Aula 14 – Estudo de Caso de Análise Exploratória (EDA) - Parte 1

Bem-vindo(a) à Aula 14 do nosso Curso de Python para Análise de Dados! Se você chegou até aqui, é porque já compreendeu a força do Python e suas bibliotecas para manipular informações. Agora, vamos dar um passo crucial: mergulhar em um cenário real, onde os dados não são perfeitos e a intuição é tão importante quanto o código.

Nesta aula, vamos desvendar a primeira parte de um Estudo de Caso de Análise Exploratória de Dados (EDA). Pense na EDA como o trabalho de um detetive: antes de tirar conclusões, você precisa entender a cena do crime, coletar as evidências e garantir que elas estejam limpas e organizadas. É a fase onde fazemos as perguntas certas e preparamos o terreno para as respostas.

Ao final desta jornada, você será capaz de definir um problema de negócio e traduzi-lo em perguntas que os dados podem responder, escolher e carregar um dataset do mundo real, e realizar os primeiros e mais importantes passos da limpeza de dados, como identificar e tratar valores nulos, duplicados e tipos incorretos. Esta base sólida é o que diferencia um bom analista de dados.

O Detetive de Dados: Definindo o Problema e as Perguntas de Negócio

Por que começar pelo problema?

Imagine que você é um consultor recém-contratado por uma empresa. Eles têm montanhas de dados, mas não sabem o que fazer com eles. Sua primeira tarefa não é escrever código, mas sim entender o que a empresa realmente precisa. Qual é a dor? Qual é o desafio? Sem essa clareza, qualquer análise será um tiro no escuro.

Definir o problema de negócio é o ponto de partida de qualquer projeto de análise de dados. É a bússola que guiará todas as suas escolhas, desde qual dataset usar até quais visualizações criar. Uma definição clara do problema garante que seu trabalho seja relevante e traga valor real para a organização.

Transformando problemas em perguntas

A partir do problema, surgem as perguntas de negócio. Elas são a ponte entre a necessidade da empresa e o potencial dos dados. Por exemplo, se o problema é "queda nas vendas", as perguntas podem ser:

- "Quais produtos tiveram a maior queda?"
- "Em quais regiões a queda foi mais acentuada?"
- "Existe alguma sazonalidade nesse declínio?"

Essas perguntas são o que você tentará responder com a EDA.

A Escolha do Cenário: Selecionando um Dataset do Mundo Real

Dados do Mundo Real

Com o problema e as perguntas de negócio em mente, o próximo passo é encontrar os dados que podem nos ajudar a respondê-las. No mundo real, os dados raramente vêm prontos e organizados. Eles podem estar em bancos de dados, planilhas, APIs ou até mesmo em documentos não estruturados.

Mina de Ouro para Aprendizado

Um dataset do mundo real é uma mina de ouro para o aprendizado. Diferente dos exemplos didáticos perfeitos, ele vem com ruídos, inconsistências e desafios que simulam a realidade do trabalho de um analista. É aqui que a teoria se encontra com a prática.

Nosso Estudo de Caso

Para este estudo de caso, vamos trabalhar com um dataset fictício de **Vendas de E-commerce**. Ele contém informações como ID do pedido, data da venda, ID do cliente, nome do produto, categoria, quantidade, preço unitário, valor total, método de pagamento e região.

💡 **Dica Importante:** A escolha do dataset é crucial, pois ele será o seu "campo de batalha". Este tipo de dado é comum em muitas empresas e nos permitirá explorar diversas facetas da análise.

Preparando o Terreno: Carregando e Inspecionando Dados com Pandas

Com nosso dataset escolhido, é hora de trazê-lo para o ambiente de trabalho. No ecossistema Python, a biblioteca **Pandas** é a ferramenta padrão e mais poderosa para manipulação e análise de dados tabulares. Ela nos permite carregar dados de diversas fontes e representá-los em estruturas de dados intuitivas, como os DataFrames.

01

Carregar os Dados

Carregar os dados é o primeiro contato real com o conjunto que você vai analisar. É como abrir uma caixa misteriosa: você sabe o que está dentro, mas ainda não viu os detalhes. O Pandas simplifica esse processo com funções como pd.read_csv() para arquivos CSV, pd.read_excel() para planilhas e muitas outras para diferentes formatos.

