INFO 01
Curso: UFCD 10793
UFCD/Módulo/Temática: UFCD 10793 - **Fundamentos de Python**
Ação: 10793_05/AG
Formador/a: Sandra Liliana Meira de Oliveira
Data:
Nome do Formando/a:

PANDAS

1. Introdução e Preparação do Ambiente

O pandas é uma biblioteca Python para análise e manipulação de dados, desenvolvida para simplificar tarefas que seriam complexas ou demoradas com ferramentas como listas ou dicionários em Python puro. Baseia-se no NumPy, uma biblioteca de arrays numéricos de alta performance, e foi projetada para trabalhar com grandes volumes de dados, mantendo a flexibilidade e simplicidade.

Porquê usar pandas?

- **Estruturas de Dados Ricas:** O pandas oferece **DataFrame** e **Series**, estruturas tabulares que são intuitivas para manipular dados etiquetados e heterogéneos.
- **Funcionalidades de Alto Nível:** Permite carregar, transformar e analisar dados rapidamente.
- **Integração:** Funciona bem com outras bibliotecas de Data Science (e.g., NumPy, Matplotlib, scikit-learn).
- **Eficiência:** É otimizado para manipulação de grandes volumes de dados em memória.

Cenários Reais

- **Limpeza de Dados:** Identificar e corrigir valores ausentes ou inconsistentes.
- **Exploração de Dados:** Extrair estatísticas resumidas e identificar padrões.
- **Transformação:** Combinar e reorganizar dados de diferentes fontes.
- **Automação:** Automatizar relatórios e análises repetitivas.

Como instalar a biblioteca

No terminal executa o comando

```
pip install pandas
```

2. Estruturas de Dados Centrais: Series e DataFrame

Series

- Uma **Series** é uma estrutura de dados unidimensional no pandas, similar a um array do NumPy ou uma lista em Python, mas com a vantagem de incluir um **índice** para cada elemento.
- Cada valor na Series é associado a uma **etiqueta única** (índice), que pode ser numérica ou textual. Isto facilita o acesso direto aos elementos através destes rótulos.

Principais Caraterísticas:

- **Homogeneidade:** Todos os elementos de uma Series são do mesmo tipo de dados.
- **Indexação:** Cada elemento tem um rótulo que permite acessá-lo de forma direta.
- **Versatilidade:** Pode ser criada a partir de listas, dicionários, arrays NumPy, ou mesmo valores escalares.

DataFrame

- O **DataFrame** é uma estrutura bidimensional (tabela) que organiza os dados em linhas e colunas.
- Cada **coluna** de um DataFrame é uma **Series**, e todas as colunas partilham o mesmo índice (linhas).
- Permite trabalhar com dados heterogéneos (e.g., texto, números, datas).

Principais Caraterísticas:

- **Heterogeneidade:** Cada coluna pode ter um tipo de dado diferente.
- **Indexação dupla:** Linhas e colunas são indexadas, permitindo acesso fácil aos dados.
- **Flexibilidade:** Pode ser criado a partir de listas, dicionários, arrays NumPy, ou ficheiros como CSV e Excel e ainda através de consultas SQL.

1. **Criar uma Series a partir de uma lista**

python Copy code

```python
import pandas as pd

# Criar uma Series simples
numeros = [10, 20, 30, 40]
serie_numeros = pd.Series(numeros)
print(serie_numeros)
```

Saída:

go Copy code

```
0    10
1    20
2    30
3    40
dtype: int64
```

Explicação:

- A Series atribui automaticamente índices (0, 1, 2, ...) a cada elemento.

2. **Criar uma Series com índices personalizados**

python Copy code

```python
alunos = ['João', 'Maria', 'Pedro', 'Ana']
notas = [15, 18, 14, 19]

serie_notas = pd.Series(notas, index=alunos)
print(serie_notas)
```

Saída:

go Copy code

```
João     15
Maria    18
Pedro    14
Ana      19
dtype: int64
```

Explicação:

- Os índices personalizados permitem aceder aos valores diretamente

python Copy code

```python
print(serie_notas['Maria'])  # Resultado: 18
```

3. **Criar uma Series a partir de um dicionário**

```python
dados = {'Braga': 120, 'Porto': 250, 'Lisboa': 340}
serie_cidades = pd.Series(dados)
print(serie_cidades)
```

Saída:

```go
Braga     120
Porto     250
Lisboa    340
dtype: int64
```

Exemplos Práticos com DataFrame

1. **Criar um DataFrame a partir de um dicionário**

```python
dados = {
    'Nome': ['João', 'Maria', 'Pedro', 'Ana'],
    'Idade': [25, 30, 22, 28],
    'Cidade': ['Braga', 'Porto', 'Lisboa', 'Coimbra']
}

df = pd.DataFrame(dados)
print(df)
```

Saída:

```markdown
    Nome  Idade   Cidade
0   João     25    Braga
1  Maria     30    Porto
2  Pedro     22   Lisboa
3    Ana     28  Coimbra
```

Explicação:

- Cada coluna do DataFrame é uma Series.

```python
print(df['Nome'])  # Retorna a Series da coluna Nome
```

2. **Criar um DataFrame a partir de uma lista de listas**

```python
dados = [
    ['João', 25, 'Braga'],
    ['Maria', 30, 'Porto'],
    ['Pedro', 22, 'Lisboa']
]

df = pd.DataFrame(dados, columns=['Nome', 'Idade', 'Cidade'])
print(df)
```

Saída:

```markdown
    Nome  Idade   Cidade
0   João     25    Braga
1  Maria     30    Porto
2  Pedro     22   Lisboa
```

3. **Carregar dados a partir de um ficheiro CSV**

```python
# Supondo um ficheiro chamado "dados.csv" com as colunas Nome, Idade, Cidade
df = pd.read_csv('dados.csv')
print(df.head())  # Visualizar as primeiras 5 linhas
```

Explicação:

- O pandas suporta a leitura de ficheiros CSV, Excel, SQL, entre outros.

- Pode-se inspecionar rapidamente os dados carregados com funções como **.head()** e **.info().**

Operações Fundamentais com DataFrames

1. Selecionar uma coluna

```python
print(df['Nome'])
```

Saída:

```vbnet
0     João
1    Maria
2    Pedro
Name: Nome, dtype: object
```

2. Selecionar várias colunas

```python
print(df[['Nome', 'Idade']])
```

Saída:

```markdown
    Nome  Idade
0   João     25
1  Maria     30
2  Pedro     22
```

3. Selecionar linhas por índice

python Copy code

```python
# Selecionar a primeira linha
print(df.iloc[0])

# Selecionar a linha onde Nome é "Maria"
print(df[df['Nome'] == 'Maria'])
```

4. Adicionar uma nova coluna

python Copy code

```python
df['Profissão'] = ['Engenheiro', 'Médico', 'Estudante']
print(df)
```

Saída:

markdown Copy code

```
    Nome  Idade   Cidade     Profissão
0   João     25    Braga   Engenheiro
1  Maria     30    Porto       Médico
2  Pedro     22   Lisboa    Estudante
```

5. Filtrar dados com condições

python Copy code

```python
# Filtrar linhas onde Idade é maior que 25
print(df[df['Idade'] > 25])
```

Saída:

markdown Copy code

```
    Nome  Idade Cidade Profissão
1  Maria     30  Porto    Médico
```

Conclusão sobre Series e DataFrame

1. **Series:**
    - Ideal para representar colunas individuais ou conjuntos de dados simples com rótulos.
    - É frequentemente usada para manipular uma única dimensão de um dataset.
2. **DataFrame:**
    - Estrutura mais rica e flexível, adequada para análises tabulares completas.
    - Facilita operações como seleção, filtro e manipulação de múltiplas dimensões dos dados.

