

Tech Challenge - FIAP / Fase 02

Grupo 03 - Pos Tech Fiap

Equipe:

- André Antônio Campos
- Clayton Gonçalves dos Santos
- Debora Fabiana Pascoarelli
- Igor Torves
- Tamires Cristofani Suhadoinik

Previsão de Fechamento Diário da IBOVESPA

Introdução

O objetivo deste projeto é desenvolver um modelo preditivo para prever o fechamento diário da IBOVESPA utilizando dados históricos. Os dados utilizados incluem o fechamento diário do índice Bovespa, abrangendo o período de 2004 a 2024. Esta análise busca não apenas prever os valores futuros, mas também identificar padrões históricos e anomalias nos dados, fornecendo insights valiosos para a tomada de decisões financeiras.

Visão Geral dos Dados (EDA):

Coleta de Dados

Os dados foram capturados do site Investing.com, abrangendo o período de 01/01/2000 a 01/01/2024.

In [105...]

```
# importando as bibliotecas necessarias
import pandas as pd
import matplotlib.pyplot as plt
```

In [106...]

```
df_path = 'https://raw.githubusercontent.com/Tamireees/Tech_Challenge_Fase02/main/Dados%20His...
```

In [107...]

```
dados = pd.read_csv(df_path, sep=',', parse_dates=[0], dayfirst=True, index_col='Data')
```

In [108...]

```
dados = dados.sort_index(ascending=True)
```

In [109...]

```
dados.head(5)
```

Out[109...]

	Último	Abertura	Máxima	Mínima	Vol.	Var%
--	--------	----------	--------	--------	------	------

Data

2004-01-02	22.445	22.233	22.450	22.208	136,42M	0,94%
2004-01-05	23.532	22.445	23.532	22.445	496,71M	4,84%
2004-01-06	23.576	23.532	23.976	23.180	472,38M	0,19%
2004-01-07	23.320	23.576	23.899	23.320	464,08M	-1,09%
2004-01-08	23.717	23.334	23.718	23.122	436,74M	1,70%

In [110...]

dados.shape

Out[110...]

(4974, 6)

Exploração e limpeza dos dados

In [111...]

dados.info()

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4974 entries, 2004-01-02 to 2024-02-01
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Último       4974 non-null   float64
 1   Abertura    4974 non-null   float64
 2   Máxima      4974 non-null   float64
 3   Mínima      4974 non-null   float64
 4   Vol.         4973 non-null   object  
 5   Var%         4974 non-null   object  
dtypes: float64(4), object(2)
memory usage: 272.0+ KB
```

In [112...]

```
def convert_to_float(value):
    if isinstance(value, str):
        value = value.replace('.', '').replace(',', '.')
        if 'M' in value:
            return float(value.replace('M', '')) * 1_000_000
        elif 'K' in value:
            return float(value.replace('K', '')) * 1_000
    return value
```

In [113...]

```
dados['Vol.'] = dados['Vol.'].apply(convert_to_float)
print(dados.head())
```

Data	Último	Abertura	Máxima	Mínima	Vol.	Var%
2004-01-02	22.445	22.233	22.450	22.208	136420000.0	0,94%
2004-01-05	23.532	22.445	23.532	22.445	496710000.0	4,84%
2004-01-06	23.576	23.532	23.976	23.180	472380000.0	0,19%
2004-01-07	23.320	23.576	23.899	23.320	464080000.0	-1,09%
2004-01-08	23.717	23.334	23.718	23.122	436740000.0	1,70%

In [114...]

```
def convert_percentage_to_float(value):
    if isinstance(value, str) and '%' in value:
        value = value.replace('%', '').replace(',', '.')
        if '-' in value:
            value = value.replace('-', '')
    return float(value) * -1
```

```
    else:  
        return float(value)  
    return value
```

In [115...]

```
dados['Var%'] = dados['Var%'].apply(convert_percentage_to_float)  
print(dados.head())
```

Data	Último	Abertura	Máxima	Mínima	Vol.	Var%
2004-01-02	22.445	22.233	22.450	22.208	136420000.0	0.94
2004-01-05	23.532	22.445	23.532	22.445	496710000.0	4.84
2004-01-06	23.576	23.532	23.976	23.180	472380000.0	0.19
2004-01-07	23.320	23.576	23.899	23.320	464080000.0	-1.09
2004-01-08	23.717	23.334	23.718	23.122	436740000.0	1.70

In [116...]

```
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 4974 entries, 2004-01-02 to 2024-02-01  
Data columns (total 6 columns):  
 #   Column      Non-Null Count  Dtype     
---  --          --          --  
 0   Último      4974 non-null   float64  
 1   Abertura    4974 non-null   float64  
 2   Máxima      4974 non-null   float64  
 3   Mínima      4974 non-null   float64  
 4   Vol.         4973 non-null   float64  
 5   Var%         4974 non-null   float64  
dtypes: float64(6)  
memory usage: 272.0 KB
```

In [117...]

```
dados.head()
```

Out[117...]

Data	Último	Abertura	Máxima	Mínima	Vol.	Var%
------	--------	----------	--------	--------	------	------

2004-01-02	22.445	22.233	22.450	22.208	136420000.0	0.94
2004-01-05	23.532	22.445	23.532	22.445	496710000.0	4.84
2004-01-06	23.576	23.532	23.976	23.180	472380000.0	0.19
2004-01-07	23.320	23.576	23.899	23.320	464080000.0	-1.09
2004-01-08	23.717	23.334	23.718	23.122	436740000.0	1.70

In [118...]

```
dados.describe().T
```

Out[118...]

	count	mean	std	min	25%	50%	75%
Último	4974.0	6.711666e+01	2.825580e+01	17.604	4.918100e+01	6.076400e+01	8.641500e+01
Abertura	4974.0	6.709595e+01	2.824949e+01	17.607	4.917550e+01	6.076250e+01	8.639650e+01
Máxima	4974.0	6.776688e+01	2.844366e+01	18.387	4.970075e+01	6.136650e+01	8.746175e+01
Mínima	4974.0	6.643207e+01	2.805835e+01	17.601	4.851325e+01	6.008150e+01	8.574500e+01
Vol.	4973.0	3.438505e+07	7.981133e+07	112100.000	2.820000e+06	4.450000e+06	1.215000e+07
Var%	4974.0	4.982509e-02	1.703218e+00	-14.780	-8.300000e-01	7.000000e-02	9.800000e-01

◀ ▶

In [119...]

```
dados.isnull().sum()
```

Out[119...]

Último	0
Abertura	0
Máxima	0
Mínima	0
Vol.	1
Var%	0

dtype: int64

Essa análise preliminar sugere a necessidade de uma investigação mais detalhada para identificar os fatores que causaram essas variações extremas e as tendências de longo prazo no valor de fechamento da Bovespa.

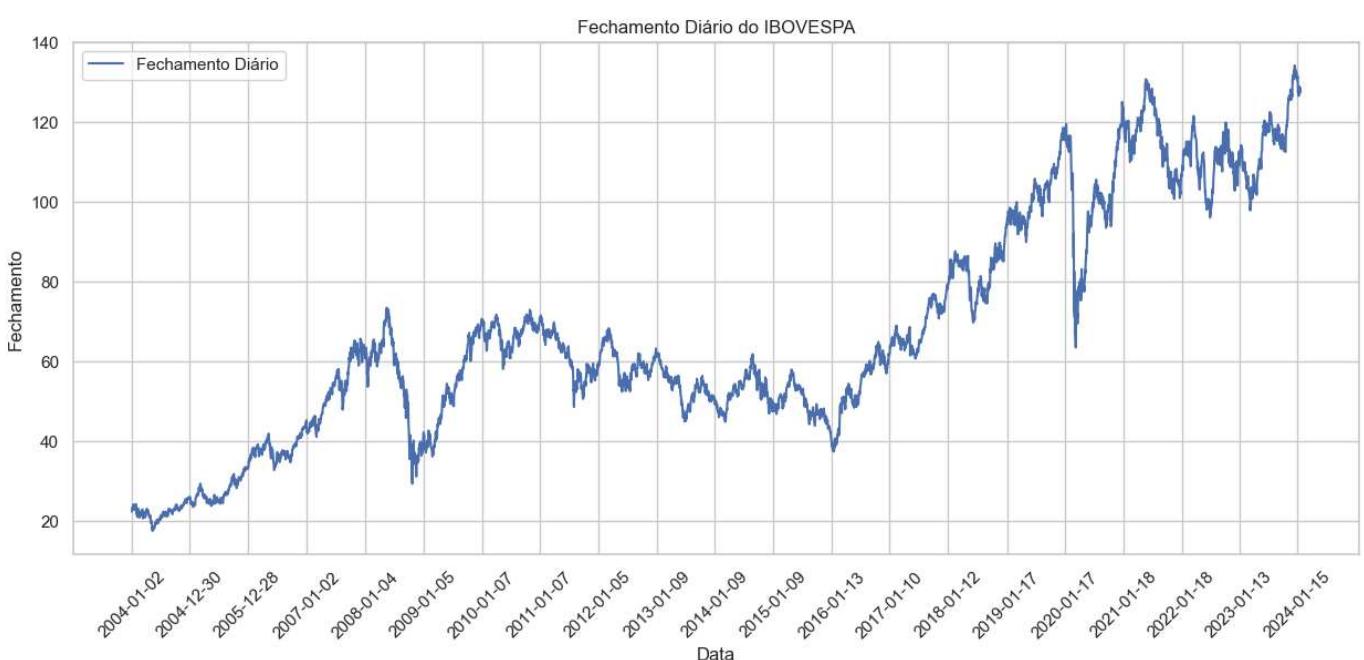
In [120...]

```
plt.figure(figsize=(15, 6))
plt.plot(dados.index, dados['Último'], label='Fechamento Diário')
plt.title('Fechamento Diário do IBOVESPA')
plt.xlabel('Data')
plt.ylabel('Fechamento')

n_labels = 20
labels = dados.index[::-len(dados) // n_labels]

plt.xticks(labels, rotation=45)

plt.legend()
plt.show()
```



Conclusões:

Crescimento e Alta Volatilidade: O gráfico mostra um crescimento significativo na Ibovespa entre 2004 e 2008, seguido por uma volatilidade considerável e oscilações acentuadas entre 2009 e 2016. Esses períodos indicam uma alta volatilidade e eventos de mercado que causaram variações significativas nos valores.

Recuperação e Novos Picos: Após uma recuperação em 2009, a Ibovespa atingiu novos picos entre 2017 e 2019, antes de enfrentar uma queda abrupta no final de 2019 e início de 2020. Isso sugere que, após períodos de alta, o mercado pode enfrentar correções significativas.

Tendência Recente: Desde 2021, a Ibovespa tem apresentado uma tendência de alta, com valores oscilando entre 100 e 130 USD. Essa faixa sugere um mercado mais estável em comparação com os anos anteriores, embora ainda possa haver volatilidade e variações.

Anomalias e Outliers: O pico de 2008 e a queda abrupta no início de 2020 são pontos de interesse, pois indicam eventos extraordinários que merecem uma investigação mais detalhada para entender os fatores subjacentes que causaram essas anomalias.

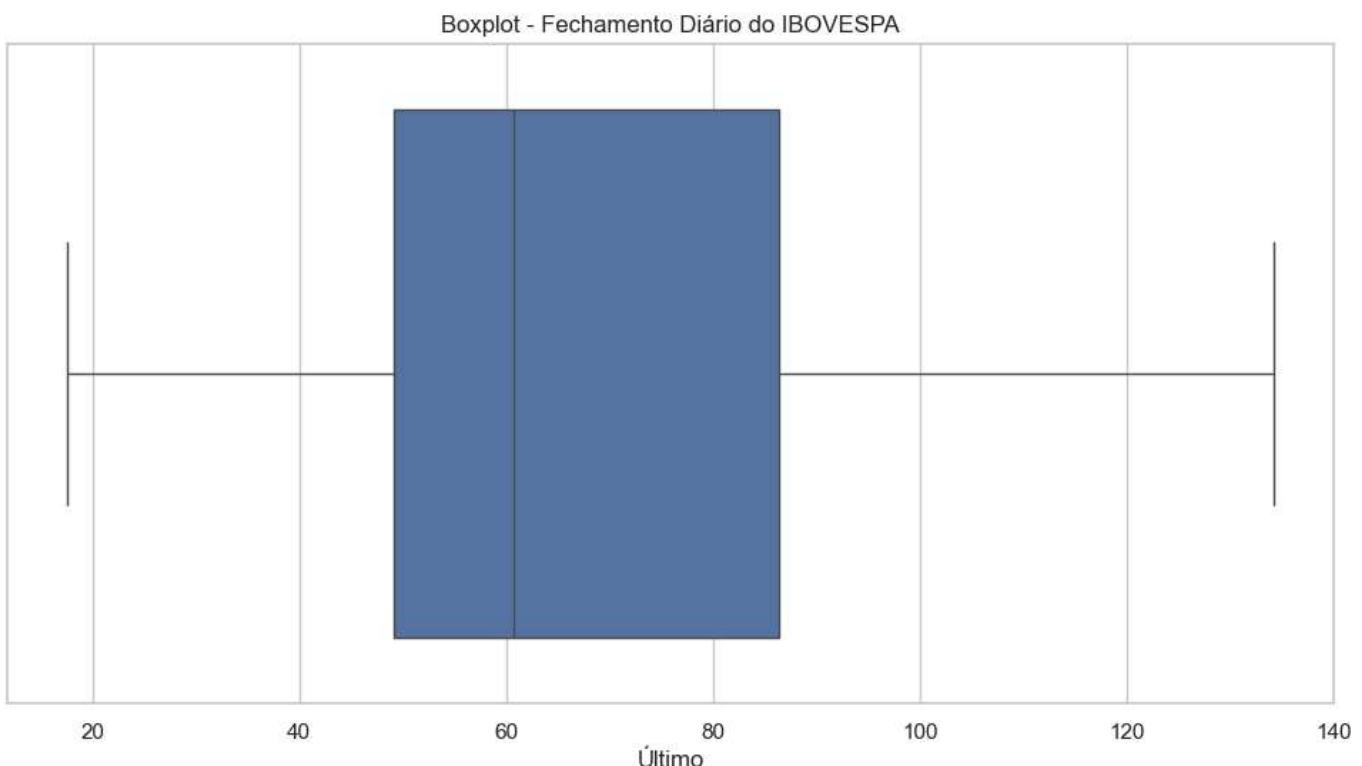
Gráfico Boxplot para Detectar Outliers:

In [121...]

