TECH CHALLENGE FIAP – Storytelling Técnico com Código Integrado

Sumário

Objetivo do Projeto

Neste projeto, atuamos como cientistas de dados responsáveis por desenvolver um modelo preditivo capaz de indicar se o índice IBOVESPA encerrará o próximo pregão em alta ou baixa. Com base em dados históricos e em atributos derivados, o objetivo é construir um modelo com acurácia mínima de 75% e aplicabilidade prática em dashboards de apoio à decisão de um fundo de investimentos brasileiro.

Etapa 1 – Análise Exploratória dos Dados (EDA)

Configuração Inicial e Carregamento de Dados

Começamos garantindo que o diretório de trabalho estivesse corretamente configurado e carregamos as bibliotecas necessárias para manipulação de dados e visualizações. Em seguida, conectamos ao banco local DuckDB e carregamos a tabela com os preços históricos de ativos relevantes ao mercado brasileiro.

```
import os
if os.path.basename(os.getcwd()) == 'notebooks':
    os.chdir('..')
print(f"Diretório de Trabalho Atual: {os.getcwd()}")

import src.config as config
import pandas as pd
import numpy as np
import duckdb
import pandas_ta as ta
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
%matplotlib inline
db_path = str(config.DB_PATH)
con = duckdb.connect(database=db_path, read_only=True)
df = con.execute("SELECT * FROM precos_diarios").fetchdf()
con.close()
```

Ajuste do Índice Temporal

Convertendo a coluna `data` para o tipo datetime e configurando-a como índice, garantimos que todas as análises posteriores sigam uma linha cronológica adequada para séries temporais.

```
df['data'] = pd.to_datetime(df['data'])
df.set_index('data', inplace=True)
display(df.head())
```

Visualização das Séries de Preço

Selecionamos os dados de fechamento de cada ativo para os últimos dois anos, visualizando as séries temporais em subgráficos individuais para facilitar a análise do comportamento dos ativos ao longo do tempo.

```
start_date_2y = '2022-07-05'
end_date_2y = '2025-07-05'
df_close = df.filter(like='close')
df_close_2y = df_close.loc[start_date_2y:end_date_2y]
fig, axes = plt.subplots(nrows=len(df_close_2y.columns), ncols=1,
figsize=(15, 20), sharex=True)
fig.suptitle(f'Séries Temporais dos Preços de Fechamento
({start_date_2y} a {end_date_2y})', fontsize=20, y=0.92)
for i, column in enumerate(df_close_2y.columns):
    ax = axes[i]
    df_close_2y[column].plot(ax=ax, legend=False)
    ax.set_title(column, fontsize=14)
    ax.set_ylabel('Preço')
plt.xlabel('Data')
plt.tight_layout(rect=[0, 0.03, 1, 0.9])
plt.show()
```

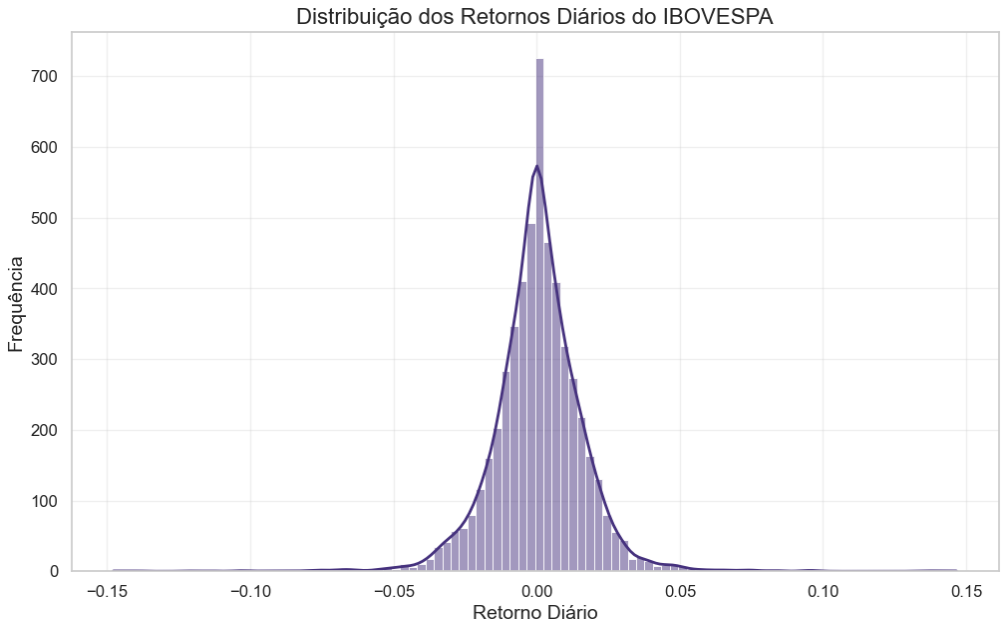
Series Temporais dos Preços de fechamento

Análise Estatística dos Retornos Diários

Calculamos os retornos diários percentuais do IBOVESPA e observamos uma distribuição leptocúrtica, ou seja, eventos extremos são mais comuns de acontecer. Isso sugere que o modelo preditivo deve ser robusto para não ser excessivamente influenciado por esses “dias atípicos”, que são uma característica inerente do mercado. Esse comportamento reforçou a decisão de trabalhar com retornos e utilizar thresholds para definição de eventos significativos.

```
df_returns = df.filter(like='close').pct_change().dropna()
plt.figure(figsize=(12, 7))
sns.histplot(df_returns['close_ibovespa'], kde=True, bins=100)
plt.title('Distribuição dos Retornos Diários do IBOVESPA')
plt.xlabel('Retorno Diário')
plt.ylabel('Frequência')
plt.show()
```

Distribuição dos Retornos Diários do IBOVESPA



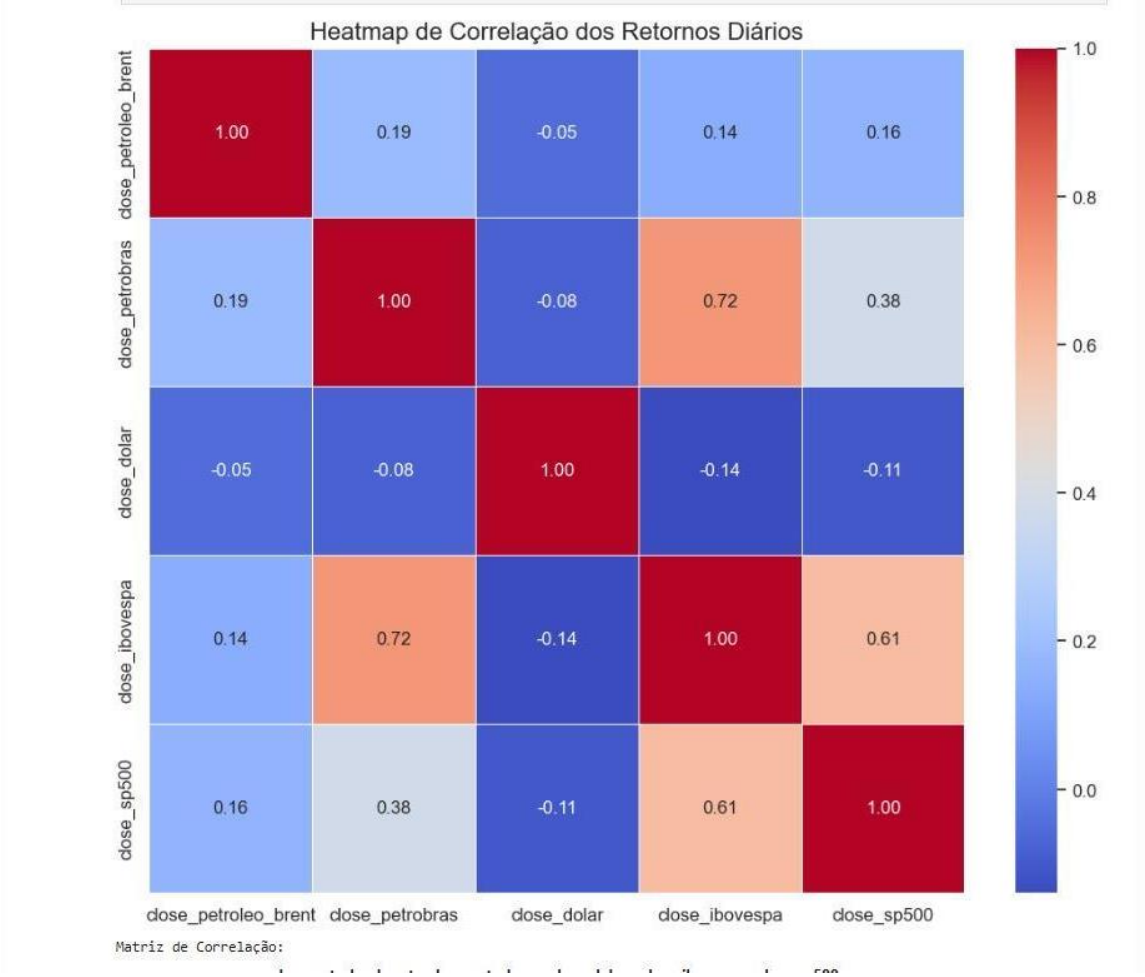
Correlação entre Ativos

Análises de correlação revelaram forte relação entre IBOVESPA e Petrobras, além de correlações positivas mais sutis com S&P 500. Enquanto o dólar mostrou, ainda que fraca, correlação negativa. Como também destacou correlação quase nula entre a abertura e o fechamento do IBOVESPA, indicando que o preço de abertura não antecipa o comportamento do fechamento do mesmo dia.

A engenharia de features considerará essas relações na construção de variáveis explicativas defasadas e compostas:

```
correlation_matrix = df_returns.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=.5)
plt.title('Heatmap de Correlação dos Retornos Diários')
plt.show()
```

Heatmap de Correlação dos Retornos Diários (OHLC de Todos os Ativos) 2023-2025



Comparativo de Correlação: Histórico vs. Período Recente

A comparação entre o histórico e os últimos 2 anos demonstra ser mais vantajoso dar mais peso aos dados recentes, já que os níveis de influência dos drivers de mercado mudaram. S&P 500 e Petrobras continuam relevantes, mas com influência menor do que o histórico completo indicava, abertura do IBOVESPA segue com correlação quase que nula.

Ativo	Histórico Completo	Últimos 2 Anos
Petrobras	+0.75	+0.44
S&P 500	+0.50	+0.39
Dólar	-0.10	-0.08

Abertura IBOVESPA	-0.05	-0.02

Engenharia de Atributos

Target com Threshold

A série `close_ibovespa` mostrou ruído em torno de variações neutras. Para tornar o target mais robusto, adotamos um threshold de 0.5%, considerando como 'alta' apenas retornos acima desse limite no dia seguinte.

```
THRESHOLD_ALTA = 0.005
retorno_seguinte = (df['close_ibovespa'].shift(-1) /
df['close_ibovespa']) - 1
df['target'] = (retorno_seguinte > THRESHOLD_ALTA).astype(int)
df.dropna(subset=['target'], inplace=True)
```

Enriquecimento da Base com Dados

Dando seguimento à engenharia de atributos, enriquecemos a base de dados com diversas features temporais utilizando os preços dos fechamentos. Gerando (1) Features de Retorno Acumulado (Momentum), olhando 2, 5, 10 e 21 dias. (2) Features de Retorno Diário Defasado (Lag), obtendo os valores dos fechamentos de dias anteriores 1, 2, 3 e 5 dias.

```
colunas_close = [col for col in df.columns if col.startswith('close_')]

periodos_momentum = [2, 5, 10, 21]
for col in colunas_close:
    nome_base = col.replace('close_', '')
    for p in periodos_momentum:
        df[f'{nome_base}_ret_acum_{p}d'] =
df[col].pct_change(periods=p)


df_retornos_diarios = df[colunas_close].pct_change(periods=1)


df_retornos_diarios.columns = [
    f'{col.replace("close_", "")}_ret_diario' for col in colunas_close]
periodos_de_lag = [1, 2, 3, 5]
colunas_para_lag = df_retornos_diarios.columns.tolist()


df_lagged = pd.DataFrame(index=df.index)
for lag in periodos_de_lag:
    for col in colunas_para_lag:
```

```
        df_lagged[f'{col}_lag_{lag}'] =
df_retornos_diarios[col].shift(lag)


df = pd.concat([df, df_retornos_diarios, df_lagged], axis=1)

df.dropna(inplace=True)
```

Sinais de Mercado - Indicadores

Aprimorando a fonte de dados para a modelagem adicionamos indicadores na tabela principal, sendo: (1) Indicadores Intradiários (Candle Features), medindo o salto entre o fechamento anterior e a abertura atual (gap), a força e direção do movimento (body) e a volatilidade do dia (range). (2) Indicadores Globais Defasados (Efeito Overnight), olhando "ret_sp500_ontem", S&P 500 do dia anterior e o "ret_dolar_ontem", variação do dólar na véspera. (3) Indicadores Técnicos, RSI - mede o momentum, MACD - capta o início e o fim de tendências, Bandas de Bollinger - mede volatilidade e extremos de preço. (4) spread_petro_ibov - indica a performance da Petrobras vs. Ibovespa, ret_diff_dolar_ibov - diferença entre retornos do dólar e do Ibovespa. (5) Variáveis Temporais Categóricas - dias da semana e o mês.

```
df.index = pd.to_datetime(df.index)
df['gap_open_close_ibov'] = df['open_ibovespa'] -
df['close_ibovespa'].shift(1)
df['candle_body_ibov'] = df['close_ibovespa'] - df['open_ibovespa']
df['candle_range_ibov'] = df['high_ibovespa'] - df['low_ibovespa']

df['ret_sp500_ontem'] = df['close_sp500'].pct_change().shift(1)
df['ret_dolar_ontem'] = df['close_dolar'].pct_change().shift(1)

df.ta.bbands(close=df['close_ibovespa'], append=True)

df.rename(columns={
    'BBL_20_2.0': 'bb_lower_ibov',
    'BBM_20_2.0': 'bb_middle_ibov',
    'BBU_20_2.0': 'bb_upper_ibov',
    'BBP_20_2.0': 'bb_percent_ibov',
    'BBB_20_2.0': 'bb_width_ibov',
    'RSI_14': 'rsi_ibov',     'MACD_12_26_9': 'macd_ibov',
    'MACDh_12_26_9': 'macd_hist_ibov',
    'MACDs_12_26_9': 'macd_signal_ibov'
}, inplace=True)

df['spread_petro_ibov'] = df['close_petrobras'] / df['close_ibovespa']

df['ret_dolar_1d'] = df['close_dolar'].pct_change(1)
df['ret_ibov_1d'] = df['close_ibovespa'].pct_change(1)
df['ret_diff_dolar_ibov'] = df['ret_dolar_1d'] - df['ret_ibov_1d']

df['dia_da_semana'] = df.index.dayofweek
```

```
df['mes'] = df.index.month

df.dropna(inplace=True)
```

Conclusão da Etapa de EDA

A análise exploratória forneceu insights essenciais para o entendimento do comportamento do IBOVESPA, revelando relações dinâmicas com outros ativos, variações estruturais ao longo do tempo e padrões de volatilidade. Estes resultados fundamentam a próxima fase do projeto: a construção do modelo preditivo.