02

Inspeção Inicial

Após carregar, a inspeção inicial é vital. Comandos como .head() (para ver as primeiras linhas), .tail() (as últimas), .info() (resumo das colunas e tipos de dados) e .shape (dimensões do dataset) nos dão uma visão panorâmica. É como dar uma olhada rápida no mapa antes de iniciar a viagem.

```
import pandas as pd

# Carregando o dataset de exemplo
df_vendas = pd.read_csv('vendas_ecommerce.csv')

# Inspecionando as primeiras 5 linhas
print("Primeiras 5 linhas do dataset:")
print(df_vendas.head())

# Verificando informações gerais do dataset
print("\nInformações gerais do dataset:")
df_vendas.info()

# Verificando as dimensões do dataset (linhas, colunas)
print(f"\nDimensões do dataset: {df_vendas.shape}")
```

Desvendando os Segredos: Primeiros Passos da Limpeza de Dados

Dados limpos = Análises confiáveis

Você já carregou seus dados e fez uma inspeção inicial. Agora, a realidade se impõe: dados do mundo real são raramente perfeitos. Eles vêm com imperfeições, inconsistências e lacunas. Ignorar essas falhas é como construir uma casa sobre areia movediça: a estrutura pode parecer boa por fora, mas desmoronará sob pressão.

A limpeza de dados é a etapa onde corrigimos essas imperfeições para garantir que nossa análise seja precisa e confiável. É um processo iterativo e muitas vezes demorado, mas absolutamente essencial. Pense nisso como a fase de polimento de uma joia bruta: removemos as impurezas para que seu brilho natural possa emergir.

1

Valores Nulos

Verificação e tratamento de dados ausentes

2

Registros Duplicados

Identificação e remoção de linhas repetidas

3

Tipos de Dados

Correção de tipos incorretos nas colunas

Nesta primeira parte da limpeza, focaremos nesses três pilares fundamentais. Dominar esses passos iniciais é a base para qualquer análise exploratória robusta e para a construção de modelos preditivos eficazes.

Onde Estão as Lacunas? Verificando Valores Nulos

Valores nulos, ou dados ausentes, são um dos problemas mais comuns e frustrantes em qualquer dataset. Eles podem surgir por diversas razões: falhas na coleta de dados, informações não preenchidas no momento do registro, ou até mesmo erros de integração entre diferentes sistemas. A presença de nulos pode distorcer suas estatísticas, invalidar modelos e levar a conclusões errôneas.

Identificar

Usar .isnull() para criar uma máscara booleana

Quantificar

Aplicar .sum() para contar nulos por coluna

Analisar

Calcular a porcentagem de nulos em relação ao total

Entender a extensão dos valores nulos é crucial. Uma coluna com 5% de nulos pode ser tratada de forma diferente de uma com 90% de nulos. É como ter algumas peças faltando em um quebra-cabeça: se faltam poucas, podemos tentar adivinhar; se faltam muitas, talvez seja melhor descartar o quebra-cabeça ou procurar outro.

```
# Verificando a quantidade de valores nulos por coluna
print("\nValores nulos por coluna antes do tratamento:")
print(df_vendas.isnull().sum())

# Verificando a porcentagem de valores nulos por coluna
print("\nPorcentagem de valores nulos por coluna:")
print((df_vendas.isnull().sum() / len(df_vendas)) * 100)
```

Lidando com o Vazio: Estratégias para Valores Nulos

Uma vez que identificamos os valores nulos, a próxima pergunta é: o que fazer com eles? Não existe uma resposta única, e a melhor estratégia depende do contexto da sua análise, da quantidade de nulos e da natureza da coluna. É uma decisão que exige bom senso e conhecimento do domínio dos dados.

Estratégia 1: Remover

Uma opção é **remover** as linhas ou colunas que contêm valores nulos. Se uma coluna tem uma quantidade muito grande de nulos (ex: mais de 70-80%), ela pode não ser útil para a análise e pode ser descartada. Da mesma forma, se poucas linhas têm nulos e o dataset é grande, remover essas linhas pode ser uma solução simples com pouco impacto. O método df.dropna() do Pandas é ideal para isso.