3. Inspecção e Limpeza de Dados

A **qualidade dos dados** é fundamental para análises confiáveis. Antes de qualquer transformação, deve-se avaliar a integridade, completude e consistência do dataset.

Processo de Inspecção

1. **Estrutura e Dimensão:**
    - Quantas linhas e colunas o dataset tem? (**df.shape**)
    - Quais são os tipos de dados em cada coluna? (**df.info()**)
2. **Resumo Estatístico:**
    - Qual é a distribuição dos valores? (**df.describe()**)
    - Existem outliers ou anomalias visíveis?
3. **Valores Ausentes:**
    - Linhas ou colunas incompletas (**df.isnull().sum()**).
4. **Coerência de Dados:**
    - Dados duplicados ou inconsistentes.
    - Formato correto em colunas (e.g., datas, números).

Tratamento de Dados Ausentes

- **Remoção:** Eliminar linhas ou colunas onde faltam valores (**.dropna()**).
- **Substituição:** Preencher valores ausentes com médias, medianas, ou valores específicos (**.fillna()**).
- **Imputação Avançada:** Utilizar algoritmos para prever os valores ausentes.

Exemplo Real:

Um dataset de vendas tem valores ausentes na coluna "Preço". Pode-se:

1. Substituir por 0 se o produto for gratuito.
2. Substituir pela média de preços.

4. Manipulação e Transformação de Dados

A manipulação de dados envolve preparar datasets para análise, reorganizando-os, combinando informações de diferentes fontes, ou resumindo-os em métricas significativas.

Ordenação

- Ordenar por uma ou mais colunas facilita a visualização de tendências ou identificação de extremos (e.g., cliente com maior volume de compras).

Combinação de Dados

1. **Concatenação:** Combina datasets empilhando linhas ou colunas (pd.concat()).
2. **Junção:** Une datasets com base numa chave comum, semelhante a SQL (**pd.merge()**).

Agrupamento e Agregação

O método .groupby() permite dividir os dados em grupos com base em valores de uma coluna, aplicar uma função de agregação (e.g., sum, mean) e obter resultados agrupados.

Exemplo Prático:
Calcular o total de vendas por região:

df.groupby('Região')['Vendas'].sum()

Transformações Avançadas

- Aplicar funções personalizadas a colunas com **.apply().**
- Criar novas colunas derivadas de outras.

5. Datas, Strings e Funcionalidades Mais Avançadas

Manipulação de Datas: Datas são frequentemente usadas em análises temporais (e.g., vendas mensais). No pandas, as colunas de texto podem ser convertidas em objetos datetime para extração de componentes como ano, mês, dia, ou cálculos de intervalos.

Exemplo:

- Coluna de texto "2023-12-20" → Objeto datetime.
- Extração: Ano = 2023, Mês = 12.

Manipulação de Strings: Colunas com texto requerem frequentemente limpeza (e.g., remover espaços, substituir caracteres). O pandas oferece métodos para manipular strings diretamente.

Exemplo:

- Substituir "n/d" por "Não disponível".
- Converter para minúsculas: df['Coluna'].str.lower().

6. Análise Exploratória e Visualização com pandas

A análise exploratória de dados (EDA) ajuda a:

1. **Entender a Distribuição:** Quais são os valores mais comuns? Existem valores atípicos?
2. **Identificar Relações:** Como variáveis se relacionam (e.g., correlação entre preço e vendas).

Visualização com pandas

Embora o pandas não seja especializado em gráficos, fornece métodos simples para criar histogramas, gráficos de barras e boxplots diretamente do DataFrame.

Exemplo Prático:

Visualizar a distribuição de idades:

```python
df['Idade'].plot(kind='hist')
```

6. Lista Geral com descrições breves das funções mais importantes da biblioteca pandas

1. Criação e Estruturas de Dados

- **pd.Series()**: Cria uma série unidimensional rotulada (similar a um array com índice).
- **pd.DataFrame()**: Cria um DataFrame bidimensional (estrutura tabular com linhas e colunas).
- **pd.read_csv() / pd.to_csv()**: Lê/escreve dados de/para ficheiros CSV.
- **pd.read_excel() / pd.to_excel()**: Lê/escreve dados de/para ficheiros Excel.
- **pd.read_sql() / pd.to_sql()**: Lê/escreve dados de/para uma base de dados SQL.

```
pd.Series()

import pandas as pd
s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
print(s)
# Resultado:
# a    1
# b    2
# c    3
pd.DataFrame()

df = pd.DataFrame({'Nome': ['João', 'Maria'], 'Idade': [30, 25]})
print(df)
# Resultado:
#    Nome  Idade
# 0  João     30
```

```
# 1 Maria     25
pd.read_csv() / pd.to_csv()

df = pd.read_csv('dados.csv')  # Lê dados de um CSV
df.to_csv('saida.csv', index=False)  # Exporta o DataFrame para um CSV
```

---

2. Visualização e Exploração

- **df.head()**: Mostra as primeiras linhas do DataFrame.
- **df.tail()**: Mostra as últimas linhas do DataFrame.
- **df.info()**: Apresenta um resumo das colunas, tipos de dados e valores nulos.
- **df.describe()**: Gera estatísticas descritivas (média, mediana, percentis, etc.) de colunas numéricas.

```
df.head()

print(df.head(2))
# Mostra as 2 primeiras linhas do DataFrame
df.tail()

print(df.tail(2))
# Mostra as 2 últimas linhas do DataFrame
df.info()

df.info()
# Mostra tipos de dados e valores não nulos em cada coluna
df.describe()

print(df.describe())
# Retorna estatísticas descritivas das colunas numéricas
```

---

3. Seleção e Indexação

- **df['coluna']**: Seleciona uma coluna.
- **df.loc[]**: Seleção baseada em rótulos (nomes de linhas/colunas).
- **df.iloc[]**: Seleção baseada em índices numéricos.
- **df.set_index()**: Define uma coluna como índice.