Etapa 2: Modelagem Preditiva Baseline

Ajuste do Diretório de Trabalho

Este trecho garante que o notebook esteja sendo executado a partir do diretório raiz do projeto, corrigindo o caminho relativo.

```
import os
   if os.path.basename(os.getcwd()) == 'notebooks':
        os.chdir('..')
```

Carregamento das Bibliotecas para a Fase 3: Modelagem Preditiva

Importação das bibliotecas essenciais para manipulação de dados, modelagem, avaliação e visualização.

```
import duckdb
import pandas as pd
import numpy as np
import lightgbm as lgb
import shap
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
classification_report, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.inspection import permutation_importance
import src.config as config
from IPython.display import display
sns.set_theme(style='whitegrid', palette='viridis')
plt.style.use("fivethirtyeight")
%matplotlib inline
```

Carregamento dos Dados para Modelagem

Conecta ao banco DuckDB, carrega os dados da tabela 'features_baseline' e define a coluna 'data' como índice.

```
try:
     con = duckdb.connect(database=str(config.DB_PATH),
read_only=True)
     df_model = con.execute("SELECT * FROM
features_baseline").fetchdf()
     con.close()
     df_model['data'] = pd.to_datetime(df_model['data'])
     df_model.set_index('data', inplace=True)
```

Separação entre Features e Variável Alvo

Separação dos dados em X (features) e y (variável alvo) para posterior treinamento do modelo.

```
X = df_model.drop(columns=['target'])
y = df_model['target']
```

Divisão Temporal dos Dados: Treino vs. Teste

Divisão temporal utilizando os últimos 6 meses como conjunto de teste, simulando uma previsão realista e o restante dos dados para treino.

```
ponto_de_corte = X.index.max() - pd.DateOffset(months=6)
X_treino = X[X.index < ponto_de_corte]
X_teste = X[X.index >= ponto_de_corte]
y_treino = y.loc[X_treino.index]
y_teste = y.loc[X_teste.index]
```

Seleção de Atributos com Permutation Importance (LightGBM)

A técnica de Permutation Importance embaralha os valores das features no conjunto de teste para avaliar o impacto de cada variável na performance do modelo. Assim, selecionamos as 20 features mais relevantes para o modelo.

```
lgbm_baseline = lgb.LGBMClassifier(random_state=42, verbosity=-
1)
lgbm_baseline.fit(X_treino, y_treino)

result = permutation_importance(
    estimator=lgbm_baseline,
    X=X_teste,
    y=y_teste,
    n_repeats=10,
    scoring='roc_auc',
    random_state=42,
    n_jobs=-1
)
df_importancia = pd.DataFrame(
    {'feature': X_treino.columns, 'importance_mean':
result.importances_mean}
)
df_importancia = df_importancia.sort_values(
    by='importance_mean', ascending=False)
plt.figure(figsize=(12, 10))
sns.barplot(
    x='importance_mean',
    y='feature',
```

```
    data=df_importancia.head(20)
)
plt.title('Top 20 Features Mais Importantes (Permutation
Importance)')
plt.xlabel('Redução Média na Performance (AUC)')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

Aplicação da Seleção de Features no Conjunto de Dados

Filtramos os conjuntos de treino e teste para conter apenas as variáveis selecionadas anteriormente.

```
X_treino_selecionado = X_treino[features_selecionadas]
X_teste_selecionado = X_teste[features_selecionadas]
```

Modelo A – Visão de Longo Prazo

O Modelo A foi treinado com todo o histórico disponível, buscando capturar padrões de longo prazo. Avaliamos a performance com métricas padrão de classificação e AUC ROC, resultando em acurácia de 53% e AUC ROC 0.5822.

```
lgbm_longo_prazo = lgb.LGBMClassifier(random_state=42,
verbosity=-1)
lgbm_longo_prazo.fit(X_treino_selecionado, y_treino)
y_pred_longo_prazo =
lgbm_longo_prazo.predict(X_teste_selecionado)
y_proba_longo_prazo =
lgbm_longo_prazo.predict_proba(X_teste_selecionado)[
    :, 1]

print("\nResultados do Modelo A (Longo Prazo):")
print(classification_report(
    y_teste, y_pred_longo_prazo, target_names=['Baixa',
'Alta']))
print(f"AUC ROC: {roc_auc_score(y_teste,
y_proba_longo_prazo):.4f}")
```

Modelo B – Visão Recente (Últimos 6 Anos)

O Modelo B foi treinado com dados mais recentes (últimos 6 anos), testando a hipótese de que o comportamento recente é mais relevante para previsões. Resultando em acurácia de 58% e AUC ROC 0.5996.

```
ponto_de_corte_recente = X_treino_selecionado.index.max() -
pd.DateOffset(years=6)
X_treino_recente =
X_treino_selecionado[X_treino_selecionado.index >=
                                          ponto_de_corte_recente]
y_treino_recente = y_treino.loc[X_treino_recente.index]

lgbm_recente = lgb.LGBMClassifier(random_state=42, verbosity=-1)
lgbm_recente.fit(X_treino_recente, y_treino_recente)

y_pred_recente = lgbm_recente.predict(X_teste_selecionado)
y_proba_recente =
lgbm_recente.predict_proba(X_teste_selecionado)[:, 1]

print("\nResultados do Modelo B (Recente):")
print(classification_report(
    y_teste, y_pred_recente, target_names=['Baixa', 'Alta']))
print(f"AUC ROC: {roc_auc_score(y_teste, y_proba_recente):.4f}")
```

O Modelo B será considerado o novo baseline para as próximas etapas, como tuning de hiperparâmetros e validação com diferentes janelas temporais.

Etapa 3: Otimização do Modelo

A metodologia empregada envolve o uso de **Machine Learning supervisionado**, especificamente o framework **LightGBM**, para prever movimentos futuros do mercado. Um ponto chave é a transformação do problema de previsão de um alvo binário para um **alvo multiclasse com três categorias**: Alta Significativa (+0.5%), Baixa Significativa (-0.5%) e Neutro (entre -0.5% e +0.5%), o que permite uma análise mais refinada dos movimentos do mercado.

O processo inclui a **preparação cuidadosa dos dados**, com a divisão temporal em conjuntos de treino e teste, e a **otimização dos hiperparâmetros do modelo** utilizando a biblioteca **Optuna** e validação cruzada temporal. A fase final consiste no treinamento do modelo com os parâmetros otimizados e a avaliação de sua performance através de métricas como **AUC ROC**, relatório de classificação e matriz de confusão.

Criação da Variável Alvo Multiclasse

Esta seção descreve a **transformação da variável alvo** de um formato binário (provavelmente indicando apenas "alta" ou "baixa") para um **formato multiclasse com três categorias**. Essa mudança visa permitir que o modelo diferencie entre movimentos de mercado mais significativos e flutuações diárias menos importantes, tornando as previsões mais úteis para aplicações práticas. Para isso definimos um threshold, calculamos o retorno do dia seguinte e criamos a variável alvo.

```
threshold = 0.005
retorno_futuro = df_model['close_ibovespa'].pct_change().shift(-1)
df_model['alvo_multiclasse'] = np.where(
    retorno_futuro > threshold,
    1,
    np.where(
        retorno_futuro < -threshold,
        -1,
        0
    )
)
df_model.dropna(subset=['alvo_multiclasse'], inplace=True)
# --- Verificação ---
print("\nDistribuição do nosso novo alvo multiclasse (em %):")
print(df_model['alvo_multiclasse'].value_counts(
    normalize=True).sort_index().map('{:.2%}'.format))
print("\nExibindo as últimas linhas com o novo alvo para validação
manual:")
display(df_model[['close_ibovespa', 'alvo_multiclasse']].tail(10))
```

Preparação dos Dados para o Modelo Multiclasse

Após a criação da variável alvo multiclasse, esta seção se concentra em **preparar os dados para o processo de modelagem**. Isso envolve a separação entre as *features* (variáveis de entrada) e o alvo (variável a ser prevista), bem como a divisão temporal dos dados em conjuntos de treino e teste. A verificação da distribuição das classes em ambos os conjuntos também é realizada para garantir a representatividade.

Explicação do Código: O código executa três etapas principais de preparação dos dados: (1) Separação entre Features (X) e Alvo (y), (2) Divisão Temporal: Treino vs. Teste. (3) Verificação da Distribuição das Classes:

```
X = df_model.drop(columns=['alvo_multiclasse', 'alvo'],
errors='ignore')
y = df_model['alvo_multiclasse']

ponto_de_corte = X.index.max() - pd.DateOffset(months=6)
X_treino = X[X.index < ponto_de_corte]
X_teste = X[X.index >= ponto_de_corte]
y_treino = y.loc[X_treino.index]
y_teste = y.loc[X_teste.index]
```

Otimização de Hiperparâmetros com Optuna (Modelo Multiclasse)

Esta etapa é dedicada à **otimização dos hiperparâmetros** do modelo LightGBM usando a biblioteca **Optuna**. O objetivo é encontrar a combinação ideal de parâmetros que maximize a métrica AUC (Area Under the Receiver Operating Characteristic Curve), especificamente a versão One-vs-Rest para classificação multiclasse, em uma validação cruzada temporal. Assim, definimos a estratégia de validação cruzada, definimos a Função Objetivo (executada para treinar e avaliar um modelo e retornar uma métrica de desempenho).

```
tscv = TimeSeriesSplit(n_splits=5)

def objective(trial):
    params = {
        'objective': 'multiclass',
        'metric': 'multi_logloss',
        'num_class': 3,
        'n_estimators': trial.suggest_int('n_estimators', 200, 1000),
        'learning_rate': trial.suggest_float('learning_rate', 0.01,
0.3),
        'num_leaves': trial.suggest_int('num_leaves', 20, 300),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.0, 1.0),
```

```
            'reg_lambda': trial.suggest_float('reg_lambda', 0.0, 1.0),
            'random_state': 42,
            'verbosity': -1,
            'n_jobs': -1
        }
        scores = []
        for train_index, val_index in tscv.split(X_treino):
            X_train_fold, X_val_fold = X_treino.iloc[train_index],
X_treino.iloc[val_index]
            y_train_fold, y_val_fold = y_treino.iloc[train_index],
y_treino.iloc[val_index]
            model = lgb.LGBMClassifier(**params)
            model.fit(X_train_fold, y_train_fold)
            y_proba = model.predict_proba(X_val_fold)
            score = roc_auc_score(y_val_fold, y_proba, multi_class='ovr')
            scores.append(score)
        return np.mean(scores)

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=100)
best_params = study.best_params
```

Fase Final: Treinamento do Modelo Multiclasse Otimizado com LightGBM

Esta é a **etapa final do processo de modelagem**, onde o modelo LightGBM é treinado usando os hiperparâmetros que foram otimizados pela biblioteca Optuna. O objetivo é criar um modelo robusto capaz de prever os movimentos do Ibovespa nas três classes definidas: Alta Significativa (1), Neutra (0) e Baixa Significativa (-1). Após o treinamento, o modelo é avaliado no conjunto de teste, utilizando diversas métricas para uma visão completa de sua performance.

```
final_params = best_params.copy()
final_params.update({
    'objective': 'multiclass',
    'metric': 'multi_logloss',
    'num_class': 3,
    'random_state': 42,
    'verbosity': -1
})
modelo_otimizado = lgb.LGBMClassifier(**final_params)

ponto_de_corte_3a = X_treino.index.max() - pd.DateOffset(years=3)
X_treino_final = X_treino[X_treino.index >= ponto_de_corte_3a]
y_treino_final = y_treino.loc[X_treino_final.index]
print(
    f"Usando {len(X_treino_final)} amostras da janela de 3 anos para o
treino final.")

modelo_otimizado.fit(X_treino_final, y_treino_final)

y_pred_otimizado = modelo_otimizado.predict(X_teste)
y_proba_otimizado = modelo_otimizado.predict_proba(X_teste)
```

```
print(classification_report(y_teste, y_pred_otimizado,
      labels=[-1, 0, 1], target_names=['Baixa Sig.', 'Neutra', 'Alta
Sig.']))
final_auc = roc_auc_score(y_teste, y_proba_otimizado,
                          multi_class='ovr', labels=[-1, 0, 1])

cm = confusion_matrix(y_teste, y_pred_otimizado, labels=[-1, 0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[
                                'Baixa Sig.', 'Neutra', 'Alta Sig.'])
fig, ax = plt.subplots(figsize=(8, 6))
disp.plot(ax=ax, cmap=plt.cm.Blues)
plt.title("Matriz de Confusão do Modelo (Padrão)")
plt.show()

df_probas = pd.DataFrame(y_proba_otimizado, columns=[
                          'proba_baixa', 'proba_neutra', 'proba_alta'])
df_probas['real'] = y_teste
df_probas['pred_default'] = y_pred_otimizado

novo_threshold_alta = 0.40
df_probas['pred_custom_alta'] = np.where(
    df_probas['proba_alta'] > novo_threshold_alta, 1, 0)
```

Etapa 4: Análise exploratória features

Objetivo da Etapa

Esta etapa foi construída com a intenção de gerar, de forma automatizada, indicadores técnicos de desvio do preço em relação às médias móveis (osciladores logarítmicos) para o IBOVESPA e para o Dólar, considerando múltiplas janelas temporais (2 a 21 dias). O raciocínio por trás disso é que, ao comparar o preço atual com a sua média recente em diferentes horizontes, é possível capturar a intensidade e a direção da tendência de curto e médio prazo dos ativos, informações fundamentais para análises preditivas. Utilizar a razão logarítmica entre o preço e a média torna as variáveis mais estáveis e simétricas, facilitando o uso em modelos estatísticos. Ao final, o código remove as primeiras linhas da base que contêm valores ausentes (NaNs) causados pelas janelas de cálculo, garantindo que o conjunto esteja pronto para análises posteriores.

Médias Móveis

```
janelas_ma = [2, 5, 7, 10, 14, 21]
for periodo in janelas_ma:
    ma_ibov_col = f'ibov_ma{periodo}'
    osc_ibov_col = f'ibov_vs_ma{periodo}'
    df[ma_ibov_col] =
df['close_ibovespa'].rolling(window=periodo).mean()
    df[osc_ibov_col] = np.log(df['close_ibovespa'] / df[ma_ibov_col])
    ma_dolar_col = f'dolar_ma{periodo}'
    osc_dolar_col = f'dolar_vs_ma{periodo}'
    df[ma_dolar_col] = df['close_dolar'].rolling(window=periodo).mean()
    df[osc_dolar_col] = np.log(df['close_dolar'] / df[ma_dolar_col])
df = df.dropna()
```

Gráfico de Médias Móveis

Com o trecho abaixo o nosso objetivo é visualizar graficamente a evolução das médias móveis. A linha de raciocínio adotada consiste, primeiro, em filtrar o dataset original para esse intervalo de datas, garantindo que apenas os dados mais recentes sejam considerados. Em seguida, são selecionadas automaticamente todas as colunas que representam médias móveis (identificadas pelo padrão ‘ibov_ma’), e é feita uma verificação de segurança para garantir que essas colunas existam. Por fim, o código gera múltiplos gráficos, um para cada média móvel, organizados verticalmente, o que permite uma análise visual clara e comparativa da trajetória de cada uma dessas médias no tempo. Essa visualização ajuda a identificar mudanças de tendência, pontos

de cruzamento e estabilidade do índice ao longo do tempo, o que é fundamental para análises técnicas e validação das features criadas

```
start_date_2y = '2023-07-05'
end_date_2y = '2025-07-05'

df_2y = df.loc[start_date_2y:end_date_2y]
df_ma = df_2y.filter(like='ibov_ma')

if df_ma.empty:
    raise ValueError("Nenhuma coluna de média móvel encontrada.
Verifique os nomes das colunas.")
num_plots = len(df_ma.columns)
fig, axes = plt.subplots(
    nrows=num_plots,
    ncols=1,
    figsize=(15, 5 * num_plots),
    sharex=True
)
if num_plots == 1:
    axes = [axes]
fig.suptitle(f'Médias Móveis ({start_date_2y} a {end_date_2y})',
fontsize=16, y=0.95)
for i, column in enumerate(df_ma.columns):
    ax = axes[i]
    df_ma[column].plot(ax=ax, legend=False, color='blue',
linewidth=1.5)
    ax.set_title(column.upper(), fontsize=12)
    ax.set_ylabel('Preço', fontsize=10)
    ax.grid(True, linestyle='--', alpha=0.6)
plt.xlabel('Data', fontsize=12)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

Considerações Técnicas e Justificativas

- O uso de médias móveis é uma prática consolidada na análise técnica de mercados financeiros. Elas atuam como indicadores de tendência e podem ser utilizadas como variáveis importantes no treinamento de modelos preditivos.
- Os osciladores logarítmicos ajudam a tratar a não estacionariedade da série de forma mais robusta do que simples subtrações.
- A escolha das janelas múltiplas (2 a 21 dias) permite ao modelo ter um olhar granular e também macro sobre os movimentos do ativo.