```
import seaborn as sns
```

In [122...]

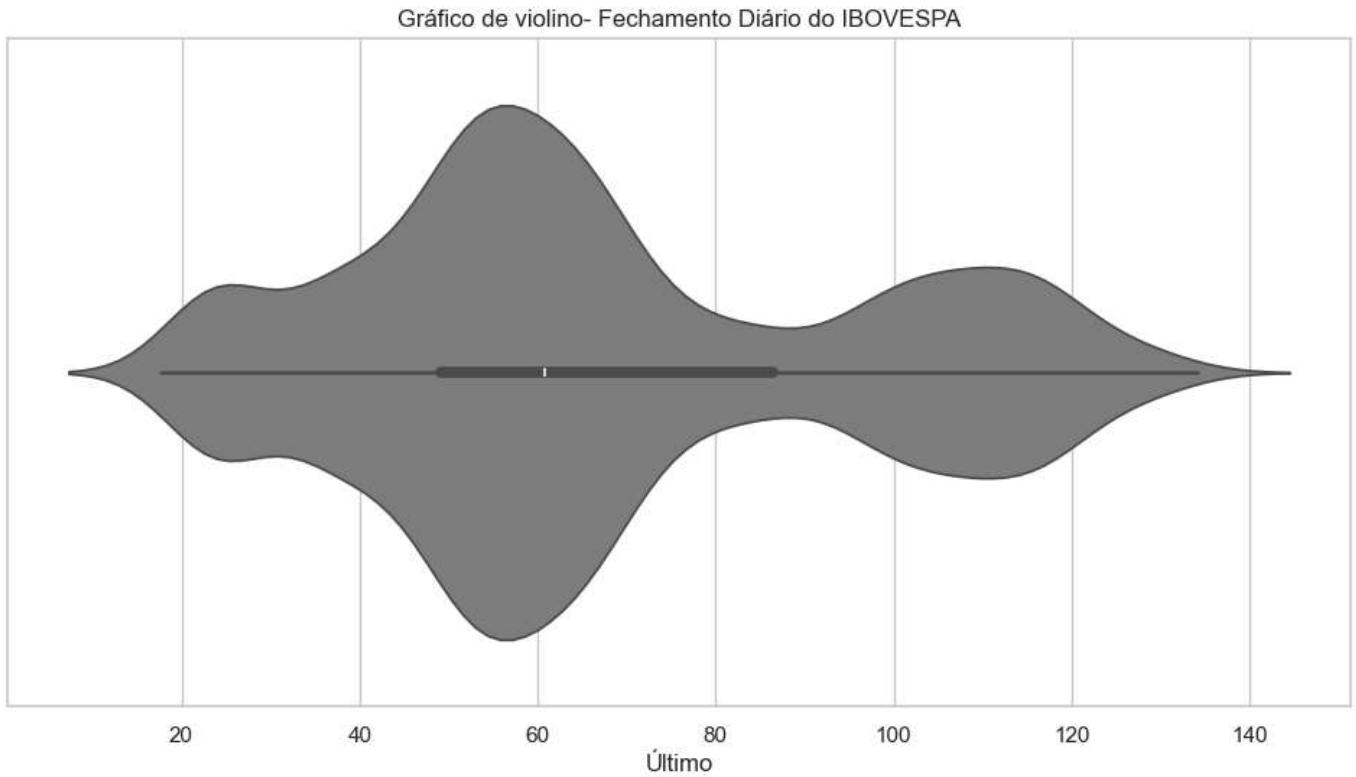
```
sns.set_theme(style="whitegrid")
fig, axes = plt.subplots(figsize=(12,6))
sns.boxplot(x=dados['Último'], data=dados)
axes.set_title('Boxplot - Fechamento Diário do IBOVESPA')
plt.show()
```



Violin Plot para observar a distribuição dos dados:

In [123...]

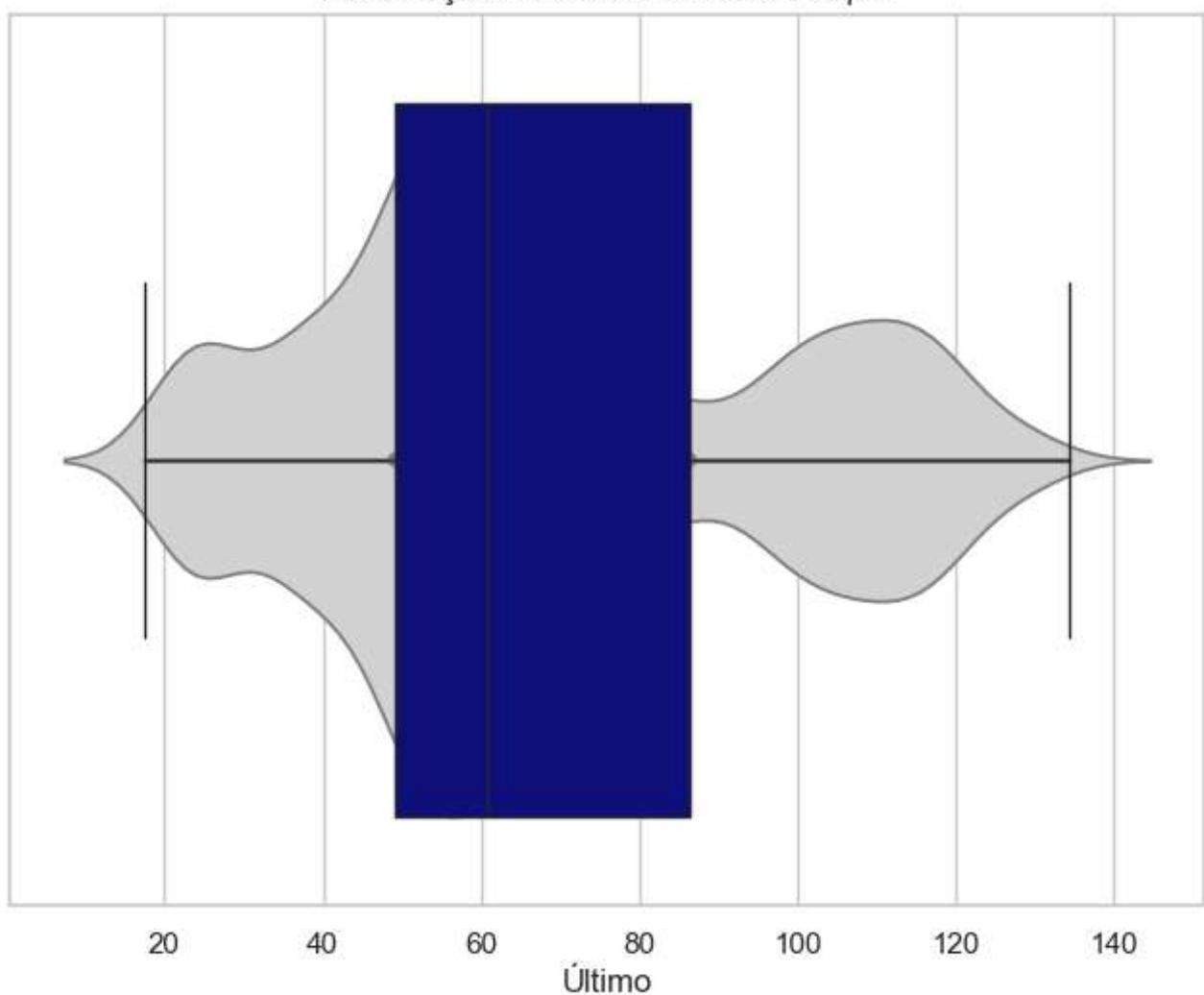
```
fig, axes = plt.subplots(figsize=(12,6))
sns.violinplot(x=dados['Último'], data=dados, color='gray')
axes.set_title('Gráfico de violino- Fechamento Diário do IBOVESPA')
plt.show()
```



In [124...]

```
fig, ax = plt.subplots(figsize=(8,6))
sns.violinplot(x=dados['Último'], data=dados, ax=ax, color='lightgray')
sns.boxplot(x=dados['Último'], data=dados, ax=ax, whis=1.5, color='darkblue')
ax.set_title('Visualização Gráfico de Violino e Boxplot')
plt.show()
```

Visualização Gráfico de Violino e Boxplot



Heatmap de Correlações:

```
In [125...]:  
numeric_dados = dados.select_dtypes(include=[float, int])  
  
# Compute the correlation matrix  
correlation_matrix = numeric_dados.corr()  
  
# Plot the heatmap  
plt.figure(figsize=(10, 6))  
corr = sns.heatmap(correlation_matrix, annot=True, cmap="Spectral_r")  
plt.show()
```



Análise Temporal

Decomposição da Série Temporal:

In [126...]

```
dados.head()
```

Out[126...]

	Último	Abertura	Máxima	Mínima	Vol.	Var%
--	--------	----------	--------	--------	------	------

Data

2004-01-02	22.445	22.233	22.450	22.208	136420000.0	0.94
2004-01-05	23.532	22.445	23.532	22.445	496710000.0	4.84
2004-01-06	23.576	23.532	23.976	23.180	472380000.0	0.19
2004-01-07	23.320	23.576	23.899	23.320	464080000.0	-1.09
2004-01-08	23.717	23.334	23.718	23.122	436740000.0	1.70

In [127...]

```
# Agrupa os dados por ano e contar o número de entradas.
dados['Data'] = pd.to_datetime(dados.index, format='%d.%m.%Y', dayfirst=True)
dados['anual_dias'] = dados['Data'].dt.year

days_per_year = dados.groupby('anual_dias').size()

print(sum(days_per_year)/20)
```

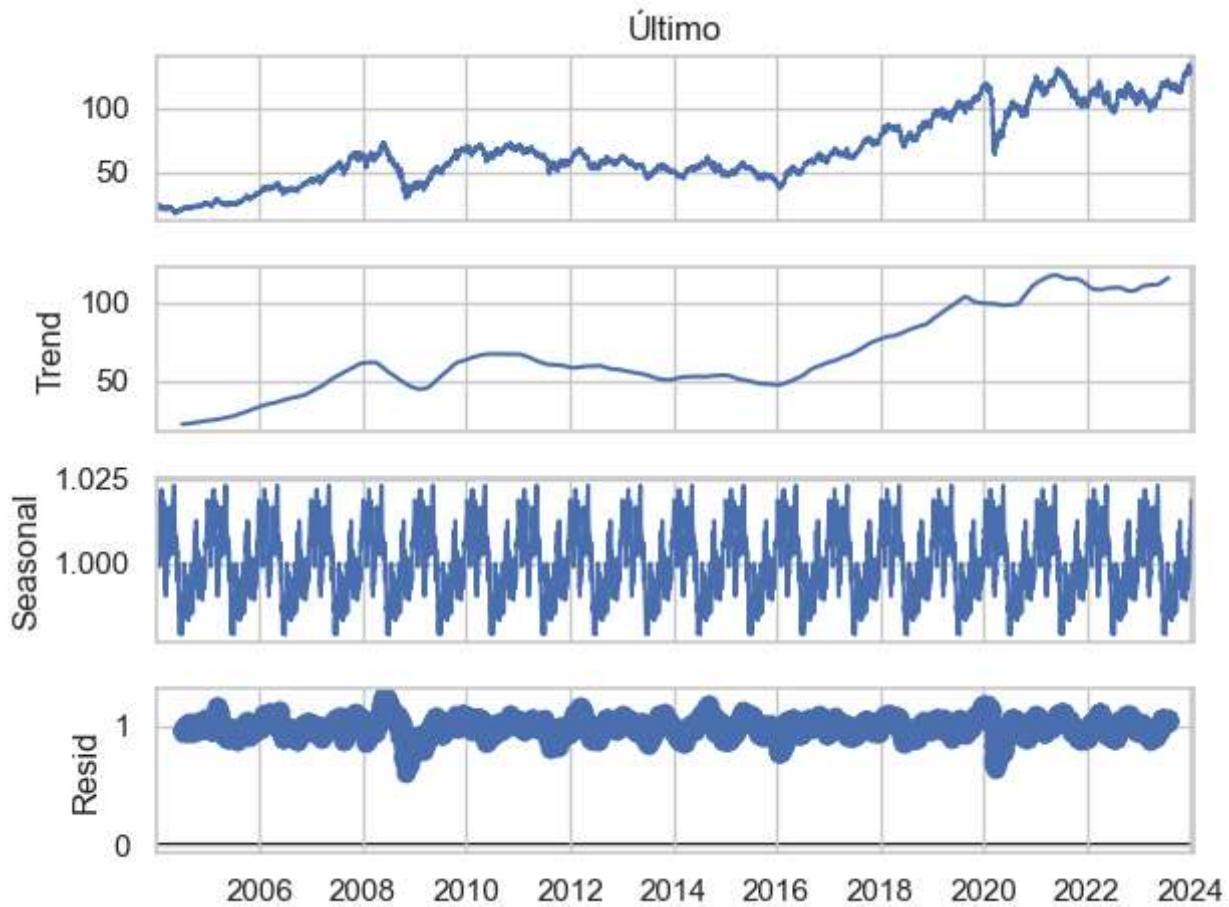
248.7

In [128...]

```
# pip install statsmodels
```

In [129...]

```
from statsmodels.tsa.seasonal import seasonal_decompose  
  
result = seasonal_decompose(dados['Último'], model='multiplicative', period=248) # Assumindo  
result.plot()  
plt.show()
```

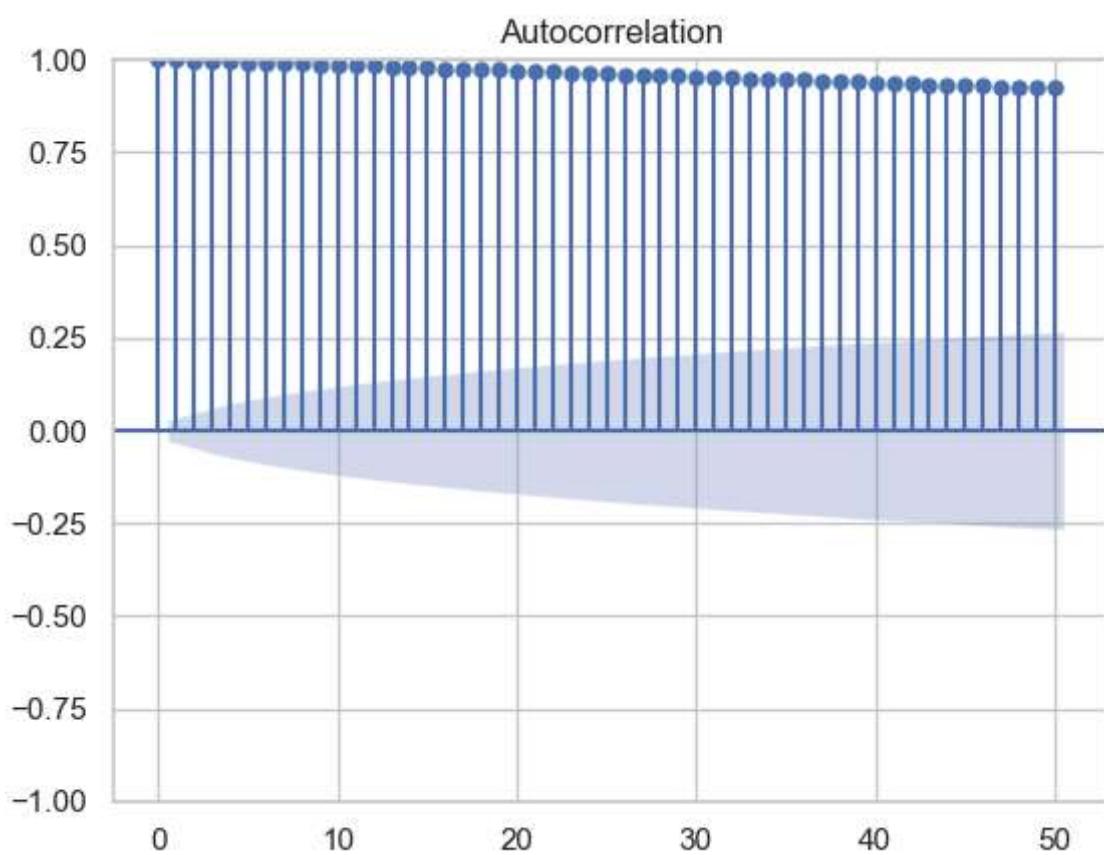


Autocorrelação e Autocorrelação Parcial

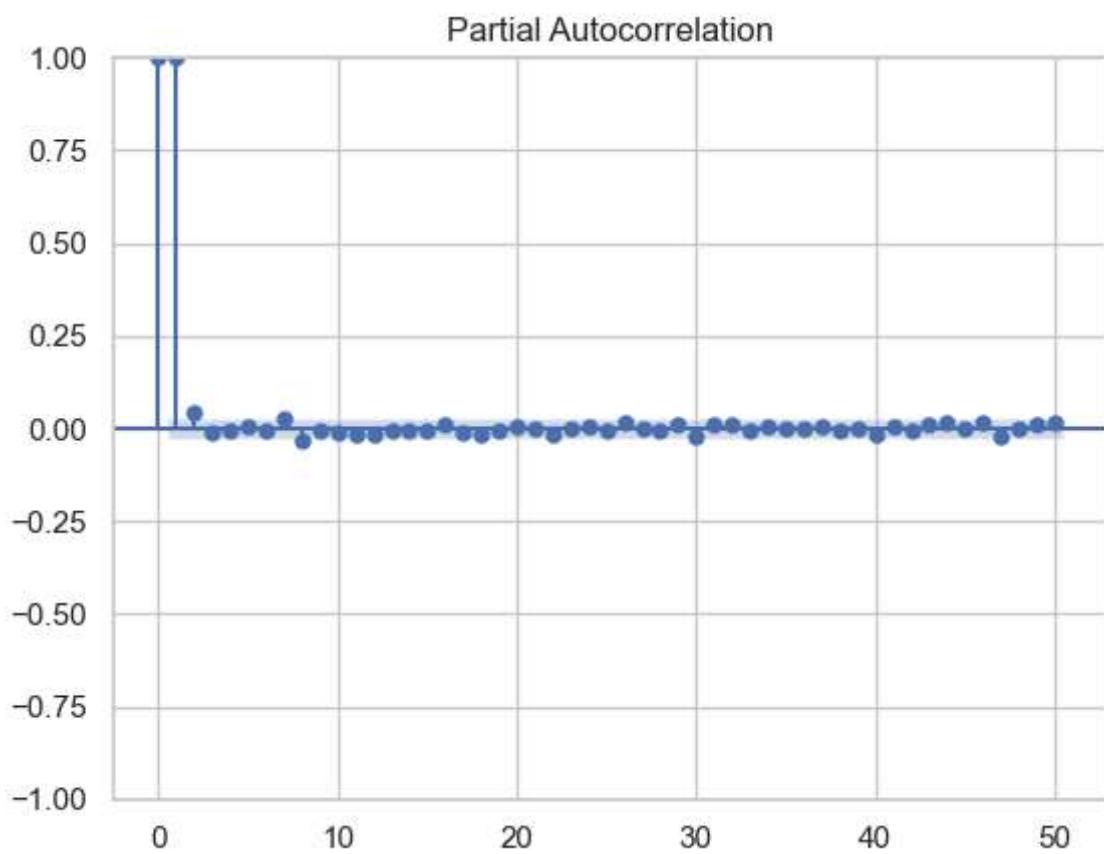
In [130...]

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf  
  
# gráficos de autocorrelação e autocorrelação parcial para entender as dependências temporais  
plt.figure(figsize=(14, 7))  
plot_acf(dados['Último'], lags=50)  
plt.show()  
  
plt.figure(figsize=(14, 7))  
plot_pacf(dados['Último'], lags=50)  
plt.show()
```

<Figure size 1400x700 with 0 Axes>



<Figure size 1400x700 with 0 Axes>



Feature Engineering (Normalização e Padronização)

Extração de Componentes de Data:

In [131...]

```
dados['Data'] = pd.to_datetime(dados.index, format='%d.%m.%Y', dayfirst=True)
dados['Dia'] = dados['Data'].dt.day
dados['Mês'] = dados['Data'].dt.month
dados['Ano'] = dados['Data'].dt.year
dados['Trimestre'] = dados['Data'].dt.quarter
```

```
dados['Dia_da_Semana'] = dados['Data'].dt.dayofweek  
dados['Fim_de_Semana'] = dados['Dia_da_Semana'].apply(lambda x: 1 if x >= 5 else 0)
```

In [132...]

```
# Criação de Lags: Lags são valores defasados de uma série temporal, ou seja, os valores anteriores  
# Na modelagem de séries temporais, os Lags ajudam a prever o valor atual com base nos valores anteriores  
  
dados['Lag_1'] = dados['Último'].shift(1) # shift() desloca a série temporal em n períodos,  
dados['Lag_5'] = dados['Último'].shift(5)  
dados['Lag_10'] = dados['Último'].shift(10)  
  
# Criação de médias móveis  
  
dados['MA_5'] = dados['Último'].rolling(window=5).mean()  
dados['MA_10'] = dados['Último'].rolling(window=10).mean()  
dados['MA_20'] = dados['Último'].rolling(window=20).mean()
```

In [133...]

```
# Média Móvel Exponencial  
dados['EMA_10'] = dados['Último'].ewm(span=10, adjust=False).mean()  
dados['EMA_20'] = dados['Último'].ewm(span=20, adjust=False).mean()
```

Índice de Força Relativa (RSI):

- Análise através da RSI:
- O RSI frequentemente acima de 70 sugere que o ativo está em condição de sobrecompra. Isso pode indicar uma forte tendência de alta e pressão compradora significativa. Apesar da alta constante, é importante estar atento a possíveis correções ou consolidações, pois níveis elevados de RSI podem levar a ajustes de preço.
- O RSI frequentemente alto pode refletir um forte momentum positivo no ativo, indicando uma tendência robusta e uma força relativa significativa. Sugerindo que o ativo tem um desempenho superior em relação ao mercado ou aos seus pares.
- O ativo tem pouca oscilação que atinge níveis abaixo de 30, indicando menos situações de sobrevenção e sugerindo resiliência durante as quedas. Sendo que tende a se recuperar rapidamente de quedas, mostrando uma menor pressão de venda.

