Enunciado

Estudo de Caso: Estimando o Tempo Ideal de Fermentação com Apoio de IA

Após entender o impacto do tempo de fermentação sobre o rendimento de etanol, a equipe da **BioFermenta S.A.** enfrenta um novo desafio técnico: **prever, com antecedência, quantas horas serão necessárias para que cada batelada atinja sua eficiência máxima**.

Com a expansão da planta industrial e a entrada em operação de novos biorreatores, o planejamento da produção precisa ser mais preciso. Estimar o tempo ideal de fermentação permitirá evitar gargalos, desperdícios e paradas improdutivas — otimizando a escala e o uso dos tanques.

A equipe técnica acredita que o tempo de fermentação pode ser influenciado por uma série de variáveis operacionais, como:

- Temperatura de operação (°C)
- pH do meio
- Concentração inicial de glicose (g/L)
- Velocidade de agitação (rpm)
- Rendimento final obtido

Como **analista de dados do time de engenharia de processos**, sua missão é construir e comparar diferentes modelos preditivos, de forma estatisticamente fundamentada, para prever o tempo necessário de fermentação sob diferentes condições.

Sua missão:

- **Ajustar modelos de regressão linear múltipla**, considerando:
    - **Variável resposta**: Tempo de fermentação (h)
    - **Variáveis candidatas**: Temperatura (°C), pH, Glicose (g/L), Agitação (rpm), Rendimento Etanol (%)

- Aplicar **três métodos estatísticos diferentes de seleção de variáveis**, a fim de identificar os subconjuntos mais informativos:
    - Best Subset Selection
    - Stepwise Forward Selection
    - Backward Elimination

- **Comparar os modelos gerados**, discutindo:

- Quais variáveis apareceram em comum entre os métodos;
- Quais foram divergentes;
- Possíveis razões técnicas ou estatísticas para essas diferenças;
- Implicações práticas dessas escolhas na operação da planta.

Formato da Entrega: Diário de Bordo do Uso de IA

Sua entrega deve ser feita como um **diário de bordo**, relatando de forma honesta e técnica **como você usou ferramentas de inteligência artificial** (como o ChatGPT, por exemplo) para resolver este desafio.

Seu texto deve abordar:

- O **código inicial sugerido pela IA** e o que você entendeu dele;
- Quais **ajustes ou personalizações você fez** no código, com apoio da IA;
- Como você **esclareceu dúvidas conceituais** com a IA (ex: como funciona o stepwise? por que certas variáveis foram excluídas?);
- O que você **interpretou a partir dos resultados** — com foco em decisões aplicáveis ao contexto da BioFermenta;
- Se você **confirmou ou questionou** alguma sugestão da IA com base em conhecimentos de engenharia química;
- Qual método você julgou mais adequado e por quê.

> **Importante**: o foco não é apenas "gerar um código e colar a resposta", mas sim **narrar o processo de descoberta**, o uso crítico da inteligência artificial, e a conexão dos resultados com a realidade de um engenheiro de processos.

Resposta modelo

Após termos trabalhado no primeiro estudo de caso da BioFermenta S.A., agora nosso desafio foi outro: prever com precisão o tempo de fermentação necessário para cada batelada alcançar a eficiência ideal, considerando diversas condições operacionais.

Logo no início, me concentrei em entender a proposta do problema. A variável que eu deveria prever era o Tempo de Fermentação (h), e as variáveis explicativas seriam:

- Temperatura (°C)
- pH
- Glicose (g/L)
- Agitação (rpm)
- Rendimento Etanol (%)

A ideia era descobrir quais dessas variáveis realmente ajudam a prever o tempo de fermentação e, para isso, eu deveria usar três métodos de seleção de variáveis:

