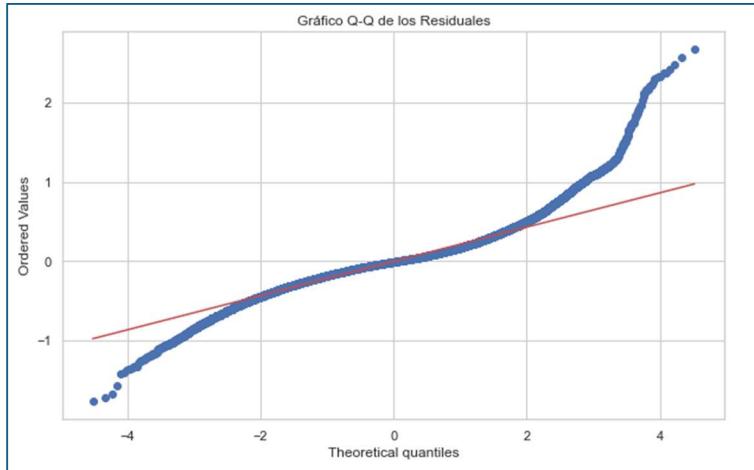
Las colas de la distribución no parecen completamente simétricas, lo que podría indicar ligeras desviaciones de la normalidad.

Frecuencia alta cerca de 0:

Hay una alta frecuencia de valores residuales pequeños, lo que indica que el modelo logra ajustar bien muchas observaciones.

Prueba de Breusch-Pagan XGB Regressor

La prueba de Breusch-Pagan es una herramienta estadística utilizada para detectar heterocedasticidad en un modelo de regresión, es decir, si la varianza de los errores/residuales no es constante a lo largo de los valores de las variables explicativas.



En el gráfico Q-Q (quantile-quantile) de los residuales se pueden observar los siguientes puntos clave:

Ajuste a la línea teórica: Los puntos están cercanos a la línea roja (línea teórica de normalidad) en el centro de la distribución, lo que indica que los residuales en esa región tienen un comportamiento aproximadamente normal. Sin embargo, en las colas (extremos izquierdo y derecho), los puntos se desvían de la línea, lo que sugiere que los residuales tienen colas más gruesas o delgadas de lo esperado en una distribución normal.

Colas pesadas: En el extremo derecho (residuales grandes) y el extremo izquierdo (residuales pequeños), los puntos se alejan significativamente de la línea. Esto indica que la distribución tiene colas más largas que las de una distribución normal, lo que podría ser un indicador de outliers o de errores sistemáticos.

Leve asimetría: Hay indicios de que los valores en las colas no son completamente simétricos, lo que podría sugerir cierta asimetría en la distribución de los residuales.

Normalidad imperfecta: Aunque hay un ajuste aceptable en la mayoría de los valores, las desviaciones en los extremos indican que los residuales no siguen exactamente una distribución normal.

Aunque la normalidad no es un requisito estricto en todos los modelos de machine learning (como XGB Regressor), estas desviaciones pueden indicar áreas de mejora en el modelo.

Es posible que el modelo esté fallando al capturar relaciones en los datos extremos o que existan outliers que distorsionen los resultados.

Prueba de Breusch-Pagan para XGB Regressor

```
# Revisar la heterocedasticidad
from statsmodels.stats.diagnostic import het_breushpagan
import statsmodels.api as sm

# Realizar la prueba de Breusch-Pagan
exog = sm.add_constant(y_test_pred_xgb) # Agregar constante
test = het_breushpagan(residuals, exog)
labels = ['LM Statistic', 'p-value', 'F-Statistic', 'F p-value']
print(dict(zip(labels, test)))

{'LM Statistic': 1597.1611105958448, 'p-value': 0.0, 'F-Statistic': 1608.760740715279, 'F p-value': 0.0}
```

Los resultados de la **prueba de Breusch-Pagan** indican que hay evidencia significativa de heterocedasticidad en el modelo:

- LM Statistic y F-Statistic: Los valores son altos, lo que sugiere una fuerte relación entre los residuales y las predicciones. p-values (0.0): Ambos son prácticamente nulos, indicando que la heterocedasticidad es estadísticamente significativa.
- La heterocedasticidad implica que la varianza de los errores no es constante en todo el rango de las predicciones. Esto puede afectar la fiabilidad de las métricas y predicciones del modelo, ya que los errores en ciertas regiones (por ejemplo, para valores extremos) son mayores que en otras.

