FIAP

TECH CHALLENGE FASE II

3DTAT

Grupo 35

RM352525

Karina Guerra do Nascimento

2024

SUMÁRIO

ÍNDICE DE ILUSTRAÇÃO

CÓDIGOS

GRÁFICOS

INTRODUÇÃO

O mercado financeiro desempenha um papel fundamental na economia brasileira, sendo um ambiente onde investidores, empresas e instituições financeiras interagem para negociar ativos como ações, títulos e moedas. No Brasil, a Bolsa de Valores é representada pela B3, que abriga o principal índice de referência, o Ibovespa.

A importância do mercado financeiro no Brasil é evidenciada pela sua capacidade de captar recursos para financiar projetos de empresas, governos e indivíduos, além de permitir a diversificação de investimentos e a mitigação de riscos. Nesse contexto, as previsões de mercado, ou *forecasting*[1], desempenham um papel crucial.

O *forecasting* no mercado financeiro refere-se à capacidade de prever o comportamento futuro dos ativos financeiros com base em análises e modelos estatísticos. Essas previsões são fundamentais para orientar as decisões de investimento, permitindo aos investidores identificar oportunidades e gerenciar riscos de forma mais eficaz.

No Brasil, o *forecasting* no mercado financeiro tem sido cada vez mais utilizado, especialmente com o avanço da tecnologia e o acesso a dados em tempo real. Modelos de *Machine Learning (ML)*[2], têm sido aplicados na tentativa de previsão de preços de ações, índices e taxas de câmbio, contribuindo para uma tomada de decisão mais fundamentada e estratégica.

Este projeto visa explorar a importância do *forecasting* no mercado financeiro brasileiro, utilizando modelos de *Machine Learning* para prever o comportamento do Ibovespa. O objetivo é fornecer insights valiosos para investidores e analistas, ajudando-os a tomar decisões mais informadas e aprimorar suas estratégias de investimento.

[1] Previsão, em estatística, é o processo de estimativas em situações de incertezas.

[2] Inteligência Artificial que utiliza algoritmos para capacitar os computadores a identificar padrões em grandes volumes de dados e realizar previsões, permitindo assim a análise preditiva.

OBJETIVO

Este relatório tem como objetivo apresentar as aplicações de técnicas avançadas de análise de dados e *machine learning*, desde a captura dos dados até a entrega de um modelo preciso e confiável. Utilizando a base de dados disponível da Ibovespa (BVSP), selecionamos o período "diário" e determinamos o intervalo de tempo de 5 anos (2019 -2024) identificado como o mais adequado para o treinamento do modelo.

Além disso, para atender aos requisitos, buscamos uma acuracidade superior a 70%, o que é crucial para garantir a confiabilidade das previsões e auxiliar nas decisões de investimento. Para atingir esse objetivo, justificamos a técnica utilizada com base na análise exploratória dos dados e na escolha de algoritmos de *machine learning* mais adequados para o problema em questão.

1. COMPREENÇÃO DOS DADOS PARA ANÁLISE

Os dados obtidos do site https://br.investing.com/indices/bovespa-historical-data para o período de 2019 a 2024 incluem as seguintes informações:

- **Data:** a data em que os dados foram registrados.
- **Último:** o preço de fechamento do índice Bovespa naquela data.
- **Abertura:** o preço de abertura do índice Bovespa naquela data.
- **Máxima:** o preço máximo atingido pelo índice Bovespa naquela data.
- **Mínima:** o preço mínimo atingido pelo índice Bovespa naquela data.
- **Vol.:** o volume de negociações do índice Bovespa naquela data, em milhões.
- **Var%:** a variação percentual do índice Bovespa em relação ao dia anterior.

A análise dos dados disponibilizados revela que são informações valiosas para prever o comportamento futuro do índice Bovespa, sendo essenciais para o desenvolvimento de modelos de *Machine Learning*.

2. BREVE HISTÓRIA SOBRE O ÍNDICE BOVESPA (IBOVESPA)

O Índice Bovespa, ou Ibovespa, é o principal indicador do desempenho médio das cotações das ações negociadas na Bolsa de Valores de São Paulo. Sua história remonta a 1968, quando foi criado o Ibov também conhecida como Ibovespa com o objetivo de ser um indicador do mercado acionário brasileiro.

Antes do surgimento do Ibovespa, existiram seus antecessores, como o Índice Bolsa de Valores (IBV) da Bolsa de Valores do Rio de Janeiro. Embora esse índice e a bolsa não existam mais, a metodologia desenvolvida para o IBV foi fundamental para a criação do que é hoje o Ibovespa.

No início, o Ibovespa era composto por apenas 17 ações, incluindo empresas como a Ambev (antiga Antarctica), o Banco Itaú, as Lojas Americanas e a Companhia Vale do Rio Doce, que mais tarde foi privatizada e passou a se chamar Vale, e que até hoje fazem parte. Ao longo dos anos, o índice passou por diversas mudanças em sua metodologia de cálculo e composição, refletindo o crescimento e a evolução do mercado de capitais no Brasil.

O Ibovespa se tornou um importante referencial para investidores, analistas e gestores de fundos de investimento, sendo utilizado como *benchmark* [3]para avaliar o desempenho de carteiras de ações e fundos de investimento. Sua trajetória ao longo das décadas reflete os altos e baixos da economia brasileira, sendo um reflexo dos momentos de otimismo e pessimismo do mercado financeiro.

Atualmente, o Ibovespa é acompanhado de perto por investidores de todo o mundo, sendo considerado um dos principais indicadores do mercado acionário brasileiro e uma importante ferramenta para compreender a dinâmica e os movimentos do mercado de capitais no país.

3. ANÁLISE EXPLORATÓRIA DOS DADOS (EDA)

3.1. CARREGAMENTO DOS DADOS E TRATAMENTO DOS DADOS

Para a construção do modelo de previsão, inicialmente foi realizado uma preparação para que esses dados possam ser devidamente utilizados para este processo foram seguidos os seguintes passos:

1. **Coleta dos Dados:** Os dados foram coletados do site Investing.com, abrangendo o período de 15/02/2019 a 15/02/2024.

2. **Carregamento dos Dados:** Utilizando a biblioteca pandas para ler um arquivo CSV contendo os dados da IBOVESPA. O arquivo foi lido a partir de uma URL pública e os dados foram armazenados em um *DataFrame*[4] chamado *df_bovespa*.

3. **Verificação e Tratamento dos Dados:**

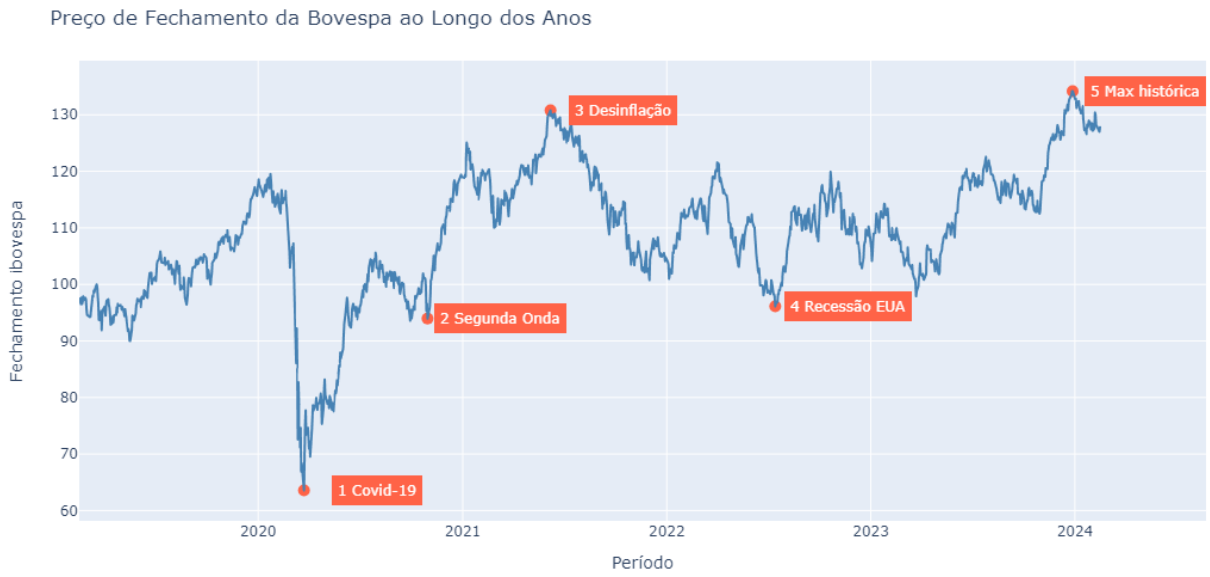
- Verificação da dimensão dos dados que são 1242 linhas e 7 colunas.

[3] processo de busca das melhores práticas de gestão da entidade numa determinada indústria e que conduzem ao desempenho superior.
[4] objetos bidimensionais, de tamanho variável. O seu formato é de uma tabela, onde os dados são organizados em linhas e colunas.

- Verificação dos tipos de dados (*dtypes*[5]), identificando colunas que precisavam de conversão (data para *datetime* e "Vol." e "Var%" para *float* ou *int*).
- Verificar se no *DataFrame* existem valores nulos e duplicados.
- Ajuste dos dados do *DataFrame*, incluindo renomeação de colunas, conversão de tipos de dados, ordenação dos dados por data e ajuste da escala dos valores de volume.

O primeiro gráfico mostra o total de fechamento da bolsa de valores no período de 2019 a 2024. Visualizar os dados por completo é importante para ter uma visão geral e identificar tendências ao longo do tempo.

Gráfico 1 - Fechamento da Ibovespa ao longo dos anos



Fonte: figuras criadas pelo próprio autor

Isso ajuda a entender como o mercado se comportou em diferentes períodos, quais foram os picos e quedas, e se há algum padrão sazonal ou tendência de longo prazo. Essas informações são cruciais para a análise de séries temporais e podem orientar a construção de modelos preditivos mais precisos.

[5] É um atributo de um DataFrame que fornece informações sobre os tipos de dados de cada coluna. Os tipos de dados mais comuns são int64 (inteiro), float64 (ponto flutuante), object (texto), datetime64 (data e hora) e bool (booleano)

No Gráfico 1, percebe-se que não há um padrão definido de comportamento e por tratar de dados econômicos a política e o cenário mundial e o que afeta. Como isso foram destacados alguns pontos para entender o que ocorreu na época.