- Best Subset Selection
- Stepwise Forward Selection
- Backward Elimination

Pedi auxílio à IA para estruturar o código de tratamento dos dados. Como já havia usado um script no estudo anterior, aproveitei parte da lógica. O ChatGPT me ajudou a criar filtros de plausibilidade físico-química (pH entre 3,5 e 6, glicose até 300 g/L e tempo entre 12 e 48 horas), o que garantiu uma base mais consistente. Aqui já percebi a importância de não apenas confiar no código, mas também verificar se os filtros faziam sentido no ponto de vista da engenharia química.

```
import pandas as pd, numpy as np, itertools
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.base import clone
# Carregar base
df = pd.read_csv("base_fermentacao_suja_5000.csv")
# Limpar base
cols = ["Tempo (h)", "Temperatura (°C)", "pH", "Glicose (g/L)", "Agitação (rpm)",
"Rendimento Etanol (%)"]
for c in cols:
    df[c] = pd.to_numeric(df[c], errors="coerce")
df = df.replace([np.inf, -np.inf], np.nan)
df_limpa = df.dropna(subset=["Tempo (h)"])  # alvo não pode ser NaN
df_limpa = df_limpa[
    (df_limpa["pH"].between(3.5, 6.0)) &
    (df_limpa["Glicose (g/L)"] <= 300) &
    (df_limpa["Tempo (h)"].between(12, 48))
```

Com a base limpa, segui para a estruturação das variáveis. Criei uma lista de strings com os nomes das colunas do DataFrame que foram usadas como variáveis explicativas/preditoras.

```
features = ["Temperatura (°C)", "pH", "Glicose (g/L)", "Agitação (rpm)", "Rendimento Etanol (%)"] X = df_limpa[features].copy() y = df_limpa["Tempo (h)"].values
```

Depois disso, junto com a IA, dividi os dados em treinamento e teste para que evite que os dados tenham overfiting. O comando `test_size=0.3` diz que 30% dos dados serão usados para teste e os outros 70% para o treino. A divisão dos dados em treino e teste foi feita de forma aleatória, porém reprodutível, utilizando o parâmetro `random_state=42`. Isso garante que os mesmos subconjuntos sejam gerados em todas as execuções do código, assegurando consistência na comparação de resultados e facilitando a validação do experimento. Me questionei se utilizar o `random_state=42` não poderia mascarar o meu dado, uma vez que os resultados poderiam só fazer sentido para aquela base em específico. Porém, após algumas trocas ele me trouxe uma resposta que me deixou confiante em usar esse comando:

"Use random_state para testes e comparações reproduzíveis. Ele garante comparações justas entre modelos diferentes usando a mesma base de treino/teste.
Mas, para avaliar se o modelo é bom mesmo, use validação cruzada."

Ou seja, como irei usar essa mesma base de dados em outros modelos é bom que meu grupo de teste e treino seja o mesmo para uma comparação mais eficiente. Além disso, existem outras formas de contornar o problema de testar seu modelo em uma única divisão dos dados. Assim, a divisão ficou da seguinte forma:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Posteriormente montamos um bloco de código que tinha o objetivo de prever o tempo de fermentação. Para anular os possíveis problemas de overfiting discutidos anteriormente, usamos o comando `n_splits`. Configura a validação cruzada em 5 dobras: cada dado de treino serve de validação 1x e de treino 4x. Isso dá uma média de desempenho que é menos sensível a variações aleatórias.

```
cv = KFold(n_splits=5, shuffle=True, random_state=42)
```

Logo após, o chat GPT propôs um pipeline base:

```
base_est = Pipeline(steps=[
    ("imp_med", SimpleImputer(strategy="median")),
                ("imp_const",    SimpleImputer(strategy="constant",    fill_value=0, keep_empty_features=True)),
    ("scaler", StandardScaler(with_mean=True, with_std=True)),
    ("linreg", LinearRegression())
```

```
])
```

Esse Pipeline encadeia todas as etapas necessárias para treinar/testar o modelo, garantindo que o pré-processamento seja feito dentro de cada dobra:

- **imp_med:** substitui NaNs pela mediana.
- **imp_const:** se uma coluna inteira ficar NaN em uma dobra, substitui por 0 para não quebrar.
- **scaler:** padroniza (média 0, desvio 1).
- **linreg:** ajusta regressão linear múltipla (OLS).

```
def rmse_cv(cols):
    Xsub = X_train[cols]
    neg_mse = cross_val_score(
        base_est, Xsub, y_train,
        cv=cv, scoring="neg_mean_squared_error",
        error_score="raise"
    )
    return np.sqrt(-neg_mse).mean()
```

