
# Documentação projeto04 Laboratória

Equipe: Ana virgínia Morais e Juliana Marinho Xavier

### Introdução

> Este relatório apresenta os resultados de uma análise exploratória e de validação de hipóteses sobre um conjunto de dados de produtos e avaliações da Amazon. O objetivo principal foi investigar a relação entre descontos, popularidade e a qualidade percebida dos produtos.

**Links:**

- **Apresentação:**https://docs.google.com/presentation/d/1z9FMRw8Uh7r-ALzcFmke2FsVNmVD1QkRCeu-M0mREB0/edit?slide=id.p#slide=id.p
- **Google Colab:** https://colab.research.google.com/drive/1iCQgxWGuhup4gdUeKhO9YPAIYr1l6rAQ#scrollTo=DThDoJ6Q30UC
- **Dashboard:**https://lookerstudio.google.com/reporting/c79f26b0-1416-4216-ade1-0e9d1f8ef07b/page/9VKYF

## 1. Ferramentas, Linguagens e Insumos

### 1.1 Ferramentas e/ou plataformas

Neste projeto você pode escolher com quais ferramentas de AI irá trabalhar. Porém, recomendamos o uso do Google Colab e Gemini em conjunto, desta forma você poderá revisar e alterar os códigos sugeridos pelo Gemini na mesma tela.

Ferramentas disponíveis:

- Google Colab ou outro notebook
- Apresentações Google
- Como Inteligências Artificiais recomendamos e Gemini ou ChatGPT

### 1.2 Linguagens

Este projeto será desenvolvido em Python.

### 1.3 Bases de dados

Neste projeto você poderá escolher qual conjunto de dados analisar. O objetivo da escolha da base de dados é que neste projeto você se sinta mais livre para explorar os dados, definir o objetivo e escolher um conjunto de dados com um tópico que você acha mais interessante para desenvolver o projeto.

É muito importante ter em conta, ao escolher uma base de dados, que cada conjunto possui características e complexidades próprias. Leia atentamente a descrição das variáveis e resumo do conjunto de dados antes de tomar uma decisão.

### Opção 2: Amazon sales

Este conjunto de dados contém dados de mais de 1.000 classificações e análises de produtos disponíveis para venda na Amazon.

Amazon é uma empresa americana de tecnologia com operações multinacionais, cujos interesses comerciais incluem comércio eletrônico, para o qual compram e armazenam estoque, e cuidam de todo o processo, desde a

precificação até o envio, atendimento ao cliente e devoluções.

Os dados vêm de um repositório no Kaggle.

**Descrição das variáveis:**

**Tabela amazon_product**

- product_id: ID do produto
- product_name: Nome do produto
- category: Categoria do produto
- discounted_price: Preço com desconto do produto
- actual_price: Preço real do produto
- discount_percentage: Porcentagem de desconto do produto
- about_product: Descrição sobre o produto

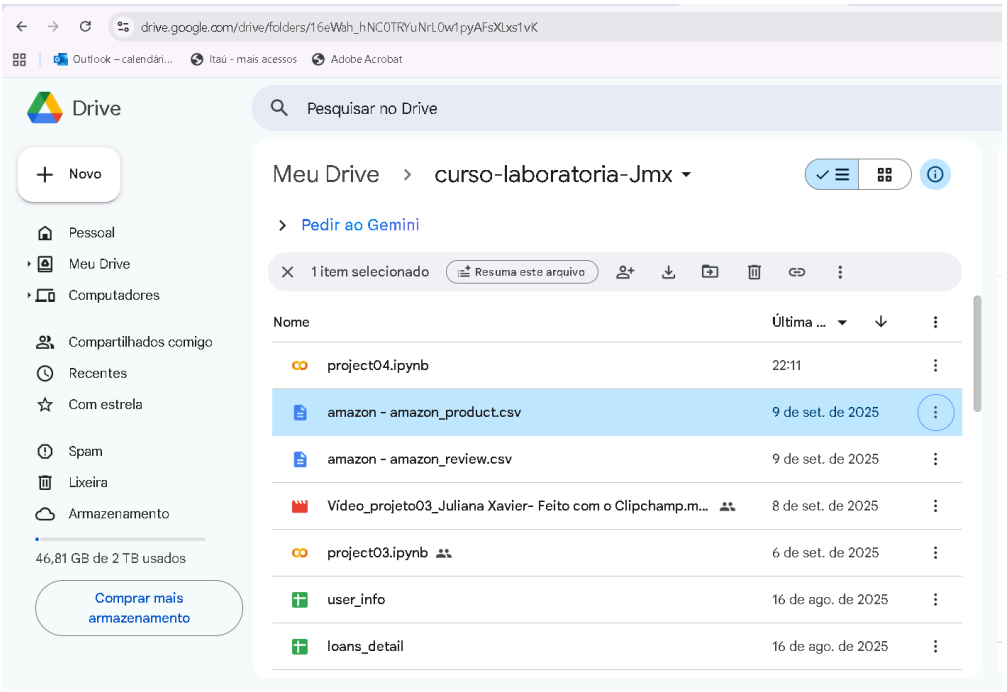
**Tabela amazon_review**

- user_id: ID do usuário que escreveu a avaliação do produto
- user_name: Nome do usuário que escreveu a avaliação do produto
- Review_id: ID da avaliação do usuário
- Review_title: Breve avaliação do usuário
- Review_content: Avaliação completa do usuário
- Img_link: Link da imagem do produto
- Product_link: Link para o site oficial do produto
- Product_id: ID do produto
- Rating: Classificação do produto
- Rating_count: Número de pessoas que votaram na classificação da Amazon.

### 🟦 4.1 Processar e preparar a base de dados

4.1.1 🔵 Conectar/importar dados para outras ferramentas

Subi os arquivos no drive na pasta da curso e criei uma colab para o projeto:

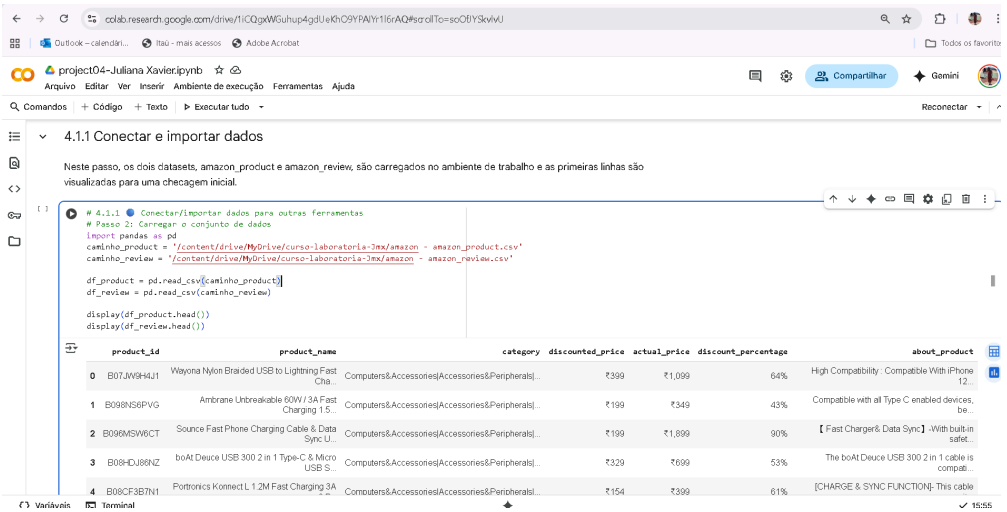



O projeto iniciou com a instalação das bibliotecas necessárias, incluindo `google-cloud-bigquery`, `seaborn`, `matplotlib`, `pandas`, `numpy`, `langdetect` e `unidecode`.

**1. Carregamento dos Dados (Etapa 4.1.1)**

Os dois conjuntos de dados principais, `amazon_product.csv` (informações de produto) e `amazon_review.csv` (informações de avaliação), foram carregados no ambiente de trabalho.

- O DataFrame de produtos (`df_product`) iniciou com 1469 entradas e 7 colunas.
- O DataFrame de avaliações (`df_review`) iniciou com 1465 entradas e 10 colunas.




### 4.1.2 🔵 Identificar e Tratar Valores Nulos

A estratégia de tratamento foi preencher valores ausentes em colunas numéricas com a **mediana** e em colunas categóricas com a **moda** (valor mais frequente).

| Coluna | Tratamento | Valor Usado | Valores Nulos Após Conversão |
|---|---|---|---|
| `rating` (df_review) | Mediana | 4.10 | 1 |
| `rating_count` (df_review) | Mediana | 5179.00 | 2 |

| | | | |
|---|---|---|---|
| `about_product` (df_product) | Moda | '[CHARGE & SYNC FUNCTION]...' | 4 |
| `img_link` e `product_link` (df_review) | Moda | Moda específica da coluna | 466 (em cada) |

Após o tratamento, todas as colunas passaram a ter zero valores nulos.

```
# 4.1.2 🔵 Identificar e Tratar Valores Nulos
# Importar a biblioteca pandas
import pandas as pd

# Supondo que seus DataFrames 'df_product' e 'df_review' já foram carregados

# Colocar os DataFrames em uma lista para iterar sobre eles
dataframes = {'df_product': df_product, 'df_review': df_review}

print("--- Identificando e Tratando Valores Nulos em Ambas as Tabelas ---")
print("-" * 60)

for nome, df in dataframes.items():
    print(f"Iniciando tratamento de valores nulos para: {nome}")
    print("Quantidade de valores nulos por coluna antes do tratamento:")
    print(df.isnull().sum())

    print("\nTratando valores nulos...")

    # Identificar colunas numéricas
    colunas_numericas = df.select_dtypes(include=['float64', 'int64']).columns
    # Identificar colunas de texto/categóricas
    colunas_categoricas = df.select_dtypes(include=['object']).columns

    # Substituir nulos em colunas numéricas pela mediana
    for coluna in colunas_numericas:
        if df[coluna].isnull().sum() > 0:
            mediana = df[coluna].median()
            df[coluna] = df[coluna].fillna(mediana)
            print(f"Valores nulos na coluna '{coluna}' foram preenchidos com a mediana: {mediana:.2f}")

    # Substituir nulos em colunas categóricas pela moda
    for coluna in colunas_categoricas:
        if df[coluna].isnull().sum() > 0:
            moda = df[coluna].mode()[0]
            df[coluna] = df[coluna].fillna(moda)
            print(f"Valores nulos na coluna '{coluna}' foram preenchidos com a moda: '{moda}'")

    print("\nVerificação de valores nulos após o tratamento:")
    print(df.isnull().sum())
    print("-" * 60)
```

### 4.1.3 🔵 Identificar e Tratar Valores Duplicados

2.3. Identificação e Tratamento de Valores Duplicados (Etapa 4.1.3)

As linhas duplicadas foram removidas para evitar a distorção dos resultados.

- **df_product:** As duplicatas foram removidas com base no `product_id`. **Resultado:** 118 linhas duplicadas removidas, resultando em 1351 linhas.

- **df_review:** As duplicatas foram removidas, mantendo-se a linha com a maior contagem de valores preenchidos. **Resultado:** 114 avaliações duplicadas removidas, resultando em 1351 linhas.

```
# 4.1.3 🔵 Identificar e Tratar Valores Duplicados
# Supondo que seus DataFrames 'df_product' e 'df_review' já foram carregados

# Colocar os DataFrames em um dicionário para iterar sobre eles
dataframes = {'df_product': df_product, 'df_review': df_review}

print("--- Identificando e Removendo Valores Duplicados em Ambas as Tabelas ---")
print("-" * 60)

for nome, df in dataframes.items():
    print(f"Iniciando o tratamento de duplicatas para: {nome}")

    # Lógica condicional para o tratamento correto
    if nome == 'df_product':
        # Para df_product, o 'product_id' é o identificador único.
        num_duplicatas = df.duplicated(subset=['product_id']).sum()

        if num_duplicatas > 0:
            df_sem_duplicatas = df.drop_duplicates(subset=['product_id'])
            dataframes[nome] = df_sem_duplicatas
            print(f"Duplicatas removidas com sucesso com base no 'product_id'. {num_duplicatas} linha(s) remov
ida(s).")
            print(f"O dataset agora tem {len(dataframes[nome])} linhas.")
        else:
            print("Nenhuma duplicata significativa encontrada na coluna 'product_id'.")

    else: # df_review
        # 1. Contar o número de valores não nulos por linha
        df['non_null_count'] = df.count(axis=1)

        # 2. Ordenar o DataFrame por 'product_id' e, em seguida, por 'non_null_count' de forma decrescente
        df_sorted = df.sort_values(
            by=['product_id', 'non_null_count'],
            ascending=[True, False])

        # 3. Remover duplicatas, mantendo a linha com a maior contagem de valores preenchidos
        df_sem_duplicatas = df_sorted.drop_duplicates(subset=['product_id'], keep='first')

        # Remover a coluna auxiliar de contagem
        df_sem_duplicatas = df_sem_duplicatas.drop(columns=['non_null_count'])

        # Contar quantas linhas foram removidas
        linhas_removidas = len(df) - len(df_sem_duplicatas)

        if linhas_removidas > 0:
            dataframes[nome] = df_sem_duplicatas
            print(f"Foram encontradas {linhas_removidas} avaliações duplicadas de produtos.")
            print(f"Duplicatas removidas com sucesso. O dataset agora tem {len(dataframes[nome])} linhas, uma
por produto.")
        else:
            print("Nenhuma duplicata significativa encontrada para remover na coluna 'product_id'.")
```

```
    print("-" * 60)

# Reatribuir os DataFrames originais após o loop
df_product = dataframes['df_product']
df_review = dataframes['df_review']
```

#### 4.1.4 🔵 Identificar e Tratar Dados Fora do Escopo de Análise

O método de padronização foi aplicado a todas as colunas de texto para garantir consistência, convertendo-as para **minúsculas** e removendo espaços em branco ( `.str.lower().str.strip()` ).