Estratégia 2: Imputar

Outra abordagem é a **imputação**, que consiste em preencher os valores nulos com alguma estimativa. Para dados numéricos, podemos usar a média (.mean()), mediana (.median()) ou moda (.mode()) da coluna. Para dados categóricos, a moda é frequentemente utilizada. O Pandas oferece o método df.fillna() para realizar a imputação. A escolha entre média e mediana, por exemplo, depende da distribuição dos dados: a mediana é mais robusta a *outliers*.

```
# Exemplo de tratamento de nulos:
# 1. Remover linhas com nulos em colunas específicas (se houver poucas)
# df_vendas.dropna(subset=['Nome_Produto', 'Preco_Unitario'], inplace=True)

# 2. Preencher nulos em colunas numéricas com a média
# df_vendas['Quantidade'].fillna(df_vendas['Quantidade'].mean(), inplace=True)

# 3. Preencher nulos em colunas categóricas com a moda
# df_vendas['Metodo_Pagamento'].fillna(df_vendas['Metodo_Pagamento'].mode()[0], inplace=True)

# Para fins didáticos, vamos simular um preenchimento simples para uma coluna numérica
# Supondo que 'Preco_Unitario' tenha nulos, preencheremos com a mediana
if df_vendas['Preco_Unitario'].isnull().any():
 mediana_preco = df_vendas['Preco_Unitario'].median()
 df_vendas['Preco_Unitario'].fillna(mediana_preco, inplace=True)
 print(f"\nValores nulos em 'Preco_Unitario' preenchidos com a mediana: {mediana_preco}")

# Verificando novamente os nulos após o tratamento
print("\nValores nulos por coluna após tratamento (exemplo):")
print(df_vendas.isnull().sum())
```

Onde a Repetição Não é Virtude: Identificando e Removendo Duplicados

Além dos valores nulos, outro problema comum que pode comprometer a integridade da sua análise são os registros duplicados. Imagine que um mesmo pedido de venda foi registrado duas vezes, ou que um cliente aparece com o mesmo ID em diferentes linhas. Isso inflaria artificialmente as contagens, distorceria médias e levaria a conclusões erradas sobre o volume de vendas ou a base de clientes.

Identificar duplicatas é crucial para garantir que cada observação em seu dataset seja única e represente uma informação distinta. O Pandas facilita essa tarefa com o método .duplicated(), que retorna uma série booleana indicando se cada linha é uma duplicata de uma linha anterior. Combinado com .sum(), podemos rapidamente saber quantos registros duplicados existem.

⚠️ **Atenção:** Remover essas duplicatas é um passo de higiene fundamental. O método .drop_duplicates() do Pandas faz exatamente isso, eliminando as linhas que são cópias exatas de outras. É como ter uma lista de convidados para um evento e garantir que ninguém foi contado duas vezes.

```
# Verificando a quantidade de linhas duplicadas
print("\nQuantidade de linhas duplicadas antes da remoção:")
print(df_vendas.duplicated().sum())

# Removendo as linhas duplicadas
df_vendas.drop_duplicates(inplace=True)

# Verificando novamente após a remoção
print("\nQuantidade de linhas duplicadas após a remoção:")
print(df_vendas.duplicated().sum())
print(f"Novo número de linhas no dataset: {df_vendas.shape[0]}")
```

A Essência dos Dados: Compreendendo e Ajustando Tipos de Dados

Os tipos de dados são a espinha dorsal de qualquer dataset. Eles definem como o Python e o Pandas interpretam e interagem com as informações em cada coluna. Uma coluna de "preço" deve ser numérica para que possamos realizar cálculos; uma coluna de "data" deve ser do tipo data/hora para que possamos fazer análises temporais. Se os tipos estiverem incorretos, operações simples podem falhar ou produzir resultados sem sentido.

Verificar com .dtypes

O método .dtypes do Pandas é seu melhor amigo aqui. Ele retorna uma série com o tipo de dado de cada coluna (ex: int64, float64, object, datetime64). É comum encontrar números armazenados como object (strings) ou datas como object porque o Pandas não conseguiu inferir o tipo correto durante o carregamento.

Identificar Tipos Incorretos

Identificar esses tipos incorretos é como verificar a etiqueta de um produto: você precisa saber se é um alimento, um eletrônico ou um brinquedo para saber como usá-lo corretamente. Um número tratado como texto não pode ser somado, e uma data tratada como texto não pode ser usada para calcular a diferença entre dias.

```
# Verificando os tipos de dados atuais
print("\nTipos de dados antes do ajuste:")
print(df_vendas.dtypes)
```

Moldando os Dados: Convertendo Tipos para Análise Eficaz

Uma vez que identificamos os tipos de dados incorretos, o próximo passo é corrigi-los. A conversão de tipos é uma operação fundamental na limpeza de dados, pois ela desbloqueia todo o potencial analítico de suas colunas. Sem os tipos corretos, muitas das funções poderosas do Pandas e de outras bibliotecas não funcionarão como esperado.