Esse trecho do código recebe um subconjunto de colunas (`cols`) para testar. Logo após ele roda o Pipeline com KFold, calculando o MSE em cada dobra e por fim retorna a média do RMSE (raiz do erro quadrático médio), que é uma métrica em “horas” — super interpretável para o problema. A última etapa desse código tem como objetivo não travar a execução caso uma coluna esteja mal formatada:

```
def try_rmse_cv(cols, verbose=False):
    try:
        val = rmse_cv(cols)
        if np.isnan(val) or np.isinf(val):
            raise ValueError("rmse_cv retornou NaN/Inf")
        return float(val)
    except Exception as e:
        if verbose:
            print(f"[WARN] Falha no CV para cols={cols}: {e}")
        return np.inf
```

Agora partimos de fato para a os três métodos de validação Best Subset Selection, Stepwise Forward Selection e Backward Elimination. Para o primeiro método é preciso testar todas as combinações possíveis de variáveis. Para cada subconjunto, ajusta o modelo e calcula um critério (no seu caso, o RMSE em validação cruzada). Ao final, ele escolhe o subconjunto com melhor desempenho.

```
best_subset_result = {"features": None, "rmse_cv": np.inf}
all_results_subset = []
for k in range(1, len(features)+1):
    for comb in itertools.combinations(features, k):
        score = try_rmse_cv(list(comb))
```

```
            all_results_subset.append((comb, score))
            if score < best_subset_result["rmse_cv"]:
                best_subset_result = {"features": list(comb), "rmse_cv": score}
    if best_subset_result["features"] is None:  # fallback
        singles = [(f, try_rmse_cv([f], verbose=True)) for f in features]
        singles.sort(key=lambda t: t[1])
            best_subset_result  =  {"features":  [singles[0][0]],  "rmse_cv":
singles[0][1]}
```

O laço `for k in range(1, len(features)+1)` define o tamanho dos subconjuntos a serem testados. O comando `itertools.combinations(features, k)` gera todas as combinações possíveis de variáveis com tamanho k. Para cada combinação, `score = try_rmse_cv(list(comb))` calcula o RMSE médio em validação cruzada. O comando `if score < best_subset_result["rmse_cv"]:` atualiza o melhor resultado encontrado até então. Por fim, `all_results_subset.append((comb, score))` armazena todas as combinações e seus erros para análise posterior.

Passando agora para o Stepwise Forward Selection, modelo no qual a cada passo testa-se incluir cada variável remanescente e adiciona-se apenas a que mais reduz o erro. O processo repete até que nenhuma nova variável traga ganho relevante, retornando um subconjunto parcimonioso. Esse foi o código gerado pela inteligência artificial:

```
remaining = set(features)
single_scores = [(f, try_rmse_cv([f])) for f in remaining]
single_scores.sort(key=lambda t: t[1])
forward_selected = [single_scores[0][0]]
best_score = single_scores[0][1]
remaining.remove(forward_selected[0])
improved = True
while improved and remaining:
    improved = False
    cand_best = None
    cand_score = best_score
    for f in list(remaining):
        cols = forward_selected + [f]
        score = try_rmse_cv(cols)
        if score < cand_score - 1e-6:
            cand_score, cand_best, improved = score, f, True
    if improved:
        forward_selected.append(cand_best)
        remaining.remove(cand_best)
        best_score = cand_score
forward_result = {"features": forward_selected, "rmse_cv": best_score}
```