- **df.reset_index()**: Redefine o índice para numeração padrão.

```
df['coluna']

print(df['Nome'])
# Resultado: Coluna "Nome"
df.loc[]

print(df.loc[0])
# Retorna a linha com índice 0
df.iloc[]

print(df.iloc[1])
# Retorna a linha na posição 1
df.set_index()

df = df.set_index('Nome')
print(df)
# Define a coluna "Nome" como índice
df.reset_index()

df = df.reset_index()
print(df)
# Redefine o índice para valores numéricos padrão
```

---

4. Manipulação de Dados

- **df.sort_values()**: Ordena linhas com base em valores de colunas.
- **df.sort_index()**: Ordena com base nos índices.
- **df.drop()**: Remove linhas ou colunas.
- **df.rename()**: Renomeia colunas ou índices.
- **df.fillna()**: Preenche valores ausentes.
- **df.dropna()**: Remove linhas ou colunas com valores ausentes.

```
df.sort_values()

df = df.sort_values(by='Idade', ascending=False)
print(df)
# Ordena o DataFrame pela coluna "Idade" em ordem decrescente
df.sort_index()
```

```
df = df.sort_index()
# Ordena o DataFrame pelo índice
df.drop()

df = df.drop(columns=['Idade'])
# Remove a coluna "Idade"
df.rename()

df = df.rename(columns={'Nome': 'Primeiro Nome'})
# Renomeia a coluna "Nome"
df.fillna()

df['Idade'] = df['Idade'].fillna(0)
# Substitui valores ausentes por 0
```

---

5. Agregação e Agrupamento

- **df.groupby()**: Agrupa dados para aplicar funções de agregação (soma, média, etc.).
- **df.agg()**: Aplica múltiplas funções de agregação.
- **df.pivot_table()**: Cria tabelas dinâmicas.
- **df.cumsum()**: Calcula a soma cumulativa de colunas numéricas.

```
df.groupby()

grupo = df.groupby('Idade').size()
print(grupo)
# Conta quantos registros existem por idade
df.agg()

result = df.agg({'Idade': ['mean', 'sum']})
print(result)
# Calcula a média e a soma da coluna "Idade"
df.pivot_table()

pivot = df.pivot_table(values='Idade', index='Nome', aggfunc='mean')
print(pivot)
# Cria uma tabela dinâmica
```

---

6. Filtros e Condições

- **df[df['coluna'] > valor]**: Filtra linhas com base numa condição.
- **df.query('coluna > valor')**: Aplica filtros usando uma string de consulta.
- **df.isnull() / df.notnull()**: Identifica valores nulos/não nulos.

```
df[df['coluna'] > valor]

df_filtrado = df[df['Idade'] > 25]
print(df_filtrado)
# Filtra linhas onde Idade > 25
df.query()

df_filtrado = df.query('Idade > 25')
print(df_filtrado)
# Alternativa para filtrar linhas
df.isnull()

print(df.isnull())
# Identifica valores ausentes
```

---

7. Estatísticas e Análise

- **df.mean()**: Calcula a média de colunas numéricas.
- **df.sum()**: Calcula a soma de colunas ou linhas.
- **df.corr()**: Calcula a correlação entre colunas numéricas.
- **df.value_counts()**: Conta valores únicos numa coluna.

```
f.mean()

media = df['Idade'].mean()
print(media)
# Calcula a média da coluna "Idade"
df.sum()

soma = df['Idade'].sum()
print(soma)
# Soma todos os valores da coluna "Idade"
df.corr()

correlacao = df.corr()
```

```
print(correlacao)
# Calcula a correlação entre colunas numéricas
df.value_counts()

contagem = df['Nome'].value_counts()
print(contagem)
# Conta valores únicos na coluna "Nome"
```

---

8. Manipulação de Strings

- **df['coluna'].str.lower() / .str.upper()**: Converte texto para minúsculas/maiúsculas.
- **df['coluna'].str.contains('texto')**: Filtra linhas que contêm determinado texto.
- **df['coluna'].str.replace('a', 'b')**: Substitui substrings.

```
df['coluna'].str.lower()

df['Nome'] = df['Nome'].str.lower()
print(df)
# Converte texto para minúsculas
df['coluna'].str.contains()

print(df['Nome'].str.contains('joão'))
# Retorna True/False para nomes que contêm "joão"
df['coluna'].str.replace()

df['Nome'] = df['Nome'].str.replace('joão', 'JOÃO')
# Substitui "joão" por "JOÃO"
```

---

9. Trabalhando com Datas

- **pd.to_datetime()**: Converte strings para objetos datetime.
- **df['coluna'].dt.year**: Extrai o ano de uma coluna datetime.
- **df['coluna'].dt.dayofweek**: Obtém o dia da semana.

```
pd.to_datetime()

df['Data'] = pd.to_datetime(df['Data'])
# Converte a coluna "Data" para datetime
```

```
df['coluna'].dt.year

df['Ano'] = df['Data'].dt.year
# Extrai o ano da coluna "Data"
df['coluna'].dt.dayofweek

df['DiaSemana'] = df['Data'].dt.dayofweek
# Retorna o dia da semana (0 = segunda-feira)
```

---

10. Visualização com Pandas

- **df.plot(kind='line')**: Gera gráficos de linha.
- **df.plot(kind='bar')**: Gera gráficos de barras.
- **df.plot(kind='hist')**: Gera histogramas.

```
df.plot(kind='line')

df['Idade'].plot(kind='line')
# Gráfico de linha da coluna "Idade"
df.plot(kind='bar')

df['Idade'].value_counts().plot(kind='bar')
# Gráfico de barras para a distribuição de idades
df.plot(kind='hist')

df['Idade'].plot(kind='hist', bins=5)
# Histograma para a coluna "Idade"
```

7. Exemplo Prático: Analisar dados – Uber Reviews Without Reviewid

Análise Introdutória

Descrição do DataSet:

O DataFrame contém 12.000 linhas e 10 colunas relacionadas a análises de utilizadores sobre o Uber.

Informação Geral:

- O DataSet tem **12000 registos** e **10 colunas**.
- Algumas colunas apresentam valores ausentes:
    - **userImage:** Não possui valores preenchidos (0 non-null).
    - **replyContent** e **repliedAt:** Apenas 33 valores preenchidos (respostas a comentários).
    - **reviewCreatedVersion** e **appVersion:** Contêm valores ausentes (~1.740 registos sem preenchimento).

Colunas e Tipos de Dados:

1. **userName** (object): Nome dos utilizadores, com valores únicos.
2. **userImage** (float64): Não contém dados. Pode ser descartada.
3. **content** (object): Texto das análises dos utilizadores.
4. **score** (int64): Avaliação numérica (1 a 5).
5. **thumbsUpCount** (int64): Número de "gostos" que a análise recebeu.
6. **reviewCreatedVersion** (object): Versão da aplicação em que a análise foi criada.
7. **at** (object): Data e hora da análise.
8. **replyContent** (object): Resposta fornecida pelo Uber ao utilizador (apenas 33 análises).
9. **repliedAt** (object): Data da resposta (também com 33 valores).
10. **appVersion** (object): Versão da aplicação instalada no dispositivo.

Descrição Estatística:

Colunas Numéricas:

1. **score:**
    - Média: **3.93** (indicando análises globalmente positivas).
    - Mínimo: **1**; Máximo: **5**.
    - 50% das análises (mediana) são **5** (positivas).

2. **thumbsUpCount:**
    - Média: **0.52**, mas há outliers (máximo: **239**).
    - A maioria das análises não recebeu "gostos" (mediana: 0).

Colunas Categóricas e de Texto:

1. **userName:** Cada utilizador é único (12.000 valores únicos).
2. **content:** Texto altamente variado (8.172 valores únicos).
    - O comentário mais comum é "Good", com **985 ocorrências**.
3. **reviewCreatedVersion:** 10.260 análises especificam a versão da aplicação.
    - Versão mais frequente: **4.554.10001**, com **3.187 análises**.
4. **replyContent:** Apenas 33 análises receberam respostas personalizadas.