Suavização exponencial

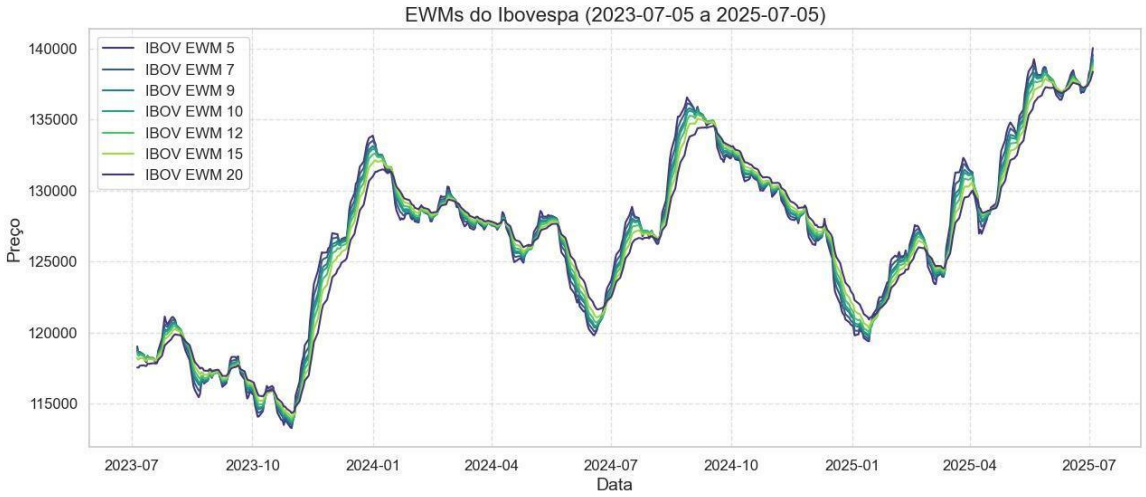
Esta etapa possui o objetivo de visualizar diferentes médias móveis exponenciais (EWM) do fechamento do IBOVESPA ao longo de um período de dois anos, com o objetivo de capturar tendências recentes com maior sensibilidade às variações mais recentes dos preços. As médias exponenciais são utilizadas porque dão maior peso aos dados mais novos, o que é útil para identificar mudanças rápidas no comportamento do mercado. Após o cálculo das EWMs para diferentes janelas, o código filtra o intervalo de análise e organiza os dados em subgráficos individuais para facilitar a comparação visual entre as curvas geradas. Isso permite uma avaliação clara da suavização de tendências em diferentes horizontes e apoia decisões sobre quais janelas de tempo melhor representam os movimentos relevantes para o modelo preditivo.

```
df['ibov_ewm_5'] = df['close_ibovespa'].ewm(span=5,
adjust=False).mean()
df['ibov_ewm_7'] = df['close_ibovespa'].ewm(span=7,
adjust=False).mean()
df['ibov_ewm_9'] = df['close_ibovespa'].ewm(span=9,
adjust=False).mean()
df['ibov_ewm_10'] = df['close_ibovespa'].ewm(span=10,
adjust=False).mean()
df['ibov_ewm_12'] = df['close_ibovespa'].ewm(span=12,
adjust=False).mean()
df['ibov_ewm_15'] = df['close_ibovespa'].ewm(span=15,
adjust=False).mean()
df['ibov_ewm_20'] = df['close_ibovespa'].ewm(span=20,
adjust=False).mean()
start_date_2y = '2023-07-05'
end_date_2y = '2025-07-05'
df_2y = df.loc[start_date_2y:end_date_2y]
df_ewm = df_2y.filter(like='ibov_ewm_')
if df_ewm.empty:
    raise ValueError("Nenhuma coluna de média móvel exponencial
encontrada. Verifique os nomes das colunas.")
num_plots = len(df_ewm.columns)
fig, axes = plt.subplots(
    nrows=num_plots,
    ncols=1,
    figsize=(15, 5 * num_plots),
    sharex=True
)
if num_plots == 1:
    axes = [axes]
fig.suptitle(
    f'Médias Móveis Exponenciais (EWM) do Ibovespa ({start_date_2y} a
{end_date_2y})',
    fontsize=16,
```

```
    y=0.95
)
colors = ['blue', 'green', 'red', 'purple', 'orange', 'brown', 'pink']

for i, column in enumerate(df_ewm.columns):
    ax = axes[i]
    df_ewm[column].plot(
        ax=ax,
        legend=False,
        color=colors[i % len(colors)],
        linewidth=1.5,
        label=column
    )
    ax.set_title(column.replace('_', ' ').upper(), fontsize=12)
    ax.set_ylabel('Preço', fontsize=10)
    ax.grid(True, linestyle='--', alpha=0.6)
    ax.legend(loc='upper right')


plt.xlabel('Data', fontsize=12)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

Agora vamos consolidar a visualização de todas as médias móveis exponenciais (EWMs) do IBOVESPA em um único gráfico.

```
plt.figure(figsize=(15, 6))
for column in df_ewm.columns:
    plt.plot(df_ewm[column], label=column.replace('_', ' ').upper(),
linewidth=1.5)
plt.title(f'EWMs do Ibovespa ({start_date_2y} a {end_date_2y})',
fontsize=16)
plt.ylabel('Preço')
plt.xlabel('Data')
plt.grid(linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```



Dataframe – Médias Exponenciais (EWS) e Preços de Fechamento

O primeiro passo é garantir que o índice do DataFrame esteja no formato de data para que a filtragem por período funcione corretamente. Em seguida, o código seleciona apenas as colunas relevantes para análise — aquelas que contêm "ewm" no nome (indicadores de tendência suavizada) e a coluna de fechamento. Essa preparação é essencial para análises futuras mais específicas, garantindo que o conjunto de dados estejam limpo, consistente e focado nos elementos-chave para avaliação de comportamento de preços ao longo do tempo. Ao final, uma verificação básica confirma o período selecionado e as colunas incluídas, ajudando a validar se o filtro e a seleção ocorreram como esperado.

```
start_date_ewm = '2023-07-05'
end_date_ewm   = '2025-07-05'
if not isinstance(df.index, pd.DatetimeIndex):
      df.index = pd.to_datetime(df.index)
df_ewm_s_e = df.loc[start_date_ewm:end_date_ewm].copy()
df_ewm = df_ewm_s_e.filter(regex='ewm|close', axis=1)
```

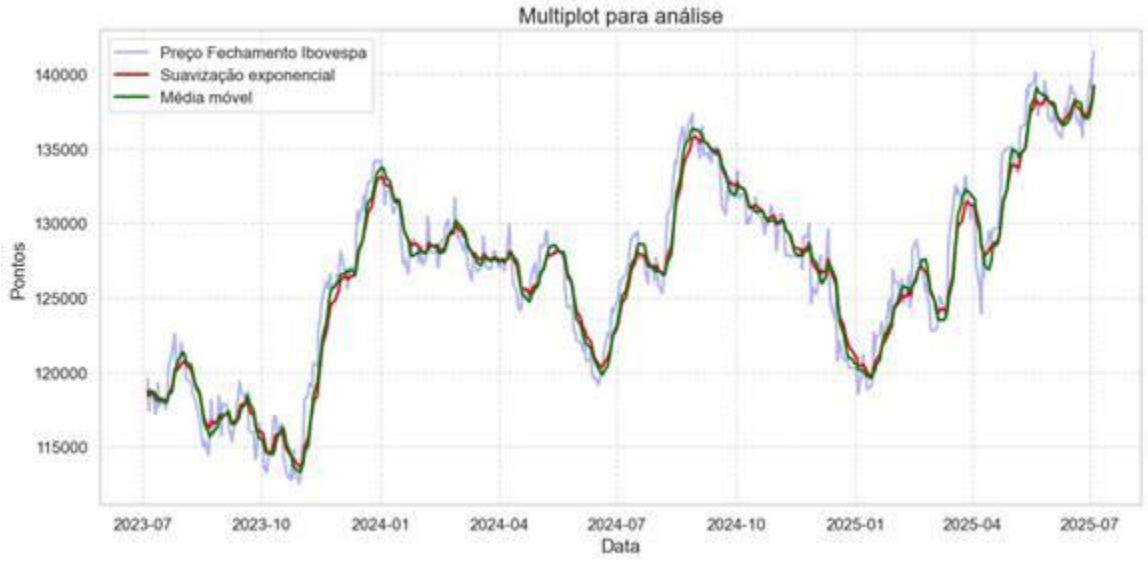
Para ver o Dataframe consultar notebook 5 no Github.

Comparação Visual entre Preço de Fechamento e Indicadores de Tendência

Nesta etapa do código buscamos comparar visualmente o comportamento do preço de fechamento com dois tipos distintos de suavização de tendência. A média móvel exponencial (EWM) e a média móvel simples (MA). Para facilitar a análise da dinâmica do mercado. Ao sobrepor essas curvas em um mesmo gráfico, o analista consegue identificar momentos em que

o preço se distancia das tendências calculadas, além de observar cruzamentos entre as curvas que podem sinalizar possíveis pontos de reversão ou continuidade da tendência. Essa abordagem auxilia na compreensão das reações de curto prazo versus movimentos suavizados, servindo como base para decisões preditivas mais embasadas.

```
plt.figure(figsize=(12, 6))
plt.plot(df_ewm.index, df_ewm['close_ibovespa'], label='Preço
Fechamento Ibovespa', color='blue', alpha=0.3)
plt.plot(df_ewm.index, df_ewm['ibov_ewm_9'], label='Suavização
exponencial', color='red', linewidth=2)
plt.plot(df_ma.index, df_ma['ibov_ma7'], label='Média móvel',
color='green', linewidth=2)
plt.title('Multiplot para análise')
plt.xlabel('Data')
plt.ylabel('Pontos')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Em seguida calculamos os retornos percentuais em diferentes janelas temporais, diária, semanal (5 dias) e mensal (21 dias) para capturar a dinâmica do mercado em horizontes distintos. Isso permite analisar não apenas a volatilidade e performance do índice dia a dia, mas também observar tendências de curto e médio prazo.

```
df['ret_ibov_diario'] = df['close_ibovespa'].pct_change()
df['ret_ibov_5d'] = df['close_ibovespa'].pct_change(5)
df['ret_ibov_21d'] = df['close_ibovespa'].pct_change(21)
```

O passo seguinte foi criar uma função estruturada para gerar um conjunto abrangente de features quantitativas baseadas no comportamento do Ibovespa e do dólar, com o objetivo de enriquecer a base de dados para análises preditivas e estatísticas. A função segue uma lógica progressiva que começa com o cálculo dos retornos logarítmicos diários e evolui para janelas de tendência (curto, médio e longo prazo), indicadores de momentum, relações entre ativos (como Ibovespa vs. dólar), comparação com médias móveis, identificação de eventos extremos e padrões de direção consecutiva. Ao final, o código realiza uma limpeza cuidadosa para garantir que os dados estejam prontos para alimentar modelos de machine learning ou análises exploratórias. O resultado é um DataFrame robusto com variáveis que capturam diferentes dimensões do comportamento de mercado.

```
def criar_features_ibovespa(df):
    df['log_ret_ibov_1d'] = np.log(df['close_ibovespa'] /
df['close_ibovespa'].shift(1))
    df['log_ret_dolar_1d'] = np.log(df['close_dolar'] /
df['close_dolar'].shift(1))
    janelas_tendencia = {
        'curto_prazo': [3, 5],
        'medio_prazo': [10, 21],
        'longo_prazo': [42, 63, 126]
    }
    for periodo, dias in janelas_tendencia.items():
        for dias in janelas_tendencia[periodo]:
            df[f'tendencia_ibov_{periodo}_{dias}d'] =
np.log(df['close_ibovespa'] / df['close_ibovespa'].shift(dias))
            df[f'tendencia_dolar_{periodo}_{dias}d'] =
np.log(df['close_dolar'] / df['close_dolar'].shift(dias))
    lags_momentum = {
        'ultimos_dias': [1, 2, 3],
        'semana_anterior': [5, 7],
        'mes_anterior': [15, 21]
    }
    for tipo, lags in lags_momentum.items():
        for lag in lags:
            df[f'momento_ibov_{tipo}_d-{lag}'] =
df['log_ret_ibov_1d'].shift(lag)
            df[f'momento_dolar_{tipo}_d-{lag}'] =
df['log_ret_dolar_1d'].shift(lag)
    df['ret_ibov_vs_dolar_1d'] = df['log_ret_ibov_1d'] -
df['log_ret_dolar_1d']
    df['spread_ibov_dolar_21d'] = df['tendencia_ibov_medio_prazo_21d']
- df['tendencia_dolar_medio_prazo_21d']
    df['vol_relativa_5d'] = df['log_ret_ibov_1d'].rolling(5).std() /
df['log_ret_dolar_1d'].rolling(5).std()
```

```
df['ratio_ret_ibov_dolar_1d'] = df['log_ret_ibov_1d'] /
(df['log_ret_dolar_1d'] + 1e-10)

    medias_moveis = {
        'curto_prazo': [5, 10],
        'medio_prazo': [20, 50],
        'longo_prazo': [100, 200]
    }
    for periodo, windows in medias_moveis.items():
        for window in windows:
            df[f'ibov_vs_ma{window}'] = df['close_ibovespa'] /
df['close_ibovespa'].rolling(window).mean() - 1
            df[f'dolar_vs_ma{window}'] = df['close_dolar'] /
df['close_dolar'].rolling(window).mean() - 1
    df['abnormal_ibov_move'] = (df['log_ret_ibov_1d'].abs() > 2 *
df['log_ret_ibov_1d'].rolling(21).std()).astype(int)
    df['ibov_direction'] = np.sign(df['log_ret_ibov_1d'])
    df['ibov_streak'] = df['ibov_direction'] * (df['ibov_direction'] ==
df['ibov_direction'].shift(1)).cumsum()

    df = df.replace([np.inf, -np.inf], np.nan)
    df.fillna(method='ffill', inplace=True)
    df.fillna(method='bfill', inplace=True)
    df = df.dropna()
    df.drop(columns=['ibov_direction'], inplace=True)
    return df
df = criar_features_ibovespa(df)
 print(df.head())
 print(df.columns)
```

close_petroleo_brent close_petrobras close_dolar \

data

2005-03-07	53.89	3.36	2.15
2005-03-08	54.59	3.32	2.15
2005-03-09	54.77	3.26	2.15
2005-03-10	53.54	3.18	2.15
2005-03-11	54.43	3.13	2.15

close_ibovespa close_sp500 high_petroleo_brent high_petrobras \

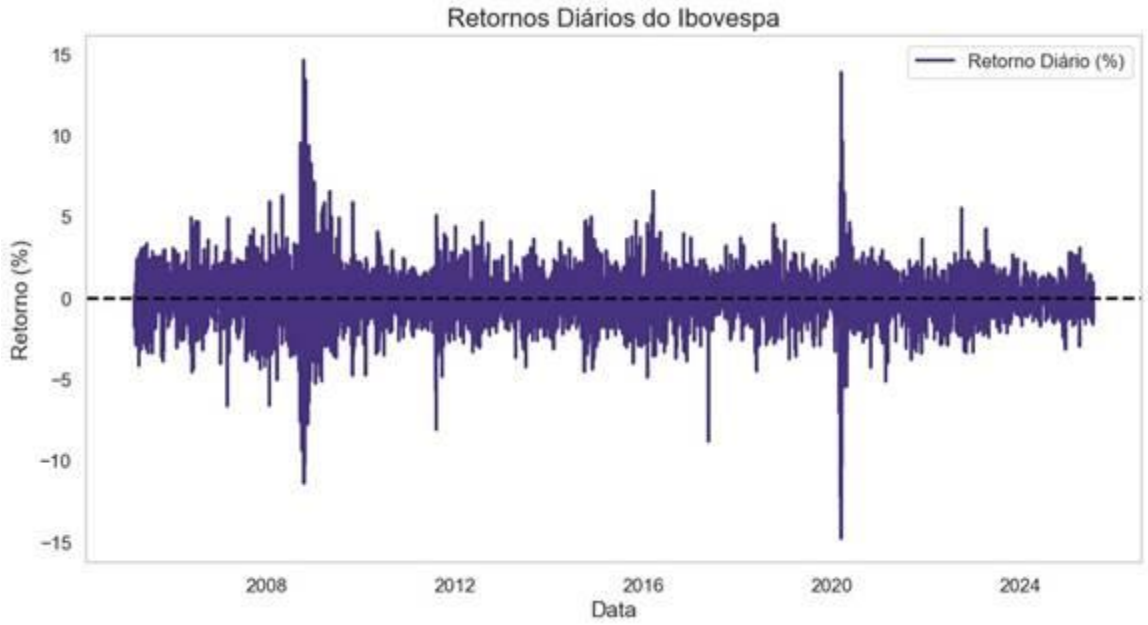
data

2005-03-07	29455.00	1225.31	53.95	3.40
2005-03-08	29021.00	1219.43	55.15	3.33
2005-03-09	28514.00	1207.01	55.65	3.36
2005-03-10	28567.00	1209.25	54.62	3.29
2005-03-11	28075.00	1200.08	54.60	3.22

	high_dolar	high_ibovespa	high_sp500	low_petroleo_brent	\
data					
2005-03-07	2.17	29584.00	1229.11	52.91	
2005-03-08	2.17	29452.00	1225.69	53.26	
2005-03-09	2.17	29058.00	1219.43	54.16	
2005-03-10	2.17	28682.00	1211.23	52.90	
2005-03-11	2.17	28932.00	1213.04	52.50	

A nossa próxima análise foi gerar um gráfico para visualizar os retornos percentuais diários ao longo do tempo, facilitando a identificação de padrões de volatilidade, oscilações extremas e frequência de retornos positivos ou negativos. Ao multiplicar os retornos por 100 e traçar uma linha horizontal em zero, o gráfico destaca claramente os dias de ganho e perda, sendo uma ferramenta essencial para análise de risco e comportamento do índice.

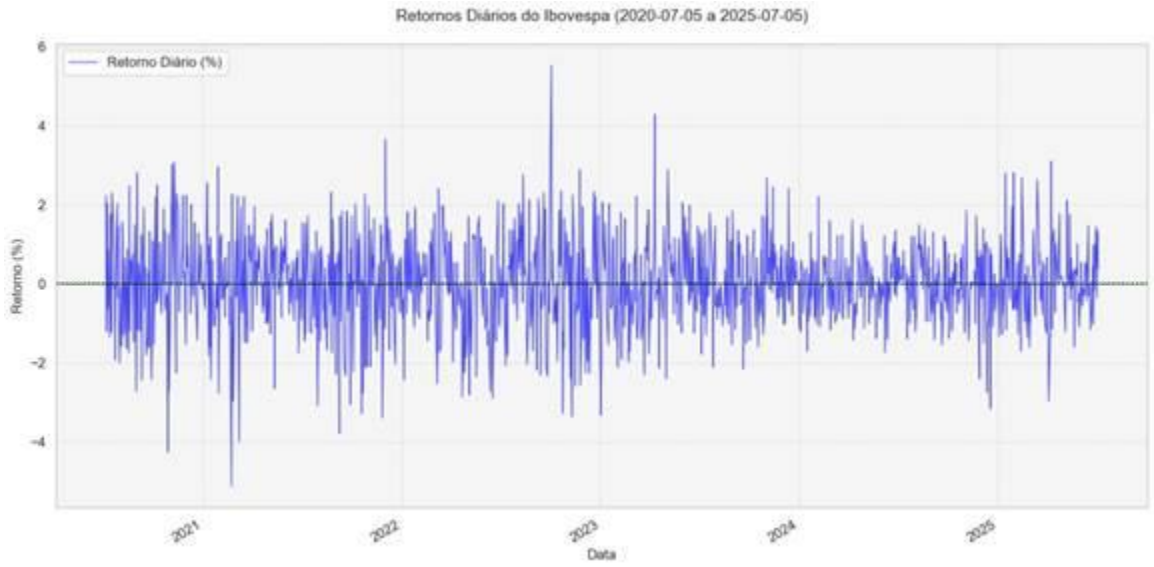
```
plt.figure(figsize=(12, 6))
plt.plot(df['ret_ibov_diario'] * 100, label='Retorno Diário (%)')
plt.axhline(y=0, color='black', linestyle='--')
plt.title('Retornos Diários do Ibovespa')
plt.ylabel('Retorno (%)')
plt.xlabel('Data')
plt.legend()
plt.grid()
plt.show()
```



Este bloco de código tem como objetivo apresentar uma visualização clara e detalhada dos retornos diários em formato percentual ao longo de um período de cinco anos. Ele começa com a verificação e preparação dos dados, assegurando que o índice de datas esteja no formato correto e que a coluna de retorno exista. Em seguida, constrói um gráfico de linha com a série de retornos diários, destacando a linha zero (ponto de retorno nulo) e adicionando uma linha tracejada representando a média dos retornos no período. A visualização é aprimorada com ajustes estéticos no fundo, na legenda e na formatação do eixo de datas, tornando o gráfico mais informativo e intuitivo para análise de desempenho e identificação de padrões de retorno ao longo do tempo.