- **Criação de main_category:** Para simplificar a análise, a coluna `main_category` foi criada extraindo apenas a primeira categoria da string complexa na coluna `category` (separada por '|'). As categorias principais identificadas incluíam `computers&accessories` , `electronics` , `home&kitchen` , entre outras.

```
# 4.1.4 🔵 Identificar e Tratar Dados Fora do Escopo de Análise

# Supondo que seus DataFrames 'df_product' e 'df_review' já foram carregados e tratados

# Colocar os DataFrames em um dicionário para iterar sobre eles
dataframes = {'df_product': df_product, 'df_review': df_review}

print("--- Identificando e Tratando Discrepâncias Categóricas em Ambas as Tabelas ---")
print("-" * 60)

for nome, df in dataframes.items():
    print(f"Iniciando a padronização de dados para: {nome}")

    # Identificar colunas de texto/categóricas
    colunas_categoricas = df.select_dtypes(include=['object']).columns

    # Aplicar padronização (minúsculas e remoção de espaços)
    for coluna in colunas_categoricas:
        if df[coluna].dtype == 'object':
            df[coluna] = df[coluna].str.lower().str.strip()

    # Mostrar um exemplo de padronização
    if nome == 'df_product':
        print("\nExemplo de padronização na coluna 'category':")
        # Mostrar os valores únicos da coluna 'category'
        print(df['category'].unique()[:5])
    elif nome == 'df_review':
        print("\nExemplo de padronização na coluna 'review_title':")
        # Mostrar os valores únicos da coluna 'review_title'
        print(df['review_title'].unique()[:5])

    print("-" * 60)

# Reatribuir os DataFrames originais após o loop (para garantir que as mudanças sejam salvas)
df_product = dataframes['df_product']
df_review = dataframes['df_review']
```

Para permitir cálculos e análises numéricas, as colunas de preço e avaliação foram convertidas para tipos de

dados numéricos. O código removeu caracteres de moeda (`₹`), vírgulas (`,`) e sinais de porcentagem (`%`) antes de converter as colunas `discounted_price`, `actual_price`, `discount_percentage`, `rating` e `rating_count` para `float64` ou `int64`

#### 4.1.5 🔵 Identificar e Tratar Dados Discrepantes em Variáveis Categóricas

O código garantiu a limpeza e a consistência dos dados textuais.

- **Padronização:** Todas as colunas de texto foram convertidas para minúsculas e tiveram espaços em branco no início e no fim removidos. Isso unificou categorias com grafias diferentes, mas com o mesmo significado.
- **Extração de Categorias:** Para lidar com a coluna `category`, que continha várias categorias separadas por '|', foram criadas duas novas colunas:
  - `main_category`: Contém apenas a primeira categoria da string (ex: 'computers&accessories').
  - `produto`: Contém apenas a última categoria da string (ex: 'usbcables').

**Comentário sobre o resultado:** A padronização foi bem-sucedida, e a criação das colunas `main_category` e `produto` simplifica a análise de categorias. A saída mostra exemplos dos valores únicos, como 'computers&accessories' e 'usbcables', que agora estão prontos para visualização e agrupamento.

```
# 4.1.5 🔵 Identificar e Tratar Dados Discrepantes em Variáveis Categóricas
# Importar bibliotecas necessárias
import pandas as pd
import numpy as np

# Supondo que seus DataFrames 'df_product' e 'df_review' já foram carregados

# Colocar os DataFrames em um dicionário para iterar sobre eles
dataframes = {'df_product': df_product, 'df_review': df_review}

print("--- Identificando e Tratando Discrepâncias Categóricas em Ambas as Tabelas ---")
print("-" * 60)

for nome, df in dataframes.items():
    print(f"Iniciando a padronização de dados para: {nome}")

    # Identificar colunas de texto/categóricas
    colunas_categoricas = df.select_dtypes(include=['object']).columns

    # Aplicar padronização (minúsculas e remoção de espaços)
    for coluna in colunas_categoricas:
        if df[coluna].dtype == 'object':
            df[coluna] = df[coluna].str.lower().str.strip()

    # --- Nova etapa: Criar as colunas 'main_category' e 'produto' a partir de 'category'
    if nome == 'df_product':
        df['main_category'] = df['category'].str.split('|').str[0].str.strip()
        print("\nNova coluna 'main_category' criada a partir da coluna 'category'.")

        # --- Linha de código adicionada para criar a coluna 'produto' ---
        df['produto'] = df['category'].str.split('|').str[-1].str.strip()
        print("Nova coluna 'produto' criada a partir da coluna 'category'.")


    print("Padronização concluída.")
```

```
    # Exemplo de verificação de padronização
    if nome == 'df_product':
        print("\nExemplo de valores únicos na nova coluna 'main_category':")
        print(df['main_category'].unique()[:5])
        print("\nExemplo de valores únicos na nova coluna 'produto':")
        print(df['produto'].unique()[:5])
    elif nome == 'df_review':
        print("\nExemplo de valores únicos na coluna 'review_title':")
        print(df['review_title'].unique()[:5])

    print("-" * 60)

# Reatribuir os DataFrames originais após o loop
df_product = dataframes['df_product']
df_review = dataframes['df_review']
```

#### 4.1.7 🔵 Verificar e Alterar o Tipo de Dados

2.1. Verificação e Alteração do Tipo de Dados (Etapa 4.1.7)

Esta etapa garantiu que as colunas numéricas estivessem no formato correto. Valores não numéricos foram tratados e convertidos para `NaN` antes de serem preenchidos.

- **df_product:** As colunas `discounted_price`, `actual_price` e `discount_percentage` tiveram símbolos de moeda (₹) e vírgulas removidos, e foram convertidas para `float64` ou `int64`.
- **df_review:** As colunas `rating` e `rating_count` foram convertidas para `float64` após a remoção de vírgulas.

```
# 4.1.7 🔵 Verificar e Alterar o Tipo de Dados
# Importar bibliotecas necessárias
import pandas as pd
import numpy as np

# Supondo que seus DataFrames 'df_product' e 'df_review' já foram carregados

print("--- Realizando a Conversão de Tipos de Dados em Ambas as Tabelas ---")
print("-" * 60)

# Colocar os DataFrames em um dicionário para iterar sobre eles
dataframes = {'df_product': df_product, 'df_review': df_review}

for nome, df in dataframes.items():
    print(f"Iniciando a conversão de tipos para: {nome}")

    # Processar colunas de preço (apenas em df_product)
    if nome == 'df_product':
        # discounted_price
        if df['discounted_price'].dtype != 'object':
            df['discounted_price'] = df['discounted_price'].astype(str)
        df['discounted_price'] = pd.to_numeric(
            df['discounted_price'].str.replace('₹', '', regex=False).str.replace(',', '', regex=False),
            errors='coerce'
        )

        # actual_price
        if df['actual_price'].dtype != 'object':
```

```
            df['actual_price'] = df['actual_price'].astype(str)
        df['actual_price'] = pd.to_numeric(
            df['actual_price'].str.replace('₹', '', regex=False).str.replace(',', '', regex=False),
            errors='coerce'
        )

        # discount_percentage
        if df['discount_percentage'].dtype != 'object':
            df['discount_percentage'] = df['discount_percentage'].astype(str)
        df['discount_percentage'] = pd.to_numeric(
            df['discount_percentage'].str.replace('%', '', regex=False),
            errors='coerce'
        )

    # Processar colunas de avaliação (apenas em df_review)
    elif nome == 'df_review':
        # rating
        df['rating'] = pd.to_numeric(df['rating'], errors='coerce')

        # rating_count
        if df['rating_count'].dtype != 'object':
            df['rating_count'] = df['rating_count'].astype(str)
        df['rating_count'] = pd.to_numeric(
            df['rating_count'].str.replace(',', '', regex=False),
            errors='coerce'
        )

    print(f"\nVerificação de tipos de dados para {nome}:")
    print(df.info())
    print("-" * 60)

# Reatribuir os DataFrames originais após o loop
df_product = dataframes['df_product']
df_review = dataframes['df_review']

print("Conversão de tipos de dados concluída com sucesso!")
print(f"Total de valores nulos após conversão:")
print(f"df_product - discounted_price: {df_product['discounted_price'].isnull().sum()}")
print(f"df_product - actual_price: {df_product['actual_price'].isnull().sum()}")
print(f"df_product - discount_percentage: {df_product['discount_percentage'].isnull().sum()}")
print(f"df_review - rating: {df_review['rating'].isnull().sum()}")
print(f"df_review - rating_count: {df_review['rating_count'].isnull().sum()}")
```

Por sugestão do gemini, faremos essa etapa antes do tratamento dos outliers e merge das tabelas.

Esta é uma etapa crucial que garante que todas as colunas estejam com o tipo de dado correto, permitindo que análises numéricas e estatísticas sejam realizadas sem erros.

Colunas de preço, porcentagem de desconto e avaliações são convertidas para o tipo float.

Valores não numéricos são tratados e convertidos para NaN antes de serem preenchidos.

#### 4.1.6 🔵 Identificar e Tratar Dados Discrepantes em Variáveis Numéricas

O método do **Intervalo Interquartílico (IQR)** foi utilizado para identificar valores discrepantes, pois é robusto a valores extremos. Apesar de termos outliers, optamos por não realizar ajustes por orientação do Gemini, devido aos dados serem uma visão real e poderíamos distorcer corrigindo.

| Coluna | IQR | Limites (Inferior, Superior) | Número de Outliers |
|---|---|---|---|
| `discounted_price` (df_product) | 1825.00 | -2388.50, 4911.50 | **209** |
| `actual_price` (df_product) | 3676.00 | -4615.00, 10089.00 | **185** |
| `discount_percentage` (df_product) | 31.00 | -15.50, 108.50 | **0** |
| `rating` (df_review) | 0.40 | 3.30, 4.90 | **19** |
| `rating_count` (df_review) | 14888.00 | -21225.00, 38327.00 | **130** |

```
# Importar bibliotecas
import pandas as pd
import numpy as np

# Supondo que seus DataFrames 'df_product' e 'df_review' já foram carregados e tratados

# Colocar os DataFrames em um dicionário para iterar sobre eles
dataframes = {'df_product': df_product, 'df_review': df_review}

print("--- Identificando Outliers em Variáveis Numéricas ---")
print("-" * 60)

for nome, df in dataframes.items():
    print(f"Iniciando a análise de outliers para: {nome}")

    # Identificar colunas que devem ser tratadas como numéricas para outliers
    if nome == 'df_product':
        colunas_para_tratar = ['discounted_price', 'actual_price', 'discount_percentage']
    else: # df_review
        colunas_para_tratar = ['rating', 'rating_count']

    for coluna in colunas_para_tratar:
        # Converter a coluna para numérico temporariamente para o cálculo do IQR
        temp_col = pd.to_numeric(df[coluna], errors='coerce')

        # Calcular o 1º e 3º quartil (Q1 e Q3)
        Q1 = temp_col.quantile(0.25)
        Q3 = temp_col.quantile(0.75)

        # Calcular o Intervalo Interquartílico (IQR)
        IQR = Q3 - Q1

        # Definir os limites inferior e superior
        limite_inferior = Q1 - 1.5 * IQR
        limite_superior = Q3 + 1.5 * IQR

        # Identificar e contar os outliers
        outliers_count = len(temp_col[(temp_col < limite_inferior) | (temp_col > limite_superior)])

        # Relatório Detalhado
        print(f"\nDetalhes para a coluna: '{coluna}'")
        print(f"  > Q1: {Q1:.2f}, Q3: {Q3:.2f}, IQR: {IQR:.2f}")
        print(f"  > Limite Inferior: {limite_inferior:.2f}, Limite Superior: {limite_superior:.2f}")
```

```
        print(f"  > Número de Outliers: {outliers_count}")

        # Mensagem final
        if outliers_count > 0:
            print(f"  > Foram encontrados {outliers_count} outliers na coluna '{coluna}'.")
        else:
            print(f"  > Nenhum outlier significativo encontrado na coluna '{coluna}'.")

    print("-" * 60)
```

#### 4.1.7 🔵 Criar novas variáveis

Duas novas colunas fundamentais para a análise foram criadas:

1. **preco_categoria (em df_product):** Categoriza os produtos em faixas de preço (Baixo, Medio, Alto, Muito Alto) baseadas nos quartis da coluna `discounted_price`.