Colunas Temporais:

- **at:** 11.949 valores únicos (indicando quase todas as análises em momentos distintos).
- **repliedAt:** 33 respostas fornecidas em diferentes datas e horas.

Ações e Análise Potencial:

1. **Limpeza:**
    - **userImage:** Coluna sem dados. Pode ser descartada.
    - **replyContent** e **repliedAt:** Apenas 33 valores preenchidos. Analisar sua relevância.
    - **reviewCreatedVersion** e **appVersion:** Tratar os valores ausentes (e.g., preencher com "Desconhecida").
2. **Exploração de Dados:**
    - Distribuição das avaliações (**score**): Analisar percentagem de 1 a 5 estrelas.
    - Análise temporal: Identificar padrões nas datas das análises.
    - Popularidade das análises: Estudo dos "gostos" (**thumbsUpCount**).
3. **Correlação:**
    - Relação entre a avaliação (**score**) e o número de "gostos" (**thumbsUpCount**).

- Relação entre versões da aplicação e o tipo de avaliação.

4. **Texto:**
   - Análise de frequência de palavras ou frases em **content**.
   - Identificar sentimentos das análises (positivas, negativas).

1. Análise, tratamento e Limpeza dos dados do Dataset com recurso a pandas

1. Carregamento e Exploração Inicial

python Copy code

```python
df = pd.read_csv(file_path)
print(df.info())
print(df.head())
```

Porquê?

- Antes de qualquer análise, é fundamental entender a estrutura do dataset: número de linhas, colunas, tipos de dados, e valores ausentes.
- `df.info()` mostra detalhes das colunas (número de entradas não nulas, tipo de dados).
- `df.head()` exibe as primeiras 5 linhas para visualizar os dados reais e identificar possíveis problemas ou padrões.

2. Limpeza de Dados

2.1 Remover a coluna `userImage`

```python
df_cleaned = df.drop(columns=['userImage'])
```

Porquê?

- A coluna `userImage` não contém valores úteis (todos são nulos). Manter colunas irrelevantes aumenta a complexidade da análise e consome memória desnecessariamente.

2.2 Preencher valores ausentes

```python
df_cleaned.fillna({'reviewCreatedVersion': 'Desconhecida'}, inplace=True)
df_cleaned.fillna({'appVersion': 'Desconhecida'}, inplace=True)
```

Porquê?

- Colunas como `reviewCreatedVersion` e `appVersion` têm valores ausentes (~1.740 registos). Preencher com `'Desconhecida'` permite manter a integridade do dataset, evitando erros em análises futuras.
- Preencher valores ausentes é uma prática comum para evitar perda de dados durante operações (e.g., agregações, filtragem).

3. Análise Descritiva

3.1 Distribuição de avaliações (`score`)

```python
df_cleaned['score'].value_counts()
```

Porquê?

- Saber a frequência de cada pontuação (1 a 5 estrelas) ajuda a entender a perceção geral do serviço.
- Uma maior concentração em pontuações extremas (1 ou 5) pode indicar polarização nas opiniões.

3.2 Resumo de "thumbsUpCount" (gostos)

```python
df_cleaned['thumbsUpCount'].describe()
```

Porquê?

- O resumo estatístico (`describe()`) fornece informações úteis:
  - **Média:** Indica a quantidade média de "gostos" recebidos.
  - **Máximo:** Identifica outliers (e.g., análises que atraíram muita atenção).
  - **Percentis (25%, 50%, 75%):** Mostram a distribuição geral dos "gostos".

4. Distribuição Temporal das Análises

```python
df_cleaned['at'] = pd.to_datetime(df_cleaned['at'])
df_cleaned['at'].dt.date.value_counts().sort_index().head()
```

Porquê?

- Converter a coluna `at` para datetime permite análises temporais (e.g., padrões de análise por data).
- Contar o número de análises por data ajuda a identificar picos de atividade (lançamento de atualizações, eventos, ou problemas reportados em massa).

5. Resumo de Respostas

```python
df_cleaned[['replyContent', 'repliedAt']].dropna().head()
```

Porquê?

- Apenas 33 análises têm respostas do Uber. Isolar estas análises permite entender quando e como a empresa responde.
- Isto pode ser útil para avaliar a proatividade da empresa no atendimento ao cliente.

6. Correlação entre `score` e `thumbsUpCount`

```python
df_cleaned[['score', 'thumbsUpCount']].corr()
```

Porquê?

- Analisar a correlação entre a pontuação (`score`) e os "gostos" (`thumbsUpCount`) identifica relações entre a qualidade percebida (avaliação) e a interação da comunidade (gostos).
- Uma correlação positiva forte indicaria que análises bem avaliadas tendem a receber mais atenção. Neste caso, a correlação é fraca.

7. Visualização

```python
df_cleaned['score'].value_counts().plot(kind='bar', title='Distribuição de Avaliações (Sco
```

Porquê?

- Visualizar a distribuição das avaliações em gráfico de barras facilita a interpretação e comunicação de resultados.
- Um gráfico é uma forma clara de identificar tendências, como polarização em avaliações ou dominância de uma única pontuação.

8. Exportação do DataFrame Limpo

```python
df_cleaned.to_csv('/mnt/data/uber_reviews_cleaned.csv', index=False)
```

Porquê?

- Salvar o DataFrame limpo garante que futuras análises possam ser feitas sem repetir a etapa de limpeza.
- É uma boa prática salvar dados processados para evitar duplicação de esforço e manter consistência entre análises.

Os passos anteriores permitiram:

1. **Compreender a estrutura inicial do dataset.**
2. **Limpar dados irrelevantes ou ausentes para análises confiáveis.**
3. **Extrair padrões e insights a partir das métricas mais relevantes (score, gostos, respostas).**
4. **Fornecer uma base sólida para visualização e interpretações futuras.**

Código para concretizar passos anteriores:

```
import pandas as pd

# Carregar o ficheiro CSV
file_path = 'uber_reviews_without_reviewid.csv'
df = pd.read_csv(file_path)

# 1. Explorar a estrutura inicial do DataFrame
print("\nInformações do DataFrame:")
print(df.info())
print("\nPrimeiras 5 linhas do DataFrame:")
print(df.head())

# 2. Limpeza de dados
# Remover a coluna 'userImage' (todos os valores são nulos)
df_cleaned = df.drop(columns=['userImage'])

# Preencher valores ausentes nas colunas 'reviewCreatedVersion' e 'appVersion'
com 'Desconhecida'

df_cleaned.fillna({'reviewCreatedVersion': 'Desconhecida'}, inplace=True)
df_cleaned.fillna({'appVersion': 'Desconhecida'}, inplace=True)

df_cleaned['reviewCreatedVersion'].fillna('Desconhecida', inplace=True)
df_cleaned['appVersion'].fillna('Desconhecida', inplace=True)

# 3. Análise descritiva
# Contagem de valores por avaliação (score)
print("\nDistribuição das Avaliações (score):")
print(df_cleaned['score'].value_counts())

# Resumo da distribuição de "thumbsUpCount" (gostos)
print("\nResumo da Distribuição de 'thumbsUpCount':")
print(df_cleaned['thumbsUpCount'].describe())

# 4. Distribuição temporal das análises
```

```
df_cleaned['at'] = pd.to_datetime(df_cleaned['at'])  # Converter para datetime
print("\nDistribuição Temporal (primeiros 5 dias):")
print(df_cleaned['at'].dt.date.value_counts().sort_index().head())

# 5. Resumo das respostas do Uber
print("\nExemplos de Respostas do Uber (primeiros 5 registos):")
print(df_cleaned[['replyContent', 'repliedAt']].dropna().head())

# 6. Correlação entre 'score' e 'thumbsUpCount'
print("\nCorrelação entre 'score' e 'thumbsUpCount':")
print(df_cleaned[['score', 'thumbsUpCount']].corr())

# 7. Visualização de insights adicionais
# Contagem de avaliações por score
df_cleaned['score'].value_counts().plot(kind='bar',   title='Distribuição   de
Avaliações (Score)', xlabel='Score', ylabel='Contagem')

# Salvar o DataFrame limpo (se necessário)
df_cleaned.to_csv('uber_reviews_cleaned.csv', index=False)

print("\nAnálise      concluída.      DataFrame      limpo      salvo      como
'uber_reviews_cleaned.csv'.")
```