```
start_date = '2020-07-05'
end_date = '2025-07-05'
if not isinstance(df.index, pd.DatetimeIndex):
      df.index = pd.to_datetime(df.index)
df_periodo = df.loc[start_date:end_date].copy()
if 'ret_ibov_diario' not in df_periodo.columns:
      raise ValueError("Coluna 'ret_ibov_diario' não encontrada no
DataFrame")


plt.figure(figsize=(14, 7))
plt.plot(df_periodo.index, df_periodo['ret_ibov_diario'] * 100,
         label='Retorno Diário (%)',
         color='blue',
         linewidth=1,
         alpha=0.7)
```

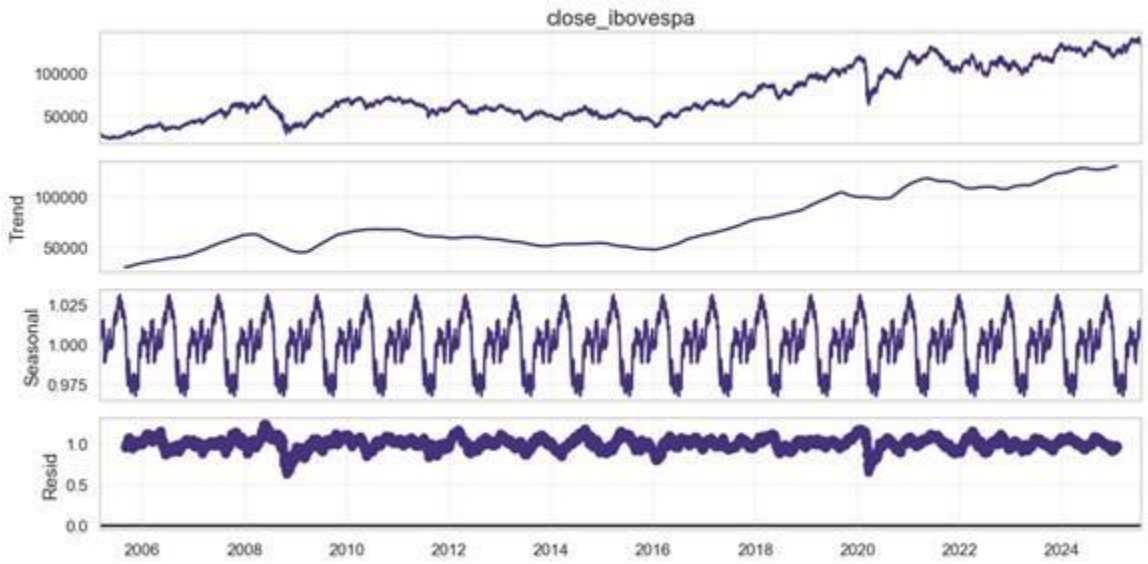
```
plt.axhline(y=0, color='black', linestyle='--', linewidth=0.8)
plt.title(f'Retornos Diários do Ibovespa ({start_date} a {end_date})',
fontsize=14, pad=20)
plt.ylabel('Retorno (%)', fontsize=12)
plt.xlabel('Data', fontsize=12)
plt.legend(fontsize=12, loc='upper left')

plt.grid(True, linestyle='--', alpha=0.5)
plt.gca().set_facecolor('#f5f5f5')
plt.gcf().autofmt_xdate()
media_retorno = df_periodo['ret_ibov_diario'].mean() * 100
plt.axhline(y=media_retorno, color='green', linestyle=':',
            label=f'Média ({media_retorno:.2f}%)', linewidth=1.5)
plt.tight_layout()
plt.show()
```



Retornos Logarítmicos

Aplicamos uma decomposição sazonal multiplicativa à série temporal de fechamento do Ibovespa, considerando um período de 252 dias (aproximadamente um ano de pregão). O objetivo é separar a série em seus componentes de tendência, sazonalidade e ruído, permitindo uma análise mais detalhada do comportamento do índice ao longo do tempo, especialmente para identificar padrões sazonais recorrentes e entender melhor a evolução do mercado em diferentes ciclos.

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomp = seasonal_decompose(
    df['close_ibovespa'], model='multiplicative', period=252)
decomp.plot()
```

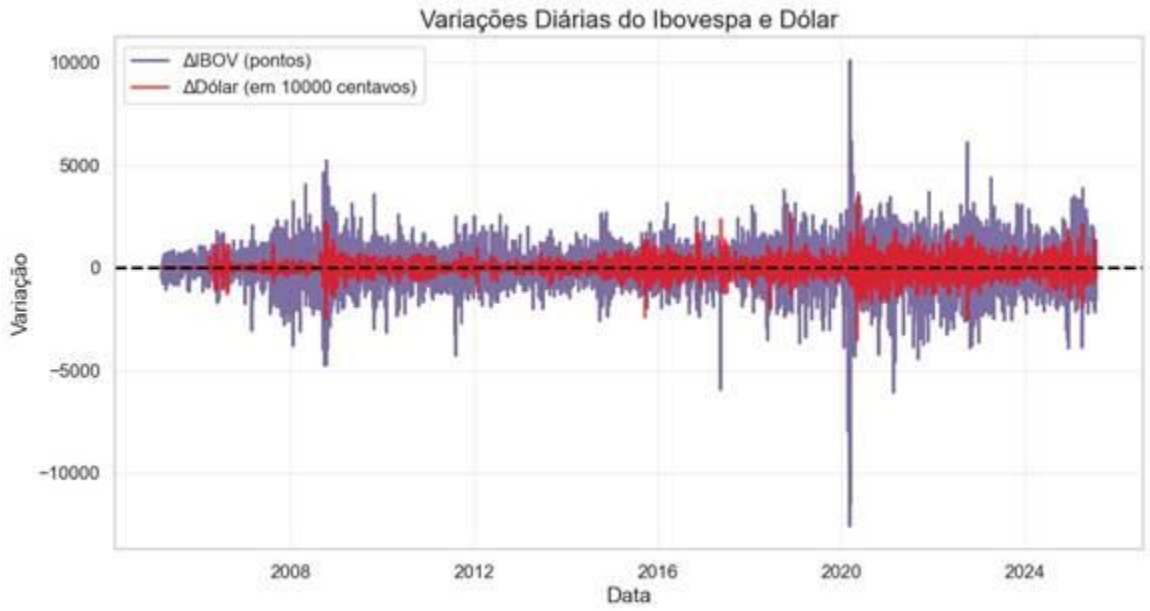


Agora temos o objetivo de analisar e visualizar as variações absolutas diárias dos valores de fechamento do Ibovespa e do Dólar. Primeiro, calcula-se a diferença bruta entre os fechamentos de dias consecutivos, o que permite observar oscilações em pontos para o Ibovespa e em centavos para o Dólar. Após a limpeza de valores nulos, são exibidas estatísticas descritivas dessas variações, e um gráfico é gerado para facilitar a comparação visual entre os dois ativos ao longo do tempo. A variação do Dólar é multiplicada por 10.000 para ter escala semelhante à do Ibovespa e permitir uma leitura mais clara no gráfico.

```
df['delta_ibov'] = df['close_ibovespa'].diff()
df['delta_dolar'] = df['close_dolar'].diff()
df = df.dropna(subset=['delta_ibov', 'delta_dolar'])
print(df[['delta_ibov', 'delta_dolar']].describe())
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['delta_ibov'], label='ΔIBOV (pontos)', alpha=0.7)
plt.plot(df.index, df['delta_dolar']*10000,
         label='ΔDólar (em 10000 centavos)', alpha=0.7, color='red')
plt.axhline(0, color='black', linestyle='--')
plt.title('Variações Diárias do Ibovespa e Dólar')
plt.ylabel('Variação')
plt.xlabel('Data')
plt.legend()
plt.grid(True)
plt.show()
```

delta_ibov delta_dolar

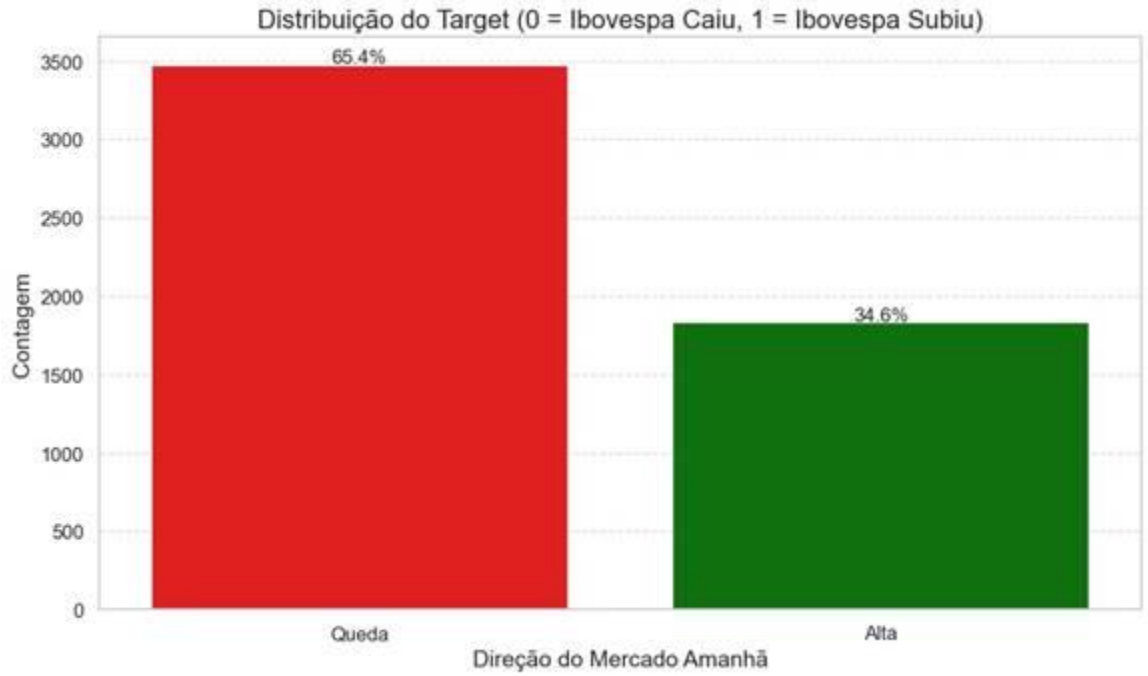
count	5313.00	5313.00
mean	19.64	0.00
std	1107.20	0.04
min	-12588.00	-0.35
25%	-500.00	-0.01
50%	0.00	0.00
75%	589.00	0.01
max	10095.00	0.34



Seguimos com a análise do balanceamento da variável alvo (target), que indica se o Ibovespa subirá (1) ou cairá (0) no próximo dia. Para isso, é gerado um gráfico de barras com a contagem de ocorrências de cada classe, usando cores distintas para alta e queda. Além da visualização, o código também insere as porcentagens relativas de cada categoria acima das barras, facilitando a interpretação sobre o equilíbrio ou desequilíbrio da base de dados em relação ao comportamento futuro do índice.

```
plt.figure(figsize=(10, 6))
sns.countplot(x='target', data=df, palette=['red', 'green'])
plt.title('Distribuição do Target (0 = Ibovespa Caiu, 1 = Ibovespa
Subiu)')
plt.xlabel('Direção do Mercado Amanhã')
plt.ylabel('Contagem')
plt.xticks([0, 1], ['Queda', 'Alta'])
plt.grid(axis='y', linestyle='--', alpha=0.7)
total = len(df)
```

```
for p in plt.gca().patches:
    height = p.get_height()
    plt.gca().text(p.get_x() + p.get_width()/2., height + 10,
                   f'{height/total:.1%}', ha='center')
plt.tight_layout()
plt.show()
```



O bloco de código tem como objetivo visualizar o comportamento do preço do Ibovespa ao longo do tempo, destacando os pontos em que o modelo sinaliza que o índice subirá (target=1) ou cairá (target=0) no dia seguinte. Essa abordagem permite observar visualmente se há padrões ou agrupamentos nos dias de alta e queda previstos, facilitando a análise da relação entre o movimento histórico dos preços e o sinal gerado para o próximo dia.

```
plt.figure(figsize=(14, 6))
plt.plot(df.index, df['close_ibovespa'],
         label='Preço do Ibovespa', color='gray', alpha=0.3)
plt.scatter(df[df['target'] == 1].index,
            df[df['target'] == 1]['close_ibovespa'],
            color='green', label='Dia Seguinte de Alta', alpha=0.5)
plt.scatter(df[df['target'] == 0].index,
            df[df['target'] == 0]['close_ibovespa'],
            color='red', label='Dia Seguinte de Queda', alpha=0.5)
plt.title('Preço do Ibovespa com Sinalização do Target')
plt.ylabel('Pontos')
plt.xlabel('Data')
plt.legend()
plt.grid(linestyle='--', alpha=0.5)
```