In [134...]

```
def calcular_RSI(data, window):  
    delta = data.diff(1) # diff(1): Calcula a diferença entre o preço atual e o preço anterior  
    gain = delta.where(delta > 0, 0) # Mantém as diferenças positivas e define as negativas como zero  
    loss = -delta.where(delta < 0, 0) # Mantém as diferenças negativas (como valores positivos)  
  
    avg_gain = gain.rolling(window=window, min_periods=1).mean() # Calcula a média móvel dos ganhos  
    avg_loss = loss.rolling(window=window, min_periods=1).mean()  
  
    rs = avg_gain / avg_loss # Relativa Força: ratio de ganho médio para perda média  
    rsi = 100 - (100 / (1 + rs))  
    return rsi  
  
dados['RSI_14'] = calcular_RSI(dados['Último'], 14)
```

In [135...]

```
dados.head(3)
```

Out[135...]

	Último	Abertura	Máxima	Mínima	Vol.	Var%	Data	anual_dias	Dia	Mês	...	Fim_
Data												
2004-01-02	22.445	22.233	22.450	22.208	136420000.0	0.94	2004-01-02		2004	2	1	...
2004-01-05	23.532	22.445	23.532	22.445	496710000.0	4.84	2004-01-05		2004	5	1	...
2004-01-06	23.576	23.532	23.976	23.180	472380000.0	0.19	2004-01-06		2004	6	1	...

3 rows × 23 columns

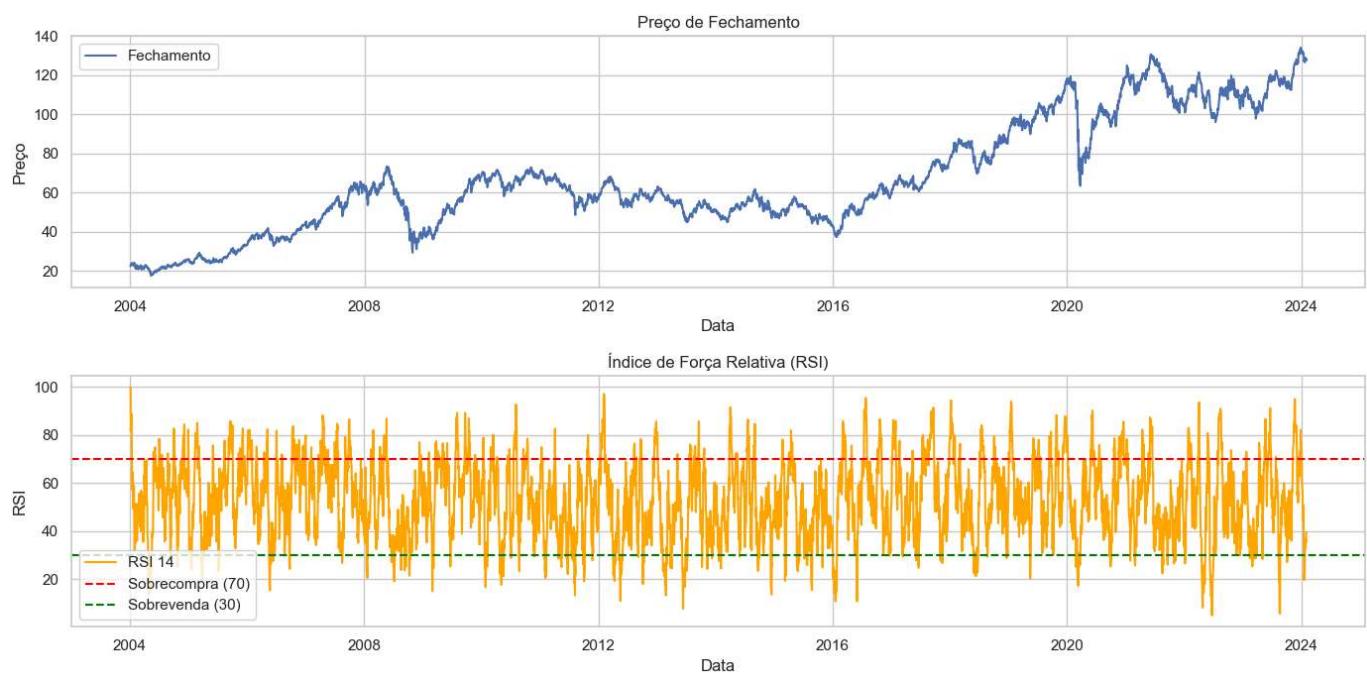
In [136...]

```
plt.figure(figsize=(14, 7))

# Preço de fechamento
plt.subplot(2, 1, 1)
plt.plot(dados['Último'], label='Fechamento')
plt.title('Preço de Fechamento')
plt.xlabel('Data')
plt.ylabel('Preço')
plt.legend()

# RSI
plt.subplot(2, 1, 2)
plt.plot(dados['RSI_14'], label='RSI 14', color='orange')
plt.axhline(y=70, color='red', linestyle='--', label='Sobrecompra (70)')
plt.axhline(y=30, color='green', linestyle='--', label='Sobrevenda (30)')
plt.title('Índice de Força Relativa (RSI)')
plt.xlabel('Data')
plt.ylabel('RSI')
plt.legend()

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



Obs.:

- Se o RSI está acima de 70, pode indicar que o ativo está sobrecomprado, sugerindo que o preço pode estar alto demais e pode ocorrer uma correção ou queda em breve.

- Se o RSI está abaixo de 30, pode indicar que o ativo está sobre vendido, sugerindo que o preço pode estar baixo demais e pode ocorrer uma recuperação ou aumento em breve.

Bandas de Bollinger:

As Bandas de Bollinger são um indicador de volatilidade que cria uma banda superior e uma inferior em torno de uma média móvel simples. Consistem em uma média móvel central (geralmente de 20 períodos) e duas bandas que são traçadas a um desvio padrão acima e abaixo da média móvel. Estas bandas se ajustam dinamicamente com a volatilidade do ativo.

https://algobulls.github.io/pyalgotrading/strategies/bollinger_bands/

In [137...]

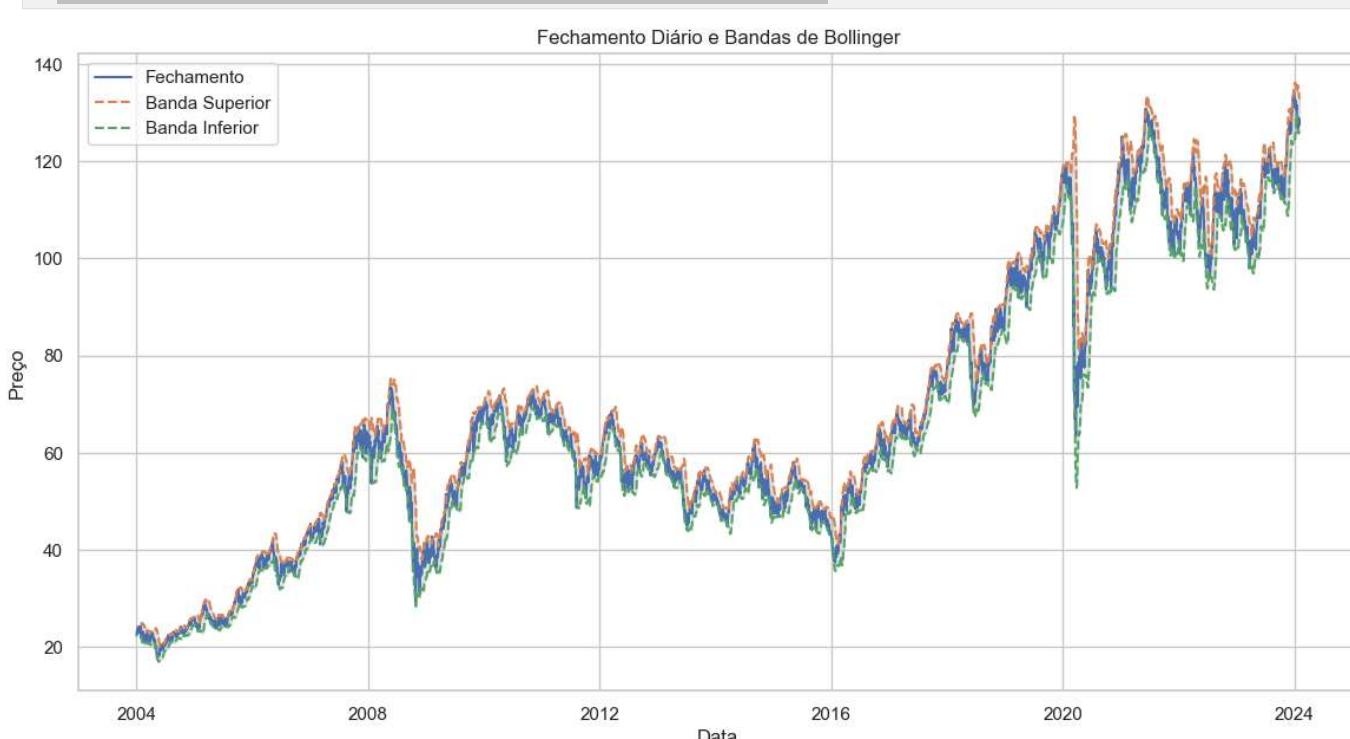
```
def calcular_bollinger_bands(data, window):
    MA = data.rolling(window=window).mean() # Calcula a média móvel simples.
    std = data.rolling(window=window).std() # Calcula o desvio padrão móvel
    upper_band = MA + (std * 2)
    lower_band = MA - (std * 2)
    return upper_band, lower_band

dados['Upper_BB'], dados['Lower_BB'] = calcular_bollinger_bands(dados['Último'], 20)
```

In [138...]

```
plt.figure(figsize=(14, 7))
plt.plot(dados['Último'], label='Fechamento')
plt.plot(dados['Upper_BB'], label='Banda Superior', linestyle='--')
plt.plot(dados['Lower_BB'], label='Banda Inferior', linestyle='--')
plt.fill_between(dados.index, dados['Upper_BB'], dados['Lower_BB'], color='gray', alpha=0.2)
plt.title('Fechamento Diário e Bandas de Bollinger')
plt.xlabel('Data')
plt.ylabel('Preço')
plt.legend()
plt.show()

# As Bandas de Bollinger fornecem uma faixa de preço dentro da qual o ativo geralmente oscila
# Quando o preço se aproxima ou ultrapassa as bandas, pode sinalizar uma possível reversão ou
```



- Queda Abrupta em 2008, a Ibovespa enfrentou uma queda abrupta. Ao analisar as Bandas de Bollinger durante este período, observamos que o preço se manteve próximo à banda inferior. Este comportamento sugere uma tendência de baixa contínua, como indicado pela proximidade com a banda inferior. A banda inferior atuou como uma referência para a pressão de venda intensa e a falta de suporte para o preço. A largura das bandas também pode ter se expandido, refletindo a alta volatilidade do mercado durante essa queda.
- Após a queda de 2008, a Ibovespa experimentou uma recuperação significativa em 2009. Durante este período, o preço se manteve próximo à banda superior por um período prolongado. A proximidade com a banda superior durante a recuperação indica uma forte tendência de alta e a continuidade dessa tendência. O ativo demonstrou uma pressão compradora significativa e uma valorização acentuada. A banda superior atuou como um nível de resistência durante a alta, e sua manutenção sugere a robustez da tendência de alta.
- Queda de 2019, a Ibovespa passou por uma queda semelhante à observada em 2008. Neste período, o comportamento das Bandas de Bollinger apresentou uma dinâmica similar. A queda de 2019 também foi caracterizada por uma aproximação do preço à banda inferior, indicando uma pressão de venda intensa e uma tendência de baixa. A largura das bandas pode ter se expandido, refletindo a alta volatilidade do mercado durante essa queda. Esta similaridade na dinâmica das bandas entre 2008 e 2019 sugere um padrão de comportamento recorrente em períodos de crise ou correção do mercado.
- Conclusão A análise das Bandas de Bollinger revela padrões importantes no comportamento da Ibovespa. Em 2008 e 2019, a proximidade com a banda inferior destacou tendências de baixa, enquanto a recuperação de 2009 e 2020 evidenciou uma forte tendência de alta com o preço próximo à banda superior.

Volume e Volatilidade:

O volume de negociações e a volatilidade são características importantes para entender a dinâmica do mercado. Eles ajudam a medir a atividade do mercado e a instabilidade dos preços, respectivamente.

In [139...]

```
dados['Var%'] = dados['Último'].pct_change() * 100 # Calcula a mudança percentual entre o preço de fechamento e o preço anterior
```

A variação percentual diária mostra a mudança percentual no preço de fechamento de um dia para o dia anterior

In [140...]

```
dados['Volatilidade_10'] = dados['Var%'].rolling(window=10).std() # Calcula o desvio padrão da variação percentual diária ao longo de 10 dias
```

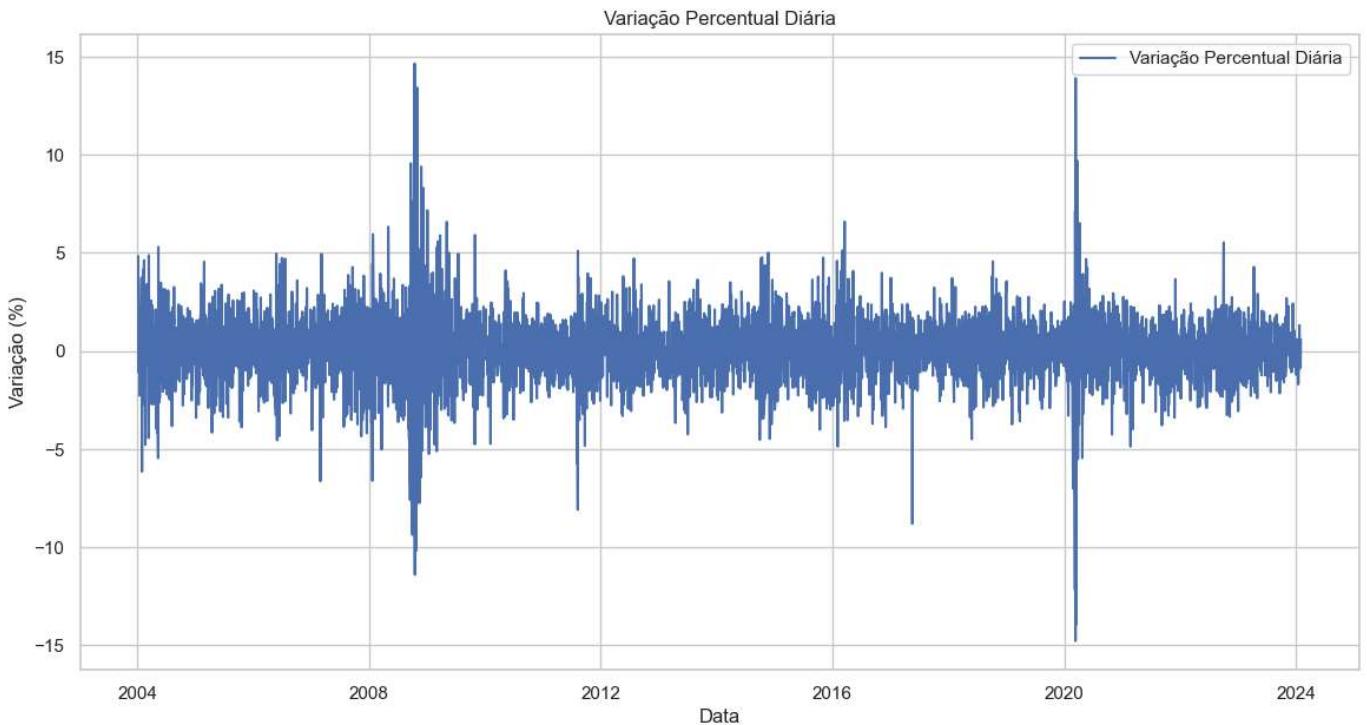
dados['Volatilidade_20'] = dados['Var%'].rolling(window=20).std()

A volatilidade mede a variação do preço de um ativo ao longo do tempo. Ela indica o quanto os preços flutuam ao longo do tempo.

Baixos valores de volatilidade indicam preços mais estáveis, enquanto altos valores indicam preços mais voláteis.

In [141...]