Intervalo de confianza aproximado para nuevas predicciones.

Finalmente, al realizar predicciones con valores nuevos, el rango estimado para el valor real se determina de la siguiente forma:

Supongamos que la predicción para el %Iron_Concentrate es **66.43%** y el **MAE** es **0.1815%**. El rango de valores en los que probablemente se encuentra el valor real sería:

- **Valor real mínimo estimado:** $66.43 - 0.1815 = 66.2485\%$
 $66.43 - 0.1815 = 66.2485\%$
- **Valor real máximo estimado:** $66.43 + 0.1815 = 66.6115\%$
 $66.43 + 0.1815 = 66.6115\%$

Entonces, el **rango estimado basado en MAE** para el valor real de **%Iron_Concentrate** en base a la predicción de **66.43%** sería aproximadamente entre **66.25%** y **66.61%**.

10. Evaluación de modelos

En la tabla se muestra el resumen de las métricas de los conjuntos de Prueba para cada uno de los modelos generados:

	Modelo	Valor de R ²	Error Absoluto Medio MAE	Error Cuadrático Medio MSE	Raíz del Error cuadrático Medio
1	Regresión lineal	0.67916	0.4921	0.4056	0.6369
2	Regresión lineal RidgeCV alpha óptima de (0.01)	0.6789	0.4923	0.4059	0.6371
3	Regresión lineal LassoCV alpha óptima de (1.25e-06)	0.6790	0.4923	0.4057	0.6370
4	Regresión lineal Reporte Automatizado	0.6792	0.4921	0.4057	0.6369
5	Random Forest Regressor	0.6444	0.5065	0.4495	0.6704
6	Gradient Boosting Forest Regressor	0.7902	0.3801	0.2652	0.5150
7	XGB Regressor	0.9611	0.1551	0.0492	0.2218

Comparación de Modelos:

1. Regresiones Lineales:

- Los modelos de regresión lineal (incluyendo RidgeCV y LassoCV) tienen valores de R² entre **0.6789** y **0.6792**, lo que indica que aproximadamente el **67.89% - 67.92%** de la variabilidad en la concentración de sílice puede ser explicada por las características del modelo.
- El **Error Absoluto Medio (MAE)** se encuentra alrededor de **0.492**, lo que indica que, en promedio, las predicciones están a aproximadamente **0.492 unidades** de la verdad.
- El **Error Cuadrático Medio (MSE)** y la **Raíz del Error Cuadrático Medio (RMSE)** son variables que también reflejan un desempeño similar entre estos modelos, justo debajo de **0.64**.

2. Random Forest Regressor:

- Este modelo muestra un **R² de 0.6444**, lo que indica una capacidad de explicación ligeramente inferior a la de los modelos de regresión lineal.
- Sin embargo, el **MAE** es de **0.5065**, un poco mayor que en las regresiones lineales, lo que sugiere predicciones menos precisas.
- El MSE y RMSE son más altos que los de los modelos de regresión lineal, indicando que las predicciones a veces son menos precisas.

3. Gradient Boosting Forest Regressor:

- Presenta el mejor rendimiento con un **R² de 0.7902**, mostrando que el **79.02%** de la variabilidad en la concentración de sílice está explicada por el modelo.
- El **MAE** y el MSE son significativamente menores, indicando que las predicciones son más precisas en comparación con los modelos de regresión lineal y Random Forest.

4. XGB Regressor:

- Este modelo exhibe el rendimiento más alto con un **R² de 0.9611**, lo que significa que **96.11%** de la variabilidad es explicada, una capacidad de ajuste excelente.
- El **MAE** es el más bajo de todos los modelos a **0.1551**, lo que indica que las predicciones son en promedio muy cercanas a los valores reales.
- El **MSE** y **RMSE** son también notablemente bajos, sugiriendo que este modelo es superior en la precisión de las predicciones.