Por fim, usamos o Backward Elimination, modelo que faz o oposto do Forward: começa com todas as variáveis incluídas no modelo. Em cada passo, testa-se a remoção de uma variável de cada vez e elimina-se a que menos prejudica (ou mais melhora) o desempenho. Esse processo continua até que retirar qualquer variável não traga ganho relevante, resultando em um subconjunto final.

```
current = features.copy()
```

```
best_score = try_rmse_cv(current)
while not np.isfinite(best_score) and len(current) > 1:
    candidates = []
    for f in current:
        cols = [c for c in current if c != f]
        candidates.append((cols, try_rmse_cv(cols)))
    candidates.sort(key=lambda t: t[1])
    current, best_score = candidates[0]

improved = True
while improved and len(current) > 1:
    improved = False
    best_drop = None
    cand_score = best_score
    for f in list(current):
        cols = [c for c in current if c != f]
        score = try_rmse_cv(cols)
        if score <= cand_score - 1e-6:
            cand_score, best_drop, improved = score, f, True
    if improved and best_drop:
        current.remove(best_drop)
        best_score = cand_score

backward_result = {"features": current, "rmse_cv": best_score}
```

Os três métodos de seleção de variáveis foram aplicados e apresentaram respostas bastante próximas. O Best Subset Selection indicou como melhor combinação todas as variáveis do estudo — Temperatura (°C), pH, Glicose (g/L), Agitação (rpm) e Rendimento Etanol (%) — com RMSE médio de aproximadamente 5,82 horas. O Forward Selection, apesar de seguir um processo incremental, também convergiu para praticamente o mesmo conjunto, começando pelo Rendimento de Etanol e adicionando sucessivamente Temperatura, Glicose, Agitação e pH, alcançando igualmente um RMSE de 5,82 horas. Por fim, o Backward Elimination, que parte do modelo completo e vai removendo variáveis irrelevantes, manteve todas as cinco variáveis iniciais, chegando ao mesmo erro médio de 5,82 horas.

Esse alinhamento entre os métodos sugere que todas as variáveis analisadas são relevantes para explicar o tempo de fermentação, e que nenhuma delas pôde ser descartada sem perda de qualidade preditiva. Do ponto de vista prático, isso reforça a importância de monitorar simultaneamente Temperatura, pH, Glicose, Agitação e Rendimento de Etanol na operação da BioFermenta, já que o comportamento do processo fermentativo depende da interação entre todos esses parâmetros. A convergência dos resultados ainda dá robustez ao modelo, transmitindo maior segurança para sua utilização no planejamento do tempo ideal de bateladas.

Na etapa final, foi criada a função `fit_and_eval`, responsável por treinar e avaliar cada modelo selecionado nos conjuntos de treino e teste. Essa função recebe como entrada o subconjunto de variáveis escolhido por cada método (Best Subset, Forward e Backward) e

retorna as métricas de desempenho mais relevantes: RMSE (Root Mean Squared Error), MAE (Mean Absolute Error) e R^2 (coeficiente de determinação).

Com base nisso, foram avaliados os três modelos finais: `res_best`, `res_forward` e `res_backward`. Em seguida, esses resultados foram organizados em um DataFrame comparativo, ordenado pelo menor RMSE no conjunto de teste. Esse passo permitiu não apenas comparar diretamente os métodos, mas também verificar a consistência entre eles.

Por fim, o código identifica o melhor modelo de acordo com a métrica de RMSE, treina-o novamente com as variáveis correspondentes e calcula os resíduos (diferença entre valores reais e preditos). Dessa forma, foi possível concluir não só qual método obteve o melhor ajuste, mas também avaliar a qualidade preditiva de maneira mais detalhada, conferindo robustez ao processo de seleção de variáveis.

```
def fit_and_eval(cols, name):
    assert cols and len(cols) > 0, "Lista de variáveis está vazia."
    est = clone(base_est)
    est.fit(X_train[cols], y_train)
    pred = est.predict(X_test[cols])

    mse = mean_squared_error(y_test, pred)     # sem 'squared'
    rmse = float(mse**0.5)                     # raiz para virar RMSE
    return {
        "modelo": name,
        "variaveis": cols,
        "RMSE_test": rmse,
        "MAE_test": mean_absolute_error(y_test, pred),
        "R2_test": r2_score(y_test, pred),
    }
res_best = fit_and_eval(best_subset_result["features"], "Best Subset")
res_forward = fit_and_eval(forward_result["features"], "Forward")
res_backward = fit_and_eval(backward_result["features"], "Backward")

comparacao = pd.DataFrame([res_best, res_forward,
res_backward]).sort_values("RMSE_test")
print("\n=== Comparação (ordem por RMSE_test) ===")
print(comparacao[["modelo", "RMSE_test", "MAE_test", "R2_test", "variaveis"]])
melhor = comparacao.iloc[0]
cols_melhor = melhor["variaveis"]
est_melhor = clone(base_est).fit(X_train[cols_melhor], y_train)
y_pred = est_melhor.predict(X_test[cols_melhor])
residuos = y_test - y_pred
print(f"\nModelo vencedor: {melhor['modelo']} | Variáveis: {cols_melhor}")
print(f"RMSE={melhor['RMSE_test']:.2f} | MAE={mean_squared_error(y_test,
y_pred)**0.5:.2f} | R2={r2_score(y_test, y_pred):.3f}")
```