8. Exemplo Prático: Analisar dados – Shoping Trends

Descrição Detalhada do Dataset

O conjunto de dados fornecido contém informações relacionadas a compras realizadas por clientes, onde cada linha representa uma transação individual. O dataset inclui várias variáveis que descrevem as características dos clientes e das suas compras. A seguir, são descritas as colunas presentes no ficheiro:

1. **Customer ID**: Identificador único do cliente. Cada cliente tem um número distinto.
2. **Age**: Idade do cliente, uma variável numérica que representa a idade do comprador no momento da compra.
3. **Gender**: Género do cliente, categorizado como "Male" (Masculino) ou "Female" (Feminino).
4. **Item Purchased**: Nome do item comprado, que pode ser uma peça de roupa, calçado ou outros produtos.
5. **Category**: Categoria do produto comprado, como "Clothing" (Roupas), "Footwear" (Calçado), etc.
6. **Purchase Amount (USD)**: Valor gasto na compra, em dólares americanos (USD).
7. **Location**: Localização do cliente, identificada pelo estado ou região onde o cliente reside.

8. **Size**: Tamanho do artigo comprado, com categorias como "S" (pequeno), "M" (médio), "L" (grande).
9. **Color**: Cor do artigo comprado, por exemplo, "Gray", "Maroon", "Turquoise".
10. **Season**: Estação do ano em que a compra foi realizada, como "Winter" (Inverno) ou "Spring" (Primavera).
11. **Review Rating**: Avaliação dada pelo cliente ao produto, numa escala numérica (por exemplo, de 1 a 5).
12. **Subscription Status**: Status de subscrição do cliente, que pode ser "Yes" (Sim) ou "No" (Não).
13. **Payment Method**: Método de pagamento utilizado para a compra, como "Credit Card" (Cartão de Crédito), "Bank Transfer" (Transferência Bancária), "Cash" (Dinheiro), entre outros.
14. **Shipping Type**: Tipo de envio escolhido pelo cliente, por exemplo, "Express", "Free Shipping" (Envio Grátis), "Next Day Air" (Envio no Dia Seguinte).
15. **Discount Applied**: Indica se foi aplicado um desconto à compra ("Yes" ou "No").
16. **Promo Code Used**: Indica se foi utilizado um código promocional para a compra ("Yes" ou "No").
17. **Previous Purchases**: Número de compras anteriores realizadas pelo cliente.
18. **Preferred Payment Method**: Método de pagamento preferido pelo cliente (por exemplo, "Credit Card" ou "PayPal").
19. **Frequency of Purchases**: Frequência com que o cliente faz compras, com opções como "Weekly" (Semanal), "Fortnightly" (Quinzenal), "Annually" (Anualmente).

Com base nestas variáveis, podemos proceder com a análise, tratamento e visualização dos dados.

Passo 1: Limpeza de Dados

Antes de realizar qualquer análise, é essencial garantir que o dataset está limpo e pronto para processamento. A limpeza de dados pode envolver as seguintes etapas:

1. **Verificação de valores ausentes**: Detectar e tratar dados faltantes.
2. **Verificação de duplicados**: Identificar e remover linhas duplicadas, se existirem.
3. **Correção de tipos de dados**: Garantir que cada coluna está com o tipo de dado correto (por exemplo, numérico, categórico).
4. **Verificação de valores inconsistentes**: Identificar dados que possam estar incorretos ou fora de contexto.

```
# Verificando valores ausentes
missing_values = shopping_trends_df.isnull().sum()

# Verificando duplicados
duplicates = shopping_trends_df.duplicated().sum()

# Exibindo tipos de dados das colunas
```

```
data_types = shopping_trends_df.dtypes
```

Interpretação:

- **Valores ausentes**: A verificação de valores ausentes permitirá identificar colunas com dados faltantes. Se existirem valores ausentes, podemos optar por preenchê-los com valores médios, modas, ou até mesmo eliminar as linhas afetadas.
- **Duplicados**: Linhas duplicadas podem distorcer os resultados da análise, portanto, é importante removê-las.
- **Tipos de dados**: As colunas devem estar no tipo adequado para garantir a correta análise (por exemplo, "Age" deve ser numérica, "Gender" deve ser categórica).

Passo 2: Tratamento de Dados

Uma vez limpos os dados, podemos proceder com o tratamento de valores e transformar variáveis conforme necessário. Aqui estão algumas operações comuns de tratamento:

1. **Conversão de tipos de dados**: Converter colunas para o tipo adequado, como inteiros ou categorias.
2. **Criação de variáveis derivadas**: Por exemplo, criar uma nova coluna indicando a faixa etária (jovem, adulto, idoso) com base na idade.

```
# Conversão de variáveis categóricas
shopping_trends_df['Gender'] =
shopping_trends_df['Gender'].astype('category')

# Criação de faixa etária
bins = [0, 18, 40, 60, 100]
labels = ['Jovem', 'Adulto', 'Meia-Idade', 'Idoso']
shopping_trends_df['Age Group'] = pd.cut(shopping_trends_df['Age'],
bins=bins, labels=labels)
```

Passo 3: Análise Exploratória de Dados (AED)

A análise exploratória de dados (AED) tem como objetivo entender melhor a distribuição dos dados e identificar padrões ou tendências interessantes. Algumas análises importantes incluem:

1. **Distribuição de variáveis numéricas**: Verificar a distribuição da idade, quantidade de compras, valor das compras.
2. **Distribuição de variáveis categóricas**: Verificar a distribuição do género, categoria de item comprado, método de pagamento.
3. **Análise de correlações**: Verificar se existem relações significativas entre variáveis numéricas, como entre a idade e o valor da compra.

```
# Estatísticas descritivas para variáveis numéricas
numerical_summary = shopping_trends_df.describe()
```

```
# Contagem de variáveis categóricas
gender_distribution = shopping_trends_df['Gender'].value_counts()
category_distribution = shopping_trends_df['Category'].value_counts()

# Correlações entre variáveis numéricas
correlations = shopping_trends_df.corr()
```