11. Predicciones

Se realizaron nuevas predicciones para un nuevo registro de valores provenientes de producción y verificarlos con los métodos analizados y observar los valores de porciento de sílice esperados para este nuevo lote.

```
nuevos_datos = pd.DataFrame({
    'percent_iron_feed': [55.89],
    'percent_silica_feed':[16.46],
    'starch_flow':[3304.190],
    'amina_flow':[465.808],
    'ore_pulp_flow':[382.737],
    'ore_pulp_ph':[10.02110],
    'ore_pulp_density':[1.705770],
    'flotation_column_01_air_flow':[299.727],
    'flotation_column_02_air_flow':[300.775],
    'flotation_column_03_air_flow':[300.806],
    'flotation_column_04_air_flow':[305.400],
    'flotation_column_05_air_flow':[484.587],
    'flotation_column_06_air_flow':[556.694],
    'flotation_column_07_air_flow':[570.694],
    'flotation_column_01_level':[195.048],
    'flotation_column_02_level':[339.004],
    'flotation_column_03_level':[356.123],
    'flotation_column_04_level':[282.322],
    'flotation_column_05_level':[443.558],
    'flotation_column_06_level':[446.37],
    'flotation_column_07_level':[523.344],
    'percent_iron_concentrate':[67.11]})
```

```

# Agregar constante al conjunto de datos para el modelo de statsmodels
# X_nueva_regressor = sm.add_constant(nuevos_datos) # Solo para 'regressor'

# Realizar predicciones
# silica_prediction_regressor = regressor.predict(X_nueva_regressor)

# Para los modelos de scikit-learn, utiliza el DataFrame sin constantes
silica_prediction_linreg = linreg.predict(nuevos_datos)
silica_prediction_lassoregcv = lassoregcv.predict(nuevos_datos)
silica_prediction_rf_reg = rf_reg.predict(nuevos_datos)
silica_prediction_gbt_reg = gbt_reg.predict(nuevos_datos)
silica_prediction_xgb_reg = xgb_reg.predict(nuevos_datos)

# Imprimir los resultados
print(f"Predicción del % Silica Concentrate (Regressor - OLS): {silica_prediction_regressor[0]:.2f}")
print(f"Predicción del % Silica Concentrate (Linear Regression): {silica_prediction_linreg[0]:.2f}")
print(f"Predicción del % Silica Concentrate (Lasso Regression): {silica_prediction_lassoregcv[0]:.2f}")
print(f"Predicción del % Silica Concentrate (Random Forest): {silica_prediction_rf_reg[0]:.2f}")
print(f"Predicción del % Silica Concentrate (Gradient Boosting Regressor): {silica_prediction_gbt_reg[0]:.2f}")
print(f"Predicción del % Silica Concentrate (XGB Regressor): {silica_prediction_xgb_reg[0]:.2f}")

Predicción del % Silica Concentrate (Linear Regression): 3.69
Predicción del % Silica Concentrate (Lasso Regression): 3.58
Predicción del % Silica Concentrate (Random Forest): 1.50
Predicción del % Silica Concentrate (Gradient Boosting Regressor): 1.51
Predicción del % Silica Concentrate (XGB Regressor): 1.32

```

Interpretación de las Predicciones

Los resultados de las predicciones indican la cantidad estimada de concentración de sílice en la muestra analizada, y cada modelo proporciona un valor diferente. A continuación, se presentan algunas interpretaciones sobre estos resultados:

1. Diferencias entre Modelos:

Los modelos de **regresión lineal y Lasso** predicen una concentración de sílice considerablemente más alta (**3.69** y **3.58**, respectivamente) en comparación con los modelos de **Random Forest, Gradient Boosting, y XGB** (que predicen entre **1.32** y **1.51**). Esto sugiere que los modelos más simples pueden estar sobreestimando la concentración de sílice.

2. Consistencia entre Modelos Complejos:

Los valores de Random Forest, Gradient Boosting, y XGB son comparativamente más cercanos entre sí, lo que podría indicar que estos modelos, que son más complejos, están capturando mejor la variabilidad en los datos y están proporcionando estimaciones más confiables.