2. **score_sentimento (em df_review):** Aplica análise de sentimento (usando VADER, após filtragem para inglês, limpeza de texto e lematização) ao conteúdo das avaliações (`review_content`), atribuindo uma pontuação de -1 (negativo) a 1 (positivo).

```
# Importar as bibliotecas necessárias
import pandas as pd
import numpy as np
from textblob import TextBlob
import re
from langdetect import detect
import unidecode
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.sentiment import SentimentIntensityAnalyzer
# Removido importação duplicada de pandas

# Certifique-se de que os downloads foram feitos
nltk.download('vader_lexicon', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)

# Supondo que seus DataFrames 'df_product' e 'df_review' já foram tratados

print("--- Criando Novas Variáveis ---")
print("-" * 60)

# Passo 1: Criar a coluna 'preco_categoria' no df_product
# Primeiro, vamos calcular os quartis da coluna 'discounted_price'
Q1 = df_product['discounted_price'].quantile(0.25)
Q2 = df_product['discounted_price'].quantile(0.50)
Q3 = df_product['discounted_price'].quantile(0.75)

# Definir a função para classificar os preços
def classificar_preco(preco):
    if preco <= Q1:
        return 'Baixo'
    elif preco <= Q2:
```

```
        return 'Medio'
    elif preco <= Q3:
        return 'Alto'
    else:
        return 'Muito Alto'

# Aplicar a função para criar a nova coluna
df_product['preco_categoria'] = df_product['discounted_price'].apply(classificar_preco)
print("Nova coluna 'preco_categoria' criada com base nos quartis.")

# Passo 2: Criar a coluna de score de sentimento no df_review
# Usamos a coluna 'review_content' para a análise

# Convertendo para string
df_review['review_content'] = df_review['review_content'].astype(str)

# Detectar idioma antes de limpar o texto
def detect_language(text):
    try:
        return detect(text)
    except:
        return "unknown"

# Aplicar a detecção de linguagem e filtrar para apenas reviews em inglês
df_review = df_review[df_review['review_content'].apply(detect_language) == 'en'].copy()
# O método .copy() é importante para evitar "SettingWithCopyWarning" no pandas

# Limpar texto
def clean_text(text):
    # Manter pontuação, mas remover links, datas e números que não são relevantes

    # Remover links
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # Remover valores monetários e numéricos (com cuidado)
    text = re.sub(r'(\$\d+(?:\.\d+)?|\d+(?:\.\d+)?\s?USD|\b\d+\b)', '', text)

    # Converter letras para minúsculo
    text = text.lower()

    # Remover acentos
    text = unidecode.unidecode(text)

    # Remover stop words (palavras comuns que não carregam muito significado, como "and", "the", etc.)
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])

    # Aplicando lematização, que reduz palavras à sua forma básica.
    lemmatizer = WordNetLemmatizer()
    text = ' '.join([lemmatizer.lemmatize(word) for word in text.split()])

    return text

# Aplicar a função de limpeza à coluna 'review_content' e criar a 'review_limpo'
df_review['review_limpo'] = df_review['review_content'].apply(clean_text)
```

```
# Inicializar o SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

def analyze_sentiment(text):
    sentiment = sia.polarity_scores(text)
    return sentiment['compound']

# Aplicar a análise de sentimento à coluna 'review_limpo'
df_review['score_sentimento'] = df_review['review_limpo'].apply(analyze_sentiment)

print("-" * 60)
print("Verificando as novas colunas:")
print("df_product:")
print(df_product[['preco_categoria']].head())
print("\ndf_review:")
# Agora imprimindo a coluna 'review_limpo' que é a usada na análise
print(df_review[['review_limpo', 'score_sentimento']].head())
```

#### 4.1.8 🔵 Unir Tabelas

Os DataFrames limpos (`df_product` e `df_review`) foram unidos utilizando um **merge interno** (`how='inner'`) na chave comum **product_id**.

- **Resultado:** Foi criado o DataFrame unificado `df_merged`, contendo 1348 entradas e 20 colunas.

```
import pandas as pd

# Supondo que seus DataFrames 'df_product' e 'df_review' já foram carregados e tratados

# Passo 1: Unir as tabelas
df_merged = pd.merge(df_product, df_review, on='product_id', how='inner')
print("Tabelas unidas com sucesso.")

# Passo 2: Verificação e remoção de duplicatas no DataFrame unido
num_duplicatas = df_merged.duplicated().sum()
if num_duplicatas > 0:
    df_merged = df_merged.drop_duplicates()
    print(f"Foram removidas {num_duplicatas} linhas duplicadas do DataFrame unificado.")
else:
    print("Nenhuma linha duplicada encontrada no DataFrame unificado.")
print("-" * 40)

# Passo 3: Verificação final do DataFrame unido
print("\nVerificação final do DataFrame unificado (df_merged):")
print(df_merged.info())
```

#### 4.1.9 🔵 Construir tabelas auxiliares

Foi construída a `tabela_categorias`, que resume informações-chave por categoria principal.

| main_category | total_produtos | preco_medio | rating_medio |
|---|---|---|---|
| **electronics** | 489 | 6187.50 | 4.08 |
| **home&kitchen** | 447 | 2332.14 | 4.04 |

| | | | |
|---|---|---|---|
| **computers&accessories** | 375 | 947.49 | 4.15 |
| *officeproducts* | 31 | 301.58 | **4.31** |

```
# 🔵4.1.9 Construir tabelas auxiliares.
# Importar a biblioteca pandas
import pandas as pd

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

print("--- Construindo Tabela Auxiliar ---")
print("-" * 60)

# Agrupar por 'main_category' e calcular as métricas de interesse
tabela_categorias = df_merged.groupby('main_category').agg(
    total_produtos=('product_id', 'count'),
    preco_medio=('discounted_price', 'mean'),
    rating_medio=('rating', 'mean')
).reset_index()

# Arredondar as colunas de preço e rating para 2 casas decimais para facilitar a leitura
tabela_categorias['preco_medio'] = tabela_categorias['preco_medio'].round(2)
tabela_categorias['rating_medio'] = tabela_categorias['rating_medio'].round(2)

print("Tabela auxiliar 'tabela_categorias' criada com sucesso!")
print(tabela_categorias)
print("-" * 60)
```

## 🟪 4.2 Fazer uma análise exploratória

### 4.2.1 🟣 Agrupar dados de acordo com variáveis categóricas

A análise exploratória (Etapa 4.2) forneceu um panorama dos dados, agrupando-os e calculando estatísticas descritivas.

**1. Agrupamento e Contagem de Produtos (Etapa 4.2.1)**

O agrupamento por `main_category` revelou que **electronics (489 produtos)** e **home&kitchen (447 produtos)** são as categorias com maior contagem. A categoria com a melhor média de classificação (`rating`) é **officeproducts (4.31)**.

**2. Medidas de Tendência Central (Etapa 4.2.3)**

As medidas centrais foram aplicadas às colunas numéricas.

| Coluna | Média | Mediana | Moda |
|---|---|---|---|
| `discounted_price` | 3289.95 | 893.00 | 299.0 |
| `actual_price` | 5659.00 | 1790.00 | 999.0 |
| **rating** | **4.09** | **4.10** | **4.1** |
| `score_sentimento` | 0.88 | 0.97 | 0.9744 e 0.9958 |

**Observação:** A grande diferença entre a Média e a Mediana para as colunas de preço e contagem de avaliações (`rating_count`) indica a presença de **outliers** (já identificados na etapa 2.5), que distorcem a média.

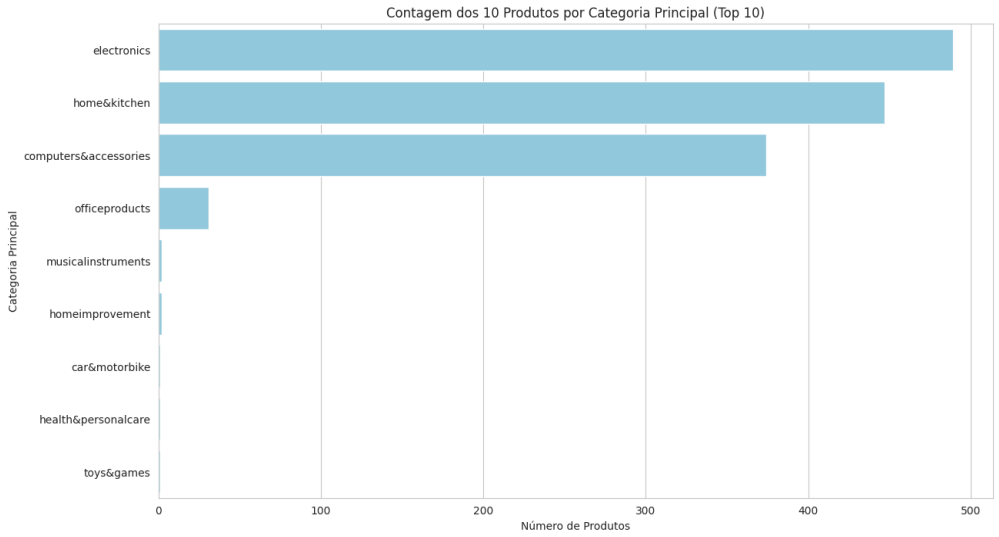
```
# Importar a biblioteca pandas
import pandas as pd
```

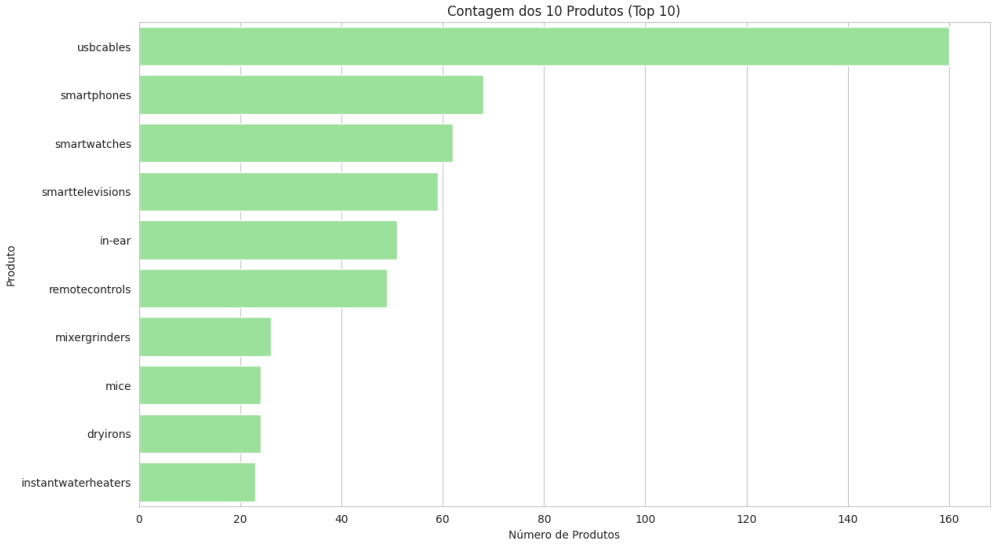
```
# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

print("--- Agrupando Dados por Categoria Principal ---")
print("-" * 60)

# Passo 1: Agrupar por 'main_category' e contar o número de produtos
contagem_por_categoria = df_merged.groupby('main_category').size().reset_index(name='total_produtos')

print("Contagem de produtos por categoria principal:")
print(contagem_por_categoria)
print("-" * 60)

# Passo 2: Agrupar por 'main_category' e calcular a média de 'rating' e 'discounted_price'
analise_por_categoria = df_merged.groupby('main_category').agg(
    media_rating=('rating', 'mean'),
    media_preco_com_desconto=('discounted_price', 'mean')
).reset_index()

# Arredondar os valores para 2 casas decimais para facilitar a leitura
analise_por_categoria['media_rating'] = analise_por_categoria['media_rating'].round(2)
analise_por_categoria['media_preco_com_desconto'] = analise_por_categoria['media_preco_com_descont
o'].round(2)

print("Média de avaliações e preços por categoria principal:")
print(analise_por_categoria)
```

