**df\_test**

1. Odf\_test é criado a partir do dataframe original (df\_ibovespa) seguindo alguns passos:
2. Separação Temporal: Primeiro, definimos uma data de corte. Para a sua tarefa, definimos essa data para ser 30 dias antes da data mais recente no dataframe original (latest\_date - pd.Timedelta(days=30)). Odf\_test é então criado contendo todas as linhas do df\_ibovespa cuja data é igual ou posterior a essa data de corte. Isso garante que o conjunto de teste contenha os dados mais recentes, que o modelo ainda não viu.
3. Engenharia de Features: Depois de separar os dados, calculamos as mesmas features baseadas em janelas deslizantes (MA\_5, MA\_20, Daily\_Change, Volatility\_5, MA\_10, Volatility\_10) que calculamos para o conjunto de treino. É crucial aplicar a mesma lógica de engenharia de features a ambos os conjuntos para garantir que o modelo esteja sendo avaliado em dados com a mesma estrutura.
4. Criação da Variável Alvo (Trend): Da mesma forma que no conjunto de treino, criamos a colunaTRENDnodf\_test. Esta coluna indica a tendência (alta ou baixa) para o dia seguinte, com base na comparação do preço de fechamento do dia atual com o do dia seguinte.
5. Remoção de Valores Ausentes (NaNs): As features baseadas em janelas deslizantes (como médias móveis e volatilidade) produzem valores ausentes (NaN) no início do conjunto de dados, pois não há dados suficientes para calcular a média ou o desvio padrão para as primeiras janelas. Removemos essas linhas comNaNs dodf\_test usandodf\_test.dropna(inplace=True).
6. Portanto, odf\_test final é um subconjunto do dataframe original contendo os últimos 30 dias de dados, com as features de engenharia e a variável alvo calculadas, e com as linhas que continham valores ausentes removidas. É sobre estedf\_test preparado que o modelo faz as previsões e a acurácia é calculada.

**TREND**

A lógica para calcular a coluna TREND é bem simples e direta. Queremos prever se o preço de fechamento do dia seguinte será maior ou menor do que o preço de fechamento do dia atual.

Para cada dia no nosso conjunto de dados, comparamos o último(preço de fechamento) desse dia com o último do **dia seguinte**.

* Se o preço de fechamento do dia seguinte for **maior** do que o preço de fechamento do dia atual, significa que a tendência foi de **alta**. Neste caso, atribuímos o valor **1** para a coluna TREND naquele dia.
* Se o preço de fechamento do dia seguinte for **menor ou igual** ao preço de fechamento do dia atual, significa que a tendência foi de **baixa ou estabilidade**. Neste caso, atribuímos o valor **0** para a colunaTRENDnaquele dia.

Usamos a função  .shift-(1) no pandas para pegar o valor da linha seguinte (o preço de fechamento do dia seguinte) e comparamos com o valor da linha atual (o preço de fechamento do dia atual). O .astype(int) converte o resultado da comparação (True/False) para 1 ou 0.

Em resumo, a coluna TRENDé uma representação binária (0 ou 1) da direção do movimento do preço de fechamento de um dia para o outro.

**ACURACIA**

**a acuracia é calculada a partir de qual valor?**

A acurácia é calculada comparando **duas sequências de valores** no **conjunto de teste**:

1. Os **valores reais** da tendência para cada dia no conjunto de teste. Estes são os valores na coluna TREND do seu df\_test.
2. As **previsões** feitas pelo seu modelo para a tendência de cada dia no conjunto de teste. Estes são os valores que o modelo previu que seriam a tendência.

A função accuracy\_score conta quantos dias o modelo previu a tendência corretamente (ou seja, onde o valor previsto é igual ao valor real) e divide esse número pelo total de dias no conjunto de teste. O resultado é a porcentagem de acertos do modelo no conjunto de teste.

Em termos simples, a acurácia é a proporção de previsões corretas em relação ao total de previsões feitas no conjunto de dados que o modelo não usou para aprender.