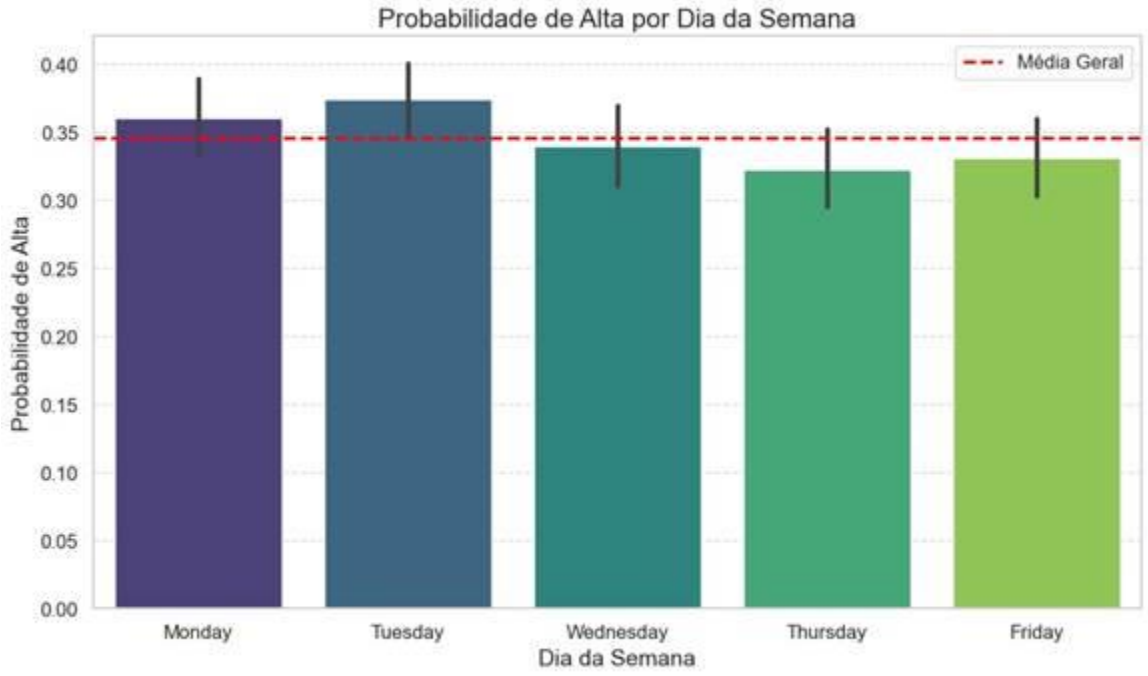
```
plt.tight_layout()
plt.show()
```

Agora queremos analisar a tendência do comportamento do mercado ao longo do tempo, utilizando uma média móvel de 30 dias da variável target, que representa se o Ibovespa subiu (1) ou caiu (0) no dia seguinte. O raciocínio é suavizar as oscilações diárias e observar períodos em que houve maior frequência de altas ou quedas, comparando essa proporção com a linha neutra de 50% — o que pode indicar fases de otimismo ou pessimismo predominante no mercado.

```
plt.figure(figsize=(14, 6))
df['media_movel_alta'] = df['target'].rolling(window=30).mean()
plt.plot(df.index, df['media_movel_alta'],
         label='Média Móvel de 30 Dias', color='blue')
plt.axhline(y=0.5, color='red', linestyle='--', label='Linha Neutra
(50%)')
plt.title('Proporção de Dias com Ibovespa em Alta (30 Dias Móveis)')
plt.ylabel('Proporção de Dias de Alta')
plt.xlabel('Data')
plt.ylim(0, 1)
plt.legend()
plt.grid(linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

Buscamos identificar se existe alguma relação entre o dia da semana e a probabilidade do Ibovespa apresentar alta no dia seguinte. Para isso, calculamos a média da variável target (indicando se houve alta) para cada dia da semana, visualizando os resultados em um gráfico de barras. A linha horizontal vermelha representa a média geral de alta no período, permitindo comparar o comportamento de cada dia em relação ao padrão geral do mercado.

```
df['dia_semana'] = df.index.day_name()
plt.figure(figsize=(10, 6))
sns.barplot(x='dia_semana', y='target', data=df,
            order=['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday'],
            palette='viridis')
plt.title('Probabilidade de Alta por Dia da Semana')
plt.xlabel('Dia da Semana')
plt.ylabel('Probabilidade de Alta')
plt.axhline(y=df['target'].mean(), color='red',
            linestyle='--', label='Média Geral')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Para o próximo passo, nosso objetivo foi criar indicadores derivados da relação entre preço e médias móveis (osciladores) e da dinâmica de volumes, de forma a medir o quão distante o preço está de sua média recente e identificar movimentações atípicas no volume negociado. Ele calcula essas métricas tanto para o Ibovespa quanto para o dólar, aplica transformações logarítmicas para dar robustez estatística e reduzir o impacto de outliers, e, ao final, remove colunas intermediárias para manter apenas as variáveis finais que serão utilizadas no modelo.

```
df['ibov_ma21'] = df['close_ibovespa'].rolling(window=21).mean()
df['preco_vs_ma21'] = np.log(df['close_ibovespa'] / df['ibov_ma21'])
df['dolar_ma21'] = df['close_dolar'].rolling(window=21).mean()
df['dolar_vs_ma21'] = np.log(df['close_dolar'] / df['dolar_ma21'])
df['log_volume_ibov'] = np.log(df['volume_ibovespa'] + 1)
df['log_ma_volume_21d'] =
df['log_volume_ibov'].rolling(window=21).mean()
df['choque_de_volume_log'] = df['log_volume_ibov'] -
df['log_ma_volume_21d']

colunas_para_dropar = [
    'volume_ibovespa',
    'ibov_ma21',
    'dolar_ma21',
    'log_ma_volume_21d',
    'volume_dolar'
]
df = df.drop(columns=colunas_para_dropar, errors='ignore')


print("DataFrame após a criação e limpeza das novas features:")
```

```
print(df[['preco_vs_ma21', 'dolar_vs_ma21', 'log_volume_ibov',
          choque_de_volume_log']].head())
```

Para ver o Dataframe consultar notebook 4.

Para a próxima etapa decidimos extrair features de ação de preço que capturem a dinâmica intradiária dos ativos Ibovespa e Dólar. A ideia é gerar variáveis que revelem comportamentos como amplitude dos preços, gaps de abertura, posição relativa do fechamento no candle e a presença de sombras superiores e inferiores, todas tratadas com escalas logarítmicas ou normalizadas. Esses indicadores são amplamente utilizados para análise técnica e ajudam o modelo a entender contextos como volatilidade diária, força de fechamento e possíveis reversões, oferecendo uma base quantitativa robusta para previsões de curto prazo.

```
df['range_log_ibov'] = np.log(df['high_ibovespa'] / df['low_ibovespa'])
df['range_log_dolar'] = np.log(df['high_dolar'] / df['low_dolar'])
df['gap_log_ibov'] = np.log(
    df['open_ibovespa'] / df['close_ibovespa'].shift(1))
df['pos_close_ibov'] = (df['close_ibovespa'] - df['low_ibovespa']
                        ) / (df['high_ibovespa'] - df['low_ibovespa'])
df['pos_close_dolar'] = (
    df['close_dolar'] - df['low_dolar']) / (df['high_dolar'] -
df['low_dolar'])
amplitude_total_ibov = df['high_ibovespa'] - df['low_ibovespa']
amplitude_total_ibov = amplitude_total_ibov.replace(0, np.nan)
df['sombra_sup_norm_ibov'] = (df['high_ibovespa'] - np.maximum(
    df['open_ibovespa'], df['close_ibovespa'])) / amplitude_total_ibov
df['sombra_inf_norm_ibov'] = (np.minimum(
    df['open_ibovespa'], df['close_ibovespa']) - df['low_ibovespa']) /
amplitude_total_ibov
df[['sombra_sup_norm_ibov', 'sombra_inf_norm_ibov']] = df[[
    'sombra_sup_norm_ibov', 'sombra_inf_norm_ibov']].fillna(0)
```

Análise de Viés

Binning

Nesta etapa, enriquecemos o conjunto de dados com variáveis de interação, combinando fatores como volatilidade, tendência, volume e posição de fechamento. A ideia é capturar relações mais complexas que não seriam percebidas analisando apenas as variáveis isoladamente. Criar variáveis de interação, combinando pares de features previamente calculadas para capturar relações não lineares e efeitos conjuntos entre diferentes fatores de mercado. A lógica principal é que variáveis como volatilidade, volume, tendência e comportamento do dólar podem ter efeitos mais informativos quando analisadas em conjunto. Essas features fornecem uma base mais rica para a próxima etapa de análise de viés por binning, onde permite identificar padrões escondidos e potenciais desequilíbrios no comportamento do mercado.

```
df['inter_vol_tend_ibov'] = df['range_log_ibov'] * df['ibov_vs_ma21']
df['inter_vol_preco_ibov'] = df['choque_de_volume_log'] *
    df['preco_vs_ma21']
df['inter_dolar_pos_close_ibov'] = df['log_ret_dolar_1d'] *
    df['pos_close_ibov']
```

Agora focaremos na **eliminação estratégica de variáveis redundantes, não estacionárias ou pouco informativas**, a fim de **refinar o conjunto de dados** antes da modelagem preditiva. As colunas descartadas incluem retornos brutos substituídos por versões logarítmicas mais robustas, médias móveis cruas já representadas por indicadores normalizados, binarizações que perdem granularidade e colunas auxiliares que não devem ser usadas como variáveis explicativas. Essa limpeza garante que o modelo trabalhe com **features mais relevantes, informativas e estatisticamente estáveis**, reduzindo ruído e potencial multicolinearidade.

```
colunas_para_dropar = [
    'ret_ibov_diario',
    'delta_ibov',
    'delta_dolar',
    'ret_ibov_5d',
    'ret_ibov_21d',
    'ibov_ma2',
    'ibov_ma5',
    'ibov_ma7',
    'ibov_ma10',
    'ibov_ma14',
    'ibov_ma21',
    'dolar_ma2',
    'dolar_ma7',
```

```
    'dolar_ma10',
    'dolar_ma14',
    'dolar_ma21',
    'ibov_ewm_9',
    'ibov_ewm_10',
    'ibov_ewm_12',
    'ibov_ewm_15',
    'ibov_ewm_20',
    'log_ret_ibov_1d_bin',
    'log_ret_dolar_1d_bin',
    'tendencia_ibov_21d_bin',
    'ibov_ma14_bin',
    'close_amanha'
]
df = df.drop(columns=colunas_para_dropar, errors='ignore')
```

O código abaixo define uma função robusta de análise de viés para variáveis preditoras numéricas com base na técnica de binning, aplicando-a a diferentes recortes temporais da base do Ibovespa. A lógica geral consiste em discretizar cada feature contínua em decis (10 faixas de valores), calcular a taxa média de acerto do target (probabilidade de alta do Ibov no dia seguinte) dentro de cada decil, e verificar se essa taxa varia significativamente entre os grupos, o que pode indicar viés ou poder preditivo. A função também permite a análise por diferentes regimes de mercado (como crise de 2008, pandemia, etc.), facilitando a investigação da estabilidade das relações ao longo do tempo. Ao final, a distribuição da performance por decil é visualizada em gráficos de barras com intervalos de confiança, destacando se a variável apresenta padrões informativos para o modelo preditivo.

```
warnings.filterwarnings('ignore', category=UserWarning)

def analisar_feature_classificacao(dataframe, feature_col,
                target_col='target',
                q=10, confianca=0.95,
                start_date=None, end_date=None):
    df_filtrado = dataframe.copy()
    if start_date:
        df_filtrado = df_filtrado[df_filtrado.index >=
pd.to_datetime(start_date)]
    if end_date:
        df_filtrado = df_filtrado[df_filtrado.index <=
pd.to_datetime(end_date)]

    if df_filtrado.empty:
        print(f"Nenhum dado encontrado para o período especificado.")
        return None

    periodo_str = f"({start_date or 'Início'} a {end_date or 'Fim'})"
    print(f"--- Análise da Feature: '{feature_col}' no Período:
{periodo_str} ---")

    bin_col = f'{feature_col}_bin'
    try:
        df_filtrado[bin_col] = pd.qcut(
            df_filtrado[feature_col], q=q, labels=False,
duplicates='drop')
```

```
    except ValueError as e:
        print(f"Não foi possível binarizar a feature '{feature_col}'.
Motivo: {e}")
        return None

    analise_agrupada =
df_filtrado.groupby(bin_col)[target_col].agg(['mean', 'count'])

    Z = norm.ppf(1 - (1 - confianca) / 2)
    analise_agrupada['ic_margem'] = Z * np.sqrt(
        analise_agrupada['mean'] * (1 - analise_agrupada['mean']) /
analise_agrupada['count'])
    analise_agrupada['ic_inferior'] = analise_agrupada['mean'] -
analise_agrupada['ic_margem']
    analise_agrupada['ic_superior'] = analise_agrupada['mean'] +
analise_agrupada['ic_margem']

    fig, ax = plt.subplots(figsize=(12, 7))
    analise_agrupada['mean'].plot(kind='bar', ax=ax, color='skyblue',
                                  yerr=analise_agrupada['ic_margem'],
                                  capsize=5, ecolor='darkgray')
    ax.axhline(0.5, color='red', linestyle='--', label=f'Sem Viés
(50%)')
    ax.set_title(f'Probabilidade de Alta do IBOV por Decil de
"{feature_col}"\nPeríodo: {periodo_str}', fontsize=16)
    ax.set_ylabel(f'Probabilidade Média de Alta (+/- {confianca:.0%}
IC)', fontsize=12)
    ax.set_xlabel(f'Decil da Feature (0=Mais Baixo, {q-1}=Mais Alto)',
fontsize=12)
    ax.legend()
    plt.xticks(rotation=0)
    plt.show()

    return analise_agrupada[['mean', 'count', 'ic_inferior',
'ic_superior']]


periodos_de_interesse = {
    'Dados Completos': (None, None),
    'Crise Subprime 2008-2009': ('2008-01-01', None),
    'Pós-Crise / Gov. Dilma I': ('2010-01-01', None),
    'Crise 2015-2016 / Impeachment': ('2015-01-01', None),
    'Pandemia COVID-19': ('2020-02-01', None),
    'Pós-Pandemia / Juros Altos': ('2021-07-01', None)
}

features_para_analisar = [
    # 'log_ret_ibov_1d',
    # 'log_ret_dolar_1d',
    # 'tendencia_ibov_21d',
    # 'ret_ibov_vs_dolar_5d',
    # 'ibov_ma14'
]

for feature in features_para_analisar:
print(f"\n\n============================================================
=========")
```

```
    print(f"    ANÁLISE COMPLETA DA FEATURE: {feature.upper()}")

print(f"================================================================
=====")

    for nome_periodo, (data_inicio, data_fim) in
periodos_de_interesse.items():
        analisar_feature_classificacao(
            dataframe=df,
            feature_col=feature,
            start_date=data_inicio,
            end_date=data_fim
        )
```

Nesta última etapa apenas salvamos o DataFrame final com todas as features geradas ao longo da análise em um banco de dados DuckDB. Para isso, o DataFrame é preparado com o índice redefinido, e uma conexão é aberta com o banco especificado. Em seguida, o código cria ou substitui uma tabela com todas as colunas do DataFrame e verifica se a operação foi bem-sucedida ao listar as tabelas existentes no banco. Por fim, a conexão é encerrada e mensagens de sucesso ou erro são exibidas conforme o resultado. Esse processo garante a persistência das features para futuras análises ou modelagens.

```
df_para_salvar = df.reset_index()
nome_tabela_features = "features_completas"
db_path = str(config.DB_PATH)
print(
    f"Salvando o DataFrame final com features na tabela
'{nome_tabela_features}' em: {db_path}")
try:
    con = duckdb.connect(database=db_path, read_only=False)
    con.execute(
        f"CREATE OR REPLACE TABLE {nome_tabela_features} AS SELECT *
FROM df_para_salvar")
    print("\nTabelas existentes no banco de dados:")
    display(con.execute("SHOW TABLES").fetchdf())
    con.close()
    print(
        f"\n✅ DataFrame de features salvo com sucesso na tabela
'{nome_tabela_features}'!")
except Exception as e:
    print(f"❌ Ocorreu um erro ao salvar no banco de dados: {e}")
```