1. O primeiro ponto dia 23 de março de 2020 os temores em relação aos impactos econômicos da pandemia do coronavírus persistem, com expectativas de recessão global apesar dos estímulos dos bancos centrais e governos. O impasse legislativo nos EUA sobre um pacote de estímulo também afetou negativamente o mercado. (G1, 2020)

2. O segundo ponto dia 30 de outubro de 2020 o diagnóstico de covid-19 de Trump e o aumento de casos na Europa e nos EUA, levando a variações significativas no mercado. Os números recordes de contaminação nos EUA e as medidas restritivas na Europa causaram preocupação entre os investidores, levando a vendas de ações em meio a temores de novos fechamentos econômicos. Além disso, as incertezas fiscais no Brasil também afetaram o mercado e a percepção internacional sobre o país. (Goeking, 2020)

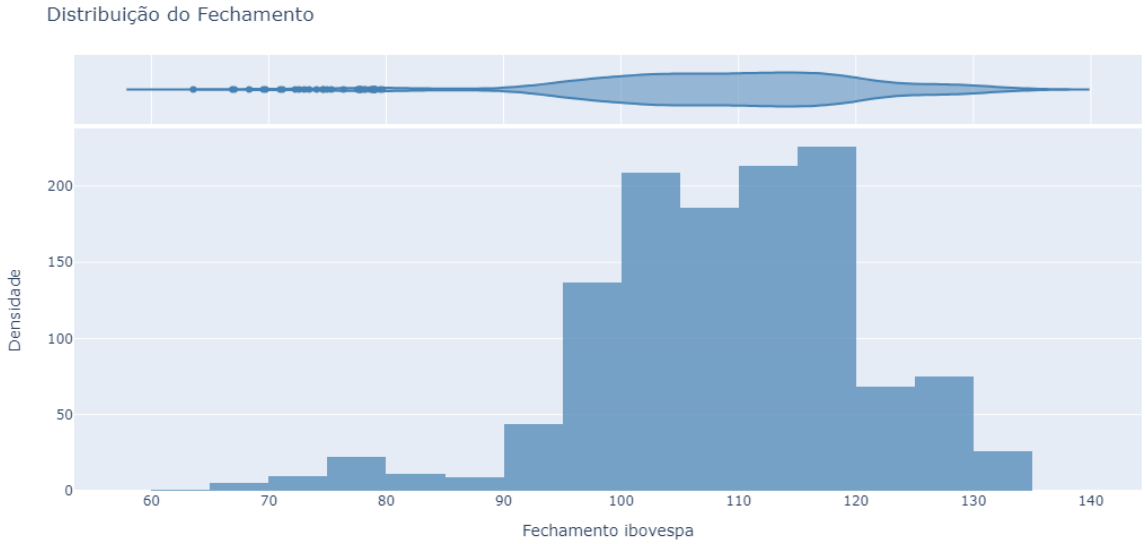
3. O terceiro ponto dia 07 junho de 2021 o mercado brasileiro reagiu a mais um corte na taxa Selic, que foi reduzida para 11,75% ao ano pelo Banco Central, em meio ao cenário de desaceleração econômica e desinflação. Um recente relatório de emprego nos EUA ajudou a aliviar temores de uma aceleração econômica que poderia aumentar os preços ao consumidor. A perspectiva de queda nos juros nos EUA também esteve em destaque, o que animou os mercados emergentes, beneficiando moedas como o real e a Bolsa. (Ferreira , 2021)

4. O quarto ponto dia 14 de julho de 2022 os indicadores refletem a valorização do dólar no cenário internacional, impulsionada pela expectativa de aumento das taxas de juros pelo Federal Reserve dos EUA para combater a inflação, que atingiu seu nível mais alto em 40 anos em junho. Essa perspectiva de taxas de juros mais altas nos EUA, embora atraia investimentos para o país, também pode reduzir o apetite por risco global, aumentando os temores de recessão. (Do UOL, 2022)

5. No quinto e último ponto dia 28 de dezembro de 2023 a moeda norte-americana registrou uma queda em relação ao real, impulsionada pelo alívio nos temores fiscais domésticos e pelo diferencial de juros entre Brasil e Estados Unidos. O governo federal anunciando medidas para aumentar a arrecadação e alcançar a meta de déficit primário zero em 2024. O ministro da

Fazenda, também anunciou limitações nas compensações tributárias para decisões judiciais acima de R$ 10 milhões. (Gomes, 2023)

Gráfico 2 - Distribuição de fechamento diário da Ibovespa



Fonte: figuras criadas pelo próprio autor

O segundo gráfico é um histograma que mostra a distribuição dos preços de fechamento da Ibovespa. Ele é útil para visualizar a dispersão dos dados e identificar a concentração dos preços em torno de um valor central, bem como a presença de *outliers*.

Cada barra representa um intervalo de preços e a altura da barra indica a frequência com que os preços caem dentro desse intervalo. Isso ajuda a ter uma ideia da distribuição dos preços e se estão mais concentrados em determinadas faixas ou distribuídos de forma mais ampla.

Observa-se que os valores de fechamento sofreram variações diárias, como esperado, e com tendência de alta no período avaliado.

3.2. MATRIZ DE CORRELAÇÃO

A correlação é uma medida estatística que descreve a relação entre duas variáveis, varia de -1 a 1, onde:

- 1 indica uma correlação positiva perfeita: à medida que uma variável aumenta, a outra variável também aumenta na mesma proporção.
- -1 indica uma correlação negativa perfeita: à medida que uma variável aumenta, a outra variável diminui na mesma proporção.
- 0 indica que não há correlação linear entre as variáveis.

Como é possível ver na imagem abaixo a matriz de correlação mostra as correlações entre todas as variáveis em um conjunto de dados. E isso e útil porque:

Identifica relações entre variáveis: Permite identificar as duas variáveis estão relacionadas e qual é a direção dessa relação.

Seleção de variáveis: Ajuda a escolher as variáveis mais relevantes para análises mais aprofundadas, e eliminando variáveis altamente correlacionadas.

Diagnóstico de modelos: Pode ser usado para diagnosticar problemas de multicolinearidade em modelos estatísticos, onde variáveis independentes estão altamente correlacionadas.

Gráfico 3 – Matriz de correlação do índice Ibovespa

Fonte: figuras criadas pelo próprio autor

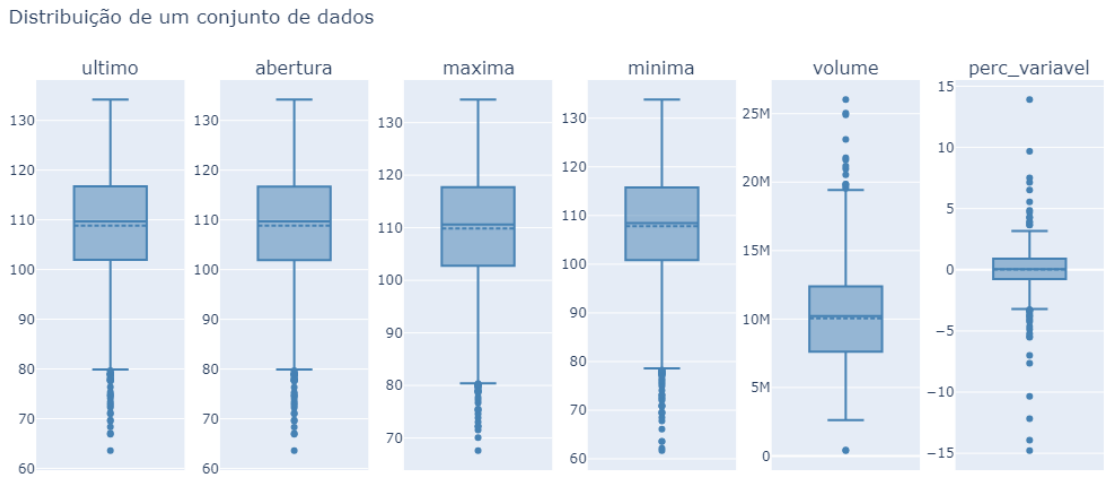
Em resumo, a matriz de correlação é uma ferramenta importante na análise de dados para entender as relações entre variáveis e tomar decisões informadas com base nessas relações. No gráfico podemos observar que as colunas *ultimo*, *abertura*, *maxima*, *minima* do nosso data frame possuem alta correlação enquanto a *volume*, *perc_variavel* uma baixa correlação com as demais.

3.3. OUTLIERS

Os outliers são pontos de dados que estão significativamente distantes do restante do conjunto de dados. Identificá-los é importante porque podem indicar erros nos dados, variações naturais extremas ou padrões interessantes que merecem investigação adicional. *Outliers* podem distorcer análises estatísticas e modelos preditivos, levando a conclusões incorretas ou imprecisas. Portanto, é essencial identificá-los e determinar se são resultados legítimos ou erros que precisam ser corrigidos.

O *box plots* é uma ferramenta gráfica útil para visualizar a distribuição de um conjunto de dados e identificar valores discrepantes (*outliers*). Eles fornecem uma representação compacta das principais estatísticas descritivas de um conjunto de dados, incluindo a mediana, quartis, intervalo interquartil (IQR) e potenciais *outliers*.

Gráfico 4 - Distribuição de um conjunto de dados (Outliers)



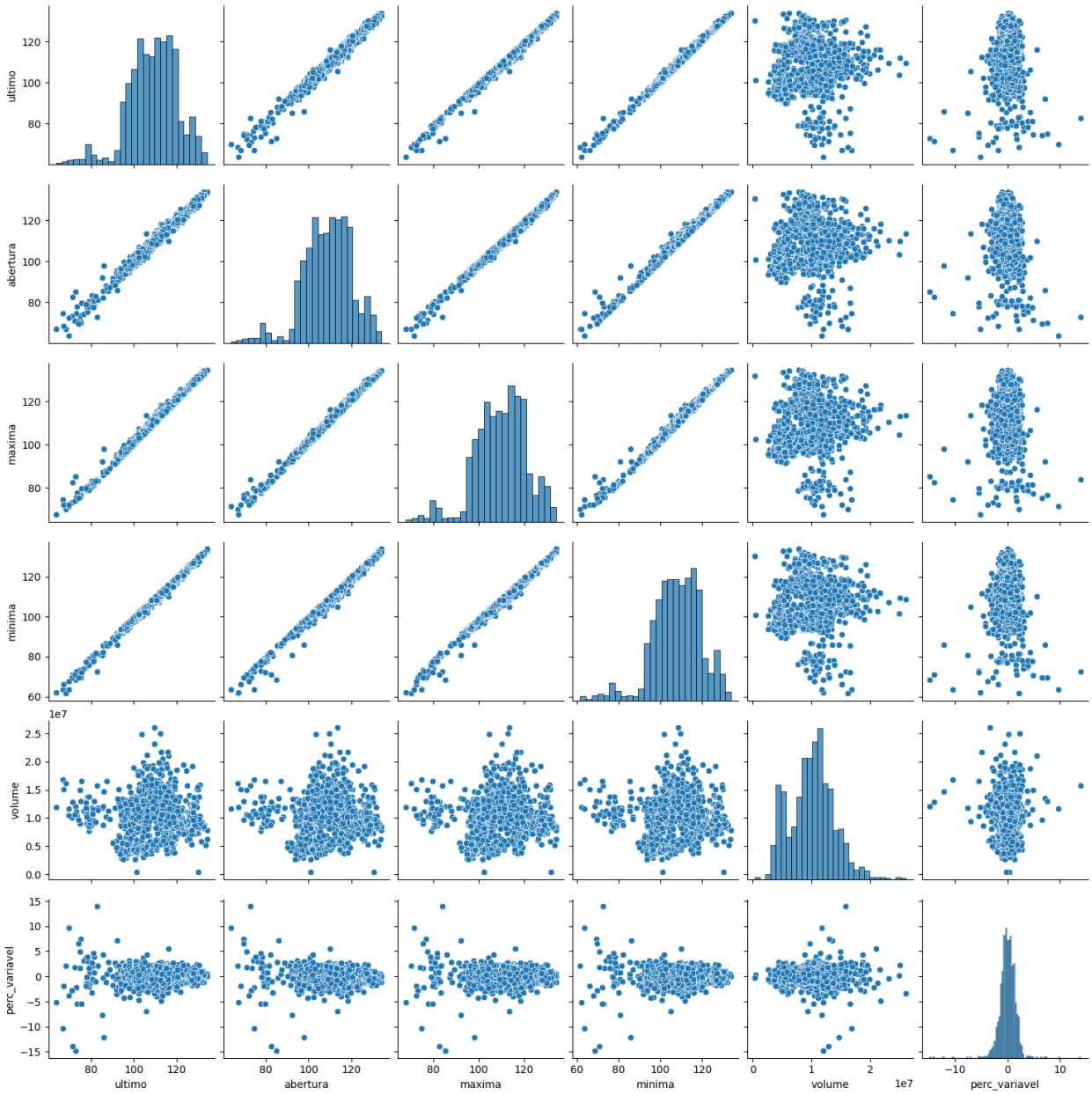
Fonte: figuras criadas pelo próprio autor

O gráfico de dispersão é uma ferramenta poderosa para visualizar as relações entre pares de variáveis em um *DataFrame*. Ele mostra a distribuição dos pontos de dados no plano cartesiano, onde cada ponto representa uma observação e suas coordenadas correspondem aos valores das variáveis. Essa visualização ajuda a identificar padrões, tendências e possíveis correlações entre as variáveis.