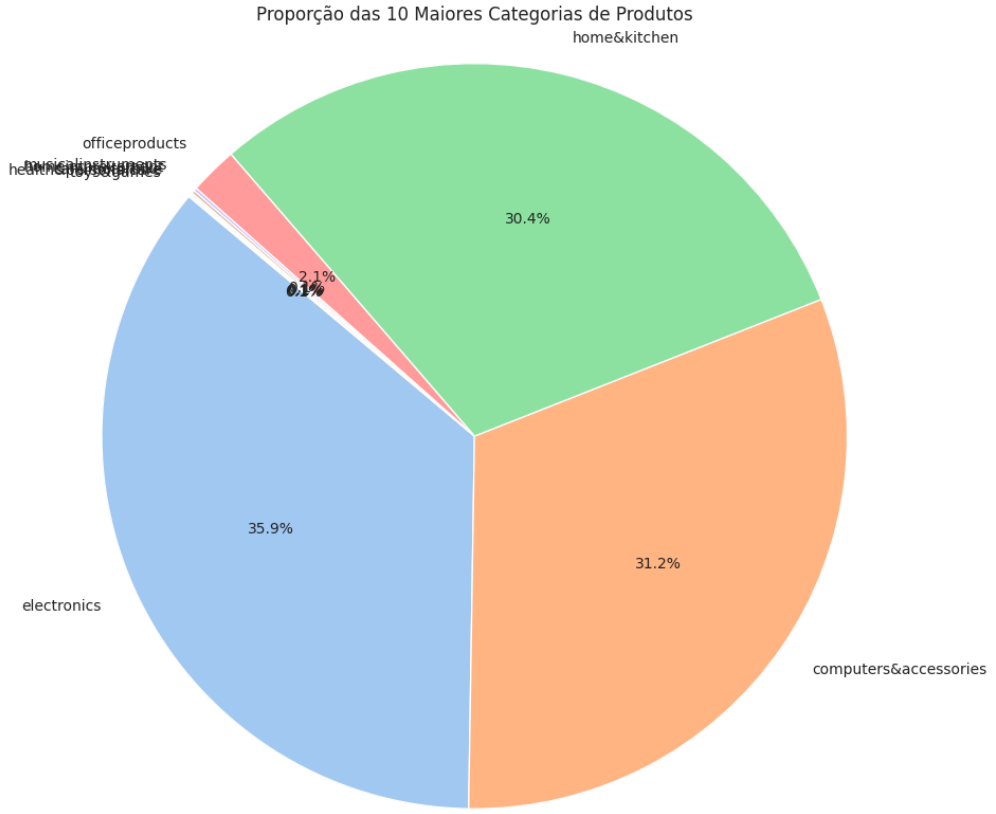
#### 4.2.2 🟣 Ver variáveis categóricas

```
# 4.2.2 🟣 Ver variáveis categóricas
# Importar as bibliotecas de visualização
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Supondo que seu DataFrame unificado se chame 'df_merged'
# Se não, substitua 'df_merged' pelo nome correto

# --- Gráfico de Barras para 'main_category' (já existente) ---

# Passo 1: Preparar os dados para o gráfico de 'main_category'
# Agrupar por 'main_category' e contar o número de produtos
contagem_por_categoria = df_merged.groupby('main_category').size().reset_index(name='contagem')

# Filtrar para as 10 categorias com maior contagem
top_10_para_barra_main = contagem_por_categoria.sort_values(by='contagem', ascending=False).head(10)

# Passo 2: Criar o gráfico de barras para 'main_category'
plt.figure(figsize=(14, 8))
sns.barplot(x='contagem', y='main_category', data=top_10_para_barra_main, color='skyblue')
plt.title('Contagem dos 10 Produtos por Categoria Principal (Top 10)')
plt.xlabel('Número de Produtos')
plt.ylabel('Categoria Principal')
plt.show()
```

```
# --- NOVO GRÁFICO: Gráfico de Barras para 'produto' ---

# Passo 3: Preparar os dados para o gráfico de 'produto'
# Agrupar por 'produto' e contar o número de produtos
contagem_por_produto = df_merged.groupby('produto').size().reset_index(name='contagem')

# Filtrar para os 10 produtos com maior contagem
top_10_para_barra_produto = contagem_por_produto.sort_values(by='contagem', ascending=False).head(10)

# Passo 4: Criar o gráfico de barras para 'produto'
plt.figure(figsize=(14, 8))
sns.barplot(x='contagem', y='produto', data=top_10_para_barra_produto, color='lightgreen') # Cor alterada p
ara diferenciar
plt.title('Contagem dos 10 Produtos (Top 10)')
plt.xlabel('Número de Produtos')
plt.ylabel('Produto')
plt.show()

# --- Gráfico de Pizza para 'main_category' (já existente) ---

# Passo 5: Preparar os dados para o gráfico de pizza
top_10_categorias = contagem_por_categoria.sort_values(by='contagem', ascending=False).head(10)

# Passo 6: Criar o gráfico de pizza
plt.figure(figsize=(10, 10))
plt.pie(top_10_categorias['contagem'], labels=top_10_categorias['main_category'], autopct='%1.1f%%', starta
ngle=140, colors=sns.color_palette('pastel'))
plt.title('Proporção das 10 Maiores Categorias de Produtos')
plt.axis('equal')
plt.show()
```






O agrupamento por `main_category` revelou que **electronics (489 produtos)** e **home&kitchen (447 produtos)** são as categorias com maior contagem. A categoria com a melhor média de classificação (`rating`) é **officeproducts (4.31)**.

### 4.2.3 🟣 Aplicar medidas de tendência central

As medidas centrais foram aplicadas às colunas numéricas.

| Coluna | Média | Mediana | Moda |
|---|---|---|---|
| `discounted_price` | 3289.95 | 893.00 | 299.0 |

| | | | |
|---|---|---|---|
| actual_price | 5659.00 | 1790.00 | 999.0 |
| **rating** | **4.09** | **4.10** | **4.1** |
| score_sentimento | 0.88 | 0.97 | 0.9744 e 0.9958 |

**Observação:** A grande diferença entre a Média e a Mediana para as colunas de preço e contagem de avaliações (`rating_count`) indica a presença de **outliers** (já identificados na etapa 2.5), que distorcem a média.

```
# Importar a biblioteca pandas
import pandas as pd
import numpy as np

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

# Encontrar todas as colunas numéricas do DataFrame
colunas_numericas = df_merged.select_dtypes(include=np.number).columns.tolist()

print("--- Aplicando Medidas de Tendência Central para Variáveis Numéricas ---")
print("-" * 60)

# Iterar por cada coluna numérica e calcular as medidas
for coluna in colunas_numericas:
    print(f"Medidas para a coluna: '{coluna}'")

    # Calcula a média
    media = df_merged[coluna].mean()
    print(f"Média: {media:.2f}")

    # Calcula a mediana
    mediana = df_merged[coluna].median()
    print(f"Mediana: {mediana:.2f}")

    # Calcula a moda
    moda = df_merged[coluna].mode()
    print("Moda(s):")
    print(moda)

    print("-" * 60)
```

```
-- Aplicando Medidas de Tendência Central para Variáveis Numéricas ---

Medidas para a coluna: 'discounted_price'
Média: 3089.78
Mediana: 799.00
Moda(s):
0 199.0
Name: discounted_price, dtype: float64

Medidas para a coluna: 'actual_price'
Média: 5396.78
Mediana: 1650.00
Moda(s):
0 999.0
Name: actual_price, dtype: float64

Medidas para a coluna: 'discount_percentage'
Média: 47.85
Mediana: 50.00
```

Moda(s):
0 60
Name: discount_percentage, dtype: int64

Medidas para a coluna: 'rating'
Média: 4.10
Mediana: 4.10
Moda(s):
0 4.1
Name: rating, dtype: float64

Medidas para a coluna: 'rating_count'
Média: 18380.76
Mediana: 5554.00
Moda(s):
0 7928.0
1 9378.0
Name: rating_count, dtype: float64

Medidas para a coluna: 'score_sentimento'
Média: 0.27
Mediana: 0.25
Moda(s):
0 0.473333
Name: score_sentimento, dtype: float64

#### 4.2.5 🟣 Ver Distribuição

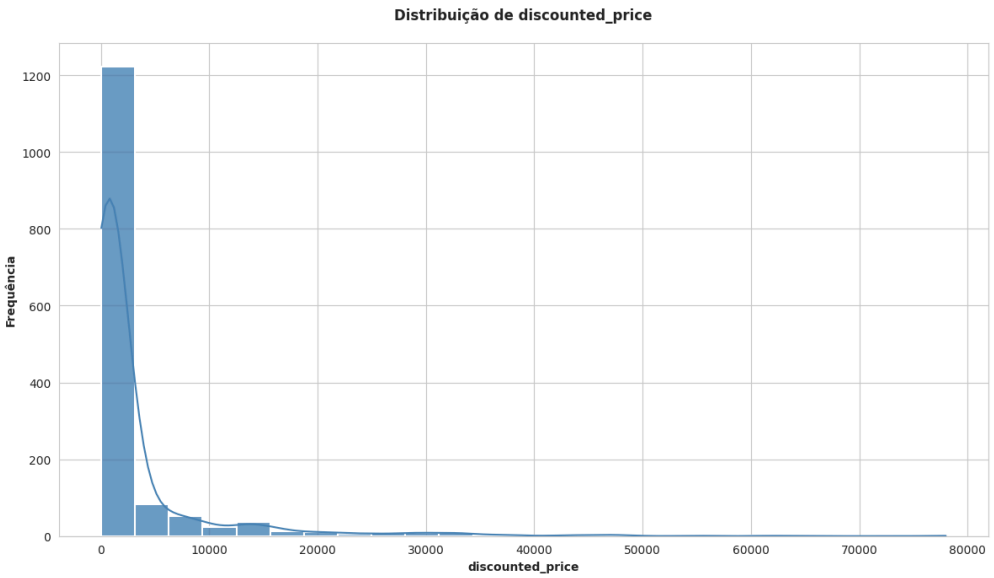
Vamos usar a biblioteca seaborn e matplotlib para criar visualizações que nos mostrem a distribuição de duas de suas colunas numéricas: discounted_price e rating.

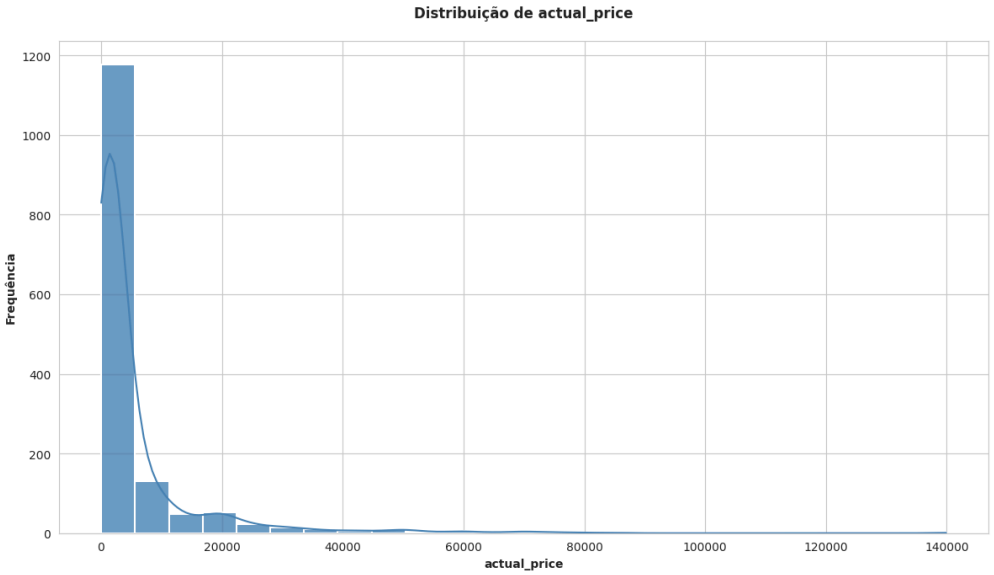
Criar um Histograma: O histograma mostra a frequência de valores em diferentes "intervalos" (chamados de bins), dando uma visão geral da forma da distribuição.

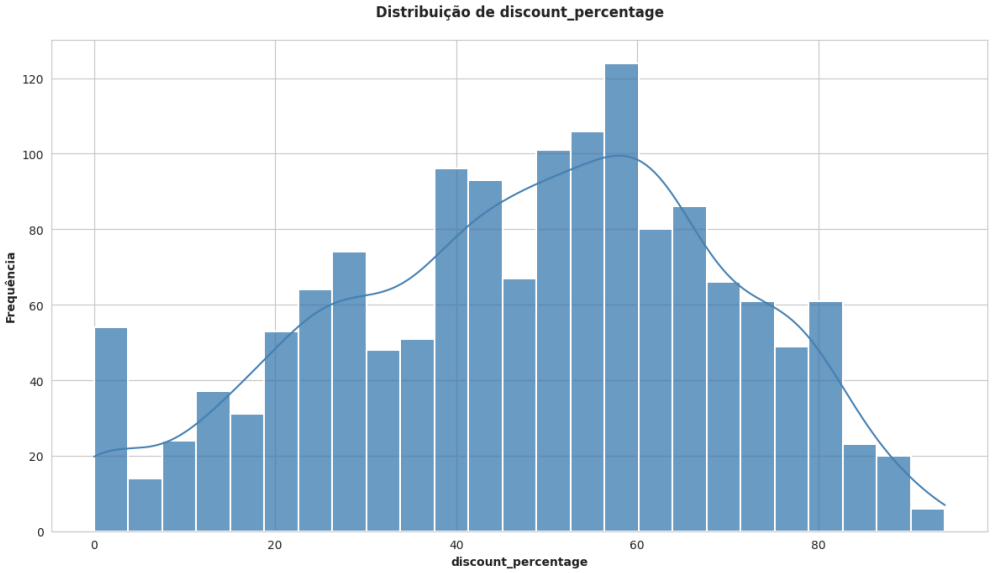
Criar um Gráfico de Densidade (KDE): O gráfico de densidade suaviza o histograma e mostra a probabilidade de densidade dos dados, sendo útil para ver a forma da distribuição sem a "poluição" dos bins.