**1. Por que escolhemos os modelos utilizados (Regressão Logística e Random Forest):**

* **Regressão Logística:** Este foi o primeiro modelo de classificação que utilizamos. A escolha inicial se deu pela sua simplicidade e interpretabilidade. É um bom ponto de partida para problemas de classificação binária (tendência de alta ou baixa), pois é fácil de entender como as features influenciam a probabilidade da classe positiva. No entanto, por ser um modelo linear, pode ter dificuldade em capturar relações mais complexas nos dados.
* **Random Forest Classifier:** Este modelo foi introduzido como uma tentativa de melhorar a acurácia, pois é um modelo mais poderoso capaz de capturar interações não lineares entre as features. Como mencionei anteriormente, é um modelo de ensemble (combinação de várias árvores de decisão) que geralmente oferece bom desempenho, é robusto a outliers e menos propenso a overfitting do que árvores de decisão individuais.

**2. Como tratamos a natureza sequencial dos dados:**

A natureza sequencial dos dados de séries temporais (dados de mercado financeiro, neste caso) foi tratada através da **engenharia de features baseada em janelas deslizantes (rolling windows)**. Em vez de usar apenas os dados de um único dia para prever a tendência do dia seguinte, criamos novas features que resumem informações de períodos anteriores:

* **Médias Móveis (MA\_5, MA\_10, MA\_20):** Calculamos a média do preço de fechamento (último) sobre as últimas 5, 10 e 20 janelas de tempo. Isso ajuda a suavizar as flutuações de curto prazo e identificar a direção geral da tendência.
* **Volatilidade (Volatility\_5, Volatility\_10):** Calculamos o desvio padrão da variação diária (Daily\_change) sobre as últimas 5 e 10 janelas de tempo. Isso nos dá uma medida de quão volátil tem sido o preço do ativo em períodos recentes.
* **Variação Diária (Daily\_Change):** Embora não seja estritamente uma janela deslizante, a variação diária (Último -Abertura) também utiliza dados de dois pontos no tempo para criar uma feature que reflete o movimento do preço dentro de um único dia.

Essas features lagged (defasadas no tempo) permitem que o modelo considere o histórico recente dos preços e da volatilidade ao tomar uma decisão sobre a tendência futura, abordando assim a dependência temporal dos dados.

**3. Quais os trade-offs entre acuracidade e overfitting:**

* **Acurácia:** Refere-se à capacidade do modelo de prever corretamente a tendência (alta ou baixa) no conjunto de dados de teste. Nosso objetivo é ter uma alta acurácia, idealmente acima de 75% como solicitado.
* **Overfitting:** Ocorre quando um modelo aprende "demais" os detalhes e ruídos do conjunto de treino, a ponto de não generalizar bem para dados novos e não vistos (o conjunto de teste). Um modelo overfitado terá uma acurácia muito alta no conjunto de treino, mas uma acurácia significativamente menor no conjunto de teste.

**Trade-offs:**

* **Modelos mais complexos (como Random Forest com muitas features):** Têm o potencial de alcançar maior acurácia no conjunto de treino, capturando padrões intrincados. No entanto, eles também têm um risco maior de overfitting, especialmente se o conjunto de treino for pequeno ou se houver muitas features que são apenas ruído. Se um modelo overfitar, sua acurácia no conjunto de teste será enganosamente baixa, pois ele não consegue aplicar o que aprendeu a novos dados de forma eficaz.
* **Modelos mais simples (como Regressão Logística):** São menos propensos a overfitting, pois têm menor capacidade de memorizar o ruído do conjunto de treino. No entanto, eles podem não ser capazes de capturar a complexidade total dos dados, resultando em uma acurácia limitada tanto no treino quanto no teste.