A importância do gráfico de dispersão está na capacidade de fornecer insights visuais sobre como as variáveis estão relacionadas. Por exemplo, ele pode mostrar se existe uma relação linear positiva, negativa ou nenhuma relação entre duas variáveis. Isso é crucial para entender o comportamento dos dados e

pode ajudar na tomada de decisões, na identificação de padrões ou anomalias e no desenvolvimento de modelos preditivos mais precisos.

Gráfico 5 - Matriz de dispersão



Fonte: figuras criadas pelo próprio autor

E assim como na matriz de correlação o gráfico mostra a alta correlação das colunas *ultimo, abertura, maxima, minima*, formando linhas bem parecidas, enquanto as colunas *volume*, *perc_variavel* aparenta uma baixa correlação com dados mais dispersos porém mais centralizados.

3.4. DECOMPOSIÇÃO SAZONAL

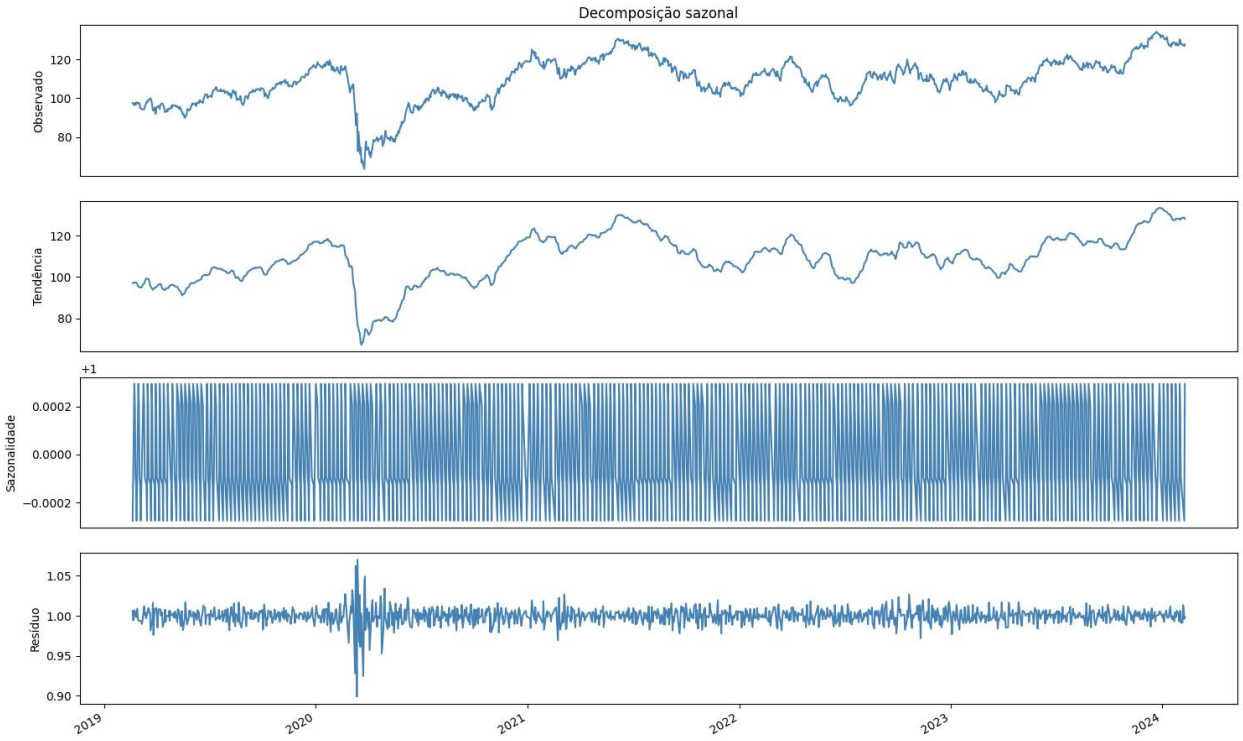
A decomposição sazonal é importante porque ajuda a entender a estrutura temporal dos dados, especialmente quando há variações periódicas relacionadas a estações, meses, dias da semana, entre outras coisas.

A decomposição sazonal é uma ferramenta útil que separa a série em diferentes componentes, como observado, tendência, sazonalidade e resíduos, permitindo visualizar como cada um desses componentes contribui para a série como um todo.

Isso pode ajudar a identificar padrões sazonais, avaliar a magnitude da sazonalidade e é uma técnica importante para analisar e modelar séries temporais, especialmente aquelas que exibem variações sazonais regulares.

E com isso ajustar modelos de previsão para considerar esses padrões de forma mais precisa.

Gráfico 6 - Decomposição sazonal



Fonte: figuras criadas pelo próprio autor

Ao analisar o *Gráfico 6* e possível ver que a tendência da série mostra uma semelhança com o próprio desenho da série original, indicando uma falta de tendência ou padrão claro em uma direção específica.

Já a sazonalidade se mostra uma forma complexa devido à natureza dos índices econômicos, que apresentam variações diárias, com padrões repetitivos.

Ao examinarmos os resíduos, notamos um ruído significativo no início da série, causado pela queda drástica no valor de fechamento do índice Bovespa em março de 2020, decorrente da pandemia de COVID-19.

3.5. SÉRIE ESTACIONÁRIA

A série estacionária é um conceito importante em estatística e análise de séries temporais. Uma série estacionária é aquela em que as propriedades estatísticas, como média, variância e autocorrelação, são constantes ao longo do tempo.

Verificar a série estacionária é importe porque muitos modelos estatísticos e técnicas de previsão de séries temporais assumem a estacionariedade dos dados. Quando uma série é estacionária, é mais fácil identificar padrões e fazer previsões precisas. Além disso, a estacionariedade muitas vezes simplifica a análise estatística, permitindo o uso de métodos mais diretos e eficazes.

Foi analisado a correlação entre 2 períodos constantes, utilizado o teste de *Dickey Fuller* para verificar se nossa série é estacionária ou não.

- H0 -> Série não estacionária (p-value > 0.05)
- H1 -> Série estacionária (p-value <= 0.05)

Para modelos ARIMA a estacionariedade dos dados é um requisito importante. Se os dados não são estacionários, é necessário aplicar diferenciação para torná-los estacionários antes de ajustar treinar o modelo. Resultado:

```
Teste de Dickey-Fuller Aumentado:
Teste ADF: -2.6231691221836604
p-valor: 0.08830260139751994
Valores críticos:
       1%: -3.4356646522289815
       5%: -2.863886926389418
       10%: -2.568019536239491
Conclusão: A série temporal provavelmente não é estacionária.
```

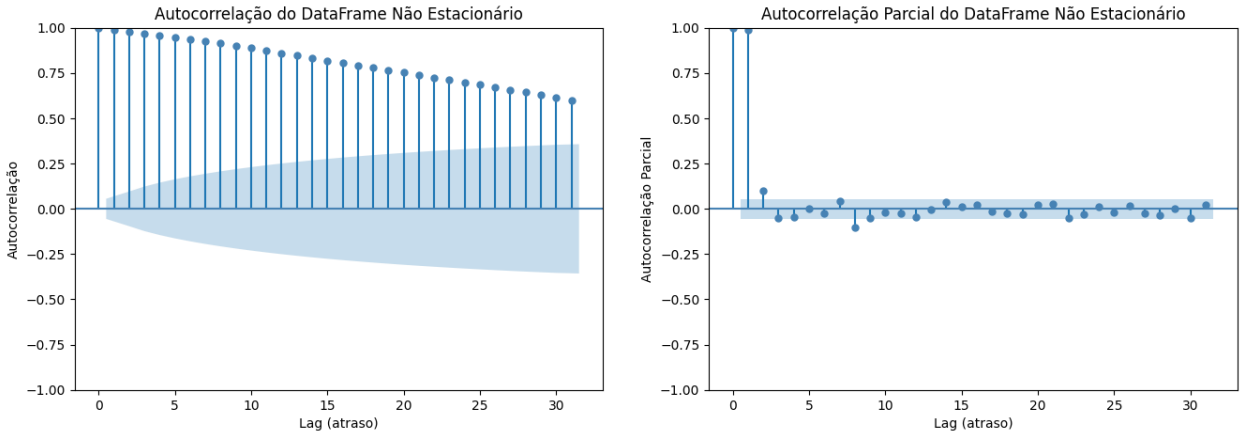
Neste caso, como o teste ADF sugere que a série temporal não é estacionária (p-valor maior que 0.05), então é preciso aplicar diferenciação para tornar a série estacionária antes de prosseguir com o modelo.

A diferenciação envolve subtrair os valores de um período dos valores do período anterior para remover tendências ou padrões de sazonalidade.

Depois de aplicar a diferenciação e obter uma série estacionária, e possível usar métodos como a análise da autocorrelação (ACF) e da autocorrelação parcial (PACF) para determinar os parâmetros p, d e q do modelo ARIMA. Estes representam a ordem da parte autoregressiva, a ordem de diferenciação e a ordem da média móvel, respectivamente.

Em resumo, a não estacionariedade da série temporal indica que é necessário aplicar diferenciação antes de modelar com ARIMA.

Gráfico 7 – Autocorrelação e Autocorrelação Parcial não estacionaria



Fonte: figuras criadas pelo próprio autor

3.6. DIFERENCIAÇÃO

Após a aplicação da diferenciação à série temporal, o teste ADF produziu um p-valor muito baixo, próximo de zero. Isso indica que, após a diferenciação, a série temporal se tornou estacionária.

Essencialmente, o que a diferenciação faz é subtrair o valor de um ponto de dados do valor do ponto de dados anterior. Isso pode ajudar a tornar a série mais "regular" em termos de comportamento ao longo do tempo, o que é uma característica de séries temporais estacionárias.

O fato de que o p-valor é muito baixo indica que há uma alta probabilidade estatística de que a série seja estacionária após a aplicação da diferenciação. Isso é uma boa notícia ao modelar a série temporal com técnicas como ARIMA, pois a estacionariedade é um requisito fundamental para esses modelos funcionarem adequadamente.

Antes de realizar a diferenciação foi utilizado o ***ndiffs*** para calcula o número de diferenciações não sazonais necessárias para tornar a série temporal estacionária.

O argumento ***test='adf'*** indica que o teste *Dickey-Fuller* Aumentado (ADF) deve ser usado para determinar a estacionariedade da série. O resultado desse código é a quantidade de diferenciações não sazonais sugeridas para tornar a série estacionária, com base no teste ADF.