Passo 4: Monitorização

A monitorização dos dados é uma etapa contínua em que se observa o comportamento das variáveis ao longo do tempo. Para isso, podemos analisar a frequência de compras dos clientes e o impacto de promoções ou descontos.

```
# Contagem das frequências de compra
purchase_frequency = shopping_trends_df['Frequency of
Purchases'].value_counts()

# Análise de compras com e sem desconto
discount_purchase_analysis = shopping_trends_df.groupby('Discount
Applied')['Purchase Amount (USD)'].mean()
```

Passo 5: Processamento e Agrupamento

O processamento dos dados pode envolver a agregação de variáveis para obter insights adicionais, como o gasto médio por cliente, por categoria de produto, ou por localização.

```
# Gasto médio por cliente
avg_spend_per_customer = shopping_trends_df.groupby('Customer
ID')['Purchase Amount (USD)'].mean()

# Gasto médio por categoria
avg_spend_per_category = shopping_trends_df.groupby('Category')['Purchase
Amount (USD)'].mean()
```

Passo 6: Visualizações Gráficas

A visualização gráfica é essencial para comunicar as descobertas de forma eficaz. Usaremos gráficos simples utilizando o `pandas` para ilustrar as distribuições e correlações.

```
import matplotlib.pyplot as plt

# Distribuição de idades
shopping_trends_df['Age'].hist(bins=10)
plt.title('Distribuição de Idades dos Clientes')
plt.xlabel('Idade')
plt.ylabel('Frequência')
plt.show()

# Gasto médio por categoria
avg_spend_per_category.plot(kind='bar')
plt.title('Gasto Médio por Categoria de Produto')
plt.xlabel('Categoria')
plt.ylabel('Gasto Médio (USD)')
plt.show()
```

```
# Correlação entre variáveis numéricas
correlations['Purchase Amount (USD)'].plot(kind='bar')
plt.title('Correlação com o Gasto Médio')
plt.xlabel('Variáveis')
plt.ylabel('Correlação')
plt.show()
```

Passo 7: Conclusões

Após a análise e visualização dos dados, podemos tirar algumas conclusões, como:

- Identificar a faixa etária predominante dos clientes.
- Verificar quais categorias de produtos são mais populares.
- Analisar o impacto de promoções e descontos nas compras.
- Observar se o género influencia o valor da compra ou a frequência de compras.

Esses passos são essenciais para realizar uma análise detalhada e obter insights valiosos sobre os comportamentos de compra dos clientes.

A limpeza de dados é uma etapa crítica para garantir a qualidade e a confiabilidade das análises subsequentes. Existem várias abordagens e técnicas que podem ser aplicadas para melhorar a limpeza de dados. Abaixo, explico algumas das melhores práticas que podem ser implementadas para uma limpeza de dados mais eficaz:

Podemos Ainda Acrescentar:

1. Identificação e Tratamento de Valores Ausentes

Valores ausentes (ou `NaN` - Not a Number) são comuns em muitos conjuntos de dados e podem afetar a análise. Existem diferentes maneiras de tratar valores ausentes:

- **Verificar a quantidade de dados ausentes por coluna**: Antes de tomar uma decisão sobre como tratar os valores ausentes, é importante entender qual a quantidade de dados ausentes em cada coluna.

```
# Verificar os valores ausentes em cada coluna
missing_values = shopping_trends_df.isnull().sum()

# Verificar a porcentagem de dados ausentes
missing_percentage = (shopping_trends_df.isnull().mean() * 100).round(2)
```

- **Preenchimento de valores ausentes**: Se o número de valores ausentes for baixo, pode-se preencher esses valores com uma estratégia apropriada. Aqui estão algumas opções:
    - **Preenchimento com média ou mediana (para variáveis numéricas)**: Preencher os valores ausentes com a média ou mediana da coluna pode ser uma boa solução, especialmente se os valores ausentes forem em pequena quantidade.

```
# Preencher valores ausentes em uma coluna numérica com a média
shopping_trends_df['Age'].fillna(shopping_trends_df['Age'].mean(),
inplace=True)
```

- **Preenchimento com moda (para variáveis categóricas)**: Para variáveis categóricas, podemos preencher os valores ausentes com a moda (o valor mais frequente da coluna).

```
# Preencher valores ausentes em uma coluna categórica com a moda
shopping_trends_df['Gender'].fillna(shopping_trends_df['Gender'].mode()[0],
inplace=True)
```

- **Remoção de linhas com valores ausentes**: Em casos onde os dados ausentes são numerosos ou não podem ser inferidos adequadamente, podemos eliminar as linhas com valores ausentes.

```
# Eliminar linhas com valores ausentes
shopping_trends_df.dropna(inplace=True)
```

2. Tratamento de Valores Duplicados

Linhas duplicadas podem distorcer a análise, e devem ser removidas para garantir que os dados estejam limpos. Identificar e remover duplicados é uma parte importante da limpeza de dados.

```
# Verificar duplicados no dataset
duplicates = shopping_trends_df.duplicated().sum()

# Remover duplicados
shopping_trends_df.drop_duplicates(inplace=True)
```

3. Correção de Tipos de Dados

Verificar e corrigir os tipos de dados é fundamental para evitar problemas durante a análise. Algumas colunas podem estar com o tipo incorreto (por exemplo, valores numéricos armazenados como texto). Podemos ajustar os tipos conforme necessário.

- **Converter variáveis numéricas para o tipo adequado**:

```
# Garantir que a coluna 'Purchase Amount (USD)' seja numérica
shopping_trends_df['Purchase Amount (USD)'] =
pd.to_numeric(shopping_trends_df['Purchase Amount (USD)'], errors='coerce')
```

- **Converter variáveis categóricas**:

```
# Garantir que as colunas 'Gender' e 'Subscription Status' sejam
categóricas
shopping_trends_df['Gender'] =
shopping_trends_df['Gender'].astype('category')
shopping_trends_df['Subscription Status'] =
shopping_trends_df['Subscription Status'].astype('category')
```

4. Detecção e Correção de Outliers

Outliers são valores que estão muito distantes da distribuição normal dos dados. Detectá-los e tratá-los pode melhorar a qualidade das análises. Existem várias maneiras de identificar e tratar outliers:

- **Uso de percentis**: Uma maneira comum de detectar outliers é utilizar o intervalo interquartil (IQR) para identificar valores fora do intervalo esperado.

```
# Calcular o intervalo interquartil (IQR)
Q1 = shopping_trends_df['Purchase Amount (USD)'].quantile(0.25)
Q3 = shopping_trends_df['Purchase Amount (USD)'].quantile(0.75)
IQR = Q3 - Q1

# Definir os limites inferior e superior para detectar outliers
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# Remover outliers
shopping_trends_df = shopping_trends_df[(shopping_trends_df['Purchase
Amount (USD)'] >= lower_limit) &
                                        (shopping_trends_df['Purchase
Amount (USD)'] <= upper_limit)]
```

5. Correção de Inconsistências e Erros Tipográficos

Em conjuntos de dados com texto, como a coluna **Location** ou **Item Purchased**, é comum encontrar inconsistências de grafia. Para garantir a integridade dos dados, pode ser necessário padronizar os valores.