```
plt.figure(figsize=(14, 7))
plt.plot(dados['Var%'], label='Variação Percentual Diária')
plt.title('Variação Percentual Diária')
plt.xlabel('Data')
plt.ylabel('Variação (%)')
plt.legend()
plt.show()
```



Conclusão

O gráfico da Ibovespa demonstra padrões claros de alta volatilidade e oscilações constantes, com quedas abruptas em 2008 e 2019 seguidas de altas percentuais significativas em 2009 e 2020. A análise do volume e da volatilidade durante esses períodos revela a intensidade das reações do mercado e a resiliência diante de crises econômicas. Esses insights são cruciais para entender a dinâmica do mercado e para a formulação de estratégias de negociação que considerem a volatilidade e os padrões de volume em resposta a eventos significativos.

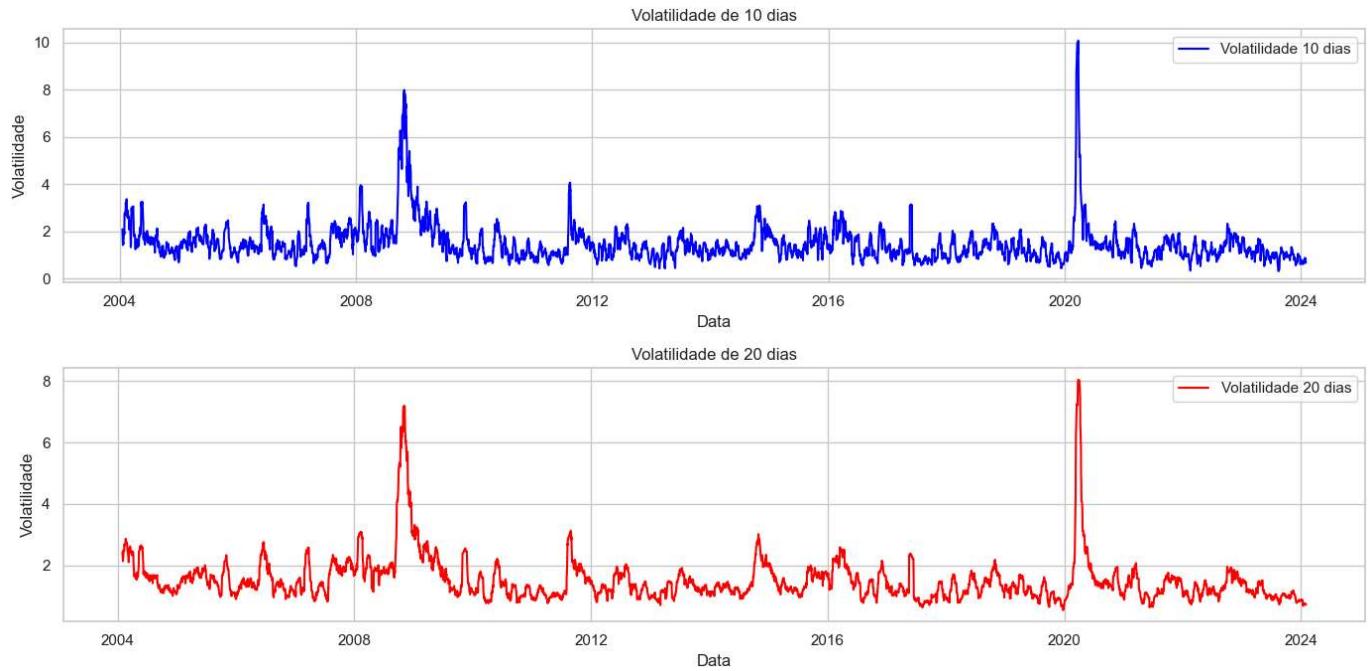
In [142...]

```
plt.figure(figsize=(14, 7))

# Volatilidade de 10 dias
plt.subplot(2, 1, 1)
plt.plot(dados['Volatilidade_10'], label='Volatilidade 10 dias', color='blue')
plt.title('Volatilidade de 10 dias')
plt.xlabel('Data')
plt.ylabel('Volatilidade')
plt.legend()

# Volatilidade de 20 dias
plt.subplot(2, 1, 2)
plt.plot(dados['Volatilidade_20'], label='Volatilidade 20 dias', color='red')
plt.title('Volatilidade de 20 dias')
plt.xlabel('Data')
plt.ylabel('Volatilidade')
plt.legend()

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



Conclusão

Os gráficos mostram padrões semelhantes (mesmos picos e vales) apesar de terem níveis diferentes de volatilidade, isso pode indicar que o comportamento do mercado é consistente ao longo dos diferentes períodos analisados. A semelhança sugere que tanto a volatilidade de curto prazo (10 dias) quanto a de médio prazo (20 dias) estão respondendo de maneira similar a eventos de mercado.

In [143...]

```
dados = dados.dropna()
```

In [144...]

```
print(dados.head())
```

	Último	Abertura	Máxima	Mínima	Vol.	Var%	\		
Data									
2004-01-30	21.851	22.384	22.615	21.649	430570000.0	-2.389887			
2004-02-02	21.787	21.842	21.867	21.336	327360000.0	-0.292893			
2004-02-03	22.281	21.794	22.287	21.794	302440000.0	2.267407			
2004-02-04	21.685	22.289	22.718	21.555	464940000.0	-2.674925			
2004-02-05	21.092	21.685	21.909	21.077	414970000.0	-2.734609			
	Data	anual_dias	Dia	Mês	...	MA_5	MA_10	MA_20	\
Data					...				
2004-01-30	2004-01-30		2004	30	1	23.3330	23.3471	23.46130	
2004-02-02	2004-02-02		2004	2	2	22.8204	23.1873	23.37405	
2004-02-03	2004-02-03		2004	3	2	22.4314	23.0475	23.30930	
2004-02-04	2004-02-04		2004	4	2	21.9980	22.8858	23.22755	
2004-02-05	2004-02-05		2004	5	2	21.7392	22.6981	23.09630	
	EMA_10	EMA_20	RSI_14	Upper_BB	Lower_BB				\
Data									
2004-01-30	23.161031	23.209975	31.874810	24.682741	22.239859				
2004-02-02	22.911207	23.074454	33.044574	24.805475	21.942625				
2004-02-03	22.796624	22.998887	41.129800	24.817367	21.801233				
2004-02-04	22.594511	22.873755	40.117629	24.901332	21.553768				
2004-02-05	22.321327	22.704064	34.965748	25.003838	21.188762				
	Volatilidade_10	Volatilidade_20							
Data									
2004-01-30		2.771464		2.435740					
2004-02-02		2.721646		2.139244					
2004-02-03		2.818224		2.216552					
2004-02-04		2.881918		2.275469					
2004-02-05		2.948485		2.282361					

[5 rows x 27 columns]

Treinamento dos Modelos:

Regressão Linear

```
In [145...]: features = ['Abertura', 'Máxima', 'Mínima', 'Vol.', 'Dia', 'Mês', 'Ano', 'Trimestre', 'Dia_da_semana', 'Fim_de_Semana', 'Lag_1', 'Lag_5', 'Lag_10', 'MA_5', 'MA_10', 'MA_20', 'EMA_10', 'RSI_14', 'Upper_BB', 'Lower_BB', 'Var%', 'Volatilidade_10', 'Volatilidade_20']
target = 'Último'

X = dados[features]
y = dados[target]
```

```
In [146...]: from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

```
In [147...]: from sklearn.linear_model import LinearRegression
```

```
In [148...]: modelo = LinearRegression()
modelo.fit(X_train, y_train)
```

Out[148...]
LinearRegression()
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [149...]
predicoes = modelo.predict(X_test)

In [150...]
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, predicoes)
print(f"Erro Quadrático Médio (MSE): {mse}")

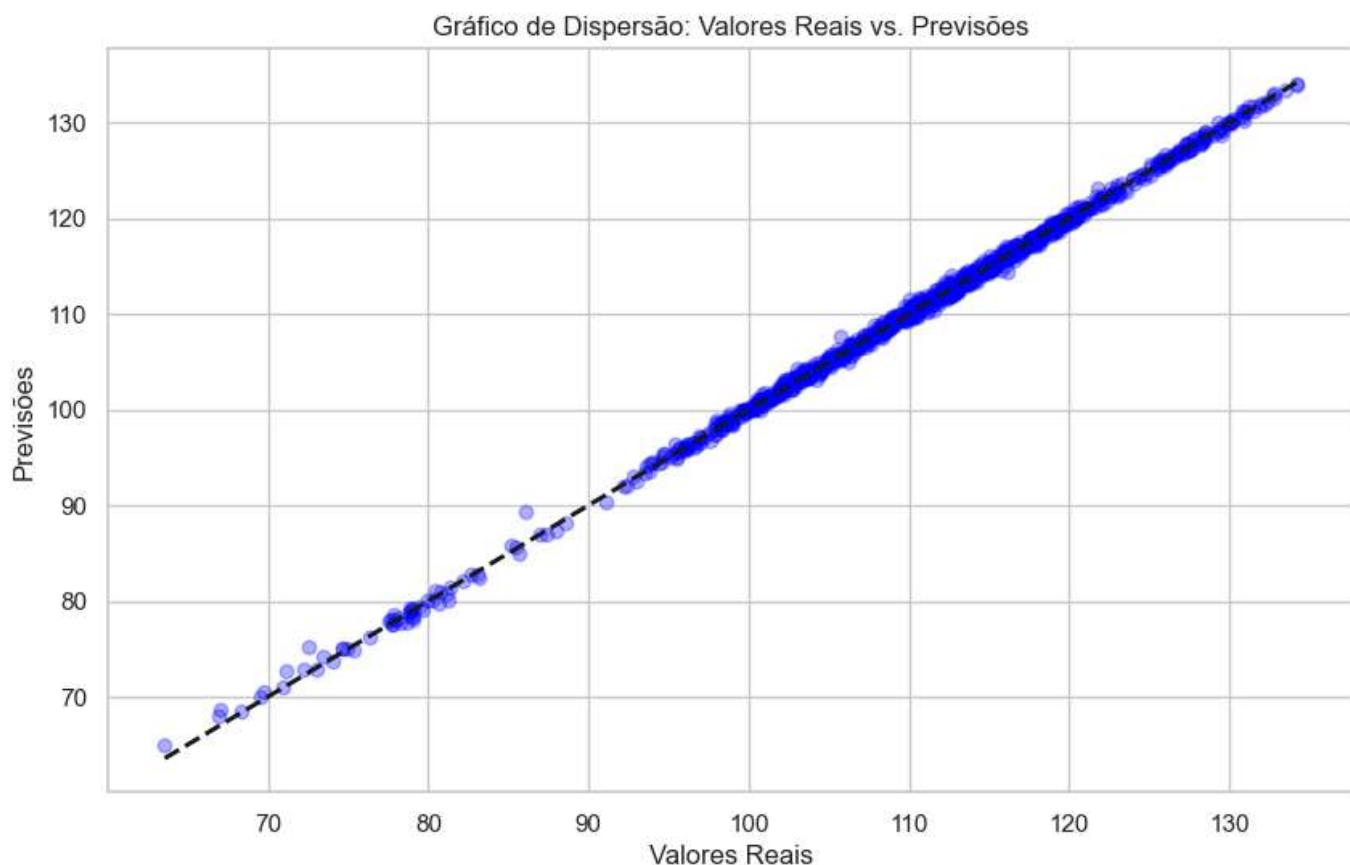
Erro Quadrático Médio (MSE): 0.20298814303678625

In [151...]
from sklearn.metrics import r2_score

r2 = r2_score(y_test, predicoes)
print(f"Coeficiente de Determinação (R²): {r2}")

Coeficiente de Determinação (R²): 0.9985532311726717

In [152...]
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predicoes, color='blue', alpha=0.3)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Valores Reais')
plt.ylabel('Previsões')
plt.title('Gráfico de Dispersão: Valores Reais vs. Previsões')
plt.show()



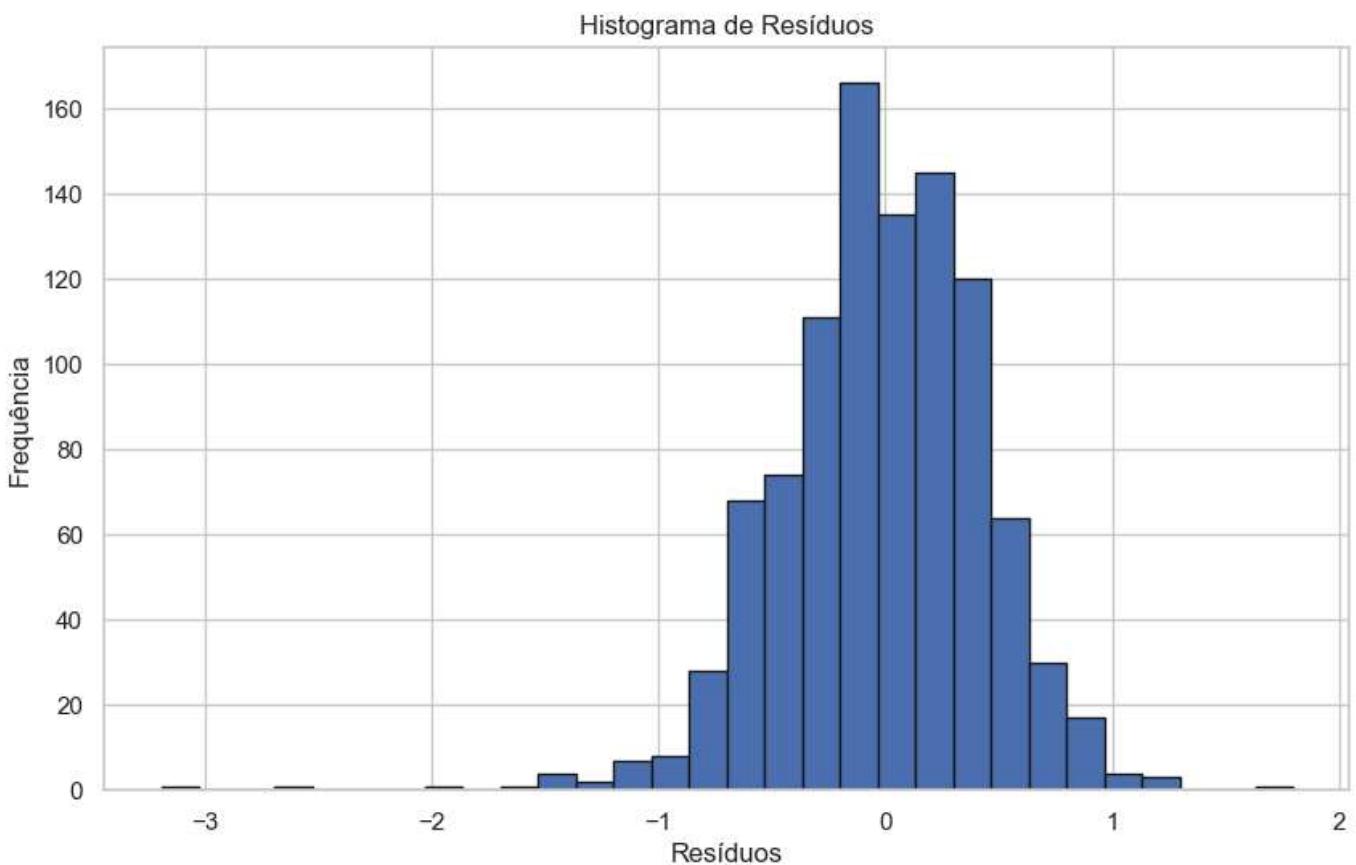
In [153...]
residuos = y_test - predicoes

plt.figure(figsize=(10, 6))
plt.hist(residuos, bins=30, edgecolor='k')

```

plt.xlabel('Resíduos')
plt.ylabel('Frequência')
plt.title('Histograma de Resíduos')
plt.show()

```



Conclusão Regressão Linear:

O modelo de regressão utilizado exibe um desempenho impressionante, com um Erro Quadrático Médio (MSE) de 0.2029881428799871, indicando que os erros das previsões são relativamente pequenos em média. Além disso, o Coeficiente de Determinação (R^2) alcança 0.9985532311737892, o que é excepcionalmente alto. Isso sugere que o modelo explica aproximadamente 99.85% da variação nos dados de fechamento da IBOVESPA, indicando um ajuste muito próximo aos dados observados.

A distribuição dos resíduos está aproximadamente normal e centrada em torno de zero, o que é um bom sinal de que o modelo não apresenta viés sistemático e que a variabilidade não explicada é aleatória. Esse comportamento é consistente com as suposições da regressão e fortalece a confiabilidade das previsões do modelo.

No entanto, é importante notar que um valor de R^2 tão alto pode levantar questões sobre a robustez e a validade do modelo. Um R^2 próximo de 100% pode, às vezes, ser um sinal de sobreajuste (overfitting), especialmente se o modelo está ajustando perfeitamente aos dados de treinamento, mas pode não generalizar bem para dados não vistos.

Portanto, é aconselhável explorar e comparar outros modelos para validar e possivelmente melhorar a previsão. Modelos mais complexos, como regressões polinomiais, árvores de decisão, random forests, e redes neurais, podem oferecer diferentes perspectivas e potencialmente revelar nuances que o modelo de regressão linear pode ter deixado de capturar. Além disso, a aplicação de técnicas de validação cruzada e a análise de modelos de séries temporais especializadas podem ajudar a garantir que o modelo final seja robusto e capaz de generalizar para novos dados.

Em resumo, enquanto o modelo atual demonstra um ajuste muito bom aos dados históricos, a exploração de outras abordagens é essencial para confirmar a precisão das previsões e garantir a

Referências

- Avaliar modelo de Regressão Linear <https://medium.com/turing-talks/como-avaliar-seu-modelo-de-regress%C3%A3o-c2c8d73dab96>

Regressões Polinominais

Preparando os Dados

In [154...]