Código 1 – Função para verificar a quantidade de diferenciações

```
# O número de diferenciações não sazonais sugeridas para tornar a série estacionária
ndiffs(df.ultimo, test='adf')

output
1
```

Fonte: trecho de código criado pelo próprio autor

Após isso foi realismo a diferenciação para o *DataFrame* e o resultado foi:

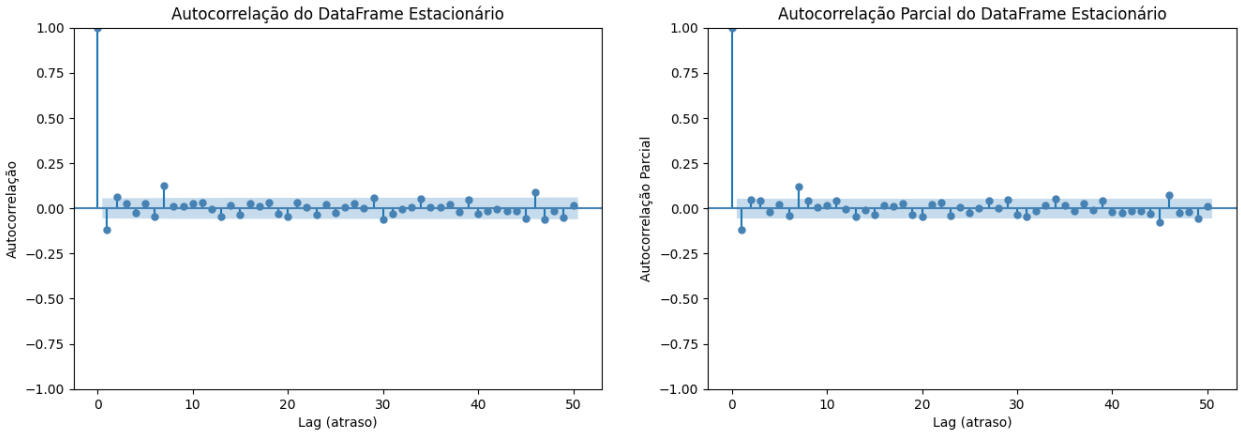
```
Teste de Dickey-Fuller Aumentado:
Teste ADF: -10.806515933553417
p-valor: 1.9570324233203174e-19
Valores críticos:
1%: -3.4356646522289815
5%: -2.863886926389418
10%: -2.568019536239491
Conclusão: A série temporal provavelmente é estacionária.
```

Conclusão: A série temporal provavelmente é estacionária, pois o p-valor (1.9570324233203174e-19) é menor que 0.05 (um nível comum de significância estatística). Isso sugere que há evidências suficientes para rejeitar a hipótese nula de que a série possui raiz unitária e, portanto, é estacionária após a diferenciação.

Para visualizar a diferenciação aplicada à série temporal, foi criado um gráfico que inclui a média móvel e o desvio padrão. A média móvel suaviza a série, destacando tendências de longo prazo, enquanto o desvio padrão mostra a variabilidade dos dados ao longo do tempo. Juntos, esses elementos ajudam a visualizar melhor as características da série diferenciada.

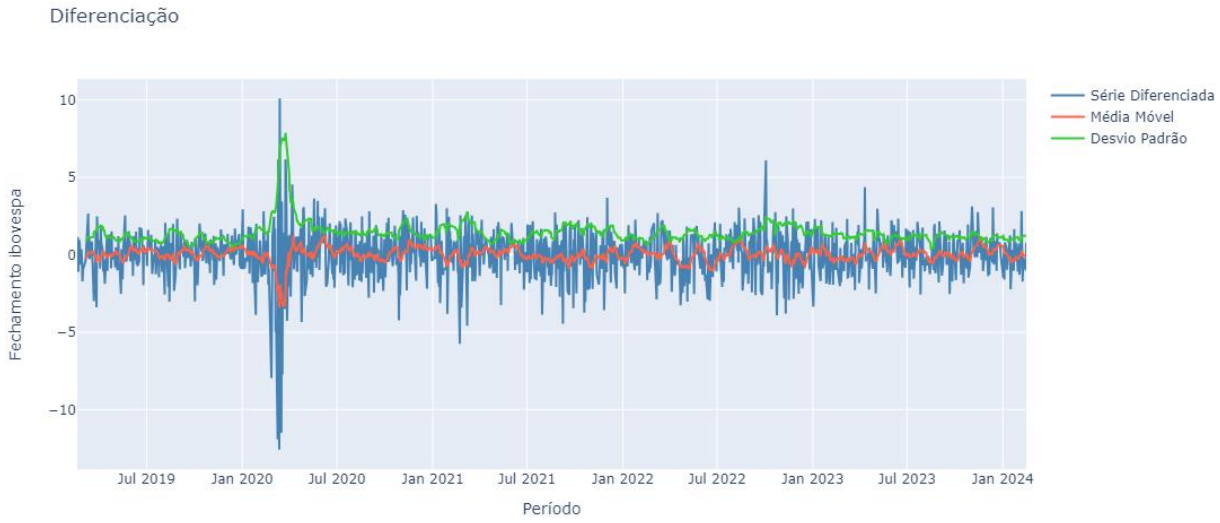
O gráfico de autocorrelação é uma ferramenta poderosa na análise de séries temporais, ajudando a entender a estrutura temporal dos dados e a selecionar e validar modelos adequados.

Gráfico 8 - Autocorrelação e Autocorrelação Parcial Estacionário



Fonte: figuras criadas pelo próprio auto

Gráfico 9 – Visualização da diferenciação



Fonte: figuras criadas pelo próprio autor

4. MÉTRICAS

Antes de explicar os processos dos modelos utilizados neste relatório, é importante mencionar as métricas utilizadas para avaliar a qualidade das previsões. As métricas escolhidas foram o Erro Quadrático Médio (MSE), Erro Absoluto Médio (MAE), Coeficiente de Determinação (R^2) e Erro Percentual Absoluto Médio (MAPE).

MSE: *Mean Squared Error* (Erro Quadrático Médio) é a média dos quadrados dos erros entre os valores observados e os valores previstos pelo modelo. É uma medida que penaliza mais fortemente os erros maiores, ou seja, quanto maior o erro, maior será sua contribuição para o MSE.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MAE: *Mean Absolute Error* (Erro Absoluto Médio) é a média dos valores absolutos dos erros entre os valores observados e os valores previstos pelo modelo. É uma medida que não considera a direção dos erros, ou seja, erros positivos e negativos têm o mesmo peso.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

R^2:*R-squared* (Coeficiente de Determinação) é uma medida que indica a proporção da variância dos valores observados que é explicada pelo modelo. Ele varia de 0 a 1, onde 0 indica que o modelo não explica nenhuma variabilidade dos dados e 1 indica que o modelo explica toda a variabilidade.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

MAPE: *Mean Absolute Percentage Error* (Erro Percentual Absoluto Médio) é a média dos valores absolutos dos erros percentuais entre os valores observados e os valores previstos pelo modelo. É uma medida útil para avaliar a precisão das previsões em termos percentuais.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

Essas métricas são importantes para avaliar o desempenho do modelo de previsão e entender o quão bem ele está se ajustando aos dados observados.

5. MODELOS

A escolha dos modelos para previsão foi baseada em critérios como robustez, adaptabilidade, eficiência computacional, capacidade de generalização, interpretação dos resultados e precisão da previsão. Isso porque diferentes modelos têm diferentes capacidades de lidar com a complexidade dos dados e capturar padrões relevantes.

Os modelos escolhidos foram:

ARIMA (*Autoregressive Integrated Moving Average*): O ARIMA é um modelo clássico de séries temporais que leva em consideração a autocorrelação e a sazonalidade dos dados. Ele é eficaz para prever séries temporais que exibem padrões de autocorrelação e tendências.

Prophet: O Prophet é um modelo de previsão desenvolvido pelo Facebook que é capaz de lidar com sazonalidade, feriados e tendências de longo prazo. Ele é simples de usar e interpretar, tornando-o uma escolha popular para previsões de séries temporais com dados sazonais.

LSTM (*Long Short-Term Memory*): é um tipo de rede neural recorrente (RNN) capaz de aprender dependências de longo prazo em sequências de dados. Ele é especialmente útil em tarefas de previsão de séries temporais, como prever o preço de uma ação, pois consegue capturar padrões complexos e de longo prazo nos dados.

Ridge Regression: é uma técnica de regressão linear regularizada que adiciona um termo de regularização à função de custo do modelo, ajudando a evitar o *overfitting* e a lidar com multicolinearidade nos dados.

5.1. ARIMA

ARIMA (*Autoregressive Integrated Moving Average*) é um modelo estatístico utilizado para analisar e prever séries temporais. Ele combina a ideia de regressão linear (parte auto regressiva), médias móveis (parte de médias móveis) e diferenciação de séries temporais (parte integrada) em um único modelo.

O modelo ARIMA é frequentemente representado como ARIMA(p, d, q), onde:

- "p" é a ordem do componente auto regressivo;
- "d" é a ordem de diferenciação;
- "q" é a ordem do componente de médias móveis.

O ARIMA é um modelo poderoso para prever séries temporais, especialmente quando a série exibe padrões complexos de autocorrelação e sazonalidade. No entanto, é importante ajustar adequadamente os parâmetros do modelo e avaliar sua precisão para obter resultados confiáveis.

Antes de criar o modelo ARIMA foi utilizada a função ***auto_arima*** da biblioteca ***pmdarima***. Essa função ajustar um modelo ARIMA automaticamente aos dados passados, buscando os melhores parâmetros. E para isso foi utilizado os seguintes argumentos:

- **d=1:** O número de diferenciações não sazonais.
- **start_p=1, start_q=1:** Valores iniciais para os parâmetros p e q.
- **max_p=3, max_q=3:** Valores máximos para os parâmetros p e q.
- **seasonal=True, m=6:** Considera sazonalidade com um período sazonal de 6.
- **D=1, start_P=1, start_Q=1:** Valores iniciais para os parâmetros sazonais P e Q.
- **max_P=2, max_Q=2:** Valores máximos para os parâmetros sazonais P e Q.
- **information_criterion='aic':** Critério de informação para selecionar o melhor modelo, utilizado o Critério de Informação de Akaike (AIC) é uma medida estatística que quantifica a qualidade relativa de um modelo estatístico para um conjunto dado de dados.
- **trace=True:** Mostra informações detalhadas sobre o processo de ajuste.
- **error_action='ignore':** Ignora erros durante o ajuste.
- **stepwise=True:** Usa um método stepwise para encontrar os melhores parâmetros.

O ***auto_arima*** irá ajustar vários modelos ARIMA com diferentes combinações de parâmetros e selecionar o melhor modelo com base no critério de informação especificado. Isso ajuda a encontrar automaticamente os melhores parâmetros para o modelo ARIMA sem a necessidade de ajuste manual.

Código 2 – Função para buscar melhores parâmetros para o modelo ARIMA

```
# Auto ARIMA nos ajuda a buscar os melhores parâmetros
fit_arima = auto_arima(df, d=1, start_p=1, start_q=1, max_p=3, max_q=3, seasonal=True, m=6, D=1, start_P=1,start_Q=1, max_P=2, max_Q=2, information_criterion='aic', trace=True, error_action='ignore', stepwise=True)

output
Performing stepwise search to minimize aic
Best model:  ARIMA(3,1,0)(2,1,0)[6]
Total fit time: 28.588 seconds
```

Fonte: trecho de código criado pelo próprio autor

Nesse trecho de código, a variável X recebe os dados da série temporal que se deseja prever, que no caso são os valores de fechamento da bolsa de valores (coluna 'ultimo' do *DataFrame* df). Em seguida, os dados são convertidos para o tipo *'float32'* para garantir que estão no formato adequado para serem utilizados no modelo.