Etapa 5: Divisão Final dos Dados

Criar introdução

Neste bloco de código, o raciocínio foi organizar os elementos fundamentais para uma futura análise segmentada por regimes de mercado e por variáveis-chave. Primeiramente, definiu-se uma lista de colunas a serem descartadas (drop_padrão), que incluem os preços brutos do Ibovespa e do dólar, pois esses dados já foram transformados em features mais informativas. Em seguida, construiu-se um dicionário que delimita diversos regimes históricos da economia brasileira e global, marcando pontos de inflexão no comportamento do mercado para futuras análises temporais. Por fim, foi criada uma lista com as melhores features selecionadas para o modelo, incluindo variáveis de retornos acumulados, volatilidade, streaks e indicadores técnicos, junto com o target, sinalizando que esse conjunto será usado como base para análise preditiva ou modelagem.

```
drop_padrão = [
    'target', 'open_ibovespa', 'high_ibovespa', 'low_ibovespa',
'close_ibovespa', 'open_dolar', 'high_dolar', 'low_dolar',
'close_dolar'
]
regimes_para_analise = {
    'Boom das Commodities': '2005-05-06',
    'Crise Financeira Global': '2008-06-01',
    'Crise Doméstica/Política': '2013-01-01',
    'Retomada Pós-Crise': '2016-09-01',
    'Pré-Pandemia Recente': '2018-01-01',
    'Pandemia e Volatilidade': '2020-01-01',
    'Pós-Pandemia (Juros Altos)': '2022-01-01'
}
melhores_features = [
    'sp500_ret_diario_lag_3', 'petroleo_brent_ret_diario',
'choque_de_volume_log', 'sp500_ret_acum_5d', 'petrobras_ret_acum_2d',
'ibovespa_ret_diario', 'sp500_ret_acum_21d',
'momento_ibov_mes_anterior_d-21', 'sombra_sup_norm_ibov',
'ibov_streak', 'petrobras_ret_diario_lag_1', 'petrobras_ret_acum_21d',
'volume_petroleo_brent', 'sp500_ret_diario_lag_1', 'candle_body_ibov',
'BBL_5_2.0', 'spread_ibov_dolar_21d', 'ibov_vs_ma200',
'petroleo_brent_ret_diario_lag_1', 'ibovespa_ret_acum_21d',
'ibov_ewm_5', 'media_movel_alta', 'volume_petrobras',
'petroleo_brent_ret_acum_5d', 'dolar_ret_acum_2d',
'petrobras_ret_diario_lag_5', 'target'
]
```

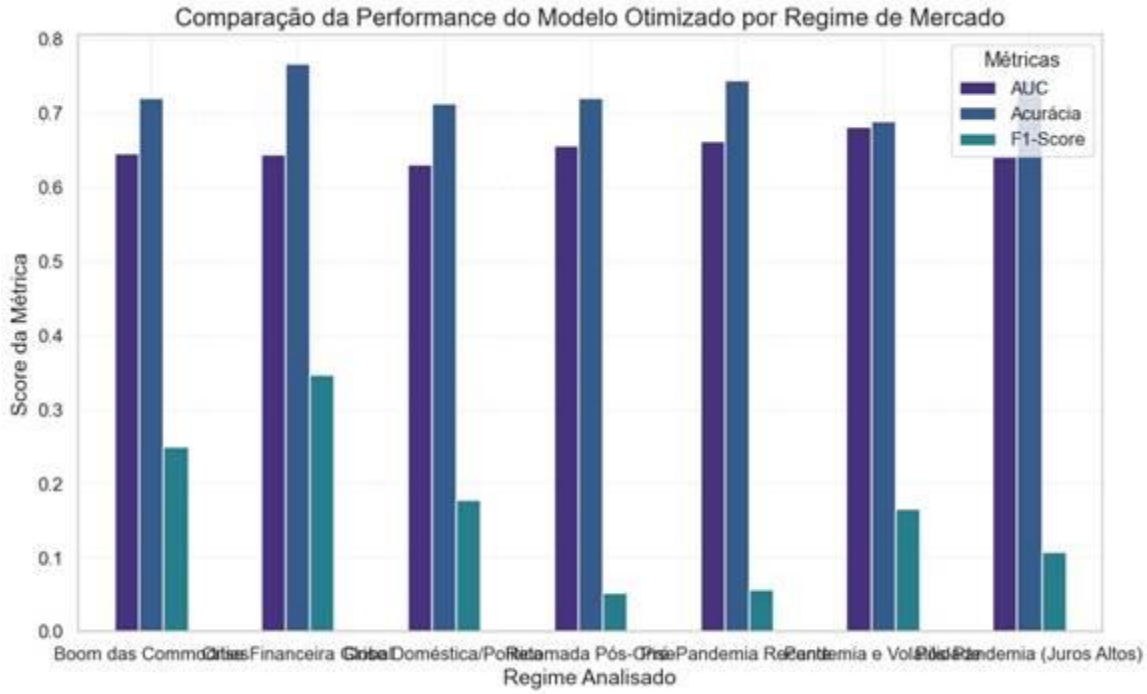
O bloco de código abaixo executa um pipeline completo de modelagem preditiva para diferentes regimes de mercado, com o objetivo de identificar as variáveis mais importantes e avaliar a performance do modelo em cada contexto. Para isso, realizamos (1) Filtragem do período relevante; (2) Separação dos dados em treino e teste; (3) Pré-processamento das features; (4) Otimização de hiperparâmetros com Optuna; (5) Treinamento final e avaliação do modelo; (6) Cálculo da importância das variáveis; (7) Armazenamento dos resultados.

```
optuna.logging.set_verbosity(optuna.logging.WARNING)

def otimizar_e_avaliar_regime(df_completo, nome_regime, data_inicio,
meses_teste=6, top_n_features=5):
    print(f"\n=====================================================")
    print(
        f"--- Processando Regime: '{nome_regime}' (Início:
{data_inicio}) ---")
    print(f"=====================================================")

    df_periodo = df_completo[df_completo.index >= data_inicio].copy()
    df_periodo.dropna(inplace=True)
    df_periodo.replace([np.inf, -np.inf], np.nan, inplace=True)
    df_periodo.dropna(inplace=True)

    if len(df_periodo) < 252:
        print(
            f"Aviso: Período com poucas amostras ({len(df_periodo)}).
Pulando regime.")
        return None

    X = df_periodo.drop(columns=drop_padrão)
    y = df_periodo['target']

    data_final = df_periodo.index.max()
    data_corte = data_final - pd.DateOffset(months=meses_teste)

    X_train = X[X.index < data_corte]
    X_test = X[X.index >= data_corte]
    y_train = y[y.index < data_corte]
    y_test = y[y.index >= data_corte]

    colunas_numericas =
X_train.select_dtypes(include=np.number).columns
    colunas_categoricas =
X_train.select_dtypes(exclude=np.number).columns
    X_train_categoricas = pd.get_dummies(
        X_train[colunas_categoricas], drop_first=True).astype(int)
    X_test_categoricas = pd.get_dummies(
        X_test[colunas_categoricas], drop_first=True).astype(int)

    scaler = StandardScaler()
    X_train_numericas_scaled =
scaler.fit_transform(X_train[colunas_numericas])
```

```
    X_test_numericas_scaled =
scaler.transform(X_test[colunas_numericas])

    X_train_numericas_scaled = pd.DataFrame(
        X_train_numericas_scaled, index=X_train.index,
columns=colunas_numericas)
    X_test_numericas_scaled = pd.DataFrame(
        X_test_numericas_scaled, index=X_test.index,
columns=colunas_numericas)

    X_train_final = pd.concat(
        [X_train_numericas_scaled, X_train_categoricas], axis=1)
    X_test_final = pd.concat(
        [X_test_numericas_scaled, X_test_categoricas], axis=1)
    X_test_final = X_test_final.reindex(
        columns=X_train_final.columns, fill_value=0)

    def objective(trial):
        params = {
            'objective': 'binary', 'metric': 'auc','verbosity': -1,
            'n_estimators': trial.suggest_int('n_estimators', 50, 500),
            'learning_rate': trial.suggest_float('learning_rate', 0.01,
0.3),
            'num_leaves': trial.suggest_int('num_leaves', 20, 300),
            'max_depth': trial.suggest_int('max_depth', 3, 12),
            'random_state': 42
        }
        model = lgb.LGBMClassifier(**params)
        model.fit(X_train_final, y_train)
        y_pred_proba = model.predict_proba(X_test_final)[:, 1]
        return roc_auc_score(y_test, y_pred_proba)
    print(
        f"\nIniciando otimização com Optuna para o regime
'{nome_regime}'...")
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=50)

    best_params = study.best_params
    print(f"Melhores parâmetros encontrados: {best_params}")

    final_model = lgb.LGBMClassifier(
        **best_params, random_state=42, verbosity=-1)
    final_model.fit(X_train_final, y_train)
    y_pred = final_model.predict(X_test_final)
    y_pred_proba = final_model.predict_proba(X_test_final)[:, 1]

    auc = roc_auc_score(y_test, y_pred_proba)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"Performance no teste: AUC={auc:.4f},
Acurácia={accuracy:.4f}")

    print("Calculando a importância das features no conjunto de
teste...")
    perm_importance_result = permutation_importance(
        estimator=final_model,
        X=X_test_final,
```

```
            y=y_test,
            n_repeats=10,
            scoring='roc_auc',
            random_state=42,
            n_jobs=-1
        )

        df_importancia = pd.DataFrame({
            'feature': X_train_final.columns,
            'importance_mean': perm_importance_result.importances_mean
        })
        df_importancia = df_importancia.sort_values(
            by='importance_mean', ascending=False)
        top_features =
df_importancia.head(top_n_features)['feature'].tolist()

        return {
            'Regime': nome_regime,
            'Data Início': data_inicio,
            'AUC': auc,
            'Acurácia': accuracy,
            'F1-Score': f1,
            'Melhores Parâmetros': best_params,
            f'Top_{top_n_features}_Features': top_features
        }


resultados_finais = []

for nome, data in regimes_para_analise.items():
    resultado = otimizar_e_avaliar_regime(df, nome, data,
top_n_features=5)
    if resultado:
        resultados_finais.append(resultado)

df_resultados = pd.DataFrame(resultados_finais)

print("\n\n--- COMPARAÇÃO FINAL DOS REGIMES E SUAS PRINCIPAIS FEATURES
---")
pd.set_option('display.max_colwidth', None)
print(df_resultados[['Regime', 'AUC', 'Acurácia', 'Top_5_Features']])

if not df_resultados.empty:
    df_plot = df_resultados.set_index(
        'Regime')[['AUC', 'Acurácia', 'F1-Score']]
    df_plot.plot(kind='bar', figsize=(12, 7), rot=0)
    plt.title('Comparação da Performance do Modelo Otimizado por Regime
de Mercado')
    plt.ylabel('Score da Métrica')
    plt.xlabel('Regime Analisado')
    plt.grid(axis='y', linestyle='--')
    plt.legend(title='Métricas')
    plt.show()
```

```
 ======================================================================
--- Processando Regime: 'Boom das Commodities' (Início: 2005-05-06) ---
======================================================================

Iniciando otimização com Optuna para o regime 'Boom das Commodities'...
Melhores parâmetros encontrados: {'n_estimators': 292, 'learning_rate': 0.2543457114916105,
'num_leaves': 40, 'max_depth': 9}
Performance no teste: AUC=0.6467, Acurácia=0.7209
Calculando a importância das features no conjunto de teste...

======================================================================
--- Processando Regime: 'Crise Financeira Global' (Início: 2008-06-01) ---
======================================================================

Iniciando otimização com Optuna para o regime 'Crise Financeira Global'...
Melhores parâmetros encontrados: {'n_estimators': 128, 'learning_rate':
0.28420683220160625, 'num_leaves': 238, 'max_depth': 6}
Performance no teste: AUC=0.6446, Acurácia=0.7674
Calculando a importância das features no conjunto de teste...

======================================================================
--- Processando Regime: 'Crise Doméstica/Política' (Início: 2013-01-01) ---
======================================================================

Iniciando otimização com Optuna para o regime 'Crise Doméstica/Política'...
Melhores parâmetros encontrados: {'n_estimators': 107, 'learning_rate': 0.183746193305833,
'num_leaves': 133, 'max_depth': 9}
...
3          [media_movel_alta, dolar_ret_diario_lag_3, choque_de_volume_log,
momento_ibov_mes_anterior_d-21, ibov_vs_ma7]
4   [media_movel_alta, sombra_sup_norm_ibov, ibov_ewm_5,
momento_dolar_mes_anterior_d-15, petrobras_ret_diario_lag_5]
5  [media_movel_alta, sombra_sup_norm_ibov, tendencia_dolar_longo_prazo_42d,
ibovespa_ret_acum_21d, candle_body_ibov]
6            [media_movel_alta, ibovespa_ret_acum_21d, sombra_sup_norm_ibov,
petrobras_ret_acum_10d, ibov_streak]
```



O próximo trecho do nosso código tem como objetivo consolidar uma lista única com as features mais relevantes identificadas por dois modelos diferentes (LightGBM e, se disponível, XGBoost), eliminando duplicatas. Para isso, ele percorre as listas de Top 5 Features de cada regime de mercado contidas nos DataFrames de resultados dos modelos, armazena todas essas variáveis em um conjunto (set), e por fim converte esse conjunto em uma lista final chamada final_feature_list. Essa lista representa um compilado das "features all-star", ou seja, as variáveis mais recorrentes e significativas identificadas durante as análises preditivas.

```
all_top_features = set()

for features_list in df_resultados['Top_5_Features']:
    all_top_features.update(features_list)

if 'df_resultados_xgb' in locals():
    for features_list in df_resultados_xgb['Top_5_Features']:
    all_top_features.update(features_list)

final_feature_list = list(all_top_features)

print(
    f"Total de features únicas e de alta importância selecionadas:
{len(final_feature_list)}")
print("Lista de Features 'All-Star':")
print(final_feature_list)
```

XGBoost

Aplicamos o modelo XGBoost com otimização de hiperparâmetros e seleção de variáveis usando Optuna, para diferentes regimes de mercado. O processo começa com o pré-processamento e divisão dos dados com base na data de início de cada regime. Em seguida, o Optuna é usado para selecionar simultaneamente os melhores hiperparâmetros e o subconjunto ideal de features com maior poder preditivo, medido pela métrica AUC. Após identificar os melhores parâmetros e variáveis, o modelo final é treinado e avaliado, retornando métricas de performance como AUC, acurácia e F1-score. Por fim, os resultados são organizados em um DataFrame e visualizados por meio de gráficos comparativos entre os diferentes regimes.

```
optuna.logging.set_verbosity(optuna.logging.WARNING)
warnings.filterwarnings("ignore")


def otimizar_xgb_com_selecao_features(df_completo, nome_regime,
data_inicio, meses_teste=6, n_trials_optuna=100):

print(f"\n================================================================
=========")
    print(f"--- Processando Regime com XGBoost/Optuna + Feature
Selection: '{nome_regime}' ---")

print(f"================================================================
=======")

    df_periodo = df_completo[df_completo.index >= data_inicio].copy()
    df_periodo.dropna(subset=['target'], inplace=True)
    df_periodo.replace([np.inf, -np.inf], np.nan, inplace=True)
    df_periodo.dropna(inplace=True)

    if len(df_periodo) < 100:
        print(f"Aviso: Período '{nome_regime}' com poucas amostras
({len(df_periodo)}). Pulando regime.")
        return None

    features_candidatas = [col for col in final_feature_list if col in
df_periodo.columns]

    X = df_periodo[features_candidatas]
    y = df_periodo['target']

    data_corte = df_periodo.index.max() -
pd.DateOffset(months=meses_teste)
    X_train, X_test = X[X.index < data_corte], X[X.index >= data_corte]
    y_train, y_test = y[y.index < data_corte], y[y.index >= data_corte]

    scaler = StandardScaler()
    X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train),
index=X_train.index, columns=X_train.columns)
```