```
# Importando as bibliotecas necessárias

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [155...]

```
dados.columns
```

Out[155...]

```
Index(['Último', 'Abertura', 'Máxima', 'Mínima', 'Vol.', 'Var%', 'Data',
       'anual_dias', 'Dia', 'Mês', 'Ano', 'Trimestre', 'Dia_da_Semana',
       'Fim_de_Semana', 'Lag_1', 'Lag_5', 'Lag_10', 'MA_5', 'MA_10', 'MA_20',
       'EMA_10', 'EMA_20', 'RSI_14', 'Upper_BB', 'Lower_BB', 'Volatilidade_10',
       'Volatilidade_20'],
      dtype='object')
```

In [156...]

```
# Separando as variáveis características e variável target

features = ['Abertura', 'Máxima', 'Mínima', 'Vol.', 'Dia', 'Mês', 'Ano', 'Trimestre', 'Dia_da_Semana',
            'Fim_de_Semana', 'Lag_1', 'Lag_5', 'Lag_10', 'MA_5', 'MA_10', 'MA_20', 'EMA_10',
            'RSI_14', 'Var%']
target = 'Último'
```

In [157...]

```
X = dados[features]
y = dados[target]
```

In [158...]

```
# Dividindo os dados em Bases de Treino e Teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [159...]

```
# Imputando dados para preencher valores faltantes

from sklearn.impute import SimpleImputer
```

In [160...]

```
imputer = SimpleImputer(strategy='mean') # Aqui também poderia ser utilizado mediana, maior
```

In [161...]

```
# Ajustando o imputador aos dados de treino e transformá-los
X_train_imputed = imputer.fit_transform(X_train)
```

In [162...]

```
# Transformando os dados de teste
X_test_imputed = imputer.transform(X_test)
```

In [163...]

```
# Transformando as características para incluir termos polinomiais
degree = 2 # Grau do polinômio escolhido = 2
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train_imputed)
X_test_poly = poly.transform(X_test_imputed)
```

In [164...]

```
# Ajustando o modelo de regressão polinomial aos dados transformados
model = LinearRegression()
model.fit(X_train_poly, y_train)
```

Out[164...]

LinearRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [165...]

```
# Fazendo as previsões:
y_train_pred = model.predict(X_train_poly)
y_test_pred = model.predict(X_test_poly)
```

Avaliando o Modelo

In [166...]

```
# Avaliação do modelo
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)
```

In [167...]

```
# Imprimindo o Resultado do Modelo
print(f'Train MSE: {train_mse}')
print(f'Train R2: {train_r2}')
print(f'Test MSE: {test_mse}')
print(f'Test R2: {test_r2}')
```

Train MSE: 0.00847523528933782
Train R²: 0.9999891463897012
Test MSE: 0.01089218688356105
Test R²: 0.9999871121482707

Visualização do Resultado

In [168...]

```
plt.figure(figsize=(10, 5))

# Plot dos dados de treino
plt.subplot(1, 2, 1)
plt.scatter(y_train, y_train_pred, alpha=0.3)
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--', lw=2)
plt.xlabel('Valores Reais')
plt.ylabel('Valores Preditos')
plt.title('Conjunto de Treino')

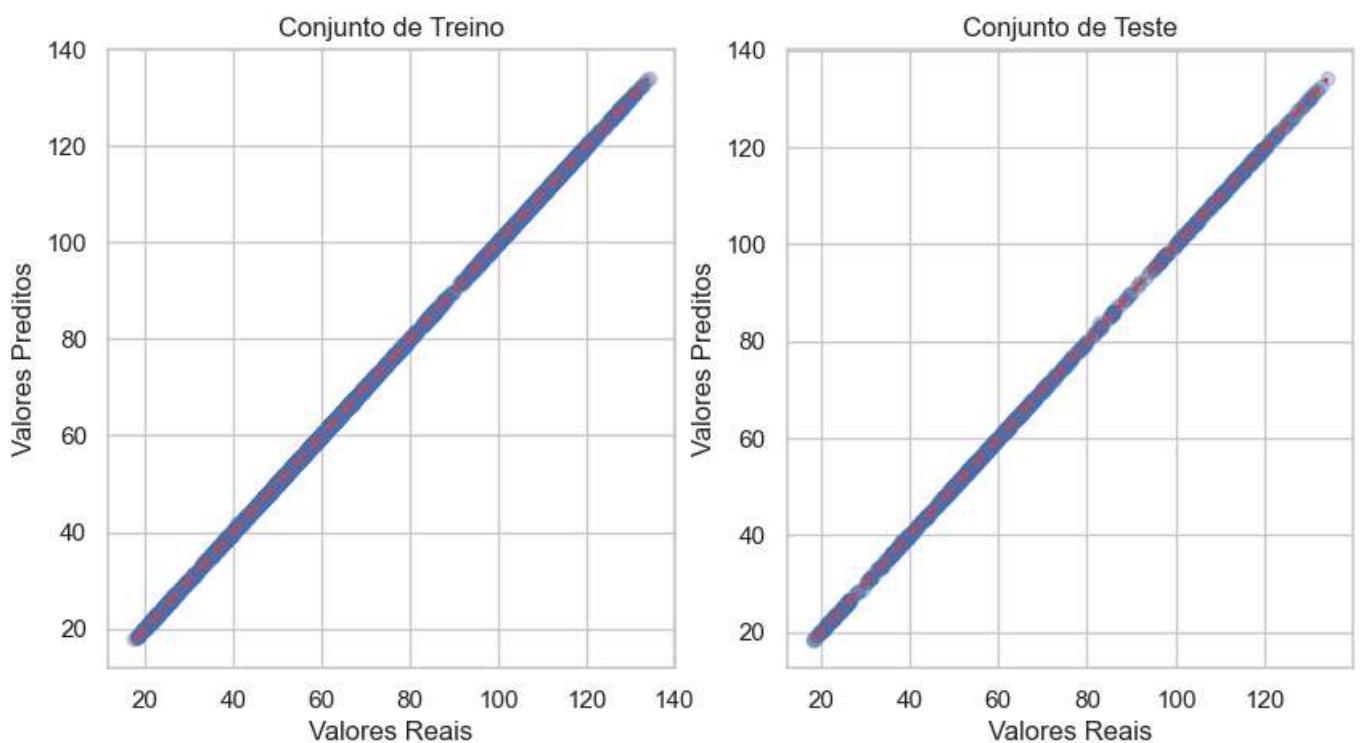
# Plot dos dados de teste
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_test_pred, alpha=0.3)
```

```

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Valores Reais')
plt.ylabel('Valores Preditos')
plt.title('Conjunto de Teste')

plt.show()

```



Conclusão Regressões Polinominais:

Um coeficiente de determinação R² de 99% indica que o modelo está explicando 99% da variância nos dados de resposta, o que pode ser um sinal de que o modelo está ajustando muito bem aos dados de treinamento. No entanto, um R² tão alto pode também ser um indicativo de que o modelo está superajustado / viciado (overfitting), especialmente se o conjunto de dados for pequeno ou se houver **muitas** características (features) em comparação ao número de observações.

Avaliando a Confiabilidade do Modelo Regressões Polinominais:

1 - Comparando R² de Treinamento e Teste

- Se os valores de R² para o conjunto de treinamento e o conjunto de teste forem muito próximos, isso sugere que o modelo generaliza bem para dados não vistos.
- Se houver uma grande diferença entre os dois, isso indica que o modelo pode estar superajustado (**Overfitting**).

In [169...]

```

# Comparando os valores de R2 para ambos os conjuntos
print(f'Train R2: {train_r2}')
print(f'Test R2: {test_r2}')

```

Train R²: 0.9999891463897012
Test R²: 0.9999871121482707

Conclusão:

- Se a diferença for pequena isso sugere que o modelo não está super ajustado
- Se a diferença for grande isso sugere que o modelo pode estar super ajustado = **Overfitting**

Concluímos que este modelo está performando bem, pois os valores de R2 para Treino e Teste estão muito próximos.

2 - Validação Cruzada

- A validação cruzada ajuda a verificar a robustez do modelo ao testar seu desempenho em diferentes subconjuntos dos dados.

In [170...]

```
# Utilizando a validação cruzada com o cross_val_score do sklearn
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X_train_poly, y_train, cv=5, scoring='r2')
print(f'Cross-Validation R² scores: {scores}')
print(f'Mean R²: {scores.mean()}')
```

```
Cross-Validation R² scores: [0.99997391 0.99999753 0.99998762 0.99998704 0.99998908]
Mean R²: 0.9999870356173973
```

Conclusão:

- A média dos scores R2 da validação cruzada e a sua variabilidade são importantes. Scores consistentemente altos sugerem que o modelo generaliza bem.
- Scores variáveis ou significativamente mais baixos que o R2 de treinamento indicam que o modelo pode não ser robusto.

Conclui-se que devido ao valor da Média de R2 ser consistentemente altos, este modelo nos mostra que a generalização performa bem.

3- Verificação de Multicolinearidade

A multicolinearidade ocorre quando duas ou mais variáveis independentes em um modelo de regressão estão altamente correlacionadas entre si. Isso significa que uma variável pode ser predita linearmente a partir das outras com um grau significativo de precisão.

Detecção da Multicolinearidade

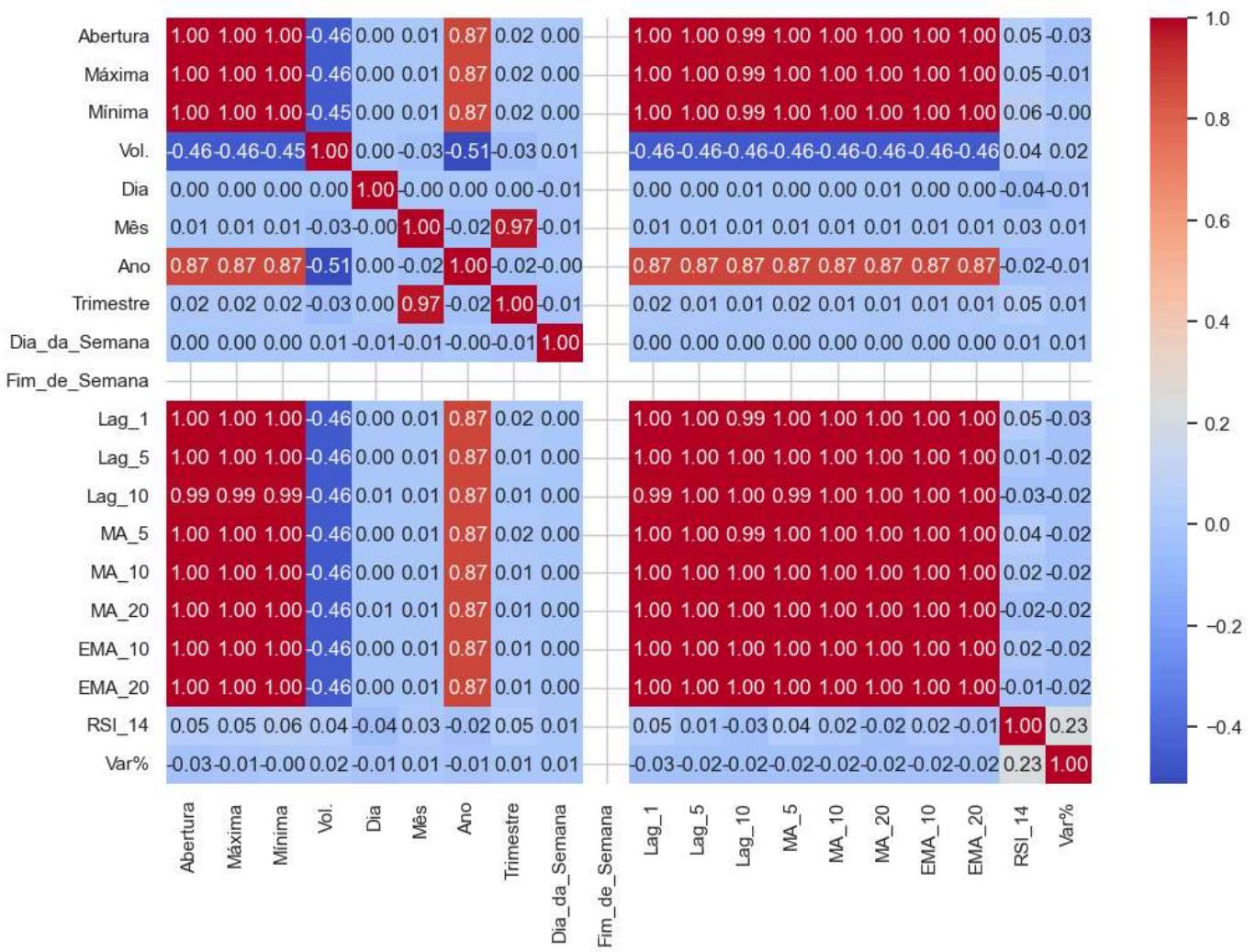
Verificando as correlações entre as variáveis independentes.

Correlações altas (próximas de 1 ou -1) indicam multicolinearidade

In [171...]

```
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrix = X.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.show()
```



Conclusão:

- Se houver fortes correlações entre as variáveis polinomiais, isso pode indicar multicolinearidade.

Ao verificar o gráfico acima, identificamos que muitas variáveis estão fortemente relacionadas, nos levando a crer que o Modelo permanece confiável

Conclusão Final referente ao Modelo de Regressões Polinominais

Após toda a análise da validação de confiabilidade do Modelo denominado Regressões Polinominais, os resultados se mostraram favoráveis, nos levando a acreditar que este tipo de Modelo está amplamente apto para considerarmos em nossas análises de previsões.

Referências

- Edisciplinas USP* https://edisciplinas.usp.br/pluginfile.php/8083873/mod_resource/content/1/Regressao%20Polinomial.pdf
- Machine Learning for Beginners / Microsoft* <https://learn.microsoft.com/pt-br/shows/machine-learning-for-beginners/improving-pumpkin-price-predictions-with-linear-and-polynomial-regression-using-scikit-learn-machine-learning-for-beginners#time=03m23s>

ARVORE DE DECISÃO (DecisionTreeRegressor)

Modelo de Arvore de Decisão: Um modelo de DecisionTreeRegressor é um modelo de árvore de decisão utilizado para resolver problemas de regressão. Esse tipo de técnica cria uma estrutura em forma de árvore para mapear relações não lineares entre as variáveis preditoras e a variável alvo.

Fonte: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>

In [172...]

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz

#Instanciando meu modelo
model_dtr = DecisionTreeRegressor(random_state=101, max_depth=10)

# Criando o modelo de DecisionTreeRegressor
model_dtr.fit(X_train, y_train)

#previsão usando um modelo de árvore de decisão
y_pred_model_dtr = model_dtr.predict(X_test)

#Avaliando o Modelo
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Mean Absolute Error (MAE) é a média do valor absoluto dos erros.
MAE = mean_absolute_error(y_test, y_pred_model_dtr)
print(f"Mean Absolute Error: ",MAE)

# Erro Quadrático Médio (MSE) é a média dos erros quadráticos
MSE = mean_squared_error(y_test, y_pred_model_dtr)
print(f"Mean Squared Error: ",MSE)

# (R-quadrado)
r2 = r2_score(y_test, y_pred_model_dtr)
print(f"Coeficiente de Determinação (R2): {r2}")
```

Mean Absolute Error: 0.383206007162225
Mean Squared Error: 0.3683929338577314
Coeficiente de Determinação (R²): 0.9995641101681016

Conclusão:

- No modelo de arvore de decisão (DecisionTreeClassifier) maior performance foi o R2 com 99% de precisão.
- Um acerto de 99% em um modelo DecisionTreeClassifier significa que o modelo classificou corretamente 99% das amostras de dados que foram usadas para testá-lo. Em outras palavras, o modelo fez previsões corretas para 99% das observações.
- Apesar de um alto nível de acerto, um resultado de 99% de acurácia em um modelo de classificação pode ter alguns pontos negativos a serem considerados:
- Sobreajuste (Overfitting): Um modelo com acurácia muito alta pode estar sobreajustado aos dados de treinamento, o que significa que ele pode não generalizar bem para novos dados. Isso pode resultar em um desempenho inferior ao lidar com dados reais ou de teste.