- **Uniformizar categorias de texto**: Se houver valores inconsistentes, podemos padronizá-los (ex: "Red" e "red" devem ser tratados como iguais).

```
# Padronizar valores de texto para evitar inconsistências
shopping_trends_df['Color'] = shopping_trends_df['Color'].str.lower()
shopping_trends_df['Location'] =
shopping_trends_df['Location'].str.strip().str.lower()
```

6. Tratamento de Variáveis Categóricas

Em variáveis categóricas, é importante verificar se todas as categorias são válidas e bem definidas. Algumas opções incluem:

- **Verificar e substituir valores inválidos em variáveis categóricas**:

```
# Substituir valores inválidos ou inesperados na coluna 'Gender'
shopping_trends_df['Gender'] = shopping_trends_df['Gender'].replace({'M':
'Male', 'F': 'Female'})
```

- **Criar novas variáveis a partir de variáveis categóricas**: Podemos criar novas variáveis a partir de variáveis existentes, como a faixa etária com base na idade, ou transformar o status de subscrição em variável binária (Sim/Não).

```
# Criar variável binária para status de subscrição
shopping_trends_df['Is Subscribed'] = shopping_trends_df['Subscription
Status'].apply(lambda x: 1 if x == 'Yes' else 0)
```

7. Verificação de Consistência nas Colunas Numéricas

É importante garantir que os valores numéricos estejam dentro de um intervalo razoável. Por exemplo, verificar se o valor gasto na compra é maior que zero e se a idade do cliente está dentro de uma faixa plausível.

```
# Remover valores de compra menores que zero
shopping_trends_df = shopping_trends_df[shopping_trends_df['Purchase Amount
(USD)'] > 0]

# Garantir que a idade esteja dentro de um intervalo plausível (18 a 100
anos)
shopping_trends_df = shopping_trends_df[(shopping_trends_df['Age'] >= 18) &
(shopping_trends_df['Age'] <= 100)]
```

Melhorar a limpeza de dados é um processo contínuo que envolve a identificação de valores ausentes, duplicados, outliers e inconsistências. Uma boa prática é garantir que todos os tipos de dados estejam corretos, e que as variáveis estejam bem tratadas e consistentes. Com esses cuidados, é possível garantir que a análise e os modelos subsequentes sejam baseados em dados confiáveis e de qualidade.

A **Análise Exploratória de Dados (EAD)** é uma fase crucial em qualquer processo de análise de dados, permitindo que os analistas entendam melhor os dados antes de aplicar técnicas mais complexas de modelagem ou análise. As principais etapas de EAD são descritas a seguir de forma detalhada, considerando que o objetivo é identificar padrões, tendências e insights nos dados, além de detectar problemas como valores ausentes, outliers e inconsistências.

1. Definição dos Objetivos da Análise

Antes de iniciar qualquer exploração dos dados, é essencial definir claramente os objetivos da análise. O que se deseja descobrir ou resolver com os dados disponíveis? Alguns objetivos comuns podem incluir:

- **Identificar tendências de consumo**: Como os clientes compram os produtos? Quais são as categorias de produtos mais populares?
- **Verificar relações entre variáveis**: Existe alguma correlação entre idade e valor da compra? Ou entre o tipo de pagamento e a frequência das compras?
- **Detecção de anomalias**: Encontrar clientes com comportamentos de compra atípicos, como um gasto muito elevado ou um número excessivo de compras.

2. Importação e Preparação dos Dados

A segunda etapa envolve carregar os dados e prepará-los para a análise. Isso pode envolver:

- **Carregar os dados em um ambiente de análise**: Importação de dados de arquivos CSV, Excel, banco de dados, entre outros.
- **Verificação de tipos de dados**: Garantir que as variáveis estão corretamente tipadas (numéricas, categóricas, etc.).
- **Limpeza preliminar**: Realizar a limpeza inicial, como remover duplicados, tratar valores ausentes, corrigir erros de digitação, entre outros.

```
# Carregar dados
import pandas as pd
df = pd.read_csv(shopping_trends.csv')

# Verificar os tipos de dados
df.dtypes
```

3. Análise Descritiva

A análise descritiva visa resumir e entender a distribuição dos dados de uma maneira geral, para fornecer uma visão inicial dos mesmos. Algumas técnicas incluem:

- **Estatísticas descritivas**: Cálculo de medidas como média, mediana, desvio padrão, mínimos e máximos.

```
# Estatísticas descritivas
df.describe()
```

- **Distribuição das variáveis**: Verificar a distribuição das variáveis numéricas (histogramas, boxplots) e categóricas (contagem de valores únicos).

```
# Visualizar a distribuição de uma variável numérica
df['Purchase Amount (USD)'].hist()

# Visualizar a distribuição de uma variável categórica
df['Category'].value_counts()
```

- **Verificação de valores ausentes**: Determinar quais variáveis têm valores ausentes e em que proporção.

```
# Verificar valores ausentes
df.isnull().sum()
```

4. Identificação de Outliers

Outliers são valores que estão fora do padrão esperado e podem distorcer as análises. É importante identificá-los e decidir se devem ser removidos ou tratados. As principais abordagens incluem:

- **Uso de boxplots**: Para detectar outliers em variáveis numéricas.

```
# Boxplot para detectar outliers
import seaborn as sns
sns.boxplot(x=df['Purchase Amount (USD)'])
```

- **Análise com o intervalo interquartil (IQR)**: Detectar outliers usando a diferença entre os quartis.

```
# Calcular IQR
Q1 = df['Purchase Amount (USD)'].quantile(0.25)
Q3 = df['Purchase Amount (USD)'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# Filtrar outliers
df = df[(df['Purchase Amount (USD)'] >= lower_limit) & (df['Purchase Amount
(USD)'] <= upper_limit)]
```

5. Detecção de Correlações

Após a análise descritiva, é essencial entender como as variáveis estão relacionadas entre si. A análise de correlação ajuda a identificar relações lineares entre variáveis numéricas. As correlações podem ser visualizadas por meio de:

- **Matriz de correlação**: Exibição das correlações entre todas as variáveis numéricas.

```
# Matriz de correlação
correlation_matrix = df.corr()
print(correlation_matrix)
```

- **Heatmap**: Um gráfico de calor pode ser usado para visualizar as correlações de forma mais intuitiva.

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

# Heatmap de correlação
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

6. Análise de Variáveis Categóricas

Além das variáveis numéricas, é importante analisar as variáveis categóricas para entender a distribuição de classes e as possíveis relações com outras variáveis. Algumas técnicas incluem:

- **Contagem de frequências**: Para variáveis como gênero, categoria do produto, status de subscrição, etc.

```
# Frequência de uma variável categórica
df['Gender'].value_counts()
```

- **Gráficos de barras**: Para visualização das distribuições das variáveis categóricas.

```
# Gráfico de barras para uma variável categórica
df['Category'].value_counts().plot(kind='bar')
```

- **Tabela de contingência**: Para verificar a relação entre duas variáveis categóricas.

```
# Tabela de contingência entre 'Gender' e 'Subscription Status'
pd.crosstab(df['Gender'], df['Subscription Status'])
```

7. Transformações de Dados

Durante a EAD, é comum aplicar transformações nos dados para facilitar a análise e melhorar a performance dos modelos futuros:

- **Normalização e padronização**: Caso haja variáveis com escalas muito diferentes (por exemplo, idade e valor da compra), pode ser necessário normalizar ou padronizar essas variáveis.

```
from sklearn.preprocessing import StandardScaler

# Padronizar a coluna 'Purchase Amount (USD)'
scaler = StandardScaler()
df['Purchase Amount (USD)'] = scaler.fit_transform(df[['Purchase Amount
(USD)']])
```