1

Tipos Numéricos

Para tipos numéricos, podemos usar .astype(float) ou .astype(int).

Datas e Horas

Para datas, pd.to_datetime() é a função ideal, capaz de inferir diversos formatos de data e hora.

Tratamento de Erros

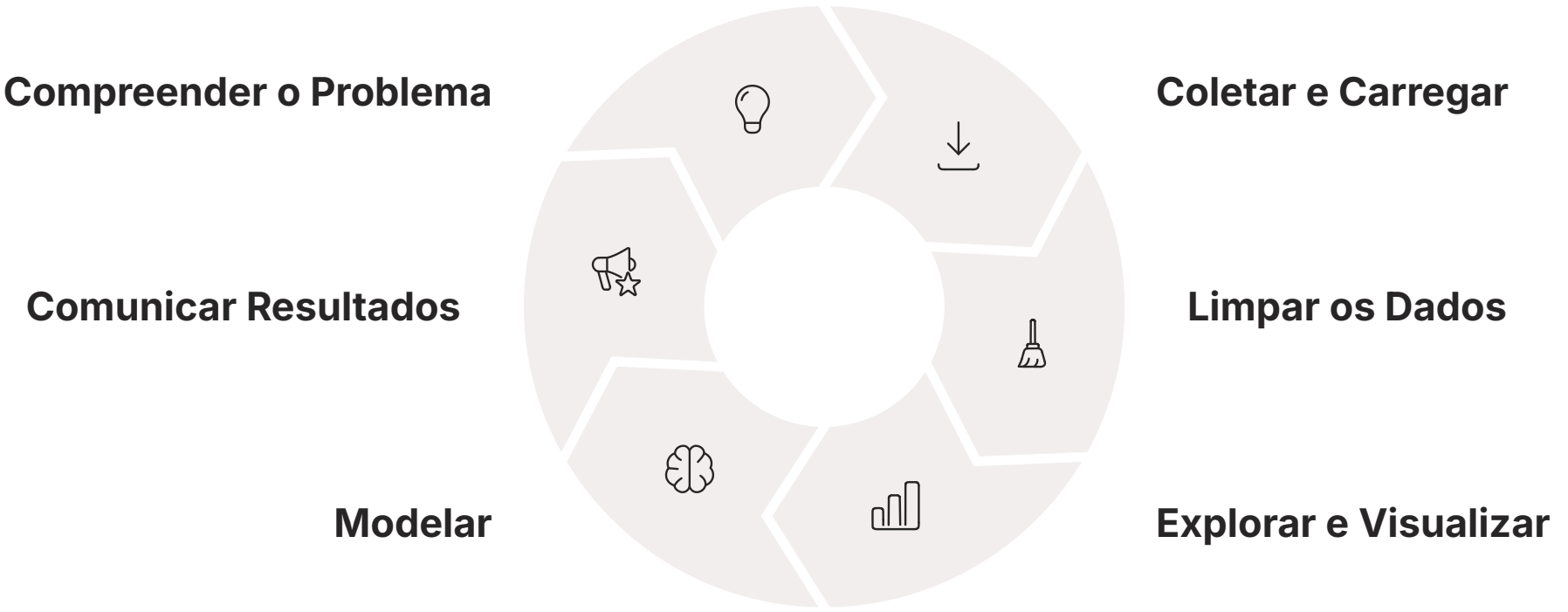
É importante estar atento a erros durante a conversão, e o parâmetro errors='coerce' pode transformar valores problemáticos em NaN.