3. Interpretación de la Variedad en Predicciones:

La diferencia entre las predicciones podría sugerir que el modelo lineal y el modelo Lasso no están considerando algunas de las interacciones o relaciones no lineales entre las variables presentes en los datos. Los modelos de árbol (como Random Forest y XGB) son más capaces de manejar estas relaciones.

Si bien los modelos de Random Forest y XGB que predicen valores más altos pueden parecer atractivos, es importante validar estos resultados con datos reales. La **elección del modelo final** se basará en los resultados de la evaluación de desempeño de cada modelo ofrecen información valiosa sobre su eficacia en la predicción de la concentración de sílice.

12. Resultados

El modelo **XGB Regressor** se destaca notablemente como el mejor modelo en términos de capacidad de explicación y precisión en las predicciones. Los modelos de regresión lineal están bien equilibrados, mientras que el Random Forest muestra un rendimiento más débil en comparación. Considerando estos resultados, se recomendaría el uso del **XGB Regressor** para futuras predicciones en la concentración de sílice, seguido por el **Gradient Boosting Forest**.

13. Resumen del proyecto

Una empresa minera solicitó realizar predicciones de la concentración de impurezas de sílice obtenidas a final del proceso concentrado de mineral de hierro. Los datos se extraen de Kaggle y se realizó un análisis exploratorio para identificar correlaciones y evaluar la calidad de los datos. Se probaron modelos de regresión lineales y también modelos basados en árboles, como Random Forest, Gradient Boosting y XGBoost.

Las métricas de evaluación usadas fueron Valor R², Error Absoluto Medio (MAE) y Error Cuadrático Medio (MSE)

Se comprobó la hipótesis ya que fue posible elegir y usar un modelo de aprendizaje automático basado en algoritmos de regresión, que considere todas las variables críticas del proceso de extracción de hierro para predecir con precisión el porcentaje de impurezas de sílice en el concentrado de mineral, permitiendo ajustes oportunos en el proceso para mejorar su eficiencia y sostenibilidad.

14. Conclusiones

Las predicciones del porciento de impurezas de sílice en la extracción de hierro se realizaron mediante técnicas de análisis avanzadas usando modelos de regresión lineal

El modelo seleccionado es XGB Regressor ya que destaca como el modelo más preciso y explicativo, el cual tiene un rendimiento de **R² de 0.9611**, lo que significa que **96.11%** de la variabilidad es explicada, una capacidad de ajuste excelente. Con un intervalo de confianza **estimado basado en MAE de ± 0.1815%**.

Los beneficios Operativos y Ambientales de realizar estas predicciones incluyen menor reprocesamiento y desperdicio. Reducción de desechos lo cual disminuye el impacto ambiental. Este modelo permitirá a los ingenieros anticiparse a problemas y tomar decisiones basadas en datos.

Bibliografía

- Magalhães, O. E. (2017). *Quality prediction in a mining process data*. Retrieved from Kaggle.com: <https://www.kaggle.com/datasets/edumagalhaes/quality-prediction-in-a-mining-process/data>
- Mohan, A. C. (2011). Web-Search Ranking with Initialized Gradient Boosted Regression Trees. *Proceedings of the Machine Learning Research*, 14:77-89 <https://proceedings.mlr.press/v14/mohan11a.html>.
- Montero, G. R. (2016). Modelos de regresión lineal múltiple. *Documentos de Trabajo en Economía Aplicada*. Universidad de Granada. España.
- Mrabet, E. Z. (2022). Random Forest Regressor-Based Approach for Detecting Fault Location and Duration in Power Systems Sensors. <https://doi.org/10.3390/s22020458>, 22(2), 458.
- Orallo, J. R. (2004). *Introducción a la minería de datos*. Madrid: Pearson Educación, S.A.
- Oyarzún, F. C. (2013). MODIFICACIONES DEL PROCESO DE FLOTACIÓN INVERSA DE HIERRO EN CELDAS NEUMÁTICAS DE PLANTA MAGNETITA . Chile.: PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAISO. FACULTAD DE INGENIERIA. ESCUELA DE INGENIERIA QUÍMICA .