Código 3 – Separa os dados com 70% para treino e 30% para teste para o modelo ARIMA

```
# a variável X recebe os dados da série
X = df.ultimo
X = X.astype('float32')

# Separa os dados com 70% dos dados para treino e 30% dos dados para teste
size = int(len(X) * 0.7)
train = X [0:size]
test = X [size:]
```

Fonte: trecho de código criado pelo próprio autor

E necessário separar os dados em conjuntos de treinamento e teste. Para isso, calcula-se o tamanho do conjunto de treinamento como 70% do tamanho total dos dados e o tamanho do conjunto de teste como os 30% restantes.

Um dos requisitos para modelos como ARIMA como explicado anteriormente e a estacionariedade, isso simplifica a modelagem e a previsão, pois permite que as propriedades estatísticas da série sejam consideradas estáveis ao longo do tempo.

Para este caso foram criadas duas funções a primeira ***difference*** implementa a diferenciação de uma série temporal. Isso pode ser útil para remover tendências ou padrões de sazonalidade da série. Como destacado nos gráficos de correlação mostrando a diferenciação nos dados

Por outro lado, a função ***inverse_difference*** reverte o efeito da diferenciação. Ela aceita a série original antes da diferenciação (***history***), o valor diferenciado a ser revertido (***previsao***), e o intervalo de diferenciação. Essa função é útil para converter os valores diferenciados de volta para a escala original da série, o que é necessário para interpretar e avaliar as previsões feitas pelo modelo ARIMA, por exemplo.

Código 4 – Cria a variável history

```
# cria a variável history
history = [x for x in train]
```

Fonte: trecho de código criado pelo próprio autor

Código 5 - Cria a variável previsões

```
# cria lista de previsões
predictions = list()
```

Fonte: trecho de código criado pelo próprio autor

Código 6 – Função que faz a diferenciação e função que reverte o valor diferenciado

```
# Cria a função que faz a diferenciação
def difference(dataset, interval=1):
  diff = list()
  for i in range(interval, len(dataset)):
    value = dataset[i] - dataset[i - interval]
    diff.append(value)
  return diff

# cria função que reverte o valor diferenciado para o original
def inverse_difference(history, previsao, interval=1):
  return previsao + history[-interval]
```

Fonte: trecho de código criado pelo próprio autor

Neste trecho de código, está sendo realizado um procedimento conhecido como *Walk-Forward Validation* (ou validação progressiva). Esse método é comumente utilizado em séries temporais para avaliar a performance de modelos de previsão, como o ARIMA (*Autoregressive Integrated Moving Average*).

Código 7 – Walk-Forward Validation

```
for t in range(len(test)):

  # difference data
  days = 10
```

```
    diff = difference(history, days)

    # cria um modelo ARIMA com os dados de history
    model = ARIMA(diff, order=(3,1,0))

    # treina o modelo ARIMA
    model_fit = model.fit()

    # a variável valor_predito recebe o valor previsto pelo modelo
    valor_predito = model_fit.forecast()[0]

    # valor_predito recebe o valor revertido (escala original)
    valor_predito = inverse_difference(history, valor_predito, days)

    # adiciona o valor predito na lista de predicões
    predictions.append(valor_predito)

    # a variável valor_real recebe o valor real do teste
    valor_real = test[t]

    # adiciona o valor real a variável history
    history.append(valor_real)

    # imprime valor predito e valor real
    print('Valor predito=%.3f, Valor esperado=%3.f' % (valor_predito, valor_real))
```

Fonte: trecho de código criado pelo próprio autor

Esse processo se repete para cada período de tempo no conjunto de teste, permitindo avaliar o desempenho do modelo ARIMA ao longo do tempo e verificar sua capacidade de fazer previsões precisas.

O *Gráfico 10* mostra a previsão feita pelo modelo ARIMA para a série temporal em questão. Ele representa a comparação entre os valores reais (observados) e os valores previstos pelo modelo para o período de teste.

Gráfico 10 – Previsão modelo ARIMA

Fonte: figuras criadas pelo próprio autor

5.1.1. MÉTRICAS DO MODELO ARIMA

E para o modelo ARIMA esses foram os resultados das métricas:

- Erro Quadrático Médio (MSE) = 3.82
- Erro Absoluto Médio (MAE) = 1.53
- Coeficiente de Determinação (R^2) = 0.94
- Erro Percentual Absoluto Médio (MAPE) = 1.35%

5.2. MODELO PROPHET

O Prophet é uma biblioteca de código aberto desenvolvida pelo *Facebook* (Meta) para previsão de séries temporais. Ele foi projetado para ser fácil de usar, mesmo por pessoas sem experiência avançada em ciência de dados ou previsão de séries temporais. Algumas características e diferenciais do modelo Prophet foi projetado para ser simples de implementar e usar, com uma API intuitiva que permite aos usuários criar modelos de previsão de séries temporais com poucas linhas de código.

Também é capaz de lidar automaticamente com diferentes tipos de sazonalidade, incluindo sazonalidade diária, semanal e anual, o que o torna adequado para uma ampla gama de aplicações. Um modelo robusto

em relação a dados ausentes e outliers, podendo lidar com eles de forma eficaz durante o processo de modelagem.

Permite a inclusão de datas de feriados como variáveis explicativas, o que pode melhorar a precisão das previsões em datas próximas aos feriados. E possui uma abordagem aditiva para modelar tendências sazonais e de longo prazo, o que pode ser mais adequado para séries temporais que exibem padrões não lineares.

No geral, o Prophet é uma ferramenta poderosa e acessível para previsão de séries temporais, sendo amplamente utilizado em diversas áreas, como finanças, *marketing*, *e-commerce* e meteorologia, entre outras.

Para que o processo de precisão funcionasse no modelo Prophet foram realizados os seguintes ajustes:

Código 8 – Ajuste de dados para o modelo Prophet

```
# Resetar o index
prophet_bovespa = prophet_b.reset_index()
# Ajustar o nome da da colunas data, ultimo para ds, y
prophet_bovespa[['ds','y']] = prophet_bovespa[['data', 'ultimo']]
# Retirar as demais colunas
prophet_bovespa.drop(columns=['data', 'ultimo', 'volume', 'perc_variavel'], axis=1,
inplace=True)
```

Fonte: trecho de código criado pelo próprio autor

Resetar o índice: O índice do *DataFrame* é redefinido para permitir manipulações posteriores.

Ajustar as colunas: As colunas *'data'* e *'ultimo'* são renomeadas para *'ds'* e *'y'*, respectivamente, para seguir a convenção do Prophet.

Remover colunas: As colunas, *'volume'* e *'perc_variavel'* são removidas, pois não são necessárias para o modelo Prophet.

Código 9 – Separa os dados com 80% para treino e 20% para teste para o modelo Prophet

```
train_prophet = prophet_bovespa.sample(frac=0.8, random_state=0)
test_prophet = prophet_bovespa.drop(train_prophet.index)
print(f'train: {train_prophet.shape}')
print(f'test: {test_prophet.shape}')

output
train: (994, 5)
test: (248, 5)
```

Fonte: trecho de código criado pelo próprio autor

Divisão entre treino e teste: Os dados são divididos em conjuntos de treinamento (80%) e teste (20%).

Código 10 – Instanciar o modelo, treinando e previsão 120 dias (4 meses)

```
# Instanciando o Modelo
prophet = Prophet(daily_seasonality=True)
# Treinando o Modelo
prophet.fit(train_prophet)

# Realizando a Previsão 120 dias (4 meses)
df_prev_prophet = prophet.make_future_dataframe(periods=120, freq='D')
prev_prophet = prophet.predict(df_prev_prophet)
```

Fonte: trecho de código criado pelo próprio autor

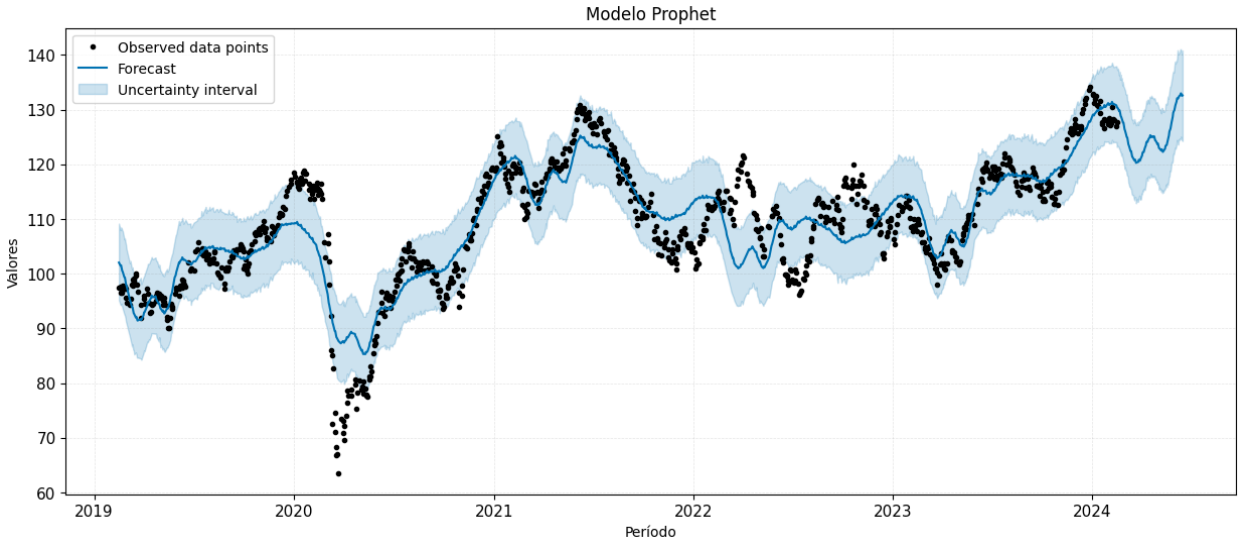
Instanciar o Modelo: Um objeto Prophet é criado com a opção de sazonalidade diária.

Treinar o Modelo: O modelo Prophet é treinado com os dados de treinamento.

Realizar a Previsão: É gerado um DataFrame com datas futuras para previsão e então é feita a previsão com o modelo treinado.

Visualização dos Resultados: São plotados gráficos mostrando a previsão gerada pelo modelo e suas componentes sazonais.