Corrigir os tipos de dados é como organizar uma biblioteca: você agrupa os livros por gênero, os filmes por categoria. Isso permite que você encontre o que precisa rapidamente e use cada item da forma correta. Com os tipos certos, você pode somar preços, calcular a duração entre eventos e filtrar dados por períodos específicos, abrindo um leque de possibilidades para a sua análise.

```
# Exemplo de conversão de tipos:
# 1. Converter 'Data_Venda' para datetime
df_vendas['Data_Venda'] = pd.to_datetime(df_vendas['Data_Venda'], errors='coerce')

# 2. Converter 'Preco_Unitario' e 'Valor_Total' para float (se não forem)
# Assumindo que já são numéricos, mas se fossem 'object' com números, faríamos:
# df_vendas['Preco_Unitario'] = pd.to_numeric(df_vendas['Preco_Unitario'], errors='coerce')
# df_vendas['Valor_Total'] = pd.to_numeric(df_vendas['Valor_Total'], errors='coerce')

# Verificando os tipos de dados após o ajuste
print("\nTipos de dados após o ajuste:")
print(df_vendas.dtypes)

# Verificando se surgiram nulos após a conversão (se errors='coerce' foi usado)
print("\nValores nulos após conversão de tipos (se aplicável):")
print(df_vendas.isnull().sum())
```

O Fluxo de Trabalho Integrado: EDA na Prática com Jupyter/Colab

Até agora, vimos cada etapa da limpeza de dados de forma isolada. Mas, na prática, a Análise Exploratória de Dados (EDA) é um processo fluido e iterativo. As ferramentas como Jupyter Notebooks e Google Colab são ideais para esse fluxo de trabalho, pois permitem que você execute código em blocos, visualize resultados imediatamente e adicione anotações explicativas.



Um Jupyter Notebook é como um laboratório interativo onde você pode experimentar, testar hipóteses e documentar suas descobertas em tempo real. Você carrega os dados, inspeciona, identifica um problema (nulos, duplicatas, tipos), resolve-o, e então re-inspeciona para ver o efeito. Esse ciclo se repete até que os dados estejam prontos para a próxima fase da análise.

Esse fluxo de trabalho "End-to-End" (de ponta a ponta) é o padrão da indústria. Começa com a compreensão do problema, passa pela coleta e limpeza dos dados, avança para a exploração e visualização, e culmina na modelagem e comunicação dos resultados. A limpeza que estamos fazendo agora é a fundação para todo o resto.

Além do Básico: Considerações Avançadas na Limpeza de Dados

Outliers

Embora tenhamos focado nos pilares da limpeza de dados – nulos, duplicados e tipos – é importante saber que o universo da limpeza é muito mais vasto. À medida que você ganha experiência, encontrará desafios mais complexos que exigirão abordagens mais sofisticadas.

Um desses desafios são os **outliers**, ou valores atípicos. São pontos de dados que se desviam significativamente da maioria das outras observações. Eles podem ser erros de registro ou eventos genuinamente raros. Identificar e decidir como tratá-los (remover, transformar, ou manter) é crucial, pois podem distorcer estatísticas e modelos.

Inconsistências Textuais

Outra questão comum são as **inconsistências textuais**. Imagine que a categoria de um produto aparece como "Eletrônicos", "eletronicos" e "Eletro". Para o computador, são três categorias diferentes. Padronizar esses valores (ex: tudo para "Eletrônicos") é essencial para uma contagem e análise corretas.

Engenharia de Features

Além disso, a **engenharia de features** – a criação de novas colunas a partir das existentes (ex: "Mês da Venda" a partir de "Data da Venda") – é uma forma de enriquecer o dataset para análises futuras.

A Importância da Documentação: Registrando Cada Passo da EDA

A Análise Exploratória de Dados não é apenas sobre escrever código; é também sobre contar uma história. Cada passo que você toma – desde a definição do problema até a limpeza dos dados – é uma parte dessa narrativa. Documentar seu processo é tão importante quanto o próprio código, especialmente em projetos colaborativos ou quando você precisa revisitar seu trabalho meses depois.

Reprodutibilidade

A documentação garante a **reprodutibilidade** da sua análise. Se outra pessoa (ou você mesmo no futuro) precisar entender como você chegou a certas conclusões, um notebook bem documentado será um guia inestimável.

Colaboração

Ela também facilita a **colaboração**, permitindo que membros da equipe compreendam suas decisões e contribuam de forma eficaz.

Em ambientes como Jupyter e Colab, você pode usar células de Markdown para adicionar explicações detalhadas, contextuais e até mesmo gráficos. Comentários no código (#) também são essenciais para explicar trechos específicos. Pense na documentação como o diário de bordo do seu projeto: ele registra a jornada, os desafios superados e as descobertas feitas.