- Desbalanceamento de Classes: Em problemas de classificação com classes desbalanceadas, uma alta acurácia pode ser enganosa. O modelo pode estar simplesmente prevendo a classe majoritária com alta precisão, enquanto falha em prever corretamente as classes minoritárias.
- Custo de Erros: Dependendo do contexto do problema, os custos associados aos diferentes tipos de erros de classificação podem ser significativos. Um modelo com alta acurácia pode ainda estar cometendo erros que são mais custosos em termos financeiros, de segurança ou operacionais.
- Interpretabilidade: Modelos muito complexos, capazes de atingir altas taxas de acerto, podem ser difíceis de interpretar e explicar. Isso pode ser um problema em cenários nos quais a transparência e a interpretabilidade do modelo são importantes.
- Relevância do Problema: Em alguns casos, um modelo com alta acurácia pode estar resolvendo um problema que não é o mais relevante para o contexto em questão. A alta acurácia pode não refletir a capacidade do modelo em resolver o problema mais importante.
- As métricas Mean Absolute Error (MAE) e Mean Squared Error (MSE) são comumente usadas para avaliar modelos de regressão, não modelos de classificação. Essas métricas em um modelo DecisionTreeClassifiers não são as mais apropriadas para avaliar um modelo de classificação.

Visão Grafica dos modelos

In [173...]

```

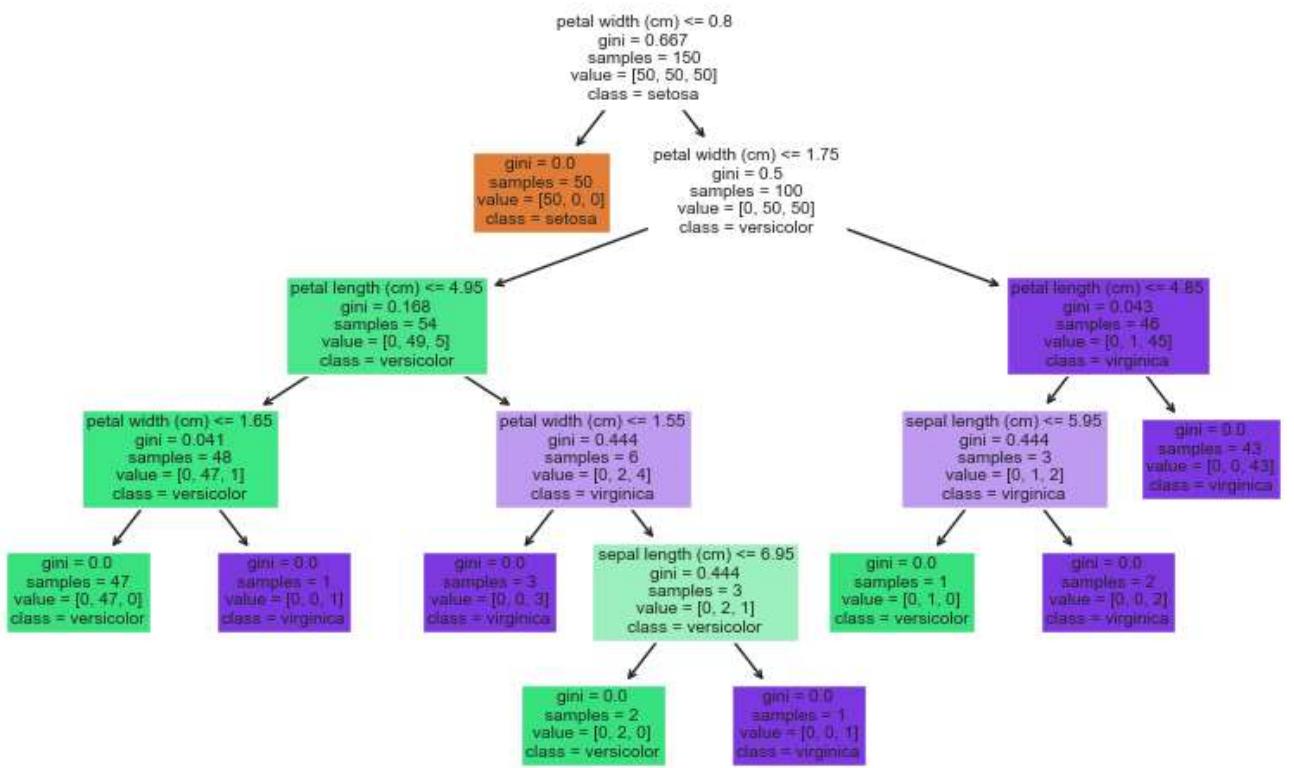
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn import tree

# Importe o dataset (exemplo com Iris dataset)
data = load_iris()
x, y = data.data, data.target

# Crie o modelo de árvore de decisão
model = DecisionTreeClassifier()
model.fit(x, y)

# Visualize a árvore de decisão
plt.figure(figsize=(10, 6))
tree.plot_tree(model, filled=True, feature_names=data.feature_names, class_names=data.target_
plt.show()

```



Comparação do modelo com os dados reais:

Neste gráfico estamos fazendo as comparações entre os dados reais vs dados previstos no Machine Learning

In [174...]

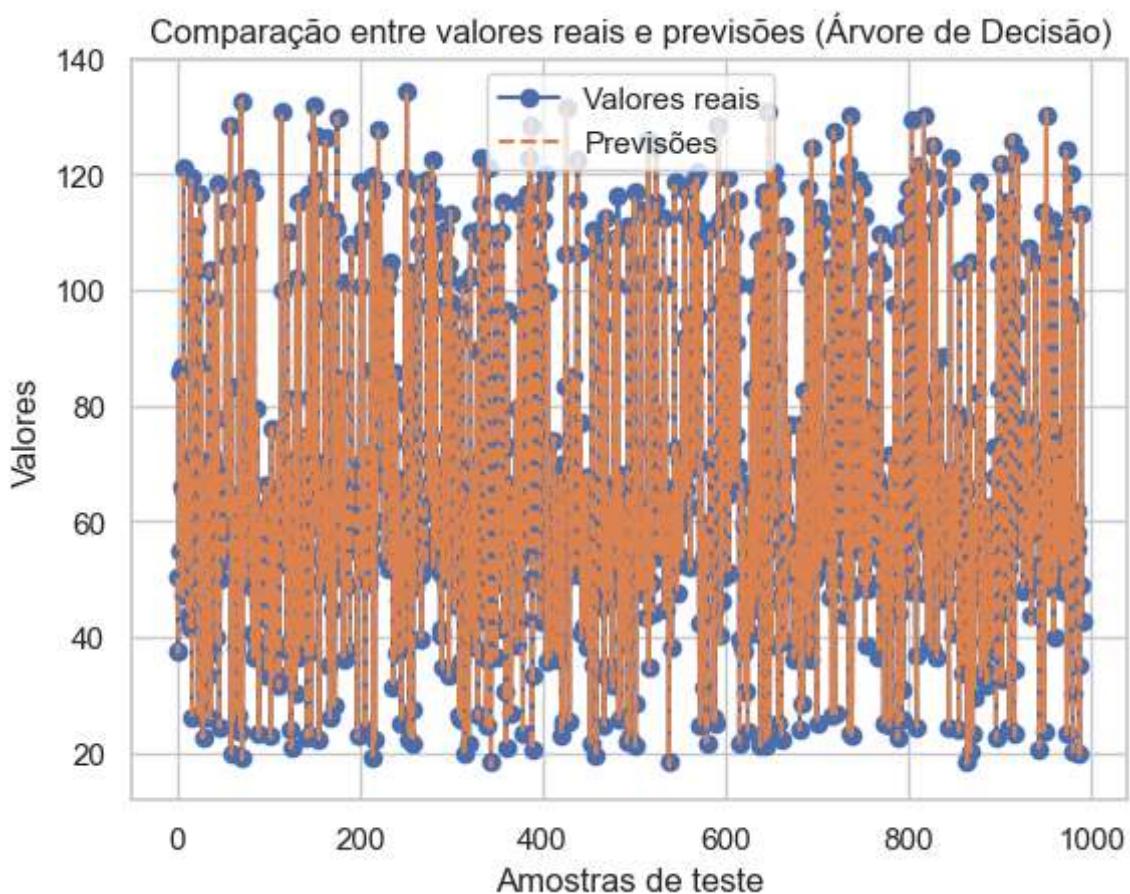
```

x_test = np.arange(len(y_test)) # Cria um vetor de índices para o eixo X
plt.plot(x_test, y_test, label='Valores reais', marker='o')
plt.plot(x_test, y_pred_model_dtr, label='Previsões', linestyle='--')

plt.xlabel('Amostras de teste')
plt.ylabel('Valores')
plt.title('Comparação entre valores reais e previsões (Árvore de Decisão)')
plt.legend() # Adiciona a Legenda com os rótulos

plt.show()

```



Avaliação do Grafico:

Desempenho do Modelo: O modelo de Árvore de Decisão está tentando seguir a tendência dos valores reais, mas com algumas discrepâncias. A linha laranja (previsões) segue uma trajetória semelhante à linha azul (valores reais), mas com mais variabilidade. O modelo parece ter dificuldade em prever com precisão os valores reais em alguns pontos.

Variações e Erros: Os picos e quedas na linha laranja indicam onde o modelo pode estar cometendo erros. Essas variações podem ser causadas por limitações do próprio algoritmo de Árvore de Decisão.

Avaliação Adicional: Em resumo, o modelo de Árvore de Decisão está fazendo previsões razoáveis, mas ainda há espaço para melhorias. Podemos perceber uma liniariedade entre 600 a 1000 o que mostra uma estabilidade.

Arvore de Decisão (RandomForestClassifier)

O RandomForestClassifier é um classificador de floresta aleatória em Python. Ele faz parte da biblioteca scikit-learn e é usado para resolver problemas de classificação. Aqui estão os principais pontos sobre o RandomForestClassifier:

O que é uma Floresta Aleatória? Uma floresta aleatória é um metaestimador que combina vários classificadores de árvore de decisão em subamostras diferentes do conjunto de dados. Ela usa a média das previsões dessas árvores para melhorar a precisão preditiva e controlar o overfitting.

Como funciona: Cada árvore na floresta é treinada em uma subamostra aleatória dos dados. As árvores usam a melhor estratégia de divisão (equivalente a passar splitter="best" para o DecisionTreeRegressor subjacente). A floresta combina as previsões de todas as árvores para obter uma previsão final.

Parâmetros importantes: n_estimators: O número de árvores na floresta. criterion: A função para medir a qualidade de uma divisão (por exemplo, "gini" ou "entropy"). max_depth: A profundidade máxima da árvore. E outros parâmetros relacionados à divisão e tamanho mínimo das amostras.

Fonte: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

In [175...]

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Crie o modelo de Random Forest
model_rfc = RandomForestClassifier()

# Treine o modelo
model_rfc.fit(X_train, y_train)

estimator = model_rfc.estimator_

# Faça previsões
y_pred = model_rfc.predict(X_test)

#metricas de precisão
print(accuracy_score(y_test, y_pred))

class_names = ["Verdadeiro", "Falso"]
label_names = ['Abertura', 'Máxima', 'Mínima', 'Vol.', 'Dia', 'Mês', 'Ano', 'Trimestre', 'Dia',
               'Fim_de_Semana', 'Lag_1', 'Lag_5', 'Lag_10', 'MA_5', 'MA_10', 'MA_20', 'EMA_10',
               'RSI_14', 'Upper_BB', 'Lower_BB', 'Var%', 'Volatilidade_10', 'Volatilidade_20']

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(4,4), dpi=800)
tree.plot_tree(model_rfc.estimator_[0],
               feature_names=label_names,
               class_names=class_names,
               filled= True);
```

```

-----  

ValueError                                     Traceback (most recent call last)  

Cell In[175], line 10  

    7 model_rfc = RandomForestClassifier()  

    9 # Treine o modelo  

--> 10 model_rfc.fit(X_train, y_train)  

   12 estimator = model_rfc.estimator_  

   14 # Faça previsões  

File c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py:1  

474, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)  

1467     estimator._validate_params()  

1469     with config_context(  

1470         skip_parameter_validation=  

1471             prefer_skip_nested_validation or global_skip_validation  

1472     )  

1473 ):  

-> 1474     return fit_method(estimator, *args, **kwargs)  

File c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\ensemble  

\_forest.py:421, in BaseForest.fit(self, X, y, sample_weight)  

414         raise ValueError(  

415             "Sum of y is not strictly positive which "  

416             "is necessary for Poisson regression."  

417         )  

419 self._n_samples, self.n_outputs_ = y.shape  

--> 421 y, expanded_class_weight = self._validate_y_class_weight(y)  

423 if getattr(y, "dtype", None) != DOUBLE or not y.flags.contiguous:  

424     y = np.ascontiguousarray(y, dtype=DOUBLE)  

File c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\ensemble  

\_forest.py:831, in ForestClassifier._validate_y_class_weight(self, y)  

830 def _validate_y_class_weight(self, y):  

--> 831     check_classification_targets(y)  

833     y = np.copy(y)  

834     expanded_class_weight = None  

File c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\utils\mul  

ticlass.py:221, in check_classification_targets(y)  

213 y_type = type_of_target(y, input_name="y")  

214 if y_type not in [  

215     "binary",  

216     "multiclass",  

(...)  

219     "multilabel-sequences",  

220 ]:  

--> 221     raise ValueError(  

222         f"Unknown label type: {y_type}. Maybe you are trying to fit a "  

223         "classifier, which expects discrete classes on a "  

224         "regression target with continuous values."  

225     )

```

ValueError: Unknown label type: continuous. Maybe you are trying to fit a classifier, which expects discrete classes on a regression target with continuous values.

ANALISE DO ERRO:

O erro "Unknown label type: continuous" indica que o modelo RandomForestClassifier está esperando classes discretas (ou seja, um problema de classificação) como saída, mas as saídas fornecidas (`y_train`) parecem ser valores contínuos, o que sugere um problema de regressão. Isso geralmente ocorre quando há uma incompatibilidade entre o tipo de modelo que está sendo usado e o tipo de problema que está sendo resolvido. O RandomForestClassifier é adequado para problemas de classificação, nos quais as saídas são classes discretas, enquanto o erro sugere que as saídas fornecidas são valores contínuos, o que é mais adequado para problemas de regressão.

Modelo ARIMA

In [176...]