No nosso caso, após adicionar as features de janela deslizante, a acurácia com Regressão Logística foi de 0.67. Ao tentar usar o Random Forest com essas features, tivemos o problema do conjunto de teste ficar vazio (o que impediu a avaliação correta). Se tivéssemos conseguido avaliar o Random Forest com um conjunto de teste válido, precisaríamos comparar a acurácia no treino e no teste. Uma grande diferença indicaria overfitting. A acurácia de 1.00 observada em uma tentativa anterior com o Random Forest (quando ajustamos a data de split de uma forma que não era a solicitada) foi um forte indicativo de overfitting devido ao tamanho reduzido do conjunto de teste após a remoção dos NaNs.

Portanto, o desafio é encontrar um modelo e um conjunto de features que capturem os padrões relevantes para prever a tendência sem memorizar o ruído específico do conjunto de treino, garantindo assim uma boa acurácia no conjunto de teste e uma capacidade de generalização para dados futuros.

**Revisar a Lógica do Cálculo da Acurácia utilizando machine learning:**

* 1. Certifique-se de que você está usando a função accurary\_score da sklearn.mtrics corretamente, passando primeiro os valores reais (y\_test) e depois as previsões do modelo (predictions).
  2. Verifique se y\_test e (predictions) têm o mesmo tamanho e estão na ordem correta, correspondendo às mesmas amostras no conjunto de teste.

**Verificar o Conjunto de Teste:**

* 1. Confirme se o conjunto de teste (df\_test) realmente contém os dados que o modelo não viu durante o treinamento. No seu caso, você já fez isso separando os últimos 30 dias.
  2. Verifique se a coluna alvo (Trend) e as features no conjunto de teste foram preparadas da mesma forma que no conjunto de treino (por exemplo, se NaNs foram tratados de maneira consistente).

**Analisar as Previsões e Valores Reais Manualmente (Amostra):**

* 1. Pegue uma pequena amostra aleatória do seu conjunto de teste.
  2. Para cada amostra, compare o valor real da tendência (y\_test) com a previsão do modelo (predictions).
  3. Calcule manualmente a porcentagem de acertos para essa pequena amostra. Isso deve ser próximo à acurácia geral calculada pela função accuracy\_score.

Você pode fazer isso exibindo uma parte do x\_test, y\_test e(predictions) lado a lado. Por exemplo:

# Exemplo para visualizar alguns resultados  
results = X\_test.copy()  
results['Real\_Trend'] = y\_test  
results['Predicted\_Trend'] =(predictions)  
display(results[['Real\_Trend', 'Predicted\_Trend']].head())

Depois, conte manualmente quantos a previsão e o valor real coincidem nas linhas exibidas.

4. **Verificar o Tratamento de NaNs:**

* Como vimos anteriormente, a remoção de NaNs pode afetar o tamanho do conjunto de teste. Certifique-se de que, após remover os NaNs, o conjunto de teste ainda contém um número razoável de amostras para uma avaliação significativa. Uma acurácia de 1.00 em um conjunto de teste muito pequeno pode ser enganosa. Use print(f"Shape of test data after dropping NaNs: {X\_test.shape}") para verificar o tamanho.

**Métricas Adicionais:**

* + A acurácia é uma métrica útil, mas pode não contar a história completa, especialmente se as classes forem desbalanceadas (muito mais dias de alta do que de baixa, por exemplo). Considere usar outras métricas como:
    - **Precisão (Precision):** Das vezes que o modelo previu alta, quantas vezes estava correto?
    - **Recall:** Das vezes que a tendência real foi de alta, quantas o modelo previu corretamente?
    - **F1-Score:** Uma média harmônica de precisão e recall.
    - **Matriz de Confusão:** Mostra o número de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos.

Você pode importar e usar essas métricas da seguinte forma:

from sklearn.metrics import confusion\_matrix, classification\_report  
  
print(confusion\_matrix(y\_test,(predictions)))  
print(classification\_report(y\_test,(predictions)))

Ao seguir esses passos, você pode ter mais confiança de que a acurácia calculada reflete o desempenho real do seu modelo no conjunto de teste.