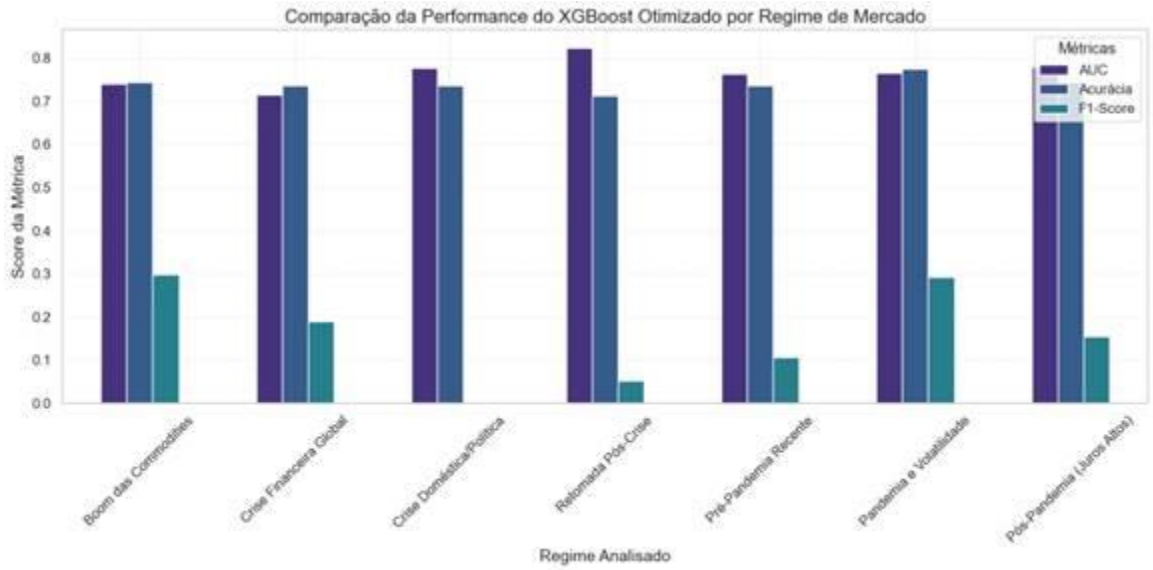
```
    X_test_scaled = pd.DataFrame(scaler.transform(X_test),
index=X_test.index, columns=X_test.columns)

    def objective(trial):
        features_a_usar = [
            feature for feature in features_candidatas
            if trial.suggest_categorical(f"feature_{feature}", [True,
False])
        ]

        if not features_a_usar:
            return 0.0

        X_train_trial = X_train_scaled[features_a_usar]
        X_test_trial = X_test_scaled[features_a_usar]

        params = {
            'objective': 'binary:logistic',
            'eval_metric': 'logloss',
            'use_label_encoder': False,
            'verbosity': 0,
            'random_state': 42,
            'n_estimators': trial.suggest_int('n_estimators', 50,
1000),
            'learning_rate': trial.suggest_float('learning_rate', 0.01,
0.3),
            'max_depth': trial.suggest_int('max_depth', 3, 10),
            'subsample': trial.suggest_float('subsample', 0.6, 1.0),
            'colsample_bytree': trial.suggest_float('colsample_bytree',
0.6, 1.0)
        }

        model = xgb.XGBClassifier(**params)
        model.fit(X_train_trial, y_train)

        y_pred_proba = model.predict_proba(X_test_trial)[:, 1]
        return roc_auc_score(y_test, y_pred_proba)

    print(f"Iniciando otimização para '{nome_regime}' com
{n_trials_optuna} trials...")
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=n_trials_optuna)

    best_params_raw = study.best_params
    best_features = [col.replace('feature_', '') for col, usar in
best_params_raw.items()
                     if col.startswith('feature_') and usar]
    best_hyperparams = {k: v for k, v in best_params_raw.items() if not
k.startswith('feature_')}

    if not best_features:
        print(f"Nenhuma feature selecionada para o regime
'{nome_regime}'. Pulando.")
        return None

    X_train_best = X_train_scaled[best_features]
    X_test_best = X_test_scaled[best_features]
```

```
    final_model = xgb.XGBClassifier(**best_hyperparams,
random_state=42, use_label_encoder=False, verbosity=0)
    final_model.fit(X_train_best, y_train)

    y_pred = final_model.predict(X_test_best)
    y_pred_proba = final_model.predict_proba(X_test_best)[:, 1]

    auc = roc_auc_score(y_test, y_pred_proba)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"\n--- Resultados Finais para o Regime: '{nome_regime}' ---
")
    print(f"Melhor AUC encontrado na otimização:
{study.best_value:.4f}")
    print(classification_report(y_test, y_pred, target_names=['Baixa',
'Alta']))

    return {
        'Regime': nome_regime,
        'AUC': auc,
        'Acurácia': accuracy,
        'F1-Score': f1,
        'Melhores_Features': best_features,
        'Melhores_Hiperparametros': best_hyperparams
    }


resultados_finais_xgb = []

for nome, data in regimes_para_analise.items():
    resultado = otimizar_xgb_com_selecao_features(df, nome, data,
n_trials_optuna=100)
    if resultado:
        resultados_finais_xgb.append(resultado)

if resultados_finais_xgb:
    df_resultados_xgb = pd.DataFrame(resultados_finais_xgb)

    print("\n\n--- COMPARAÇÃO FINAL DOS REGIMES (XGBoost) E SUAS
PRINCIPAIS FEATURES ---")
    pd.set_option('display.max_colwidth', None)
    print(df_resultados_xgb[['Regime', 'AUC', 'Acurácia', 'F1-Score',
'Melhores_Features']])

    df_plot_xgb = df_resultados_xgb.set_index('Regime')[['AUC',
'Acurácia', 'F1-Score']]
    df_plot_xgb.plot(kind='bar', figsize=(14, 7), rot=45)
    plt.title('Comparação da Performance do XGBoost Otimizado por
Regime de Mercado')
    plt.ylabel('Score da Métrica')
    plt.xlabel('Regime Analisado')
    plt.grid(axis='y', linestyle='--')
    plt.legend(title='Métricas')
    plt.tight_layout()
    plt.show()
```

O retorno foi:

```
 ======================================================================
--- Processando Regime com XGBoost/Optuna + Feature Selection: 'Boom das Commodities' ---
======================================================================
Iniciando otimização para 'Boom das Commodities' com 100 trials...

--- Resultados Finais para o Regime: 'Boom das Commodities' ---
Melhor AUC encontrado na otimização: 0.7412
              precision    recall  f1-score   support

           Baixa       0.77      0.94      0.84        95
           Alta       0.54      0.21      0.30        34

    accuracy                           0.74       129
   macro avg       0.65      0.57      0.57       129
weighted avg       0.71      0.74      0.70       129


======================================================================
--- Processando Regime com XGBoost/Optuna + Feature Selection: 'Crise Financeira Global' ---
======================================================================
Iniciando otimização para 'Crise Financeira Global' com 100 trials...

--- Resultados Finais para o Regime: 'Crise Financeira Global' ---
Melhor AUC encontrado na otimização: 0.7161
...
3   [petrobras_ret_acum_2d, tendencia_dolar_longo_prazo_42d, choque_de_volume_log,
ibovespa_ret_acum_21d, sp500_ret_diario_lag_2, media_movel_alta,
momento_ibov_mes_anterior_d-21, ibov_streak, sombra_sup_norm_ibov, ibov_ewm_5,
petrobras_ret_diario_lag_5, ibov_vs_ma7]
4   [petrobras_ret_acum_2d, tendencia_dolar_longo_prazo_42d, choque_de_volume_log,
ibovespa_ret_acum_21d, media_movel_alta, momento_ibov_mes_anterior_d-21, ibov_streak,
sombra_sup_norm_ibov, ibov_ewm_5, petrobras_ret_diario_lag_5, petrobras_ret_acum_10d,
ibov_vs_ma7, spread_ibov_dolar_21d, momento_dolar_mes_anterior_d-15]
5   [petrobras_ret_acum_2d, ibovespa_ret_acum_21d, sp500_ret_diario_lag_2,
media_movel_alta, momento_ibov_mes_anterior_d-21, ibov_streak, ibov_vs_ma50,
ibov_ewm_5, petrobras_ret_diario_lag_5, ibov_vs_ma7, spread_ibov_dolar_21d]
6   [petroleo_brent_ret_diario_lag_2, tendencia_dolar_longo_prazo_42d,
choque_de_volume_log, ibovespa_ret_acum_21d, media_movel_alta,
momento_ibov_mes_anterior_d-21, candle_body_ibov, ibov_streak, ibov_ewm_5,
petrobras_ret_diario_lag_5]
```

Para mais detalhes consultar notebook 4



Tensorflow

O Tensorflow implementa um pipeline completo de modelagem preditiva usando redes neurais LSTM otimizadas com Optuna. A lógica segue quatro etapas principais: preparação dos dados históricos da Ibovespa, seleção dinâmica de variáveis relevantes por regime de mercado, otimização simultânea da arquitetura da LSTM (como número de unidades, dropout e time steps) e dos hiperparâmetros, e finalmente treinamento e avaliação do melhor modelo encontrado com base nas métricas de AUC, acurácia e F1-Score. O objetivo central é identificar o conjunto ideal de variáveis e configurações que melhor capturam os padrões temporais do mercado em diferentes regimes, maximizando o poder preditivo do modelo. Ao final, os resultados são agregados e comparados visualmente por regime;

```
optuna.logging.set_verbosity(optuna.logging.WARNING)
warnings.filterwarnings("ignore")

def otimizar_lstm_com_selecao_features(df_completo, nome_regime,
data_inicio, meses_teste=6, n_trials_optuna=50):

print(f"\n==================================================================
=========")
    print(f"--- Processando Regime com LSTM/Optuna: '{nome_regime}' ---
")

print(f"==================================================================
=======")

    df_periodo = df_completo[df_completo.index >= data_inicio].copy()
    df_periodo.dropna(subset=['target'], inplace=True)
    df_periodo.replace([np.inf, -np.inf], np.nan, inplace=True)
    df_periodo.dropna(inplace=True)

    if len(df_periodo) < 100:
        print(
```

```
            f"Aviso: Período '{nome_regime}' com poucas amostras
({len(df_periodo)}). Pulando regime.")
        return None

    features_candidatas = [
        col for col in final_feature_list if col in df_periodo.columns]
    X = df_periodo[features_candidatas]
    y = df_periodo['target']

    data_corte = df_periodo.index.max() -
pd.DateOffset(months=meses_teste)
    X_train, X_test = X[X.index < data_corte], X[X.index >= data_corte]
    y_train, y_test = y[y.index < data_corte], y[y.index >= data_corte]

    scaler = StandardScaler()
    X_train_scaled = pd.DataFrame(scaler.fit_transform(
        X_train), index=X_train.index, columns=X_train.columns)
    X_test_scaled = pd.DataFrame(scaler.transform(
        X_test), index=X_test.index, columns=X_test.columns)

    def create_sequences(X, y, time_steps=10):
        Xs, ys = []
        for i in range(len(X) - time_steps):
            Xs.append(X.iloc[i:(i + time_steps)].values)
            ys.append(y.iloc[i + time_steps])
        return np.array(Xs), np.array(ys)

    def objective(trial):
        features_a_usar = [
            feature for feature in features_candidatas
            if trial.suggest_categorical(f"feature_{feature}", [True,
False])
        ]

        if not features_a_usar:
            return 0.0

        X_train_trial = X_train_scaled[features_a_usar]
        X_test_trial = X_test_scaled[features_a_usar]

        time_steps = trial.suggest_int('time_steps', 5, 20)
        lstm_units = trial.suggest_int('lstm_units', 32, 128)
        dropout_rate = trial.suggest_float('dropout_rate', 0.1, 0.5)
        learning_rate = trial.suggest_float(
            'learning_rate', 1e-4, 1e-2, log=True)

        X_train_seq, y_train_seq = create_sequences(
            X_train_trial, y_train, time_steps)
        X_test_seq, y_test_seq = create_sequences(
            X_test_trial, y_test, time_steps)

        if len(X_train_seq) == 0 or len(X_test_seq) == 0:
            return 0.0

        tf.random.set_seed(42)
        model = Sequential([
            LSTM(units=lstm_units, input_shape=(
```

```
                time_steps, X_train_seq.shape[2])),
            Dropout(dropout_rate),
            Dense(1, activation='sigmoid')
        ])
        model.compile(loss='binary_crossentropy', optimizer=Adam(
            learning_rate=learning_rate), metrics=['AUC'])
        model.fit(X_train_seq, y_train_seq, epochs=20,
                  batch_size=32, verbose=0, shuffle=False)

        y_pred_proba = model.predict(X_test_seq, verbose=0).flatten()

        if len(y_test_seq) != len(y_pred_proba):
            return 0.0

        return roc_auc_score(y_test_seq, y_pred_proba)

    print(
        f"Iniciando otimização da LSTM para '{nome_regime}' com
{n_trials_optuna} trials...")
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=n_trials_optuna)

    print("\nOtimização concluída. Treinando o melhor modelo para
avaliação final...")
    best_params = study.best_params

    best_features_names = [col.replace('feature_', '') for col, usar in
best_params.items(
    ) if col.startswith('feature_') and usar]
    best_hiperparams = {
        k: v for k, v in best_params.items() if not
k.startswith('feature_')}

    if not best_features_names:
        print(
            f"Nenhuma feature selecionada para o regime
'{nome_regime}'. Pulando.")
        return None

    X_train_best = X_train_scaled[best_features_names]
    X_test_best = X_test_scaled[best_features_names]
    best_time_steps = best_hiperparams['time_steps']

    X_train_seq, y_train_seq = create_sequences(
        X_train_best, y_train, best_time_steps)
    X_test_seq, y_test_seq = create_sequences(
        X_test_best, y_test, best_time_steps)

    tf.random.set_seed(42)
    final_model = Sequential([
        LSTM(units=best_hiperparams['lstm_units'], input_shape=(
            best_time_steps, X_train_seq.shape[2])),
        Dropout(rate=best_hiperparams['dropout_rate']),
        Dense(1, activation='sigmoid')
    ])
    final_model.compile(loss='binary_crossentropy', optimizer=Adam(
```

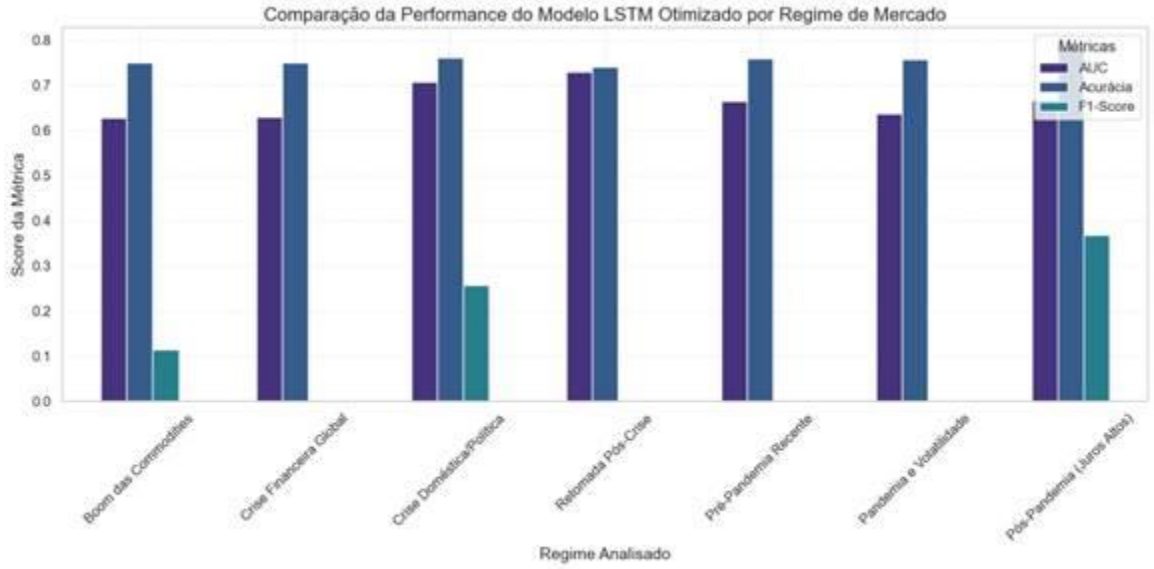
```
        learning_rate=best_hiperparams['learning_rate']),
metrics=['AUC'])
    final_model.fit(X_train_seq, y_train_seq, epochs=20,
                    batch_size=32, verbose=0, shuffle=False)

    y_pred_proba = final_model.predict(X_test_seq, verbose=0).flatten()
    y_pred = (y_pred_proba > 0.5).astype(int)

    auc = roc_auc_score(y_test_seq, y_pred_proba)
    accuracy = accuracy_score(y_test_seq, y_pred)
    f1 = f1_score(y_test_seq, y_pred)

    print(f"\n--- Resultados Finais para o Regime: '{nome_regime}' ---
")
    print(f"Melhor AUC encontrado na otimização:
{study.best_value:.4f}")
    print(classification_report(y_test_seq,
          y_pred, target_names=['Baixa', 'Alta']))

    return {
        'Regime': nome_regime,
        'AUC': auc,
        'Acurácia': accuracy,
        'F1-Score': f1,
        'Melhores_Features': best_features_names,
        'Melhores_Hiperparametros': best_hiperparams
    }


resultados_finais_lstm = []
for nome, data in regimes_para_analise.items():
    resultado = otimizar_lstm_com_selecao_features(
        df, nome, data, n_trials_optuna=50)
    if resultado:
        resultados_finais_lstm.append(resultado)

if resultados_finais_lstm:
    df_resultados_lstm = pd.DataFrame(resultados_finais_lstm)

    print("\n\n--- COMPARAÇÃO FINAL DOS REGIMES (LSTM) E SUAS
PRINCIPAIS FEATURES ---")
    pd.set_option('display.max_colwidth', None)
    print(df_resultados_lstm[['Regime', 'AUC',
          'Acurácia', 'F1-Score', 'Melhores_Features']])

    df_plot_lstm = df_resultados_lstm.set_index(
        'Regime')[['AUC', 'Acurácia', 'F1-Score']]
    df_plot_lstm.plot(kind='bar', figsize=(14, 7), rot=45)
    plt.title(
        'Comparação da Performance do Modelo LSTM Otimizado por Regime
de Mercado')
    plt.ylabel('Score da Métrica')
    plt.xlabel('Regime Analisado')
    plt.grid(axis='y', linestyle='--')
    plt.legend(title='Métricas')
    plt.tight_layout()
    plt.show()
```