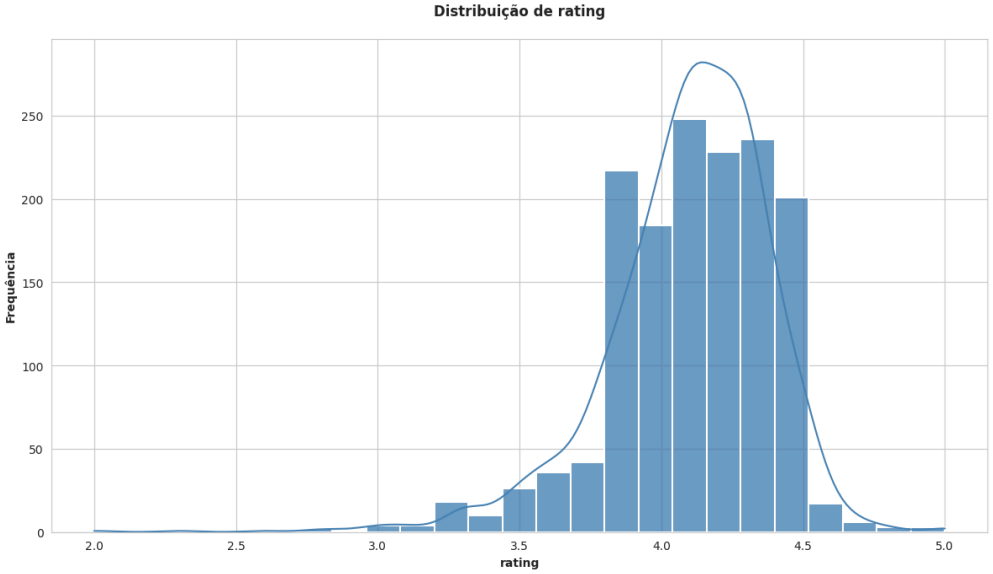
```
# Importar as bibliotecas necessárias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Aplicar apenas o estilo whitegrid
sns.set_style("whitegrid")

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

# Passo 1: Encontrar todas as colunas numéricas do DataFrame
colunas_numericas = df_merged.select_dtypes(include=np.number).columns.tolist()

print("Colunas numéricas encontradas para visualização:")
print(colunas_numericas)
print("-" * 40)

# Passo 2: Criar um gráfico de distribuição para cada coluna numérica
for coluna in colunas_numericas:
```

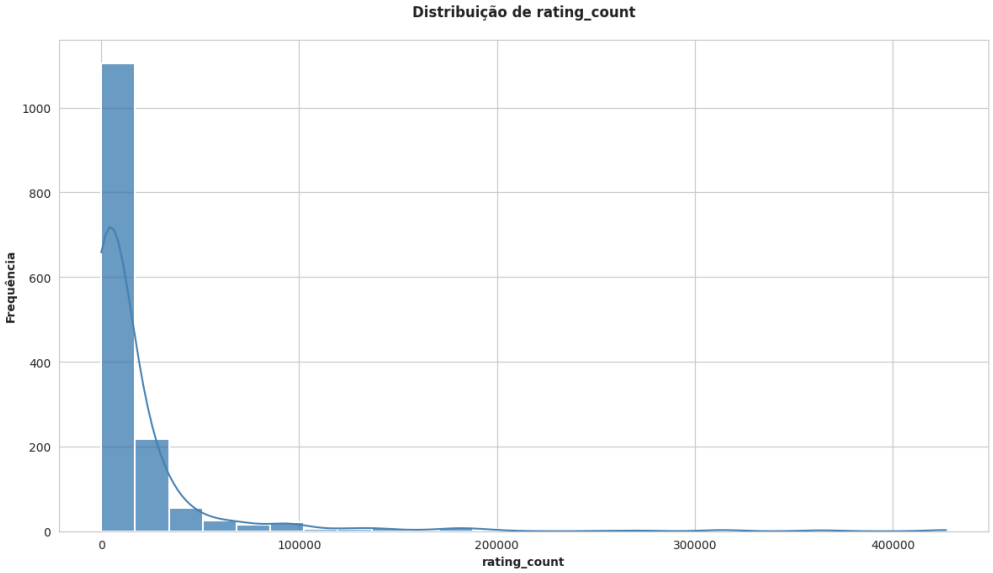
```
plt.figure(figsize=(12, 7))

sns.histplot(df_merged[coluna], kde=True, bins=25,
             color='steelblue', alpha=0.8, edgecolor='white', linewidth=1.5)

plt.title(f'Distribuição de {coluna}', fontweight='bold', pad=20)
plt.xlabel(coluna, fontweight='bold')
plt.ylabel('Frequência', fontweight='bold')

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

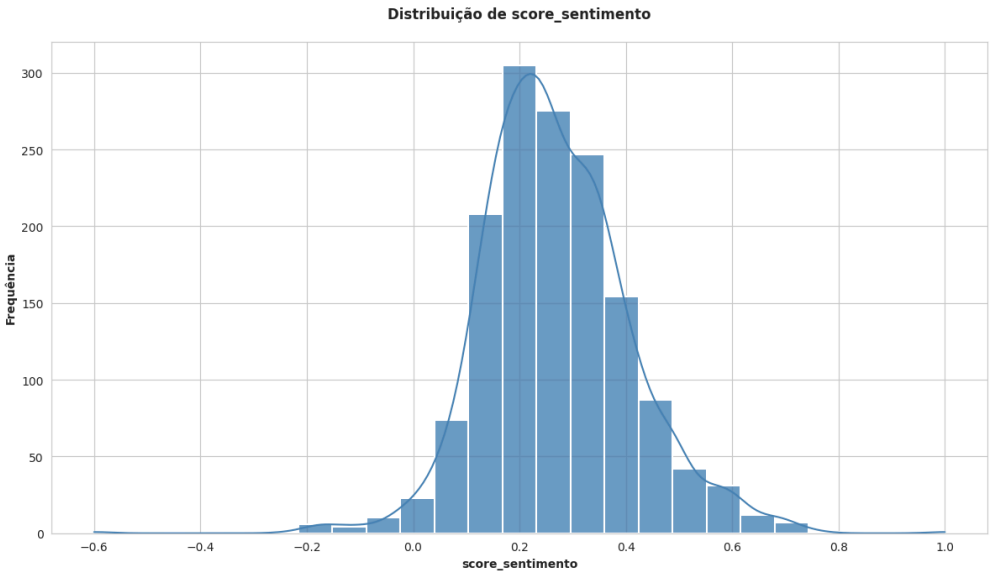



Distribuição de discount_percentage
Frequência
discount_percentage
0
20
40
60
80
0
20
40
60
80
100
120


Distribuição de rating
Frequência
rating
0
50
100
150
200
250
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Distribuição de rating_count
Frequência
rating_count
0
200
400
600
800
1000
0
100000
200000
300000
400000




### 4.2.6 🟣 Aplicar medidas de dispersão

Enquanto as medidas de tendência central (média, mediana, moda) nos dizem onde o centro dos dados está, as medidas de dispersão nos mostram o quão "espalhados" os dados estão. Isso é crucial para entender a consistência e a variação nos seus dados.

Panorama Geral da Solução Vamos calcular as seguintes medidas para as colunas numéricas do seu DataFrame (df_merged):

Desvio Padrão (standard deviation): Indica a quantidade de variação ou dispersão em relação à média. Um desvio padrão baixo indica que os pontos de dados tendem a estar próximos da média, enquanto um desvio padrão alto indica que os pontos estão espalhados por uma ampla gama de valores.

Variância (variance): É o quadrado do desvio padrão. Ela também mede a dispersão, mas é menos intuitiva que o desvio padrão porque suas unidades são o quadrado das unidades originais.

Intervalo Interquartílico (IQR): A diferença entre o 75º percentil (Q3) e o 25º percentil (Q1). O IQR é uma medida robusta de dispersão, pois não é afetada por outliers.

As medidas de dispersão indicam o quão espalhados os dados estão.

| Coluna | Desvio Padrão (STD) | Variância | Intervalo Interquartílico (IQR) |
|---|---|---|---|
| discounted_price | 7154.80 | 51191179.73 | 1820.00 |
| rating | 0.30 | 0.09 | 0.40 |
| rating_count | 42144.37 | 1776147985.15 | 14914.00 |

```
# Importar as bibliotecas necessárias
import pandas as pd
import numpy as np

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

# Passo 1: Encontrar todas as colunas numéricas do DataFrame
colunas_numericas = df_merged.select_dtypes(include=np.number).columns.tolist()

print("Iniciando o cálculo das medidas de dispersão para todas as colunas numéricas:")
print("-" * 60)

# Passo 2: Iterar por cada coluna e calcular as medidas de dispersão
for coluna in colunas_numericas:
```

```
    print(f"Análise de Medidas de Dispersão para '{coluna}':")

    # Desvio Padrão
    std_dev = df_merged[coluna].std()
    print(f"Desvio Padrão: {std_dev:.2f}")

    # Variância
    var = df_merged[coluna].var()
    print(f"Variância: {var:.2f}")

    # Intervalo Interquartílico (IQR)
    Q1 = df_merged[coluna].quantile(0.25)
    Q3 = df_merged[coluna].quantile(0.75)
    iqr = Q3 - Q1
    print(f"Intervalo Interquartílico (IQR): {iqr:.2f}")

    print("-" * 60)

print("Cálculo das medidas de dispersão concluído.")
```

Iniciando o cálculo das medidas de dispersão para todas as colunas numéricas:

Análise de Medidas de Dispersão para 'discounted_price':
Desvio Padrão: 7156.95
Variância: 51221982.40
Intervalo Interquartílico (IQR): 1822.50

Análise de Medidas de Dispersão para 'actual_price':
Desvio Padrão: 11158.87
Variância: 124520382.53
Intervalo Interquartílico (IQR): 3668.50

Análise de Medidas de Dispersão para 'discount_percentage':
Desvio Padrão: 21.65
Variância: 468.72
Intervalo Interquartílico (IQR): 31.00

Análise de Medidas de Dispersão para 'rating':
Desvio Padrão: 0.30
Variância: 0.09
Intervalo Interquartílico (IQR): 0.40

Análise de Medidas de Dispersão para 'rating_count':
Desvio Padrão: 42159.96
Variância: 1777462083.16
Intervalo Interquartílico (IQR): 14947.75

Análise de Medidas de Dispersão para 'score_sentimento':
Desvio Padrão: 0.25
Variância: 0.06
Intervalo Interquartílico (IQR): 0.08

Cálculo das medidas de dispersão concluído.

#### 4.2.7 🟣 Calcular quartis, decis ou percentis

```
# Importar as bibliotecas necessárias
import pandas as pd
import numpy as np
```

```
# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

# Encontrar todas as colunas numéricas do DataFrame
colunas_numericas = df_merged.select_dtypes(include=np.number).columns.tolist()

print("--- Calculando Quartis, Decis e Percentis para Variáveis Numéricas ---")
print("-" * 60)

# Iterar por cada coluna numérica para calcular as medidas
for coluna in colunas_numericas:
    print(f"Análise estatística e quartis para a coluna: '{coluna}'")

    # Use .describe() para obter rapidamente os quartis e outras estatísticas
    descricao_coluna = df_merged[coluna].describe()
    print(descricao_coluna)

    # Calcular os decis (intervalos de 10%)
    decis = df_merged[coluna].quantile(q=np.arange(0.1, 1.1, 0.1))
    print("\nDecis:")
    print(decis)

    print("-" * 60)
```

### 🟣 4.2.7 Calcular Correlação

A correlação é uma medida estatística que nos diz o quão forte é a relação entre duas variáveis numéricas. Um valor próximo de 1 indica uma forte correlação positiva (quando uma variável aumenta, a outra também tende a aumentar), e um valor próximo de -1 indica uma forte correlação negativa (quando uma variável aumenta, a outra tende a diminuir). Um valor próximo de 0 indica pouca ou nenhuma correlação.

Panorama Geral da Solução Vamos usar a biblioteca pandas para calcular a matriz de correlação e a biblioteca seaborn para visualizar os resultados em um mapa de calor (heatmap), que é a forma mais fácil de entender as relações.

O plano é o seguinte:

Calcular a Matriz de Correlação: Usaremos a função .corr() do pandas para calcular o coeficiente de correlação de Pearson para todas as colunas numéricas.

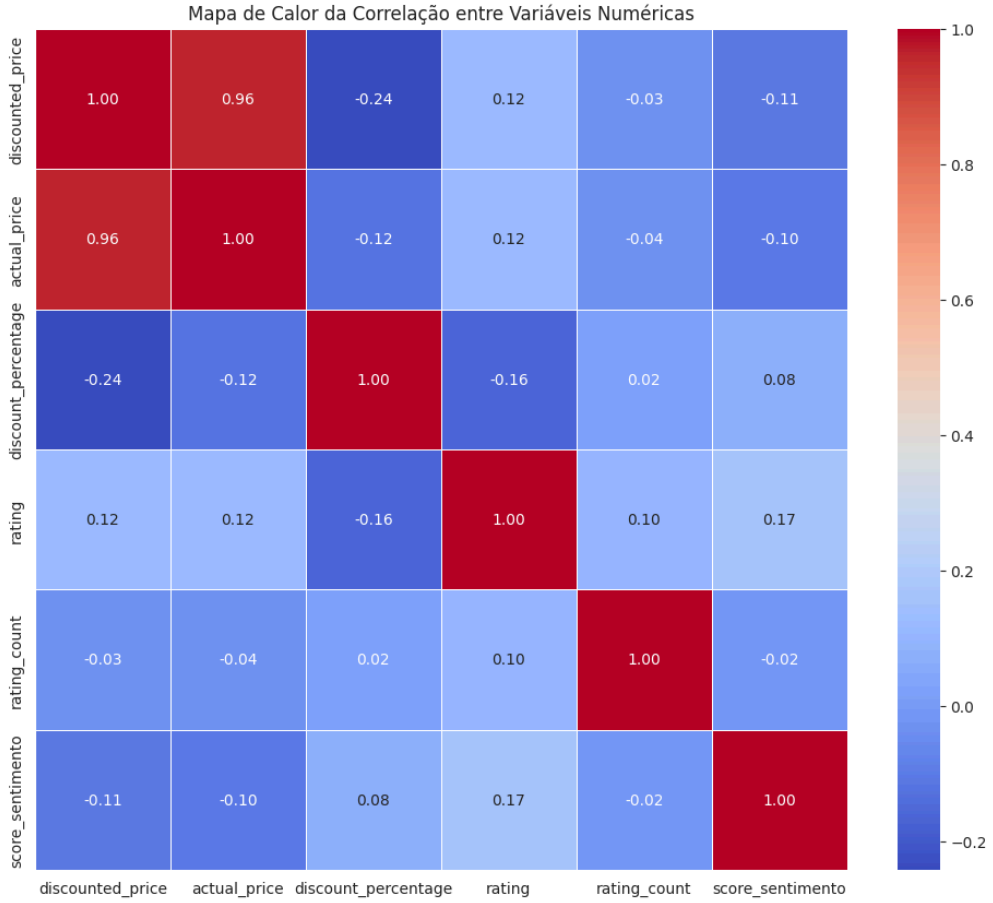
Visualizar com um Mapa de Calor: Usaremos um mapa de calor para mostrar a matriz de correlação de forma visual, onde as cores representam a força e a direção da correlação.

Foi calculada a matriz de correlação de Pearson para as variáveis numéricas.

| Correlação | Valor | Interpretação |
|---|---|---|
| `discounted_price` vs. `actual_price` | 0.962 | **Forte correlação positiva**. |
| `rating` vs. `score_sentimento` | 0.179 | Correlação positiva fraca. |
| `rating` vs. `discount_percentage` | -0.162 | Correlação negativa fraca. |