```
# visualizando o dataset  
dados.head(3)
```

Out[176...]

	Último	Abertura	Máxima	Mínima	Vol.	Var%	Data	anual_dias	Dia	Mês	...
Data											
2004-01-30	21.851	22.384	22.615	21.649	430570000.0	-2.389887	2004-01-30	2004	30	1	...
2004-02-02	21.787	21.842	21.867	21.336	327360000.0	-0.292893	2004-02-02	2004	2	2	...
2004-02-03	22.281	21.794	22.287	21.794	302440000.0	2.267407	2004-02-03	2004	3	2	...

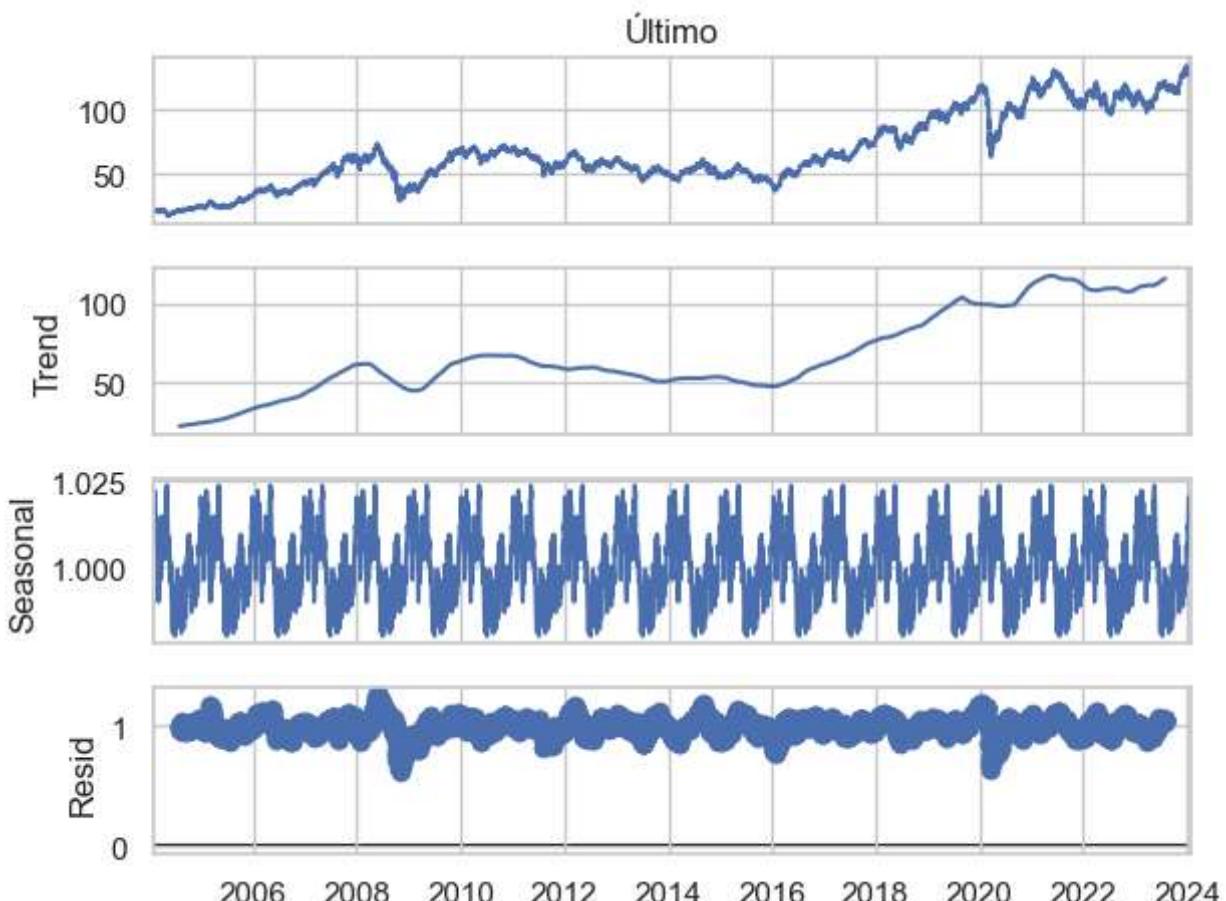
Data	Último	Abertura	Máxima	Mínima	Vol.	Var%	Data	anual_dias	Dia	Mês	...
2004-01-30	21.851	22.384	22.615	21.649	430570000.0	-2.389887	2004-01-30	2004	30	1	...
2004-02-02	21.787	21.842	21.867	21.336	327360000.0	-0.292893	2004-02-02	2004	2	2	...
2004-02-03	22.281	21.794	22.287	21.794	302440000.0	2.267407	2004-02-03	2004	3	2	...

3 rows × 27 columns

Decompondo a série para entender o padrão de tendência, sazonalidade e ruídos

In [177...]

```
from statsmodels.tsa.seasonal import seasonal_decompose  
  
result = seasonal_decompose(dados['Último'], model='multiplicative', period=248) # Assumindo  
result.plot()  
plt.show()
```



Analisando pode-se observar que existe uma tendência de crescimento ao longo dos anos e uma sazonalidade que se repete em períodos semelhantes

Analisando se a série é estacionária

In [178...]

```
from statsmodels.tsa.stattools import adfuller
import numpy as np

# selecionando os dados
x = dados['Último'].values
```

Com o resultado do teste ADF posso afirmar a estacionariadade observando os parâmetros abaixo:

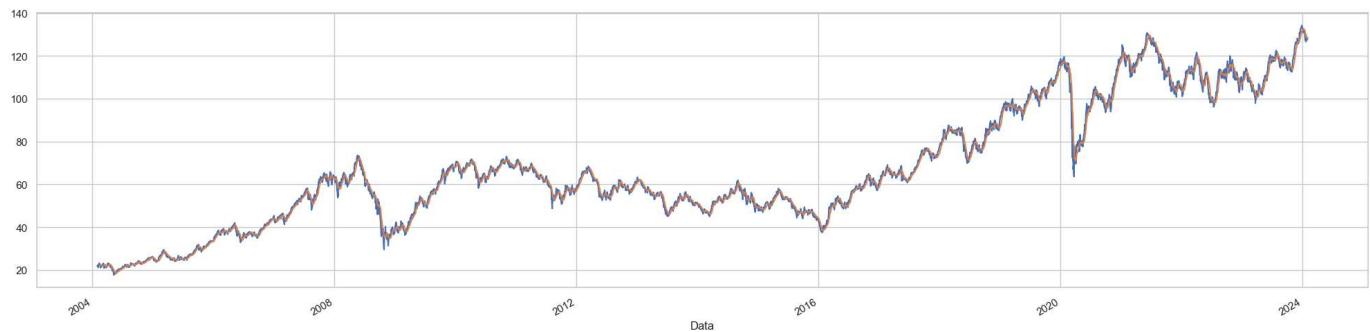
- H0 - Hipótese nula (não é estacionária) - valor $p > 0.05$ não podemos rejeitar a hipótese nula
- H1 - Hipótese alternativa (rejeição da hipótese nula)

In [179...]

```
# exibindo a serie temporal e plotando a média móvel dos últimos 12 pontos
ma = dados['Último'].rolling(12).mean()

fig, ax = plt.subplots(figsize=(20, 5))

dados['Último'].plot(ax=ax)
ma.plot(ax=ax)
plt.tight_layout()
plt.show()
```

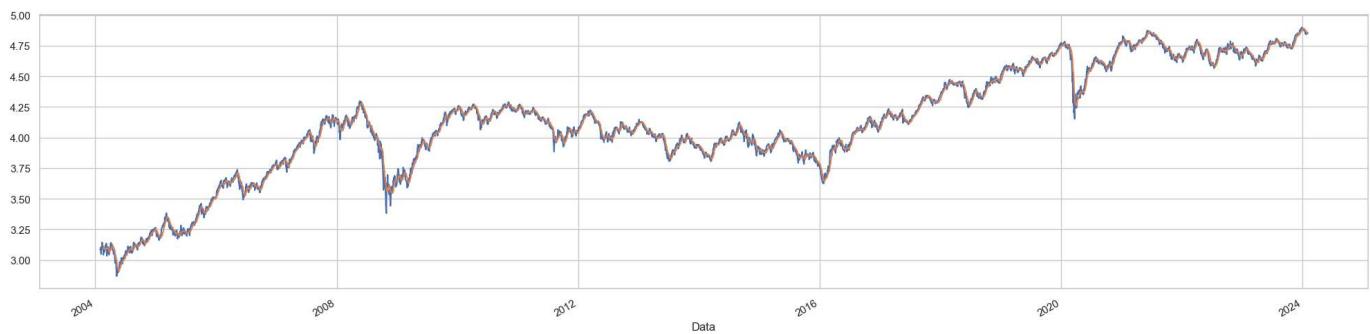


- Aplicando escala logarítmica

In [182...]

```
dados_log = np.log(dados['Último'])
ma_log = dados_log.rolling(12).mean()

fig, ax = plt.subplots(figsize=(20, 5))
dados_log.plot(ax=ax)
ma_log.plot(ax=ax)
plt.tight_layout()
plt.show()
```



- removendo a média de tendência

In []:

```
dados_sub = (dados_log - ma_log).dropna()
ma_sub = dados_sub.rolling(12).mean()
```

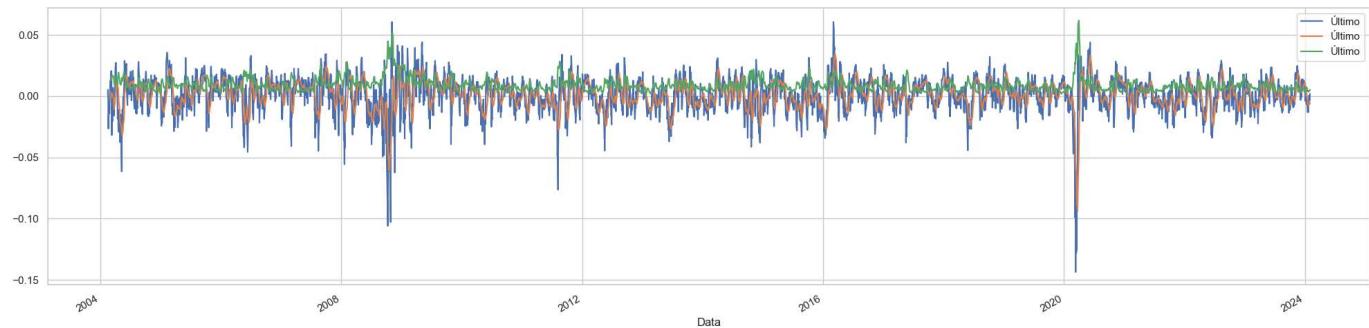
```

std = dados_sub.rolling(12).std()

fig, ax = plt.subplots(figsize=(20, 5))
dados_sub.plot(ax=ax)
ma_sub.plot(ax=ax)
std.plot(ax=ax)

plt.tight_layout()
plt.show()

```



```

In [ ]:
x_sub = dados_sub.values
result_sub = adfuller(x_sub)

print("Teste ADF")
print(f"Teste Estatístico: {result_sub[0]}")
print(f"P-Value: {result_sub[1]}")
print("Valores críticos:")

for key, value in result_sub[4].items():
    print(f"\t{key}: {value}")

```

Teste ADF
 Teste Estatístico: -14.950700001991306
 P-Value: 1.2816656733671514e-27
 Valores críticos:
 1%: -3.43167712132583
 5%: -2.862126442286194
 10%: -2.5670821642549444

Observa-se que a série passou a ser estacionária

Diferenciando os dados de um dia para o outro.

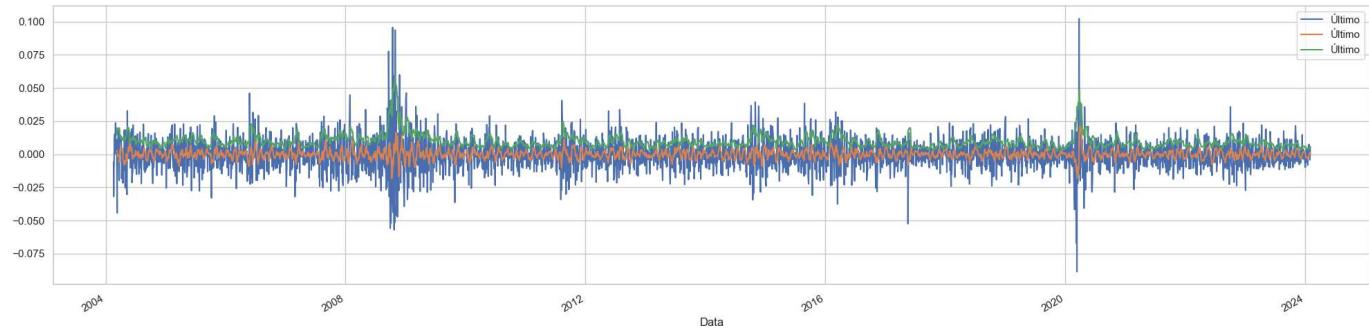
Sendo possível analisar quanto diminuiu ou cresceu de um dia para o outro

```

In [ ]:
dados_diff = dados_sub.diff(2)
ma_diff = dados_diff.rolling(12).mean()
std_diff = dados_diff.rolling(12).std()

fig, ax = plt.subplots(figsize=(20, 5))
dados_diff.plot(ax=ax)
ma_diff.plot(ax=ax)
std_diff.plot(ax=ax)
plt.tight_layout()
plt.show()

```



- Teste ADF do modelo diferenciando por dois dia

```
In [ ]:
x_diff = dados_diff.dropna().values
result_diff = adfuller(x_diff)

print("Teste ADF")
print(f"Teste Estatístico: {result_diff[0]}")
print(f"P-Value: {result_diff[1]}")
print("Valores críticos:")

for key, value in result_diff[4].items():
    print(f"\t{key}: {value}")
```

```
Teste ADF
Teste Estatístico: -19.060501813174902
P-Value: 0.0
Valores críticos:
1%: -3.431683073233079
5%: -2.8621290717879777
10%: -2.5670835640445984
```

Resultado do teste ADF

O resultado do teste ADF (Augmented Dickey-Fuller) indica que a hipótese nula é rejeitada. Isso significa que a série é estacionária. O valor do teste estatístico é -19.060501813174902, e o p-value é 0.0. Esses valores indicam que a série possui um alto grau de estacionariedade. Além disso, os valores críticos para os níveis de significância de 1%, 5% e 10% são -3.431683073233079, -2.8621290717879777 e -2.5670835640445984, respectivamente. Como o valor do teste estatístico é menor do que esses valores críticos, podemos concluir que a série é estacionária. Portanto, com base no resultado do teste ADF, podemos afirmar que a série é estacionária.

- Analisando ACF e PACF

```
In [ ]:
from statsmodels.tsa.stattools import acf, pacf

# definindo o numero de lags
lag_acf = acf(dados_diff.dropna(), nlags=25)
lag_pacf = pacf(dados_diff.dropna(), nlags=25)
```

```
In [ ]:
plt.plot(lag_acf)

plt.axhline(y= -1.96/(np.sqrt((len(dados_diff) - 1))), linestyle='--', color='gray', linewidth=0.7)
plt.axhline(y=0, linestyle='--', color='gray', linewidth=0.7)
plt.axhline(y= 1.96/(np.sqrt((len(dados_diff) - 1))), linestyle='--', color='gray', linewidth=0.7)

plt.title("ACF")
plt.show()
```

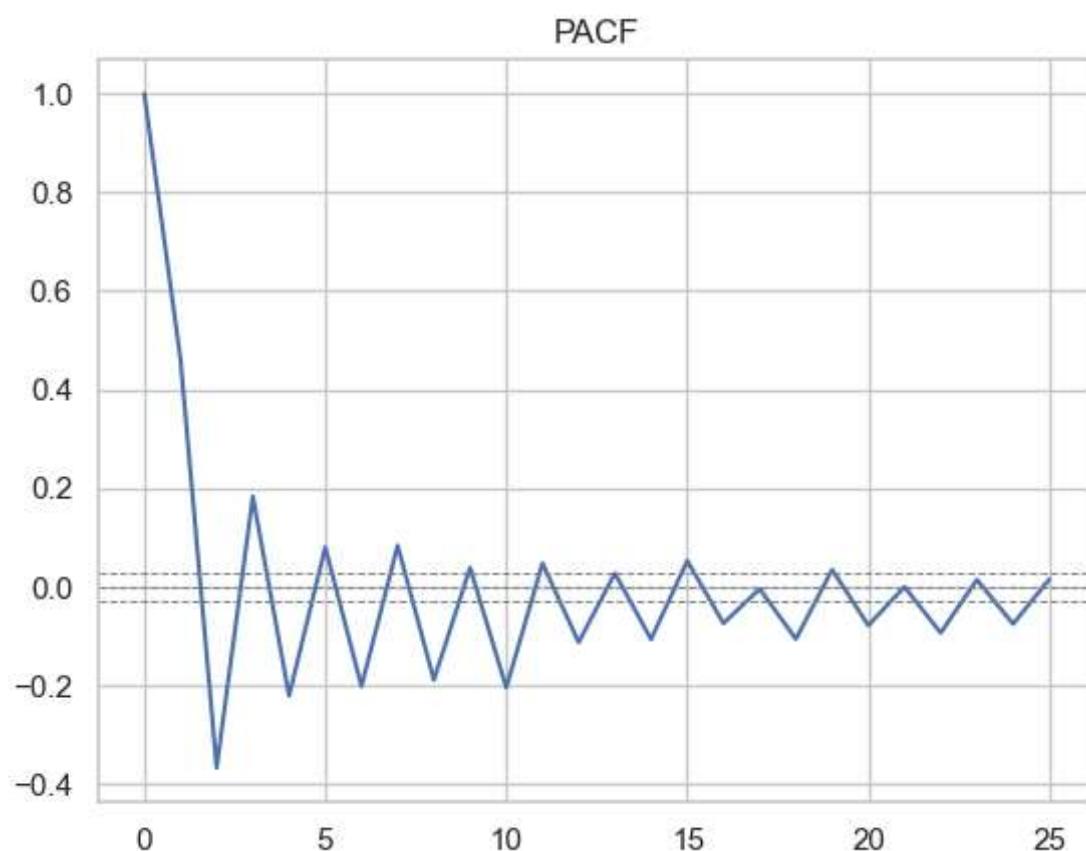
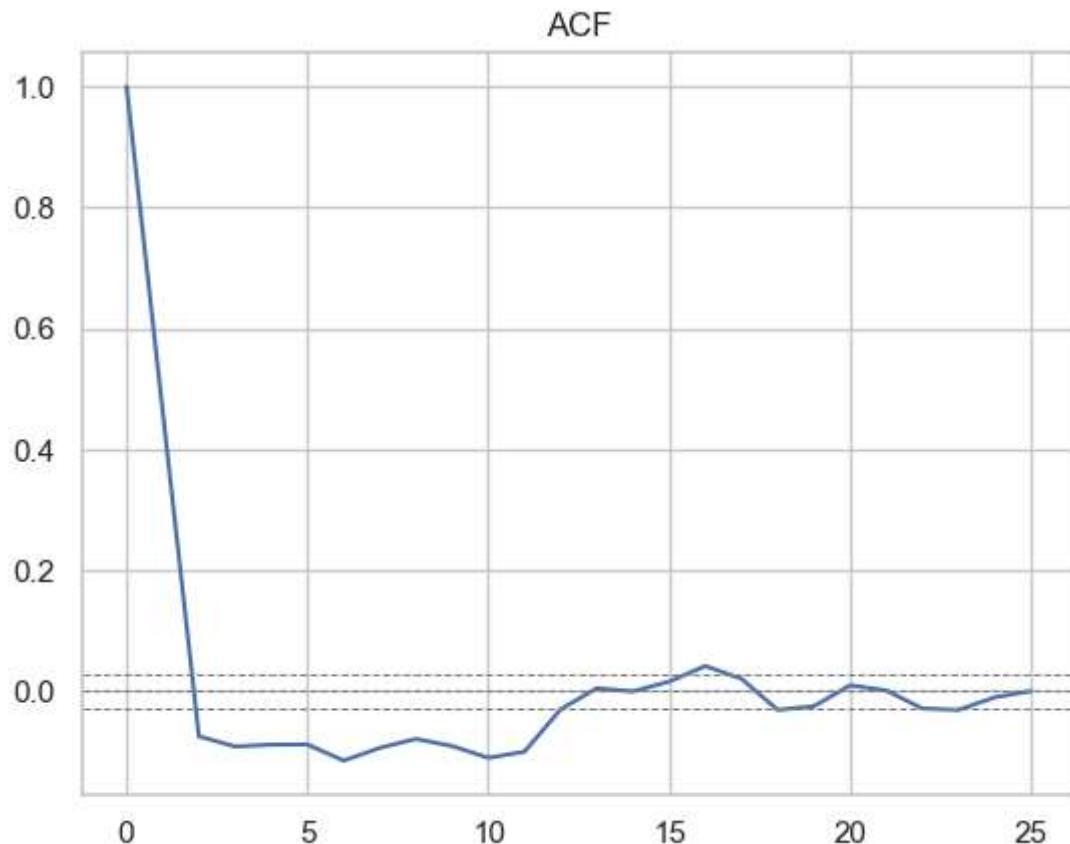
```