Gráfico 11 – Previsão do modelo Prophets



Fonte: figuras criadas pelo próprio autor

O **plot_components** gera um conjunto de gráficos que mostram as principais componentes do modelo Prophet aplicado aos dados. Esses gráficos são úteis para visualizar como diferentes fatores, como tendência e sazonalidade, contribuem para as previsões. Foram criados os seguintes componentes:

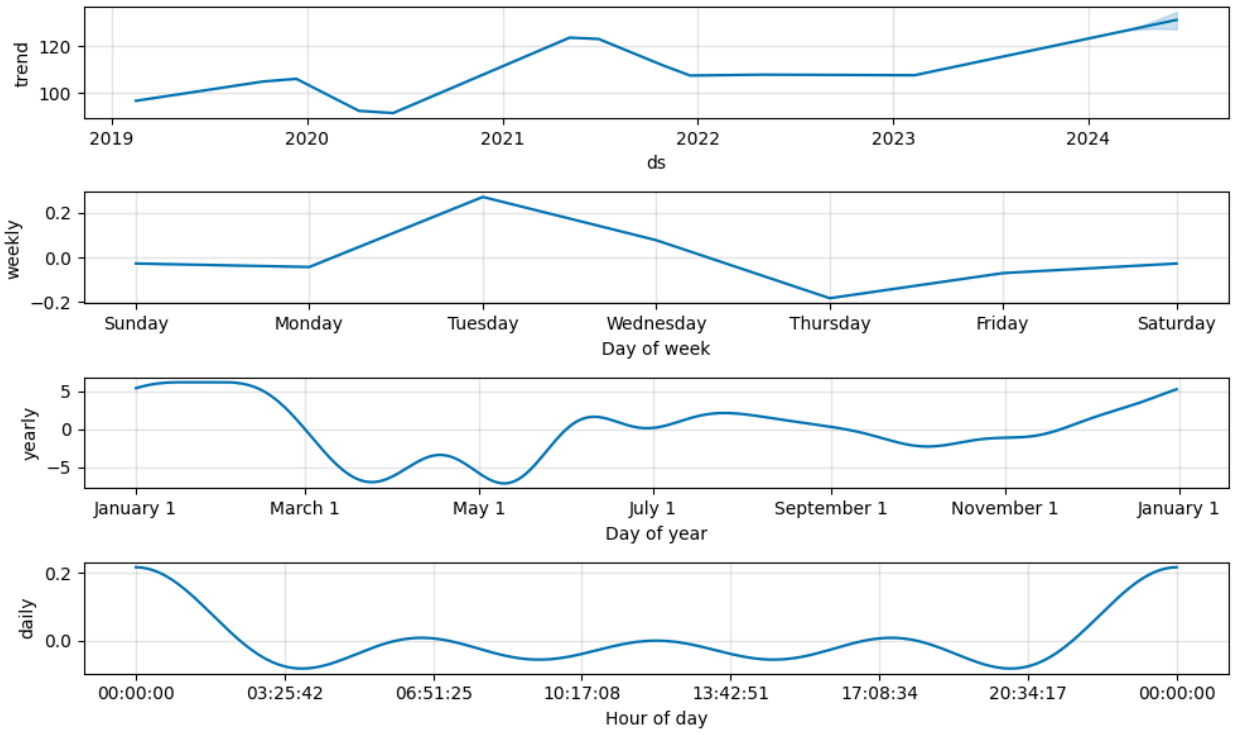
trend (tendência): representa a direção geral dos dados ao longo do tempo, indicando se os valores estão aumentando, diminuindo ou permanecendo estáveis em uma determinada tendência.

weekly (sazonalidade semanal): captura padrões sazonais que se repetem a cada semana, como variações nos dados que ocorrem em determinados dias da semana.

yearly (sazonalidade anual): modela variações sazonais que ocorrem em intervalos de um ano, como picos de vendas durante as festas de fim de ano ou quedas devido a férias prolongadas.

daily (sazonalidade diária): representa padrões sazonais que se repetem a cada dia, como variações nos dados ao longo do dia, como horários de pico de tráfego, vendas ou atividades.

Gráfico 12 – Conjunto de gráficos, trend, weekly, yearly, daily



Fonte: figuras criadas pelo próprio autor

5.2.1. MÉTRICAS DO MODELO PROPHETS

Avaliar o modelo, calculando as métricas de erro, como MSE, MAE, R^2 e MAPE, para avaliar a qualidade da previsão. Resultado das métricas são:

- Erro Quadrático Médio (MSE) = 29.80
- Erro Absoluto Médio (MAE) = 4.14
- Coeficiente de Determinação (R^2) = 0.78
- Erro Percentual Absoluto Médio (MAPE) = 3.97%

5.3. MODELO LSTM

As redes neurais LSTM (*Long Short-Term Memory*) são um tipo especial de rede neural recorrente (RNN) projetada para lidar com problemas de sequência e memória de longo prazo. Elas são especialmente úteis

em tarefas em que a dependência de eventos passados é crucial para a previsão ou classificação de eventos futuros.

A principal vantagem das LSTMs sobre as RNNs tradicionais é sua capacidade de manter e atualizar informações ao longo do tempo, o que ajuda a evitar problemas com o gradiente que surgem em RNNs mais simples. Isso é conseguido através de unidades de memória especializadas chamadas "células de memória" e de estruturas de portas que controlam o fluxo de informações.

Utilizando o modelo LSTM para prever os preços de fechamento de ações. Aqui está uma explicação passo a passo do que foi feito:

No início os dados de fechamento ('ultimo') são extraídos e transformados em um *array*, e foi separado, representando 80% dos dados para treinamento.

Código 11 – Preparação de dados para modelo

```
#transformar em array
close = lstm_bovespa['ultimo'].to_numpy().reshape(-1,1)

# separar 80% para treinamento
train_close= int(len(close)*0.8)
train_close
```

Fonte: trecho de código criado pelo próprio autor

Nesse trecho de código, é utilizado o ***MinMaxScaler*** da biblioteca ***sklearn.preprocessing*** para realizar a normalização dos dados. A normalização é um processo comum em modelos de *machine learning* para padronizar as variáveis de entrada em uma faixa específica, geralmente entre 0 e 1, o que ajuda no desempenho do modelo.

Código 12 – Normalizar os dados

```
# Escalar os dados entre 0 e 1 para processamento
scaler = MinMaxScaler(feature_range=(0, 1))
scaler_train = scaler.fit_transform(close[0: train_close, :])
scaler_test = scaler.transform(close[train_close:,:])

scaled_data = list(scaler_train.reshape(len(scaler_train))) +
list(scaler_test.reshape(len(scaler_test)))
scaled_data = np.array(scaled_data).reshape(len(scaled_data),1)
scaled_data
```

Fonte: trecho de código criado pelo próprio autor

Com os dados normalizados (***scaled_data***) são divididos em conjuntos de treinamento (***X_train***, ***y_train***) para o modelo LSTM

Código 13 – Divisão em conjunto de treinamento (X_train, y_train)

```
# Para o treinamento sera visto 30 dias anteriores
train_data = scaled_data[0: train_close,:]

X_train = []
y_train = []

for i in range(30, len(train_data)):
  X_train.append(train_data[i - 30:i, 0])
  y_train.append(train_data[i, 0])

  if i <= 31:
    print(X_train)
    print(y_train)
```

Fonte: trecho de código criado pelo próprio autor

Antes da construção do modelo os dados de treinamento (***X_train*** e ***y_train***) são reformulados para o formato 3D, que é o formato esperado pelo modelo LSTM

Código 14 – Reformulados para o formato 3D

```
# Reformular os dados para o formato 3d
X_train, y_train = np.array(X_train), np.array(y_train)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
```

Fonte: trecho de código criado pelo próprio autor

Construir modelo LSTM utilizando a biblioteca *Keras* com *TensorFlow* como *backend*:

- ***model = Sequential()*** cria um modelo sequencial, onde as camadas são empilhadas uma após a outra.
- ***model.add(LSTM(units=30, return_sequences=True, input_shape=(X_train.shape[1], 1)))*** adiciona a primeira camada LSTM com 30 unidades de memória, configurada para retornar

sequências completas ***(return_sequences=True)*** e definindo a forma de entrada como o formato 3D dos dados de treinamento ***(input_shape=(X_train.shape[1], 1)).***

- ***model.add(LSTM(units=30, return_sequences=True))*** adiciona uma segunda camada LSTM com 30 unidades de memória e configurada para retornar sequências completas.
- ***model.add(LSTM(units=30))*** adiciona uma terceira camada LSTM com 30 unidades de memória, que por padrão não retorna sequências completas, pois será a última camada LSTM.
- ***model.add(Dense(10))*** adiciona uma camada densa com 10 neurônios.
- ***model.add(Dense(1))*** adiciona a camada de saída com 1 neurônio, que prevê o valor do preço de fechamento.
- ***model.compile(optimizer='adam', loss='mean_squared_error')*** compila o modelo, configurando o otimizador 'adam' e a função de perda 'mean_squared_error'.
- ***model.summary()*** exibe um resumo do modelo, mostrando as camadas, o número de parâmetros e a forma de saída de cada camada.

Código 15 – Construir o modelo LSTM

```
# Construir o modelo LSTM
model = Sequential()
model.add(LSTM(units=30, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(units=30, return_sequences=True))
model.add(LSTM(units=30))
model.add(Dense(10))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()
```

Fonte: trecho de código criado pelo próprio autor

Treinamento do modelo com os dados de entrada ***X_train*** e os rótulos ***y_train*** por 30 épocas, onde uma época é uma passagem pelo conjunto de dados completo.

- O parâmetro ***batch_size=10*** especifica o tamanho do lote de dados utilizado para atualizar os pesos do modelo a cada iteração.
- O parâmetro ***verbose=2*** controla o nível de detalhes das mensagens de treinamento exibidas durante o processo. Um valor de 2 significa que será exibida uma linha por época de treinamento, mostrando o número da época e a métrica de perda.

Código 16 – Treinamento do modelo LSTM

```
# Treinar o modelo
train_model = model.fit(X_train, y_train, epochs=30, batch_size=10, verbose= 2)
```

Fonte: trecho de código criado pelo próprio autor

Os dados de teste são preparados para serem usados e convertidos em um array numpy e remodelados para o formato 3D, a fim de serem compatíveis com o modelo LSTM

Código 17 – Preparação dos dados para teste e converter para array

```
test_data = scaled_data[train_close - 30:, :]

X_test = []
y_test = close[train_close:,:]

for i in range(30, len(test_data)):
  X_test.append(test_data[i - 30: i, 0])

X_test = np.array(X_test)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

Fonte: trecho de código criado pelo próprio autor

Usando o modelo LSTM para fazer previsões com os dados de teste X_test

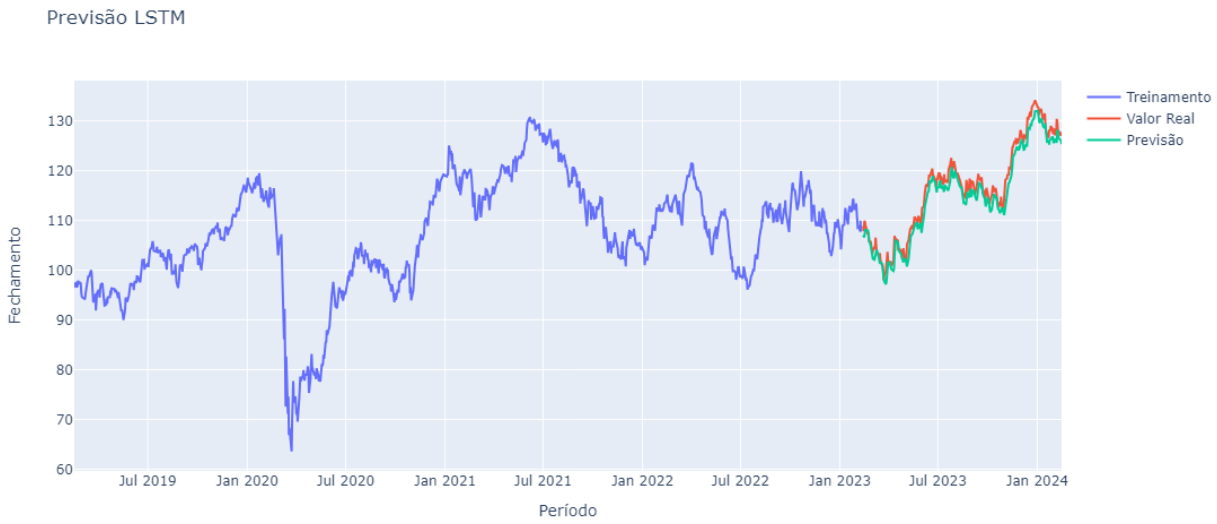
Código 18 – Criar predição LSTM