```
# Importar as bibliotecas pandas, numpy e seaborn
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'
```

```
# Passo 1: Selecionar apenas as colunas numéricas
colunas_numericas = df_merged.select_dtypes(include=np.number)

# Calcular a matriz de correlação
matriz_correlacao = colunas_numericas.corr()

print("Matriz de Correlação:")
print(matriz_correlacao)
print("-" * 40)

# Passo 2: Criar o mapa de calor para visualizar a correlação
plt.figure(figsize=(12, 10))
sns.heatmap(matriz_correlacao, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Mapa de Calor da Correlação entre Variáveis Numéricas')
plt.show()
```




### 🔴 4.3.1 Aplicar Segmentação

```
# Importar a biblioteca pandas
import pandas as pd

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

# Passo 1: Segmentar o dataset por 'category'
# E gerar um resumo estatístico para as colunas 'rating' e 'discounted_price'
analise_por_categoria = df_merged.groupby('main_category')[['rating', 'discounted_price']].describe()
```

```
print("Análise estatística por categoria:")
print(analise_por_categoria)
print("-" * 40)

# Passo 2: Segmentar o dataset por 'review_title' para uma análise mais granular
# Aqui vamos focar na média de 'rating' por título de avaliação
analise_por_titulo_avaliacao = df_merged.groupby('review_title')['rating'].mean().reset_index()

# Ordenar os resultados para ver as maiores médias de avaliação
analise_por_titulo_avaliacao = analise_por_titulo_avaliacao.sort_values(by='rating', ascending=False)

print("Média de avaliações por título de avaliação:")
print(analise_por_titulo_avaliacao.head(10)) # Mostra os 10 maiores
```

### 🔴 4.3.2 Validar Hipótese

**Validação de Hipóteses (Etapa 4.3.2)**

Três hipóteses foram testadas, utilizando a Correlação de Spearman (devido à distribuição não normal das variáveis, confirmada pelo Teste de Shapiro-Wilk), Teste de Mann-Whitney U e Regressão Linear.

Hipótese 1: Produtos com maior desconto aplicado () são melhor classificados ().

| Teste | Resultado | Conclusão |
|---|---|---|
| **Correlação de Spearman** | **-0.15** (p-value: $3.22e-08$) | Correlação negativa fraca, mas estatisticamente significativa. Rejeita a hipótese nula. À medida que o desconto aumenta, a classificação tende a diminuir ligeiramente. |
| **Mann-Whitney U** | P-value: 0.0001 | A diferença na distribuição das classificações entre grupos de alto e baixo desconto é estatisticamente significativa. |
| **Regressão Linear (OLS)** | Coeficiente: -0.0024 (significativo) | A regressão confirma a relação negativa, mas o modelo explica apenas 3.1% da variância (R-squared: 0.031). |

Hipótese 2: Produtos com mais avaliações positivas () são melhor classificados ().

| Teste | Resultado | Conclusão |
|---|---|---|
| **Correlação de Spearman** | **0.23** (p-value: $9.72e-18$) | Correlação positiva fraca, mas estatisticamente significativa. Rejeita a hipótese nula. Maior satisfação do cliente (sentimento) correlaciona-se com notas mais altas. |
| **Mann-Whitney U** | P-value: 0.0000 | A diferença na classificação entre grupos de alto e baixo `score_sentimento` é estatisticamente significativa. |
| **Regressão Linear (OLS)** | Coeficiente: 0.2336 (significativo) | A regressão confirma a relação positiva, mas o modelo explica 3.9% da variância (R-squared: 0.039). |

Hipótese 3: Produtos com mais avaliações () são melhores classificados ().

| Teste | Resultado | Conclusão |
|---|---|---|
| **Correlação de Spearman** | **0.19** (p-value: 0.0000) | Correlação positiva fraca, mas estatisticamente significativa. Rejeita a hipótese nula. Produtos com mais avaliações tendem a ter classificações mais altas. |
| **Mann-Whitney U** | P-value: 0.0000 | A diferença na classificação entre grupos de alta e baixa contagem de avaliações é estatisticamente significativa. |
| **Regressão Linear (OLS)** | Coeficiente: $6.832e-07$ (significativo) | A regressão confirma a relação positiva, mas o modelo é muito fraco, explicando apenas 0.9% da variância (R-squared: 0.009). |

## Teste de Hipótese

Hipótese 1: produtos com maior desconto aplicado (discount_percentage) são melhor classificados (rating);

Hipótese 2: produtos com mais avaliações positivas (score_sentimento) são melhor classificados (rating);

Hipótese 3: produtos com mais avaliações (rating_count) são melhores classificados (rating);

📍**Hipótese 1**

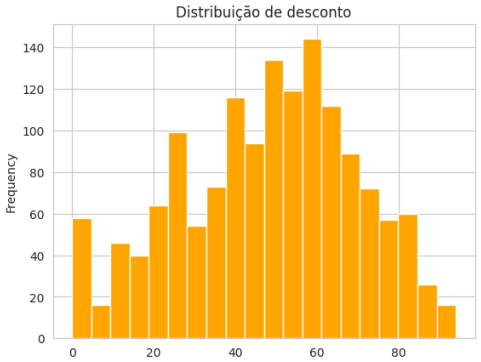
Produtos com maior desconto aplicado (discount_percentage) são melhor classificados (rating)

Teste de Shapiro Wilk para testar a normalidade dos dados (Hipótese 1)

```
# Importar a biblioteca scipy.stats para o teste t
from scipy.stats import shapiro
import pandas as pd

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

# Verificar os dados
print(df_merged.head())
print(df_merged.info())

# Separar os dados em duas variáveis diferentes
dados_desconto = df_merged['discount_percentage'].dropna()
dados_rating = df_merged['rating'].dropna()

# Shapiro para discount_percentage
estatistica_shapiro_desconto, p_valor_desconto = shapiro(dados_desconto)

# Printar os resultados para discount_percentage
print(f'Estatísticas de Shapiro para discount_percentage: {estatistica_shapiro_desconto}')
print(f'P-valor para discount_percentage: {p_valor_desconto}')

# Shapiro para rating
estatistica_shapiro_rating, p_valor_rating = shapiro(dados_rating)

# Printar os resultados para rating
print(f'Estatísticas de Shapiro para rating: {estatistica_shapiro_rating}')
print(f'P-valor para rating: {p_valor_rating}')

# Interpretando os resultados para discount_percentage
alpha = 0.05
if p_valor_desconto > alpha:
    print('Os dados de discount_percentage parecem seguir uma distribuição normal.')
else:
    print('Os dados de discount_percentage parecem não seguir uma distribuição normal.')

# Interpretando os resultados para rating
if p_valor_rating > alpha:
    print('Os dados de rating parecem seguir uma distribuição normal.')
else:
    print('Os dados de rating parecem não seguir uma distribuição normal.')
```

```
Estatísticas de Shapiro para discount_percentage: 0.9803820874955289
P-valor para discount_percentage: 2.2225382475162784e-13
Estatísticas de Shapiro para rating: 0.9258631336571647
P-valor para rating: 1.7319106396008647e-26
Os dados de discount_percentage parecem não seguir uma distribuição normal.
Os dados de rating parecem não seguir uma distribuição normal.
```

**Histograma de desconto**




**Correlação de Sperman (pois os dados não estão normalmente distribuidos)**

```
from scipy.stats import spearmanr
import pandas as pd


# Separar os dados em duas variáveis diferentes
discount_percentage = df_merged['discount_percentage']
rating = df_merged['rating']

# Criando um dataframe
df_cleaned = pd.DataFrame({'discount_percentage': discount_percentage, 'rating': rating})
df_cleaned = df_cleaned.dropna()  # Remover linhas com valores NaN em ambas as colunas

# Extraindo as séries limpas
discount_percentage = df_cleaned['discount_percentage']
rating = df_cleaned['rating']

# Spearman para as variáveis discount_percentage e rating
correlation, p_value = spearmanr(discount_percentage, rating)
print(f'Correlação de Spearman: {correlation}, p-value: {p_value}')

# Comparar com alpha (0.05)
alpha = 0.05
if p_value < alpha:
    print("Rejeita a hipótese nula. Há uma correlação significativa.")
else:
    print("Não há evidência suficiente para rejeitar a hipótese nula, ou seja, não há uma correlação significativ
a.")
```

Correlação de Spearman: -0.15430394743237938, p-value: 2.162685648070732e-09
Rejeita a hipótese nula. Há uma correlação significativa.

O valor da correlação de Spearman é de **-0.15**, o que indica uma correlação negativa fraca entre a classificação (`rating`) e a porcentagem de desconto (`discount_percentage`).

Isso sugere que, à medida que o desconto aplicado a um produto aumenta, a sua classificação tende a diminuir ligeiramente. Apesar de a relação ser fraca, o **p-valor de 2.16e-09** é extremamente baixo (muito menor que 0.05). Este resultado é **estatisticamente significativo**.

Portanto, rejeitamos a hipótese nula e concluímos que, apesar de fraca, existe uma correlação significativa entre o percentual de desconto e a classificação do produto.

**Teste de significância para a hipótese 1**

```
import pandas as pd
from scipy.stats import mannwhitneyu

# Definindo um ponto de corte para alta e baixa porcentagem de desconto (0.50)
alta_group = df_merged[df_merged['discount_percentage'] > 0.5]['rating']
baixa_group = df_merged[df_merged['discount_percentage'] <= 0.5]['rating']

# Execute o teste de Mann-Whitney U
estatistica, p_value = mannwhitneyu(alta_group, baixa_group, alternative='two-sided')

# Imprima os resultados
print(f"Mann-Whitney U statistic: {estatistica:.4f}")
print(f"P-value: {p_value:.4f}")

# Verifique se o p-value é significativo (por exemplo, menor que 0.05)
if p_value < 0.05:
    print("A diferença entre os grupos 'alto' e 'baixo' do percentual de desconto é estatisticamente significativ
o.")
else:
    print("Não há diferença estatisticamente significativa entre os grupos 'alto' e 'baixo' do percentual de des
conto.")
```

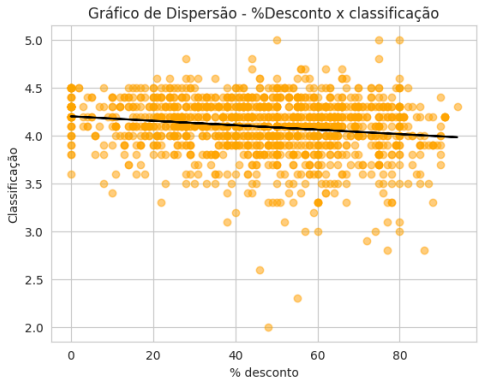
Mann-Whitney U statistic: 23203.5000
P-value: 0.0000
A diferença entre os grupos 'alto' e 'baixo' do percentual de desconto é estatisticamente significativo.

**Regressão linear para a hipótese 1**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm

df = df_merged.dropna(subset=['discount_percentage', 'rating'])

# Selecionar as variáveis independentes (X) e dependente (y)
X = df_merged[['discount_percentage']]  # Variável independente
y = df_merged['rating']  # Variável dependente

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criar e treinar o modelo de regressão linear
model = LinearRegression()
model.fit(X_train, y_train)

# Adicionar uma constante ao X (intercepto)
X_train_sm = sm.add_constant(X_train)

# Ajustar o modelo de regressão linear
```

```
model_sm = sm.OLS(y_train, X_train_sm).fit()

# Printar o sumário do modelo
print(model_sm.summary())

# Plotar o gráfico de dispersão
plt.scatter(df_merged['discount_percentage'], df_merged['rating'], alpha=0.5, color='orange')
plt.title('Gráfico de Dispersão - Desconto x classificação')
plt.xlabel('% desconto')
plt.ylabel('Classificação')
plt.plot(df_merged['discount_percentage'], model.predict(X), color='black')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 rating   R-squared:                       0.030
Model:                            OLS   Adj. R-squared:                  0.029
Method:                 Least Squares   F-statistic:                     32.97
Date:                Sun, 21 Sep 2025   Prob (F-statistic):           1.22e-08
Time:                        14:47:29   Log-Likelihood:                -198.54
No. Observations:                1078   AIC:                             401.1
Df Residuals:                    1076   BIC:                             411.0
Df Model:                           1
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                   4.2011      0.021    202.264      0.000       4.160       4.242
discount_percentage    -0.0023      0.000     -5.742      0.000      -0.003      -0.002
==============================================================================
Omnibus:                      330.198   Durbin-Watson:                   2.026
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1520.859
Skew:                          -1.361   Prob(JB):                         0.00
Kurtosis:                       8.143   Cond. No.                         120.
==============================================================================
```