Conectando os Pontos: Da Limpeza à Próxima Fase da Análise

- [ ] **Definir um problema de negócio e traduzi-lo em perguntas de análise**
- [ ] **Carregar e realizar uma inspeção inicial de um dataset usando Pandas**
- [ ] **Identificar e tratar valores nulos, seja removendo-os ou imputando-os**
- [ ] **Detectar e eliminar registros duplicados para garantir a unicidade dos dados**
- [ ] **Verificar e corrigir os tipos de dados das colunas, preparando-as para análises específicas**

Chegamos ao final da primeira parte do nosso estudo de caso de Análise Exploratória de Dados. Percorremos um caminho fundamental: partimos da compreensão de um problema de negócio, selecionamos um dataset real e aplicamos os primeiros e mais importantes passos da limpeza de dados. Agora, temos um dataset mais confiável e pronto para ser explorado.

> 🎯 **Próximos Passos:** Com os dados limpos e organizados, a fundação está lançada. A próxima etapa natural é começar a visualizar e resumir esses dados para descobrir padrões, tendências e anomalias. É onde a história dos dados realmente começa a se revelar.

Consolidação e Próximos Passos

Resumo da Aula

Nesta aula, desvendamos a importância da fase inicial da Análise Exploratória de Dados (EDA), focando na preparação e limpeza dos dados. Compreendemos que um bom analista de dados atua como um detetive, definindo o problema, inspecionando as evidências (dados) e garantindo sua integridade antes de tirar qualquer conclusão. Dominar a identificação e o tratamento de nulos, duplicados e tipos de dados é a base para qualquer análise robusta e confiável.

Em Prática

Ao iniciar um novo projeto de dados, sempre comece com a definição clara do problema. Em seguida, carregue seus dados e dedique um tempo significativo à inspeção inicial e à limpeza. Verifique nulos com isnull().sum(), remova duplicatas com drop_duplicates() e ajuste os tipos com astype() ou pd.to_datetime(). Lembre-se: dados limpos são a chave para insights precisos.

Autoavaliação

1. Qual é a principal razão para definir o problema de negócio e as perguntas de negócio antes de iniciar a análise exploratória de dados?
   - a) Para garantir que o código seja escrito de forma mais rápida.
   - b) Para focar a análise em objetivos claros e relevantes para a empresa.
   - c) Para determinar qual biblioteca Python deve ser utilizada.
   - d) Para evitar a necessidade de limpeza de dados.
2. Ao inspecionar um dataset com Pandas, qual método é mais adequado para obter um resumo rápido dos tipos de dados de cada coluna e a contagem de valores não nulos?
   - a) df.head()
   - b) df.describe()
   - c) df.info()
   - d) df.shape
3. Você identificou que a coluna 'Preco_Unitario' em seu DataFrame possui valores nulos. Qual das seguintes abordagens é geralmente a mais robusta para preencher esses nulos se a coluna tiver *outliers*?
   - a) Remover todas as linhas que contêm nulos na coluna 'Preco_Unitario'.
   - b) Preencher os nulos com a média da coluna 'Preco_Unitario'.
   - c) Preencher os nulos com a mediana da coluna 'Preco_Unitario'.
   - d) Preencher os nulos com um valor fixo, como zero.
4. Após carregar um dataset, você percebe que a coluna 'Data_Venda' está como tipo 'object'. Qual é a melhor prática para convertê-la para um tipo de data/hora adequado para análises temporais?
   - a) Usar df['Data_Venda'].astype(str).
   - b) Usar df['Data_Venda'].astype(datetime).
   - c) Usar pd.to_datetime(df['Data_Venda']).
   - d) Não é necessário converter, 'object' funciona para datas.
5. Explique a importância de remover registros duplicados em um dataset e cite um exemplo de como a presença de duplicatas poderia distorcer uma análise de vendas.

Gabarito e Recursos Adicionais

1	2	3	4
Resposta: b)	**Resposta: c)**	**Resposta: c)**	**Resposta: c)**

Próxima Aula

Na **Aula 15 – Estudo de Caso de Análise Exploratória (EDA) - Parte 2**, continuaremos nosso estudo de caso, mergulhando na visualização de dados e na identificação de padrões e tendências.

Recursos Adicionais

- **Documentação oficial do Pandas**: Para aprofundar nos métodos de manipulação de dados.
- **Kaggle Datasets**: Para praticar com outros datasets do mundo real.
- **Livros sobre EDA**: Para explorar conceitos teóricos e práticos mais avançados.

> 📌 **NOTA IMPORTANTE:** As informações técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações nas bibliotecas ou melhores práticas da indústria.