Após rodar os métodos de seleção e avaliação, percebi que o código não chegava a mostrar a função de regressão linear múltipla ajustada, limitando-se apenas às métricas de desempenho (RMSE, MAE e R^2). Diante disso, solicitei ao ChatGPT um ajuste no código para que, além da comparação entre os modelos, fosse possível extrair e escrever a

equação final do modelo, explicitando o intercepto e os coeficientes associados a cada variável na escala original. Essa modificação permitiu visualizar a função completa que relaciona o tempo de fermentação com os parâmetros operacionais, tornando os resultados mais interpretáveis e aplicáveis ao contexto da BioFermenta.

A equação final ajustada foi:

$$\text{Tempo (h)} = 10{,}908 - 0{,}344 \cdot \text{Temperatura (°C)} - 0{,}368 \cdot \text{pH} - 0{,}027 \cdot \text{Glicose (g/L)} - 0{,}010 \cdot \text{Agitação (rpm)} + 0{,}925 \cdot \text{Rendimento Etanol (\%)}$$

Essa representação deixa claro o peso de cada variável no modelo: enquanto temperatura, pH, glicose e agitação apresentam coeficientes negativos (indicando efeito redutor sobre o tempo), o rendimento de etanol tem coeficiente positivo, refletindo sua contribuição direta para estimar a duração da fermentação.

Ao final, podemos concluir que o estudo demonstrou que o uso de inteligência artificial foi essencial para estruturar a análise preditiva do tempo de fermentação, tanto na programação dos métodos estatísticos quanto no suporte conceitual. A aplicação dos três métodos de seleção de variáveis — Best Subset Selection, Stepwise Forward Selection e Backward Elimination — revelou resultados idênticos em termos de desempenho: todos apontaram para o uso conjunto das cinco variáveis (Temperatura, pH, Glicose, Agitação e Rendimento de Etanol), com valores médios de RMSE ≈ 5,88 horas, MAE ≈ 4,73 horas e R^2 ≈ 0,68. Esses números indicam um modelo com bom poder preditivo, capaz de reduzir incertezas na estimativa da duração das bateladas.

A interpretação dos coeficientes reforça a lógica do processo: aumentos de temperatura, pH, glicose e agitação tendem a reduzir o tempo de fermentação, enquanto maiores rendimentos de etanol estão associados a durações mais longas, coerente com a necessidade de maturação do processo para alcançar maior eficiência.

Do ponto de vista prático, a convergência entre os métodos dá segurança para afirmar que todas as variáveis são relevantes e devem ser monitoradas de forma integrada. A equação obtida permite não apenas prever o tempo ideal de fermentação com base em condições operacionais reais, mas também subsidiar ajustes de processo, otimizar a utilização dos biorreatores e reduzir gargalos produtivos. Entre os métodos utilizados, o Best Subset Selection se destaca pela busca exaustiva, mas a coincidência de resultados nos três critérios reforça a robustez do modelo final. Assim, a integração entre análise estatística, inteligência artificial e conhecimento de engenharia de processos se confirma como um caminho promissor para ampliar a eficiência e a confiabilidade na operação industrial da BioFermenta S.A.

Link para o código no colab

classificacao_binária_etanol.ipynb