O retorno foi:

```
================================================================
--- Processando Regime com LSTM/Optuna: 'Boom das Commodities' ---
================================================================
Iniciando otimização da LSTM para 'Boom das Commodities' com 50 trials...
WARNING:tensorflow:5 out of the last 9 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x000001B5885FE8E0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True
option that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 12 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x000001B5885FE8E0> triggered tf.function retracing. Tracing is expensive and the excessive
number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing
tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True
option that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.

Otimização concluída. Treinando o melhor modelo para avaliação final...

--- Resultados Finais para o Regime: 'Boom das Commodities' ---
Melhor AUC encontrado na otimização: 0.6471
              precision    recall  f1-score   support

       Baixa       0.75      0.99      0.85        92
        Alta       0.67      0.06      0.11        32

    accuracy                           0.75       124
   macro avg       0.71      0.53      0.48       124
weighted avg       0.73      0.75      0.66       124

 ================================================================
--- Processando Regime com LSTM/Optuna: 'Crise Financeira Global' ---
================================================================
...
3    [tendencia_dolar_longo_prazo_42d, choque_de_volume_log, sp500_ret_diario_lag_2,
sombra_sup_norm_ibov, ibov_vs_ma50]
4    [momento_ibov_mes_anterior_d-21, dolar_ret_diario_lag_3, ibov_streak,
sp500_ret_diario_lag_1, sombra_sup_norm_ibov, ibov_vs_ma50, petrobras_ret_acum_10d,
ibov_vs_ma7, momento_dolar_mes_anterior_d-15]
```

5 [petrobras_ret_acum_2d, choque_de_volume_log, sp500_ret_diario_lag_2, momento_ibov_mes_anterior_d-21, dolar_ret_diario_lag_3, ibov_streak, sp500_ret_diario_lag_1, sombra_sup_norm_ibov, spread_ibov_dolar_21d, momento_dolar_mes_anterior_d-15]
6 [petroleo_brent_ret_diario_lag_2, media_movel_alta, dolar_ret_diario_lag_3, sp500_ret_diario_lag_1, ibov_vs_ma50, ibov_ewm_5, petrobras_ret_acum_10d, ibov_vs_ma7]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings…

Para mais detalhes consultar notebook 4



Logistic Regression

Temos como objetivo aplicar um processo completo de *AutoML* com Optuna para uma Regressão Logística, combinando seleção automática de features e otimização de hiperparâmetros, com o intuito de encontrar o melhor modelo preditivo possível para diferentes regimes de mercado da base histórica do Ibovespa. O raciocínio parte da filtragem e preparação dos dados por regime, separando os períodos de treino e teste, escalando os dados e aplicando Optuna para avaliar diversas combinações de variáveis e parâmetros do modelo. Após a otimização, o melhor conjunto de features e hiperparâmetros é utilizado para treinar e avaliar um modelo final, cujos resultados são armazenados e comparados entre os regimes por meio de métricas como AUC, acurácia e F1-score, além de uma visualização gráfica que resume a performance por regime analisado.

```
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

```
def otimizar_logistica_com_selecao_features(df_completo, nome_regime,
data_inicio, meses_teste=6, n_trials_optuna=100):
    print(f"\n====================================================")
    print(
        f"--- Processando Regime com Regressão Logística/Optuna:
'{nome_regime}' ---")
    print(f"\n====================================================")
    df_periodo = df_completo[df_completo.index >= data_inicio].copy()
    df_periodo.dropna(subset=['target'], inplace=True)
    df_periodo.replace([np.inf, -np.inf], np.nan, inplace=True)
    df_periodo.dropna(inplace=True)

    if len(df_periodo) < 100:
        print(
            f"Aviso: Período '{nome_regime}' com poucas amostras
({len(df_periodo)}). Pulando regime.")
        return None

    features_candidatas = [
        col for col in final_feature_list if col in df_periodo.columns]
    X = df_periodo[final_feature_list]
    y = df_periodo['target']

    data_corte = df_periodo.index.max() -
pd.DateOffset(months=meses_teste)
    X_train, X_test = X[X.index < data_corte], X[X.index >= data_corte]
    y_train, y_test = y[y.index < data_corte], y[y.index >= data_corte]

    scaler = StandardScaler()
    X_train_scaled = pd.DataFrame(scaler.fit_transform(
        X_train), index=X_train.index, columns=X_train.columns)
    X_test_scaled = pd.DataFrame(scaler.transform(
        X_test), index=X_test.index, columns=X_test.columns)

    def objective(trial):
        features_a_usar = [
            feature for feature in features_candidatas
            if trial.suggest_categorical(f"feature_{feature}", [True,
False])
        ]

        if not features_a_usar:
            return 0.0

        X_train_trial = X_train_scaled[features_a_usar]
        X_test_trial = X_test_scaled[features_a_usar]

        c_param = trial.suggest_float('C', 1e-4, 1e2, log=True)
        penalty_param = trial.suggest_categorical('penalty', ['l1',
'l2'])
        solver_param = 'saga'

        model = LogisticRegression(
            C=c_param,
            penalty=penalty_param,
            solver=solver_param,
```

```
            class_weight='balanced',
            random_state=42,
            max_iter=1000
        )
        model.fit(X_train_trial, y_train)

        y_pred_proba = model.predict_proba(X_test_trial)[:, 1]
        return roc_auc_score(y_test, y_pred_proba)

    print(
        f"Iniciando otimização para '{nome_regime}' com
{n_trials_optuna} trials...")
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=n_trials_optuna)

    best_params = study.best_params
    best_features = [col.replace('feature_', '') for col, usar in
best_params.items(
    ) if col.startswith('feature_') and usar]

    if not best_features:
        print(
            f"Nenhuma feature selecionada para o regime
'{nome_regime}'. Pulando.")
        return None

    X_train_best = X_train_scaled[best_features]
    X_test_best = X_test_scaled[best_features]

    final_model = LogisticRegression(
        C=best_params.get('C'),
        penalty=best_params.get('penalty'),
        solver='saga',
        class_weight='balanced',
        random_state=42,
        max_iter=1300
    )
    final_model.fit(X_train_best, y_train)
    y_pred = final_model.predict(X_test_best)
    y_pred_proba = final_model.predict_proba(X_test_best)[:, 1]

    auc = roc_auc_score(y_test, y_pred_proba)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"\n--- Resultados Finais para o Regime: '{nome_regime}' ---
")
    print(f"Melhor AUC encontrado na otimização:
{study.best_value:.4f}")
    print(classification_report(
        y_test, y_pred, target_names=['Baixa', 'Alta']))

    return {
        'Regime': nome_regime,
        'AUC': auc,
        'Acurácia': accuracy,
        'F1-Score': f1,
```

```
        'Top_Features': best_features,
        'Melhores_Hiperparametros': {k: v for k, v in
best_params.items() if not k.startswith('feature_')}
    }


resultados_finais = []
for nome, data in regimes_para_analise.items():
    resultado = otimizar_logistica_com_selecao_features(
        df, nome, data, n_trials_optuna=100)
    if resultado:
        resultados_finais.append(resultado)

if resultados_finais:
    df_resultados = pd.DataFrame(resultados_finais)
    pd.set_option('display.max_colwidth', None)

    print("\n\n--- COMPARAÇÃO FINAL DOS REGIMES (Regressão Logística) -
--")
    print(df_resultados[['Regime', 'AUC',
          'Acurácia', 'F1-Score', 'Top_Features']])

    df_plot = df_resultados.set_index(
        'Regime')[['AUC', 'Acurácia', 'F1-Score']]
    df_plot.plot(kind='bar', figsize=(14, 7), rot=45)
    plt.title(
        'Comparação da Performance da Regressão Logística por Regime de
Mercado')
    plt.ylabel('Score da Métrica')
    plt.xlabel('Regime Analisado')
    plt.grid(axis='y', linestyle='--')
    plt.legend(title='Métricas')
    plt.tight_layout()
    plt.show()
```

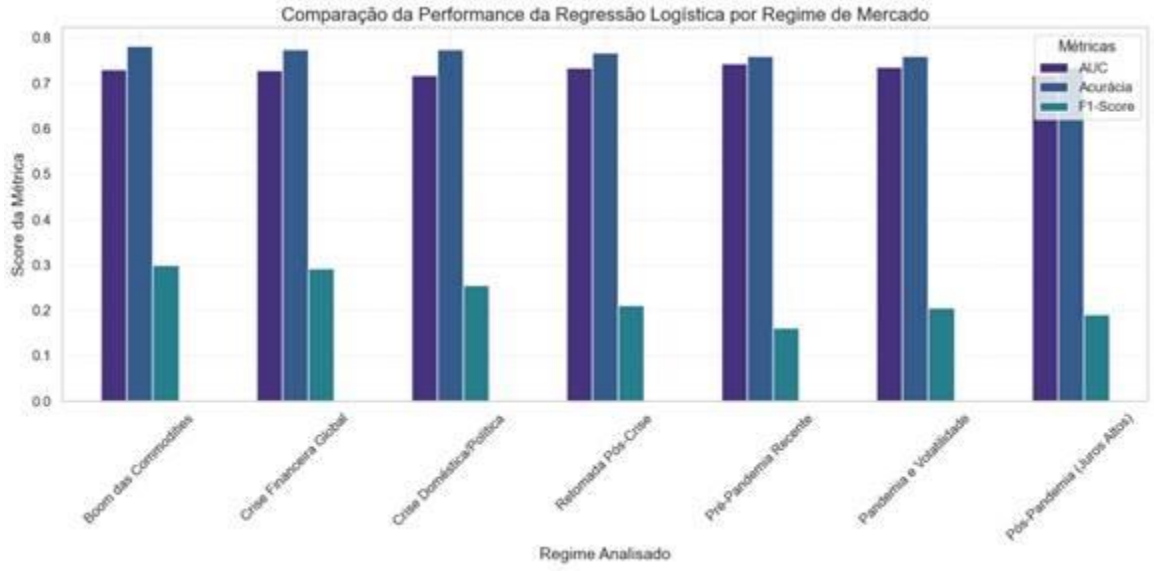
O retorno foi:

```
 ======================================================================
--- Processando Regime com Regressão Logística/Optuna: 'Boom das Commodities' ---
======================================================================
Iniciando otimização para 'Boom das Commodities' com 100 trials...

--- Resultados Finais para o Regime: 'Boom das Commodities' ---
Melhor AUC encontrado na otimização: 0.7313


precision    recall  f1-score   support

 Baixa       0.77    1.00      0.87        95
 Alta        1.00    0.18      0.30        34

accuracy                       0.78       129
```

macro avg	0.89	0.59	0.59	129
weighted avg	0.83	0.78	0.72	129

```
======================================================================
--- Processando Regime com Regressão Logística/Optuna: 'Crise Financeira Global' ---
======================================================================
Iniciando otimização para 'Crise Financeira Global' com 100 trials...

--- Resultados Finais para o Regime: 'Crise Financeira Global' ---
Melhor AUC encontrado na otimização: 0.7285
...
3    [tendencia_dolar_longo_prazo_42d, ibovespa_ret_acum_21d, media_movel_alta,
momento_ibov_mes_anterior_d-21, dolar_ret_diario_lag_3, ibov_streak,
sombra_sup_norm_ibov, ibov_vs_ma50, ibov_ewm_5]
4    [choque_de_volume_log, ibovespa_ret_acum_21d, media_movel_alta,
dolar_ret_diario_lag_3, ibov_streak, sp500_ret_diario_lag_1, sombra_sup_norm_ibov,
ibov_ewm_5, petrobras_ret_diario_lag_5, petrobras_ret_acum_10d, spread_ibov_dolar_21d,
momento_dolar_mes_anterior_d-15]
5    [tendencia_dolar_longo_prazo_42d, ibovespa_ret_acum_21d, media_movel_alta,
momento_ibov_mes_anterior_d-21, ibov_streak, sp500_ret_diario_lag_1,
sombra_sup_norm_ibov, ibov_ewm_5, spread_ibov_dolar_21d,
momento_dolar_mes_anterior_d-15]
6    [tendencia_dolar_longo_prazo_42d, choque_de_volume_log, ibovespa_ret_acum_21d,
sp500_ret_diario_lag_2, media_movel_alta, candle_body_ibov, ibov_streak,
sp500_ret_diario_lag_1, sombra_sup_norm_ibov, ibov_vs_ma50, ibov_ewm_5,
petrobras_ret_acum_10d]
```

Para mais detalhes consultar notebook 4



Conclusão - Escolha final do modelo

Análise Comparativa e Seleção do Modelo Final

Ao longo deste projeto, nossa abordagem evoluiu de uma fase inicial de modelagem e otimização para uma etapa final de validação rigorosa. Essa transição revelou insights cruciais e alterou nossa percepção sobre qual era o modelo mais eficaz.

Inicialmente, sem uma separação formal entre dados de validação e teste, os modelos LSTM otimizados com Optuna aparentavam ser os mais promissores. Contudo, a implementação de uma metodologia de validação mais robusta, com conjuntos de treino, validação e teste distintos, nos permitiu identificar o verdadeiro desempenho de cada algoritmo, levando a uma nova conclusão.

Após a validação final, a **Regressão Logística treinada no regime 'Pós-Pandemia (Juros Altos)'** emergiu como o modelo mais promissor e útil na prática.

Por que o Modelo 'Pós-Pandemia (Juros Altos)' é o Melhor?

A superioridade deste modelo não está nas métricas superficiais como acurácia ou AUC, mas sim na sua performance nas métricas que realmente importam para o nosso objetivo de negócio.

- **Recall da Classe "Alta" Muito Superior:**
  - **Modelo 'Pós-Pandemia':** recall de **0.41** (41%).
  - **Modelo 'Retomada Pós-Crise':** recall de 0.18 (18%).

  Interpretação: Este é o fator decisivo. O modelo 'Pós-Pandemia' foi capaz de identificar corretamente 41% de todos os dias que realmente foram de alta no seu conjunto de teste. Em comparação, o modelo do regime 'Retomada Pós-Crise' só encontrou 18%[4]. Na prática, o modelo 'Pós-Pandemia' é muito mais eficaz em sinalizar oportunidades de mercado.
- **Melhor F1-Score para "Alta":**
  - **Modelo 'Pós-Pandemia':** F1-Score de **0.39**.
  - **Modelo 'Retomada Pós-Crise':** F1-Score de **0.28**.

  Interpretação: O F1-Score, que balanceia precisão e recall, confirma a superioridade do modelo 'Pós-Pandemia', indicando um melhor equilíbrio para a classe minoritária, que é a mais difícil de prever.
- O Paradoxo do AUC e o Trade-off Inteligente:
  É notável que o modelo 'Pós-Pandemia' tenha um AUC ROC mais baixo (0.6341) que o modelo 'Retomada' (0.6783)9. Isso não é um sinal de fraqueza. Pelo contrário, demonstra que nossa estratégia de usar

class_weight='balanced' funcionou como esperado. O modelo foi forçado a ser menos "pessimista", sacrificando parte de sua capacidade geral de separação de probabilidades (AUC) para aumentar drasticamente o recall da classe "Alta". Para este problema de negócio, esse é um trade-off excelente e desejável.

Veredito Final

A implementação de um processo de validação rigoroso foi fundamental para descartar modelos que pareciam bons, mas eram enganosos. O modelo de **Regressão Logística para o regime 'Pós-Pandemia (Juros Altos)'** foi o único que, após a validação, demonstrou potencial prático real, pois consegue identificar uma porção significativa dos dias de alta. Ele é, portanto, o modelo escolhido para as próximas etapas de refinamento e para a apresentação final do projeto.