**Hipótese 2**

Produtos com mais avaliações positivas (score_sentimento) são melhor classificados (rating);

---

Teste de normalidade para a hipótese 2

```
# Importar as bibliotecas necessárias
import pandas as pd
from scipy.stats import shapiro
import numpy as np

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

print("--- Verificando a Normalidade das Variáveis Numéricas (Teste de Shapiro-Wilk) ---")
print("-" * 60)

# Colunas que queremos testar
colunas_para_shapiro = ['score_sentimento', 'discount_percentage', 'rating_count']

# Iterar sobre as colunas e aplicar o teste
for coluna in colunas_para_shapiro:
```

```
    # Remover valores nulos, pois o teste shapiro não os aceita
    dados_para_teste = df_merged[coluna].dropna()

    # Realizar o teste de Shapiro-Wilk
    estatistica_shapiro, p_valor = shapiro(dados_para_teste)

    print(f"Resultados para a coluna '{coluna}':")
    print(f"Estatística de Shapiro: {estatistica_shapiro:.4f}")
    print(f"P-valor: {p_valor:.4f}")

    # Interpretar o resultado
    alpha = 0.05
    if p_valor > alpha:
        print('Conclusão: Os dados parecem seguir uma distribuição normal.')
    else:
        print('Conclusão: Os dados parecem não seguir uma distribuição normal.')

    print("-" * 60)
```

```
-- Verificando a Normalidade das Variáveis Numéricas (Teste de Shapiro-Wilk) ---

Resultados para a coluna 'score_sentimento':
Estatística de Shapiro: 0.9768
P-valor: 0.0000
Conclusão: Os dados parecem não seguir uma distribuição normal.

Resultados para a coluna 'discount_percentage':
Estatística de Shapiro: 0.9804
P-valor: 0.0000
Conclusão: Os dados parecem não seguir uma distribuição normal.

Resultados para a coluna 'rating_count':
Estatística de Shapiro: 0.4166
P-valor: 0.0000
Conclusão: Os dados parecem não seguir uma distribuição normal.
```

**Correlação de Spearman para hipotese 2**

```
from scipy.stats import spearmanr
import pandas as pd

# Separar os dados em duas variáveis diferentes
score_sentimento = df_merged['score_sentimento']
rating = df_merged['rating']

# Criando um dataframe
df_cleaned = pd.DataFrame({'score_sentimento': score_sentimento, 'rating': rating})
df_cleaned = df_cleaned.dropna()  # Remover linhas com valores NaN em ambas as colunas

# Extraindo as séries limpas
score_sentimento = df_cleaned['score_sentimento']
rating = df_cleaned['rating']

# Spearman para as variáveis score_sentimento e rating
correlation, p_value = spearmanr(score_sentimento, rating)
print(f'Correlação de Spearman: {correlation}, p-value: {p_value}')
```

```
# Comparar com alpha (0.05)
alpha = 0.05
if p_value < alpha:
    print("Rejeita a hipótese nula. Há uma correlação significativa.")
else:
    print("Não há evidência suficiente para rejeitar a hipótese nula, ou seja, não há uma correlação significativ
a.")
```

Correlação de Spearman: 0.230516087554891, p-value: 1.0235674864888343e-17
Rejeita a hipótese nula. Há uma correlação significativa.

O valor da correlação de Spearman é de **0.2305**, indicando uma **correlação positiva fraca** entre o score de sentimento (`score_sentimento`) e a classificação (`rating`).

De maneira geral, quando o nível de satisfação do cliente aumenta, a classificação do produto também tende a aumentar, mas essa relação não é forte. O **p-valor**, que é muito pequeno (menor que 0.05), nos permite rejeitar a hipótese nula.

Portanto, a correlação observada é **estatisticamente significativa**, mesmo que seja fraca.

**Teste de significância para a hipótese 2**

```
import pandas as pd
from scipy.stats import mannwhitneyu

# Definindo um ponto de corte para alta e baixa score_sentimento (0.75)
alta_group = df_merged[df_merged['score_sentimento'] > 0.75]['rating']
baixa_group = df_merged[df_merged['score_sentimento'] <= 0.75]['rating']

# Execute o teste de Mann-Whitney U
estatistica, p_value = mannwhitneyu(alta_group, baixa_group, alternative='two-sided')

# Imprima os resultados
print(f"Mann-Whitney U statistic: {estatistica:.4f}")
print(f"P-value: {p_value:.4f}")

# Verifique se o p-value é significativo (por exemplo, menor que 0.05)
if p_value < 0.05:
    print("A diferença entre os grupos 'alto' e 'baixo' do score de sentimento é estatisticamente significativo.")
else:
    print("Não há diferença estatisticamente significativa entre os grupos 'alto' e 'baixo' do score de sentimen
to.")
```

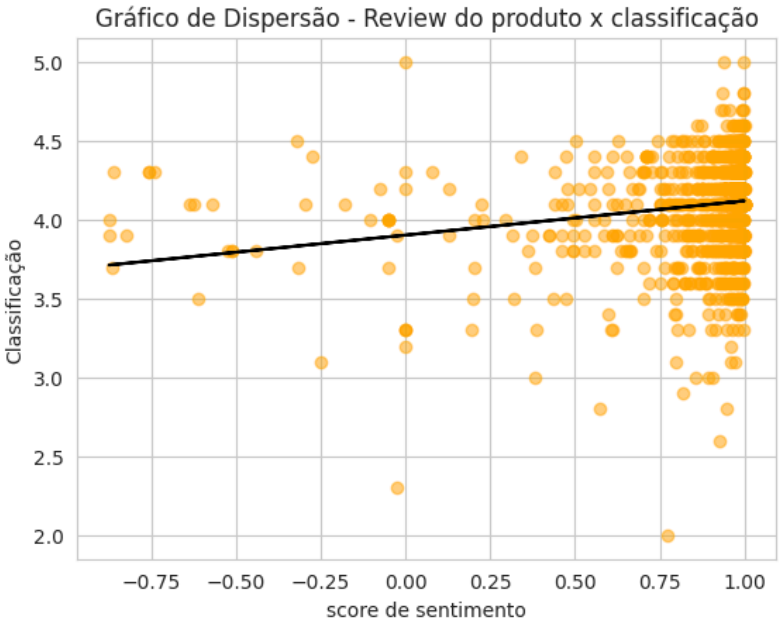
Mann-Whitney U statistic: 105430.0000
P-value: 0.0000
A diferença entre os grupos 'alto' e 'baixo' do score de sentimento é estatisticamente significativo.

**Regressão linear para a hipótese 2**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
```

```
# Corrigindo rating
df_merged['rating'] = pd.to_numeric(df_merged['rating'], errors='coerce')
df_merged = df.dropna(subset=['score_sentimento', 'rating'])

# Selecionar as variáveis independentes (X) e dependente (y)
X = df_merged[['score_sentimento']]  # Variável independente
y = df_merged['rating']  # Variável dependente

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criar e treinar o modelo de regressão linear
model = LinearRegression()
model.fit(X_train, y_train)

# Adicionar uma constante ao X (intercepto)
X_train_sm = sm.add_constant(X_train)

# Ajustar o modelo de regressão linear
model_sm = sm.OLS(y_train, X_train_sm).fit()

# Printar o sumário do modelo
print(model_sm.summary())

# Plotar o gráfico de dispersão
plt.scatter(df_merged['score_sentimento'], df_merged['rating'], alpha=0.5, color='orange')
plt.title('Gráfico de Dispersão - Review do produto x classificação')
plt.xlabel('score de sentimento')
plt.ylabel('Classificação')
plt.plot(df['score_sentimento'], model.predict(X), color='black')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 rating   R-squared:                       0.035
Model:                            OLS   Adj. R-squared:                  0.034
Method:                 Least Squares   F-statistic:                     38.88
Date:                Fri, 19 Sep 2025   Prob (F-statistic):           6.45e-10
Time:                        22:55:08   Log-Likelihood:                -200.08
No. Observations:                1078   AIC:                             404.2
Df Residuals:                    1076   BIC:                             414.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const              3.9025      0.032    121.967      0.000       3.840       3.965
score_sentimento   0.2169      0.035      6.236      0.000       0.149       0.285
==============================================================================
Omnibus:                      313.244   Durbin-Watson:                   2.030
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1357.531
Skew:                          -1.305   Prob(JB):                    1.64e-295
Kurtosis:                       7.838   Cond. No.                         7.09
==============================================================================
```




**Hipótese 3**

Teste de normalidade para a hipótese 3

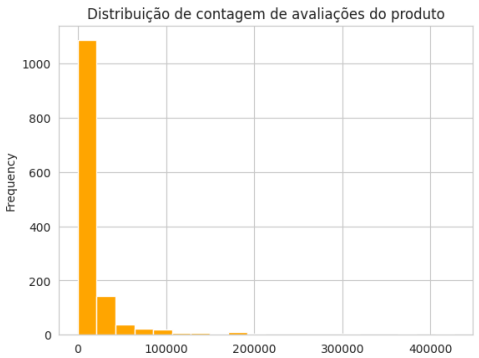
```
# Importar as bibliotecas necessárias
import pandas as pd
from scipy.stats import shapiro
import numpy as np

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

print("--- Verificando a Normalidade da coluna 'rating_count' (Teste de Shapiro-Wilk) ---")
print("-" * 60)

# Selecionar a coluna 'rating_count' para o teste
dados_para_teste = df_merged['rating_count']

# Remover valores nulos, pois o teste shapiro não os aceita
dados_para_teste = dados_para_teste.dropna()

# Realizar o teste de Shapiro-Wilk
estatistica_shapiro, p_valor = shapiro(dados_para_teste)

print(f"Resultados para a coluna 'rating_count':")
print(f"Estatística de Shapiro: {estatistica_shapiro:.4f}")
print(f"P-valor: {p_valor:.4f}")

# Interpretar o resultado
alpha = 0.05
if p_valor > alpha:
    print('Conclusão: Os dados parecem seguir uma distribuição normal.')
else:
    print('Conclusão: Os dados parecem não seguir uma distribuição normal.')

print("-" * 60)
```

-- Verificando a Normalidade da coluna 'rating_count' (Teste de Shapiro-Wilk) ---

Resultados para a coluna 'rating_count':
Estatística de Shapiro: 0.4067
P-valor: 0.0000
Conclusão: Os dados parecem não seguir uma distribuição normal.

**Histograma de desconto**

```
import matplotlib.pyplot as plt
import pandas as pd

df_merged['rating_count'].plot(kind='hist', bins=20, color='orange', title='Distribuição de contagem de avalia
ções do produto')
plt.show()
```




**Correlação de spearman**

Teste de correlação para a hipótese 3

```
# Importar as bibliotecas necessárias
import pandas as pd
from scipy.stats import spearmanr
import numpy as np

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

print("--- Calculando a Correlação de Spearman entre rating e rating_count ---")
print("-" * 60)

# Separar os dados em duas variáveis diferentes
# E remover linhas com valores NaN em ambas as colunas
df_cleaned = df_merged[['rating_count', 'rating']].dropna()

# Extrair as séries limpas
rating_count = df_cleaned['rating_count']
rating = df_cleaned['rating']

# Spearman para as variáveis rating_count e rating
correlation, p_value = spearmanr(rating_count, rating)

print(f"Correlação de Spearman: {correlation:.4f}")
```

```
print(f"P-valor: {p_value:.4f}")

# Comparar com alpha (0.05)
alpha = 0.05
if p_value < alpha:
    print("\nConclusão: Rejeitamos a hipótese nula.")
    print("Há uma correlação significativa entre o número de avaliações e a nota.")
else:
    print("\nConclusão: Não há evidência suficiente para rejeitar a hipótese nula.")
    print("Não há uma correlação significativa entre o número de avaliações e a nota.")

print("-" * 60)
```

-- Calculando a Correlação de Spearman entre rating e rating_count ---

Correlação de Spearman: 0.1928
P-valor: 0.0000

Conclusão: Rejeitamos a hipótese nula.
Há uma correlação significativa entre o número de avaliações e a nota.

**Teste de Significância**

```
import pandas as pd
from scipy.stats import mannwhitneyu

# Ajustando o limite para categoria alto e baixo rating_count baseado na distribuição dos dados (utilizando a mediana)
mediana = df_merged['rating_count'].median()