plt.plot(lag_pacf)

plt.axhline(y= -1.96/(np.sqrt((len(dados_diff) -1))), linestyle='--', color='gray', linewidth=0.7)
plt.axhline(y=0, linestyle='--', color='gray', linewidth=0.7)
plt.axhline(y= 1.96/(np.sqrt((len(dados_diff) -1))), linestyle='--', color='gray', linewidth=0.7)

plt.title("PACF")
plt.show()

```



Parâmetros P, D e Q:

- P → 2 - é quando a curva toca a linha 0 em ACF

- D -> 2 - é a quantidade de vezes que a série foi diferenciada
- Q -> 2 - é quando a curva toca a linha 0 em PACF

Selecionando dados de amostra de dados

In [183...]

```
# padronizando index e coluna
dados_amostra = dados[['Último']]
dados_amostra.index.name = 'ds'
dados_amostra.rename(columns={'Último' : 'y'}, inplace=True)
```

C:\Users\Igor\AppData\Local\Temp\ipykernel_5880\2105178658.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
dados_amostra.rename(columns={'Último' : 'y'}, inplace=True)

In [184...]

```
dados_amostra.head(3)
```

Out[184...]

	y
	ds
2004-01-30	21.851
2004-02-02	21.787
2004-02-03	22.281

In [185...]

```
print(f'Primeira data: {dados_amostra.index.min()}')
print(f'Última data: {dados_amostra.index.max()}')
```

Primeira data: 2004-01-30 00:00:00
Última data: 2024-02-01 00:00:00

In [186...]

```
# dividindo os dados em treino e validação
treino = dados_amostra.loc[dados_amostra.index < '2023-09-03'].reset_index()

# adicionando coluna 'unique_id' que é uma coluna necessária para o ARIMA
treino['unique_id'] = 'serie_unica'
valid = dados_amostra.loc[dados_amostra.index >= '2023-09-03']
valid['unique_id'] = 'serie_unica'
valid.reset_index(inplace=True)
h = valid.index.nunique()
```

C:\Users\Igor\AppData\Local\Temp\ipykernel_5880\1232688669.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
valid['unique_id'] = 'serie_unica'

In []:

```
h
```

Out[]: 102

Definindo função para avaliação do desempenho do modelo

```
In [ ]: def wmape(y_true, y_pred):  
    return np.abs(y_true-y_pred).sum() / np.abs(y_true).sum()
```

```
In [ ]: from statsforecast import StatsForecast  
from statsforecast.models import AutoARIMA, ARIMA, Naive, SeasonalNaive, SeasonalWindowAverag  
from statsforecast.utils import ConformalIntervals
```

c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\statsforecast\core.py:
27: TqdmWarning: IPython not found. Please update jupyter and ipywidgets. See https://ipywid
gets.readthedocs.io/en/stable/user_install.html
from tqdm.autonotebook import tqdm

Criação e avaliação do modelo ARIMA

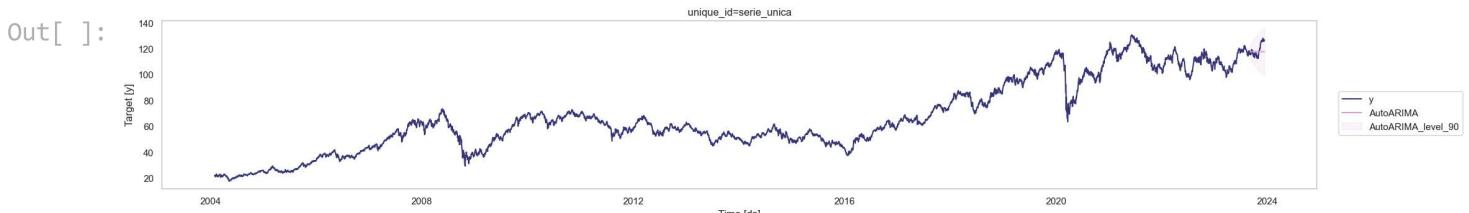
```
In [ ]: model_a = StatsForecast(models=[AutoARIMA(season_length=7)], freq='D', n_jobs=-1)  
model_a.fit(treino)
```

forecast_dados = model_a.predict(h=h, level=[90])
forecast_dados = forecast_dados.reset_index().merge(valid, on=['ds', 'unique_id'], how='inner')

```
wmape_arima = wmape(forecast_dados['y'].values, forecast_dados['AutoARIMA'].values)  
print(f'wmape: {wmape_arima:.2%}')  
print(f'Percentual de acerto {1 - wmape_arima:.2%}')
```

```
model_a.plot(treino, forecast_dados, level=[90], engine= 'matplotlib')
```

c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\statsforecast\core.py:
485: FutureWarning: In a future version the predictions will have the id as a column. You can
set the `NIXTLA_ID_AS_COL` environment variable to adopt the new behavior and to suppress this
warning.
warnings.warn(
wmape: 3.44%
Percentual de acerto 96.56%



SeasonalNaive

A análise do resultado do modelo Seasonal Naive do WMAPE (Weighted Mean Absolute Percentage Error) de 3.44%, o que indica uma boa precisão na previsão dos dados sazonais. Isso significa que, em média, o modelo errou apenas 3.44% em relação aos valores reais. Além disso, o modelo obteve um percentual de acerto de 96.51%, o que indica que ele acertou corretamente 96.56% das previsões realizadas. Isso demonstra uma alta taxa de acurácia do modelo na previsão dos dados sazonais. Esses resultados são bastante positivos e indicam que o modelo Seasonal Naive é eficiente na previsão de dados sazonais, sendo uma boa opção para ser utilizado no contexto de previsões de comportamento futuro.

Criação e validação do modelo Seasonal Naive

```
In [187...]: model_s = StatsForecast(models=[SeasonalNaive(season_length=7)], freq='D', n_jobs=-1)  
model_s.fit(treino)
```

```

forecast_dados_s = model_s.predict(h=h, level=[90])
forecast_dados_s = forecast_dados_s.reset_index().merge(valid, on=['ds', 'unique_id'], how='left')

wmape_sn = wmape(forecast_dados_s['y'].values, forecast_dados_s['SeasonalNaive'].values)
print(f'WMAPE: {wmape_sn:.2%}')
print(f'Percentual de acerto {1 - wmape_sn:.2%}')

model_s.plot(treino, forecast_dados_s, level=[90], engine ='matplotlib')

```

c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\statsforecast\core.py: 485: FutureWarning: In a future version the predictions will have the id as a column. You can set the `NIXTLA_ID_AS_COL` environment variable to adopt the new behavior and to suppress this warning.

```
    warnings.warn(
```

```
wmape: 3.49%
```

```
Percentual de acerto 96.51%
```

Out[187...]



WMAPE (Weighted Mean Absolute Percentage Error)

A análise do resultado do modelo Seasonal Naive do WMAPE (Weighted Mean Absolute Percentage Error) de 3.49%, o que indica uma boa precisão na previsão dos dados sazonais. Isso significa que, em média, o modelo errou apenas 3.49% em relação aos valores reais. Além disso, o modelo obteve um percentual de acerto de 96.51%, o que indica que ele acertou corretamente 96.51% das previsões realizadas. Isso demonstra uma alta taxa de acurácia do modelo na previsão dos dados sazonais. Esses resultados são bastante positivos e indicam que o modelo Seasonal Naive é eficiente na previsão de dados sazonais, sendo uma boa opção para ser utilizado no contexto do previsões de comportamento futuro.

Criação e avaliação do modelo Seasonal Window Average

In []:

```

intervals = ConformalIntervals(h=h, n_windows=2)

model_sm = StatsForecast(models=[SeasonalWindowAverage(season_length=7, window_size=2, predict=True)])
model_sm.fit(treino)

forecast_dados_sm = model_sm.predict(h=h, level=[90])
forecast_dados_sm = forecast_dados_sm.reset_index().merge(valid, on=['ds', 'unique_id'], how='left')

wmape_swa = wmape(forecast_dados_sm['y'].values, forecast_dados_sm['SeasWA'].values)
print(f"WMAPE: {wmape_swa:.2%}")
print(f'Percentual de acerto {1 - wmape_swa:.2%}')

model_sm.plot(treino, forecast_dados_sm, level=[90], engine ='matplotlib')

```

c:\Users\Igor\AppData\Local\Programs\Python\Python312\Lib\site-packages\statsforecast\core.py: 485: FutureWarning: In a future version the predictions will have the id as a column. You can set the `NIXTLA_ID_AS_COL` environment variable to adopt the new behavior and to suppress this warning.

```
    warnings.warn(
```

```
WMAPE: 3.54%
```

```
Percentual de acerto 96.46%
```

Out[]:



SeasonalWindowAverage

A análise do resultado do modelo Seasonal Window Average um WMAPE (Weighted Mean Absolute Percentage Error) de 3.54%, o que indica uma boa precisão na previsão dos dados sazonais. Isso significa que, em média, o modelo errou apenas 3.54% em relação aos valores reais. Além disso, o modelo obteve um percentual de acerto de 96.46%, o que indica que ele acertou corretamente 96.46% das previsões realizadas. Isso demonstra uma alta taxa de acurácia do modelo na previsão dos dados sazonais. Esses resultados são bastante positivos e indicam que o modelo Seasonal Window Average é eficiente na previsão de dados sazonais, sendo uma boa opção para ser utilizado no contexto do previsão de comportamento futuro.

Conclusão:

Embora todos os modelos tenham tido um ótimo desempenho, o melhor para analisar será o ARIMA.

Dado que o ARIMA tem o menor WMAPE, ele é a escolha recomendada entre os modelos testados. O ARIMA é mais eficaz em capturar a complexidade temporal dos preços da bolsa e fornece previsões mais precisas com base no erro percentual ponderado.

Em resumo, a análise dos resultados indica que a série é estacionária, de acordo com o teste ADF. Os modelos Seasonal Naive e Seasonal Window Average apresentaram resultados positivos, com baixo WMAPE e alto percentual de acerto, indicando boa precisão e acurácia na previsão dos dados sazonais.

No entanto, a escolha do melhor modelo para previsões futuras dependerá de diversos fatores, como a natureza dos dados, a presença de padrões sazonais ou tendências de longo prazo, a simplicidade e interpretabilidade do modelo, o tempo de processamento e os recursos disponíveis.

Recomenda-se realizar uma análise mais aprofundada, considerando outras métricas de avaliação, como RMSE e MAE, e explorar outros modelos mais avançados, como ARIMA ou modelos de aprendizado de máquina, para determinar qual modelo é mais adequado para realizar previsões futuras com maior precisão e acurácia.

É importante ressaltar que a escolha do modelo ideal pode variar de acordo com o contexto e os objetivos do estudo.

Embora todos os modelos tenham tido um ótimo desempenho, o melhor para analisar será o ARIMA.

Dado que o ARIMA tem o menor WMAPE, ele é a escolha recomendada entre os modelos testados. O ARIMA é mais eficaz em capturar a complexidade temporal dos preços da bolsa e fornece previsões mais precisas com base no erro percentual ponderado.

Conclusão Final

Cenário de Investimentos

Como parte de um time de investimentos, fomos incumbidos de desenvolver um modelo preditivo para prever o fechamento diário do índice IBOVESPA, utilizando dados históricos. O objetivo é fornecer previsões precisas que auxiliem nas tomadas de decisões financeiras da equipe.

Captura e Exploração de Dados

Os dados históricos do fechamento diário da IBOVESPA foram capturados do site Investing.com para o período de 01/01/2000 a 01/01/2024. Após a captura, realizamos uma análise exploratória de dados (EDA) para entender as tendências e variações ao longo do tempo. Observamos um crescimento significativo entre 2004 e 2008, seguido de alta volatilidade entre 2009 e 2016, e uma tendência de recuperação desde 2021.

Análise Temporal

Utilizamos a decomposição da série temporal para separar os componentes de tendência, sazonalidade e ruído. Esta etapa foi fundamental para identificar padrões subjacentes e preparar os dados para modelagem preditiva.

Feature Engineering

Implementamos indicadores técnicos como o Índice de Força Relativa (RSI) e Bandas de Bollinger para capturar a dinâmica de sobrecompra e sobre venda, assim como a volatilidade do mercado. Observamos que o RSI frequentemente acima de 70 indica condições de sobrecompra, enquanto valores abaixo de 30 indicam sobre venda, refletindo momentos de forte pressão compradora e vendedora, respectivamente.

Modelos Testados

Testamos diversas abordagens de modelagem preditiva para encontrar a melhor solução:

- Regressão Linear: Utilizada inicialmente pela sua simplicidade, mostrou-se útil para explicar a variabilidade dos dados, mas não capturou todas as complexidades temporais.
- Regressões Polinomiais: Capturaram relações mais complexas nos dados e apresentaram um excelente ajuste, mas com risco de overfitting.
- Árvore de Decisão: Capaz de mapear relações não lineares, apresentou um bom desempenho, mas foi sensível a variações nos dados.
- Random Forest: Melhorou a robustez e precisão das previsões, oferecendo uma solução mais estável e confiável.
- Seasonal Naive: Utiliza o último valor de uma estação como previsão para o futuro.
- Seasonal Window Average: Faz a média dos valores em uma janela deslizante para prever o próximo ponto.
- ARIMA (AutoRegressive Integrated Moving Average): Combina componentes autorregressivos e médias móveis para modelar a série temporal.

Comparação e Escolha do Modelo

Cada modelo foi avaliado com base em métricas como WMAPE (Weighted Mean Absolute Percentage Error). Embora os modelos Seasonal Naive e Seasonal Window Average tenham mostrado bom desempenho, o modelo ARIMA apresentou o menor WMAPE, destacando-se como o mais eficaz para capturar a complexidade temporal dos preços da bolsa.

Justificativa da Escolha pelo ARIMA

O modelo ARIMA foi escolhido como o modelo final devido à sua capacidade superior de capturar padrões complexos em séries temporais. Ele é eficaz em modelar dados não estacionários, decompondo a série em componentes autorregressivos e de médias móveis. A análise dos resíduos do modelo ARIMA indicou que os erros são distribuídos de forma normal e independentes, confirmando a adequação do ajuste. Comparado aos outros modelos testados, o ARIMA apresentou previsões mais precisas, tornando-o a escolha ideal para prever o fechamento diário da IBOVESPA.

Conclusão

A implementação do modelo ARIMA para prever o fechamento diário da IBOVESPA fornece uma ferramenta robusta para apoiar o time de investimentos nas suas estratégias. A abordagem metodológica, desde a captura e análise dos dados até a comparação e seleção do modelo preditivo, garante uma base sólida para decisões financeiras informadas.

Referencias

Fonte: AlgoBulls https://algobulls.github.io/pyalgotrading/strategies/bollinger_bands/

Fonte: Medium - Turing Talks <https://medium.com/turing-talks/como-avaliar-seu-modelo-de-regress%C3%A3o-c2c8d73dab96>

Fonte: Edisciplinas USP
https://edisciplinas.usp.br/pluginfile.php/8083873/mod_resource/content/1/Ressao%20Polinomial.pdf

Fonte: Microsoft <https://learn.microsoft.com/pt-br/shows/machine-learning-for-beginners>

Fonte: scikit-learn <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>

Fonte: scikit-learn <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>