```
# Criar a predição
prev_lstm = model.predict(X_test)

# Inverter os dados da predição
prev_lstm = scaler.inverse_transform(prev_lstm)

prev_lstm
```

Fonte: trecho de código criado pelo próprio autor

Com isso a variável ***prev_lstm*** agora contém as previsões do modelo LSTM para os dados de teste, prontos para serem comparados com os valores reais.

Gráfico 13 – Previsão modelo LSTM



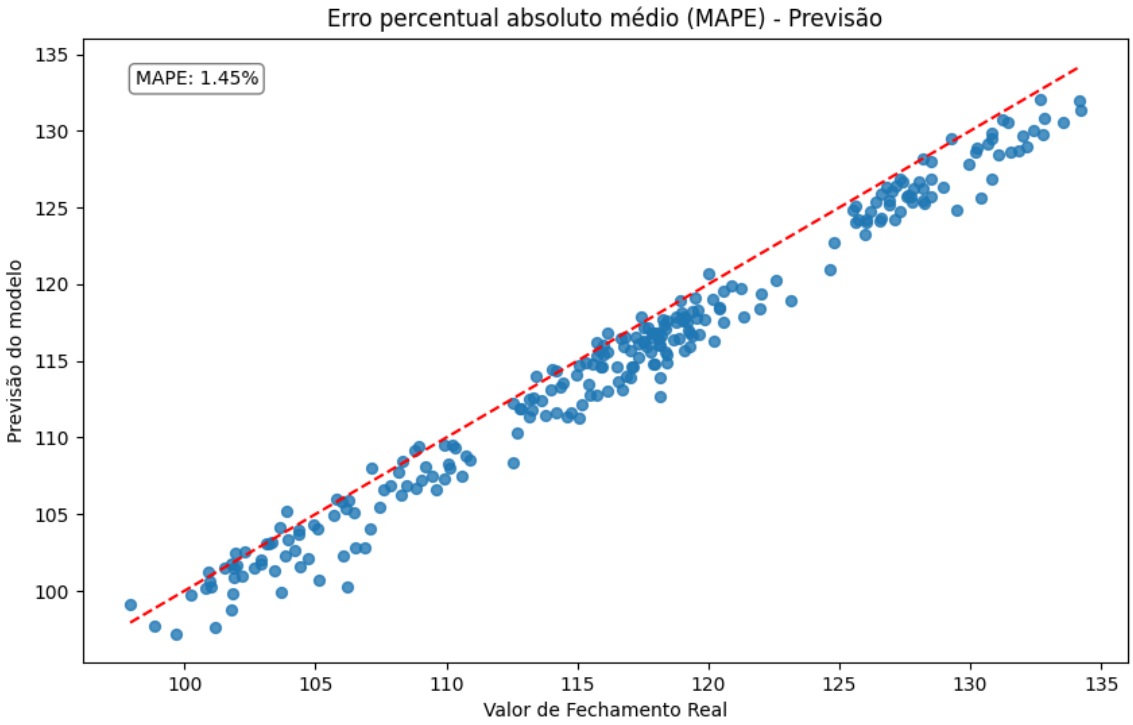
Fonte: figuras criadas pelo próprio autor

O *Gráfico 13* mostra a previsão feita pelo modelo LSTM representa a comparação entre os valores reais e os valores previstos pelo modelo para o período de teste. E é possível observar que a previsão do modelo está bem próxima dos valores reais. Também possível visualizar no *Gráfico 14*.

Gráfico 14 – Previsão aproximada

Fonte: figuras criadas pelo próprio autor

Gráfico 15 - Erro percentual absoluto médio (MAPE) – Previsão LSTM



Fonte: figuras criadas pelo próprio autor

O erro percentual absoluto médio (MAPE) de 1.51% indica que, em média, as previsões do modelo estão muito próximas dos valores reais, sugerindo que o modelo LSTM é eficaz na previsão de preços de ações.

5.3.1. MÉTRICAS DO MODELO LSTM

Avaliar o modelo, calculando as métricas de erro, como MSE, MAE, R^2 e MAPE, para avaliar a qualidade da previsão. Resultado das métricas são:

- Erro Quadrático Médio (MSE) = 4.25
- Erro Absoluto Médio (MAE) = 1.70
- Coeficiente de Determinação (R^2) = 0.95
- Erro Percentual Absoluto Médio (MAPE) = 1.45%

5.4. MODELO RIDGE REGRESSION

A Ridge Regression é um método de regressão linear regularizado que inclui um termo de regularização L2 na função de custo do modelo. Essa técnica é eficaz para evitar o *overfitting*, pois penaliza os coeficientes da regressão, forçando-os a permanecer pequenos.

Essa abordagem é especialmente útil quando há multicolinearidade nos dados, o que significa alta correlação entre as variáveis independentes. Nesses casos, a Ridge Regression reduz a variância dos coeficientes, tornando o modelo menos sensível a pequenas alterações nos dados de entrada. Além disso, a Ridge Regression pode ser uma ferramenta valiosa para lidar com o *overfitting* em modelos de regressão linear que possuem muitos atributos (variáveis independentes).

Seleção de Features: As colunas *'abertura'*, *'máxima'*, *'mínima'*, *'volume'* e *'perc_variavel' são* removidas do *DataFrame*, mantendo apenas a coluna *'último'* (valor de fechamento).

Criação do Target: A coluna *'target'* é criada, deslocando os valores da coluna *'último'* uma posição para frente, para que cada linha contenha o valor de fechamento do dia seguinte.

Criação de Médias Móveis: São criadas duas novas *features* (*'mm7d'* e *'mm30d'*) que representam as médias móveis de 7 e 30 dias, respectivamente, do valor de fechamento.

Remoção de Valores Nulos: As linhas com valores nulos são removidas.

Código 19 – Ajuste de dados para o modelo Ridge Regression

```
ridge_bovespa.drop(columns=['abertura','maxima','minima', 'volume','perc_variavel'],
inplace = True)

# Empurrando para frente os valores de fechamento
ridge_bovespa['target'] = ridge_bovespa['ultimo'].shift(-1)

# Criando campos de médias móveis para acrescentar mais features ao modelo
ridge_bovespa['mm7d'] = ridge_bovespa.ultimo.rolling(7).mean()
ridge_bovespa['mm30d'] = ridge_bovespa.ultimo.rolling(30).mean()

# retirando os dados nulos
ridge_bovespa.dropna(inplace=True)
# reset de index
ridge_bovespa.reset_index(inplace =True)
```

Fonte: trecho de código criado pelo próprio autor

Separação dos Dados: Os dados são separados em variáveis de entrada (X) e variável de saída (y).

Divisão em Conjuntos de Treinamento e Teste: Os dados são divididos em conjuntos de treinamento (70%) e teste (30%).

Código 20 – Separa os dados com 70% para treino e 30% para teste, modelo Ridge Regression

```
# Separar os dados em variáveis de entrada (X) e variável de saída (y)
X = ridge_bovespa.drop(columns=['data','target', 'mm30d'])
y = ridge_bovespa['target']

# Dividir os dados em conjuntos de treinamento e teste test_size=0.3 indica que 30% dos dados serão usados como conjunto de teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

Fonte: trecho de código criado pelo próprio autor

Normalização dos Dados: Os dados são normalizados usando o MinMaxScaler.

Código 21 – Normalização de dados

```
# Gerando o novo padrão
scaler = MinMaxScaler(feature_range= (0,1))
scaled_df = scaler.fit_transform(ridge_bovespa.drop(columns='data'))
scaled_df
```

Fonte: trecho de código criado pelo próprio autor

Criação e Treinamento do Modelo: Um modelo de regressão Ridge é criado e treinado com os dados de treinamento.

Código 22 – Treinamento e previsão

```
# Crie um modelo de regressão Ridge com um valor alfa de 1
rg= Ridge(alpha=1.0)

# Ajustar o modelo aos dados de treinamento
rg.fit(X_train, y_train)

# Use o modelo para fazer previsões sobre os dados de teste
y_pred = rg.predict(X_test)
```

Fonte: trecho de código criado pelo próprio autor

Nota de acurácia de treinamento

Código 23 – Acurácia de do treinamento

```
accuracy = rg.score(X_train, y_train)
print('Ridge Regression score:', accuracy)
output
0.9796897768567819
```

Fonte: trecho de código criado pelo próprio autor

Nota de acurácia de teste

Código 24 – Acurácia de teste

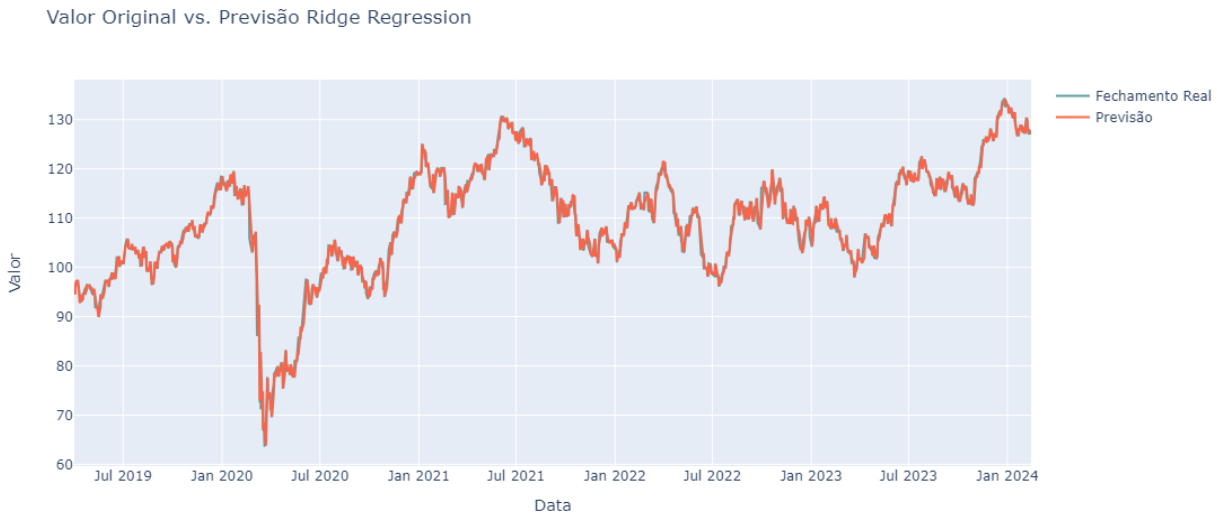
```
accuracy = rg.score(X_test, y_test)
print('Ridge Regression score:', accuracy)
output
0.9809913610533814
```

Fonte: trecho de código criado pelo próprio autor

Visualização dos Resultados: Os resultados da previsão são visualizados em um gráfico de dispersão, comparando os valores reais e previstos, e em um gráfico de linha, mostrando a evolução dos valores reais e previstos ao longo do tempo.

O *Gráfico 16* mostra a previsão feita pelo modelo Ridge Regression para a série temporal em questão. Ele representa a comparação entre os valores reais (observados) e os valores previstos pelo modelo para o período de teste.