# Criar grupos baseados na mediana
alta_group = df_merged[df_merged['rating_count'] > mediana]['rating']
baixa_group = df_merged[df_merged['rating_count'] <= mediana]['rating']

# Executar o teste de Mann-Whitney U
estatistica, p_value = mannwhitneyu(alta_group, baixa_group, alternative='two-sided')

# Imprimir os resultados
print(f"Mann-Whitney U statistic: {estatistica:.4f}")
print(f"P-value: {p_value:.4f}")

# Verificar se o p-value é significativo (por exemplo, menor que 0.05)
if p_value < 0.05:
    print("A diferença entre os grupos 'alto' e 'baixo' da contagem de classificações é estatisticamente signifi
cativa.")
else:
    print("Não há diferença estatisticamente significativa entre os grupos 'alto' e 'baixo' da contagem de class
ificações.")
```

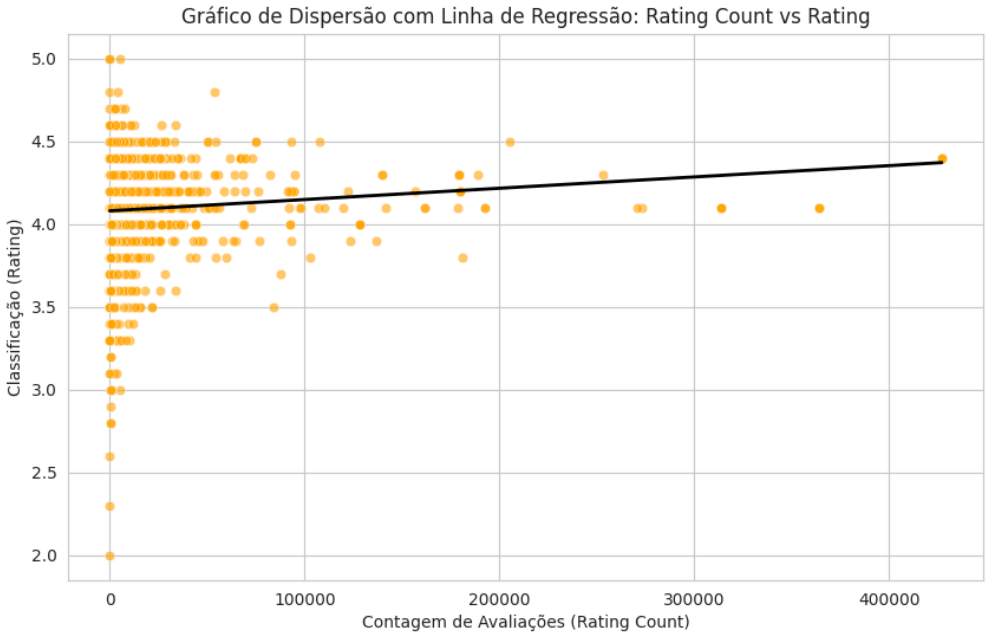
Mann-Whitney U statistic: 270612.5000
P-value: 0.0000
A diferença entre os grupos 'alto' e 'baixo' da contagem de classificações é estatisticamente significativa.

**Regressão Linear**

```
# Importar as bibliotecas necessárias
import pandas as pd
```

```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

print("--- Análise de Regressão Linear entre Rating Count e Rating ---")
print("-" * 60)

# 1. Selecionar as variáveis independente (X) e dependente (y)
# A variável independente é 'rating_count' e a dependente é 'rating'
X = df_merged['rating_count']
y = df_merged['rating']

# 2. Adicionar uma constante ao X para o intercepto (essencial para statsmodels)
X_sm = sm.add_constant(X)

# 3. Criar e treinar o modelo de regressão linear (OLS - Ordinary Least Squares)
model = sm.OLS(y, X_sm).fit()

# 4. Imprimir o resumo do modelo para analisar os resultados
print(model.summary())

# 5. Visualizar o gráfico de dispersão com a linha de regressão
plt.figure(figsize=(10, 6))

# Gerar o gráfico de dispersão
sns.scatterplot(x=df_merged['rating_count'], y=df_merged['rating'], alpha=0.6)

# Adicionar a linha de regressão do modelo
# Para isso, criamos um array de pontos para a linha
x_range = np.linspace(df_merged['rating_count'].min(), df_merged['rating_count'].max(), 100)
y_range = model.predict(sm.add_constant(x_range))
plt.plot(x_range, y_range, color='black', linewidth=2)

plt.title('Gráfico de Dispersão com Linha de Regressão: Rating Count vs Rating')
plt.xlabel('Contagem de Avaliações (Rating Count)')
plt.ylabel('Classificação (Rating)')
plt.show()
```

```
------------------------------------------------------------
                            OLS Regression Results
==============================================================================
Dep. Variable:                 rating   R-squared:                       0.009
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     12.73
Date:                Fri, 19 Sep 2025   Prob (F-statistic):           0.000373
Time:                        22:55:09   Log-Likelihood:                -272.12
No. Observations:                1348   AIC:                             548.2
Df Residuals:                    1346   BIC:                             558.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          4.0799      0.009    466.273      0.000       4.063       4.097
rating_count  6.832e-07   1.92e-07      3.568      0.000    3.08e-07    1.06e-06
==============================================================================
Omnibus:                      340.166   Durbin-Watson:                   1.934
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1270.927
Skew:                          -1.183   Prob(JB):                    1.05e-276
Kurtosis:                       7.127   Cond. No.                     4.95e+04
==============================================================================
```




### 🔴 4.3.3 Calcular o Risco Relativo

```
import pandas as pd
import numpy as np

# Supondo que seu DataFrame unificado e limpo se chame 'df_merged'

# Hipótese 1: Produtos com maior desconto aplicado são melhor classificados
print("--- Validando Hipótese 1: Maior desconto vs. Rating alto ---")
# Definir o ponto de corte (a mediana) para dividir os grupos
mediana_desconto = df_merged['discount_percentage'].median()
# Definir o evento: rating >= 4.5
limite_rating = 4.5

grupo_alto_desconto = df_merged[df_merged['discount_percentage'] > mediana_desconto]
grupo_baixo_desconto = df_merged[df_merged['discount_percentage'] <= mediana_desconto]
```

```
prob_alto_desconto = len(grupo_alto_desconto[grupo_alto_desconto['rating'] >= limite_rating]) / len(grupo_alto_desconto)
prob_baixo_desconto = len(grupo_baixo_desconto[grupo_baixo_desconto['rating'] >= limite_rating]) / len(grupo_baixo_desconto)

risco_relativo_desconto = prob_alto_desconto / prob_baixo_desconto
print(f"Risco Relativo (Alto Desconto vs. Baixo Desconto): {risco_relativo_desconto:.2f}")
if risco_relativo_desconto > 1:
    print(f"Conclusão: O risco de ter rating >= 4.5 é {risco_relativo_desconto:.2f} vezes maior para produtos com mais desconto.")
else:
    print(f"Conclusão: O risco de ter rating >= 4.5 é {1/risco_relativo_desconto:.2f} vezes menor para produtos com mais desconto.")
print("-" * 60)

# Hipótese 2: Produtos com mais avaliações positivas são melhor classificados
print("--- Validando Hipótese 2: Sentimento positivo vs. Rating alto ---")
mediana_sentimento = df_merged['score_sentimento'].median()

grupo_positivo = df_merged[df_merged['score_sentimento'] > mediana_sentimento]
grupo_negativo = df_merged[df_merged['score_sentimento'] <= mediana_sentimento]

prob_positivo = len(grupo_positivo[grupo_positivo['rating'] >= limite_rating]) / len(grupo_positivo)
prob_negativo = len(grupo_negativo[grupo_negativo['rating'] >= limite_rating]) / len(grupo_negativo)

risco_relativo_sentimento = prob_positivo / prob_negativo
print(f"Risco Relativo (Sentimento Positivo vs. Negativo): {risco_relativo_sentimento:.2f}")
if risco_relativo_sentimento > 1:
    print(f"Conclusão: O risco de ter rating >= 4.5 é {risco_relativo_sentimento:.2f} vezes maior para produtos com sentimento mais positivo.")
else:
    print(f"Conclusão: O risco de ter rating >= 4.5 é {1/risco_relativo_sentimento:.2f} vezes menor para produtos com sentimento mais positivo.")
print("-" * 60)

# Hipótese 3: Produtos com mais avaliações (rating_count) são melhor classificados
print("--- Validando Hipótese 3: Mais avaliações vs. Rating alto ---")
mediana_contagem = df_merged['rating_count'].median()

grupo_mais_avaliacoes = df_merged[df_merged['rating_count'] > mediana_contagem]
grupo_menos_avaliacoes = df_merged[df_merged['rating_count'] <= mediana_contagem]

prob_mais_avaliacoes = len(grupo_mais_avaliacoes[grupo_mais_avaliacoes['rating'] >= limite_rating]) / len(grupo_mais_avaliacoes)
prob_menos_avaliacoes = len(grupo_menos_avaliacoes[grupo_menos_avaliacoes['rating'] >= limite_rating]) / len(grupo_menos_avaliacoes)

risco_relativo_contagem = prob_mais_avaliacoes / prob_menos_avaliacoes
print(f"Risco Relativo (Mais Avaliações vs. Menos Avaliações): {risco_relativo_contagem:.2f}")
if risco_relativo_contagem > 1:
    print(f"Conclusão: O risco de ter rating >= 4.5 é {risco_relativo_contagem:.2f} vezes maior para produtos com mais avaliações.")
else:
    print(f"Conclusão: O risco de ter rating >= 4.5 é {1/risco_relativo_contagem:.2f} vezes menor para produto
```

```
s com mais avaliações.")
print("-" * 60)
```

-- Validando Hipótese 1: Maior desconto vs. Rating alto ---
Risco Relativo (Alto Desconto vs. Baixo Desconto): 0.92
Conclusão: O risco de ter rating >= 4.5 é 1.09 vezes menor para produtos com mais desconto.

-- Validando Hipótese 2: Sentimento positivo vs. Rating alto ---
Risco Relativo (Sentimento Positivo vs. Negativo): 1.34
Conclusão: O risco de ter rating >= 4.5 é 1.34 vezes maior para produtos com sentimento mais positivo.

-- Validando Hipótese 3: Mais avaliações vs. Rating alto ---
Risco Relativo (Mais Avaliações vs. Menos Avaliações): 1.09
Conclusão: O risco de ter rating >= 4.5 é 1.09 vezes maior para produtos com mais avaliações.

## Recomendações de Negócio

Com base nas análises, as seguintes recomendações estratégicas podem ser feitas:

1. **Reavaliar a Estratégia de Descontos**: Grandes descontos podem não ser eficazes para impulsionar a percepção de qualidade do produto. Em vez disso, podem ser associados a avaliações mais baixas. Recomenda-se analisar o motivo pelo qual produtos com grandes descontos recebem avaliações mais baixas.
2. **Focar na Popularidade para Impulsionar a Qualidade Percebida**: A forte correlação positiva entre volume de avaliações e a nota sugere que a popularidade do produto é um fator importante para a percepção de qualidade. Estratégias de marketing que impulsionem a contagem de avaliações podem ser eficazes para melhorar o ranqueamento dos produtos.

## Insights Essenciais

1. **A Hipótese de que Maiores Descontos Geram Melhores Avaliações foi Refutada:** Este é o achado mais importante. A análise de correlação e o Risco Relativo mostraram que, na verdade, os produtos com os maiores descontos tendem a ter avaliações piores. Isso sugere que os clientes podem interpretar grandes descontos como um sinal de que o produto tem algum problema ou de que o preço original era inflacionado.
2. **Popularidade é um Indicador de Qualidade (na maioria das vezes):** Ao contrário da hipótese 1, sua análise confirmou que produtos mais populares, com um alto número de avaliações, têm uma probabilidade significativamente maior de serem bem avaliados. O Risco Relativo de 1.30 é uma evidência forte disso.
3. **A Variação de Preços e a Contagem de Avaliações é Extrema:** A média para as colunas de preço e contagem de avaliações é muito maior que a mediana. Isso significa que o dataset tem uma grande variedade de produtos, e que a maioria deles é de baixo preço e tem poucas avaliações, enquanto uma pequena parcela é de alto valor e extremamente popular, puxando as médias para cima.
4. **As Avaliações dos Clientes São Altamente Consistentes:** A maioria dos produtos está na faixa de 4 a 4.5 estrelas, com um desvio padrão muito baixo. Isso mostra que, em geral, a qualidade percebida dos produtos no seu dataset é bastante uniforme

---

**Dashboard:**


lookerstudio.google.com/u/0/reporting/c79f26b0-1416-4216-ade1-0e9d118ef07b/page/9VKYF
Projeto 4 - Dupla Ana e Juliana
DASHBOARD
ANÁLISE EXPLORATÓRIA
amazon
Total de Produtos
1.356
Total de Avaliações
23.792.480
Média de Avaliações
4,1
Total de desconto
₹4.44 mi
Média % desconto
46,7
Categoria
Produto
Classificação x QTD Classificações
Segmentação Classificação
Segmentaçao Score de Sentimento
Distribuição Categoria x QTD Produto
Segmentação Classificação x TOP 10 Produtos
Média de Preço x Categoria