- **Criação de novas variáveis**: A criação de variáveis derivadas pode ser útil para a análise. Por exemplo, a criação de uma coluna que classifique os clientes em diferentes faixas de idade.

```
# Criação de faixas etárias
bins = [0, 18, 40, 60, 100]
labels = ['Jovem', 'Adulto', 'Meia-Idade', 'Idoso']
df['Age Group'] = pd.cut(df['Age'], bins=bins, labels=labels)
```

8. Visualizações Avançadas

Além de visualizar distribuições e correlações, a visualização avançada pode ser usada para identificar padrões mais complexos, como:

- **Gráficos de dispersão**: Para entender a relação entre duas variáveis numéricas.

```
# Gráfico de dispersão entre 'Age' e 'Purchase Amount (USD)'
df.plot(kind='scatter', x='Age', y='Purchase Amount (USD)')
plt.show()
```

- **Gráficos de linhas ou séries temporais**: Se os dados incluírem uma variável temporal, podemos observar as tendências ao longo do tempo.

```
# Gráfico de série temporal (exemplo com data)
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df['Purchase Amount (USD)'].resample('M').sum().plot()
plt.show()
```

9. Conclusões Preliminares

Após a análise exploratória de dados, é importante resumir as principais descobertas e insights. Isso inclui:

- **Padrões encontrados**: Como as variáveis estão distribuídas? Existe alguma tendência óbvia, como aumento no gasto durante uma determinada estação?
- **Correlação entre variáveis**: Quais variáveis estão fortemente correlacionadas? Quais não apresentam relação significativa?
- **Outliers e anomalias**: Quais dados podem estar distorcendo as análises?
- **Transformações necessárias**: Quais variáveis precisam ser transformadas ou criadas para melhorar a análise?

Apresenta-se de seguida todo o código utilizado neste exemplo:

```
# Importação das bibliotecas necessárias
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Passo 1: Carregar os Dados
# Carregar o arquivo CSV
df = pd.read_csv('shopping_trends.csv')

# Passo 2: Verificação de Tipos de Dados
# Exibir os tipos de dados para garantir que as colunas estão corretamente
tipadas
print("Tipos de dados:\n", df.dtypes)

# Passo 3: Verificação de Valores Ausentes
# Identificar se há valores ausentes em qualquer coluna
missing_values = df.isnull().sum()
print("\nValores ausentes por coluna:\n", missing_values)

# Verificar a porcentagem de valores ausentes
missing_percentage = (df.isnull().mean() * 100).round(2)
print("\nPorcentagem de valores ausentes por coluna:\n", missing_percentage)

# Passo 4: Verificação de Duplicados
```

```
# Identificar e remover linhas duplicadas
duplicates = df.duplicated().sum()
print("\nNúmero de linhas duplicadas:", duplicates)

# Remover duplicados
df.drop_duplicates(inplace=True)

# Passo 5: Conversão de Variáveis Categóricas para Variáveis Dummy (One-Hot
Encoding)
# As colunas 'Gender' e 'Category' são categóricas, então vamos convertê-las
para variáveis dummy
df = pd.get_dummies(df, columns=['Gender', 'Category'], drop_first=True)

# Passo 6: Estatísticas Descritivas
# Exibir estatísticas descritivas para variáveis numéricas
print("\nEstatísticas descritivas:\n", df.describe())

# Passo 7: Visualizações - Distribuição de Variáveis Numéricas
# Visualizar a distribuição da variável 'Purchase Amount (USD)'
plt.figure(figsize=(10, 6))
df['Purchase Amount (USD)'].hist(bins=20, color='skyblue', edgecolor='black')
plt.title('Distribuição do Valor das Compras (USD)')
plt.xlabel('Valor da Compra (USD)')
plt.ylabel('Frequência')
plt.show()

# Passo 8: Visualizações - Boxplot para Identificar Outliers
# Boxplot para a variável 'Purchase Amount (USD)' para identificar outliers
plt.figure(figsize=(10, 6))
sns.boxplot(x=df['Purchase Amount (USD)'], color='orange')
plt.title('Boxplot do Valor das Compras (USD)')
plt.xlabel('Valor da Compra (USD)')
plt.show()

# Passo 9: Análise de Correlação entre Variáveis Numéricas
# Filtrando apenas as colunas numéricas
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Calcular a matriz de correlação para as variáveis numéricas
correlation_matrix = numeric_df.corr()
print("\nMatriz de Correlação:\n", correlation_matrix)

# Visualização da Matriz de Correlação com Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matriz de Correlação entre Variáveis Numéricas')
```

```
plt.show()

# Passo 10: Verificação de Outliers Usando Intervalo Interquartil (IQR)
# Detectar e remover outliers na variável 'Purchase Amount (USD)' utilizando o
IQR
Q1 = df['Purchase Amount (USD)'].quantile(0.25)
Q3 = df['Purchase Amount (USD)'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# Remover outliers
df = df[(df['Purchase Amount (USD)'] >= lower_limit) & (df['Purchase Amount
(USD)'] <= upper_limit)]

# Passo 11: Criação de Faixas Etárias (Novo Grupo)
# Criar uma nova variável para faixa etária (faixa etária por grupos)
bins = [0, 18, 40, 60, 100]
labels = ['Jovem', 'Adulto', 'Meia-Idade', 'Idoso']
df['Age Group'] = pd.cut(df['Age'], bins=bins, labels=labels)

# Visualizar a nova variável 'Age Group'
print("\nDistribuição das faixas etárias:\n", df['Age Group'].value_counts())

# Passo 12: Visualização das Frequências de Compra
# Exibir a distribuição das frequências de compra dos clientes
purchase_frequency = df['Frequency of Purchases'].value_counts()
print("\nDistribuição das Frequências de Compras:\n", purchase_frequency)

# Gráfico de barras para mostrar a distribuição de frequências de compras
plt.figure(figsize=(10, 6))
purchase_frequency.plot(kind='bar', color='lightgreen', edgecolor='black')
plt.title('Distribuição das Frequências de Compras')
plt.xlabel('Frequência de Compras')
plt.ylabel('Número de Clientes')
plt.show()

# Passo 13: Identificação de Relação entre 'Age' e 'Purchase Amount (USD)'
# Gráfico de dispersão entre 'Age' e 'Purchase Amount (USD)'
plt.figure(figsize=(10, 6))
plt.scatter(df['Age'], df['Purchase Amount (USD)'], alpha=0.5, color='blue')
plt.title('Relação entre Idade e Valor da Compra (USD)')
plt.xlabel('Idade')
plt.ylabel('Valor da Compra (USD)')
plt.show()
```

```python
# Passo 14: Verificação de Valores Ausentes Após Limpeza
# Recalcular valores ausentes após os tratamentos
missing_values_after_cleaning = df.isnull().sum()
print("\nValores ausentes após limpeza:\n", missing_values_after_cleaning)

# Passo 15: Verificação de Duplicados Após Limpeza
# Verificar se há duplicados após a remoção
duplicates_after_cleaning = df.duplicated().sum()
print("\nNúmero de linhas duplicadas após limpeza:",
duplicates_after_cleaning)
```