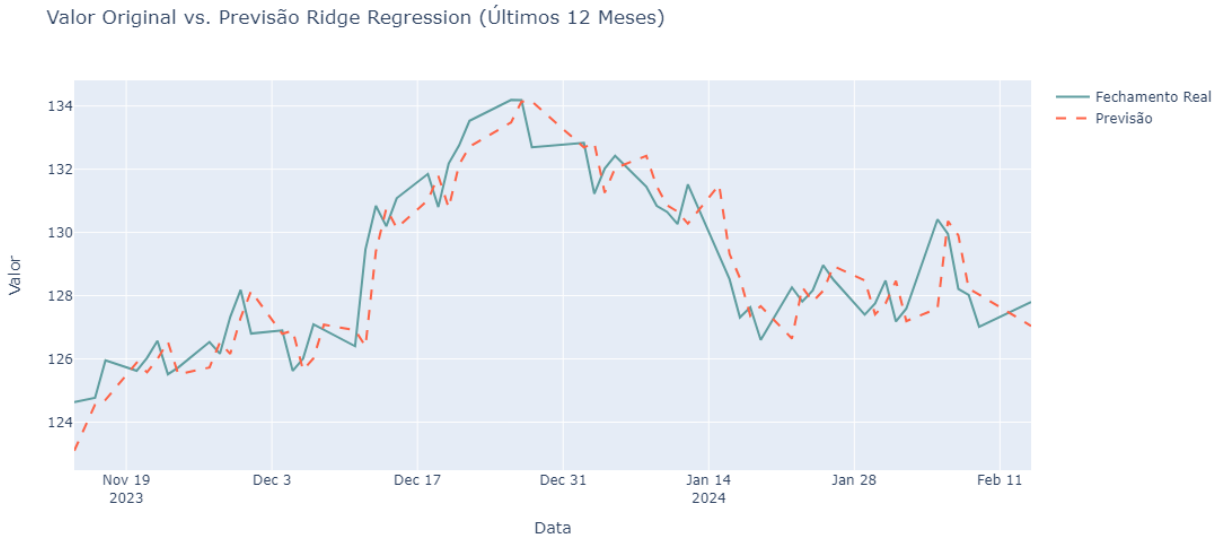
Gráfico 16 - Previsão Ridge Regression



Fonte: figuras criadas pelo próprio autor

O *Gráfico 17* mostra a previsão focado nos últimos 12 meses da série temporal, permitindo uma análise mais detalhada do desempenho do modelo nesse período.

Gráfico 17 – Previsão Ridge Regression (Últimos 12 Meses)



Fonte: figuras criadas pelo próprio autor

O *Gráfico 18* mostra o erro percentual absoluto médio (MAPE) da previsão feita pelo modelo Ridge Regression ao longo do tempo. O MAPE é uma métrica que calcula a média percentual dos erros de previsão em relação aos valores reais.

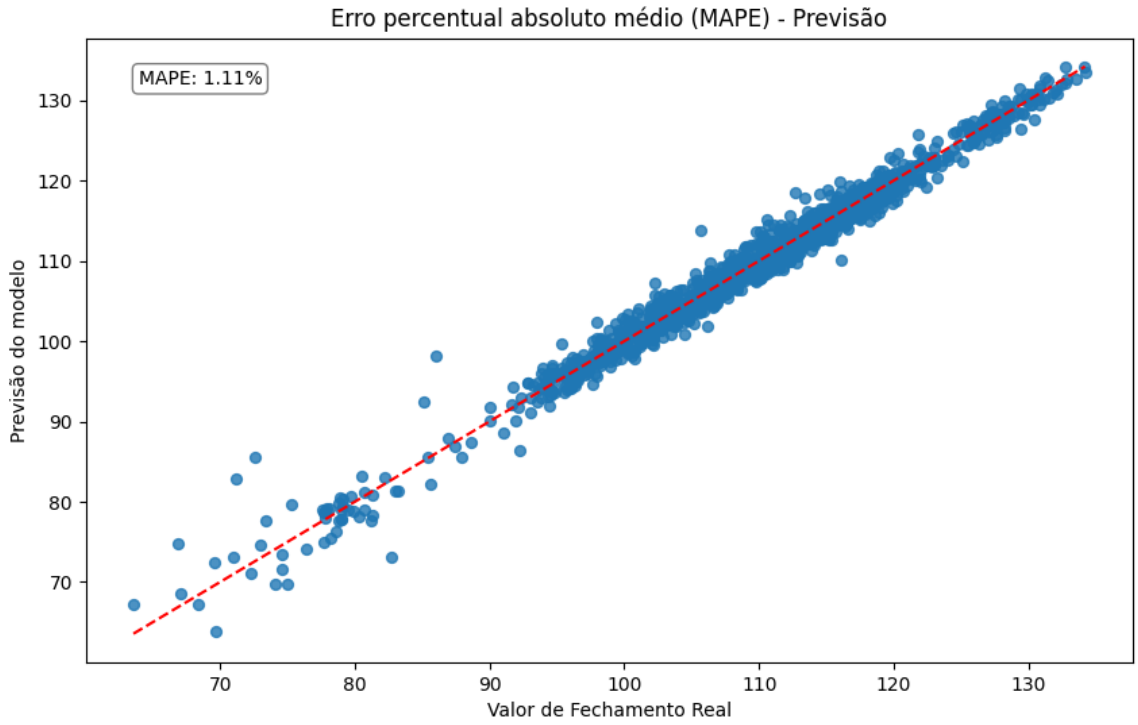


Gráfico 18 – Erro percentual absoluto médio (MAPE) - Previsão

Fonte: figuras criadas pelo próprio autor

O Erro Percentual Absoluto Médio (MAPE) foi de 1.12%, indicando que, em média, as previsões estão a 1.12% de erro em relação aos valores reais. Esses resultados sugerem que o modelo de Regressão Ridge é eficaz na previsão dos valores.

5.4.1. MÉTRICAS DO MODELO RIDGE REGRESSION

Avaliar o modelo, calculando as métricas de erro, como MSE, MAE, R^2 e MAPE, para avaliar a qualidade da previsão. Resultado das métricas são:

- Erro Quadrático Médio (MSE) = 2.72
- Erro Absoluto Médio (MAE) = 1.16
- Coeficiente de Determinação (R^2) = 0.98
- Erro Percentual Absoluto Médio (MAPE) = 1.12%

6. CONCLUSÃO

Em conclusão a análise e previsão do índice Bovespa utilizando os modelos ARIMA, Prophet, LSTM e Ridge Regression revelou insights importantes para compreender o comportamento do mercado financeiro brasileiro. A escolha desses modelos baseou-se em critérios como robustez, adaptabilidade, eficiência computacional, capacidade de generalização e precisão da previsão.

O modelo ARIMA mostrou-se eficaz na captura de padrões de autocorrelação e tendências da série temporal do índice Bovespa, apresentando métricas satisfatórias, como um Erro Quadrático Médio (MSE) de 3.82, Erro Absoluto Médio (MAE) de 1.53, Coeficiente de Determinação (R^2) de 0.94 e Erro Percentual Absoluto Médio (MAPE) de 1.35%.

Por outro lado, o modelo Prophet, desenvolvido pelo *Facebook*, destacou-se pela capacidade de lidar com diferentes tipos de sazonalidade, feriados e tendências de longo prazo, mesmo para usuários sem experiência avançada em ciência de dados. No entanto, apresentou métricas de erro um pouco mais elevadas, com um MSE de 29.80, MAE de 4.14, R^2 de 0.78 e MAPE de 3.97%.

O modelo LSTM obteve resultados bastante promissores na previsão do preço de fechamento da ação, com um erro médio quadrático (MSE) de 4.25 e um erro absoluto médio (MAE) de 1.70. Além disso, o coeficiente de determinação (R^2) de 0.95 indica que o modelo é capaz de explicar 95% da variância dos dados, demonstrando uma boa capacidade de ajuste aos dados observados. O erro percentual absoluto médio (MAPE) de 1.45% indica que, em média, as previsões do modelo estão muito próximas dos valores reais, sugerindo que o modelo LSTM é eficaz na previsão de preços de ações.

Já a Ridge Regression, uma técnica de regressão linear regularizada, mostrou-se eficiente para evitar o *overfitting* e lidar com multicolinearidade nos dados, apresentando métricas de erro bastante baixas, com um MSE de 2.72, MAE de 1.16, R^2 de 0.98 e MAPE de 1.12%.

Esses resultados indicam que o modelo Ridge Regression foi o mais preciso na previsão do índice Bovespa, seguido pelo LSTM, ARIMA e Prophet. No entanto, cada modelo possui suas vantagens e limitações, e a escolha do modelo ideal deve levar em consideração o contexto específico da análise e os objetivos do estudo.

Em suma, a combinação desses modelos pode fornecer uma visão mais abrangente e robusta do comportamento do mercado financeiro, auxiliando investidores, analistas e tomadores de decisão a compreender e antecipar as tendências do índice Bovespa.

REFERÊNCIAS

Do UOL. (14 de Julho de 2022). *COTAÇÕES.* Acesso em 02 de fevereiro de 2024, disponível em UOL: https://economia.uol.com.br/cotacoes/noticias/redacao/2022/07/14/dolar-ibovespa-operacao-14-julho.htm

Ferreira , G. (07 de junho de 2021). *É HEXA! Ibovespa bate 6º recorde seguido ao som das promessas de Lira.* Acesso em 02 de fevereiro de 2024, disponível em valorinveste: https://valorinveste.globo.com/mercados/renda-variavel/bolsas-e-indices/noticia/2021/06/07/e-hexa-ibovespa-bate-6o-recorde-seguido-ao-som-das-promessas-de-lira.ghtml

G1. (23 de março de 2020). *Bovespa segue exterior e fecha com forte queda.* Acesso em 02 de fevereiro de 2024, disponível em G1: https://g1.globo.com/economia/noticia/2020/03/23/bovespa.ghtml

Goeking, W. (30 de outubro de 2020). *Ibovespa tomba 7%, tem pior semana desde março e 3º mês seguido no vermelho.* Acesso em 02 de fevereiro de 2024, disponível em Valor Investe: https://valorinveste.globo.com/mercados/renda-variavel/bolsas-e-indices/noticia/2020/10/30/ibovespa-tomba-7percent-tem-pior-semana-desde-marco-e-3o-mes-seguido-no-vermelho.ghtml

Gomes, B. (28 de dezembro de 2023). *Bolsa fecha último pregão com 134 mil pontos e acumula alta de 22% no ano.* Acesso em 02 de fevereiro de 2024, disponível em UOL: https://economia.uol.com.br/cotacoes/noticias/redacao/2023/12/28/dolar-bolsa-28-de-dezembro.htm#:~:text=O%20Ibovespa%20fechou%20praticamente%20est%C3%A1vel,%2C%20aos%20134.193%2C72%20pontos.

Meta. (2024). *Documantation Prophet.* Acesso em 20 de janeiro de 2024, disponível em Prophet: https://facebook.github.io/prophet/docs/quick_start.html

O que é Ibovespa? Conheça o índice mais famoso da B3. (02 de janeiro de 2024). Acesso em 20 de janeiro de 2024, disponível em Toroinvestimentos Blog: https://blog.toroinvestimentos.com.br/bolsa/ibovespa/#:~:text=O%20Ibov%20foi%20criado%20em,do%20mercado%20financeiro%20do%20pa%C3%ADs.

sklearn. (2024). *sklearn.linear_model.Ridge.* Acesso em 03 de janeiro de 2024, disponível em https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

statsmodels. (2024). *statsmodels.tsa.arima.model.ARIMA.* Acesso em 19 de janeiro de 2024, disponível em https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html